Introdução
Ao
OpenOffice.org Basic

por

Noelson Alves Duarte

Índice

Apresentação

Este documento é o resultado do meu esforço para aprender o OpenOffice.org Basic.

Ele está focado, principalmente, nas extensões do OpenOffice.org ao Basic, isto é, na API do OpenOffice.org. Assim sendo, as características básicas da linguagem de programação Basic não são abordadas.

Se você já programou ou conhece algum dos "sabores" Basic e deseja escrever macros para o OpenOffice.org este é um bom local para começar. Senão, recomendo procurar algum curso na rede, e se familiarizar com o Basic, antes de se aventurar no OOo Basic. É muito importante dominar os fundamentos de uma linguagem de programação.

Todos os exemplos de código fonte foram testados com o OpenOffice.org, versão 1.0.1, exceto os pequenos blocos de código fora das sub-rotinas.

Para utilizar este texto, o OpenOffice.org deve estar instalado em seu computador. Após o Capítulo 3, recomendo que você instale, também, o Manual de Referência da API do OpenOffice.org (consulte o capítulo Mais Informações). O Manual de Referência sofre atualizações periódicas e sua versão atualizada está disponível, também, para consultas "on-line".

Espero ter tempo para continuar aprendendo e acrescentando informações úteis aos programadores que, como eu, não dominam a língua inglesa. Portanto, periodicamente, procure por novas versões deste documento.

Como resultado de uma aprendizagem, esta Introdução, seguramente, contém muitos erros e inconsistências, espero contar com a sua ajuda para corrigí-los.

Em 20 de junho de 2003

O Autor

1 Macros OpenOffice.org Basic

1.1 Introdução

Uma macro é um programa escrito na linguagem OpenOffice.org Basic com a finalidade de automatizar tarefas do OpenOffice.org.

A linguagem OpenOffice.org Basic mantém as principais características das versões atuais do BASIC, no que diz respeito à sintaxe, tipos de dados, operadores, comandos, funções internas e organização geral do programa. Além disto, o OpenOffice.org Basic permite o acesso a uma grande quantidade de objetos, com seus métodos e propriedades, específicos do OpenOffice.

O OpenOffice.org tem um IDE (**I**ntegrated **D**evelopment **E**nvironment - Ambiente de Desenvolvimento Integrado) completo, incluíndo: edição de código fonte, verificação de erros, criação de diálogos e gerenciamento de bibliotecas.

1.2 Criando a Primeira Macro

A nossa macro será bem simples. O operador vai digitar uma frase numa caixa de entrada, em seguida esta frase será adicionada na posição corrente do cursor de texto de um documento ativo do Writer.

Para começar:

a) Execute o **Writer**, crie um novo ducumento e salve-o como **Primeira_Macro.sxw**.

b) Na barra de menu, selecione **Ferramentas – Macro**. O diálogo Macro será exibido.

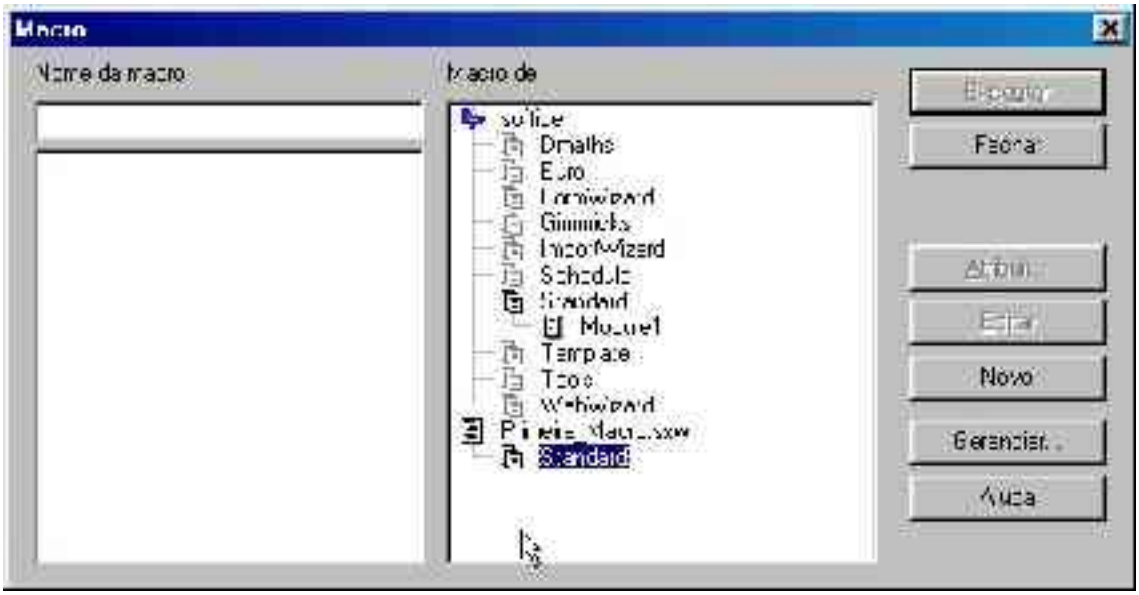


Figura 1: Diálogo Macro

Observe a árvore **Macro de**, no centro do diálogo, temos duas entradas principais **soffice** e **Primeira_Macro.sxw**, elas são recipientes (containers) para bibliotecas. Um nível abaixo, nas duas entradas, temos a biblioteca **Standard.** Dentro de **soffice - Standard** temos o módulo **Module1**.

O recipiente **soffice** sempre será carregado com o OpenOffice.org, ele tem bibliotecas globais, usadas em todos os documentos do OpenOffice.org. Observe que alguns módulos de **soffice** estão desativados.

Cada documento aberto tem seu próprio container com uma biblioteca Standard. Na figura acima, o único documento aberto é Primeira_Macro.sxw.

A lista, à esquerda do diálogo, exibe os nomes dos procedimentos (Sub-rotinas e funções) do módulo selecionado. O campo **Nome da macro** exibe o nome do procedimento selecionado. Ao selecionar o nome de um procedimento, os botões **Executar**, **Atribuir** e **Editar** são ativados.

Vamos continuar com a criação da nossa primeira macro:

a) Selecione a biblioteca **Standard** de **Primeira_Macro.sxw**.

b) Clique sobre o botão **Novo** para criar um novo módulo.

c) Aparece a caixa de entrada Novo Modulo, sugerindo o nome Module1.

d) Clique no botão OK para criar o novo módulo. O IDE Basic é carregado.

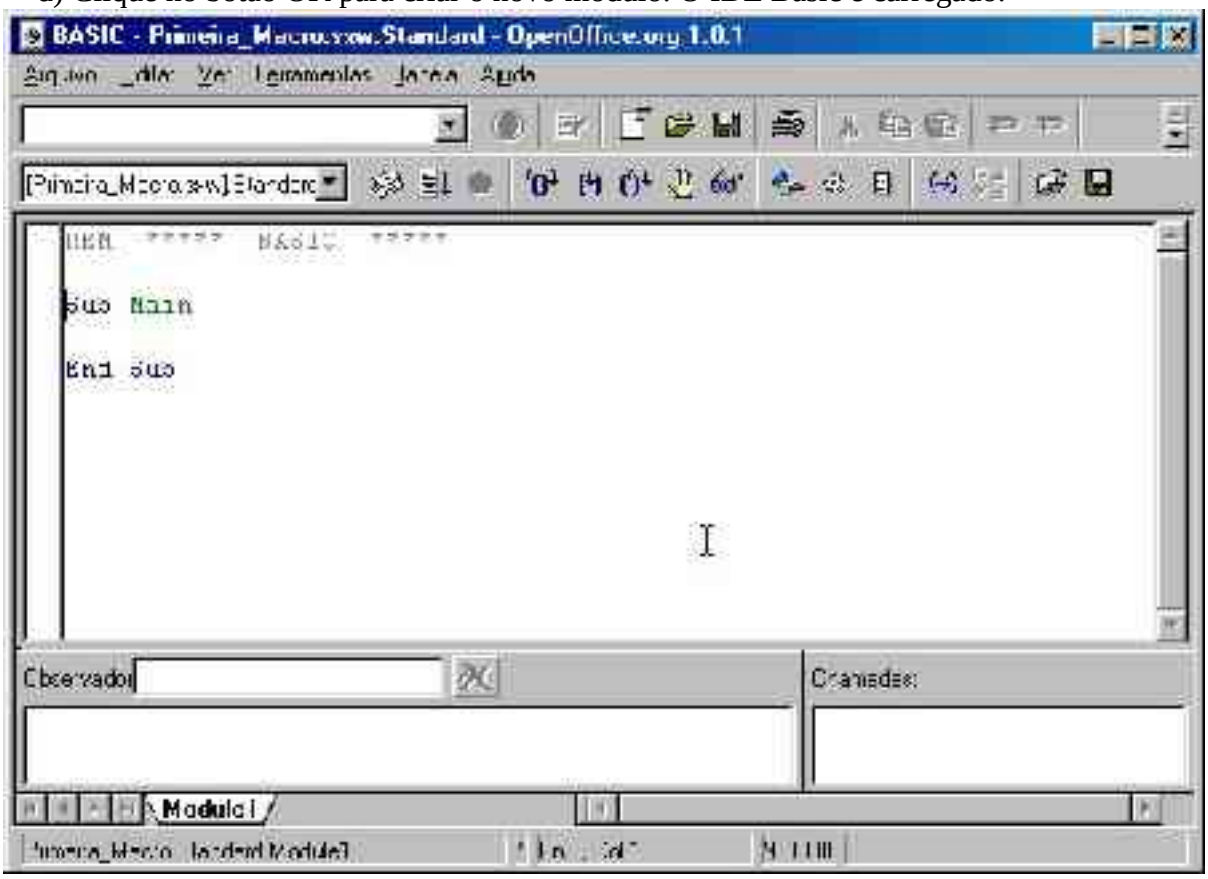


Figura 2: IDE Basic

e) Observe, no editor Basic, a definição de uma sub-rotina *Sub Main ... End Sub*. Note também que a aba **Module1** está ativa.

f) Digite (ou copie e cole) o código fonte abaixo entre as linhas *Sub Main* e *End Sub*.

```
Dim oDocumento as Object
Dim oTexto as Object
Dim oCursorVista as Object
Dim oCursor as Object
Dim sInserir as String

' solicita o texto
sInserir = InputBox("Digite o texto:", "Inserir Texto", "")
' testa se algo foi digitado
if sInserir = "" then
   exit sub
endif
' obtém o modelo do documento
oDocumento = ThisComponent
' obtém o serviço Text do documento
oTexto = oDocumento.getText()
' obtém a posição do cursor na GUI e cria um cursor de texto
oCursorVista = oDocumento.getCurrentController().getViewCursor()
oCursor = oTexto.createTextCursorByRange(oCursorVista.getStart())
' insere o texto na posição corrente do cursor
oTexto.insertString(oCursor, sInserir, FALSE)
```

Pronto, terminamos a nossa primeira macro. Lembre-se que ela está embutida no documento Primeira_Macro.sxw.

1.3 Executando a Primeira Macro

Antes de continuarmos, vamos analisar a janela do IDE Basic. Temos, na parte inferior da janela, uma área **Observador**, à esquerda, e outra **Chamadas**, à direita. A primeira será usada para observar valores de variáveis e a segunda mostra as chamadas a procedimentos, tudo durante a execução da macro.

Vejamos, também, a finalidade de alguns ícones da janela do IDE Basic:

Executar – executa uma macro até encontrar um Ponto de Interrupção.

Ponto de interrupção – define um ponto de interrupção.

Passo a passo – executa uma macro, linha a linha.

Parar – termina a execução de uma macro. Será ativado quando a execução da macro iniciar.

Macros – permite selecionar uma macro para execução.

Salvar código fonte como – salva o módulo corrente como um **arquivo Basic**.

Inserir código fonte - insere o código fonte de um **arquivo Basic**.

É chegada a hora de testarmos a nossa macro:

a) No editor Basic, posicione o cursor de texto na linha *sInserir = InputBox(...).* Clique no ícone **Ponto de interrupção**. Surge uma marca vermelha à esquerda da linha.

b) No campo **Observador**, digite **sInserir** e tecle **Enter**, para inspecionar o valor da variável durante a execução da macro. Note que o ícone **Remover observador**, à direita do campo, é ativado.

c) Clique sobre o ícone **Executar**. A execução da macro é iniciada e, no Ponto de interrupção, ela pára – note a seta amarela, à direita.

d) Clicando no ícone **Passo a passo**, acompanhe a execução da macro. Quando a caixa de entrada for exibida, digite o seu nome e clique sobre o botão OK. Continue a execução até o final, quando o ícone Parar for desativado.

Alterne para a janela do documento PrimeiraMacro.sxw, e veja se o seu nome foi inserido na posição do cursor, como desejado.

Se ocorreu algum erro, revise o código fonte.

Caso uma macro seja utilizada com freqüência, podemos configurar o OpenOffice.org para executá-la de modo mais fácil. Na opção de menu **Ferramentas – Configurar**, temos entradas para **Menus** – **Teclado** - **Barra de ferramentas** e **Eventos**, onde é possível associar macros a opções de menus, a ícones na barra de ferramentas, a uma combinação de teclas e a um evento do OpenOffice.org.

Antes de encerrar esta seção, retorne à janela do IDE Basic, para simularmos um erro no código fonte:

a) Na penúltima linha de código – *oTexto.InsertString(...)* - altere o nome da variável *oCursor* para *oCurso*.

b) Execute a macro. Surge uma caixa com uma mensagem de erro e um botão OK. Note que a seta de acompanhamento da execução torna-se vermelha na linha onde ocorre o erro. Após clicar em OK a execução da macro é finalizada e a linha com o erro permanece selecionada.

Figura 3: Erro de execução

c) Corrija o nome da variável de *oCurso* para *oCursor*. Feche o IDE Basic.

Salve o documento Primeira_Macro.sxw.

No próximo passo, veremos como disponibilizar uma macro para todos os documentos do OpenOffice.org.

1.4 Compartilhando uma Macro

As macros de uma biblioteca do container **soffice / Standard** são carregadas juntamente com o OpenOffice.org. Aquelas de um container de um documento qualquer são carregadas juntamente com o mesmo. Portanto, estarão disponiveis para execução somente quando o documento estiver aberto.

Nesta seção, veremos como exportar uma macro de um documento para o aplicativo OpenOffice.org. Aqui, mostraremos apenas os passos necessários para efetuar a tarefa.

Para disponibilizar uma macro de um documento para outros documentos do OpenOffice.org, siga os passos abaixo:

a) Selecione **Ferramentas – Macro** e clique sobre o botão **Gerenciar**.

b) Aparece o diálogo Gerenciar, clique sobre a aba **Bibliotecas**.

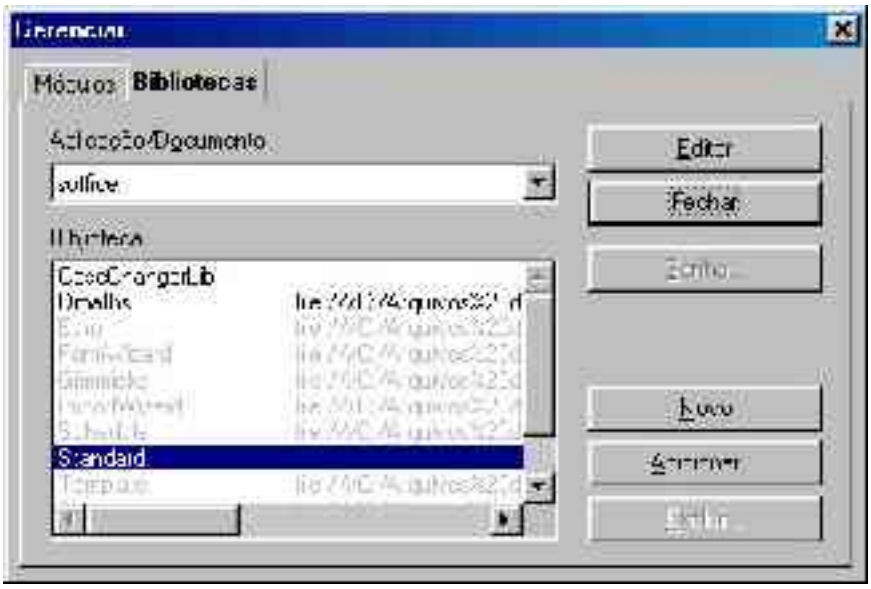


Figura 4: Diálogo Gerenciar

c) Selecione, na caixa de listagem **Aplicação / Documento**, a entrada **soffice** e pressione o botão **Adicionar**.

d) Um diálogo **Abrir Arquivo** será exibido. Localize e selecione o arquivo que contém a macro a ser compartilhada e clique sobre o botão **Abrir**.

e) O diálogo **Inserir Bibliotecas** será exibido. Note que as bibliotecas existentes no arquivo estão marcadas para compartilhamento. Desmarque aquelas que não serão compartilhadas e clique sobre o botão **Ok**.

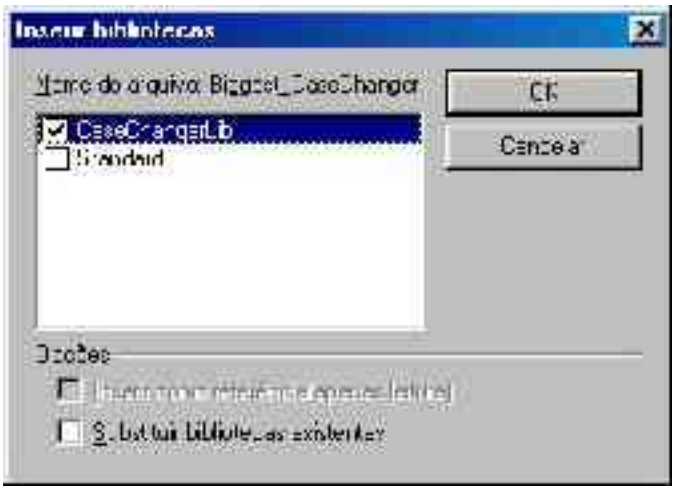


Figura 5: Inserir Bibliotecas

Pronto, agora a sua macro faz parte do repositório soffice e está disponível para execução no OpenOffice.org.

2 Diálogos Personalizados

2.1 Introdução

O OpenOffice.org Basic permite a criação de Diálogos Personalizados, através de um editor de Diálogos intuitivo e poderoso.

Diálogos são uma excelente solução para interação com o usuário, permitindo que o programador desenhe uma interface gráfica consistente para obter ou passar informações ao operador.

2.2 Criando um Diálogo

O Diálogo que vamos criar será bem simples. Na verdade, substituiremos a caixa de entrada usada em nossa primeira macro por um diálogo personalizado.

Antes de começarmos, precisamos recuperar o nosso arquivo **Primeira_Macro.sxw**:

a) Carregue o OpenOffice.org.

b) Abra o arquivo Primeira_Macro.sxw.

c) Escolha **Ferramentas – Macro**, selecione o **Module1** da nossa macro e clique sobre o botão **Editar** para ativar o IDE Basic.

d) Clique com o botão direito do mouse sobre a aba **Module1**. No menu instatâneo, selecione **Inserir – Diálogo**. Aparece o editor de diálogos Basic, com um diálogo vazio. Note que foi criada uma página **Dialog1**.

e) Clique sobre o ícone **Controles**  . Surge um quadro com vários ícones. Arraste este quadro pela barra de título (faixa superior) para fora da barra de ferramentas, posicionando-o do lado direito do diálogo.

