Programando Macros Para O OpenOffice.org

Versão β_18_03_2006

Autor: Noelson Alves Duarte

Índice

Apresentação

REESCREVER!!

1 Trabalhando com Macros

1.1 Introdução

Uma macro é um programa escrito numa linguagem suportada pelo OpenOffice.org com a finalidade de automatizar tarefas do OpenOffice.org. Atualmente, as linguagens suportadas são OOoBasic, JavaScript, JavaBeans, Java e Python.

O nosso trabalho será dirigido para a linguagem OpenOffice.org Basic (**OOoBasic**).

1.2 Estrutura Geral das Macros

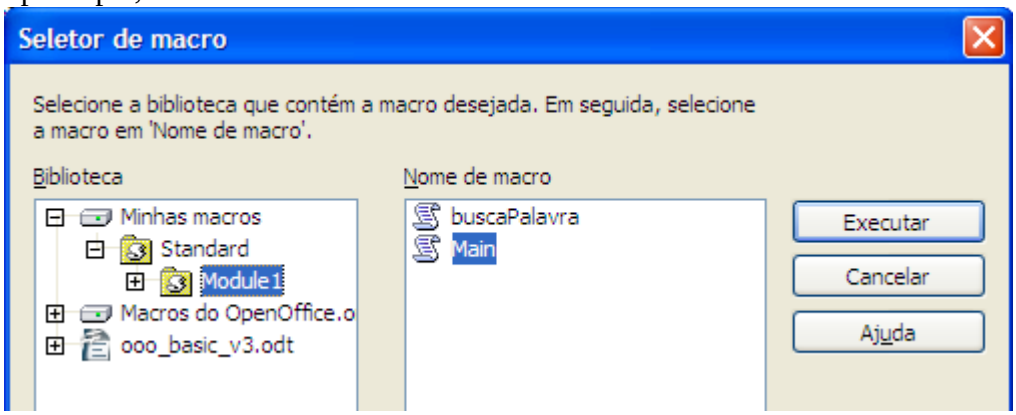
As macros do OpenOffice.org são organizadas num container do aplicativo ou do documento. As macros gravados num documento podem ser executadas apenas quando o documento estiver aberto. As macros do aplicativo podem ser executadas a qualquer momento.

As macros do aplicativo podem ser criadas para serem usadas por todos os usuários ou por apenas um usuário. No primeiro caso, elas são instaladas sob um diretório subordinado ao diretório de instalação do OpenOffice.org. No segundo, são instaladas num diretório subordinado ao diretório da instalação do OpenOffice.org para aquele usuário.

A macros do container documento são armazenadas dentro do próprio documento.

Cada um destes containeres pode ter uma ou mais bibliotecas. Uma biblioteca pode conter um ou mais módulos. Um módulo pode conter uma ou mais macros (rotinas) e, ainda, uma ou mais caixas de diálogo (desenhadas com o editor de diálogos do IDE Basic).

Para compreender esta estrutura, ative o diálogo **Selector de macro**, selecionando, na barra de menu principal, **Ferramentas => Macros => Executar Macro**.



Na figura acima, observe os containeres do aplicativo – **Minhas macros** e **Macros do OpenOffice.org** – o primeiro contém as macros do usuário e o segundo as macros compartilhadas por todos os usuários. Logo abaixo, temos o container **ooo_basic_v3.odt** de um documento aberto no OO.o.

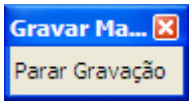
Note a biblioteca **Standard**, contendo o módulo **Module1** e, do lado direito, as macros **Main** e **buscaPalavra** existentes no Module1.

As bibliotecas Standard - do usuário ou do documento - são criadas automaticamente pelo OpenOffice.org. Ainda, o próprio OpenOffice.org gerencia, automaticamente, a carga destas bibliotecas para a memória.

