

Série “Como Fazer”



Criando menus, barras de comando e botões personalizados no Excel usando VBA

por Robert Friedrich Martim

Autor: Robert F Martim
Publicado: www.juliobattisti.com.br
Contato: rm@faircourt.com

Criado em: 19/5/2004 11:28:00
Última edição: 12/4/2005 20:00:00

EXCEL – Série Como Fazer

Criando menus, barras de comando e botões personalizados no Excel usando VBA

O autor descreve, de forma detalhada, como criar os diversos tipos de menus e barras de comandos disponíveis no Excel. Crie atalhos no teclado para acessar menu personalizado, FacelIDs personalizadas para os aplicativos, menus de atalho com o botão direito do mouse e muito mais. Uma referência que não pode faltar aos usuários que desejam criar aplicações profissionais com o Excel, definindo seus próprios menus e barras de comandos, personalizadas. **O módulo é acompanhado por 10 pastas de trabalho desenvolvidas com vários exemplos práticos, os quais ajudarão você a entender e a acompanhar os exemplos propostos no curso.**

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelvbamenu/default.asp>

Formulários no Excel Utilizando VBA: Listbox e Combobox

O autor descreve, de forma detalhada, como manipular caixas de manipulação e caixas de listagem. Neste curso o amigo leitor aprenderá, em detalhes, como:

1. Utilizar nomes dinâmicos para preencher caixas de listagem e caixas de combinação.
2. Fazer referência entre os dois controles e outros controles em um formulário, a partir de seleções em uma caixa de combinação ou caixa de listagem.
3. Classificar itens em ordem crescente/decrescente. O leitor aprenderá a lógica por trás da classificação utilizada no Excel. O leitor também aprenderá a criar funções para ordenar listas.
4. Adicionar itens únicos a partir de uma lista onde vários itens se repetem.
5. Referenciar itens que pertencem a uma lista.
6. O leitor aprenderá a lógica por trás de listas de itens únicos e como criar funções para retornar tais listas.
7. Passar itens entre caixas de listagem.
8. Mover itens dentro de uma caixa de listagem.
9. Conectar uma caixa de listagem ao Outlook e filtrar a lista de contatos.
10. Utilizar a lista de contatos filtrada para enviar e-mail utilizando um servidor SMTP virtual.
11. Conectar caixas de combinação a dados de uma tabela ou consulta do Access.
12. Filtrar, ler, escrever e apagar registros no Access, usando programação VBA no Excel.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelvbaforms/default.asp>

Fórmulas no Excel: Manipulando Datas e Horas

- Dúvidas sobre como fazer cálculos com valores de horas no Excel?
- Dúvidas sobre como fazer cálculos com valores de datas no Excel?
- Procurando soluções práticas, fáceis de entender e Adaptar para o seu uso?

Este módulo trata sobre a manipulação de datas e horas no Excel. Você irá entender exatamente como o Excel armazena valores de datas e horas, aprenderá a criar fórmulas e usar funções para cálculos com datas e horas. Além do arquivo .PDF, **você também receberá 13 planilhas, com os exemplos utilizados no curso.**

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelatashoras/default.asp>

BRINDE: Na compra do **CD-04** você ganha um vídeo curso com 22 minutos de duração, o qual mostra, passo a passo, como criar um controle do tipo calendário(Calendar Control) no VBA.

Fórmulas no Excel: Fazendo Milagres com Fórmulas Matriciais

Veja se você se encaixa em um dos itens a seguir:

- Dificuldades para fazer cálculos complexos no Excel??
- Chegou a um ponto em que acha que não tem solução para alguns cálculos no Excel??
- Procurando soluções práticas, fáceis de entender e Adaptar para o seu uso??

O autor descreve, de forma detalhada, como funcionam e como manipular fórmulas matriciais no Excel. Este módulo trata sobre a manipulação e construção de fórmulas matriciais, do ponto de vista prático. Os exemplos visam desenvolver a compreensão do leitor sobre como tais funções funcionam como elas podem ser aplicadas na resolução de problemas práticos. Uma referência que não pode faltar ao leitor que deseja entender como funcionam os cálculos e manipulações complexas de dados no Excel. **O módulo é acompanhado por 9 pastas de trabalho (em um total de 22 planilhas)**, as quais apresentam inúmeros exemplos práticos.

Além disso, o leitor poderá adquirir um curso em vídeo, com mais de 2 horas de duração, o qual fornece uma série de outros exemplos.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelmatric/default.asp>

Fórmulas no Excel: Fazendo Milagres com Fórmulas Matriciais (Vídeo Curso)

Este novo curso introduz a interatividade do vídeo para ensinar sobre fórmulas e funções matriciais no Excel e VBA. Você aprenderá através das explicações e visualização como as fórmulas são entradas e manipuladas. Aprenda através de aulas em vídeo, interativas, de fácil acompanhamento.

Este vídeo é a companhia perfeita para quem já comprou o curso "Fórmulas e Funções Matriciais no Excel", no formato .PDF. Embora, ambos tratem do mesmo assunto, aspectos diferentes, assim como exemplos diferentes, são tratados e mostrados

no vídeo. Ou seja, no vídeo você terá diversos exemplos novos, não presentes no curso em .PDF.

Combinando o material escrito no curso .PDF com o vídeo o aluno terá uma referência completa sobre fórmulas matriciais no Excel... Além disso, o leitor descobrirá que nem sempre o Ajuda do Excel é de "grande ajuda". Principalmente, quando ele informa a maneira incorreta para se manipular matrizes constantes.

IMPORTANTE: Este curso está disponível somente para envio em CD, pelos Correios. Devido ao tamanho dos arquivos de vídeo - quase 400 MB, fica inviável disponibilizá-los via download ou via e-mail.

Para você que gosta de cursos interativos, com vídeo e som, esta sem dúvidas é uma excelente opção. Este é apenas o primeiro de uma série de cursos que serão disponibilizados no formato de vídeo-aulas.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/videocursos/excelmatric/default.asp>

Fórmulas no Excel: Funções de Procura e Referência

Veja se você se encaixa em um dos itens a seguir:

- Dificuldades para usar as funções PROCV e PROCH??
- Dificuldades para utilizar as funções de pesquisa do Excel??
- Procurando soluções práticas, fáceis de entender e adaptar para o seu uso??

Uma referência completa que não pode faltar a todos os usuários que têm a necessidade de trabalhar com fórmulas e funções de pesquisa no Excel.

Neste novo curso o autor descreve, de forma detalhada, como funcionam e como utilizar as funções e fórmulas para pesquisa e validação de dados no Excel. Este módulo trata sobre a manipulação e criação de fórmulas de procura e referência do ponto de vista prático. **CHEGA DE DÚVIDAS SOBRE COMO USAR AS FUNÇÕES PROCV E PROCH, DENTRE OUTRAS.**

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelproc/default.asp>

Autor: Robert F Martim
Publicado: www.juliobattisti.com.br
Contato: rm@faircourt.com

Criado em: 19/5/2004 11:28:00
Última edição: 12/4/2005 20:00:00

Guia avançado do Registro do Excel

O autor descreve de forma detalhada, como manipular o registro do Excel. Este módulo mostra como modificar a Registry do Excel (Windows) de forma a personalizar dezenas de aspectos do Excel, tais como: número de ações que podem ser desfeitas, modelos de gráfico padrão, gerenciamento de abertura de arquivos, segurança de macros, segurança de Internet, etc.

O autor utiliza uma linguagem extremamente didática, de fácil compreensão. O curso é todo baseado em exemplos práticos, detalhadamente explicados.

DEZENAS DE EXEMPLOS PRÁTICOS E ÚTEIS DE CONFIGURAÇÕES DO EXCEL, USANDO A REGISTRY:

- ENTENDA COMO FUNCIONA A REGISTRY
- SAIBA COMO CONFIGURAR O EXCEL USANDO A REGISTRY
- APRENDA A MODIFICAR DEZENAS DE OPÇÕES DO EXCEL
- CONFIGURAÇÕES NÃO-DOCUMENTADAS NA AJUDA DO EXCEL
- APRENDA A ALTERAR O TIPO DE GRÁFICO PADRÃO
- ALTERE AS OPÇÕES PADRÃO DE FONTE
- APRENDA A REMOVER LINHAS DE GRADE
- APRENDA A MODIFICAR AS CONFIGURAÇÕES DE AUTO-RECUPERAÇÃO
- DEZENAS DE OUTRAS CONFIGURAÇÕES

Você não vai acreditar nas configurações que podem ser feitas no Excel usando a Registry do Windows.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/excelregistry/default.asp>

WORD – Série Como Fazer

Criando menus, barras de comando e botões personalizados no MS Word usando VBA

Veja se você se encaixa em um dos itens a seguir:

- Dificuldades em entender como funcionam os menus e barras de ferramentas no Word??
- Não sabe como personalizar os menus e barras de ferramentas do Word??
- Não sabe como criar novos menus e novas barras de ferramentas no Word?

Então, sem dúvidas, este curso foi feito sob encomenda para você. Chega de dúvidas, é hora de dominar os MENUS e BARRAS DE FERRAMENTAS no Word.

O autor descreve, de forma detalhada, como manipular barras de comandos, menus e botões de comandos no Word. Este curso trata sobre a manipulação destas ferramentas no Word e como elas podem ser utilizadas para personalizar o desenvolvimento de aplicativos no Word. Este curso segue a linha do curso equivalente no Excel: Criando Menus Personalizados no Excel, porém, a abordagem, conteúdo e exemplos são totalmente diferentes, tornando o curso no Word e no Excel complementares, quando o assunto é desenvolvimento no Office. Se você já possui o curso sobre menus no Excel, este curso de menus no Word é o complemento ideal para uma referência ainda mais completa sobre como manipular estes objetos entre os aplicativos Office.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/wordmenus/default.asp>

Autor: Robert F Martim
Publicado: www.juliobattisti.com.br
Contato: rm@faircourt.com

Criado em: 19/5/2004 11:28:00
Última edição: 12/4/2005 20:00:00

ACCESS – Série Como Fazer

Criando menus, barras de comando e botões personalizados no MS Access usando VBA

O autor descreve, de forma detalhada, como criar os diversos tipos de menus e barras de comandos disponíveis no Microsoft Access. Este módulo, visa a criação de atalhos no teclado para acessar menu personalizado, FaceIDs personalizadas para os aplicativos, menus de atalho com o botão direito do mouse e muito mais. Uma referência que não pode faltar às pessoas que estão sérias quando o assunto é menu personalizado no MS Access. **O módulo é acompanhado por 14 bancos de dados e uma pasta de trabalho do Excel como recurso externo.**

- O que é um menu
- O que é uma barra de comando
- Como menus e barras de comando são criadas no Access
- Como remover os menus personalizados
- Removendo menus manualmente
- Removendo menus via código
- Posicionando os objetos
- Executando ações e atalhos de teclado
- Utilizando os FaceIDs do Office no Access
- Faces personalizadas
- Criando “popups” de atalho
- Adicionando um controle a um menu de atalho popup já existente
- Menus “Combobox”
- Automatizando a criação de menus
- Substituindo menu principal do Access pelo seu menu personalizado
- Listando os menus e sub-menus do Access
- Listando somente as barras de comando/ferramentas em um arquivo texto
- Listando as barras de comando/ferramentas e sub-menus no Access

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/accessmenus/default.asp>

MATEMÁTICA E ESTATÍSTICA – Série Como Fazer

Introdução a Matemática Financeira

Este curso é um curso teórico sobre Matemática Financeira. O curso apresenta desde uma revisão dos elementos básicos da Matemática, passando pelos elementos básicos da Matemática Financeira, tais como: juros simples, juros compostos, valor presente, valor futuro, fluxo de caixa, capitalização, etc.

O autor utiliza uma linguagem extremamente didática, de fácil compreensão. O curso é todo baseado em exemplos práticos, detalhadamente explicados.

MAIS DE 250 EXERCÍCIOS RESOLVIDOS, OS QUAIS COBREM DIVERSAS SITUAÇÕES PRÁTICAS ENCONTRADAS NO SEU DIA-A-DIA, TAIS COMO:

- **CÁLCULOS DE EMPRÉSTIMOS**
- **CÁLCULOS DE FINANCIAMENTOS**
- **CÁLCULOS DE PRESTAÇÕES DO AUTOMÓVEL**
- **CÁLCULOS DE PRESTAÇÕES DA CASA PRÓPRIA**
- **CÁLCULOS PARA FUNDOS DE APOSENTADORIA**
- **CÁLCULOS PARA FINANCIAMENTO DE CARTÃO DE CRÉDITO**
- **DEZENAS DE OUTROS EXEMPLOS PRÁTICOS**

Para cada capítulo, há uma breve introdução aos conceitos teóricos da Matemática Financeira, e logo a seguir são apresentados exemplos práticos, detalhadamente explicados, resolvidos passo-a-passo. Não são apresentadas longas discussões teóricas, pois este não é o foco do curso. O foco é apresentar o conceito e colocá-lo em prática logo em seguida, para que o leitor possa ver como é o funcionamento dos cálculos.

MESMO QUE VOCÊ NÃO SEJA "MUITO AMIGO DA MATEMÁTICA", COM ESTE CURSO VOCÊ VERÁ COMO É FÁCIL APRENDER MATEMÁTICA FINANCEIRA.

APENAS: R\$ 10,00

Para comprá-lo, visite <http://www.juliobattisti.com.br/cursos/intmatfin/default.asp>

Nota sobre direitos autorais

Este eBook é de autoria de Robert F Martim, sendo comercializado através do site www.juliobattisti.com.br ou através do site de leilões Mercado Livre: www.mercadolivre.com.br.

Ao adquirir este eBook você tem o direito de lê-lo na tela do seu computador e de imprimir quantas cópias desejar, desde que sejam para uso pessoal. É vetada a distribuição deste eBook, mediante cópia ou qualquer outro meio de reprodução, para outras pessoas. Se você recebeu este eBook através de e-mail ou via FTP de algum site da Internet, ou através de CD de Revista, saiba que você está com uma cópia pirata, não autorizada. Se for este o seu caso entre em contato com o autor através do e-mail rm@faircourt.com ou comunique diretamente ao nosso site através do e-mail webmaster@juliobattisti.com.br.

Ao regularizar a sua cópia, você estará remunerando, mediante uma pequena quantia, o trabalho do autor e incentivando que novos trabalhos sejam disponibilizados.

Visite periodicamente o site www.juliobattisti.com.br para ficar por dentro das novidades!

Pré-requisitos

Para completar este curso, você precisa ter um conhecimento mínimo de manipulação das planilhas do Excel. Porém, o conhecimento básico de VBA será necessário para que o jargão (embora ele tenha sido reduzido ao máximo) não atrapalhe a compreensão, desenvolvimento e aprendizado.

Este curso não tem por objetivo ensinar qualquer outra coisa a não ser a criação de menus, barras de comando e botões personalizados no Excel utilizando VBA (Visual Basic for Application).

Os leitores interessados podem adquirir o curso básico de Excel em 120 lições no seguinte endereço: www.juliobattisti.com.br/excel120/excel120.asp

Objetivos deste eBook

Este eBook foi dividido em módulos. Este primeiro módulo visita a criação de menus, barras de comando e botões personalizados no Excel usando o Visual Basic for Application (VBA).

O trabalho foi desenvolvido a partir da demanda dos usuários do site www.juliobattisti.com.br. O material procura analisar questões pertinentes ao dia-a-dia de seu trabalho. Vários exemplos são retirados das dúvidas colocadas no fórum do site ou enviadas diretamente para o autor pelos assinantes do site.

A linguagem utilizada é descontraída e com o mínimo de jargão possível. O objetivo é ter um eBook com conteúdo relevante e de fácil compreensão.

Qualquer dúvida referente a este módulo podem ser colocadas diretamente no fórum Excel avançado no endereço: http://www.juliobattisti.com.br/forum/forum.asp?FORUM_ID=3

Comentários e sugestões para melhora do material podem ser enviados diretamente para o autor no endereço rm@faircourt.com

ÍNDICE ANALÍTICO

Introdução	1
Bem-vindo a série “Como Fazer”	1
Antes de continuar	1
1. O que é um menu e o que é uma barra de comando	2
2. Como menus e barras de comando são criadas	3
3. Dimensionando os objetos corretamente.....	8
4. Posicionando os objetos	10
5. Criando e executando ações nos menus.....	14
6. Colocando os FacelDs.....	19
6.1. FacelD personalizadas.....	19
7. Criando menus de atalho	22
8. Menus “Dropdown”	28
9. Botões Toggle	34
10. Criando um menu com o controle Edit.....	38
11. Removendo os menus personalizados	42
12. Bloqueando e reiniciando Menus Padrões	45
13. Criando um menu camaleão	48
14. Automatizando a criação de menus.....	56
15. Sobre o autor	63

SÉRIES: COMO FAZER

Criando menus, barras de comando e botões personalizado no Excel usando VBA

por Robert Friedrich Martim

Introdução

Bem-vindo a série “Como Fazer”.