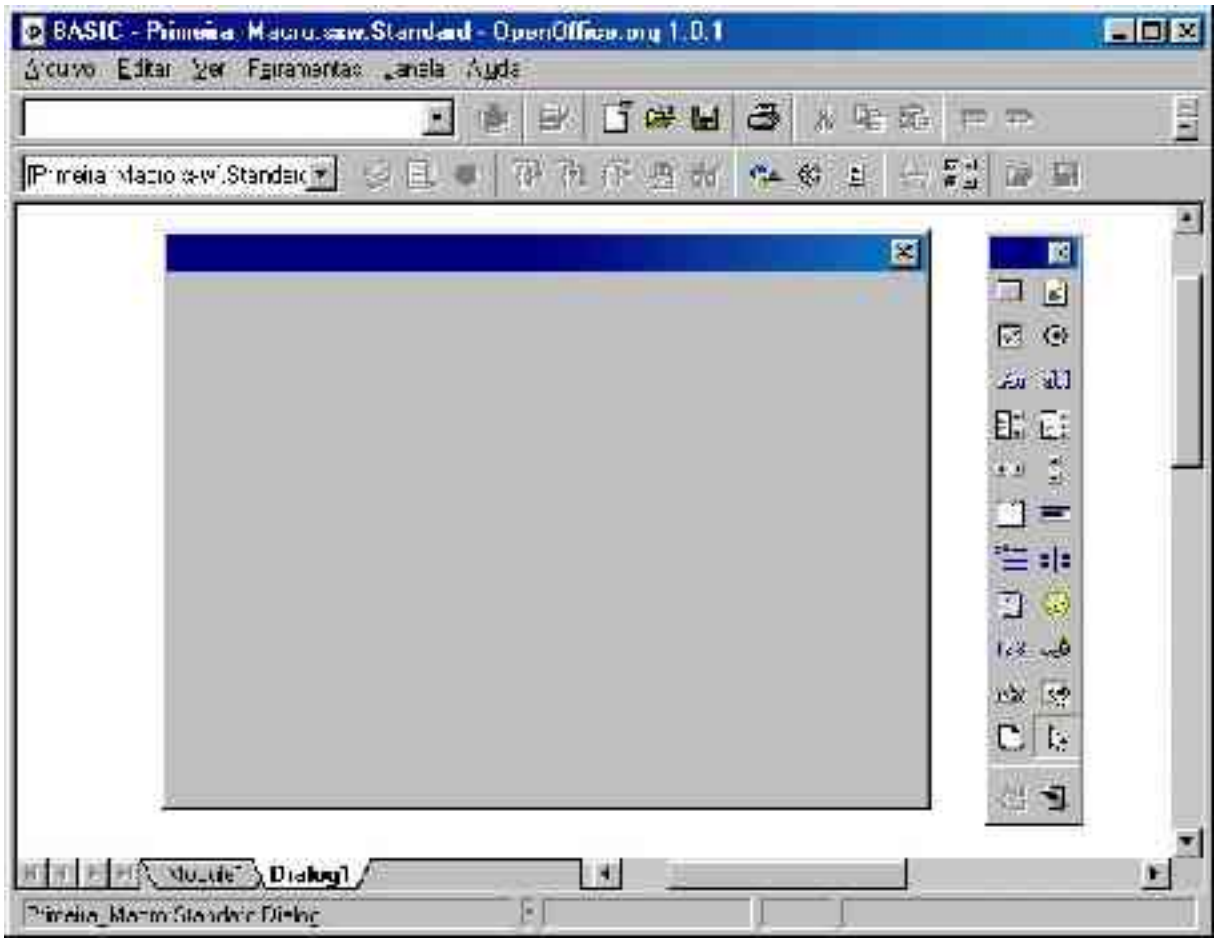


Figura 6: Editor de Diálogos

Vejamos a finalidade de alguns ícones existentes no **quadro de Controles**.

Etiqueta – usado para colocar etiquetas no diálogo.

Botão – usado para colocar botões no diálogo.

Caixa de texto – permite inserir uma caixa de texto num diálogo.

Propriedades – exibe a janela de propriedades do controle selecionado. Será ativado quando algum controle for selecionado. Além da edição das propriedades, permite associar procedimentos (código) a eventos ocorridos no controle.

Ativar modo de teste – exibe o diálogo como se estivesse em execução.

Inserir Controles num Diálogo é uma operação em três passos:

1) ativar o ícone do controle no quadro de controles;
2) definir a área do diálogo onde o controle será posicionado. Para isto pressione o botão esquerdo do mouse e arraste formando um retângulo, em seguida, libere o botão do mouse;
3) definir as propriedades do controle.

Quando um controle é adicionado ao diálogo, o OpenOffice.org Basic automaticamente define um valor padrão para cada propriedade. Para exibir as propriedades de um controle selecionado, clique sobre o ícone **Propriedades** ou clique duas vezes sobre o controle. Note, também, que a janela Propriedades possui o botão **Subir – Descer**.

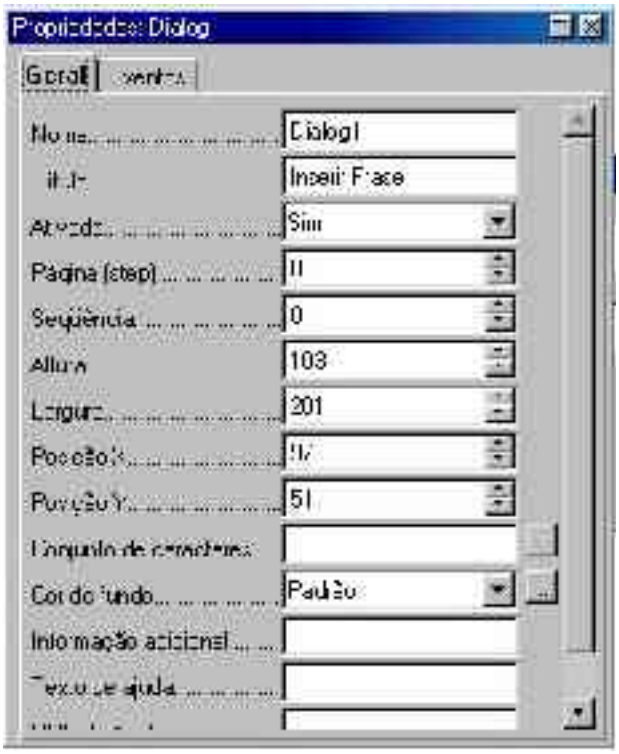


Figura 7: Janela Propriedades

Vamos trabalhar em nosso diálogo:

a) Selecione o diálogo vazio, clicando na sua **borda externa**. Note que aparecem marcas em torno do mesmo. Podemos usar estas marcas para redimensionar o nosso diálogo.

b) Clique no ícone **Propriedades** para exibir a sua janela, como na figura acima. No campo **Título** – aba Geral – digite *Inserir Frase* e tecle Enter. Clique no ícone **Subir** de Propriedades e observe o título do diálogo.

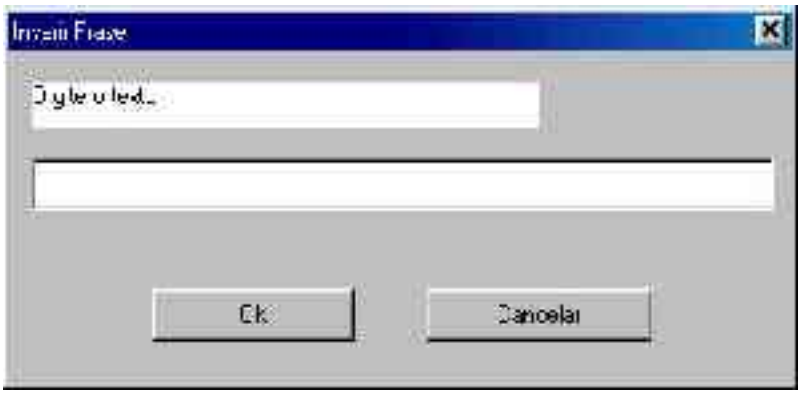
c) Adicione um controle **Etiqueta** ao diálogo. Para isto, clique no ícone **Etiqueta**, em seguida defina um retângulo na parte superior do diálogo. Defina a sua propriedade título para *Digite o texto.*

d) Coloque uma **Caixa de Texto** no diálogo, logo abaixo da etiqueta. Observe a sua propriedade Nome: *TextField1*. No código fonte, nos referimos a um controle pela sua propriedade **Nome**, daí a sua importância. Note, também, que este controle não tem a propriedade Título.

e) Vamos colocar dois botões, lado a lado, na parte inferior do diálogo, abaixo da caixa de texto. Defina os seus títulos para *OK* e *Cancelar.*

f) Ajuste o diálogo e seus controles de modo que fiquem com a aparência da figura abaixo.

Figura 8: Diálogo Inserir Frase



Bem, terminamos o nosso diálogo. Vamos verificar a sua aparência quando em execução. Clique sobre o ícone **Ativar modo de teste** no quadro de controles e, após observar o diálogo, clique em seu ícone **Fechar** para retornar ao editor.

Na próxima seção, daremos funcionalidade ao nosso Diálogo.

2.3 Manipulando Eventos do Diálogo

Vejamos como o diálogo que acabamos de projetar vai interagir com o operador: após a exibição do diálogo, o operador deve digitar o texto e depois pressionar o botão Ok ou pode, simplesmente, pressionar o botão Cancelar. Para cada uma destas ações (e muitas outras), o sistema emite mensagens de eventos.

Para dar funcionalidade ao diálogo, precisamos interceptar os eventos que nos interessam, isto é, o pressionamento do botão Ok ou Cancelar e, em seguida, fazer o processamento requerido.

O IDE Basic permite a associação de **procedimentos** a eventos de controles através da janela **Propriedades** e sua aba **Eventos**. A palavra procedimentos, acima, indica que precisamos de código fonte.

No IDE Basic, selecione o módulo **Module1**.

No editor de código Basic, digite (ou copie e cole) o código fonte abaixo, **acima** da Sub Main.

```
Private oDialogo as Variant

Sub dlgExecutaDialogo
   oDialogo = createUnoDialog(DialogLibraries.Standard.Dialog1)
   oDialogo.execute()
End Sub

Sub dlgCancelaDialogo
   oDialogo.endExecute()
End Sub

Sub dlgInserirFrase
   Dim oDocumento as Object
   Dim oTexto as Object
   Dim oCursorVista as Object
   Dim oCursor as Object
   Dim sFrase as String
```

```
   oDialogo.endExecute()
   ' note a referencia ao controle TextField1 do dialogo
   sFrase =  oDialogo.Model.TextField1.Text

   if sFrase = "" then
      exit sub
   endif
   ' obtém o modelo do documento
   oDocumento = ThisComponent
   ' obtém o serviço Text do documento
   oTexto = oDocumento.getText()
   ' obtém a posição do cursor na GUI e cria um cursor de texto
   oCursorVista = oDocumento.getCurrentController().getViewCursor()
   oCursor = oTexto.createTextCursorByRange(oCursorVista.getStart())
   ' insere o texto na posição corrente do cursor
   oTexto.insertString(oCursor, sFrase, FALSE)

End Sub
```

No código acima, declaramos a variável oDialogo (Private oDialogo as Variant) fora dos procedimentos, o que a torna visível em todo o módulo **Module1**.

Antes de exibir o diálogo, criamos um objeto *oDialogo* com *createUnoDialog()* e, para executá-lo, chamamos o seu método *execute*. Para fechar o diálogo chamamos o seu método *endExecute* (é chamado em dois procedimentos). Objetos UNO (criados com *createUno*) devem ser declarados com o **tipo Variant** e não Object.

A seguir, vamos associar os procedimentos aos eventos do diálogo.

Selecione a página **Dialog1** para ativar o editor de diálogos.

Em nosso diálogo, selecione o botão **OK** e clique no ícone **Propriedades**. Surge a janela Propriedades, clique na aba **Eventos** e, em seguida, clique no botão  à direita do campo do evento **Ao Iniciar**. Surge a janela **Atribuir macros** com o evento **Ao Iniciar** selecionado. Na árvore **Macros**, expanda as entradas **Primeira_Macro.sxw** e **Standard**, clique sobre **Module1**. Surge a relação de procedimentos do módulo, selecione o procedimento **dlgInserirFrase** e clique sobre o botão **Atribuir**. Para fechar a janela, clique no botão **OK**.

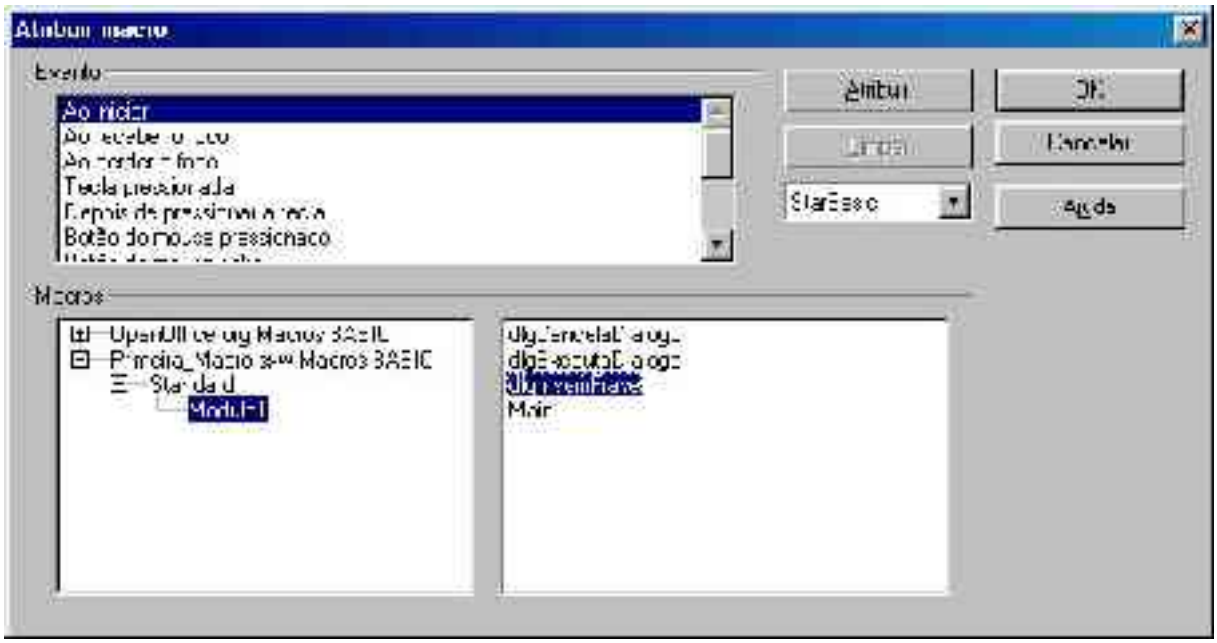


Figura 9: Diálogo Atribuir macro

Agora, repita os passos acima para atribuir o procedimento **dlgCancelaDialogo** ao evento **Ao Iniciar,** do botão **Cancelar** do nosso diálogo.

Para o OpenOffice.org Basic, o primeiro procedimento no módulo é o ponto de entrada da macro. Logo, a **Sub dlgExecutaDialogo** será automaticamente executada toda vez que executarmos a nossa macro.

Teste a macro e o diálogo personalizado, clicando sobre o ícone **Executar**, na barra de ferramentas do IDE Basic. Digite uma frase qualquer na caixa de texto do diálogo e clique sobre o botão **Ok**. Após a execução da macro, verifique se a frase foi adicionada corretamente ao documento.

No próximo capítulo, vamos transformar o nosso diálogo simples num Piloto Automático.

3 Pilotos Automáticos

3.1 Introdução

Um Piloto Automático é um Diálogo exibido em vários passos. É uma forma eficiente de guiar o usuário na execução de uma tarefa.

O OpenOffice.org traz, em sua instalação padrão, vários Pilotos Automáticos. Verifique o funcionamento de um deles, selecionando **Arquivo – Piloto Automático – Carta**.

3.2 Criando um Piloto Automático

Nesta seção, veremos como transformar o Diálogo Personalizado do capítulo anterior, num Piloto Automático.

A idéia geral é, num primeiro passo, obter a frase e, num segundo passo, solicitar uma posição para inserir a frase digitada. Para dar este efeito passo a passo, o OpenOffice.org permite a criação de diálogos com múltiplas páginas. O diálogo com a propriedade **Página** igual a 1 será exibido no primeiro passo, o da página 2 no segundo, e assim sucessivamente.

Os controles do OpenOffice.org Basic têm uma propriedade **Página (Step)**, que controla em qual página de diálogo ele será exibido. Assim, os controles com a propriedade Página igual a 1 (um) serão visíveis no diálogo da página 1 (um), aqueles com a Página igual a 2 (dois) serão visíveis no diálogo da página 2 (dois) e assim por diante. Um controle será exibido em todos os passos quando a sua propriedade Página for igual a 0 (zero).

O último diálogo acessado no editor de diálogos, independente do número da sua página, será o primeiro a ser exibido na execução da macro. Portanto, após desenhar os diálogos do Piloto Automático, retorne para o diálogo da página 1 (um). Se você definir a página do diálogo para 0 (zero), todos os controles, de todas as páginas, serão exibidos.

Na transformação do diálogo em Piloto Automático, precisaremos de mais alguns controles e, também, alterar algumas propriedades dos controles existentes.

Carregue o **Writer**, abra o documento **Primera_Macro.sxw**, selecione **Ferramentas – Macro**, navegue até o modulo **Primera_Macro - Standard – Module1** e pressione o botão **Editar**, para ativar o IDE Basic.

Clique sobre a aba **Dialog1**, ativando o Editor de Diálogos e redimensione o diálogo tornando-o mais largo. Selecione os botões Ok e Cancelar (clique em Ok, pressione a tecla Shift e clique em Cancelar), mova horizontalmente estes botões para a direita.

Na barra de ferramentas do IDE basic, clique no ícone **Controles** e posicione o quadro de controles ao lado do diálogo.

Clique no ícone **Botão**, no quadro de controles e adicione dois novos botões no diálogo, lado a lado e à esquerda do botão Ok.

Selecione o botão *CommandButton3*, clique no ícone **Propriedades** e defina a propriedade **Titulo** para **<< Voltar**, altere o valor da propriedade **Ativado** de Sim para **Não**.

Selecione o botão *CommandButton4* e defina a sua propriedade **Titulo** para **Próximo >>**.

Selecione o botão *Ok* e altere o seu **Título** para **Concluir**.

Selecione o diálogo (clique na sua borda externa) e defina a sua propriedade **Página (Step)** para **1** (um).

Selecione o controle *Etiqueta* e defina a sua propriedade **Página** para **1** (um).

Selecione o controle *Caixa de Texto* e defina a sua propriedade **Página** para **1** (um).

Agora, retoque a primeira página do Piloto Automático, tornando-a semelhante à figura abaixo.

Figura 10: Primeira Página do Diálogo do Piloto Automático

Bem, vamos prosseguir em nosso projeto, desenhando a Página 2 do Piloto Automático.

Selecione o diálogo e altere a propriedade **Página** para **2** (dois). Note que os controles Etiqueta e Caixa de Texto desaparecem, pois eles pertencem à Página 1. Os botões, que têm a página igual a 0 (zero), continuam visíveis.

Adicione uma **Etiqueta** na parte superior esquerda do diálogo e defina o seu **Título** para **Selecione a posição**. Observe que a propriedade página é igual a 2 (dois), o mesmo valor do diálogo.

Clique sobre o ícone Botão de Opção, no quadro de controles, e ponha três botões de opção, na parte superior direita do diálogo, um abaixo do outro.

Defina o **Título** do primeiro botão de opção para **no Cursor** e a sua propriedade **Estado** para **Selecionado**. Defina o **Título** do segundo botão de opção para **no Início**. Defina o **Título** do terceiro botão de opção para **no Final**.

Agora, ajuste os controles da segunda página do Piloto Automático, tornando-a semelhante à figura abaixo. Note que o botão de opção **no Cursor** é o padrão. A ativação e desativação dos botões **Voltar** e **Próximo** será feita em tempo de execução (pelo código fonte).

Figura 11: Segunda Página do Diálogo do Piloto Automático.

Para finalizar o desenho do Piloto Automático, selecione o diálogo e retorne para a Página 1 (um), pois o OpenOffice.org, como padrão, exibe a última página acessada no Editor de Diálogos.

Na próxima seção, veremos o código fonte necessário para dar funcionalidade ao Piloto Automático.

3.3 Manipulando Eventos do Piloto Automático

O nosso Piloto Automático tem dois novos botões (Voltar e Próximo) e precisamos de código fonte para interceptar e processar os seus eventos.

Ative o IDE Basic e seu editor e digite (ou copie e cole) o código abaixo **acima** da **Sub Main**.

```
Sub dlgProximoDialogo
   oDialogo.Model.Step = 2
   oDialogo.Model.CommandButton3.Enabled = true
   oDialogo.Model.CommandButton4.Enabled = false
End Sub

Sub dlgVoltarDialogo
   oDialogo.Model.Step = 1
   oDialogo.Model.CommandButton3.Enabled = false
   oDialogo.Model.CommandButton4.Enabled = true
End Sub
```

Antes de inserir a frase, precisamos verificar a posição, selecionada no segundo passo (botões de opção) e tratar essa escolha adequadamente.

Digite o código abaixo na **Sub dlgInserirFrase**, após a linha *oCursor = oTexto.Create-TextCursorByRange(...)* e antes da linha *oTexto.insertString(...).*

```
   ' a posição da frase é no início ou no final
   if oDialogo.Model.OptionButton2.State = 1 then
      oCursor.gotoStart(FALSE)
   elseif oDialogo.Model.OptionButton3.State = 1 then
```

```
      oCursor.gotoEnd(FALSE)
   endif
```

Agora, vamos ligar os procedimentos aos eventos que nos interessam.

Alterne para o editor de diálogos.

Selecione o botão **Voltar** e clique no ícone **Propriedades**. Surge a janela Propriedades, clique na aba **Eventos** e, em seguida, clique no botão  à direita do campo do evento **Ao Iniciar**. Surge a janela **Atribuir macros** com o evento **Ao Iniciar** selecionado. Na árvore **Macros**, expanda as entradas **Primeira_Macro.sxw** e **Standard**, clique sobre **Module1**. Surge a relação de procedimentos do módulo, selecione o procedimento **dlgVoltarDialogo** e clique sobre o botão **Atribuir**. Para fechar a janela, clique no botão **OK**.