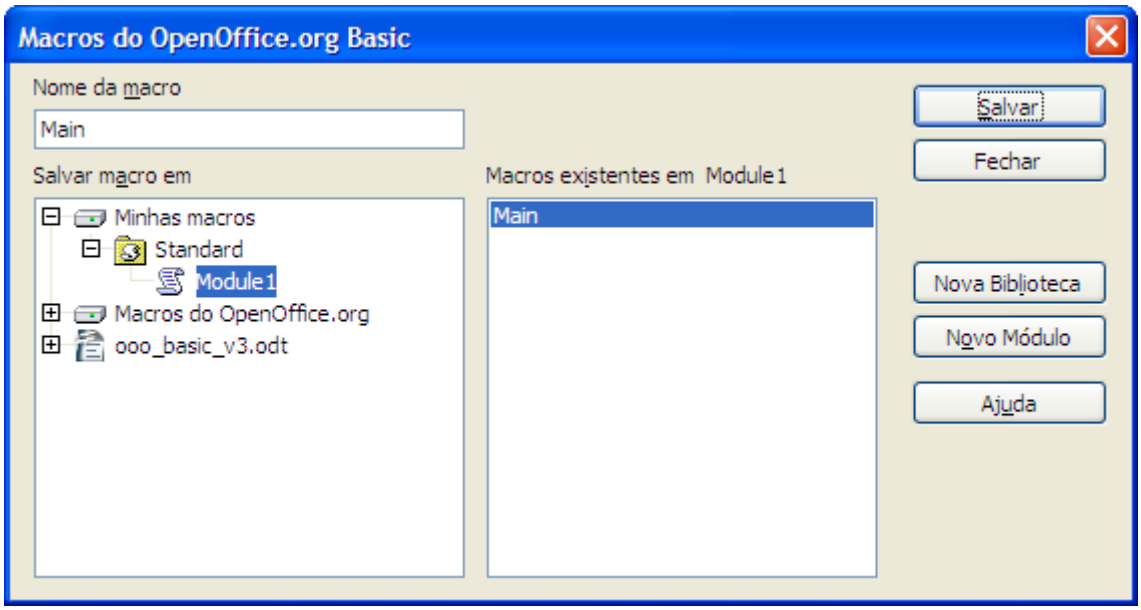
1.3 Gravando Macros

A partir da versão 1.1.x, o OO.o incluiu um gravador de macros que gera código OOoBasic. Esta ferramenta facilita a automação de tarefas simples por usuários que não são desenvolvedores. O gravador encadeia os comandos executados na interface gráfica e depois salva estes comandos como uma macro. Está disponível apenas para o Writer e Calc.

Crie um novo documento do Writer. Agora, suponha que você execute com frequência a busca da palavra **capítulo** num texto. Vamos criar uma macro para esta tarefa:



- Selecione **Ferramentas** => **Macros** => **Gravar Macro**;
- A ferramenta **Gravar Macro** torna-se ativa. A partir deste momento, todos os comandos serão memorizados para gravação;
- Selecione **Editar** => **Localizar e Substituir**, digite **capítulo** na caixa **Procurar por**, clique sobre o botão **Localizar** e, em seguida sobre o botão **Fechar**;
- Para encerrar a gravação, clique sobre **Parar Gravação** na ferramenta **Gravar Macro**.
- Surge o diálogo **Macros do OpenOffice.org Basic**. Selecione **Module1,** na biblioteca **Standard,** no container **Minhas Macros** e, em **Nome da Macro,** digite **buscaPalavra**;



- Para gravar a macro **buscaPalavra**, clique sobre o botão **Salvar**.
- Se o diálogo de criação de novo módulo for exibido, digite um nome para o módulo ou aceite o nome sugerido e clique sobre o botão **Ok**.
- Após a gravação, o diálogo **Macros do OpenOffice.org Basic** será fechado.

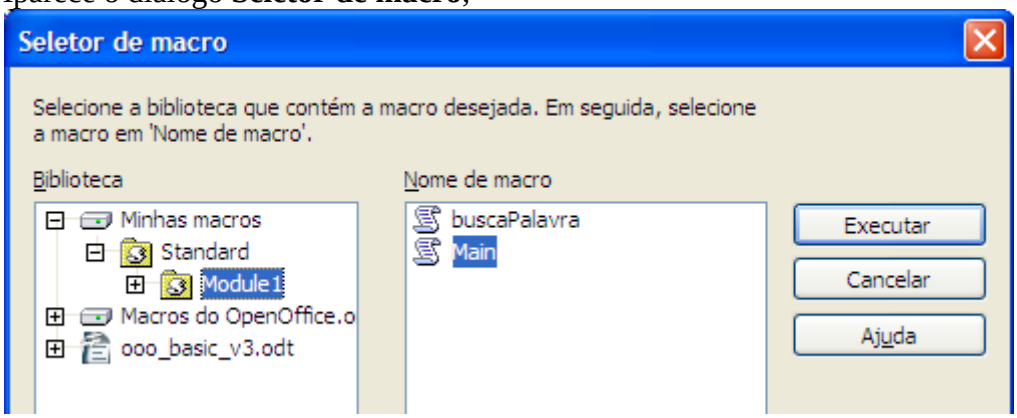
Antes de gravar a macro o usuário poderá criar uma biblioteca (botão **Nova Biblioteca**) e/ou um módulo (botão **Novo Módulo**). Mas, pela interface gráfica, uma nova biblioteca **não** pode ser criada no container **Macros do OpenOffice.org**, porque este container é protegido.

Tenha em mente que o gravador de macros é uma funcionalidade recente e alguns comandos da interface gráfica estão indisponíveis para gravação. Normalmente, estes comandos são gravados como comentários da linguagem OOoBasic.