Nas series que serão escritas estaremos olhando em aspectos distintos do Excel de acordo com a demanda do fórum Excel Júlio Battisti (<http://www.juliobattisti.com.br>). A intenção principal é fornecer ao internauta uma ferramenta que concentre a atenção na solução de um problema específico.

Nesta primeira série estaremos vendo as diversas possibilidades de se criar barras de comandos e menus personalizados. Iremos utilizar métodos manuais e dinâmicos para criar nossas barras e menus. Estaremos olhando os elementos que compõem os menus e barras de comando e como eles são utilizados na solução de nossos problemas.

Antes de continuar

O trabalho desenvolvido foi testado para compatibilidade com Excel 2002 e 2003. Devido à rápida mudança em termos de tecnologia de software, atenção sempre será dada às versões mais recentes do aplicativo Excel.

Sugestões serão sempre bem-vindas e esperamos que o leitor participe pro ativamente no desenvolvimento do material aqui apresentado.

Finalmente, quando objetos e variáveis são dimensionados nesta apostila é utilizada a convenção de dimensionamento dos objetos e variáveis. Por exemplo, um botão é dimensionado como Dim btn.

1. O que é um menu e o que é uma barra de comando

Menus e barras de comando são objetos da área de trabalho do Excel (e qualquer outro aplicativo que utilize estas ferramentas) que permitem o usuário interagir com o ambiente de trabalho. Menus e barras de comando deixam o trabalho mais dinâmico e eficiente. Embora vários comandos possam ser acessados através de teclas de atalho, muitos outros não o são. Desta forma, menus e barras de comando desempenham um papel crítico em nossa produtividade.

Ao criarmos menus e barras de comando queremos personalizar a área de trabalho ou desejamos acrescentar funções que não estão disponíveis nos menus e barras padrões do aplicativo. A figura abaixo mostra o menu do Excel e várias barras de comando:

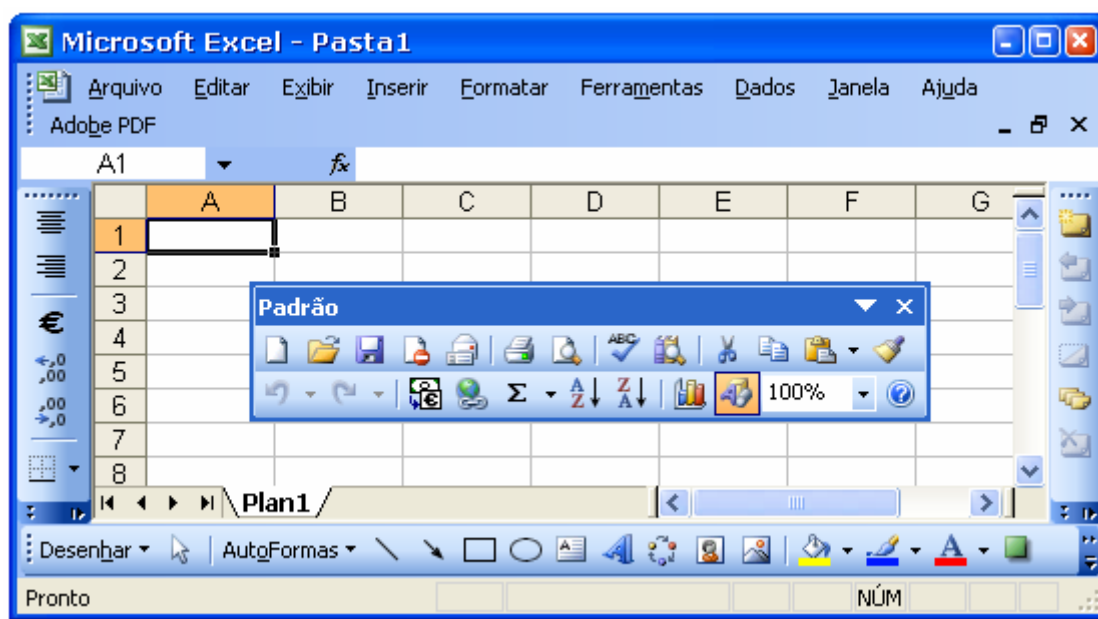


Figura 1-1

O menu principal está no topo do aplicativo. Do lado esquerdo do aplicativo temos a barra de formatação, na parte inferior temos a barra de Desenho, no canto direito temos a barra de gráficos e flutuando na área de trabalho temos a barra padrão.

Como podemos ver, ao criarmos barras de comando podemos fixá-las em qualquer parte da área de trabalho.

Hoje, não existe uma linha entre “menus” e “barras de comando”, em outras palavras, o “menu” principal do Excel, na verdade, está dentro de uma barra de comando e pode ser arrastado e colocado em um local qualquer da área de trabalho. Ele não é fixo com o aplicativo em si. Isto representa uma flexibilidade enorme em termos de manuseio do objeto em questão.

2. Como menus e barras de comando são criadas

Menus e barras de comando são objetos e como tais possuem métodos e propriedades. Para se criar este tipo de objeto precisamos colocar o código em um módulo. Para inserir um módulo de programação precisamos estar na área de edição de código conhecida como VBE (Visual Basic Editor). Para acessar esta área pressione Alt+F11. Quando o VBE for aberto clique em Inserir --> Modulo para inserir um módulo VBA.

No módulo, precisamos definir a sub-rotina que conterà o código que criará nossos menus. A figura abaixo mostra a abertura e fechamento de uma sub-rotina:

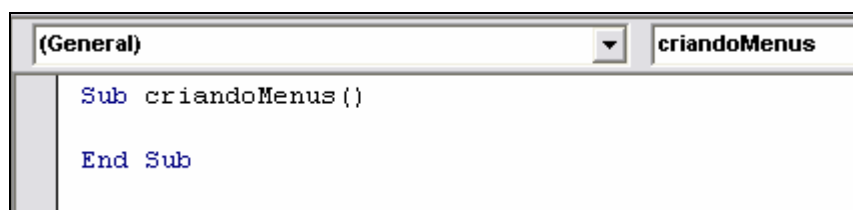


Figura 2-1

Qualquer coisa escrita em visual basic em Sub-End Sub será executada pelo programa.

Como foi dito, menus e barras de comando são objetos. Quando criamos nossos códigos é boa prática definir a dimensão (Dim) destes objetos ou variáveis. Por exemplo:

```
Sub criandoMenus()  
    Dim cmdBar As Object  
    Dim mnu As Object  
End Sub
```

Quando criamos estes objetos, precisamos dar um “set” para que o programa reconheça o tipo de objeto. Por exemplo, se pergunte: o que é “Object”? “Object” apenas não diz nada sobre o que o objeto é ou faz. O Exemplo abaixo mostra como instruir o programa a reconhecer o objeto:

```
Sub criandoMenus()  
    Dim cmdBar As Object  
    Dim mnu As Object  
  
    Set cmdBar = CommandBars.Add(Name:="Criando Menu")  
End Sub
```

Neste exemplo, estamos instruindo o VBA a acrescentar um objeto chamado cmdBar à coleção de barras de comando do aplicativo. O nome desta nova barra de comando é “Criando Menu”. Ao rodarmos o código acima, podemos verificar que o objeto é realmente adicionado à coleção de barras de comando indo até Ferramentas --> Personalizar. A figura abaixo mostra esta nova barra na coleção:



Figura 2-2

Para que esta barra de comando fique visível, precisamos selecioná-las na caixa de barras de comando. Uma alternativa é incluir no código uma linha que faça isso:

```
Sub criandoMenus()  
    Dim cmdBar As Object  
    Dim mnu As Object  
  
    Set cmdBar = CommandBars.Add(Name:="Criando Menus")  
    cmdBar.Visible = True  
End Sub
```

Ao executarmos o código, a barra é apresentada na tela:

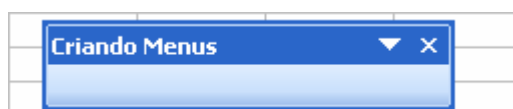


Figura 2-3

Porém, como vimos anteriormente, a barra de comando pode estar em qualquer posição em nossa área de trabalho. O VBA nos permite posicionar a barra nas “cinco” cantos da área de trabalho. Estas posições são:

- msoBarFloating (barra flutuante na tela)
- msoBarTop (barra é colocada no topo)
- msoBarLeft (barra é colocada no canto esquerdo)

- msoBarRight (barra é colocada no canto direito)
- msoBarBottom (barra é colocada na parte inferior)

Para posicionar a nossa barra de comando utilizando `Position:=nomeDaPosição` (conforme a lista anterior). O código revisto fica:

```
Sub criandoMenus()  
    Dim cmdBar As Object  
    Dim mnu As Object  
  
    Set cmdBar = CommandBars.Add(Name:="Criando Menu", _  
        Position:=msoBarFloating)  
    cmdBar.Visible = True  
End Sub
```

A barra de comando é o objeto que recebe os menus (`msoControlPopUp`) e botões (como os botões que possuem ícones de salvar, abrir, etc.)

O objeto `mnu` criado anteriormente não diz nada sobre o objeto e precisamos defini-lo como fizemos com o `cmdBar`. A lógica de inserção é a mesma que a anterior, isto é, como inserimos a barra de comando a coleção de barras de comando do Excel, agora, vamos adicionar um menu a esta barra de comando:

```
Sub criandoMenus()  
    Dim cmdBar As Object  
    Dim mnu As Object  
  
    Set cmdBar = CommandBars.Add(Name:="Criando Menu", _  
        Position:=msoBarFloating)  
    Set mnu = cmdBar.Controls.Add(Type:=msoControlPopUp)  
    With mnu  
        .Caption = "meu menu"  
        .Width = 200  
    End With  
    cmdBar.Visible = True  
End Sub
```

Com isso criamos nossa barra de comando contendo um menu:

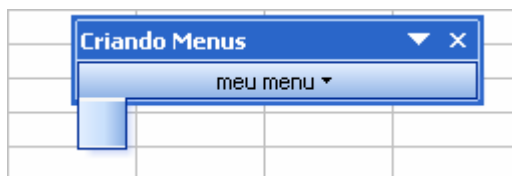


Figura 2-4

O bloco `With-End With` nos permite definir as propriedades e métodos do objeto `mnu` sem precisar repetir o seu nome (`mnu`) para cada propriedade ou método que definimos.

Por analogia, podemos criar outros menus, precisando apenas defini-los no código. Para cada mnu que entrará no código. Por exemplo:

```

Sub criandoMenus()
    '... Código anterior suprimido para este exemplo
    Set mnu = cmdBar.Controls.Add(Type:=msoControlPopUp)
    With mnu
        .Caption = "meu menu"
    End With
    Set mnu = cmdBar.Controls.Add(Type:=msoControlPopUp)
    With mnu
        .Caption = "Seu menu"
    End With
End Sub

```

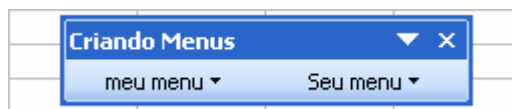


Figura 2-5

Porém, o controle `msoControlPopUp` apenas abre o “popup”. Ele é incapaz de executar instruções. Para se executar instruções precisamos adicionar o controle `msoControlButton`. Este controle pode ser inserido na barra ao no menu.

```

Sub criandoMenus()
    '... Código anterior suprimido para este exemplo
    Set mnu = cmdBar.Controls.Add(Type:=msoControlButton)
    mnu.FaceId = 326

    Set mnu = cmdBar.Controls.Add(Type:=msoControlPopUp)
    mnu.Caption = "meu menu"

    Set mnu = mnu.Controls.Add(Type:=msoControlButton)
    mnu.FaceId = 926

    Set mnu = cmdBar.Controls.Add(Type:=msoControlPopUp)
    mnu.Caption = "Seu menu"
End Sub

```

Ao executarmos o código obtemos o seguinte resultado:

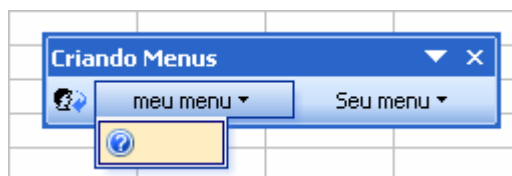


Figura 2-6

Observe que não é preciso definir um objeto para o botão. Podemos utilizar o objeto `mnu` para fazer isso (poderíamos até usar o `cmdBar`)¹.

¹ Embora seja possível, esta prática não é aconselhável. Mais adiante veremos o porquê.

Os leitores acostumados com VBA devem ter notado que ao inserir o ponto (.) após o nome do objeto a caixa contendo a lista de propriedades e métodos não é exibida. Mas quando tentamos forçar a exibição da caixa o VBE não aceita. Este tipo de problema diz respeito ao objeto em si.

Lembra quando perguntamos que objeto era esse? Pois bem, quando definimos a dimensão do objeto como sendo “Object” ele pode ser qualquer coisa. Em outras palavras, a caixa de propriedades e métodos não fica disponível porque o VBA não reconhece o objeto. A solução para o nosso problema é bem simples: basta dimensionar o objeto corretamente.

Quando criamos nossa barra de comando, nós utilizamos os seguintes objetos:

- `CommandBar` (para o `cmdbar`)
- `CommandBarPopup` (para o `msoControlPopup`)
- `CommandBarButton` (para o `msoControlButton`)

Embora tenhamos utilizados estes objetos, nós não os dimensionamos explicitamente como sendo este objeto. Quando dimensionamos o objeto é importante dimensioná-lo com a classe correta. Embora isso não afete o resultado, ele certamente afeta o desempenho e a nossa produtividade. Nos exemplos anteriores fica difícil saber qual propriedade ou método a ser utilizado. A menos que você os conheça o seu trabalho será muito menos produtivo.

3. Dimensionando os objetos corretamente

Não há nada que diga que você não possa utilizar o método apresentado anteriormente para criar suas barras de comando, menus e botões.

A figura abaixo mostra a criação das dimensões dos objetos utilizados:

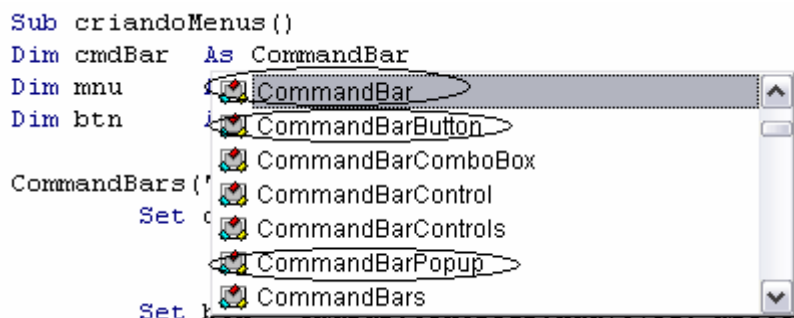


Figura 3-1

Ao definirmos explicitamente o tipo do objeto estamos deixando o código mais claro para outra pessoa ler e estamos facilitando o nosso trabalho também. As dimensões são redefinidas abaixo:

```

Sub criandoMenus()
    Dim cmdBar As CommandBar
    Dim mnu As CommandBarPopup
    Dim btn As CommandBarButton
End Sub

```

Com dimensões definidas com a classe dos objetos corretamente, as vantagens começa aparecer. Vamos ver o exemplo do botão (btn):

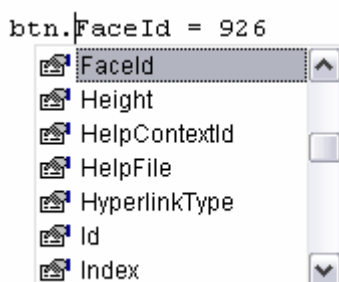


Figura 3-2

O VBA disponibiliza as propriedades e métodos para o objeto btn porque ele sabe em qual classe ele deve buscar tais informações. O nosso trabalho fica mais dinâmico e eficiente pois podemos rapidamente correr a lista e buscar o que procuramos. Antes, precisávamos adivinhar ou saber de antemão quais eram as propriedades e métodos de cada objeto.