Seguindo os passos acima, atribua o procedimento **dlgProximoDialogo** ao evento **Ao Iniciar** do botão **Próximo**.

Execute a macro e verifique se tudo ocorreu como o esperado.

Agora, você pode se aventurar escrevendo suas próprias macros. Como um ponto de partida, consulte o próximo capítulo em busca de informações complementares.

4 Introdução a API do OpenOffice.org

4.1 Introdução

A chave para a criação de programas que usam a API do OpenOffice.org são os serviços.

Um **serviço** é uma especificação de um objeto, que engloba um conjunto de interfaces e propriedades. Uma **interface** é uma coleção de métodos. Uma **propriedade** é um valor que determina uma característica de um serviço e é formada por um **nome** e um **valor**.

Para criar um serviço, usamos a função Basic **createUnoService()**, que retorna um objeto com suporte ao serviço ou Null, se não for possível a criação. Por exemplo, para obter o serviço Desktop usamos a chamada:

```
oDesktop = createUnoService( “com.sun.star.frame.Desktop” )
```

O parâmetro da função é a especificação completa para o serviço. Todos os serviços iniciam com **com.sun.star**, em seguida vem o nome do módulo, neste caso, **frame** e, por fim, o nome do serviço **Desktop**. Note também que a especificação é passada como uma cadeia **“com.sun.star.frame.Desktop”**.

Após a chamada, podemos verificar se o serviço foi criado com:

```
If isNull( oDesktop ) Then
    ' Erro na criação do serviço
End If
```

Existem serviços que são dependentes de um contexto. Por exemplo, você não pode criar uma célula fora de uma planilha. Outros serviços não precisam de um ambiente para operar. Existem, ainda, serviços que não oferecem nenhuma interface, servindo unicamente para obter e definir propriedades.

O Basic oferece duas propriedades que facilitam o acesso a serviços, são elas:

StarDesktop – é equivalente ao objeto retornado por uma chamada à função createUnoService(“com.sun.star.frame.Desktop”). Digite o código abaixo numa nova sub-rotina, execute e observe o resultado:

```
MsgBox StarDesktop.Dbg_SupportedInterfaces
'
' é o mesmo que:
'
Dim oDesktop
oDesktop = CreateUnoService( "com.sun.star.frame.Desktop" )
MsgBox oDesktop.Dbg_SupportedInterfaces
```

ThisComponent – retorna o objeto documento que contém o código Basic, existe apenas em documentos do Writer, Calc, Impress ou Draw. Algumas de suas interfaces dependem do tipo de documento.

```
Dim oDocumento As Object
oDocumento = ThisComponent
```

O OOo Basic tem três propriedades muito úteis para a inspeção de objetos, são elas:

Dbg_SupportedInterfaces
retorna as interfaces suportadas pelo objeto

Dbg_Properties
retorna as propriedades do objeto

Dbg_Methods
retorna os métodos suportados pelo objeto

Para obter estas propriedades, use a forma **Objeto.Propriedade**. Por exemplo, verifique as interfaces suportadas pelo modelo de documento, executando o código abaixo:

```
Sub Main
    MsgBox ThisComponent.Dbg_SupportedInterfaces
End Sub
```

A interface com.sun.star.lang.XServiceInfo possui os métodos abaixo, úteis para inspeção de serviços:

getImplementationName ( ) As String
retorna o nome de implementação do serviço

supportsService (sNome As String) As Boolean
retorna True se o serviço sNome for suportado

supportedServiceNames ( ) As String ()
retorna um conjunto de cadeias com os nomes dos serviços suportados, inclusive os indiretos

Para obter ou definir valores de **propriedades**, temos dois casos:

a) se o número de propriedades for fixo, então usamos atribuições simples do Basic, como:

```
Dim nCorAntiga As Long
nCorAntiga  = oRectangleShape.FillColor
oRectangleShape.FillColor = RGB(255,0,0)
```

b) se for uma sequência, devemos definir um vetor antes de manipular as propriedades:

```
Dim mPropArquivo(1) As New com.sun.star.beans.PropertyValue
mPropArquivo(0).Name  = "FilterName"
mPropArquivo(0).Value = "swriter: StarWriter 5.0"
mPropArquivo(1).Name  = "AsTemplate"
mPropArquivo(1).Value = True
```

A sequência **com.sun.star.beans.PropertyValue** implementa o serviço **MediaDescriptor**, que descreve de onde (ou aonde) e como um recurso deve ser carregado ou gravado. Observe que as propriedades são definidas por um par **Name** e **Value**.

A seguir, vamos analisar um exemplo de código que usa alguns destes conceitos.

4.2 Criando um novo documento

Vamos, agora, desenvolver uma sub-rotina para criar um novo documento a partir de um modelo existente. O código fonte Basic, em linhas gerais, precisa tratar dos seguintes detalhes:

a) obter o caminho completo para o arquivo modelo;

b) carregar o arquivo, informando ao OpenOffice.org que se trata de um modelo;

A sub-rotina *Sub criaNovoDocumento*, abaixo, executa estas tarefas apropriadamente.

```
' ----------------------------------------
' Cria um novo documento a partir de um modelo
' ----------------------------------------
Sub criaNovoDocumento
   Dim oDesktop As Variant
   Dim oDocumento As Object
   Dim mPropArquivo(0) As New com.sun.star.beans.PropertyValue
   Dim sUrl As String

   ' cria o objeto oDesktop
   oDesktop = createUnoService("com.sun.star.frame.Desktop")
   ' define a URL do arquivo modelo, ALTERE para seu sistema
   sUrl = "file:///D:/nad/openoffice/documentation.stw"
   ' define a propriedade AsTemplate para True
   mPropArquivo(0).Name = "AsTemplate"
   mPropArquivo(0).Value = True
   ' cria o objeto oDocumento
   oDocumento = oDesktop.loadComponentFromURL(sUrl,"_blank",0,mPropArquivo())
   ' INSERIR CÓDIGO PARA SALVAR AQUI
End Sub
```

As linhas iniciadas pelo comando **Dim** declaram todas as variáveis usadas pela sub-rotina.

Para carregar um arquivo, usamos o método *loadComponentFromURL()*, que é definido por uma interface do serviço **Desktop**. Então, precisamos de um objeto UNO Desktop, declarado na linha *Dim oDesktop As Variant*. Aqui, o tipo Variant segue uma recomendação do Developers Guide para a declaração de objetos UNO.

A variável **sUrl** (tipo String) recebe a especificação completa do caminho para o arquivo. Será passada como parâmetro do método *loadComponentFromURL()*.

O método *loadComponentFromURL()* retorna um objeto Document. A linha *Dim oDocumento As Object* reserva memória para este objeto.

O método *loadComponentFromURL()*, recebe uma sequência de propriedades como parâmetro. A linha *Dim mPropArquivo* declara e define um vetor para esta sequência. A definição

do objeto é feita com o operador **New** seguido do tipo de objeto (*As New com.sun.star.beans. PropertyValue*).

A linha *oDesktop = createUnoService(...)* se encarrega da criação do objeto Desktop.

A linha *oDocumento = oDesktop.loadComponentFromURL(...)* carrega o documento de acordo com os parâmetros e retorna um objeto do tipo Document. Note que a sequência mPropArquivo é passada como argumento seguida de (). O parâmetro "**_blank**" significa que um novo quadro será criado para o documento.

Pronto, agora digite (ou copie e cole) a **Sub criaNovoDocumento**, alterando a variável sUrl para conter um caminho completo do seu sistema, e execute a macro para ver seu resultado.

No próximo capítulo, aprenderemos mais sobre programação com documentos do OpenOffice.org.

5 Trabalhando com Documentos

5.1 Carregando Documentos

Para carregar ou criar um documento do OpenOffice.org devemos:

a) **Criar um objeto Desktop**

```
Dim oDesktop As Variant
oDesktop = createUnoService("com.sun.star.frame.Desktop")
```

b) **Definir a URL**

Você deve fornecer uma cadeia de caracteres com a especificação completa do caminho do documento, caso esteja abrindo um documento existente, na forma:

file:///caminho_completo_do_arquivo

Para novos documentos, criados a partir do modelo padrão, existem URLs pré-definidas, que devem ser usadas de acordo com o tipo de documento:

URL	Descrição
private:factory/swriter	Cria um novo documento do Writer
private:factory/scalc	Cria um novo documento do Calc
private:factory/sdraw	Cria um novo documento do Draw
private:factory/simpress	Cria um novo documento do Impress

c) **Definir as propriedades do descritor de mídia**

O descritor de mídia é um serviço (*com.sun.star.document.MediaDescriptor*) que define como um recurso deve ser carregado. As propriedades do descritor são definidas numa sequência **com.sun.star.beans.PropertyValue**. Declare um vetor para a sequência, de acordo com o número de propriedades a alterar, como abaixo:

```
' declara um vetor vazio, para efeito de passagem como parâmetro
Dim mPadrao()
' define um vetor para 1 propriedade (índice 0)
Dim mPropriedades(0) As New com.sun.star.beans.PropertyValue
mPropArquivo(0).Name = "AsTemplate"
mPropArquivo(0).Value = True
' define um vetor para 2 propriedades (índices 0 e 1)
Dim mPropriedades(1) As New com.sun.star.beans.PropertyValue
```

Seguem as descrições de algumas propriedades (existem várias):

Propriedade	Tipo	Descrição
AsTemplate	boolean	Se **True** cria um novo documento, a partir do documento carregado, mesmo que ele não seja um modelo. Se for um modelo e AsTemplate for **False** o documento será carregado para edição.
FilterName	String	Define um filtro a ser usado para carregar ou salvar o documento.
JumpMark	String	Após a carga, salta para uma posição (célula, bookmark, etc).
Overwrite	boolean	Sobrescreve um documento ao salvar. Padrão é True.
ReadOnly	boolean	Se o documento deve ser somente leitura ou leitura / gravação.
Unpacked	boolean	Na gravação o documento não será compactado (zip).
Password	String	Senha para gravar ou carregar o documento.

d) **Definir o quadro de destino**

É o "frame" onde o documento será carregado / criado, se já existir um quadro com o nome especificado, ele será usado, senão um novo será criado.

Os nomes pré-definidos, a seguir, nunca devem ser usados como nome de quadros: "_blank"; "_default"; "_self"; "_parent"; "_top" e "_beamer". Use **"_blank"** para criar um novo quadro.

e) **Definir o flag de busca**

É uma constante que define o tipo de algoritmo usado para encontrar o quadro de destino. Use um valor 0 para desconsiderar.

f) **Chamar o método loadComponentFromURL()**

Este método, do objeto Desktop, carrega o componente definido pela URL, dentro do quadro de destino. Vejamos seus detalhes:

```
loadComponentFromURL( URL As String,
                      FrameDestino As String,
                      FlagBusca As Long,
                      Propriedades As <com.sun.star.beans.PropertyValue> )
```

O valor de retorno é um objeto XComponent ou Null em caso de erro.

Eis um exemplo de chamada:

```
oDocumento = oDesktop.loadComponentFromURL(sUrl, "_blank", _
             0, mPropArquivo())
```

Isto é tudo, analise novamente o código da **Sub criaNovoDocumento**, do capítulo anterior.

5.2 Salvando Documentos

Para salvar um documento, devemos:

(as letras a), b) e c), abaixo, foram vistas no item 5.1 Carregando documentos)

a) **Criar um objeto Document**

Nomalmente, por uma chamada a loadComponentFromURL() ou ThisComponent.

b) **Definir a URL**

Trata-se do caminho completo do recurso.

c) **Definir as propriedades**

São as propriedades do descritor de mídia.

d) **Chamar o método adequado**

Os documentos do OpenOffice.org suportam os métodos abaixo para salvar componentes:

```
store ( )
storeAsURL ( sURL As String, Propriedades As <com.sun.star.beans.PropertyValue> )
storeToURL ( sURL As String, Propriedades As <com.sun.star.beans.PropertyValue> )
```

O método store ( ) simplesmente sobrescreve um arquivo e não deve ser usado num novo documento sem nome.

O método storeAsURL ( ) recebe dois parâmetros e funciona como o comando **Salvar Como** do OpenOffice.org.

O método storeToURL ( ) recebe dois parâmetros, salva o conteúdo do documento em sURL e não altera o arquivo original. Útil para exportar arquivos.

Exemplos de chamadas:

```
' salva um documento que já existe
ThisComponent.store ( )
' ou, supondo que oDocumento foi definido
oDocumento.store ( )
```

Documentos abertos com loadComponentFromURL ( ), devem usar os métodos storeAsURL ou storeToURL.

Como exemplo, digite (ou copie e cole) o código fonte abaixo, após a linha

```
' INSERIR CÓDIGO PARA SALVAR AQUI
```

da **Sub criaNovoDocumento ( )**, edite a variável **sUrl** conforme o seu sistema, execute a macro e verifique que um novo arquivo foi gravado em sURL..

```
' --------------------
' Grava o novo arquivo
' --------------------
' Define a URL do novo arquivo (ALTERE para seu sistema)
sUrl = "file:///D:/nad/openoffice/novo_texto.sxw"
' define a propriedade Overwrite para False
mPropArquivo(0).Name = "Overwrite"
mPropArquivo(0).Value = FALSE
' grava o novo documento
oDocumento.storeAsURL(sUrl,  mPropArquivo())
```

Temos, ainda, os métodos abaixo, relacionados com a tarefa de salvar arquivos:

```
isModified() As Boolean
```

Retorna True se o recurso foi modificado, senão retorna False.

hasLocation ( ) As Boolean
Retorna True se o arquivo já existe (foi gravado), senão retorna False.

GetLocation ( ) As String
Retorna uma String com a URL do recurso.

isReadonly ( ) As Boolean
Retorna True se for somente leitura, em caso contrário retorna False

A seguir, temos um fragmento de código mais elaborado para salvar arquivos:

```
If (oDocumento.isModified) Then
    If (oDocumento.hasLocation And (Not oDocumento.isReadOnly)) Then
        oDocumento.store()
    Else
        oDocumento.storeAsURL(sURL, mPropArquivo())
    End If
End If
```

5.3 Imprimindo Documentos

O serviço OfficeDocument contém a interface XPrintable, que fornece os métodos para a impressão e definição de impressora no OpenOffice.org, são eles:

getPrinter ( ) As Variant < com::sun::star::beans::PropertyValue >
setPrinter (aImpressora As Variant < com::sun::star::beans::PropertyValue >)
print (xOpcoes As Variant < com::sun::star::beans::PropertyValue >)

Vamos analisar as características e chamadas de cada um destes métodos.

getPrinter () As Variant < com::sun::star::beans::PropertyValue >
Este método retorna uma sequência com o descritor da impressora corrente. O descritor (PrinterDescriptor) contém a fila de impressão e as definições da impressora, conforme a tabela abaixo:

Propriedade	Descrição
Name	Uma cadeia (String) com o nome da fila de impressão
PaperOrientation	Contém a orientação do papel
PaperFormat	Especifica um tamanho de papel padrão ou definido pelo usuário
PaperSize	Especifica o tamanho do papel em 1/100 mm
IsBusy	True se a impressora estiver ocupada; False senão
CanSetPaperOrientation	True se a impressora permite mudar a orientação do papel
CanSetPaperFormat	True se a impressora permite mudar o formato do papel
CanSetPaperSize	True se a impressora permite mudar o tamanho do papel

As quatro últimas propriedades são somente leitura.

Vejamos um exemplo que acessa as propriedades da impressora corrente:

```
Sub obtemDescritorImpressora
    Dim oDoc As Object
    Dim oDescImpr As Variant
    Dim sMsg As String

    oDoc = ThisComponent
    ' obtem o vetor de estruturas PrinterDescriptor
    oDescImpr = oDoc.getPrinter()
    ' obtem tamanho do vetor de estruturas
    iNrEstruturas% = UBound(oDescImpr)
    ' extrai os nomes das propriedades da impressora
    For n = 0 To iNrEstruturas%
       sMsg=sMsg + oDescImpr(n).Name + " "
    Next n
    ' exibe os nomes das propriedades
    MsgBox sMsg,0,"Propriedades Impressora"
    ' obtem e exibe o valor da propriedade PageFormat
    ' A3=0; A4=1; Letter/Carta=5; ...
    MsgBox oDescImpr(2).Value
    ' verifica se PageFormat pode ser alterada
    If oDescImpr(6).Value Then
       MsgBox "Formato da Página, pode ser alterado!"
       ' então altera para A4
       oDescImpr(2).Value = 1
    Else
       MsgBox "Formato da Página, não pode ser alterado!"
    End If
    ' exibe o valor de PageFormat novamente
    MsgBox oDescImpr(2).Value
End Sub
```

O código acima chama o método getPrinter ( ) para obter as propriedades da impressora corrente. Em seguida, no laço For ... Next, extrai os nomes de cada propriedade. E, mais abaixo, obtém e altera o formato da página, se possível.

setPrinter (aImpressora As Variant < com::sun::star::beans::PropertyValue >)
Atribui uma nova impressora ao documento. Pode implicar numa reformatação.

As propriedades da impressora são as mesmas da tabela acima.

print (xOpcoes As Variant < com::sun::star::beans::PropertyValue >)
Imprime o documento de acordo com as opções de impressão (PrintOptions) em xOpcoes. Estas opções são descritas na tabela a seguir:

Opção	Descrição
CopyCount	(Int) Especifica o número de cópias a imprimir
FileName	Cadeia com o nome de um arquivo para a impressão
Collate	Se True todo o documento será impresso antes da próxima cópia, senão imprime todas as cópias página a página
Pages	Uma cadeia com as páginas a serem impressas, por exemplo: “1;3;5-8;12”

Vejamos alguns exemplos de código fonte. Para imprimir um documento:

```
' imprimir o documento com as definições correntes
Dim aOpcoes ()
oDocumento.print (aOpcoes() )
```

Para imprimir somente as páginas 1; 3; 5-8 e 12.

```
Dim aOpcoes(0)
aOpcoes(0).Name = “Page”
aOpcoes(0).Value = “1;3;5-8;12”
ThisComponent.print ( aOpcoes() )
```

Para imprimir numa impressora que não a padrão, você precisa alterar a propriedade “Name” da impressora e chamar setPrinter definindo uma nova impressora, como abaixo:

```
Dim mImpressora(0) As New com.sun.star.beans.PropertyValue
Dim aOpcoes() ' somente para passar o parâmetro
'
mImpressora(0).Name="Name"
mImpressora(0).value="Segunda_Impressora"
' define a nova impressora para o objeto,
' pode ocorrer uma formatação do documento
oDocumento.setPrinter = mImpressora()
' imprime com as opcoes padrão
oDocumento.print( aOpcoes() )
```

5.4 Fechando Documentos

Para fechar um documento, devemos considerar dois casos:

a) **Aberto com loadComponentFromURL()**

Vimos que este método retorna um objeto XComponent. A interface deste serviço fornece o método dispose(), que é usado para fechar um componente, sua chamada é simples:

```
oDocumento.dispose()
```

Acrescente a linha acima após a última linha da **Sub criaNovoDocumento()**, execute a macro e observe o resultado.

b) **Aberto pela interface do OpenOffice.org**

Se o documento foi carregado com o comando **Arquivo – Abrir** do OpenOffice.org e não houve a definição de um objeto Document, use a propriedade **ThisComponent**, como abaixo:

```
ThisComponent.dispose()
```

Cuidado, o método dispose() descarta todas as alterações efetuadas no documento. Portanto, antes de chamá-lo, certifique-se de gravá-las. Ou, ainda, use **isModified()** para alertar o usuário.

Existem maneiras mais seguras para controlar o fechamento de componentes, porém estão além do escopo desta Introdução.

5.5 Identificando os Documentos Abertos

Todos os documentos abertos são gerenciados pelo objeto Desktop. Cada documento é considerado um componente deste objeto, contudo nem todo componente é um documento. Por exemplo, se tivermos uma planilha e um documento de texto abertos, com o Navegador de Fonte de Dados ativo, existem três componentes no Desktop, dos quais dois são documentos.