1.4 Executando Macros

Vamos, agora, testar a macro **buscaPalavra**:

- Selecione **Ferramentas => Macros => Executar Macro**;
- Aparece o diálogo **Seletor de macro**;



- Expanda a entrada **Minhas macros** e selecione o **Module1**, da biblioteca **Standard**;
- Selecione a macro **buscaPalavra** e clique sobre o botão **Executar**.

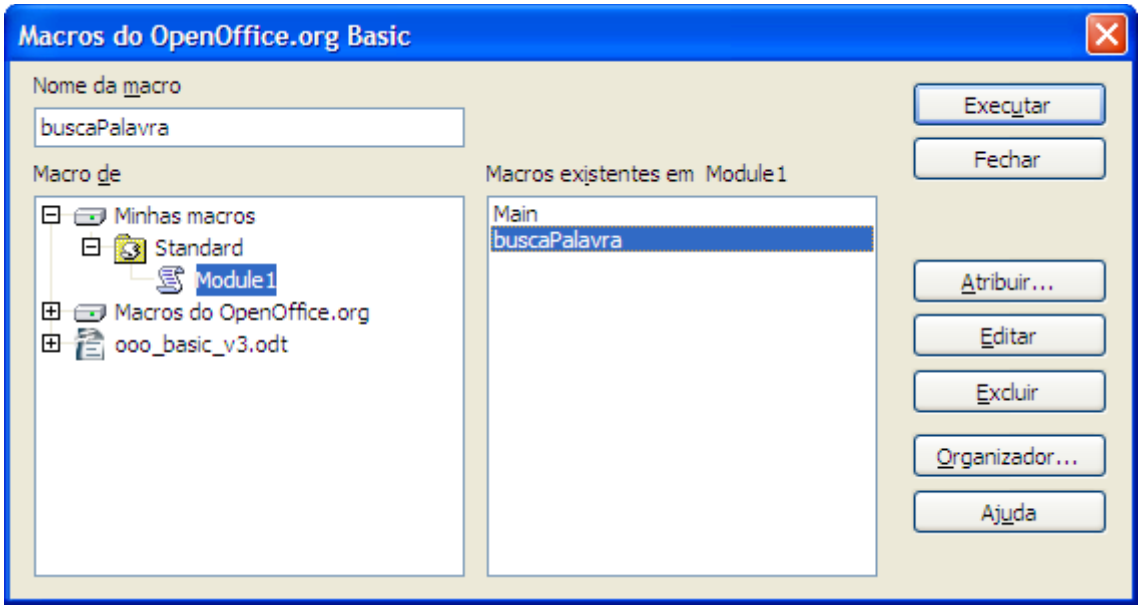
Este procedimento deve ser usado para rodar qualquer macro do OpenOffice.org. Contudo, podemos rodar macros a partir da linha de comando (útil para scripts). Veja alguns exemplos:

```
> soffice.exe “macro://nome_documento/Biblioteca.Modulo.Macro”
executa uma Macro num documento aberto
> soffice.exe “macro:///Standard.Modulo.Macro”
executa uma Macro num documento aberto e ativo
> soffice.exe url_do_documento “macro:///Standard.Modulo.Macro”
Abre um documento e executa uma macro
```

Existem, ainda, configurações do aplicativo que podem afetar a execução de uma macro. Selecione **Ferramentas => Opções**, expanda a entrada **OpenOffice.org**, selecione **Segurança** e clique sobre o botão **Segurança de macro** para verificar as restrições da sua instalação.

1.5 Organizando Macros

Através do diálogo **Macros do OpenOffice.org Basic**, podemos efetuar outras operações sobre macros. Para ativá-lo, selecione **Ferramentas => Macros => Organizar macros => OpenOffice.org Basic**.



As operações possíveis são:

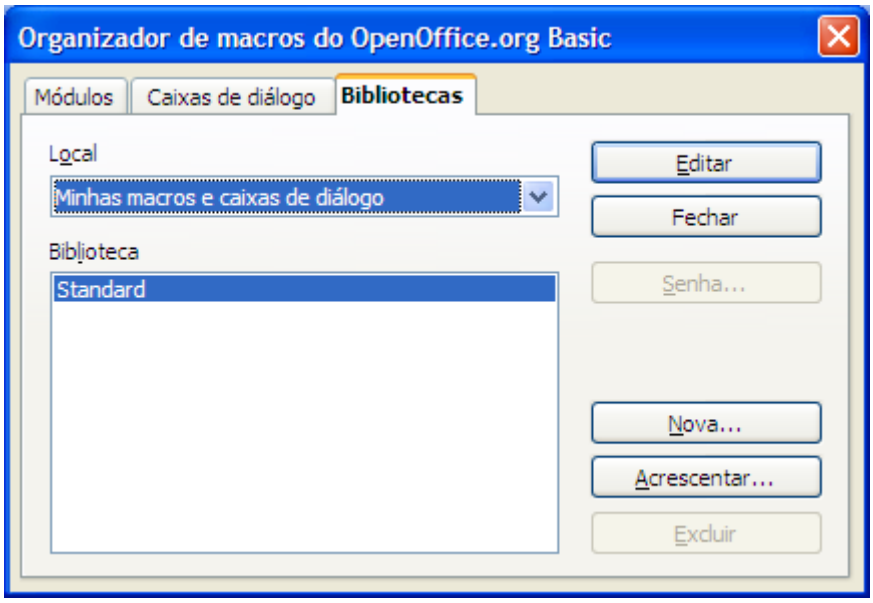
Botão Executar: outro modo para executar a macro selecionada. Aqui, apenas as macros do OOoBasic estão disponíveis. Com o diálogo **Seletor de macros**, podemos executar macros escritas em qualquer uma das linguagens suportadas pelo OpenOffice.org.