Como os objetos possuem o prefixo CommandBars fica relativamente fácil dimensionar os objetos corretamente. O código completo comentado e dimensionado corretamente fica:

```
Sub criandoMenus()  
    Dim cmdBar As CommandBar  
    Dim mnu As CommandBarPopup  
    Dim btn As CommandBarButton  
  
    On Error Resume Next 'Continua a execução mesmo que haja um erro  
    'Deleta o menu anterior se ele existir  
    CommandBars("Criando Menus").Delete  
  
    'Adiciona o objeto cmdBar a coleção de barras de comando do Excel  
    'A barra adicionada está posicionada como "msoBarFloating".  
    'Outras opções de posicionamento incluem:  
    'msoBarTop, msoBarLeft, msoBarRight e msoBarBottom  
    Set cmdBar = CommandBars.Add(Name:="Criando Menus", _  
        Position:=msoBarFloating)  
  
    'Adiciona um botão a barra de comando cmdBar  
    Set btn = cmdBar.Controls.Add(Type:=msoControlButton)  
    btn.FaceId = 326 'Define a figura a ser mostrada no botão  
  
    ' Adiciona um "menu" a barra de comando cmdBar  
    Set mnu = cmdBar.Controls.Add(Type:=msoControlPopup)  
    With mnu  
        .Caption = "meu menu" 'Define o nome a ser exibido no menu  
    End With  
  
    Set btn = mnu.Controls.Add(Type:=msoControlButton)  
    btn.FaceId = 984  
  
    Set mnu = cmdBar.Controls.Add(Type:=msoControlPopup)  
    With mnu  
        .Caption = "Seu menu"  
    End With  
  
    cmdBar.Visible = True 'Exibi a barra de comando cmdBar  
End Sub
```

4. Posicionando os objetos

Uma questão que devemos responder é: onde devemos colocar nosso menu? Via de regra, nós criamos menus para facilitar o trabalho e acesso do usuário a certos comandos, automações ou funções que criamos. Portanto, não adianta criar um menu do tipo `msoControlPopup` nos cantos pois o acesso ao menu ficará “estranho”. Geralmente barras de comando são colocadas no topo utilizando-se o `msoBarTop`. Um outro método bastante utilizado é criar um menu personalizado na barra de comando contendo os menus do Excel.

Nesta parte estaremos fazendo exatamente isso. Estaremos posicionando o nosso menu em diferentes pontos de nossa área de trabalho.

No primeiro exemplo, repetiremos o código desenvolvido anteriormente para adicionar alguns extras. A primeira coisa que desejamos fazer é posicionar a nossa barra de comando. No momento, a barra está para flutuante e iremos colocá-la no lado esquerdo:

```
Sub criandoMenus()  
    'Somente as partes novas serão repetidas. O resto será suprimido  
    Set cmdBar = CommandBars.Add(Name:="Criando Menus", _  
        Position:=msoBarLeft)  
    'Fixa a barra do lado esquerdo  
  
    'Protege a barra contra movimentos e redimensionamento  
    cmdBar.Protection = msoBarNoMove + msoBarNoResize  
    cmdBar.Visible = True 'Exibi a barra de comando cmdBar  
End Sub
```

O código acima introduz mais uma versatilidade das barras de comando. Muitas vezes não queremos que o usuário fique movimentando as barras de comando, pois isso pode prejudicar o layout geral do aplicativo que desenvolvemos. A maneira como isso é feita é demonstrada acima.

O posicionamento é na lateral esquerda e é fixo. O usuário não pode movimentar ou modificar o tamanho da barra de comando. Outras formas de proteção incluem:

- `msoBarNoChangeDock` (não permite acoplar a barra de comando aos cantos do aplicativo)
- `msoBarNoChangeVisible` (não permite esconder a barra de comando)
- `msoBarNoCustomize` (não permite que o usuário personalize a barra)
- `msoBarNoHorizontalDock` (permite acoplar somente na vertical, isto é, nas laterais esquerda e direita do aplicativo)
- `msoBarNoMove` (não permite a movimentação da barra)
- `msoBarNoProtection` (a barra não possui proteção alguma)
- `msoBarNoResize` (não permite o redimensionamento da barra)

Autor: [Robert F Martin](#)
Publicado: www.juliobattisti.com.br
Contato: rm@faircourt.com

Criado em: 19/5/2004 11:28:00
Última edição: 12/4/2005 20:00:00

- `msoBarNoVerticalDock` permite acoplar somente na horizontal, isto é, na parte superior ou inferior do aplicativo. Veja `msoBarNoHorizontalDock`)

Neste caso específico, estamos lidando com o posicionamento de nossa barra de comando. E o posicionamento de nosso menu? Como podemos posicionar o nosso menu no Excel?

Geralmente, quando criamos menus queremos adicioná-los ao menu principal do Excel. Porém, é importante saber onde ele será posicionado. Por exemplo, se desejamos posicionar nosso menu antes do menu “Arquivo”, como procedemos? Os menus são indexados o que facilita o nosso trabalho. Como o menu Arquivo é o primeiro, a sua posição é 1. A numeração continua assim até o último menu da barra de comando padrão.

Para posicionar nosso menu no topo, modificaremos o nosso código e dimensão dos objetos ligeiramente:

```
Sub criandoMenus()  
    Dim cmdBar As CommandBarPopup  
    Dim mnu As CommandBarPopup  
    Dim btn As CommandBarButton  
End Sub
```

Como estamos interessados em adicionar um controle tipo menu ao menu principal do Excel, precisamos redimensioná-lo para `CommandBarPopup`. Novamente, precisamos instruir o Excel de forma que ele saiba o que fazer com os objetos:

```
Sub criandoMenus()  
    On Error Resume Next 'Continua a execução mesmo que haja um erro  
    'Deleta o menu anterior se ele existir  
    CommandBars(1).Controls("Criando Menus").Delete  
  
    'Position defini a posição onde o menu será inserido  
    'Neste caso ele é inserido antes do menu "Arquivo"  
    Set cmdBar = CommandBars(1).Controls.Add(Type:=msoControlPopup, _  
        before:=1)  
    cmdBar.Caption = "Criando Menus"  
    Set btn = cmdBar.Controls.Add(Type:=msoControlButton)  
    With btn  
        .Caption = "Botão 1"  
        .FaceId = 326  
    End With  
    Set mnu = cmdBar.Controls.Add(Type:=msoControlPopup)  
    mnu.Caption = "menu 1" 'Define o nome a ser exibido no menu  
    Set btn = mnu.Controls.Add(Type:=msoControlButton)  
    With btn  
        .Caption = "Botão 2"  
        .FaceId = 984  
    End With  
End Sub
```

O resultado:



Figura 4-1

Além de podemos adicionar os menus em qualquer posição do menu padrão do Excel, podemos também inserir botões nos próprios menus do Excel.

Por exemplo, vamos supor que desejamos acrescentar um botão no menu Arquivo antes do botão Novo (o primeiro botão no menu Arquivo). Neste caso, precisamos saber qual o ID do menu em questão. No caso do menu Arquivo o ID é 30002.²

```
Sub criandoMenus()  
    Dim mnu As CommandBarPopup  
  
    On Error Resume Next  
    Set mnu = CommandBars(1).FindControl(ID:=30002)  
    Set btn = mnu.Controls.Add(Type:=msoControlButton, Before:=1)  
    With btn  
        .Caption = "MEU BOTAO"  
        .FaceId = 984  
    End With  
End Sub
```



Figura 4-2

Um botão é inserido no menu Arquivo antes do botão Novo. A instrução *Before* (Antes de) indica onde o botão será inserido. Não existe a opção *After* (Depois de), pois por analogia tudo que for antes não pode ser depois. Em outras palavras, se nosso botão está antes de algum botão ou ele é o primeiro de tudo ou ele está posicionado depois de algum outro botão e, portanto, antes de algum outro. Quando não definimos, o Excel sempre joga o botão no final da lista.

² Uma lista contendo os números de identificação dos itens dos menus pode ser encontrada em: <http://www.microsoft.com>

Com o menu criado, precisamos adicionar eventos aos botões. Observe que somente os botões recebem eventos. Barras de comando e menus do tipo popup não executam comandos.

5. Criando e executando ações nos menus

Quando criamos menus queremos executar algo. Não adianta criar algo lindo e ele não funciona. Como os botões temos duas maneiras para executar um comando no botão:

- OnAction
- Execute

A diferença entre OnAction e Execute é que OnAction executa um comando que nós mesmos criamos ao passo que Execute executa um comando interno do Excel como Imprimir, por exemplo. Porém, o Execute é acionado quando criamos nosso botão, portanto, iremos utilizar o ID do botão interno para executar o comando.

Primeiramente, vamos criar o código que montará nosso menu:

```
Sub executandoMenus()  
    Dim mnu As CommandBarPopup  
    Dim btn As CommandBarButton  
  
    On Error Resume Next  
    CommandBars(1).Controls("Executar comandos").Delete  
    Set mnu = CommandBars(1).Controls.Add(Type:=msoControlPopup,  
before:=1)  
    mnu.Caption = "Executar comandos"  
    Set btn = mnu.Controls.Add(Type:=msoControlButton)  
    With btn  
        .Caption = "Exemplo OnAction"  
        .FaceId = 400  
        'Mensagem refere-se a macro a ser executada  
        .OnAction = "Mensagem"  
    End With  
  
    'Defini o ID do botão como sendo 901 (Filtro Avançado).  
    'Quando o botão for clicado o Excel carrega a  
    'caixa do Filtro Avançado  
    Set btn = mnu.Controls.Add(Type:=msoControlButton, ID:=901)  
    With btn  
        .Caption = "Exemplo Execute"  
        .FaceId = 600  
        'Este método está comentado pois utilizaremos  
        'o ID 901 para executá-lo  
        '.Execute  
    End With  
End Sub
```

O nosso menu tem o seguinte aspecto:



Figura 5-1

Os faceID's ficam a critério do leitor. Mais adiante veremos como personalizar as carinhas dos botões de menu.

Observe que no OnAction nós colocamos o nome da macro a ser executada. Porém, no caso do Execute estaremos utilizando o número referente ao comando que desejamos executar para fazer o que desejamos. Como o OnAction necessita de uma macro precisamos acrescentá-la ao nosso projeto. Como exemplo, podemos utilizar a seguinte sub-rotina:

```
Sub Mensagem()  
    MsgBox "Esta é uma mensagem de teste gerada em: " & Date  
End Sub
```

Quando clicamos no primeiro botão ele executa o OnAction e a mensagem é passada para o usuário:

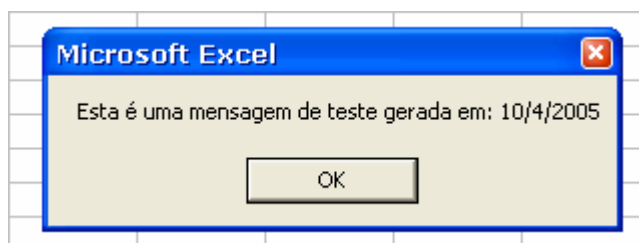


Figura 5-2

Já no segundo botão estamos executando um comando interno do Excel com um ID igual a 901:



Figura 5-3

Uma vez que definimos qual é o ID do comando o método `Execute` torna-se redundante, pois ao clicarmos no botão o comando será executado. Porém, há situações onde queremos que o comando seja executado quando o código é rodado.

Os comandos acima são acessados através de um “clique”, porém, podemos também acrescentar um combinação de teclas para criar um atalho para o nosso botão.

Atalhos são acessados diretamente do teclado e facilitam a execução de comando, pois não precisamos recorrer ao mouse toda vez que desejamos executar tal comando.

O processo é o mesmo que o `OnAction`. Na verdade, o `OnAction` é que determina a macro a ser executada. O que precisamos fazer é utilizar a propriedade `ShortcutText` para criar o Atalho e defini-lo no código

O procedimento a seguir descreve como adicionar um atalho de teclado ao menu. Primeiro, precisamos criar o menu. Neste exemplo, anexaremos o nosso botão ao menu Arquivo:

```
Sub mnuAtalho()
    Dim mnu As CommandBarPopup
    Dim btn As CommandBarButton

    On Error Resume Next
    'apaga o menu caso ele já exista
    Call deletarMenu

    Set mnu = CommandBars(1).FindControl(Id:=30002)
    Set btn = mnu.Controls.Add(Type:=msoControlButton, before:=1)
    With btn
        .Caption = "&Abrir texto" 'O ampersand (&) indica o atalho
        .ShortcutText = "Ctrl+Shift+A"
        .OnAction = "abrirTxt"
    End With
End Sub
```

```
'Adiciona o atalho a macro que desejamos rodar
Application.MacroOptions Macro:="abrirTxt", _
    HasShortcutKey:=True, _
    ShortcutKey:="A" 'É a letra que comanda o atalho.
                    'Neste caso a letra "A"

End Sub
```

A compilação acima é composta de duas partes. A primeira monta o menu que desejamos e a segunda cria o Atalho (Shortcut) que desejamos. É importante lembrar que certos atalhos podem entrar em conflito com algum atalho do sistema. No caso anterior, se o atalho for definido como Ctrl+Alt+A ao pressionar esta combinação de teclas obtemos “á” ao invés da execução do comando.

Observe também que, diferentemente do que foi feito antes, para excluir o menu anterior chamamos uma sub-rotina diferentes. Esta forma de construção é melhor do que a utilizada anteriormente, pois possibilita a chamada da exclusão sem construir um novo menu. Nos exemplos que o leitor encontrará neste módulo a utilização de ambos os métodos é feita propositalmente para forçar o aprendizado.

Voltando da digressão, ao pressionarmos Ctrl+Shift+A a caixa de diálogo abrir é mostrada e filtrada para mostrar somente arquivos texto:

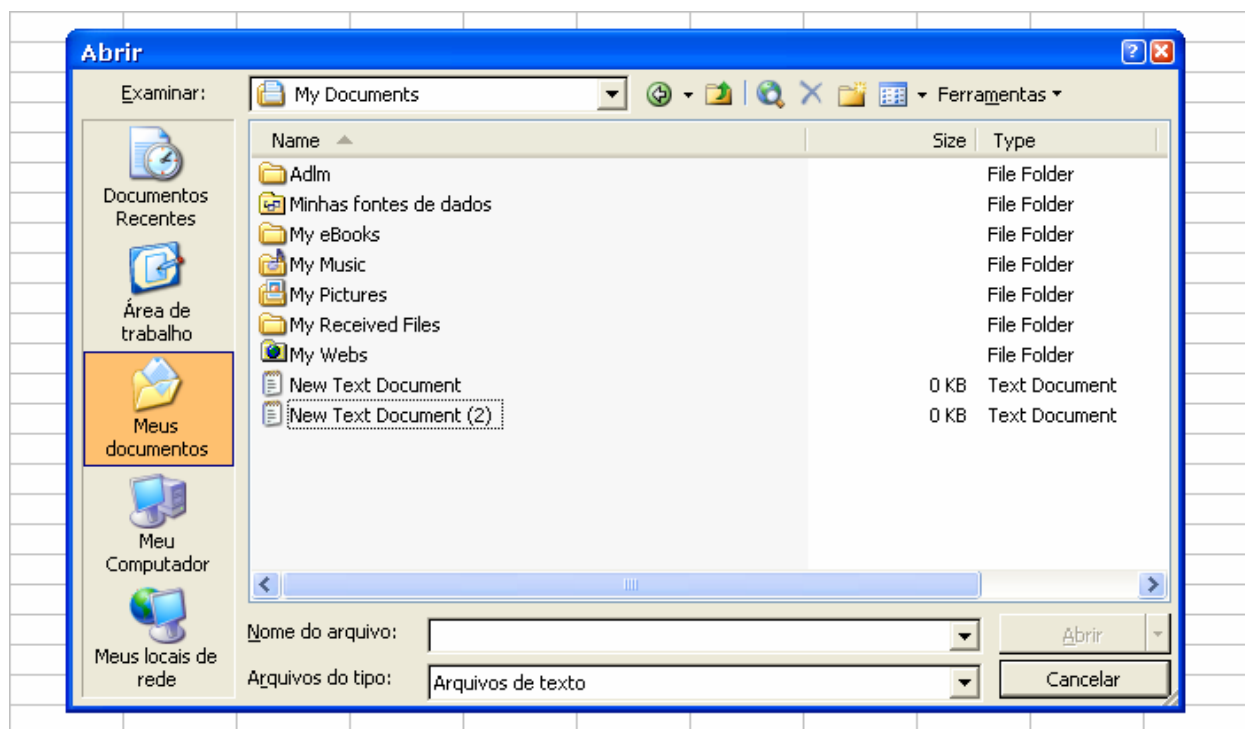


Figura 5-4

A criação de atalhos para menu deve ser pensada de antemão para que os atalhos tenham significado e sejam intuitivos.

6. Colocando os FaceIDs

Nos exemplos anteriores em vários casos foi colocado um FaceID no botão. O único objeto do menu que pode receber a carinha é o botão.