A **Sub exibeComponentes**, abaixo, obtém uma enumeração dos componentes ativos, verifica quais são modelos de documentos e exibe informações sobre o seu tipo.

```
Sub exibeComponentes
   Dim oComponentes As Variant
   Dim oComp As Variant
   Dim sURL As String
   Dim sTipo As String

   oComponentes = StarDesktop.getComponents().createEnumeration()
   n = 0
   Do While (oComponentes.hasMoreElements())
      oComp = oComponentes.nextElement()
      ' nem todos componentes são modelos com uma URL
      If HasUnoInterfaces(oComp, "com.sun.star.frame.XModel") Then
         sURL = oComp.getURL()
         If oComp.SupportsService("com.sun.star.sheet.SpreadsheetDocument") Then
            sTipo = "Calc"
         ElseIf oComp.SupportsService("com.sun.star.text.TextDocument") Then
            sTipo = "Writer"
         ElseIf oComp.SupportsService("com.sun.star.drawing.DrawingDocument") Then
            sTipo = "Draw"
         ElseIf oComp.SupportsService("com.sun.star.formula.FormulaProperties") Then
            sTipo = "Math"
         Else
            sTipo = "Outro"
         End If
         MsgBox sTipo + Chr$(13) + sURL
      End If
      n = n + 1
   Loop
   MsgBox "Componentes: " + Str$(n)
End Sub
```

Analise atentamente o código fonte acima, ele demonstra aspectos importantes da programação da API do OpenOffice.org. Note como se dá a criação de uma enumeração e como visitamos os seus elementos.

Após a identificação de um documento, você pode, por exemplo, comandar a sua impressão ou o seu fechamento, além, é claro, de poder manipular o seu conteúdo, aplicando as técnicas que serão vistas nos próximos capítulos.

6 Documentos do Writer

6.1 Introdução

Neste capítulo, veremos como efetuar tarefas simples de processamento de texto, como: edição, formatação, busca e substituição.

Vamos abordar o assunto através de exemplos práticos, com uma breve explicação dos aspectos mais importantes.

Um documento do Writer contém principalmente texto, organizado em parágrafos e tabelas. Além destes elementos básicos, ele suporta molduras, objetos embutidos, campos, gráficos, marcadores, índices e muitos outros objetos. Estes dados compreendem um modelo de documento.

Através do modelo, podemos manipular os dados independentemente da sua representação visual. Para lidar com o aspecto visual do documento, o modelo dispõe de um objeto controlador.

Alguns elementos de um documento (gráficos, molduras, etc) são nomeados e estão ancorados num parágrafo, caractere ou página. Os parágrafos não recebem um nome e nem um índice, de modo que seu acesso deve ser sequêncial.

6.2 Editando texto

Antes de começar a editar o texto de um documento, devemos ter em mente o seguinte:

a) Criar uma instância do componente a editar, isto é, um objeto que represente o documento do Writer;

b) Obter o conteúdo a editar, neste caso, um objeto com o texto do documento;

c) Fazer a edição do conteúdo, usando os métodos apropriados.

Crie um novo documento do Writer, vá para o IDE Basic, digite o código abaixo e execute a Sub editaTexto para ver o resultado.

```
Sub editaTexto
    Dim oDoc As Object
    Dim oTxt As Object
    Dim oTxtRange As Object
    Dim sStr As String

    oDoc = ThisComponent
    oTxt = oDoc.getText()
    sStr = "Esta é uma "
    oTxt.setString( sStr )
    oTxtRange = oTxt.getEnd()
    sStr = "cadeia de caracteres "
    oTxtRange.setString( sStr )
```

```
End Sub
```

Neste exemplo, usamos a interface XTextRange para editar texto. Eis os seus métodos:

getText() As Text
retorna um objeto Text

getString() As String
retorna a String contida no objeto

setString(sStr As String)
define a String como o conteúdo do objeto

getStart() As Object <TextRange >
retorna um objeto TextRange apontando para o início do objeto chamador

getEnd() As Object <TextRange>
retorna um objeto TextRange apontando para o final do objeto chamador

Inicialmente, chamamos o método getText ( ), que retorna um objeto Text, este passo corresponde ao item b) acima.

Em seguida, na linha *oTxt.setString(sStr)*, definimos uma cadeia de caracteres para o nosso objeto Text, isto significa que todo o conteúdo do documento – objeto oTxt - será substituído pela cadeia sStr de setString. Antes de inserir mais texto, precisamos obter o final do conteúdo de oTxt com a chamada ao método getEnd( ) e usar este novo objeto – TextRange – para inserir mais texto. Através do método getStart, poderíamos inserir texto no início.

Note, ainda, que as variáveis dos serviços Text e TextRange são declaradas como Object.

Atenção, oTxt também é um TextRange, assim podemos substituir a variável oTxtRange pelo encadeamento de métodos abaixo:

```
oTxt.getEnd ().setString ( sStr )
```

Este processo de edição é muito limitado. Podemos apenas substituir todo o conteúdo do texto e inserir texto no início ou no final do documento.

6.3 Movendo-se pelo texto

Para aumentar a flexibilidade na edição de texto, o modelo de documento nos oferece dois tipos de cursor: um cursor de texto e um cursor da vista. O primeiro opera sobre o conteúdo do documento independente da interface gráfica e o segundo representa o cursor visível na interface gráfica. Você pode criar quantos cursores desejar.

O alcance de um cursor tem um início e um final, que pode abranger uma faixa variável de texto ou coincidir num mesmo ponto. O alcance do cursor pode ser expandido durante o deslocamento do mesmo.

No Writer, além do cursor de texto, temos ainda: cursor de palavra, cursor de sentença e cursor de parágrafo, todos derivados do cursor de texto.

Vejamos, agora, as interfaces e métodos usados para flexibilizar a edição de texto.

A interface XSimpleText, derivada de XTextRange, é usada na criação do cursor de texto, na inserção de cadeias e de caracteres de controle, ela possui os seguintes métodos:

createTextCursor ( ) As Object <TextCursor>
Retorna um objeto cursor de texto (no início do conteúdo)

createTextCursorByRange (aPos As TextRange) As Object <TextCursor>
Cria um objeto cursor de texto na posição especificada no parâmetro

insertString(aPos As TextRange, sStr As String, bFlag As Boolean)
Insere a cadeia na posição. Se bFlag for True o conteúdo de aPos será substituído por sStr, senão sStr será acrescentada no final de aPos.

insertControlCharacter(aPos As TextRange, iChar As Integer, bFlag As Boolean)
Insere um caractere de controle na posição aPos.

O serviço TextCursor, através da interface XTextCursor, provê métodos para o controle do estado e movimentação do cursor.

a) Métodos para mover o cursor:

goLeft (iNrChar As Integer, bExpande As Boolean) As Boolean
move o cursor para a esquerda iNrChar caracteres

goRight (iNrChar As Integer, bExpande As Boolean) As Boolean
move o cursor para a direita iNrChar caracteres

gotoStart (bExpande As Boolean)
move o cursor para o início do texto

gotoEnd (bExpande As Boolean)
move o cursor para o final do texto

gotoRange (xRange As <TextRange>, bExpande As Boolean)
move ou expande o cursor sobre o objeto TextRange

Em todos os métodos acima, se bExpande for True o alcance do cursor será expandido.

b) Métodos para controlar o estado do cursor:

collapseToStart ( )
move a posição do final do cursor para o seu início

collapseToEnd ( )
move a posição do início do cursor para o seu final

isCollapsed ( ) As Boolean
retorna True se as posições inicial e final do cursor forem iguais

As interfaces XWordCursor, XSentenceCursor e XparagraphCursor, todas derivadas de XTextCursor, também fornecem métodos para controle e movimento do cursor.

a) Métodos de XWordCursor:

gotoNextWord (bExpande As Boolean) As Boolean
gotoPreviousWord (bExpande As Boolean) As Boolean
gotoEndOfWord (bExpande As Boolean) As Boolean
gotoStartOfWord (bExpande As Boolean) As Boolean
isStartOfWord ( ) As Boolean

```
isEndOfWord ( ) As Boolean
```

b) Métodos de XSentenceCursor:

```
gotoNextSentence (Expande As Boolean) As Boolean
gotoPreviousSentence (bExpande As Boolean) As Boolean
gotoStartOfSentence (bExpande As Boolean) As Boolean
gotoEndOfSentence (bExpande As Boolean) As Boolean
isStartOfSentence ( ) As Boolean
isEndOfSentence ( ) As Boolean
```

c) Métodos de XParagraphCursor:

```
gotoStartOfParagraph (bExpande As Boolean) As Boolean
gotoEndOfParagraph (bExpande As Boolean) As Boolean
gotoNextParagraph (bExpande As Boolean) As Boolean
gotoPreviousParagraph (bExpande As Boolean) As Boolean
isStartOfParagraph ( ) As Boolean
isEndOfParagraph ( ) As Boolean
```

Antes de apresentar um exemplo, vejamos os caracteres de controle que podem ser inseridos num documento do Writer. Na interface gráfica, eles são chamados de caracteres não imprimíveis e têm funções especiais na apresentação final do documento.

Para facilitar o seu emprego, estes caracteres estão definidos como constantes, no grupo de constantes *com.sun.star.text.ControlCharacter*. Eles podem ser inseridos no texto com o método *insertControlCharacter ()* ou como parte do conteúdo de uma cadeia, neste caso, usando o seu código Unicode.

Eis uma relação dos caracteres de controle com os seus respectivos códigos:

PARAGRAPH_BREAK	Insere uma quebra de parágrafo	0x000D
LINE_BREAK	Quebra de linha dentro de um parágrafo	0x000A
HARD_HYPHEN	Caractere que aparece como um travessão. Hifenização não remove.	0x2011
SOFT_HYPHEN	Marca uma posição preferida para hifenização.	0x00AD
HARD_SPACE	Espaço com tamanho fixo.	0x00A0
APPEND_PARAGRAPH	Acrescenta um novo parágrafo	Não tem

Vamos encerrar esta seção com um exemplo demonstrando o emprego de alguns dos conceitos e métodos apresentados.

Ative o IDE Basic, digite (ou copie e cole) o código fonte a seguir e execute a sub-rotina *Sub processaTexto*. Ela insere texto, caracteres de controle e uma quebra de página num novo documento sem nome. Depois, seleciona e substitui duas palavras por uma cadeia vazia.

```
Sub processaTexto
    Dim oDesktop As Variant
    Dim oDoc As Object
    Dim oCursor As Object
```

```
    Dim oTexto As Object
    Dim mPropArquivo()
    Dim sStr As String

    oDesktop = createUnoService("com.sun.star.frame.Desktop")
    oDoc = oDesktop.loadComponentFromURL("private:factory/swriter",_
           "_blank", 0, mPropArquivo())
    oTexto = oDoc.getText()
    oCursor = oTexto.createTextCursor()
    sStr = "Este é o texto do primeiro parágrafo do documento"
    With oTexto
        ' insere o texto na posição corrente do cursor
        .insertString(oCursor, sStr, False)
        ' insere um parágrafo
        .insertControlCharacter(oCursor,_
               com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
        .insertString(oCursor, "Este é o segundo parágrafo ", False)
        .insertControlCharacter(oCursor, _
               com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
    End With
    ' insere uma nova página
    oCursor.gotoStartOfParagraph(True)
    oCursor.breakType = com.sun.star.style.BreakType.PAGE_BEFORE
    oTexto.insertString(oCursor, "Estou numa nova página.", False)
    ' move o cursor sem expansão
    oCursor.gotoStart(False)
    oCursor.gotoNextWord(False)
    oCursor.gotoNextWord(False)
    oCursor.gotoNextWord(False)
    ' move o cursor expandindo, i.é, seleciona “texto do”
    oCursor.gotoNextWord(True)
    oCursor.gotoNextWord(True)
    ' substitui o texto entre o início e o final do cursor
    oTexto.insertString(oCursor, "", True)

End Sub
```

No código acima, estude o emprego do parâmetro **bExpande** (True ou False) ele é muito importante.

Outro aspecto é a inserção da **quebra de página**. Não existe um caractere de controle para quebra de página ou coluna. Ela é uma propriedade do parágrafo. Note que usamos oCursor para definir uma propriedade do parágrafo (*oCursor.breakType*). Isto é possível porque o objeto TextCursor herda características de TextRange.

Os valores possíveis para a propriedade breakType (quebra de página e coluna) estão enumerados em **com.sun.star.style.BreakType**, aqui usamos PAGE_BEFORE.

Finalmente, observe, no novo documento, que o cursor da vista encontra-se no final do mesmo, apesar da última ação (seleção e substituição) ter ocorrido no primeiro parágrafo, o que demonstra a independência entre o cursor de texto e a interface gráfica.

Lembre-se, ainda, que o acesso à interface gráfica se dá através do controlador do modelo do documento. O código fonte da sub-rotina **dlgInserirFrase**, do Capítulo 3 (Piloto Automático), mostra como acessar o controlador e criar um cursor da vista.

6.4 Formatando texto

Esta é outra tarefa comum em processamento de textos. Temos duas abordagens para formatar um conteúdo de texto: a primeira é aplicar as características desejadas diretamente sobre o objeto, sobrescrevendo as definições básicas, por exemplo, selecionar um parágrafo e mudar o seu ajustamento; a segunda consiste em aplicar um estilo pré-definido de formatação, por exemplo selecionar um parágrafo e mudar o seu estilo de Padrão para Título.

Inicialmente, vamos demonstrar a primeira abordagem. Existe uma grande quantidade de propriedades relacionadas à formatação, como propriedades de caracteres e de parágrafos.

O serviço **com.sun.star.style.CharacterProperties** define as propriedades comuns de caracteres. Eis algumas delas:

Propriedade	Descrição
CharFontName	Especifica o nome do estilo da fonte
CharFontStyleName	Contém o nome do estilo da fonte
CharFontFamily	Contém a família da fonte conforme (com.sun.star.awt.FontFamily)
CharHeight	Valor com a altura do caracteres em pontos
CharUnderline	Contém o valor do caractere do sublinhado (com.sun.star.awt.Underline)
CharWeight	Contém o valor do peso da fonte (com.sun.star.awt.FontWeight)
CharPosture	Contém o valor da postura da fonte (com.sun.star.awt.FontSlant)
CharFlash	Se True o caractere pisca. É opcional
CharStrikeout	Define o tipo de risco do caractere (com.sun.star.awt.FontStrikeout)
CharRotation	Define a rotação do caractere em graus

Na tabela acima, as referências ao módulo **com.sun.star.awt**, contém possíveis valores para as respectivas propriedades dos caracteres.

O serviço **com.sun.star.style.ParagraphProperties** define as propriedades dos parágrafos. A tabela abaixo apresenta algumas:

Propriedade	Descrição
ParaAdjust	Define o tipo de ajuste (com.sun.star.style.ParagraphAdjust)
ParaBackColor	Contém a cor de fundo, é opcional
ParaLastLineAdjust	Define o ajuste da última linha
ParaLeftMargin	Define a margem esquerda em 1/100 mm (As Long)
ParaRightMargin	Define a margem direita em 1/100 mm (As Long)
ParaTopMargin	Define a margem superior do parágrafo em 1/100 mm (As Long)
ParaBottomMargin	Define a margem inferior do parágrafo em 1/100 mm (As Long)
ParaStyleName	Contém o nome do estilo de parágrafo corrente, é opcional.

LeftBorder	Define a borda esquerda, é opcional (com.sun.star.table.BorderLine)
RightBorder	Define a borda direita, é opcional (com.sun.star.table.BorderLine)
TopBorder	Define a borda superior, é opcional (com.sun.star.table.BorderLine)
BottomBorder	Define a borda inferior, é opcional (com.sun.star.table.BorderLine)
BorderDistance	Valor com a distância da borda ao objeto (As Long)
ParaFirstLineIndent	Valor longo com a identação da primeira linha, é opcional
ParaVertAlignment	Define o alinhamento vertical (com.sun.star.text.ParagraphVertAlign)

Novamente, as referências **com.sun.star** contém valores possíveis para as respectivas propriedades, geralmente um valor de 0 a N para constantes. Veja a seguir algumas delas.

Constantes de com.sun.star.style.ParagraphAdjust: LEFT, RIGHT, BLOCK, CENTER e STRETCH.

Constantes d e com.sun.star.text.ParagraphVertAlign: AUTOMATIC, BASELINE, TOP, CENTER e BOTTOM.

Existem muitas outras propriedades para caracteres e parágrafos, uma consulta ao Developers Guide e ao Reference Manual, ambos da Sun Microsystems Inc, é indispensável para aqueles que desejam programar para o OpenOffice.org.

Vamos retomar o nosso último exemplo a **Sub processaTexto**. Desejamos mudar a inclinação e cor da fonte da palavra “segundo” e, também, o estilo de um parágrafo de Padrão para Título. Digite o código abaixo, após a última linha da sub-rotina, execute-a e observe o resultado.

```
' ----------------------
' código para formatação
' ----------------------
oCursor.gotoNextParagraph(FALSE)
oCursor.gotoNextWord(False)
oCursor.gotoNextWord(False)
oCursor.gotoNextWord(False)
oCursor.gotoNextWord(True)
oCursor.CharPosture = com.sun.star.awt.FontSlant.REVERSE_ITALIC
oCursor.CharColor = RGB(255,0,0)
' aplica estilo de parágrafo
oCursor.gotoNextParagraph(FALSE)
oCursor.paraStyleName = "Heading"
```

Note a atribuição da propriedade CharPosture a um valor enumerado REVERSE_ITALIC e, ainda, a alteração do estilo de parágrafo paraStyleName para “Heading”. Em se tratando de programação, os nomes permanecem em inglês, diferindo dos nomes apresentados pelo OpenOffice.org no Catálogo de Estilos.

6.5 Formatando com estilo

Vejamos, agora, a segunda abordagem para formatação, que é a aplicação de um estilo sobre o objeto. Sempre que possível, formate os seus documentos usando este método.Isto facilita futuras alterações, pois basta mudarmos as características de um estilo para que todos os objetos com ele formatados sejam alterados.

O módulo **com.sun.star.style** possui diversas interfaces para formatação, definição e acesso aos estilos usados pelo OpenOffice.org.

O Writer disponibiliza diversos tipos de estilos. Temos estilos de parágrafos, estilos de páginas, estilos de caracteres, estilos de numeração e estilos de molduras. Todos eles estão agrupados dentro da Família de Estilos.

```
Família de Estilos
          Estilos de Páginas     ( estilo_1, estilo_2, ..., estilo_n )
          Estilos de Parágrafos  ( estilo_1, estilo_2, ..., estilo_n )
          Estilos de Caracteres  ( estilo_1, estilo_2, ..., estilo_n )
          Estilos de Molduras    ( estilo_1, estilo_2, ..., estilo_n )
          Estilos de Numeração   ( estilo_1, estilo_2, ..., estilo_n )
```

O serviço **com.sun.star.style.StyleFamilies** contém as famílias de estilos de um documento, para obter esta coleção faça:

```
Dim oFamiliasEstilos As Object
oFamiliasEstilos = ThisComponent.getStyleFamilies()
```

Num documento texto, uma chamada ao método **getElementNames ()** retorna um vetor com os nomes das seguintes famílias:

CharacterStyles	- estilos de caracteres
ParagraphStyles	- estilos de parágrafos
FrameStyles	- estilos de molduras
PageStyles	- estilos de páginas
NumberingStyles	- estilos de numeração

O serviço **com.sun.star.style.StyleFamily** contém os estilos de uma mesma família, para obter esta coleção, use o acesso nomeado, isto é, **getByName (“nomeDaFamília”)**:

```
oEstParagrafos = ThisComponent.StyleFamilies.getByName(“ParagraphStyles”)
```

O serviço **com.sun.star.style.Style** define as características de um estilo, para obter um estilo acesse-o pelo seu índice, por exemplo, se o objeto *oEstParagrafos* foi definido, como acima:

```
oEstilo = oEstParagrafos (1)
```

retorna o segundo estilo dentro da coleção de estilos de parágrafos.

A API do OpenOffice.org provê interfaces para facilitar o acesso a elementos de uma dada coleção. Ao trabalhar com estilos precisamos conhecer os métodos:

con.sun.star.container.XNameAcess

```
getByName (sNome As String ) As Variant
getElementNames ( ) As aStrings ( )
hasByName ( sNome As String ) As Boolean
```

con.sun.star.container.XIndexAcess

```
getCount ( ) As Long
getByIndex ( nIndice As Long ) As Variant
```

con.sun.star.container.XNameContainer

```
insertByName ( sNome As String, oElemento As Variant)
removeByName ( sNome As String )
```

O serviço **com.sun.star.style.PageStyle** define as propriedades mais comuns da página, abaixo temos algumas delas:

Propriedade	Descrição
BackColor	Valor Long da cor de fundo da página
LeftMargin	Valor Long da margem esquerda em 1/100 mm
RightMargin	Valor Long da margem direita em 1/100 mm
TopMargin	Valor Long da margem superior em 1/100 mm
BottomMargin	Valor Long da margem inferior em 1/100 mm
IsLandscape	Determina se o formato da página é paisagem (True / False)
PageStyleLayout	Determina o “layout” da página (com.sun.star.style.PageStyleLayout)
Size	Valores Long definindo o tamanho do papel (Size.Width e Size.Height)
Width	Valor Long definindo a largura da página em 1/100 mm
Height	Valor Long definindo a altura da página em 1/100 mm
HeaderLeftMargin	Valor Long definindo a margem esquerda do cabeçalho em 1/100 mm
HeaderRightMargin	Valor Long definindo a margem direita do cabeçalho em 1/100 mm
HeaderIsOn	Determina se o cabeçalho está ativo na página (True / False)
FooterLeftMargin	Valor Long definindo a margem esquerda do rodapé em 1/100 mm
FooterRightMargin	Valor Long definindo a margem direita do rodapé em 1/100 mm
FooterIsOn	Determina se o rodapé está ativo na página (True / False)

Vamos a um exemplo. Queremos exibir o número de estilos de páginas disponíveis num documento, os seus nomes e a propriedade Size do estilo de página Padrão (Standard). Os passos básicos são:

a) obter as familias de estilos do documento;

b) obter os estilos de páginas;

c) obter o vetor com os nomes dos estilos de páginas;

d) usar um índice para extrair cada um dos nomes;

e) obter a página padrão, usando o acesso nomeado.