Botão Atribuir: atribui uma macro a um item de menu, uma combinação de teclas, um ícone na barra de ferramentas ou a um evento do documento ou do aplicativo. Esta operação será vista em detalhes noutro tópico.

Botão Editar: abre o editor do **IDE Basic** para a edição do código fonte da macro selecionada, com o cursor posicionado no início da mesma.

Botão Excluir: apaga a macro selecionada, após a confirmação do usuário.

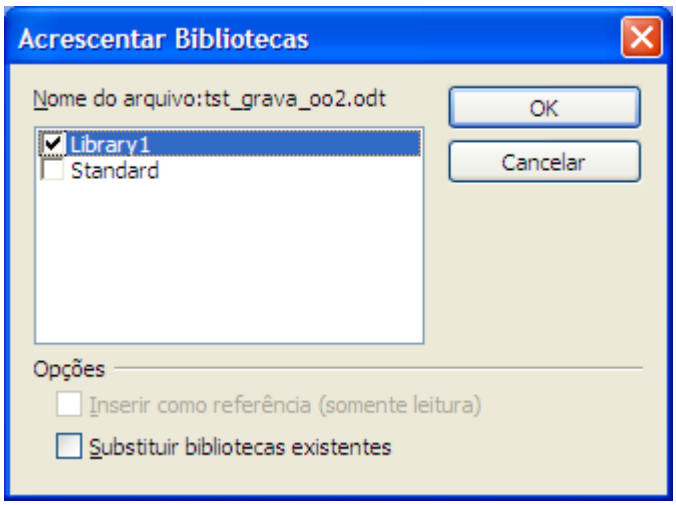
Botão Organizador: ativa o diálogo **Organizador de Macros do OpenOffice.org Basic**. Este diálogo permite operações sobre módulos, caixas de diálogos e bibliotecas. Clique sobre o botão para ativar o diálogo e vamos analisá-lo.



Na guia **Módulos** temos opções para criar e excluir módulos. Na guia **Caixas de diálogo** temos opções para criar e excluir diálogos. Estas tarefas também podem ser executadas diretamente no IDE Basic. Não existem opções para mover ou copiar módulos e caixas de diálogos entre bibliotecas. Mas, estas operações podem ser feitas, usando os recursos de "arrastar e soltar" dentro deste diálogo, nas guias correspondentes.

Na guia **Bibliotecas** podemos selecionar o container (na lista **Local**) e, a seguir, efetuar uma das operações abaixo:

- **Botão Editar**: abre o IDE Basic para a edição do código fonte ou das caixas de diálogos da biblioteca. Esta é a única operação disponível para as bibliotecas do container Macros do OpenOffice.org (vimos que ele é protegido);
- **Botão Senha**: protege uma biblioteca, exceto a Standard que **não** pode ser protegida. Após a proteção de uma biblioteca, qualquer operação sobre a mesma, somente será possível mediante a entrada da senha. Para executar uma macro protegida sem a entrada da senha, podemos associá-la a um evento ou chamá-la através de outra macro numa biblioteca não protegida;
- **Botão Nova**: cria uma biblioteca no container selecionado, após a definição do nome da nova biblioteca;
- **Botão Excluir**: exclui a biblioteca selecionada, após a confirmação do usuário. A biblioteca Standard **não** pode ser excluída;
- **Botão Acrescentar**: acrescenta uma biblioteca ao container selecionado na lista **Local**. Através desta opção, podemos instalar bibliotecas distribuídas em documentos do OpenOffice.org. Um clique neste botão comanda a abertura do diálogo de seleção de arquivos, para a escolha do documento contendo a biblioteca a ser acrescentada. Após a seleção do documento, o diálogo **Acrescentar Biblioteca** será exibido. Para acrescentar bibliotecas ao container **Macros e caixas de diálogo do OpenOffice.org**, consulte a seção Gerenciando Pacotes, neste manual.