O Excel possui centenas dessas carinhas que podem ser utilizadas em nossos projetos. A figura abaixo mostra uma planilha do Excel contendo diversas carinhas e suas respectivas referências numéricas³:

	A	O	P	Q	R	S	T	U	V
1		13	14	15	16	17	18	19	20
2	0								
3	100								
4	200								
5	300								
6	400								
7	500								
8	600								
9	700								
10	800								
11	900								
12	1000								
13	1100								

Figura 6-1

Além das carinhas disponibilizadas com o Excel podemos também utilizar as nossas carinhas personalizadas. Para utilizar as carinhas disponíveis com o Excel basta seguir os exemplos anteriores. Porém, para criar as nossas próprias carinhas precisamos desenvolver nosso código um pouco. A figura abaixo mostra uma barra de comando contendo FaceIDs personalizadas:

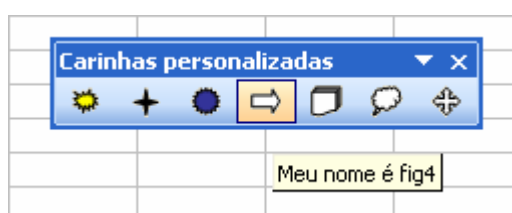


Figura 6-2

6.1. FaceID personalizadas

Para se criar este tipo de temos que passar por dois processos:

1. Copiar a figura
2. Colar a figura no botão

³ Esta planilha pode ser baixada no seguinte endereço: www.j-walk.com/ss/excel/tips/faceidgrid.exe

Para iniciar vamos criar várias figuras em uma planilha. As figuras podem ser desenhadas (como no exemplo abaixo) ou importadas de arquivos qualquer.

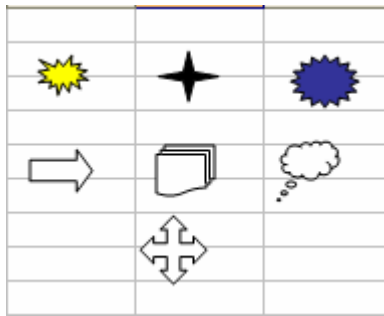


Figura 6-3

Acima, temos 7 figuras distintas que utilizaremos para criar a barra de comando personalizada como mostrado anteriormente. Para o que desejamos fazer, precisamos dar nomes que as figuras. O ideal é que os nomes sejam em série. Neste caso específico, os nomes são fig1, fig2, fig3, fig4, fig5, fig6 e fig7.

Como o processo de copiar envolve a seleção do objeto a ser copiado, precisamos levar isso em conta no nosso código. Além disso, tendo em vista que copiamos estas figuras para uma planilha em nossa pasta de trabalho precisamos fazer referência a esta pasta também.

Sendo assim, o código que resolve nosso problema pode ser dado por:

```

Sub carinhasPersonalizadas()
    Dim cmdBar      As CommandBar
    Dim btn         As CommandBarButton
    Dim ws          As Worksheet

    On Error Resume Next
    CommandBars("Carinhas personalizadas").Delete

    Set ws = ThisWorkbook.Sheets("FaceID")
    Set cmdBar = CommandBars.Add(Name:="Carinhas personalizadas", _
        Position:=msoBarFloating)

    For i = 1 To 7
        ws.Shapes("fig" & i).Select
        Selection.Copy
        Set btn = cmdBar.Controls.Add(Type:=msoControlButton)
        With btn
            .Caption = "Meu nome é fig" & i
            .Width = 30
            .PasteFace
        End With
    Next i
    cmdBar.Visible = True
End Sub

```

O que o código está fazendo

Primeiramente precisamos definir os objetos. O objeto novo é o ws (worksheet – planilha). Uma vez que dimensionamos o objeto precisamos dizer ao programa para definir a ws como sendo a planilha `FaceID`.

O código continuar a se desenvolver como nos exemplos anteriores, isto é, damos um “set” no `cmdBar` e nos botões. A única diferença aqui é que ao dar o “set” no `Caption` do botão fazemos isso dentro de um loop.

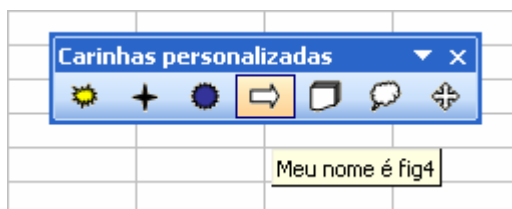


Figura 6-4

7. Criando menus de atalho

Um outro tipo de menu interessante é o menu de atalho. Este menu é acessado quando clicamos com o botão direito do mouse sobre a planilha ou qualquer outro local de nossa área de trabalho como mostra figura abaixo:

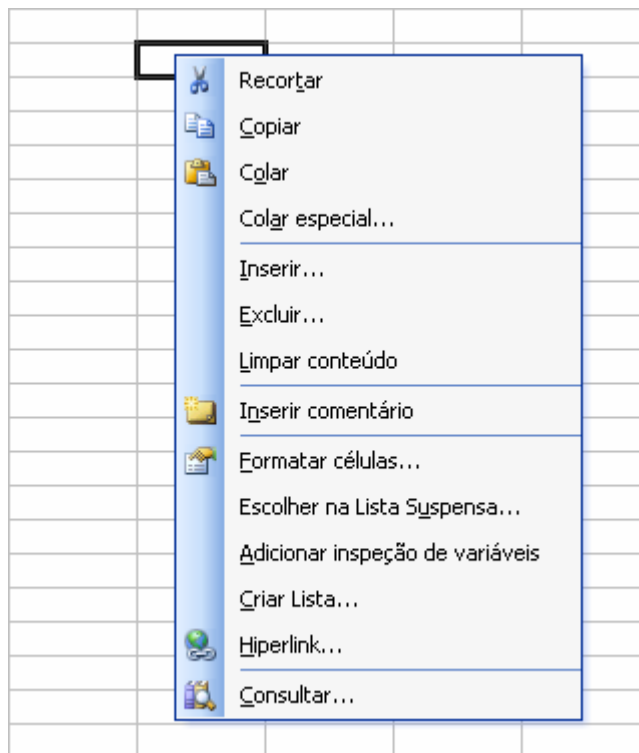


Figura 7-1

Nesta parte estaremos vendo exatamente este tipo de menu.

Assim como os menus anteriores, podemos acrescentar o nosso botão ao menu de atalho. Contudo, podemos também criar os nossos menus de atalho que respondem quando clicamos em determinados locais da área de trabalho.

A lógica para criação deste tipo de menu é a mesma que a anterior.

O que precisamos definir claramente é o evento `BeforeRightClick` na planilha onde ele ocorrerá. Além disso iremos definir a criação do atalho assim que a planilha abrir e a destruição do menu quando a planilha fechar. Esta é uma boa prática que estaremos desenvolvendo agora, pois não é uma boa idéia deixar barras de comando ou menus sem suas referências.

Primeiro, vamos criar o menu. Para isso utilizaremos dois módulos: um que conterá o código para a criação dos menus e outro que conterá as ações (OnAction) efetuadas pelo menu⁴.

O menu que estaremos criando conterá as seguintes opções:

- Negrito (transforma a seleção em negrito)
- Itálico (transforma a seleção em itálico)
- Sublinhado (transforma a seleção em sublinhado)
- Sobre (mostra informações sobre este aplicativo)
- Ajuda (mostra a ajuda para do Excel XP)

O código de ser escrito em um módulo para menu somente (neste exemplo, o nome do módulo é modMenu). O código construído com as opções acima fica:

```
'constante do nome do menu
Public Const BARRA As String = "mnuAtalho"

Sub mnuAtalho()
    Dim cmdBar As CommandBar
    Dim mnu As CommandBarButton

    delAtalho 'Apaga a barra de comando anterior caso ela exista
    Set cmdBar = CommandBars.Add _
        (Name:=BARRA, Position:=msoBarPopup, Temporary:=True)
    Set mnu = cmdBar.Controls.Add(Type:=msoControlButton)
    With mnu
        .Caption = "Negrito"
        .OnAction = "negrito"
        .FaceId = 113
    End With
    Set mnu = cmdBar.Controls.Add(Type:=msoControlButton)
    With mnu
        .Caption = "Itálico"
        .OnAction = "itálico"
        .FaceId = 114
    End With
    Set mnu = cmdBar.Controls.Add(Type:=msoControlButton)
    With mnu
        .Caption = "Sublinhado"
        .OnAction = "sublinhado"
        .FaceId = 115
    End With
    Set mnu = cmdBar.Controls.Add(Type:=msoControlButton)
    With mnu
        .Caption = "&Sobre..."
        .OnAction = "sobre"
        .FaceId = 326
        .BeginGroup = True
    End With
```

⁴ Não é necessária a criação de dois módulos para fazer isso. Os módulos criados aqui são para melhor organização do material.

```

Set mnu = cmdBar.Controls.Add(Type:=msoControlButton)
With mnu
    .Caption = "&Ajuda"
    .OnAction = "ajuda"
    .FaceId = 984
    .BeginGroup = True
End With
End Sub

```

Uma vez compilado, o menu terá o seguinte formato:

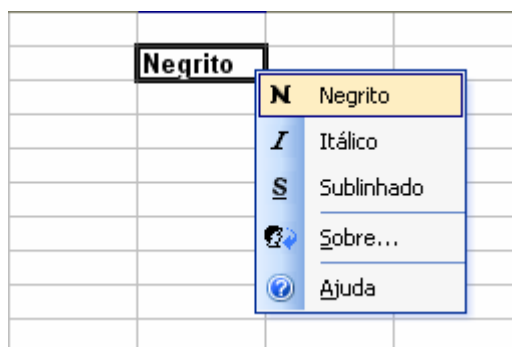


Figura 7-2

Porém, ainda não estamos prontos para a compilação. Precisamos definir os `OnAction` para cada botão em nosso menu e os eventos que irão disparar o menu acima.

No mesmo módulo acrescente o código para deleção do menu:

```

Sub delAtalho()
    'Apaga a barra de comando anterior caso ela exista
    On Error Resume Next
    CommandBars(BARRA).Delete
End Sub

```

Crie um módulo chamado `modOnAction` onde entraremos os seguintes códigos:

```

Sub negrito()
    'Modifica a seleção para negrito
    Selection.Font.Bold = True
End Sub

Sub itálico()
    'Modifica a seleção para itálico
    Selection.Font.Italic = True
End Sub

```

```
Sub sublinhado()  
    'Modifica a seleção para sublinhado  
    Selection.Font.Underline = True  
End Sub  
  
Sub sobre()  
    msg = "Este exemplo foi criado por Robert F Martim como parte " _  
        & "do módulo sobre criação "  
    msg = msg & "de menus de atalho utilizando VBA no " _  
        & "Excel." & vbCr & vbCr  
    msg = msg & "Para maiores informações visite: " _  
        & "www.juliobattisti.com.br" & vbCr & vbCr  
    msg = msg & "Para falar com o autor escreva " _  
        & "para: rm@faircourt.com"  
  
    MsgBox msg, vbInformation, "Sobre este módulo..."  
End Sub  
  
Sub ajuda()  
    'Mostra o Help do Excel XP  
    'Caso o seu Excel seja o 2000, mude para XLMAIN9  
    'Se for o 2003, mude para XLMAIN11  
    Application.Help "XLMAIN10.CHM"  
End Sub
```

Como o menu é disparado quando clicamos com o botão direito do mouse, precisamos criar este evento. Nesta parte estamos interessados no RightClick na Plan1. Abra o VBE da Plan1 e entre o seguinte código:

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As _  
    Excel.Range, Cancel As Boolean)  
    If Union(Target.Range("A1"), Range("A1:O32")).Address = _  
        Range("A1:O32").Address Then  
        CommandBars(BARRA).ShowPopup  
        Cancel = True 'Cancela o Popup padrão do Excel  
    End If  
End Sub
```

Está é a parte final de nosso código. O código para criação e deleção do menu quando a planilha é aberta ou fechada encontra-se na planilha que acompanha este tópico e não será repetido abaixo.

Ao clicar e executar o menu:

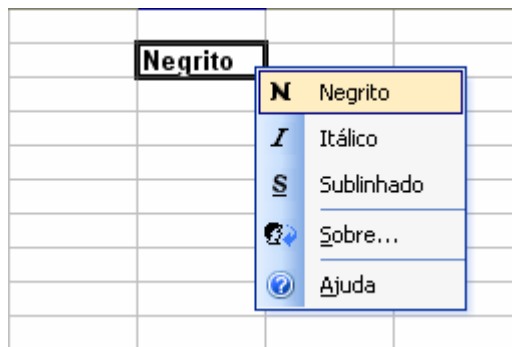


Figura 7-3

A nossa próxima parada é adicionar o nosso botão a um dos menus de atalho padrão do Excel. Para fazermos isso, precisamos saber o índice ou nome do menu ao qual acrescentaremos. Com este módulo há um aplicativo que cria uma lista dos nomes destes menus para sua futura referência.

A barra de comando que nos interessa é a `Cell` cujo índice é 29⁵. Para adicionar um botão a esta barra de comando, procedemos da mesma forma quando adicionamos o botão ao menu principal do Excel:

```
Sub mnuAtalho()
    Dim mnu As CommandBarButton

    delAtalho
    Set mnu = CommandBars(29).Controls.Add(Type:=msoControlButton, _
        before:=1)
    With mnu
        .Caption = "Sobre..."
        .FaceId = 326
        .OnAction = "sobre"
    End With
End Sub
```

No código anterior, chamamos a sub-rotina `delAtalho`. Diferentemente dos exemplos anteriores não iremos utilizar o método `Delete`, mas o método `Reset`:

```
Sub delAtalho()
    On Error Resume Next
    CommandBars(29).Reset
End Sub
```

Após a compilação do menu e o clique “direito” sobre qualquer área da planilha:

⁵ No Excel 2002, o índice é 28. Você pode utilizar o nome da barra de comando. Neste caso, use o nome “Cell”

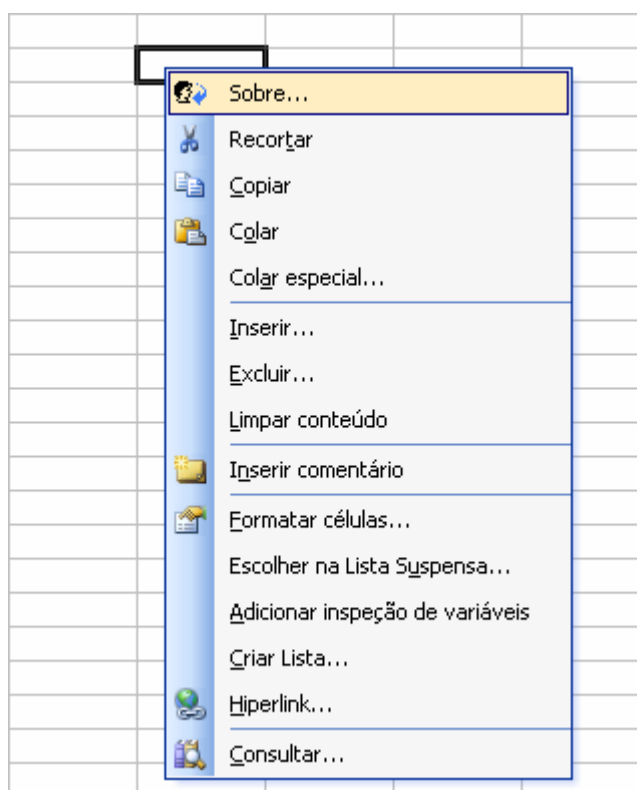


Figura 7-4

8. Menus “Dropdown”

Os menus tipo dropdown (também chamados de combobox) são os menus que contêm uma lista que quando clicada disponibiliza os itens da lista para o usuário. A figura abaixo mostra o exemplo clássico do Excel:

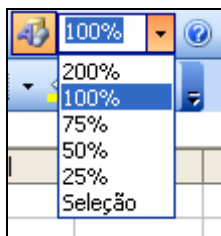


Figure 8-1

Através desta lista podemos selecionar a visualização de nossa área de trabalho, isto é, podemos ampliar ou reduzir o tamanho da área de trabalho conforme necessário.

O exemplo que estaremos olhando é retirado de uma de minhas respostas no fórum sobre este assunto. Contudo, a resposta no fórum utiliza apenas uma das possibilidades. Aqui estaremos vendo como construir este tipo de combobox de uma forma diferente.

A nossa combobox conterá uma lista de todas as planilhas disponíveis na pasta de trabalho e quando um item da lista é selecionado a planilha em questão é selecionada. Este método pode ser interessante se possuímos um número elevado de planilhas ou se desejamos esconder as guias das planilhas, por exemplo.

O menu que construiremos é mostrado na figura abaixo:

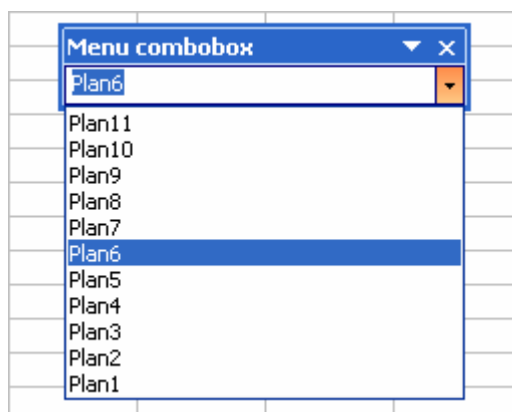


Figure 8-2

Primeiramente, vamos definir o nome da barra de comando como sendo uma `Public Const` (constante pública). No topo do módulo devemos entrar a seguinte linha:

Autor: [Robert F Martim](#)
Publicado: www.juliofattisti.com.br
Contato: rm@faircourt.com

Criado em: 19/5/2004 11:28:00
Última edição: 12/4/2005 20:00:00