A sub-rotina **Sub exibeEstilosPagina**, a seguir, implementa o nosso exemplo:

```
Sub exibeEstilosPagina
    Dim oDoc As Object
    Dim oFamiliaEstilos As Object
    Dim oEstilosPagina As Object
    Dim oPaginaPadrao As Object
    Dim mNomesEstilosPagina As Variant
    Dim sEstilo As String
    Dim sMsg As String

    oDoc = ThisComponent
    ' obtém a Família de Estilos (note o uso de oDoc)
    oFamiliaEstilos = oDoc.StyleFamilies
    ' obtém os Estilos de Página
    oEstilosPagina = oFamiliaEstilos.getByName("PageStyles")
    ' exibe quantidade de estilos de página
    MsgBox oEstilosPagina.Count
    ' obtém e exibe os nomes dos estilos de páginas
    mNomesEstilosPagina = oEstilosPagina.getElementNames()
    For n = LBound(mNomesEstilosPagina) To UBound(mNomesEstilosPagina)
        sMsg=sMsg + mNomesEstilosPagina(n) + " "
    Next n
    MsgBox sMsg,0,"Estilos de Páginas"
    ' obtém o Estilo de página Padrão (Standard)
    oPaginaPadrao = oEstilosPagina.getByName("Standard")
    ' testando o tamanho da página
    lLPag = oPaginaPadrao.Size.Width
    lAPag = oPaginaPadrao.Size.Height
    MsgBox Str$(lLPag) + " " + Str$(lAPag)

End Sub
```

A destacar, o acesso pelo índice dentro do laço For ... Next, o acesso nomeado ao estilo da página “Standard” e o acesso à propriedade Size.

Note, ainda, a chamada ao método getStylesFamilies ( ):

```
oFamiliaEstilos = oDoc.StylesFamilies
```

no OpenOffice.org Basic você pode omitir o **get** e o **set** ao programar propriedades.

Este exemplo pode ser usado como modelo para diversas operações com estilos.

Durante a formatação de texto, é comum a criação e aplicação de novos estilos. Esta é uma tarefa simples de executar dentro de uma macro. Para isto devemos seguir os passos:

a) Obter as famílias de estilos do documento;

b) Obter a família do grupo aonde será criado o novo estilo;

c) Criar uma instância do objeto do novo estilo;

d) Acrescentar o novo estilo ao grupo;

e) Definir as propriedades do novo estilo;

f) Aplicar o novo estilo.

Vamor retomar a sub-rotina **Sub processaTexto** e acrescentar código fonte para criar e aplicar um novo estilo de parágrafo. Digite (ou copie e cole) o código abaixo, após a última linha da Sub processaTexto, execute a macro e observe as definições de estilo do primeiro parágrafo do documento.

```
' ----------------------
' código para novo estilo
' ----------------------
Dim oFamiliasEstilos As Object
Dim EstilosParagrafo As Object
Dim oNovoEstilo As Object
' obtem as familias de Estilos
oFamiliasEstilos = oDoc.StyleFamilies
' obtem os estilos de parágrafos
EstilosParagrafo = oFamiliasEstilos.getByName("ParagraphStyles")
' cria um novo estilo de paragrafo
oNovoEstilo = oDoc.createInstance("com.sun.star.style.ParagraphStyle")
' acrescenta o novo estilo com o nome "novo_estilo"
EstilosParagrafo.insertByName("novo_estilo",oNovoEstilo)
' define as propriedades do estilo
oNovoEstilo.Name = "novo_estilo"
oNovoEstilo.CharFontName = "Arial"
oNovoEstilo.CharHeight = 16
oNovoEstilo.ParaAdjust = com.sun.star.style.ParagraphAdjust.CENTER
oNovoEstilo.CharWeight = com.sun.star.awt.FontWeight.BOLD
' entao, aplica o novo estilo ao primeiro paragrafo
oCursor.gotoStart(FALSE)
oCursor.gotoEndOfParagraph(TRUE)
oCursor.paraStyleName = "novo_estilo"
```

Acima, deve-se notar a criação de um novo objeto, no caso uma instância de um estilo de parágrafo, com a chamada ao método **createInstance (...)**. E, ainda, as definições das propriedades do novo estilo.

6.6 Obtendo o objeto selecionado

A API do OpenOffice.org permite que uma macro extraia o conteúdo de uma seleção para, em seguida, aplicar alguma ação sobre o mesmo. A interface **com.sun.star.frame.XModel** provê o método abaixo, que retorna um ou mais objetos com a seleção atual do controlador corrente:

```
getCurrentSelection ( ) As Object
```

O objeto retornado é do tipo *com.sun.star.uno.XInterface* e depende do conteúdo da seleção.

Conteúdo da seleção	Objeto retornado
Texto	Um ou mais objetos apontando para TextRange
Células de tabela	Um objeto apontando para um cursor de tabela
Caixa de texto	Um objeto apontando para uma caixa de texto
Gráfico	Um objeto apontando para um gráfico

Desenho	Um ou mais objetos apontando para ShapeCollection

Para seleção de texto, o método getCurrentSelection ( ) opera da seguinte forma,:

a) se nada estiver selecionado, retorna um objeto TextRange com a posição do cursor da vista;

b) numa seleção simples, retorna um objeto TextRange com a seleção;

c) numa seleção múltipla, retorna objetos TextRange para o cursor da vista e para cada uma das seleções.

No exemplo abaixo, obtemos as cadeias selecionadas, uma ou mais se múltipla seleção, e mudamos a cor de cada uma para vermelha:

```
Sub processaSelecao
   Dim oDoc As Object
   Dim oSel As Object
   Dim oCurTxt As Object

   oDoc = ThisComponent
   oSel = oDoc.getCurrentSelection()
   For i = 0 To oSel.Count-1
      oCurTxt = oSel(i)
      oCurTxt.CharColor = RGB(255,0,0)
   Next i
   MsgBox oSel.Count
End Sub
```

Note o acesso aos objetos selecionados através de um índice e a criação de um cursor de texto antes da mudança de cor.

Para extrair a cadeia selecionada, chame o método getString ( ), como abaixo:

```
sCadeia = oSel ( i ).getString ( )
```

O método **getString ( )** retorna uma cadeia vazia para o primeiro objeto nos casos a) e c) acima. Insira o código abaixo no laço For ... Next de *processaSelecao* e observe a saída:

```
MsgBox Str$ ( i ) + ": " + oSel ( i ).getString ( )
```

Se você quiser aplicar alguma ação na palavra sob o cursor da vista, faça:

```
      If Len(oSel(0).getString())= 0 Then
         oCurTxt = oSel(0).Text.createTextCursorByRange(oSel(0).getStart())
         oCurTxt.gotoStartOfWord(False)
         oCurTxt.gotoEndOfWord(True)
      End If
```

Analise como se dá a criação do cursor de texto. Em seguida, como exercício, faça com que a *Sub processaSeleção* aplique a cor também na palavra sob o cursor da vista.

A API do OpenOffice.org tem, ainda, a interface **com.sun.star.view.XSelectionSupplier**, que implementa o método **getSelection ( )** que retorna o conteúdo selecionado na interface gráfica, para usá-lo acesse o controlador do modelo (veja o código abaixo).

Existem, ainda, dois métodos que podem ser úteis para a identificação do conteúdo selecionado, são eles:

```
getImplementationName ( ) As String
```

retorna o nome da implementação do objeto

getName ( ) As String
retorna o nome do objeto, se o mesmo for nomeado, caso de gráficos e tabelas.

```
Dim oControlador As Object
oControlador = oDoc.getCurrentController()
oSel = oControlador.getSelection()
sNomeImpl = oSel.getImplementationName()
sNome = oSel.getName()
```

Agora, altere a **Sub processaSelecao**, para alertar o usuário quando ele selecionar algo diferente de texto (dica, use getImplementationName).

6.7 Localizando objetos

Um documento do Writer pode conter outros objetos além de texto, como gráficos, tabelas, campos, caixas de texto, etc. Para cada tipo existe uma interface com métodos que retornam um recipiente com os objetos existentes no documento. Eis alguns:

getTextTables ( ) As <com.sun.star.container.XNameAccess >
getTextFrames ( ) As <com.sun.star.container.XNameAccess >
getGraphicObjects ( ) As <com.sun.star.container.XNameAccess >
getTextSections ( ) As <com.sun.star.container.XNameAccess >
getBookMarks ( ) As <com.sun.star.container.XNameAccess >
getTextFields ( ) As <com.sun.star.container.XEnumerationAccess >
getFootNotes ( ) As <com.sun.star.container.XIndexAccess >
getEndNotes ( ) As <com.sun.star.container.XIndexAccess >
getDocumentIndexes ( ) As <com.sun.star.container.XIndexAccess >

A interface **XNameAccess** implementa os seguintes métodos:

getByName (sNomeDoObjeto As String) As Variant
retorna o objeto nomeado

getElementNames ( ) As String ()
retorna uma sequência de cadeias com os nomes dos objetos da coleção

hasByName (sNome As String) As Boolean
retorna True se o objeto de nome sNome existe na coleção

A interface **XIndexAccess** define os métodos a seguir:

getByIndex (iIndice As Long) As Object
retorna o objeto na posiçao iIndice da coleção

getCount ( ) As Long
retorna o número de objetos da coleção

A interface **XEnumerationAccess** define o método:

createEnumeration ( ) As Object <com.sun.star.container.XEnumeration>
retorna um objeto XEnumeration

A interface **XEnumeration** implementa os métodos abaixo:

hasMoreElements ( ) As Boolean
retorna True se ainda existem elementos no recipiente

nextElement ( ) As Variant
retorna o próximo elemento

A localização é simples, basta chamar o método apropriado e a seguir, usar o acesso nomeado ou indexado para obter o objeto.

A sub-rotina **Sub localizaGraficos**, abaixo, exibe os nomes de todos os objetos gráficos do documento:

```
Sub localizaGraficos

   Dim oGraficos As Object
   Dim oGrafico As Object
   Dim n As Integer

   oGraficos = ThisComponent.getGraphicObjects()
   For n = 0 To oGraficos.Count -1
      oGrafico = oGraficos (n)
      MsgBox oGrafico.getName()
   Next n
   MsgBox "Gráficos no documento: " + Str$(oGraficos.Count)

End Sub
```

Para apagar um gráfico do seu documento, você pode usar o bloco de código:

```
oGrafico = oGraficos.getByName (“Gráfico2”)
oGrafico.dispose ()
```

No exemplo a seguir, que demonstra a criação de uma enumeração, exibimos o comando de cada um dos campos de um documento:

```
Sub localizaCampos

   Dim oCampos As Object
   Dim oCampo As Object
   Dim n As Integer

   oCampos = ThisComponent.getTextFields().createEnumeration()
   n = 0
   Do While (oCampos.hasMoreElements())
      oCampo = oCampos.NextElement()
      MsgBox oCampo.getPresentation(True)
      n = n + 1
   Loop
   MsgBox "Campos no documento: " + Str$(n)

End Sub
```

Observe que a criação e o acesso aos elementos de uma enumeração é sequêncial e não indexado ou nomeado. O método getPresentation() retorna o conteúdo do campo se o parâmetro for False.

Geralmente, para recuperar o objeto que nos interessa dentro da coleção retornada por um dos métodos acima, precisamos conhecer o seu nome ou a sua posição dentro da coleção.

6.8 Busca e Substituição

A API do OpenOffice.org tem mecanismos avançados para a busca e substituição de texto num documento.

Para localizar e substituir texto devemos: criar um descritor de busca ou substituição; definir as propriedades e chamar o método apropriado.

A interface **com.sun.star.util.XSearchable** provê os métodos abaixo para busca:

createSearchDescriptor ( ) As Object <SearchDescriptor>
cria um descritor de busca para receber as propriedades da busca.

FindAll (oDescritor As Object) As Object <com.sun.star.container.XIndexAccess>
retorna uma coleção de TextRanges com o resultado da busca

FindFirst (oDescritor As Object) As Object <com.sun.star.uno.XInterface>
retorna um TextRange com o resultado da busca

FindNext (oInicio As Object, oDescritor As Object) As Object <Xinterface>
retorna um TextRange com o resultado da busca. O parâmetro oInicio é ponto de partida para a busca, normalmente o resultado da última chamada a FindFirst ou FindNext.

O serviço **SearchDescriptor** tem os métodos abaixo:

getSearchString ( ) As String
obtém a cadeia de busca

setSearchString (sCadeia As String)
define a cadeia de busca

Eis algumas de suas propriedades:

Propriedade	Descrição
SearchBackwards	Se True busca para o início do documento
SearchCaseSensitive	Se True considera maiúsculas / minúsculas
SearchWords	Se True busca palavras completas
SearchStyles	Se True busca por um estilo

No exemplo a seguir, vamos localizar, no documento, todas as ocorrências da cadeia “( )” e trocar a cor da fonte para vermelha. Digite (ou copie e cole) o código abaixo e execute a sub-rotina.

```
Sub buscaTodas
   Dim oDoc As Object
   Dim oDescBusca As Object

   oDoc = ThisComponent
   oDescBusca = oDoc.createSearchDescriptor()
   oDescBusca.SearchWords = True
   oDescBusca.setSearchString ("( )")
   oResultado = oDoc.findAll( oDescBusca )
```

```
   For n% = 0 To oResultado.Count - 1
     oResultado(n%).CharColor = RGB(255,0,0)
   Next
   MsgBox "Ocorrências de "+oDescBusca.getSearchString()+ _
          " " + Str$(oResultado.Count)
End Sub
```

A tarefa acima pode, também, ser executada com os métodos findFirst e findNext:

```
   oResultado = oDoc.findFirst( oDescBusca )
   Do Until IsNull(oResultado)
     oResultado.CharColor = RGB(250,100,50)
     oResultado = oDoc.findNext(oResultado, oDescBusca )
   Loop
```

A substituição de texto é implementada pela interface **XReplaceable**, que herda as características de **XSearchable**. Ela provê os métodos:

createReplaceDescriptor ( ) As Object <com.sun.star.util.XReplaceDescriptor>
retorna um descritor de substituição para as propriedades

replaceAll (xDescritor As Object <com.sun.star.util.XReplaceDescriptor>) As Long
busca e substitui todas as ocorrências, retorna o total de substituições

O serviço **ReplaceDescriptor** tem métodos para as propriedades da cadeia:

getReplaceString ( ) As String
obtém a cadeia substituta

setReplaceString (sCadeia As String)
define a cadeia substituta

Eis um exemplo simples de busca e substituição:

```
Sub substituiTodas
  Dim oDoc As Object
  Dim oDescritor As Object

  oDoc = ThisComponent
  oDescritor = oDoc.createReplaceDescriptor()
  oDescritor.setSearchString( "( )" )
  oDescritor.setReplaceString( "()" )
  n = oDoc.replaceAll(oDescritor)
  MsgBox n

End Sub
```

Note que não temos um método para substituição interativa. Porém, através da busca e edição, é possível implementar, com facilidade, este procedimento.

6.9 Inserindo objetos

Vamos abordar, rapidamente, outra tarefa comum em processamento de textos. A inserção de objetos de origem externa, como gráficos e documentos, num documento do Writer.

Inserir um documento texto é muito fácil, basta criar um cursor de texto e chamar o método:

insertDocumentFromURL (sURL As String, mPropriedades ( ) As Object)

Os parâmetros deste método são idênticos aos já vistos na seção *Carregando Documentos.*

```
Dim oCursorTxt As Object
Dim mPropriedades ( )
Dim sURL As String
oCursorTxt = ThisComponent.createTextCursor ( )
' sURL = "caminho_completo_do_arquivo"
oCursorTxt.insertDocumentFromURL ( sURL, mPropriedades() )
```

Para inserir na posição do cursor da interface gráfica, crie o cursor de texto a partir do cursor da vista.

A inserção de arquivos gráficos é um pouco mais complicada, para isso você deve:

a) criar um objeto gráfico;

b) definir as suas propriedades;

c) inserir o objeto.

Observe a sub-rotina **Sub inserirGrafico** a seguir:

```
Sub inserirGrafico
   Dim oTxt As Object
   Dim oCursorTxt As Object
   Dim oGrafico As Object
   Dim sURL As String

   oTxt = ThisComponent.getText()
   oCursorTxt = oTxt.createTextCursor()
   oGrafico = ThisComponent.createInstance("com.sun.star.text.GraphicObject")
   sURL = "file:///D:/NAD/OPENOFFICE/HOWTO/FIGURAS/DIALOGO.JPG"
   oGrafico.GraphicURL = sURL
   oGrafico.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
   oTxt.insertTextContent(oCursorTxt.getStart(), oGrafico, False)
End Sub
```

A chamada ao método **createInstance ( )** cria o objecto gráfico, depois definimos as propriedades **GraphicURL** e **AnchorType** e, finalmente, inserimos o gráfico chamando o método **insertTextContent ( )** com os parâmetros adequados.

A propriedade **AnchorType** pode assumir, ainda, os seguintes valores: AT_CHARACTER, AT_PAGE e AS_CHARACTER.

O API do OpenOffice.org (versão 1.0.1) salva no documento apenas a URL do gráfico. Logo, se você removê-lo da sua localização original, o Writer não terá como encontrá-lo.

6.10 Tabelas

Uma tabela é um conjunto de células organizadas por linhas e colunas, onde cada célula tem um nome, normalmente indicado por letras e números. As letras referem-se às colunas e os números às linhas. Contudo, se existirem células mescladas ou divididas, o conceito de coluna poderá desaparecer e o esquema de nomeação torna-se complicado.

Uma célula pode conter texto simples (cadeias e números), gráficos, fórmulas e campos com várias características de formatação.

Uma tabela recebe um nome único no documento, podendo servir como origem para gráficos, ter o seu contéudo ordenado e ser automaticamente formatada.

A API do OpenOffice.org tem diversos serviços e interfaces para lidar com tabelas, vejamos as principais.

A interface com.sun.star.text.XTextTable encarrega-se do gerenciamento de tabelas num documento e contém os métodos abaixo:

initialize (nLinhas As Long, nColunas As Long)
Inicializa uma tabela, recebe o número de linhas e colunas como parâmetros.

getCellNames ( ) As Object < Strings >
Obtém os nomes das células, retorna um conjunto de cadeias

getCellByName (sNomeCelula As String) As Object < com.sun.star.table.XCell >
Obtém a célula nomeada no parâmetro, retorna um objeto XCell

getRows ( ) As Object < com.sun.star.table.XTableRows >
Obtém as linhas da tabela, retorna um objeto para XTableRows.

getColumns ( ) As Object < com.sun.star.table.XTableColumns >
Obtém as colunas da tabela, retorna um objeto para XTableColumns.

createCursorByCellName (sNomeCelula As String) As Object < XTextTableCursor >
Cria um cursor de tabela posicionado na célula nomeada no parâmetro

O serviço TextTable possui diversas propriedades, eis algumas delas:

Propriedade	Descrição
LeftMargin	Contém a margem esquerda da tabela, valor Long
RightMargin	Contém a margem direita da tabela, valor Long
Split	Um valor False impede a divisão da tabela em duas páginas
TableBorder	Contém a descrição da borda da tabela
TableColumnSeparators	Contém a descrição dos separadores de colunas da tabela
BackGraphicURL	Contém a URL do gráfico de fundo da célula

As interfaces XTableRows e XtableColumns, retornadas pelos métodos getRows () e getColumns (), acima, possuem os métodos a seguir:

insertByIndex (nIndice As Long, nTotal As Long)

```
removeByIndex ( nIndice As Long, nTotal As Long )
getElementType ( ) As Type
hasElements ( ) As Boolean
getByIndex ( ) As Variant
getCount ( ) As Long
```

A interface XTextTableCursor, retornada pelo método createCursorByCellName, acima, provê os métodos:

```
getRangeName ( ) As String
```

Retorna a extensão de células do cursor, isto é, os nomes das células superior esquerda e inferior direita, por exemplo A2:D25 (note o separador **:**).

```
goLeft ( nQuant As Integer, bExpande As Boolean ) As Boolean
goRight ( nQuant As Integer, bExpande As Boolean ) As Boolean
goUp ( nQuant As Integer, bExpande As Boolean ) As Boolean
goDown ( nQuant As Integer, bExpande As Boolean ) As Boolean
gotoStart ( bExpande As Boolean )
gotoEnd ( bExpande As Boolean )
```

Métodos para movimentação e seleção, nQuant é a quantidade de células e, se bExpande for True, extende a seleção durante o deslocamento do cursor.

```
gotoCellByName (sNome As String, bExpande As Boolean ) As Boolean
```

Desloca o cursor para a célula nomeada no parâmetro.

```
mergeRange ( ) As Boolean
splitRange ( nQuant As Integer, bHoriz As Boolean ) As Boolean
```

Métodos para fundir e dividir células.