Supondo que você deseja acrescentar uma biblioteca Library1 ao container selecionado na lista **Local**, marque a opção **Library1**, desmarque a opção **Standard** e clique sobre o botão **Ok**. Numa re-instalação de bibliotecas, marque também a opção **Substituir bibliotecas existentes**. A biblioteca Standard de um container qualquer **não** pode ser substituída. A opção **Inserir como referência**, quando ativa, acrescenta a biblioteca no modo somente leitura e deve ser usada para macros localizadas em pastas diferentes das padronizadas.

Ao acrescentar bibliotecas do container da aplicação ou de um diretório, no diálogo de seleção de arquivos, selecione o diretório contendo a macro. Você verá dois arquivos: script.xlb e dialog.xlb, selecione o arquivo **script.xlb** e comande Abrir. A seguir proceda como já explicado.

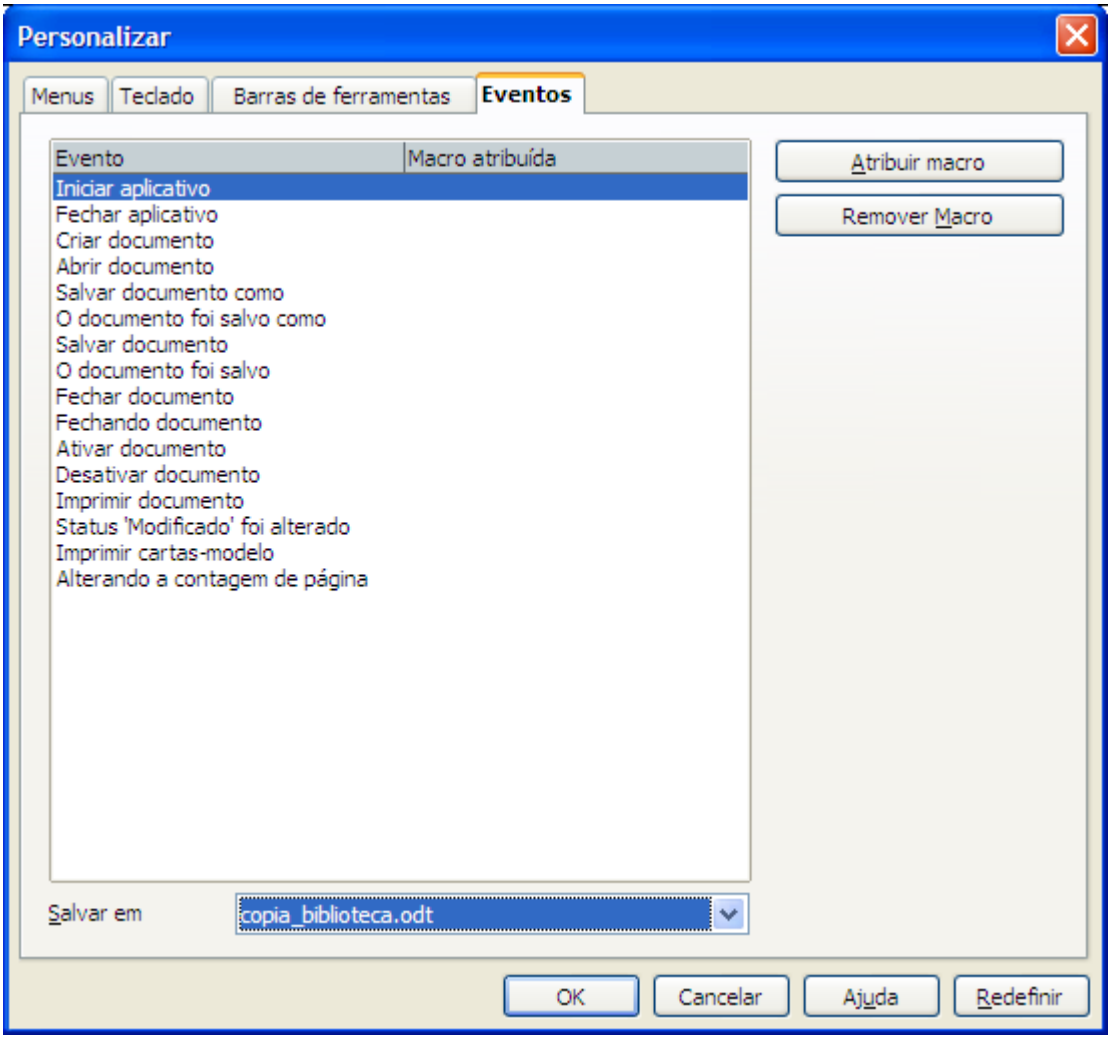
Finalmente, as bibliotecas do container **Macros e caixas de diálogo do OpenOffice.org** são gravadas como um subdiretório, com o nome igual ao da biblioteca, subordinado ao diretório <OOo_Install>/share/basic. As bibliotecas do container **Minhas macros e caixas de diálogo** são gravadas, também, em subdiretórios do diretório <OOo_User_Install>/user/basic.