```
Public Const CMDBARNOME As String = "Menu combobox"
```

O nome da constante pode ser qualquer coisa que o leitor desejar. Agora, a constante CMDBARNOME pode ser acessada de qualquer sub-rotina que escrevemos sem a necessidade de redimensioná-la. O próximo passo é a construção do menu:

```
Sub wsMenus()  
    Dim cmdBar As CommandBar  
    Dim btnDropDown As Object  
  
    Dim ws As Worksheet  
    Dim wb As Workbook  
    Dim wsNome As String  
  
    Set wb = ActiveWorkbook  
  
    On Error Resume Next  
    CommandBars(CMDBARNOME).Delete  
  
    Set cmdBar = CommandBars.Add(Name:=CMDBARNOME, _  
        Position:=msoBarFloating)  
  
    Set btnDropDown = cmdBar.Controls.Add(msoControlComboBox)  
    btnDropDown.Width = 200  
    For Each ws In wb.Sheets  
        wsNome = ws.Name  
        btnDropDown.AddItem wsNome  
    Next  
  
    btnDropDown.ListIndex = 1  
    btnDropDown.OnAction = "wsAcessar"  
    cmdBar.Visible = True  
End Sub
```

Como o controle btnDropDown não está disponível nos objeto Commandbar, o definimos como sendo um Object apenas. O método para se adicionar um item ao btnDropDown é o mesmo que utilizamos para adicionar um item a uma listbox ou combobox em um formulário. Desta forma, se houver alguma dúvida sobre os métodos e propriedades disponíveis em um msoControlComboBox podemos procurar nas propriedades e métodos de um combobox para formulário, pois ambos são quase idênticos.

O ListIndex da combobox é colocado para 1, pois queremos que o primeiro item da lista seja selecionado. Se pularmos esta linha, o primeiro item da lista aparece em branco como mostram as figuras abaixo:



Figura 8-1

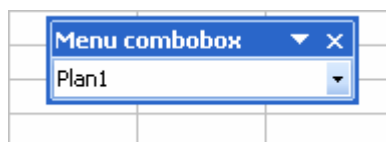


Figura 8-2

Este é apenas um detalhe e fica a critério do leitor definir em qual modo se pretende apresentar a barra de comando.

Com a barra de comando e o `btnDropDown` prontos, precisamos definir a macro que executa o `OnAction`. Em nosso código chamamos esta macro de “`wsAcessar`”. Quando um dos itens da lista é selecionado esta sub-rotina é executada.

```

Sub wsAcessar()
    Dim wsÍndice As Integer

    wsÍndice= CommandBars(CMDBARNOME).Controls(1).ListIndex
    ThisWorkbook.Sheets(wsÍndice).Activate
End Sub

```

A dimensão `wsÍndice` refere-se ao índice do item na `btnDropDown`. Contudo, como este é o índice referente à planilha, também, podemos utilizar valor para ativar a planilha selecionada na lista. Um outro ponto a ser observado é que podemos dispensar o `wsÍndice` e escrever somente uma linha de código nesta sub-rotina:

```

ThisWorkbook.Sheets(CommandBars(CMDBARNOME). _
    Controls(1).ListIndex).Activate

```

Portanto, podemos cortar o dimensionamento de `wsÍndice` e a linha que o define como sendo igual ao índice do item selecionado no controle `btnDropDown`.

Infelizmente, o método anterior deixa várias perguntas no ar: e se uma planilha for removida ou adicionada? E se uma planilha mudar de posição? Se fizermos isso, veremos que o método anterior falha em nos fornecer as respostas às questões colocadas. A solução é rever o método utilizado e procurar uma solução melhor.

Contudo, soluções não são vendidas em latinhas no mercado da esquina e precisamos pensar em uma maneira mais eficiente e pesquisar uma solução. Em programação uma solução não existe até que ela seja inventada. O próximo exemplo utiliza uma `Class` para inventar uma resposta para nossa questão.

Primeiro, vamos definir o problema. Como a `Class` manipula os eventos e as propriedades, estamos interessados em uma classe que:

- Reconheça quando uma planilha é adicionada ou removida da pasta atual
- Reconheça quando mudamos uma planilha de posição
- Reconheça a pasta de trabalho atual e atualize a `btnDropDown`

A primeira parte requer a definição do evento que desejamos capturar:

```
Public WithEvents appXL As Application
```

Os eventos são referentes ao aplicativo Excel. Veja que a definição é pública como fizemos com a constante do nome de nossa barra de comando. Assim como definimos a constante como sendo um `String` aqui definimos os eventos sendo capturados como sendo `Application`.

A seguir, escrevemos o código que manipulará o evento dentro da classe. Primeiramente, capturaremos os eventos que ocorrem na pasta ativa e dizem respeito às planilhas desta pasta:

```
Private Sub appXL_SheetActivate(ByVal Sh As Object)
    Dim btnDropDown As Object
    Dim ws As Worksheet
    Dim i As Long

    Set btnDropDown = CommandBars(CMDBARNOME).FindControl _
        (Type:=msoControlDropdown, Tag:="Lista")
    btnDropDown.Clear
    For Each ws In Sh.Parent.Sheets
        btnDropDown.AddItem ws.Name
    Next

    For i = 1 To btnDropDown.ListCount
        If btnDropDown.List(i) = Sh.Name Then _
            btnDropDown.ListIndex = i: Exit For
    Next
End Sub
```

A declaração das variáveis segue os exemplos anteriores. Em seguida, definimos o `btnDropDown` através do método `FindControl` o qual é aplicado em nossa barra de comando. A busca é feita através do `Tag` que utilizaremos na sub-rotina que desenvolveremos. O `FindControl` também pode ser utilizado com o número do índice do controle, porém, quando o desenvolvimento não é de larga escala é mais fácil visualizar o que estamos fazendo através de `Tags`.

Após a definição do objeto `btnDropDown`, limpamos quaisquer valores que estão presentes no `btnDropDown`. Embora neste exemplo isto não seja crítico é boa prática sempre limpar qualquer lista antes de adicionar uma lista nova.

Com o objeto `ws` fazemos, agora, um loop em todas as planilhas presentes na pasta ativa. Utilizamos o método `AddItem` como fizemos em nosso primeiro exemplo.

Finalmente, checamos o índice do `btnDropDown` e casamos este valor com o índice e nome da planilha a qual ele se refere.

Como isso terminamos a captura dos eventos ocorridos nas planilhas. Agora, precisamos capturar os eventos da pasta de trabalho. Aqui, teremos menos trabalhos pois queremos apenas saber a pasta atual para que o evento da planilha seja reprocessado. Assim sendo, tudo que desejamos saber é quando uma pasta nova é ativada para que o evento da planilha seja rodado:

```
Private Sub appXL_WorkbookActivate(ByVal Wb As Workbook)
    appXL_SheetActivate Wb.ActiveSheet
End Sub
```

Quando uma pasta de trabalho (workbook) é ativada o evento para a planilha é rodado e definido como a planilha ativa (`ActiveSheet`) na pasta de trabalho atual (`ActiveWorkbook`). Quando o evento ocorre as planilhas listadas no `btnDropDown` são as da pasta ativa.

Para terminar precisamos construir nossa barra de comando contendo a combobox. Depois das várias construções anteriores, não há segredo algum nesta construção, apenas algumas novidades:

```
Public Const CMDBARNOME As String = "Menu combobox"
'Dimensiona o objeto contendo os eventos como sendo a classe onde
'onde eventos foram definidos
Dim appExcel As New Class1

Sub wsMenus()
    Dim cmdBar As CommandBar
    Dim btnDropDown As Object

    Set cmdBar = CommandBars.Add(CMDBARNOME, msoBarFloating)
    cmdBar.Width = 150
    Set btnDropDown = cmdBar.Controls.Add(Type:=msoControlDropdown)
    With btnDropDown
        .Tag = "Lista"
        .OnAction = "selPlanilha"
        .Width = 150
    End With

    'Define os eventos do aplicativo que serão monitorados
    Set appExcel.appXL = Application

    With cmdBar
        .Visible = True
        .Protection = msoBarNoChangeDock + msoBarNoResize
    End With
End Sub
```

Primeiramente, precisamos acessar as informações contidas na `Classe1`. Para fazer isso dimensionamos um objeto (neste caso `appExcel`) como uma nova classe que é referida como a classe por nós criada.

Em seguida criamos nosso menu como fizemos anteriormente. Quando definimos as propriedades do `btnDropDown` acrescentamos um `Tag` ao `btnDropDown` pois é este o valor sendo procurado na `Classe1` através do `FindControl`.

Finalmente, definimos a propriedade `appXL` da dimensão `appExcel` como sendo o aplicativo (`Application`). O restante do código já foi discutido, sendo desnecessário explicar novamente.

O código anterior criar o menu e aplica os eventos ao menu. Porém, ainda precisamos definir a sub-rotina que ativa a planilha selecionada na lista. Esta sub-rotina foi chamada de `selPlanilha` na propriedade `OnAction`:

```
Sub selPlanilha()  
    Dim btnDropDown As Object  
    On Error Resume Next  
    Set btnDropDown = CommandBars.FindControl _  
        (Type:=msoControlDropdown, Tag:="Lista")  
    ActiveWorkbook.Sheets(btnDropDown.Text).Activate  
End Sub
```

Mais uma vez utilizamos a `Tag` para encontrar o controle `btnDropDown`. Quando o controle é encontrado, instruímos o Excel a selecionar a planilha cujo nome é igual ao texto contido no `btnDropDown`.

Terminamos aqui a solução de nosso problema.

Os dois exemplos deste tópico encontram-se respectivamente nas planilhas `MenuCombobox1.xls` e `MenuCombobox1.xls` que acompanham este módulo.

9. Botões Toggle

Botões do tipo toggle são úteis quando desejamos ativar ou desativar algo em nossa pasta. Vamos supor por exemplo que desejamos esconder as guias das planilhas..

O processo é um pouco lento pois precisamos ir até **Ferramentas** → **Opções** e depois selecionar a opção que desejamos. Com um botão do tipo toggle podemos fazer exatamente isso facilitando o nosso trabalho enormemente.

No tópico anterior, tivemos que criar uma classe para lidar com os eventos. Aqui, teremos que fazer a mesma coisa. O que acontece é que se mudarmos de uma pasta onde as guias estão escondidas para uma onde elas não estão, sem uma classe para lidar com o evento de mudança de uma pasta para outra o botão toggle não refletirá a mudança e permanecerá “ticado” ou não. A figura abaixo mostra um exemplo de botão toggle:

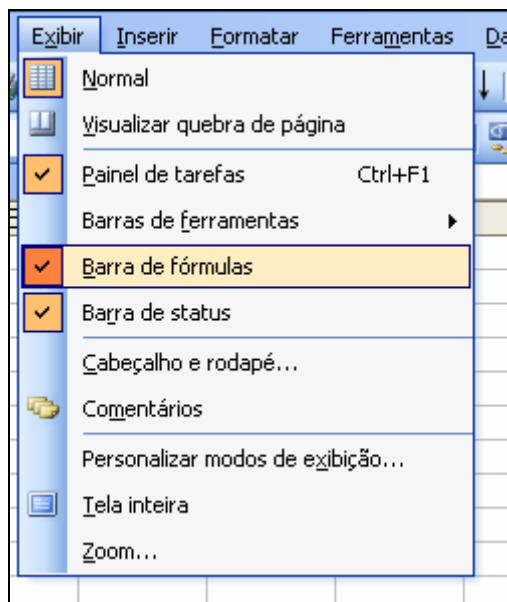


Figura 9-1

Novamente, iremos iniciar pela classe que lidará com os seguintes eventos:

- Ativação da planilha
- Ativação da pasta
- Ativação da janela

Como cada planilha dentro de uma pasta pode ter as guias ativas ou não, precisamos ter certeza que quando mudamos de planilha o estado do botão toggle também muda. O mesmo vale para a pasta de trabalho quando uma nova pasta é adicionada, por exemplo.

A ativação da janela é um outro evento, pois podemos estar trabalhando em pastas diferentes as quais acessamos conforme desenvolvemos o trabalho. Os eventos aqui são mais fáceis do que os do tópico anterior:

```
Public WithEvents appXL As Application
```

```
Private Sub appXL_SheetActivate(ByVal Sh As Object)  
    Call mostrarGuias  
End Sub
```

```
Private Sub appXL_WorkbookActivate(ByVal Wb As Workbook)  
    Call mostrarGuias  
End Sub
```

```
Private Sub appXL_WindowActivate(ByVal Wb As Workbook, _  
    ByVal Wn As Window)  
    Call mostrarGuias  
End Sub
```

O primeiro evento diz respeito a ativação da planilha (worksheet). O segundo é o evento da pasta (workbook) e o terceiro é o evento conjunto da janela (window) e da pasta (workbook). Com os eventos definidos, podemos construir o módulo que criará o menu:

```
Dim appExcel As New Class1  
Public Const MNUNOME = "Exemplo Toggle"
```

```
Sub wsMenus()  
  
    Dim mnu As CommandBarPopup  
    Dim btn As CommandBarButton  
  
    Set mnu = CommandBars(1).Controls.Add _  
        (Type:=msoControlPopup, before:=1)  
    mnu.Caption = MNUNOME  
  
    Set btn = mnu.Controls.Add(Type:=msoControlButton)  
    With btn  
        .Caption = "&Mostrar guia"  
        .OnAction = "guias"  
        .Tag = "ExemploToggle"  
    End With  
    Set appExcel.appXL = Application  
End Sub
```

Como pode ser observado a construção do menu é idêntica as anteriores. Precisamos agora construir a macro “guias” que é chamada quando clicamos no botão:

```

Sub guias()
    If TypeName(ActiveSheet) = "Worksheet" Then
        ActiveWindow.DisplayWorkbookTabs = _
            Not ActiveWindow.DisplayWorkbookTabs
        Call mostrarGuias
    End If
End Sub

```

Esta sub-rotina checa primeiramente se o tipo é uma planilha. Se sim, checamos o estado das guias. Vamos supor que mostrar a guia (DisplayWorkbookTabs) seja falso (false). No código acima estamos invertendo este valor, pois após o clique o DisplayWorkbookTabs deve assumir o valor verdadeiro. Como não sabemos de antemão qual é o estado das guias, utilizamos o estado atual das guias para definir o novo estado.

A função Not no VBA, assim como na planilha, é utilizada para inverter o valor Boolean de um expressão. Em outras palavras, se o valor é verdadeiro (true) a função Not retorna falso (false) e vice-versa. No próxima sub-rotina, isso deve ficar mais claro.

Resolvida a sub-rotina anterior, chamamos a sub-rotina que mostrará ou esconderá as guias:

```

Sub mostrarGuias()
    Dim btn As Object
    Dim btnDropDown As Object
    On Error Resume Next
    Set btn = CommandBars.FindControl _
        (Type:=msoControlButton, Tag:="ExemploToggle")

    If Not ActiveWindow.DisplayWorkbookTabs = True Then
        btn.State = msoButtonUp
    Else:
        btn.State = msoButtonDown
    End If
End Sub

```

Observe como a construção do Not tem um significado literal: If not true (se não for verdadeiro) Then (então). Em outras palavras, se ActiveWindow.DisplayWorkbookTabs for falso (false), então o estado do botão toggle é msoButtonUp. Por outro lado, se ActiveWindow.DisplayWorkbookTabs for verdadeiro (true) o estado do botão é msoButtonDown. Esta mesma construção poderia ter sido:

```

If ActiveWindow.DisplayWorkbookTabs = False Then
    btn.State = msoButtonUp
ElseIf ActiveWindow.DisplayWorkbookTabs = True
    btn.State = msoButtonDown
End If

```

Porém, com a segunda opção acrescentamos linhas extras de código desnecessárias.

O trabalho final para o botão toggle:

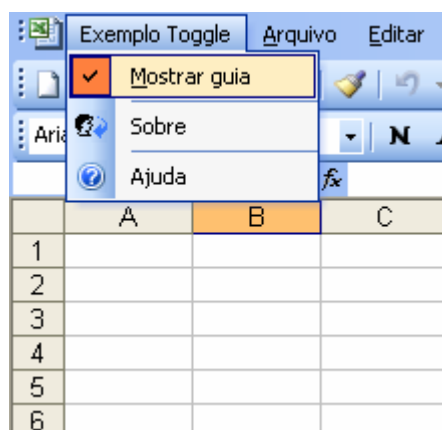


Figure 9-1

10. Criando um menu com o controle Edit

Um menu que pode ser bastante útil é o controle Edit. Como o nome sugere, ele é um controle de edição. Imagine uma caixa de texto, como a barra de fórmula, onde você pode digitar algo. Pois bem, o controle Edit serve exatamente para isso.

Muitas vezes, utilizamos um formulário para solicitar a senha e nome de usuário para acesso em um documento Excel. Ao invés de utilizar tal ferramenta, poderíamos criar o menu como segue:

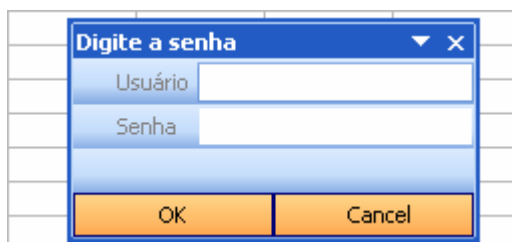


Figure 10-1

Nele, o usuário digita a senha e o nome para login. Ao clicar OK a autenticação é feita e o usuário pode ou não acessar o documento.

O problema deste tipo de senha é que o código precisa ser rodado para a construção do menu. Sem isso, o usuário continua tendo acesso ao documento. Porém, a idéia acima é apenas para estudo do leitor. O que estarei fazendo é algo diferente, pois o assunto acima já havia discutido no fórum.

A idéia é criar um código que nos dê informações em um menu popup quando clicamos em uma célula qualquer. Veja a figura:

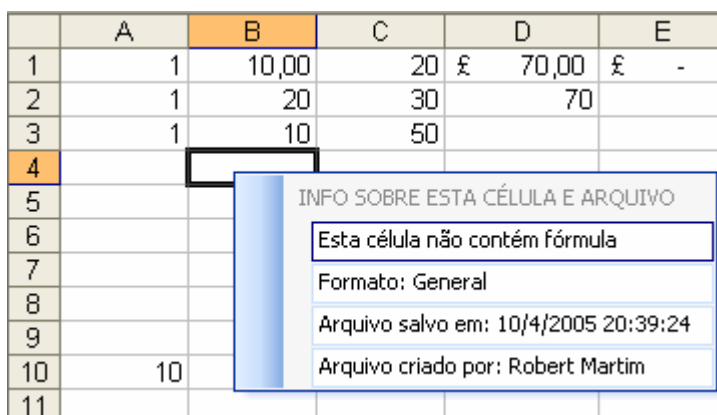


Figure 10-2

Quando o usuário clica com o botão direito do mouse sobre uma célula um popup contendo informações sobre a célula é ativado.

A criação de um menu popup já foi discutida anteriormente, porém, aqui, estarei utilizando uma forma diferente para acessar o menu de atalho. Se você ainda lembra do tópico, o menu de atalho é chamado através do método `Union`.

No exemplo que segue, utilizarei uma função para determinar se o menu deve ou não aparecer. Esta é uma forma de introduzir ao leitor coisas novas que serão úteis no futuro em outros códigos que você eventualmente criará.

Para iniciar, iremos criar o código que construirá e removerá o popup quando a pasta for aberta e fechada. Abra o VBE da pasta de trabalho onde você deve inserir os códigos:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call delPopup
End Sub

Private Sub Workbook_Open( )
    Call mnuPopup
End Sub
```

O segundo passo requer a criação do código que mostrará o menu quando clicarmos com o botão direito sobre uma área em uso da planilha. Abra o VBE da planilha onde o seguinte código deve ser inserido:

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
    Cancel As Boolean)
    ' Cancela o evento padrão do Excel para o botão direito
    Cancel = True

    ' Se a célula não estiver em uso
    If Not emUso(ActiveCell.Address) Then
    ' Não cancelar o evento padrão para o botão direito do mouse
    Cancel = False
    ' Sair da rotina
    Exit Sub
    Else:
    ' Se a célula estiver em uso
    ' Chamar a rotina "Info"
    Call info
    ' Mostrar o popup
    CommandBars( "POPUP" ).ShowPopup
    End If
End Sub
```

```

Private Function emUso(endereco As String) As Boolean
    ' Dimensiona o objeto cél
    Dim cél As Range

    ' Define o resultado inicial da função como falso
    emUso = False

    ' Checar em cada célula se ...
    For Each cél In ActiveSheet.UsedRange
        ' o endereço da célula ativa pertence ao
        ' conjunto de células usadas
        If endereco = cél.Address Then
            ' Se pertencer, o resultado da função
            ' é verdadeiro
            emUso = True
            ' Termina a função
            Exit Function
        End If
    Next cél
End Function

```

Finalmente, estamos prontos para o código que criará o menu. Como o código contém os mesmos objetos que apresentados anteriormente ele não está comentado:

```

Sub mnuPopup()

    Dim mnu        As CommandBar
    Dim edit       As CommandBarControl
    Dim btn        As CommandBarButton

    On Error Resume Next
    CommandBars("POPUP").Delete

    Set mnu = CommandBars.Add(Name:="POPUP", Position:=msoBarPopup)

    Set btn = mnu.Controls.Add(Type:=msoControlButton)
    With btn
        .Caption = "INFO SOBRE ESTA CÉLULA E ARQUIVO"
        .Width = 40
        .Enabled = False
    End With

    For i = 1 To 4
        Set edit = mnu.Controls.Add(Type:=msoControlEdit)
        edit.Width = 200
    Next i

    With mnu
        .Width = 200
        .Protection = msoBarNoChangeDock + msoBarNoCustomize _
            + msoBarNoResize
    End With
End Sub

```

O menu contém quatro controles Edit. Para adicioná-los utilizo um loop, pois todos são exatamente iguais.

Com esta parte fora do caminho, precisamos adicionar a rotina “Info” que é chamada quando o clique direito ocorre. Esta rotina irá adicionar a informações da célula aos controles Edit contidos no popup:

```
Sub info()  
    Dim ctl As CommandBarControl  
    With CommandBars("POPUP").Controls  
        On Error Resume Next  
        ' Se a célula não contém fórmula...  
        If ActiveCell.Formula = "" Then  
            ' mostre o texto "Esta célula não contém fórmula"  
            ' no controle Edit  
            .Item(2).Text = "Esta célula não contém fórmula"  
  
            ' Caso contrário...  
            ElseIf ActiveCell.Locked = True Then  
                ' Mostrar a fórmula da célula selecionada  
                .Item(2).Text = "Fórmula: " & ActiveCell.Formula  
            End If  
            ' Adicionar o formato ao controle Edit 3  
            .Item(3).Text = "Formato: " & ActiveCell.NumberFormat  
            ' Adicionar a data da última vez que foi salvo  
            ' ao controle Edit 4  
            .Item(4).Text = "Arquivo salvo em: " _  
                & ThisWorkbook.BuiltinDocumentProperties("Last Save Time")  
            ' Adicionar o nome do autor ao controle Edit 5  
            .Item(5).Text = "Arquivo criado por: " _  
                & ThisWorkbook.BuiltinDocumentProperties("Author")  
        End With  
    End Sub
```

O código para remoção do menu ficará por conta do leitor.

Ao clicar com o botão direito sobre uma área qualquer da planilha se a área estiver sendo utilizada (UsedRange) o menu será mostrado, caso contrário o popup padrão para células é mostrado.

11. Removendo os menus personalizados

Se você abriu cada arquivo que acompanha este módulo até este ponto, você deve ter notado que os menus são removidos. É sempre uma boa idéia remover os seus menus personalizados para assegurar que o próximo usuário não tenha uma desagradável surpresa ao tentar utilizar o Excel.

Neste breve tópico mostro as diversas formas de levar isso a cabo. Um exemplo óbvio é o método utilizado até o momento, isto é, utilizar o *método* Delete do controle em questão para remover o menu.

Por exemplo:

```
Sub delMenu()  
    On Error Resume Next  
    CommandBars("MENU").Delete  
End Sub
```

O exemplo acima simplesmente remove a barra de comando “MENU” do aplicativo.

Porém, você terá situações onde deseja remover um controle que foi inserido no menu Ajuda do Excel. Neste caso, o ideal é fazer uma referência ao ID do controle que contém o controle que criamos e, então, remover o controle que desejamos. Observe a figura:

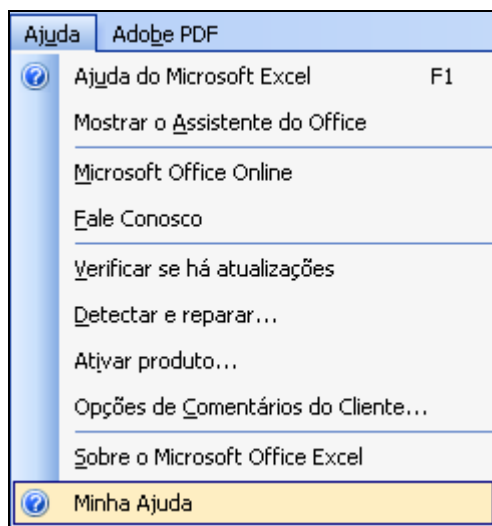


Figure 11-1

O menu foi inserido utilizando o código:

```
Sub menu()  
    Dim mnu As CommandBarPopup  
    Dim btn As CommandBarButton  
  
    Set mnu = CommandBars(1).FindControl(ID:=30010)  
    Set btn = mnu.Controls.Add(msoControlButton)  
  
    With btn  
        .Caption = "Minha Ajuda"  
        .FaceId = 984  
    End With  
End Sub
```

Para remover o código, podemos fazer o seguinte:

```
Sub delMenu()  
    On Error Resume Next  
    Dim mnu As CommandBarPopup  
    Dim btn As CommandBarButton  
  
    Set mnu = CommandBars(1).FindControl(ID:=30010)  
  
    mnu.Controls("Minha Ajuda").Delete  
End Sub
```

O método acima é um tanto quanto laborioso. A grande vantagem que vejo é o fato de você estar sempre praticando os métodos e fixando o seu conhecimento de VBA. Contudo, nem sempre podemos nos dar a este luxo e o que queremos é realmente um código limpo e curto.

Neste caso, a forma mais prática de se livrar do que você fez é:

```
Sub resetMenu()  
    Dim mnu As CommandBarPopup  
    Set mnu = CommandBars(1).FindControl(ID:=30010)  
    mnu.Reset  
End Sub
```

Porém, se a forma acima ainda não lhe parece prática por ter que declarar e instanciar o objeto, você pode resolver o problema com uma linha de código apenas:

```
Sub resetMenu2()  
    CommandBars(1).Controls(10).Reset  
End Sub
```

O que você notará é que o método Reset é mais versátil do que o Delete, principalmente porque o método Reset não resulta em erro se o objeto realmente existe e pode ser “resetado”.

Além disso, você não precisa “resetar” exatamente o objeto onde se encontra o seu controle. Se você “resetar” o objeto pai, todos os controles sob ele serão reiniciados.

Você deve ter notado que na figura acima existe um menu popup do Adobe PDF logo após o menu Ajuda. Ao utilizar a sub-rotina `resetMenu2` estou apenas “resetando” o controle de índice 10 na barra de comando 1.

Porém, eu poderia simplesmente “resetar” a barra por completo:

```
Sub resetMenu3()  
    CommandBars(1).Reset  
End Sub
```

A figura abaixo mostra a barra de menu reiniciada:

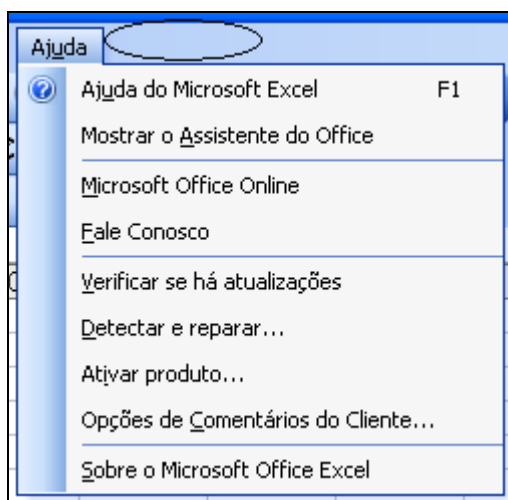


Figure 11-2

Desta vez não só o botão é removido como o menu do Adobe.

Uma outra forma de se remover menus no Excel é manualmente. Se você adicionou apenas um controle e não deseja criar código para removê-lo a melhor forma é utilizar a ferramenta de personalização de menu.

Para tanto, clique em Exibir → Barras de ferramentas → Personalizar. Com a janela de personalização aberta, basta clicar na barra de comando que contém o menu ou no menu que contém o botão, clicar sobre o controle e arrastá-lo para fora do controle principal.

12. Bloqueando e reiniciando Menus Padrões

Uma questão que vez ou outra aparece no fórum é sobre como remover os menus do Excel e colocar os menus personalizados pelo usuário.

Se uma barra de comando é um menu, então, precisamos definir o argumento "MenuBar" como sendo True para que ele possa substituir o menu principal do Excel.

Eu, particularmente, sou contra a remoção total dos menus do Excel por um simples motivo: e se o usuário precisa de um item do menu para fazer o trabalho? Com a remoção total, ele ficará preso ao que você disponibilizou o que pode não ser a forma mais eficiente.

A verdade é que a criação de menus na criação deve ser vista como um apêndice ao que já existe e adicionar ferramentas que não estão disponíveis. Em outras palavras, a idéia de criar os menus tem o intuito de aumentar a produtividade do usuário e não reduzi-la.

De qualquer modo, este é um assunto importante também e por este motivo resolvi incluí-lo nesta revisão do curso.

Primeiramente, precisamos definir o que realmente desejamos fazer. Por exemplo, você deseja remover todos os menus visíveis ou apenas substituir o menu padrão do Excel pelo seu próprio? Se a resposta é apenas substituir o menu padrão do Excel, então, podemos fazer o seguinte:

```
Sub substituirMenu()  
    Dim cmdbar As CommandBar  
    Dim popup As CommandBarPopup  
  
    Set cmdbar = CommandBars.Add(Name:="Minha barra", _  
        Position:=msoBarTop, MenuBar:=True)  
  
    Set popup = cmdbar.Controls.Add(Type:=msoControlPopup)  
    popup.Caption = "Popup 1"  
  
    Set popup = cmdbar.Controls.Add(Type:=msoControlPopup)  
    popup.Caption = "Popup 2"  
  
    Set popup = cmdbar.Controls.Add(Type:=msoControlPopup)  
    popup.Caption = "Popup 3"  
  
    cmdbar.Visible = True  
End Sub
```

Ao definir o argumento MenuBar como True, nós estamos efetivamente removendo o menu padrão do Excel pelo nosso. O último passo é simplesmente passar a propriedade Visible da barra de comando para True.

Ao rodarmos a sub-rotina:

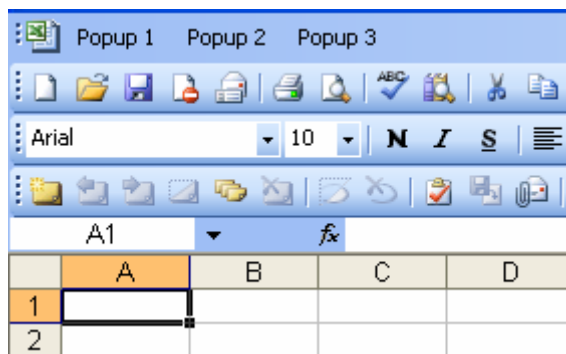


Figure 12-1

Resolvido o primeiro problema. Contudo as outras barras ainda continuam visíveis, então, como removê-las? Existem várias formas de se fazer isso, porém, mostrarei a que acredito ser a melhor forma de se resolver o problema.

Contudo, antes de continuar, você precisa saber como restaurar o seu menu original. Há várias opções, mas a mais rápida é:

```
Sub resetMenu()  
    On Error Resume Next  
    CommandBars("Minha barra").Delete  
End Sub
```

Tudo que precisamos fazer é excluir a barra que criamos sobre a barra padrão do Excel que a barra padrão volta ao seu estado original.

Voltando a questão da remoção de todos os menus, podemos utilizar a propriedade Enabled para resolver a questão de forma prática e rápida sem precisar definir exatamente quais menus somem e quais ficam.

Neste caso, podemos utilizar a seguinte sub-rotina:

```
Sub removerMenus()  
    Dim cmdbar As CommandBar  
    For Each cmdbar In Application.CommandBars  
        cmdbar.Enabled = False  
    Next cmdbar  
End Sub
```

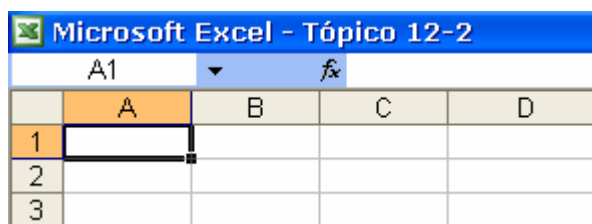


Figure 12-2

O processo para restaurar o menu é exatamente o inverso da remoção:

```
Sub restaurarMenus ()  
    Dim cmdbar As CommandBar  
  
    For Each cmdbar In Application.CommandBars  
        cmdbar.Enabled = True  
    Next cmdbar  
  
End Sub
```

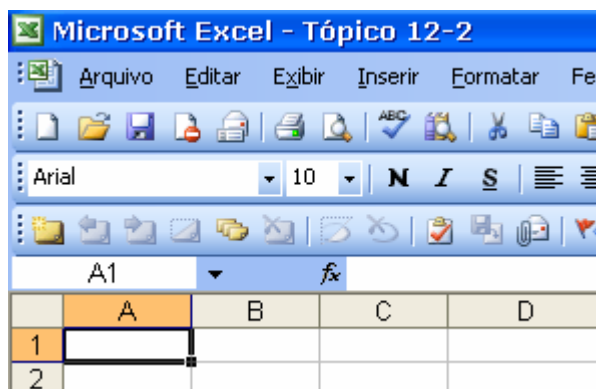


Figure 12-3

13. Criando um menu camaleão

Vários clientes que adquiriram a primeira edição deste módulo me escreveram perguntando como fazer para criar um menu “camaleão”, aquele que muda conforme a necessidade. Supondo que você se encontra na planilha **clientes**, então, um menu fica disponível. Se você se encontra na planilha **contas**, outro menu aparece e assim sucessivamente.

A princípio, estava um pouco relutante de escrever esta parte, afinal, o código para fazer isso já havia sido apresentado de forma indireta bastando apenas uma adaptação para o desejado. Contudo, após vez a real dificuldade de visualização da solução do problema, eu acredito que a decisão seja a mais acertada para que o leitor tenha em mãos um curso que seja realmente completo.

Primeiramente, precisamos visualizar o problema. O que realmente desejamos fazer? A idéia é ter um menu que seja modificado quando certa planilha é ativada. E o que é isso exatamente? Quando ativamos uma planilha nós disparamos um evento no aplicativo Excel, quando o evento é disparado podemos capturar o evento e utilizá-lo fazer uma outra coisa, como modificar a estrutura do menu ativo.

É exatamente esta idéia que precisamos colocar em prática. Uma forma simples de fazer isso é utilizar os eventos da própria planilha, por exemplo:

```
Private Sub Worksheet_Activate()  
    Call criarMenu  
End Sub
```

Quando a planilha que contém o código acima é ativada, a sub-rotina **criarMenu** é chamada e o menu construído. Podemos fazer a mesma coisa na planilha seguinte, e na seguinte, e na seguinte, etc. Porém, teríamos que fazer isso para cada planilha o que pode se tornar uma bola de neve. Além do que ela não resolve o problema da seleção de uma planilha em uma outra pasta ativa.

Vamos supor que o código acima se encontra na **Plan1** da **Pasta1**. O que ocorre se você seleciona a **Plan1** da **Pasta2**? Sem dúvida que não acontecerá o que você espera, ou melhor, não acontecerá nada.

O formato do menu será o seguinte:

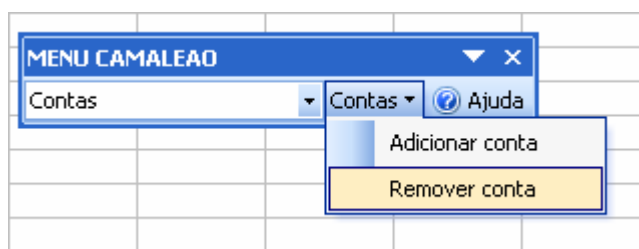


Figura 13-1

Neste caso, temos um combobox que lista as planilhas disponíveis. O popup que irá mudar é o popup contas. Quando a planilha de fornecedores for ativada, o menu será modificado para o menu fornecedores e assim por diante:

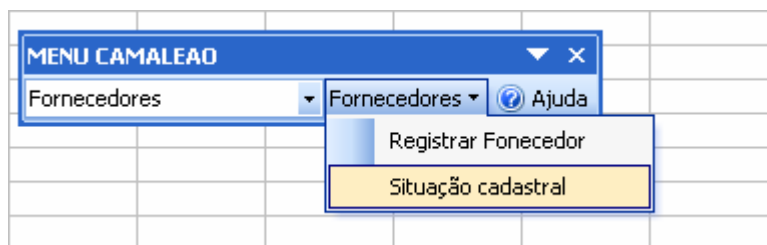


Figura 13-2

A complexidade de seu menu deve ser ditada pela sua real necessidade.

Algumas considerações antes de continuarmos:

- O que ocorre quando mudamos de planilha?
Resp: Como dito, o menu deve mudar
- O que ocorre quando mudamos de pasta?
Resp: O menu atual é removido e o combobox desativado
- Qual o nível de complexidade?
Resp: Dependerá das suas necessidades, porém, eventos não existentes precisam ser criados através de classes e instanciados

Para o momento, estes são os três pontos que precisamos ter em mente. O primeiro, já havia discutido, o segundo é um extra e o terceiro é apenas para lembrá-lo que o código pode se tornar muito mais complexo do que o atual.

Então, vamos iniciar pela parte mais simples. Abra a janela de código da pasta de trabalho. Nela você deverá inserir as seguintes linhas:

Autor: [Robert F Martim](#)
Publicado: www.juliobattisti.com.br
Contato: rm@faircourt.com

Criado em: 19/5/2004 11:28:00
Última edição: 12/4/2005 20:00:00

```

[-] Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call wsMenuDel
End Sub

[-] Private Sub Workbook_Open( )
    Call wsMenus
End Sub

```

As rotinas funcionam nos eventos **Open** e **BeforeClose** da pasta de trabalho.

Iniciaremos pela declaração de algumas variáveis comuns a todas as sub-rotinas. Adicione um módulo e no topo do módulo adicione as seguintes variáveis e constantes:

```

'Constante públicas utilizadas no código
Public Const CMDBARNOME As String = "MENU CAMALEAO"
Public Const MENUFORNECEDORES As String = "Fornecedores"
Public Const MENUCLIENTES As String = "Clientes"
Public Const MENUCONTA As String = "Contas"

'Dimensiona os objetos contendo os eventos como sendo a classe onde
'onde eventos foram definidos
'appExcel refere-se aos eventos do aplicativo (Excel)
Dim appExcel As New Class1
' cboBox refere-se aos eventos da combobox
Dim cboBox As New Class1

```

Os objetos referentes à **Classe1** serão discutidos nos comentários da classe.

O próximo passo é escrever todas as sub-rotinas para criação, remoção e reconstrução do menu.

No mesmo módulo, você deverá adicionar as seguintes sub-rotinas:

```

[-] Sub wsMenus( )
    ' Dimensiona os objetos
    Dim cmdBar As CommandBar
    Dim btn As CommandBarButton
    Dim btnDropDown As Object

    ' Remove a barra de comando caso ela já exista
    Call wsMenuDel

    ' Cria a nova barra de comando
    Set cmdBar = CommandBars.Add(CMDBARNOME, msoBarFloating)
    cmdBar.Width = 150

    ' Adiciona o menu dropdown
    Set btnDropDown = cmdBar.Controls.Add(Type:=msoControlDropdown)
    With btnDropDown
        ' A Tag (etiqueta) é utilizada na classe
        .Tag = "Lista"
        ' Ação a ser executada
        .OnAction = "selPlanilha"
        .Width = 150
    End With

    ' Adiciona um botão de ajuda a barra de comando
    Set btn = cmdBar.Controls.Add(Type:=msoControlButton)

```

```

        With btn
            .Caption = "Ajuda"
            .OnAction = "ajuda"
            .Style = msoButtonIconAndCaption
            .FaceId = 984
        End With

        'Define os eventos do aplicativo que serão monitorados
        Set appExcel.appXL = Application

        'Define a combobox que deve ser monitorada pela Classel
        cboBox.setDrop btnDropDown

        With cmdBar
            .Visible = True
            .Protection = msoBarNoChangeDock + msoBarNoResize
        End With
    End Sub

```

```

Sub wsMenuDel()
    On Error Resume Next
    Application.CommandBars(CMDBARNOME).Delete
End Sub

```

```

Sub selPlanilha()
    Dim btnDropDown As Object
    On Error Resume Next
    Set btnDropDown = CommandBars.FindControl( _
        Type:=msoControlDropdown, Tag:="Lista")
    ActiveWorkbook.Sheets(btnDropDown.Text).Activate
End Sub

```

```

Sub mnuContas()
    Dim cmdBar           As CommandBar
    Dim menu              As CommandBarPopup
    Dim btn               As CommandBarButton

    Set cmdBar = CommandBars(CMDBARNOME)

    ' Remove os botões antes de adicionar o novo
    On Error Resume Next
    cmdBar.Controls(MENUCONTA).Delete
    cmdBar.Controls(MENUCLIENTES).Delete
    cmdBar.Controls(MENUFORNECEDORES).Delete

    ' Adiciona o botão antes do botão "Ajuda"
    Set menu = cmdBar.Controls.Add(Type:=msoControlPopup, BEFORE:=2)
    menu.Caption = MENUCONTA
    Set btn = menu.Controls.Add(Type:=msoControlButton)
    btn.Caption = "Adicionar conta"

    Set btn = menu.Controls.Add(Type:=msoControlButton)
    btn.Caption = "Remover conta"

End Sub

```

```
Sub mnuFornecedores()  
    Dim cmdBar As CommandBar  
    Dim menu As CommandBarPopup  
    Dim btn As CommandBarButton  
  
    Set cmdBar = CommandBars(CMDBARNOME)  
  
    On Error Resume Next  
    cmdBar.Controls(MENUCONTA).Delete  
    cmdBar.Controls(MENUCLIENTES).Delete  
    cmdBar.Controls(MENUFORNECEDORES).Delete  
  
    Set menu = cmdBar.Controls.Add(Type:=msoControlPopup, BEFORE:=2)  
    menu.Caption = MENUFORNECEDORES  
  
    Set btn = menu.Controls.Add(Type:=msoControlButton)  
    btn.Caption = "Registrar Fornecedor"  
  
    Set btn = menu.Controls.Add(Type:=msoControlButton)  
    btn.Caption = "Situação cadastral"  
  
End Sub
```

```
Sub mnuClientes()  
    Dim cmdBar As CommandBar  
    Dim menu As CommandBarPopup  
    Dim btn As CommandBarButton  
  
    Set cmdBar = CommandBars(CMDBARNOME)  
  
    On Error Resume Next  
    cmdBar.Controls(MENUCONTA).Delete  
    cmdBar.Controls(MENUCLIENTES).Delete  
    cmdBar.Controls(MENUFORNECEDORES).Delete  
  
    Set menu = cmdBar.Controls.Add(Type:=msoControlPopup, BEFORE:=2)  
    menu.Caption = MENUCLIENTES  
  
    Set btn = menu.Controls.Add(Type:=msoControlButton)  
    btn.Caption = "Registrar Cliente"  
  
    Set btn = menu.Controls.Add(Type:=msoControlButton)  
    btn.Caption = "Situação pgto"  
  
End Sub
```

```

Sub mnuRemove()
    Dim cmdBar As CommandBar

    Set cmdBar = CommandBars(CMDBARNOME)

    ' Remove todos os botões
    On Error Resume Next
    cmdBar.Controls(MENUCONTA).Delete
    cmdBar.Controls(MENUCLIENTES).Delete
    cmdBar.Controls(MENUFORNECEDORES).Delete
End Sub

Sub ajuda()
    ActiveWorkbook.FollowHyperlink _
        "http://www.juliobattisti.com.br", _
        NewWindow:=True, AddHistory:=True
End Sub

```

As três sub-rotinas para reestruturação do menu podem ser colocadas em uma única rotina, porém, deixarei esta parte como exercício para o leitor. Tudo que o leitor precisa fazer é observar os pontos em comuns nas sub-rotinas e utilizá-los na construção de uma única sub.

Com as sub-rotinas resolvidas, adicione uma classe ao seu projeto. Lembre-se que nos referimos à classe como **Classe1** quando declaramos os objetos **Application** e **ComboBox**. Portanto, este deve ser o nome da classe. Caso queira, utilize um outro nome qualquer, mas não esqueça de modificar no módulo.

Abaixo se encontra a sequência na classe com os comentários para melhor compreensão do desenvolvimento do código:

```

'Declara os objetos que serão monitorados
Public WithEvents appXL As Application
Public WithEvents drop As Office.CommandBarComboBox

'Rotina para monitorar o ativamento de planilha
Private Sub appXL_SheetActivate(ByVal Sh As Object)
    Dim btnDropDown As Object
    Dim ws As Worksheet
    Dim wb As Workbook
    Dim i As Long

    ' Define o objeto wb como sendo a pasta ativa
    Set wb = ActiveWorkbook

    ' Se a pasta ativa não for a "Tópico 13.xls", então
    ' sair da rotina
    If Not wb.Name = "Tópico 13.xls" Then Exit Sub

    ' Define o objeto btnDropDown como sendo a combobox na
    ' barra de comando CMDBARNOME. O controle é encontrado
    ' através da etiqueta (Tag)

```



```

Set btnDropDown = CommandBars(CMDBARNOME).FindControl _
    (Type:=msoControlDropDown, Tag:="Lista")

' Limpa a combobox
btnDropDown.Clear

' Para cada planilha na pasta de trabalho
For Each ws In Sh.Parent.Sheets
' Adicionar o nome da planilha a combobox
    btnDropDown.AddItem ws.Name
Next

' Define o item ativa na combobox
For i = 1 To btnDropDown.ListCount
    If btnDropDown.List(i) = Sh.Name Then _
        btnDropDown.ListIndex = i: Exit For
Next

' Chama a rotina de mudança na combobox
' Observe que nao somente estamos monitorando o aplicativo
' como também a combobox
Call drop_Change(btnDropDown)

End Sub

```

```

Private Sub appXL_WorkbookActivate(ByVal wb As Workbook)

Dim btnDropDown As Object

Set btnDropDown = CommandBars(CMDBARNOME).FindControl _
    (Type:=msoControlDropDown, Tag:="Lista")

' Se a pasta de trabalho atual for a "Tópico 13.xls"
If wb.Name = "Tópico 13.xls" Then
' Entao, ativar a combobox e
    btnDropDown.Enabled = True
' chamar a rotina de ativacao de planilha
    appXL_SheetActivate wb.ActiveSheet
Else:
' Se nao for, entao remover o menu
    Call mnuRemover
' E desabilitar a combobox
    btnDropDown.Enabled = False
End If

End Sub

```

```

Public Sub setDrop(box As Office.CommandBarComboBox)
' Define a combobox como sendo a "box"
' Esta ativacao é feita quando a rotina de criacao
' do menu é rodada. A linha no módulo que faz isso é
' "cboBox.setDrop btnDropDown"
Set drop = box

End Sub

```

```

Private Sub drop_Change(ByVal Ctrl As Office.CommandBarComboBox)
    Dim macro As String

    ' Aqui, chamo de "macro" apenas para me lembrar que ao selecionar
    ' uma planilha da lista na combobox uma macro deve ser rodada.
    ' A "macro" é o texto contido na combobox
    macro = Ctrl.Text