Através do serviço TextTableCursor podemos efetuar a formatação das células, pois, ele inclui os serviços CharacterProperties e ParagraphProperties.

A interface com.sun.star.table.XCellRange define os métodos abaixo para lidar com extensões de células:

```
getCellByPosition ( nColuna As Long, nLinha As Long ) As Object < XCell >
getCellRangeByPosition ( nEsq, nSup, nDir, nInf  ) As Object < XCellRange >
getCellRangeByName ( sRange As String ) As Object < XCellRange >
```

O serviço com.sun.star.table.CellRange inclui com.sun.star.table.CellProperties, que define as propriedades de uma ou mais células. Eis algumas:

Propriedade	**Descrição**
CellBackColor	Valor Long com a cor de fundo da célula
HoriJustify	Alinhamento horizontal do conteúdo da célula (enum CellHoriJustify)
VertJustify	Alinhamento vertical do conteúdo da célula (enum CellVertJustify)
IsTextWrapped	Se True, o texto muda de linha automaticamente
Orientation	Orientação do conteúdo da célula (enum CellOrientation)

RotateAngle	Define a rotação do conteúdo da célula (em 1/100 graus)
TableBorder	Descreve a borda das células (struct TableBorde)
TopBorder	Descreve a borda superior de cada célula (struct BorderLine)
BottomBorder	Descreve a borda inferior de cada célula (struct BorderLine)
LeftBorder	Descreve a borda esquerda de cada célula (struct BorderLine)
RightBorder	Descreve a borda direita de cada célula (struct BorderLine)
NumberFormat	Índice do formato numérico usado nas células (serviço NumberFormatter)
CellProtection	Descreve a proteção da célula (serviço CellProtection)

A interface com.sun.star.table.XCell oferece os métodos abaixo para manipular o conteúdo de uma célula:

```
getFormula ( ) As String
setFormula ( sFormula As String )
```
métodos para obter ou definir a fórmula de uma célula

```
getValue ( ) As Double
setValue ( nValor As Double )
```
métodos para obter ou definir o valor de uma célula

```
getType ( ) As Long < com.sun.star.table.CellContentType >
```
retorna o tipo do conteúdo de uma célula, como enumerado em CellContentType

```
getError ( ) As Long
```
retorna o erro de uma célula, útil para rastrear erros em fórmulas

O nome de uma tabela pode ser obtido ou definido com os métodos a seguir, da interface com.sun.star.container.XNamed:

```
getName ( ) As String
setName ( sNomeTabela As String )
```

O conteúdo de uma tabela pode ser ordenado com a interface com.sun.star.util.XSortable e seus métodos:

```
createSortDescriptor ( ) As Variant <com.sun.star.beans.PropertyValue>
```
retorna um descritor para as propriedades da ordenação

```
sort (xDescritor As Variant <com.sun.star.beans.PropertyValue> )
```
executa a ordenação conforme o descritor

Seguem algumas das propriedades do descritor de ordenação (SortDescriptor):

Propriedade	**Descrição**
IsCaseSensitive	Se True, considera maiúsculas / minúsculas
SortAscending	Define a ordem da ordenação (??)
SortColumns	Se True ordena as colunas, senão ordena as linhas.

O serviço com.sun.star.table.TableSortDescriptor inclui o serviço SortDescriptor e possui as propriedades abaixo:

Propriedade	Descrição
SortFields	Descreve os campos de ordenação <com.sun.star.util.SortField>
MaxFieldCount	Define o número máximo de campos, somente leitura
Orientation	Define a orientação da ordenação <com.sun.star.table.TableOrientation>
ContainsHeader	Se True, não ordena a primeira linha ou coluna

É hora de apresentar um exemplo. Inicialmente, vamos criar, inicializar e inserir uma tabela, com cinco linhas e três colunas, no início do documento corrente. Depois, vamos definir os títulos das colunas. Digite o código abaixo, execute e observe o resultado:

```
Sub criaTabela
   Dim oTxt As Object
   Dim oTab As Object
   '
   ' cria, inicializa e insere a tabela no inicio do documento
   oTxt = ThisComponent.getText()
   oTab = ThisComponent.createInstance("com.sun.star.text.TextTable")
   oTab.initialize(5,3)
   oTxt.insertTextContent(oTxt.createTextCursor(), oTab, FALSE)
   '
   ' preenche os títulos das colunas (células A1, B1 e C1 )
   Dim oCelula As Object
   oCelula = oTab.getCellByName("A1")
   oCelula.setString("Coluna A")
   oCelula = oTab.getCellByName("B1")
   oCelula.setString("Coluna B")
   oCelula = oTab.getCellByName("C1")
   oCelula.setString("Coluna C")
End Sub
```

Primeiro criamos o objeto tabela com *createInstance ()*, depois definimos as linhas e colunas com *initialize ()*, inserimos a tabela no documento com uma chamada ao método *insertTextContent ( )* e, então , obtemos cada uma das células do cabeçalho e definimos o seu conteúdo.

O bloco de código abaixo demonstra o emprego do cursor de tabela, navegação, formatação e mesclagem de células. Acrescente-o à **Sub criaTabela**.

```
   '
   ' cria um cursor de tabela na célula CellNames(0) = A1
   Dim oCurTab As Object
   oCurTab = oTab.createCursorByCellName(oTab.CellNames(0))
   ' seleciona as células A1, B1 e C1
   oCurTab.gotoStart(FALSE)
   oCurTab.goRight(2, True)
   ' Aplica o estilo de parágrafo Título na seleção
   oCurTab.paraStyleName = "Heading"
   ' exibe o nome da extensão selecionada (range)
   ' Note a forma de apresentação (invertida)
   MsgBox oCurTab.getRangeName()
   '
```

```
' nomeia as celulas a partir da linha 2 e
' escreve número da linha na coluna C
Dim sNomes() As Variant
Dim sNomeCelula As String
sNomes = Array("A","B","C")
For i% = 1 To 4
   For j% = 1 To 3
      sNomeCelula = sNomes(j%-1)+ Str$(i%+1)
      oCelula = oTab.getCellByName(sNomeCelula)
      If (j% - 1 = 2) Then
         ' define um valor numérico para a célula
         oCelula.setValue(i% + 1)
      Else
         oCelula.setString(sNomeCelula)
      End If
   Next j%
Next i%
'
' define uma fórmula (soma de C2 até C4) para a célula C5
oCelula = oTab.getCellByName("C5")
oCelula.setFormula("sum <C2:C4>")
'
' funde as células A5 e B5 e muda seu conteúdo
oCurTab.gotoCellByName("A5", False)
oCurTab.goRight(1, True)
oCurTab.mergeRange()
oCelula = oTab.getCellByName("A5")
oCelula.setString("Total")
```

Note a criação do cursor da tabela com createCursorByCellNames (...), o uso do cursor para formatação, o uso dos métodos setValue () e setFormula () e a seleção e mesclagem das células A5 e B5.

No próximo bloco de código, veremos a ordenação de uma extensão de células (A2:C4) da nossa tabela. Basta inserir o código na **Sub criaTabela** e executar para ver o resultado.

```
'
' ordena de A2 até C4
Dim oCampoOrd(0) As New com.sun.star.util.SortField
Dim oDescrOrd As Variant
Dim oExtensao As Object
'
' define a extensão a ser ordenada
oExtensao = oTab.getCellRangeByName("A2:C4")
' define o campo de ordenação e suas propriedades
oCampoOrd(0).Field = 0
oCampoOrd(0).SortAscending = False
oCampoOrd(0).FieldType = com.sun.star.util.SortFieldType.ALPHANUMERIC
' cria o descritor de ordenação
oDescrOrd = oTab.createSortDescriptor()
' define as propriedades do descritor
oDescrOrd(0).Name = "SortFields"
oDescrOrd(0).Value = oCampoOrd()
oDescrOrd(1).Name = "ContainsHeader"
oDescrOrd(1).Value = False
oDescrOrd(2).Name = "SortColumns"
oDescrOrd(2).Value = False
' ordena a extensão
oExtensao.sort(oDescrOrd())
```

Note que iniciamos com a declaração das variáveis, criamos uma extensão de células com uma chamada ao método getCellRangeByName (), definimos as propriedades do campo de ordenação, criamos o descritor e definimos as suas propriedades (o campo de ordenação é uma propriedade do descritor) e, finalmente, invocamos o método sort () para executar a sua tarefa.

Antes de terminar esta seção, vejamos como se dá o acesso às tabelas existentes num documento texto. O processo é o mesmo já visto na seção *Localizando Objetos.*

O método getTextTables () da interface XTextTablesSupplier retorna uma coleção contendo as tabelas do documento. Ele percorre o documento do início para o final.

getTextTables () As Object <com.sun.star.container.XNameAccess>

O exemplo abaixo demonstra a localização das tabelas de um documento, depois obtém as coleções de linhas e colunas da tabela com o nome “Tabela3”.

```
Sub localizaTabelas
   Dim oTabelas As Object
   Dim oTabela As Object
   Dim oLinhas As Object
   Dim oColunas As Object
   Dim n As Integer

   oTabelas = ThisComponent.getTextTables()
   For n = 0 To oTabelas.Count -1
      oTabela = oTabelas.getByIndex(n)
      MsgBox oTabela.getName()
   Next n
   MsgBox "Tabelas no documento: " + Str$(oTabelas.Count)
   '
   If (oTabelas.hasByName (“Tabela3”)) Then
      oTabela = oTabelas.getByName("Tabela3")
      oLinhas = oTabela.getRows()
      oColunas = oTabela.getColumns()
      MsgBox Str$(oLinhas.Count)+" - "+Str$(oColunas.Count)
   End If
End Sub
```

Lembre-se que, para acessar uma tabela, você deve conhecer o seu nome ou o seu índice dentro da coleção e, em seguida, chamar o método getByName () ou getByIndex ().

Após obter a tabela desejada, use as técnicas já apresentadas para editar o seu conteúdo.

7 Documentos do Calc

7.1 Introdução

Um documento do Calc contém uma ou mais folhas de planilhas. Cada planilha é formada por células organizadas em linhas e colunas, de modo bem parecido com o de uma tabela do Writer. Estas células contém, basicamente, texto, que pode representar uma cadeia de caracteres, um valor numérico, um campo ou uma fórmula.

Cada um dos elementos acima (documento, planilha e célula), possui características próprias. Algumas podem ser alteradas, como por exemplo: estilo de página, formato numérico, parágrafo e fonte.

Além das tarefas comuns de edição, podemos aplicar, sobre o conteúdo de uma planilha operações como: ordenação, filtragem, sub-totalização, geração de gráficos, etc.

Podemos, também, inserir, numa planilha, objetos como imagens, desenhos, dados de uma fonte externa e objetos OLE.

Como você já deve ter notado, dado a riqueza e o poder das operações com documentos do Calc, a sua programação é um assunto extenso. Nas próximas seções, tentarei apresentar o básico.

7.2 Planilhas

As funções básicas já abordadas no capítulo *Trabalhando com Documentos*, podem ser usadas também com documentos do Calc. Assim, esta seção vai cobrir algumas operações sobre as folhas das planilhas de um documento do Calc.

O módulo **com.sun.star.sheet**, da API do OpenOffice.org, contém os principais serviços, interfaces e grupos de constantes relacionados com planilhas.

O serviço **SpreadsheetDocument** representa o modelo de um documento do Calc, contendo atributos e uma ou mais planilhas. A interface **XSpreadsheetDocument** provê o método getSheets (), que retorna uma coleção com as planilhas do documento.

```
getSheets ( ) As Object <com.sum.star.sheet.XSpreadsheets>
```

Os objetos desta coleção podem ser acessados pelos seus nomes ou índices. Podemos, ainda, criar uma enumeração e usar o acesso seqüêncial.

A interface **XSpreadsheets**, retornada por getSheets (), define os métodos abaixo:

```
insertNewByName ( sNome As String, iPosicao As Integer )
moveByName ( sNome As String, iDestino As Integer )
copyByName ( sNomeOrigem As String, sNomeCopia As String, iPosicao As Integer )
```

Para fixar os conceitos acima, no exemplo a seguir, vamos criar um documento e efetuar algumas operações com planilhas.

```
Sub criaDocCalc
   Dim oDesk As Variant
   Dim oDoc As Object
   Dim mProp() As Variant
   Dim sURL As String

   ' cria documento do Calc
   oDesk = createUnoService("com.sun.star.frame.Desktop")
   sUrl = "private:factory/scalc"
   oDoc = oDesk.LoadComponentFromURL(sUrl,"_blank",0,mProp())
   ' exibe a propriedade casas decimais padrão do documento
   MsgBox oDoc.StandardDecimals
   '
   ' navega pelas planilhas existentes no documento
   Dim oPlanilhas As Object
   Dim oPlanilha As Object
   Dim sMsg As String
   '
   oPlanilhas = oDoc.getSheets()
   sMsg = ""
   For n=0 To oPlanilhas.Count - 1
      oPlanilha = oPlanilhas.getByIndex(n)
      sMsg = sMsg + oPlanilha.getName() + Chr$(13)
   Next n
   MsgBox sMsg + Str$(oPlanilhas.Count)
   ' acesso nomeado
   If oPlanilhas.hasByName("Planilha2") Then
      oPlanilha = oPlanilhas.getByName("Planilha2")
      MsgBox oPlanilha.PageStyle
   End If
   ' insere uma folha no início
   oPlanilhas.insertNewByName("Planilha4",0)
   MsgBox oPlanilhas.Count
   ' move para o final
   oPlanilhas.moveByName("Planilha4",4)
End Sub
```

Note os parâmetros para a criação do documento, o acesso aos objetos da coleção e o uso dos métodos para inserir e mover uma folha de planilha.

O serviço **SpreadsheetView** fornece funcionalidades relacionadas à vista corrente de um documento do Calc. Ele inclui o serviço **SpreadsheetViewSettings**, que define algumas propriedades de cada uma das planilhas do documento e, a sua interface **XspreadsheetView**, traz os métodos a seguir, para a definição da planilha ativa no documento:

```
getActiveSheet ( ) As Object < XSpreadsheet >
setActiveSheet ( oPlanilha As Object < XSpreadsheet >)
```

Lembre-se que as operações relacionadas com a interface gráfica são gerenciadas pelo objeto controlador do documento.

Digite o código fonte abaixo no final da **Sub criaDocCalc**, execute e observe o resultado.

```
   ' ativa a Planilha4 na GUI
   If oPlanilhas.hasByName("Planilha4") Then
```

```
      MsgBox oDoc.getCurrentController().getActiveSheet().getName()
      oPlanilha = oPlanilhas.getByName("Planilha4")
      oDoc.getCurrentController().setActiveSheet(oPlanilha)
   End If
```

Inicialmente, exibimos o nome da planilha ativa e, então, selecionamos a planilha de nome Planilha4.

Como já demonstrado no capítulo *Documentos do Writer*, podemos alterar o conteúdo de um documento independente da interface gráfica.

7.3 Editando

Para editar o conteúdo de uma ou mais células, devemos ter em mente o seguinte:

a) obter a planilha onde se encontram as células a editar;

b) obter uma extensão contendo as células a serem editadas, este passo é opcional, pois uma planilha é uma extensão de células;

c) obter a célula a editar;

d) definir o conteúdo da célula.

O item a) deve ser solucionado com o uso do método **getSheets ()**, associado ao acesso nomeado, indexado ou enumerado, como demonstrado na seção sobre *Planilhas*.

As ferramentas para solucionar os itens b), c) e d) também já foram apresentadas na seção sobre *Tabelas* e envolve os métodos das interfaces XCellRange e XCell.

Métodos da interface **com.sun.star.table.XCellRange**:

```
getCellByPosition ( nColuna As Long, nLinha As Long ) As Object < XCell >
getCellRangeByPosition ( nEsq, nSup, nDir, nInf  ) As Object < XCellRange >
getCellRangeByName ( sRange As String ) As Object < XCellRange >
```

Devemos lembrar que a posição de uma célula, dentro de uma extensão (range), é relativa ao início da extensão.

Métodos da interface **com.sun.star.table.XCell**:

```
getFormula ( ) As String
setFormula ( sFormula As String )
getValue ( ) As Double
setValue ( nValor As Double )
getType ( ) As Long < com.sun.star.table.CellContentType >
getError ( ) As Long
```

Para relembrar, vejamos um exemplo de edição do conteúdo das células de uma planilha. Digite o código abaixo, execute e observe o resultado.

```
Sub editaPlanilha
   Dim oDesk As Variant
   Dim oDoc As Object
```

```
    Dim mProp() As Variant
    Dim sURL As String
    '
    ' cria documento do Calc
    oDesk = createUnoService("com.sun.star.frame.Desktop")
    sUrl = "private:factory/scalc"
    oDoc = oDesk.LoadComponentFromURL(sUrl, "_blank", 0, mProp())
    '
    ' edita células da planilha
    Dim oPlanilha As Object
    Dim oCelula As Object
    Dim sTitCol() As String

    ' escreve os titulos das colunas
    sTitCol = Array ("CÓDIGO","MATERIAL","QUANT","P. UNIT","P. TOTAL")
    oPlanilha = oDoc.getSheets().getByIndex(0)
    For i% = 0 To 4
        oCelula = oPlanilha.getCellByPosition(i%, 0)
        oCelula.setString(sTitCol(i%))
    Next i%
    '
    ' preenche as células com texto, valor e formula
    For i% = 1 To 6
       For j% = 0 To 4
          oCelula = oPlanilha.getCellByPosition(j%, i%)
          If (j% = 0) Then
             If i% < 4 Then
                oCelula.setString("A" + Str$(i%))
             Else
                oCelula.setString("A" + Str$(i%-3))
             End If
          ElseIf (j% = 1) Then
             oCelula.setString("Material de construção " + Str$(i%))
          ElseIf (j% = 2) Then
             oCelula.setValue(i% * j%)
          ElseIf (j% = 3) Then
             oCelula.setValue(100 * Rnd())
          Else
             sLinha = Trim$(Str$(i%+1))
             sFormula = "=C" + sLinha + " * " + "D" + sLinha
             oCelula.setFormula(sFormula)
          End If
       Next j%
    Next i%
    ' resumo
    oCelula = oPlanilha.getCellByPosition(0, 7)
    oCelula.setString("CUSTO TOTAL")
    '
    sFormula = "=Sum(E1:E6)"
    oCelula = oPlanilha.getCellByPosition(4, 7)
    oCelula.setFormula(sFormula)
    '
End Sub
```

Note que, na segunda chamada do método setFormula (), definimos a fórmula usando o nome de uma função em inglês. Para utilizar os nomes das funções exibidos pela interface gráfica, use a propriedade **FormulaLocal**, do serviço **com.sun.star.sheet.SheetCell**., por exemplo:

```
oCelula = oPlanilha.getCellByPosition(4, 7)
sFormula = "=Soma(E1:E6)"
```

```
oCelula.FormulaLocal = sFormula
```

O serviço **SheetCell** define as seguintes propriedades:

Propriedade	Descrição
Position	Somente leitura, posição da célula na planilha
Size	Somente leitura, tamanho da célula em 1/100 mm
FormulaLocal	Cadeia com o nome local da fórmula
FormulaResultType	Somente leitura, tipo do resultado de uma fórmula
ConditionalFormat	Definições de formatação condicional da célula
ConditionalFormatLocal	Definições locais de formatação condicional da célula
Validation	Definições de validação da célula
ValidationLocal	Definições locais de validação da célula

Na próxima seção, trataremos da criação e emprego de um cursor, outra funcionalidade dos documentos do Calc.

7.4 Navegando pelas Células

Uma célula ou extensão de células é identificada pelo seu endereço, que é formado pelos índices da planilha, coluna inicial, linha inicial, coluna final e linha final. A interface **XCellRangeAddressable** define o método abaixo, para a recuperação deste endereço:

getRangeAddress ( ) As Object < com.sun.star.table.CellRangeAddress >
retorna uma estrutura **CellRangeAddress** com os elementos Sheet, StartColumn, StartRow, EndColumn e EndRow.

Numa planilha, também podemos criar um cursor. Este cursor é uma extensão de células com funcionalidades para navegação, formatação e edição, dentre outras.