1.6 Atribuindo Macros a Eventos

Um evento ocorre a partir de uma ação do usuário sobre a interface gráfica. As macros do OpenOffice.org podem ser associadas a eventos do aplicativo ou do documento. Além destes eventos predefinidos, podemos atribuir uma macro a uma combinação de teclas, a um ícone na barra de ferramentas ou a uma entrada na barra de menu. Após a atribuição, sempre que o evento ocorrer a macro será executada.

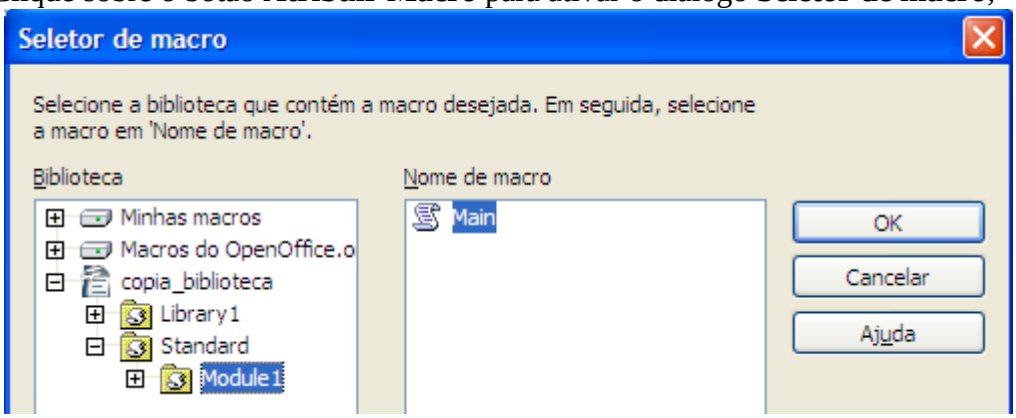
Um evento do aplicativo ocorre para todo o OpenOffice.org e um evento do documento ocorre somente no documento. Por exemplo, o evento Fechar Documento está disponível para o aplicativo e para um documento. Uma macro atribuída a este evento do aplicativo será executada quando qualquer documento do OpenOffice.org for fechado, do contrário (evento do documento) a macro será executada apenas quando o documento a que se refere for fechado.

A associação de macros a eventos pode ser executada a partir do botão **Atribuir** no diálogo **Macros do OpenOffice.org Basic** ou selecionando **Ferramentas => Personalizar** na barra de menus. Após o clique sobre o botão **Atribuir**, o diálogo **Personalizar** será exibido.



Para atribuir uma macro a um evento:

- Clique sobre a guia **Eventos** para ver a relação de eventos disponíveis;
- Escolha o tipo do evento (documento/aplicativo) na caixa de listagem **Salvar em**;
- Selecione o evento desejado na relação de eventos;
- Clique sobre o botão **Atribuir Macro** para ativar o diálogo **Seletor de macro**;



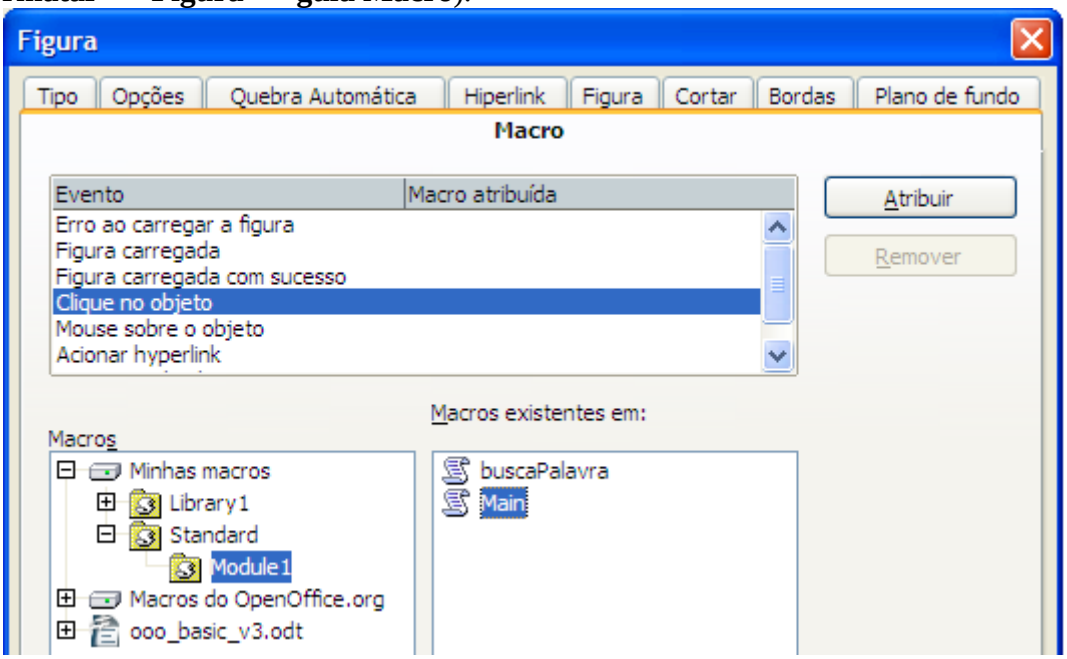
- Selecione o módulo e a macro e clique sobre o botão **OK**. Note que o URL da macro será acrescentada na coluna **Macro atribuída**, ao lado do evento selecionado.