    ' Seleciona o caso "macro"
    ' Para cada caso, uma rotina diferente é chamada
    Select Case UCase(macro)
        Case "FORNECEDORES"
            Call mnuFornecedores
        Case "CLIENTES"
            Call mnuClientes
        Case "CONTAS"
            Call mnuContas
        Case Else
            Call mnuRemover
    End Select
End Sub

```

Montado o nosso menu, ao rodá-lo dentro da planilha de criação o menu será reajustado de acordo com a planilha selecionada:

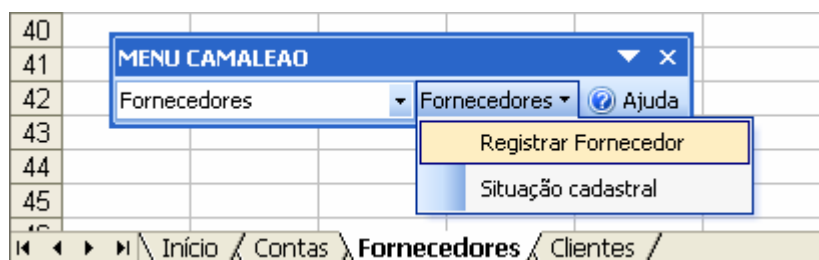


Figure 13-1

Caso você ative (ou crie) uma nova pasta de trabalho, novamente o menu é reajustado e a combobox travada:

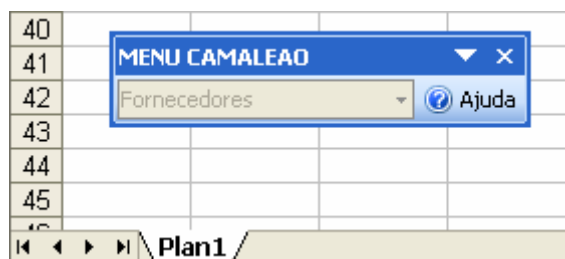


Figure 13-2

14. Automatizando a criação de menus

Até este ponto criamos vários menus e o leitor deve ter percebido que se o volume de menus e botões for elevado teremos um grande problema, pois o código ficará muito longo.

Esta parte é dedicada à automação deste processo. Embora os métodos anteriores sejam considerados “automações”, aqui automação quer dizer separar o código dos itens que compõem a barra de comando e menus.

Vamos supor que desejamos inserir um menu antes do menu Arquivo no Excel. Este novo menu conterá 8 diferentes botões e dois popups. O primeiro popup conterá 5 botões e o segundo popup conterá 6 botões. Coloque agora no contexto dos códigos anteriores.

É desnecessário dizer que o código para a construção de tal menu será enorme. Sem contar que se houver necessidade de acrescentar ou retirar menus e botões temos que editar diretamente no código. Você esqueceu um acento e tem que correr para o código. Um menu foi inserido na ordem errada e lá vamos nós para o código novamente.

Com certeza deve haver uma forma mais fácil para fazer isso, certo? Correto. Contudo, formas mais fáceis de fazer algo estão sempre evoluindo. Antes viajar de cavalo era eficiente, depois foi carro e hoje é avião. Em breve será foguetes e quem sabe um dia viajaremos por teletransportação como no Jornada nas Estrelas.

Os métodos apresentados funcionam assim. Hoje, eles são eficientes. Amanhã alguém inventa uma idéia melhor. Portanto, ao desenvolver os métodos apresentados pense em como você pode melhorá-los.

Deixando o papo furado de lado, vamos ao que interessa: como melhorar o procedimento de criação de menus.

Primeiramente, precisamos estar claros do que desejamos fazer. Por exemplo, queremos acrescentar um FacelID? Queremos acrescentar um atalho? Qual a ação a ser executada? O que está sendo adicionado é uma barra de comando, menu ou botão? Em que ordem eles entram? Onde eles estarão posicionados? Flutuando ou anexado ao menu principal do Excel?

Estas são perguntas que precisamos fazer antes de construirmos qualquer coisa. Observe que quando perguntamos algo, efetivamente já temos as respostas do que procuramos. Pense nisso. Você sabe as respostas, porém, não sabe como implementá-las. Esta é a diferença. Portanto, se há uma pergunta tem que haver uma resposta, caso contrário a pergunta é inválida.

Como isto em mente estamos prontos para iniciar a construção de nossa folha de solução. Aqui, passarei o meu raciocínio por trás da solução por mim apresentada.

O Excel é uma grande matriz e utilizaremos isso em nosso favor. Por exemplo, imagine a Plan1 como sendo a nossa barra de comando. Tudo que está dentro da Plan1 é, portanto, um item de nossa barra de comando, por analogia.

Agora imagine a primeira linha com diversos cabeçalhos e embaixo de cada cabeçalho um série de itens. Se cada cabeçalho é um menu (popup), então, cada item abaixo é um botão deste popup.

Outra forma de visualizar isso é dizer que cada planilha é um menu (popup) e os itens contidos dentro de cada planilha representam os botões destes popups. O nome da barra de comando seria, digamos, o nome da pasta.

Como podemos ver, com um pouco de imaginação estamos contemplando formas diferentes de solucionar o mesmo problema. Como cada objeto tem suas propriedades, precisamos definir quais propriedades nos interessam. Para o primeiro exemplo estaremos utilizando as seguintes propriedades:

- Name/Caption
- Faceld (para botões somente)
- OnAction (para botões somente)
- ShortcutText (para botões somente)
- BeginGroup

Como a barra de comando é composta por três objetos, estaremos manipulando estes três casos:

- CMDBAR (a barra de comando em si)
- MMENU (o menus tipo popup)
- BOTAO (os botões que executarão os comandos)

Para evitar erro na digitação (o que nos causaria dores de cabeça no código) os três itens acima serão selecionados a partir de uma lista de validação.

Uma vez construída a sua tabela em uma planilha do Excel, o formato geral deve ter a seguinte cara:

	A	B	C	D	E	F
1	PROPRIEDADES					
2	Tipo	Name/Caption	Faceld	OnAction	ShortcutText	BeginGroup
3	CMDBAR	BARRA DE COMANDO				
4	MMENU	MENU1				
5	BOTAO	BTN1	326		CTRL+1	
6	BOTAO	BTN2	327		CTRL+2	

Figura 14-1

Como cada propriedade está escrita em inglês no VBA, os cabeçalhos também foram inseridos em inglês para facilitar a referência, leitura e compreensão.

Como de praxe, iniciamos o nosso código com a declaração das variáveis:

```
Sub MenusAutomatizados()

    Dim ws      As Worksheet
    Dim linha   As Long
    Dim cBar    As CommandBar
    Dim mnu     As CommandBarPopup
    Dim btn     As CommandBarButton

    Set ws = ThisWorkbook.Sheets("ItensMenus")
    linha = 3
End Sub
```

A variável `ws` é importante pois precisamos definir onde estão os itens que fazem parte de nossa barra de comando. A variável `linha` é importante, pois efetuaremos um loop na primeira coluna para varrer todos os itens entrados. Como o primeiro item está na linha número 3 o valor inicial de nossa linha é definido como sendo 3.

O nosso próximo passo é definir os casos:

```
Sub MenusAutomatizados ()
    'Aqui entram as linhas apresentadas acima
    With ws
        Do Until IsEmpty(.Cells(linha, 1))
            TIPO = .Cells(linha, 1)
            Select Case TIPO
                Case "CMDBAR"
                    'Código de construção da barra entrará aqui
                Case "MMENU"
                    'Código de construção da menu entrará aqui
                Case "BOTAO"
                    'Código de construção do botão entrará aqui
            End Select
            linha = linha + 1
        Loop
    End With
End Sub
```

O que o código acima está fazendo é definir o TIPO conforme o loop ocorre, seleciona o caso do tipo (ie “CMDBAR”, “MMENU” ou “BOTAO”). As casos estão entre aspas porque são textos (string). Uma vez decidido qual é o caso contido na ws.cells(linha,1) o caso é executado.

Como os objetos são inseridos em ordem, precisamos definir isso claramente. Como foi dito anteriormente, se em um planilha temos um coluna com um cabeçalho que representa o menu, então, tudo abaixo deste cabeçalho representa os botões deste menu.

A estrutura apresentada acima segue esta mesma lógica. Se esta ordem não for seguida, os botões serão adicionados aos menus errados.

Definindo o primeiro caso (`Case "CMDBAR"`), temos:

```
Case "CMDBAR"  
    Set cBar = CommandBars.Add _  
        (Name:=.Cells(linha, 2), Position:=msoBarFloating)
```

Se este foi o caso encontrado durante o loop ele é executado. O nosso próximo caso é o menu popup (`Case "MMENU"`). A construção deste caso é:

```
Case "MMENU"  
    Set mnu = cBar.Controls.Add(Type:=msoControlPopup)  
    mnu.Caption = ws.Cells(linha, 2)
```

Como o rótulo (`Caption`) encontra-se na segunda coluna, basta definir a linha atual do loop como sendo o local onde o texto se encontra. O mesmo vale para o `BeginGroup`.

Finalmente, temos o caso do botão (`Case "BOTAO"`):

```
Case "BOTAO"  
    Set btn = mnu.Controls.Add(Type:=msoControlButton)  
    With btn  
        .Caption = ws.Cells(linha, 2)  
        .FaceId = ws.Cells(linha, 3)  
        .OnAction = ws.Cells(linha, 4)  
        .ShortcutText = ws.Cells(linha, 5)  
        .BeginGroup = ws.Cells(linha, 6)  
    End With
```

Aqui, as propriedades do botão são inseridas conforme a sua respectiva posição na tabela apresentada acima.

Montando o quebra-cabeça do código acima temos:

```
Sub MenusAutomatizados ()

Dim ws As Worksheet
Dim linha As Long
Dim cBar As CommandBar
Dim mnu As CommandBarPopup
Dim btn As CommandBarButton

Set ws = ThisWorkbook.Sheets("ItensMenus")
linha = 3

delBarra 'Deleta a barra anterior caso ela exista
With ws
Do Until IsEmpty(.Cells(linha, 1))
TIPO = .Cells(linha, 1)
Select Case TIPO
Case "CMDBAR"
Set cBar = CommandBars.Add _
(Name:=.Cells(linha, 2), Position:=msoBarFloating)
Case "MMENU"
Set mnu = cBar.Controls.Add(Type:=msoControlPopup)
mnu.Caption = ws.Cells(linha, 2)
Case "BOTAO"
Set btn = mnu.Controls.Add(Type:=msoControlButton)
With btn
.Caption = ws.Cells(linha, 2)
.FaceId = ws.Cells(linha, 3)
.OnAction = ws.Cells(linha, 4)
.ShortcutText = ws.Cells(linha, 5)
.BeginGroup = ws.Cells(linha, 6)
End With
End Select
linha = linha + 1
Loop
End With
'Utiliza o mesmo argumento que o BeginGroup
cBar.Visible = ws.Cells(3, 6)
End Sub
```

Como podemos ver, o volume de código é drasticamente reduzido e uma vez que ele esteja funcionando, precisamos somente nos concentrar na digitação dos itens que compõem nossa barra de comando.

Após a execução de nosso código, obtemos o seguinte resultado baseado nas informações digitadas na planilha contendo os itens do menu:

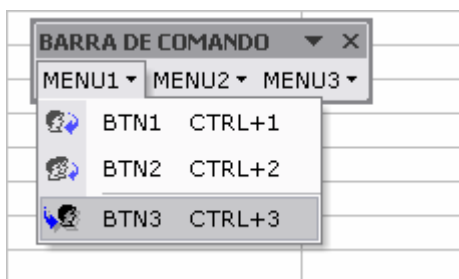


Figura 14-2

Para provar a versatilidade deste métodos, iremos acrescentar mais um caso: “ATALHOS”. Aqui, estamos interessados em inserir um botão de atalho no menu de atalho para as células (barra de comando número 29). À nossa lista anterior vamos acrescentar a palavra “ATALHO” conforme a figura abaixo:

	A	B
1		
2	Tipo	Name/Caption
3	CMDBAR	BARRA DE COMANDO
4	CMDBAR	MENU1
5	MMENU	BTN1
6	BOTAO	BTN2
7	ATALHO	BTN3
8	MMENU	MENU2

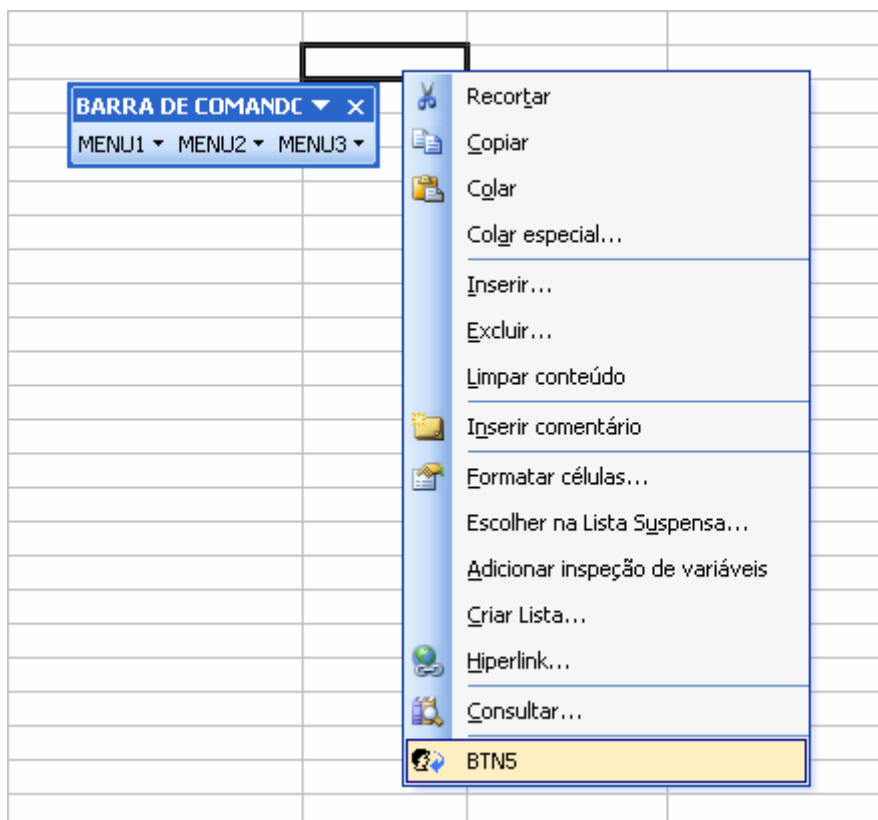
Figura 14-3

Assim sendo, temos mais um caso (Case) a ser avaliado no código anterior. Conforme o loop ocorre o novo caso sendo avaliado é:

```
Case "ATALHO"
Set btn = CommandBars(.Cells(linha, 7)).Controls.Add
With btn
.Caption = ws.Cells(linha, 2)
.FaceId = ws.Cells(linha, 3)
.OnAction = ws.Cells(linha, 4)
.BeginGroup = ws.Cells(linha, 6)
End With
```

Basta inserir o código acima no código final anterior e temos mais caso que é devidamente tratado pelo VBA.

Após a execução tudo é construído e temos um belo resultado:

**Figura 14-4**

O método utilizado acima é o mesmo método utilizado pelo Excel para listar todos os menus e comandos disponíveis. Se você pretende criar menus mais complexos, estude a maneira como o Excel lista as barras de comandos. Utilize a pasta de trabalho `Listar Menus` para fazer isso.

Terminamos aqui a nossa jornada no mundo dos menus em Excel. Qualquer dúvida ou sugestão, não hesite em contatar o autor no e-mail abaixo.

O autor pode ser encontrado diariamente no fórum Júlio Battisti no endereço abaixo respondendo perguntas sobre diversos aspectos do Excel.

15. Sobre o autor

FORMAÇÃO ACADÊMICA:

- Formado e Pós-Graduado em Finanças pela Universidade de Londres, Reino Unido
- Membro da Sociedade Brasileira de Econometria

LINGUAGENS DE PROGRAMAÇÃO E PLATAFORMAS:

- Visual Basic, Calculadores Programáveis Casio e Sharp
- BDs: MS Access and Lotus Approach
- Plataformas: Windows NT, 2000, XP, Linux Red Hat

EXPERIÊNCIA PROFISSIONAL

out 02-abr 04 **FAIRCOURT CAPITAL LIMITED (REINO UNIDO)**

- ***Diretor TI***

fev96-maio02 **MELVALE GROUP (REINO UNIDO)**

- ***Gerente de Exportação para a África Ocidental***
- ***Gerente de TI***

OUTRAS ESPECIALIZAÇÕES

- Inspeção e regulamentações Nigerianas para importação e exportação (Nigerian-British Chamber of Commerce & Cotecna International)
- Procedimentos de exportação no Reino Unido (The Institute of Export, Reino Unido)
- ICC 500 e Incoterms (The Institute of Export, Reino Unido)

OUTRAS ATIVIDADES

Fornece suporte *pro bono* em TI à entidade de caridade Nigeriana NIDOE (Nigerians in Diaspora Organisation Europe) desde 2001. Participou ativamente na organização da conferência sobre Boa Governança e Responsabilidade Fiscal promovida pelo ONG em Abuja, Nigéria, em Novembro 2003. Foi um dos principais colaboradores na elaboração do relatório final sobre a conferência entregue a presidência da República Nigeriana em maio de 2004.

Autor do livro Access e VBA na Modelagem Financeira: Uma abordagem prática (no prelo). Editora Axcel Books, 2005.

Colaborador ativo do fórum Access Avançado do site www.juliobattisti.com.br, onde divide seu conhecimento e experiência com outros membros do espaço.

Colunista dos sites www.linhadecodigo.com.br e www.ativoaccess.com.br

Autor: Robert F Martim
Publicado: www.juliobattisti.com.br
Contato: rm@faircourt.com

Criado em: 19/5/2004 11:28:00
Última edição: 12/4/2005 20:00:00