O serviço Spreadsheet, através da interface **XSpreadsheet**, define os métodos a seguir, para a criação de um cursor de células:

createCursor ( ) As Object < XSheetCellCursor >
retorna um cursor de célula contendo toda a planilha

createCursorByRange (oExtensao As Object < XSheetCellCursor >)
retorna um cursor de célula contendo a extensão de células do parâmetro

O serviço **SheetCellCursor** fornece toda funcionalidade para operarmos com um cursor de célula.Ele possui as interfaces **XSheetCellCursor** e **XUsedAreaCursor**, além de suportar os serviços **table.CellCursor** e **SheetCellRange**.

A interface **XSheetCellCursor** define os métodos a seguir, para contrair ou expandir o cursor:

collapseToCurrentRegion ()
collapseToCurrentArray ()

```
collapseToMergedArea ()
expandToEntireColumns ()
expandToEntireRows ()
collapseToSize ( nColunas As Long,  nLinhas As Long )
```

Os métodos abaixo, da interface **XUsedAreaCursor**, são úteis quando desejamos operar sobre a área utilizada de uma planilha. Por exemplo, identificar a última célula com dados numa planilha.

```
gotoStartOfUsedArea ( bExpande As Boolean )
gotoEndOfUsedArea (  bExpande As Boolean )
```

A interface **XCellCursor**, do serviço **CellCursor**, provê os seguintes métodos, já explicados na seção *Tabelas*, do capítulo *Documentos do Writer*:

```
gotoStart ( )
gotoEnd ( )
gotoNext ( )
gotoPrevious ( )
gotoOffSet ( nDeslocColuna As Long, nDeslocLinha  As Long )
```

Para demonstrar o uso de um cursor, digite o código fonte abaixo e execute, observando o resultado.

```
Sub criaCursorCelulas
   Dim oDesk As Variant
   Dim oDoc As Object
   Dim mProp() As Variant
   Dim sURL As String
   '
   ' cria documento do Calc
   oDesk = createUnoService("com.sun.star.frame.Desktop")
   sUrl = "private:factory/scalc"
   oDoc = oDesk.LoadComponentFromURL(sUrl, "_blank", 0, mProp())
   '
   ' trabalha com um cursor
   Dim oPlan As Object
   Dim oCursor As Object

   oPlan = oDoc.getSheets().getByIndex(0)
   oCursor = oPlan.createCursorByRange(oPlan.getCellByPosition(0,0))
   ' suporta XSpreadSheet
   MsgBox oCursor.getSpreadsheet().getName()
   ' edita algumas células
   ' note que a posição da célula é relativa
   oCursor.getCellByPosition(0,0).setFormula("Célula A1")
   oCursor.gotoNext()
   oCursor.getCellByPosition(0,0).setFormula("Célula B1")
   oCursor.gotoNext()
   oCursor.getCellByPosition(0,0).setFormula("Célula C1")
   ' move o cursor 5 celulas abaixo
   oCursor.gotoOffSet(0,5)
   oCursor.getCellByPosition(0,0).setFormula("Abaixo de C1")
   oCursor.CharHeight = 16
```

```
   ' obtém toda a área em uso
   oCursor.gotoStartOfUsedArea(False)
   oCursor.gotoEndOfUsedArea(True)
   ' obtem e exibe o endereço do cursor (índices)
   oEnd = oCursor.getRangeAddress()
   MsgBox Str$(oEnd.Sheet)+Str$(oEnd.StartColumn)+Str$(oEnd.StartRow)
   MsgBox Str$(oEnd.Sheet)+Str$(oEnd.EndColumn)+Str$(oEnd.EndRow)
End Sub
```

Aqui, usamos o método createCursorByRange ( ), mas poderíamos ter usado createCursor ( ). Note que, para definir o conteúdo de uma célula, devemos obtê-la chamando o método getCellByPosition ( ), com a posição relativa ao início da extensão de células do cursor. Observe, ainda, como se dá a seleção e identificação da área usada pela planilha.

7.5 Selecionando pela Interface Gráfica

Para obter o conteúdo selecionado num documento do Calc, devemos considerar situações distintas.

O código abaixo define um objeto célula com a primeira célula da extensão.

```
   ' para uma célula: com.sun.star.sheet.SheetCell
   oCel = ThisComponent.getCurrentSelection().getCellByPosition(0,0)
```

O trecho de código a seguir obtém uma extensão de células contínuas e exibe os índices da planilha, coluna inicial e final, linha inicial e final.

```
   ' para uma extensão: com.sun.star.sheet.CellRange
   oExt = oDoc.getCurrentSelection()
   oEnd = oExt.getRangeAddress()
   MsgBox Str$(oEnd.Sheet)+Str$(oEnd.StartColumn)+Str$(oEnd.StartRow)
   MsgBox Str$(oEnd.EndColumn)+Str$(oEnd.EndRow)
```

Podemos, ainda, selecionar extensões de células não contínuas, numa mesma planilha ou em planilhas diferentes, o bloco de código fonte abaixo lida com esta situação. Num documento do Calc, selecione extensões não contínuas, crie a macro e execute para ver a saída.

```
Sub selecaoExtensoes
 Dim oDoc As Object
 Dim oExts As Object

 oDoc = ThisComponent
 ' oExts suporta com.sun.star.sheet.SheetCellRanges
 oExts = oDoc.getCurrentSelection()
 MsgBox oExts.getCount()
 oEnd = oExts.getRangeAddresses()
 For n=0 To UBound(oEnd)
  MsgBox Str$(oEnd(n).Sheet)+Str$(oEnd(n).StartColumn)+Str$(oEnd(n).StartRow)
 Next n
 MsgBox oExts.getRangeAddressesAsString()
```

```
End Sub
```

A API do OpenOffice oferece outras funcionalidades como seleção de um ou mais objetos Shape e a seleção de uma extensão durante a execução de uma macro. Porém, deixo por conta do leitor a exploração destas operações.

7.6 Formatando

A apresentação final de um trabalho é muito importante e o Calc oferece uma vasta gama de possibilidades para nos auxiliar na formatação dos nossos documentos.

Formantando Parágrafos e Caracteres

Inicialmente, podemos aplicar ao conteúdo de uma célula as técnicas de formatação de parágrafos e caracteres. Os principais serviços, já apresentados no capítulo sobre *Documentos do Write*, na seção *Formatando Texto* são:

```
com.sun.star.style.ParagraphProperties
com.sun.star.style.CharacterProperties
```

O serviço **com.sun.star.table.TableRow** define as seguintes propriedades para formatação de linhas:

Propriedade	Descrição
Height	Define a altura da linha em 1/100 mm
OptimalHeight	Se True, ajusta automaticamente a altura da linha
IsVisible	Se True, a linha será exibida
IsStartOfNewPage	Se True, insere uma quebra vertical de página nesta linha

O serviço **com.sun.star.table.TableColumn** define as propriedades abaixo para formatação de colunas:

Propriedade	Descrição
Width	Define a largura da coluna em 1/100 mm
OptimalWidth	Se True, ajusta automaticamente a largura da coluna
IsVisible	Se True, a coluna será exibida
IsStartOfNewPage	Se True, insere uma quebra horizontal de página nesta coluna

Outras propriedades para a formatação de células são cobertas pelos serviços:

```
com.sun.star.table.CellProperties
com.sun.star.table.TableBorder
com.sun.star.table.BorderLine
```

O serviço CellProperties contém diversas propriedades. As mais importantes foram apresentadas no capítulo *Documentos do Writer*, na seção sobre *Tabelas*.

Vamos aplicar alguma formatação ao exemplo criado pela **Sub editaPlanilha**. Adicione o código fonte abaixo no final da sub-rotina, execute e observe a saída:

```
    ' Formatação
    Dim oExtensao As Object
    Dim oLinha As Object
    Dim oColuna As Object
    '
    ' formata parágrafos e caracteres
    oExtensao = oPlanilha.getCellRangeByPosition(0, 0, 5, 0)
    oExtensao.CharHeight = 12
    oExtensao.CharWeight = com.sun.star.awt.FontWeight.BOLD
    oExtensao.ParaAdjust = com.sun.star.style.ParagraphAdjust.CENTER
    ' ajusta a altura da linha 1
    oLinha = oPlanilha.getRows().getByIndex(0)
    oLinha.OptimalHeight = True
    ' ajusta a largura da coluna B
    oColuna = oPlanilha.getColumns().getByIndex(1)
    oColuna.OptimalWidth = True
```

A destacar, o método de recuperação de linhas e colunas, via acesso indexado.

Mesclando Células

Numa planilha, podemos mesclar células, a interface **com.sun.star.util.XMergeable** define os métodos abaixo, com esta finalidade:

merge (bMerge As Boolean)
Se bMerge é True mescla a área do objeto, senão separa.

getIsMerged ( ) As Boolean
Retorna True se a área do objeto estiver mesclada, senão retorna False

Inserindo Bordas

Outro aspecto importante na apresentação de uma planilha é a colocação de bordas em torno das células. O serviço **CellProperties** contém as seguintes propriedades, que definem o tipo de borda de uma célula ou extensão de células:

Popriedade	Descrição
TableBorder	Define a borda de uma célula ou extensão de células
TopBorder	Define a borda superior de cada célula da extensão
BottomBorder	Define a borda inferior de cada célula da extensão
RightBorder	Define a borda direita de cada célula da extensão
LeftBorder	Define a borda esquerda de cada célula da extensão

O principal elemento de cada uma destas propriedades é a estrutura **BorderLine**, contendo, dentre outros, os campos **Color** e **OuterLineWidth** para a cor e a espessura da linha. Para a propriedade **TableBorder**, podemos definir quatro linhas: **TopLine**, **BottomLine**, **RightLine** e **LeftLine**.

Prosseguindo com o nosso exemplo, vamos mesclar algumas células e, depois, definir bordas para a nossa planilha. Acrescente o código fonte a seguir na sub-rotina **Sub editaPlanilha** e observe o resultado da execução:

```
' mescla células
oExtensao = oPlanilha.getCellRangeByPosition(0, 7, 3, 7)
oExtensao.merge(True)
' Define Borda
Dim oBorda As New com.sun.star.table.TableBorder
Dim oLinBorda As New com.sun.star.table.BorderLine
'
oExtensao = oPlanilha.getCellRangeByPosition(0, 0, 4, 7)
' define a espessura e a cor da linha da borda
oLinBorda.OuterLineWidth = 30
oLinBorda.Color = CLng( "&H000099" )
oExtensao.setPropertyValue("TopBorder", oLinBorda)
oExtensao.setPropertyValue("RightBorder", oLinBorda)
oExtensao.setPropertyValue("LeftBorder", oLinBorda)
oExtensao.setPropertyValue("BottomBorder", oLinBorda)
' define e aplica uma TableBorder
oLinBorda.OuterLineWidth = 100
oLinBorda.Color = CLng( "&HAABBCC" )
oBorda.TopLine = oLinBorda
oBorda.BottomLine = oLinBorda
oBorda.RightLine = oLinBorda
oBorda.LeftLine = oLinBorda
oBorda.IsTopLineValid = True
oBorda.IsBottomLineValid = True
oBorda.IsRightLineValid = True
oBorda.IsLeftLineValid = True
oExtensao.setPropertyValue("TableBorder", oBorda)
```

Inicialmente definimos as bordas de todas as células de *oExtensao*, depois aplicamos uma borda na extensão. Note os campos **Is...Valid** definidos como **True**.

Formatação Numérica

O OpenOffice possui diversas categorias de formatação numérica. Cada categoria possui vários formatos pré-definidos. Através da interface com.sun.star.util.XNumberFormatsSupplier podemos ler, modificar e adicionar novos formatos aos nossos documentos.

A interface **XNumberFormatsSupplier** contém os métodos abaixo:

```
getNumberFormats ( ) As Object < com.sun.star.util.XNumberFormats >
getNumberFormatsSettings ( ) As Object < com.sun.star.beans.XPropertySet >
```

Entre os métodos da interface **XNumberFormats**, temos:

```
getByKey (nChave As Long) As Object < XPropertySet >
queryKeys (nCategoria As Long, nLocal As Long, bInserir As Boolean) As aVetor ( )
queryKey (sFormato As String, nLocal As Long, bProcura As Boolean) As Long
```

addNew (sFormato As String, nLocal As Long) As Long

A interface com.sun.star.util.XNumberFormatTypes contém métodos para obter o índice de alguns formatos pré-definidos. Aqui, usaremos o método:

getStandardFormat (nTipo AS Long, nLocal As Long) As Long

O grupo de constantes **com.sun.star.util.NumberFormat** define valores para, dentre outras, as seguintes categorias: DATE, TIME, CURRENCY, NUMBER, DATETIME.

O serviço **CellProperties** contém a propriedade **NumberFormat**, que define o tipo formatação numérica de uma célula ou extensão de células.

Vejamos um exemplo ilustrativo. Num documento do Calc, crie a macro abaixo, execute-a e observe o resultado.

```
Sub exibeFormatosNumericos
   Dim oDoc As Object
   Dim oFormatos As Object
   Dim mChaves As Variant
   Dim mProp As Variant

   oDoc = ThisComponent
   oFormatos = oDoc.getNumberFormats()
   oSettings = oDoc.getNumberFormatSettings()
   oInfo = oSettings.getPropertySetInfo()
   oProp = oInfo.getProperties()
   MsgBox UBound(oProp)
   For n = 0 To UBound(oProp)
      MsgBox oProp(n).Name
   Next n
   mChaves = oFormatos.queryKeys (0, oLocal, FALSE)
   MsgBox UBound(mChaves)
   mProp = oFormatos.getByKey(11).getPropertyValues()
   ' obtem os nomes das propriedades
   sMsg = ""
   For n=0 To UBound(mProp)
     sMsg = sMsg + mProp(n).Name + Chr$(13)
   Next n
   MsgBox sMsg
   ' exibe o valor de FormatString
   MsgBox mProp(0).Value

End Sub
```

Agora, vamos retornar à nossa sub-rotina **Sub editaPlanilha** para aplicar formatação numérica em algumas células. Adicione o código abaixo, execute e observe a saída:

```
   ' Formatos Numéricos
   Dim oFormatos As Object
   Dim aLocal() As New com.sun.star.lang.Locale
   ' obtem os formatos numericos do modelo
   oFormatos = oDoc.getNumberFormats()
   ' obtém o índice do formato padrão para MOEDA
   n%=oFormatos.getStandardFormat(com.sun.star.util.NumberFormat.CURRENCY,aLocal())
```

```
' obtem uma extensão de células
oExtensao = oDoc.getSheets().getByIndex(0).getCellRangeByPosition(3,1,4,6)
' altera o formato padrão para MOEDA
oExtensao.NumberFormat = n%
' obtém uma célula e altera para MOEDA
oCelula = oPlanilha.getCellByPosition(4, 7)
oCelula.NumberFormat = n%
```

Note a declaração dos objetos, o uso do método getStandardFormat () e o uso da propriedade **NumberFormat** para alterar a formatação das células.

Formatação Condicional

O serviço **com.sun.star.sheet.SheetCellRange** contém as seguintes propriedades para formatação condicional:

conditionalFormat - define condições para formatação condicional
conditionalFormatLocal - define condições locais para formatação condicional

A interface **XSheetConditionalEntries** provê os métodos a seguir para operações com as condições de formatação:

addNew (mCond () As < com.sun.star.beans.PropertyValue >)
o parâmetro mCond é um vetor com as seguintes entradas:

- "Operator" com os possíveis valores, definidos em com.sun.star.sheet.ConditionOperator

  NONE, EQUAL, NOT_EQUAL, GREATER, GREATER_EQUAL,

  LESS, LESS_EQUAL, BETWEEN, NOT_BETWEEN, FORMULA

- "Formula1" contendo uma cadeia com um valor ou fórmula

- "Formula2" contendo um valor ou fórmula, usada quando o operador for BETWEEN

- "StyleName" contendo um nome de estilo de formatação de célula

removeByIndex (nIndice As Long)
remove a condição nIndice

clear ( )
limpa as condições atuais de formatação

Acrescente o código abaixo no final da **Sub editaPlanilha**, execute e observe a saída:

```
' FORMATAÇÃO CONDICIONAL
'
Dim mCond (2) As New com.sun.star.beans.PropertyValue
Dim oEntradas As Variant
' obtem uma extensão de células
oExtensao = oDoc.getSheets().getByIndex(0).getCellRangeByPosition(4,1,4,6)
' obtém a formatação condicional corrente
oEntradas = oExtensao.getPropertyValue("ConditionalFormat")
' define as propriedades de uma condição para formatação
mCond(0).Name = "Operator"
mCond(0).Value = com.sun.star.sheet.ConditionOperator.GREATER
mCond(1).Name = "Formula1"
```

```
    mCond(1).Value = "500"
    mCond(2).Name = "StyleName"
    mCond(2).Value = "Result"
    ' limpa as condiçoes existentes
    oEntradas.clear()
    ' adiciona a nova condição
    oEntradas.addNew(mCond())
    ' aplica a formatação condicional
    oExtensao.setPropertyValue("ConditionalFormat", oEntradas)
```

Neste código, aplicamos o estilo “Resultado” se o valor da célula for maior que R$ 500,00.

Podemos, também, aplicar estilos de formatação de células definidos pelos usuários.

7.7 Busca e Substituição

A busca e substituição, num documento do Calc, funciona de modo semelhante àquele já demonstrado na seção *Busca e Substituição*, do capítulo *Documentos do Writer*.

Relembrando, a interface **com.sun.star.util.XSearchable** provê os métodos abaixo para busca:

```
createSearchDescriptor ( ) As Object <SearchDescriptor>
FindAll ( oDescritor As Object ) As Object <com.sun.star.container.XIndexAccess>
FindFirst ( oDescritor As Object ) As Object <com.sun.star.uno.XInterface>
FindNext ( oInicio As Object, oDescritor As Object ) As Object <Xinterface>
```

O serviço **SearchDescriptor** tem os métodos abaixo:

```
getSearchString ( ) As String
setSearchString ( sCadeia As String )
```

Eis algumas de suas propriedades:

Propriedade	Descrição
SearchBackwards	Se True, busca para o início do documento
SearchCaseSensitive	Se True, considera maiúsculas / minúsculas
SearchWords	Se True, localiza apenas as células com o texto e nada mais, senão localiza também as células onde o texto é parte do conteúdo.
SearchStyles	Se True, busca por um estilo de célula

Vejamos um exemplo de busca: crie um novo documento do Calc, preencha algumas células com o texto “TOTAL” e “TOTAL GERAL”, em seguida crie a macro abaixo e execute.

```
Sub buscaTexto
    Dim oDoc As Object
    Dim oDescBusca As Object
    Dim oPlanilha As Object

    oDoc = ThisComponent
```

```
   oPlanilha = oDoc.Sheets(0)
   oDescBusca = oPlanilha.createSearchDescriptor()
   oDescBusca.SearchWords = True
   oDescBusca.setSearchString ("TOTAL")
   oResultado = oPlanilha.findFirst( oDescBusca )
   i = 0
   Do Until IsNull(oResultado)
     oResultado.CharColor = RGB(250,100,50)
     oResultado = oPlanilha.findNext(oResultado, oDescBusca )
     i = i + 1
   Loop
   MsgBox "Ocorrências de "+oDescBusca.getSearchString()  + " " + Str$(i)
End Sub
```

Agora, altere a propriedade **SearchWords** para **False** e execute novamente a macro observando o seu resultado.

A substituição de texto é implementada pela interface **XReplaceable**, que herda as características de **XSearchable**. Ela provê os métodos:

```
createReplaceDescriptor ( ) As Object <com.sun.star.util.XReplaceDescriptor>
replaceAll ( xDescritor As Object <com.sun.star.util.XReplaceDescriptor>) As Long
```

O serviço **ReplaceDescriptor** tem métodos para as propriedades da cadeia:

```
getReplaceString ( ) As String
setReplaceString ( sCadeia As String )
```

Eis uma busca e substituição simples, no mesmo documento do exemplo acima:

```
Sub substituiTexto
  Dim oDoc As Object
  Dim oDescritor As Object
  Dim oPlanilha As Object

  oDoc = ThisComponent
  oPlanilha = oDoc.Sheets(0)
  oDescritor =  oPlanilha.createReplaceDescriptor()
  oDescritor.setSearchString( "TOTAL GERAL" )
  oDescritor.setReplaceString( "Total" )
  n =  oPlanilha.replaceAll(oDescritor)
  sMsg = oDescritor.getSearchString()+“ por “+oDescritor.getReplaceString()
  MsgBox sMsg + “ = “ + Str$(n)
End Sub
```

Note que devemos executar a busca e a substituição numa extensão de células. Nos exemplos, utilizamos toda a planilha.

7.8 Ordenando

Para a ordenação de uma extensão de células, devemos utilizar os mesmos conceitos e serviços já apresentados na seção *Tabelas*, do capítulo sobre *Documentos do Writer*.

Seguem, abaixo, os principais métodos e propriedades utilizados nesta tarefa.