– Clique sobre o botão **OK** no diálogo **Personalizar**.

Para remover uma atribuição de macro a um evento, selecione o evento e clique sobre o botão **Remover Macro**, no diálogo Personalizar.

Neste mesmo diálogo podemos definir outras maneiras para rodar uma macro. Na guia **Menus** é possível acrescentar um item de menu, na guia **Teclado** pode-se definir uma combinação de teclas e na guia **Barra de Ferramentas** um ícone para comandar a execução da macro.

Além dos eventos já mencionados, alguns objetos do OpenOffice.org como Controles, Gráficos, Quadros e Hiperlinks, possuem eventos que podem ser associados a macros. Abaixo vemos alguns eventos disponíveis para o objeto **Figura** (clique sobre uma figura e selecione **Formatar => Figura => guia Macro**).



Consulte a ajuda do OpenOffice.org para uma relação dos objetos, eventos e comandos para fazer a atribuição de macros.

Outro modo comum de executar macros é através de um evento de um controle de formulário num documento, por exemplo, um clique sobre um **Botão de Ação**. Esta operação será apresentada adiante, no capítulo *Formulários*.

Finalmente, no Writer, em **Inserir => Campos => Outros**, na guia **Função**, existe o campo **Executar macro**, que dispara a macro associada ao campo toda vez que o usuário clicar sobre o mesmo.

1.7 Gerenciando Pacotes

Vimos como instalar módulos e bibliotecas usando o Gerenciador de Macros. Contudo, este método tem limitações. Por exemplo, um desenvolvedor que deseja distribuir um **suplemento** (Addon), que instale suas próprias configurações (dados, interface gráfica, bibliotecas, etc) não pode usar o Gerenciador de Macros.

O OpenOffice.org usa o conceito de **pacote** para facilitar a distribuição de componentes. Um **pacote** é um arquivo no formato **zip**, contendo todos os arquivos do componente. Para a versão 2.0, os pacotes devem ter o nome na forma: ***.uno.pkg** (por exemplo: meuSuplemento.uno.pkg) e, além dos arquivos do componente, ele deve conter uma pasta **META-INF** com um arquivo **manifest.xml**.

Num ambiente do OpenOffice.org, um pacote pode ser instalado para um usuário ou para todos os usuários (rede) ou, ainda, para uma estação de trabalho.

O gerenciador de pacotes, para o OO.o 1.x.x, é o programa **pkgchk.exe** um utilitário de linha de comando, que pode ser encontrado no diretório **<ooo_install>/program**. Antes de usá-lo, feche todos os processos do OpenOffice, inclusive o início rápido. Para ver as suas opções, no "shell" do seu sistema digite:

```
<ooo_install>\program\pkgchk --help
```