Métodos da interface **com.sun.star.util.XSortable**:

```
createSortDescriptor ( ) As Variant <com.sun.star.beans.PropertyValue>
sort (xDescritor As Variant <com.sun.star.beans.PropertyValue> )
```

Algumas das propriedades do descritor de ordenação (SortDescriptor):

Propriedade	Descrição
IsCaseSensitive	Se True, considera maiúsculas / minúsculas
SortAscending	Define a ordem da ordenação
SortColumns	Se True ordena as colunas, senão ordena as linhas.

O serviço **com.sun.star.table.TableSortDescriptor** inclui o serviço SortDescriptor e possui as propriedades abaixo:

Propriedade	Descrição
SortFields	Descreve os campos de ordenação <com.sun.star.util.SortField>
MaxFieldCount	Define o número máximo de campos, somente leitura
Orientation	Define a orientação da ordenação <com.sun.star.table.TableOrientation>
ContainsHeader	Se True, não ordena a primeira linha ou coluna

No próximo bloco de código, veremos a ordenação de uma extensão de células (A2:E7), do documento criado pela **Sub editaPlanilha**. Digite o código abaixo no final da macro, execute e observe a saida.

```
' ================================
' ORDENANDO UMA EXTENSÃO DE CÉLULAS
' ================================
Dim oCampoOrd(0) As New com.sun.star.util.SortField
Dim oDescrOrd As Variant
'
' define a extensão a ser ordenada (A2:E7)
oExtensao = oDoc.getSheets().getByIndex(0).getCellRangeByPosition(0,1,4,6)
' define o campo de ordenação e suas propriedades
oCampoOrd(0).Field = 0
oCampoOrd(0).SortAscending = True
oCampoOrd(0).FieldType = com.sun.star.util.SortFieldType.ALPHANUMERIC
' cria o descritor de ordenação
oDescrOrd = oExtensao.createSortDescriptor()
' define as propriedades do descritor
oDescrOrd(0).Name = "SortFields"
oDescrOrd(0).Value = oCampoOrd()
oDescrOrd(1).Name = "ContainsHeader"
oDescrOrd(1).Value = False
oDescrOrd(2).Name = "SortColumns"
oDescrOrd(2).Value = False
' ordena a extensão
oExtensao.sort(oDescrOrd())
```

Note que iniciamos com a declaração das variáveis, criamos uma extensão de células com uma chamada ao método getCellRangeByPosition (), definimos as propriedades do campo de ordenação, criamos o descritor e definimos as suas propriedades (o campo de ordenação é uma propriedade do descritor) e, finalmente, invocamos o método sort () para executar a sua tarefa.

7.9 Filtrando dados

Nesta seção, veremos os principais serviços e interfaces relacionados com a aplicação de filtros numa extensão de células.

A interface **com.sun.star.sheet.XSheetFilterable** define os seguintes métodos:

createFilterDescriptor (bVazio As Boolean) As Object < XSheetFilterDescriptor >
cria um descritor vazio se bVazio for True, senão preserva as informações

filter (oDescritor As Object < XSheetFilterDescriptor >)
aplica o filtro na extensão de células

O serviço **SheetFilterDescriptor** controla as condições da operação de filtragem. Eis algumas de suas propriedades:

IsCaseSensitive	- se True, destingue maiúsculas e minúsculas
SkipDuplicates	- seTrue, as entradas duplicadas serão excluídas
Orientation	- filtra por Linhas ou Colunas (TableOrientation)
ContainsHeader	- a primeira linha ou coluna é um cabeçalho
CopyOutputData	- se True, o resultado será copiado para outro local
OutputPosition	- local da cópia (estrutura CellAddress)

A interface **XSheetFilterDescriptor** contém os métodos abaixo para obter e definir os campos com os critérios do filtro:

getFilterFields ( ) aCampos () As Object < TableFilterField >
retorna os campos com os critérios do filtro

setFilterFields (aCampos () As Object < TableFilterField >)
define os campos com os critérios do filtro

A estrutura **com.sun.star.table.TableFilterField** contém os elementos a seguir:

Connection	- como será a conexão com a condição anterior (Or / And)
Field	- a coluna usada na condição
Operator	- operador condicional (FilterOperator)
IsNumeric	- se True, o valor do campo (Field) é numérico
NumericValue	- valor do campo, usar se IsNumeric for True
StringValue	- cadeia de caracteres, usar se IsNumeric for False

Os passos necessários para filtrar dados numa planilha são:

a) obter a extensão de células a ser filtrada;

b) definir a estrutura dos campos com os critérios do filtro;

c) criar e definir os dados do descritor do filtro;

d) aplicar o filtro.

Vejamos um trecho de código fonte que aplica um filtro simples numa extensão de células.

```
Sub aplicaFiltro
   Dim oDoc As Object
   Dim oPlan As Object
   Dim oExt As Object
   Dim oDescFiltro As Variant
   Dim oCamposFiltro(0) As New com.sun.star.sheet.TableFilterField

   oDoc = ThisComponent
   oPlan = oDoc.getSheets().getByIndex(0)
   oExt = oPlan.getCellRangeByPosition(0,1,4,6)
   ' define a estrutura TableFilterField
   oCamposFiltro(0).Field = 4
   oCamposFiltro(0).IsNumeric = True
   oCamposFiltro(0).Operator = com.sun.star.sheet.FilterOperator.GREATER_EQUAL
   oCamposFiltro(0).NumericValue = 300
   ' cria o descritor do filtro vazio (True)
   oDescFiltro = oExt.createFilterDescriptor (True)
   ' define o campo de filtro
   oDescFiltro.setFilterFields (oCamposFiltro())
   ' aplica o filtro
   oExt.filter (oDescFiltro)
   '
   MsgBox "OK para filtrar Caracteres"
   ' redefine a estrutura TableFilterField
   oCamposFiltro(0).Field = 0
   oCamposFiltro(0).IsNumeric = False
   oCamposFiltro(0).Operator = com.sun.star.sheet.FilterOperator.GREATER
   oCamposFiltro(0).StringValue = "A 1"
   ' cria um descritor de filtro vazio (True)
   oDescFiltro = oExt.createFilterDescriptor (True)
   ' define o campo de filtro
   oDescFiltro.setFilterFields (oCamposFiltro())
   ' aplica o filtro
   oExt.filter (oDescFiltro)
End Sub
```

Se você quiser experimentar este código, crie uma macro no documento gerado pela **Sub editaPlanilha** e execute, observando a saída. Note que a estrutura TableFilterField pode ter mais de um campo com critérios de filtragem.

Tente acrescentar código para copiar o resultado do filtro na Planilha2 do documento.

7.10 Inserindo Subtotais

Operações de subtotais, sobre numa extensão de células, são de grande praticidade. A API do OpenOffice.org oferece diversos serviços e interfaces para a programação de subtotais.

A interface **XSubTotalCalculatable** define os métodos abaixo:

createSubTotalDescriptor (bVazio As Boolean) As Object < XSubTotalDescriptor >
cria um descritor de subtotal vazio se bVazio for True, senão preserva dados anteriores

applySubTotals (oDesc As Object < XSubTotalDescriptor >, bSubstitui As Boolean)

aplica o subtotal, se bSubstitui for True o resultado anterior será substituído

removeSubTotals ( )
remove os subtotais do objeto

O serviço **SubTotalDescriptor** é uma representação de como os subtotais serão criados. A seguir, algumas de suas propriedades:

InsertPageBreaks	- insere quebra de página após cada subtotal
IsCaseSensitive	- considera letras maiúsculas / minúsculas
EnableUserSortList	- se True permite a definição de uma lista para ordenação
UserSortListIndex	- a lista para ordenação, se EnableUserSortList for True
EnableSort	- define se o conteúdo dos campos serão ordenados
SortAscending	- se True ordenação crescente (depende de EnableSort)

Além das propriedades acima, o descritor deve conter os campos de subtotais. A interface **XSubTotalDescriptor** provê métodos para adicionar ou limpar campos:

addNew (aColunas() As < SubTotalColumn >, nColGrupo As Long)
adiciona os campos ao descritor, nColGrupo especifica a coluna base para agrupamento

clear ( )
remove todos os campos de subtotais do objeto

A estrutura **com.sun.star.sheet.SubTotalColumn** contém os elementos:

Column	- índice da coluna a subtotalizar
Function	- o tipo de subtotal, definido em com.sun.star.sheet.GeneralFunction

Os passos necessários para adicionar linhas com subtotais numa extensão de células são:

a) obter a extensão de células a ser subtotalizada;

b) definir a estrutura dos campos de subtotal;

c) criar e definir os dados do descritor de subtotal;

d) aplicar a operação de subtotal.

Vejamos um trecho de código fonte que aplica um subtotal numa extensão de células.

```
Sub aplicaSubTotal
   Dim oDoc As Object
   Dim oPlan As Object
   Dim oExt As Object
   Dim oDescSubTotal As Variant
   Dim oCamposSubTotal(0) As New com.sun.star.sheet.SubTotalColumn

   oDoc = ThisComponent
   oPlan = oDoc.getSheets().getByIndex(0)
   oExt = oPlan.getCellRangeByPosition(0,1,4,6)
   ' define a estrutura SubTotalColumn
   oCamposSubTotal(0).Column = 4
   oCamposSubTotal(0).Function = com.sun.star.sheet.GeneralFunction.SUM
   ' cria o descritor do subtotal
   oDescSubTotal = oExt.createSubTotalDescriptor (True)
   ' adiciona o campo subtotal e a coluna base
   oDescSubTotal.addNew (oCamposSubTotal(), 0)
```

```
   ' aplica o subtotal
   oExt.applySubTotals(oDescSubTotal, True)
End Sub
```

Se você quiser experimentar este código, crie uma macro no documento gerado pela **Sub editaPlanilha**, execute e analise o resultado.

7.11 Gráficos

Um gráfico é um documento embutido num outro documento do OpenOffice. A API do OpenOffice.org contém diversos serviços e interfaces para a geração de gráficos, a partir dos dados contidos numa planilha.

O serviço **com.sun.star.table.TableCharts** suporta os métodos das interfaces XTableCharts, XIndexAccess e XEnumerationAccess.

Métodos da interface **com.sun.star.table.XTableCharts**:

```
addNewByName ( sNome As String,
                    oArea As Object < com.sun.star.awt.Rectangle >,
                    oExt ( ) As Object < com.sun.star.table.CellRangeAddress >,
                    bTitCol As Boolean, bTitLin As Boolean )
```

Adiciona o gráfico na coleção de gráficos da planilha. Os seus parâmetros são:

```
sNome      - o nome do gráfico
oArea      - a área (retângulo) aonde o gráfico será plotado
oExt       - a extensão de células com os dados
bTitCol    - se True, os dados da linha superior serão usados na legenda do gráfico
bTitLin    - se True, os dados da primeira coluna serão usados como títulos no eixo
```

```
removeByName ( sNome As String )
```

remove o gráfico com o nome **sNome** da coleção.

O serviço **com.sun.star.table.TableChart** suporta a interface XTableChart com os métodos:

```
getHasColumnHeaders ( ) As Boolean
setHasColumnHeaders ( bTituloColuna As Boolean )
getHasRowHeaders ( ) As Boolean
setHasRowHeaders ( bTituloColuna As Boolean )
getRanges ( ) As sExtensoes ( ) < CellRangeAddress >
setRanges ( sExtensoes ( ) As Object < CellRangeAddress > )
```

Um documento gráfico contém uma referência para uma fonte de dados, um diagrama e algumas propriedades como título, sub-título e legenda. O serviço **ChartDocument** é o modelo do documento gráfico e suporta as interfaces XChartDocument, XPropertySet e XMultiServiceFactory . Ele possui as seguintes propriedades:

```
HasMainTitle        - Se True, exibe o título principal
HasSubTitle                 - Se True, exibe o subtítulo
HasLegend           - Se True, exibe a legenda
```

A interface com.sun.star.chart.XChartDocument provê os métodos:

```
getTitle ( ) As < com.sun.star.drawing.XShape >
getSubTitle ( ) As < com.sun.star.drawing.XShape >
getLegend ( ) As < com.sun.star.drawing.XShape >
getArea ( ) As < com.sun.star.beans.XPropertySet >
getDiagram ( ) As < com.sun.star.chart.XDiagram >
setDiagram ( oDiagrama As < com.sun.star.chart.XDiagram >)
getData ( )  As < com.sun.star.chart.XChartData >
attachData ( oDadosExternos  As < com.sun.star.chart.XChartData >)
```

O diagrama é o objeto que contém a forma do gráfico, baseada no serviço **Diagram** e suas interfaces XDiagram e XPropertySet. Diferentes tipos de diagramas podem ser criados com o método **createInstance ()**, da interface **XMultiServiceFactory**. Eis alguns serviços para tipos de diagramas:

```
com.sun.star.chart.BarDiagram
com.sun.star.chart.LineDiagram
com.sun.star.chart.PieDiagram
com.sun.star.chart.AreaDiagram
com.sun.star.chart.XYDiagram
```

Podemos alterar a propriedade **Diagram** de um gráfico com o código a seguir:

```
   oDiagrama = oGraf.createInstance("com.sun.star.chart.PieDiagram")
   oGrafico.setDiagram ( oDiagrama )
```

O serviço **com.sun.star.chart.Diagram** tem as propriedades abaixo:

DataRowSource	- define se a série de dados é por linhas ou colunas
DataCaption	- define como a legenda dos dados será exibida

Para criar um gráfico devemos seguir os passos abaixo:

a) definir os dados a serem exibidos no gráfico;

b) definir a área retangular aonde o gráfico será desenhado;

c) adicionar o gráfico à coleção;

d) recuperar o objeto gráfico embutido.

e) definir as propriedades do gráfico.

Vejamos um exemplo simples. Digite a macro a seguir e execute para ver a saída:

```
Sub criaGrafico
   Dim oDesk As Variant
   Dim oDoc As Object
   Dim mProp() As Variant
   Dim sURL As String
   ' cria documento do Calc
   oDesk = createUnoService("com.sun.star.frame.Desktop")
   sUrl = "private:factory/scalc"
```

```
   oDoc = oDesk.LoadComponentFromURL(sUrl, "_blank", 0, mProp())
   ' edita células da planilha
   Dim oPlanilha As Object
   Dim oCelula As Object
   Dim sTitCol() As String
   ' escreve os titulos das colunas
   sTitCol = Array ("Período","Computador","Periférico","Serviço","Total")
   oPlanilha = oDoc.getSheets().getByIndex(0)
   For i% = 0 To 4
       oCelula = oPlanilha.getCellByPosition(i%, 0)
       oCelula.setString(sTitCol(i%))
   Next i%
   ' preenche as células com texto, valor e formula
   For i% = 1 To 4
      For j% = 0 To 4
         oCelula = oPlanilha.getCellByPosition(j%, i%)
         If (j% = 0) Then
            oCelula.setString(Str$(i%) + "º Trim")
         ElseIf (j% > 0 And j < 4) Then
            oCelula.setValue(1000 * Rnd())
         Else
            sExt = "B" + Trim$(Str$(i%+1)) + ":" + "D" + Trim$(Str$(i%+1))
            sFormula = "=Soma(" + sExt + ")"
            oCelula.FormulaLocal = sFormula
         End If
      Next j%
   Next i%
   '
   ' insere um gráfico de colunas
   Dim oRet As New com.sun.star.awt.Rectangle
   Dim oEndExt(0) As New com.sun.star.table.CellRangeAddress
   Dim oGraficos As Object
   ' define o nome do gráfico
   sNome = "Trimestral"
   ' define o endereço da extensão de células com os dados
   oEndExt(0).Sheet = 0
   oEndExt(0).StartColumn = 0
   oEndExt(0).StartRow = 0
   oEndExt(0).EndColumn = 3
   oEndExt(0).EndRow = 4
   ' define a área do gráfico
   oRet.X = 1000
   oRet.Y = 3000
   oRet.Width = 12000
   oRet.Height = 12000
   ' obtém a coleção de gráficos da Planilha1
   oGraficos = oPlanilha.Charts
   ' adiciona um gráfico à coleção
   oGraficos.addNewByName(sNome, oRet, oEndExt(), True, True)
   ' define as propriedades(Titulo)
   oGraf = oGraficos.getByName(sNome).getEmbeddedObject()
   oGraf.Title.String = "Faturamento Trimestral"
   oGraf.HasSubTitle = True
   oGraf.SubTitle.String = "Ano 2003"
End Sub
```

Começamos preenchendo as células com os dados, depois, definimos o nome do gráfico, a extensão de células com os dados e a área (retângulo) aonde o gráfico será plotado. Após estes passos preliminares, obtemos a coleção dos gráficos da planilha e adicionamos o nosso gráfico na coleção, com uma chamada ao método **addNewByName ( )**. Finalmente, obtemos o objeto gráfico embutido, com uma chamada ao método **getByName ( )** associada ao método **getEmbeddedObject ( )** e, então, alteramos algumas propriedades do gráfico.

Aproveitando o exemplo acima, vamos mostrar como alterar a propriedade **Diagram** de um gráfico. Acrescente o código fonte abaixo ao final da **Sub criaGrafico** e execute para criar um gráfico do tipo Pizza, representando os dados da linha 2 da planilha.

```
' Criando um gráfico do tipo Pizza
Dim oExtPizza(1) As New com.sun.star.table.CellRangeAddress
'
MsgBox "OK para uma Pizza"
' remove o gráfico de barras
oGraficos.removeByName(sNome)
sNome = "GrafPizza"
' define o endereço da extensão de células com legendas
oExtPizza(0).Sheet = 0
oExtPizza(0).StartColumn = 0
oExtPizza(0).StartRow = 0
oExtPizza(0).EndColumn = 3
oExtPizza(0).EndRow = 0
' define o endereço da extensão de células com dados
oExtPizza(1).Sheet = 0
oExtPizza(1).StartColumn = 0
oExtPizza(1).StartRow = 1
oExtPizza(1).EndColumn = 3
oExtPizza(1).EndRow = 1
' adiciona um gráfico à coleção
oGraficos.addNewByName(sNome, oRet, oExtPizza(), True, True)
' define as propriedades
oGraf = oGraficos.getByName(sNome).getEmbeddedObject()
oGraf.Diagram = oGraf.createInstance("com.sun.star.chart.PieDiagram")
oGraf.Diagram.DataRowSource = com.sun.star.chart.ChartDataRowSource.ROWS
oGraf.Title.String = "1º Trimestre – 2003"
```

Observe que temos um vetor de estruturas **CellRangeAddress**, onde o primeiro elemento contém os dados da legenda do gráfico e o segundo os dados a serem representados. Note também a mudança da propriedade **Diagram** com uma chamada ao método **createInstance ( )** e a mudança da propriedade **DataRowSource** do diagrama.

Note, ainda, que o fato de podermos definir um vetor CellRangeAddress, indica que podemos ter extensões de células, consecutivas ou não, contendo os dados a serem representados graficamente.

8 Mais informações

8.1 Na rede

OpenOffice.org :

http://www.openoffice.org

http://www.openoffice.org.br

Projeto de Documentação do OpenOffice.org:

http://documentation.openoffice.org

API do OpenOffice.org http://api.openoffice.org

Este é o local para quem está interessado em desenvolvimento com o OpenOffice.org. Além do SDK, você encontra:

- OpenOffice.org 1.0.2 – Developers Guide (~ 12 Mb)
- Manual de Referência da API OpenOffice.org (~ 8 Mb)
- StarOffice Programmers Tutorial (~ 1 Mb)
- Exemplos do Developers Guide (~ 1 Mb)

OOoDocs.Org http://www.ooodocs.org

Documentação e notícias sobre o OpenOffice.org.

OOExtras http://ooextras.sourceforge.net/

Repositório de modelos, macros e assistentes para o OpenOffice.org. Baixe todas as macros e estude o código fonte para aprender mais.

Pitonyak.org www.pitonyak.org/

Andrew Pitonyak e seus colaboradores merecem a nossa gratidão. Esta página contém um **excelente** documento sobre macros e sobre a linguagem Basic. Consulte-a periodicamente.

8.2 Com o autor

Aceito solicitações para correções, inclusão de novos tópicos, contribuições ou esclarecimentos sobre o OpenOffice.org Basic. Envie uma mensagem para:

noelsonduarte@globo.com

9 Créditos, Agradecimentos, Licença

9.1 Créditos

Autor do layout gráfico do modelo: Mirto Silvio Busico <m.busico@ieee.org>

Autor do texto explanatório do modelo: Gianluca Turconi <luctur@openoffice.org>

9.2 Agradecimentos

A Sun Microsystems, Inc pelo apoio para a criação e desenvolvimento do OpenOffice.org.

A Sun Microsystems, Inc, mais uma vez, pela disponibilização da documentação sobre a API do OpenOffice.org, sem a qual este trabalho não seria possível.

A todos os voluntários que, com seu trabalho, contribuem para o crescimento do OpenOffice.org.

9.3 Licença