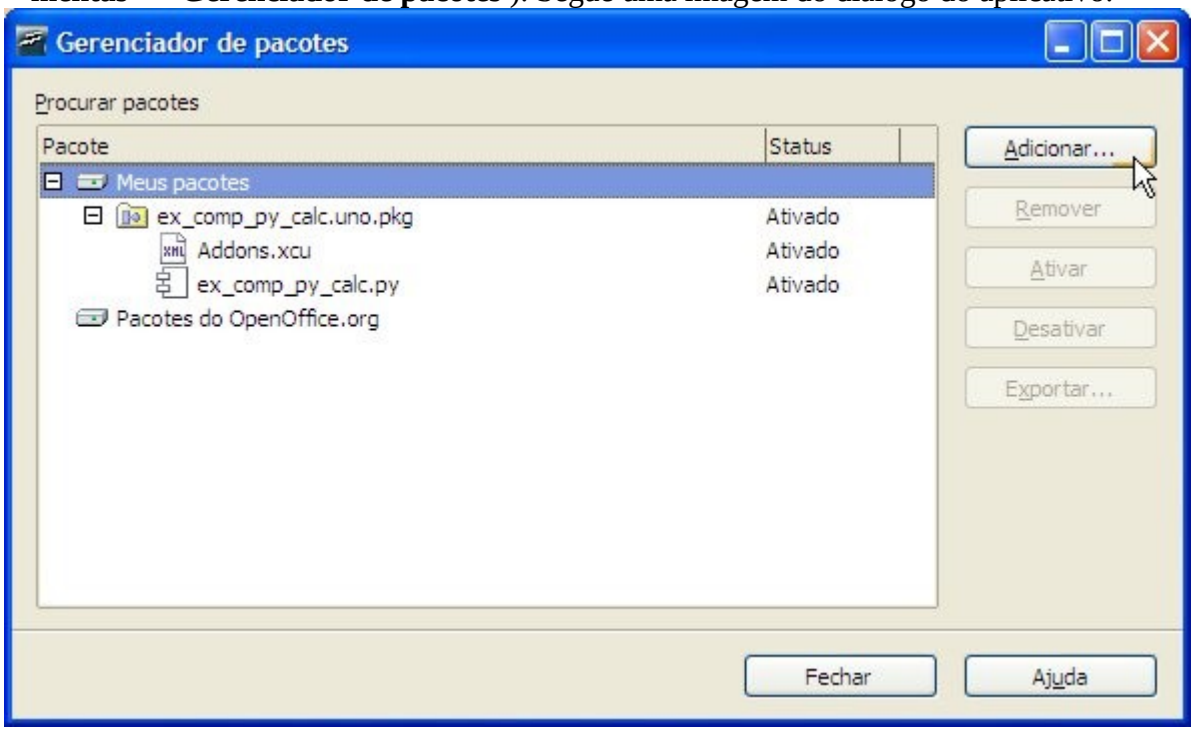
A versão 2.0 do OpenOffice.org traz um novo gerenciador de pacotes, o programa **unopkg.exe**, também localizado no diretório **<ooo_install>/program.** Evite usar o pkgchk nesta versão, pois o unopkg reconhece os pacotes das versões anteriores. Para ver as suas opções, digite na linha de comando do sistema:

```
<ooo_install>\program\unopkg --help
```

ou, para ativar a sua interface gráfica:

```
<ooo_install>\program\unopkg gui
```

A interface gráfica pode ser ativada, também, através do menu do OpenOffice.org (**Ferramentas => Gerenciador de pacotes**). Segue uma imagem do diálogo do aplicativo.



Note que existem dois containeres: **Meus pacotes** e **Pacotes do OpenOffice.org**. O primeiro contém os pacotes instalados para o usuário e o segundo os pacotes compartilhados por todos

os usuários. Não podemos instalar pacotes do segundo tipo a partir do OpenOffice.org, para isto o usuário deve usar a linha de comando e ter direitos de escrita na pasta <ooo_install>.

Eis as opções de gerenciamento:

Botão Adicionar : permite acrescentar um pacote ao ambiente do OpenOffice.org. Um clique neste botão, abre o diálogo **Adicionar Pacote**, para a seleção do pacote a ser instalado.

Botão Remover : remove o pacote selecionado.

Botão Ativar : ativa o pacote selecionado.

Botão Desativar : desativa o pacote selecionado.

Botão Exportar : exporta um pacote no formato do pkgchk para o novo formato (acrescenta a pasta META-INF com o arquivo manifest.xml). A extensão do nome do arquivo deve ser alterada manualmente.

O **unopkg** pode ser utilizado para instalar macros de qualquer linguagem, inclusive as macros que devem ser compartilhadas por todos os usuários (container Macros do OpenOffice.org). Para tal, você deve criar um arquivo **zip** com todos os arquivos da sua macro, ter os direitos de escrita no diretório <ooo_install> e usar o gerenciador a partir da linha de comando.

2 Linguagem OpenOffice.org Basic

A linguagem **BASIC** (**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) foi criada no ano de 1963, pelos matemáticos John George Kemeny e Tom Kurtzas, no Dartmouth College. Desde então, pela facilidade de uso e aplicação geral, tornou-se uma linguagem de programação de computadores muito popular, em todo o mundo.

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.org.

2.1 Componentes da linguagem

Os principais componentes da linguagem são:

Linguagem Basic: Define os construtores básicos da linguagem, os operadores e tipos de dados suportados e as regras de combinação dos elementos.

Biblioteca de Funções do Basic (RTL Basic): Contém diversas funções predefinidas e compiladas juntamente com o interpretador OOoBasic.

API do OpenOffice.org: Permite o acesso aos objetos UNO do OpenOffice.org.

IDE Basic: Ambiente de desenvolvimento para a edição e depuração de código fonte e para o desenho de caixas de diálogos.

Ajuda do OOoBasic: Ajuda para todas as características da linguagem (funções, comandos, tipos de dados, operadores, etc). Entretanto, não oferece ajuda para os objetos da API.

Eis algumas **limitações** da linguagem: Não permite a criação de objetos, não há suporte para linhas de execução (threads) e não permite a escrita de componentes UNO (exceto listeners).

2.2 Ambiente de Desenvolvimento

O OpenOffice.org possue um Ambiente de Desenvolvimento Integrado (IDE Basic) que oferece as funções básicas para a programação de macros OOoBasic.

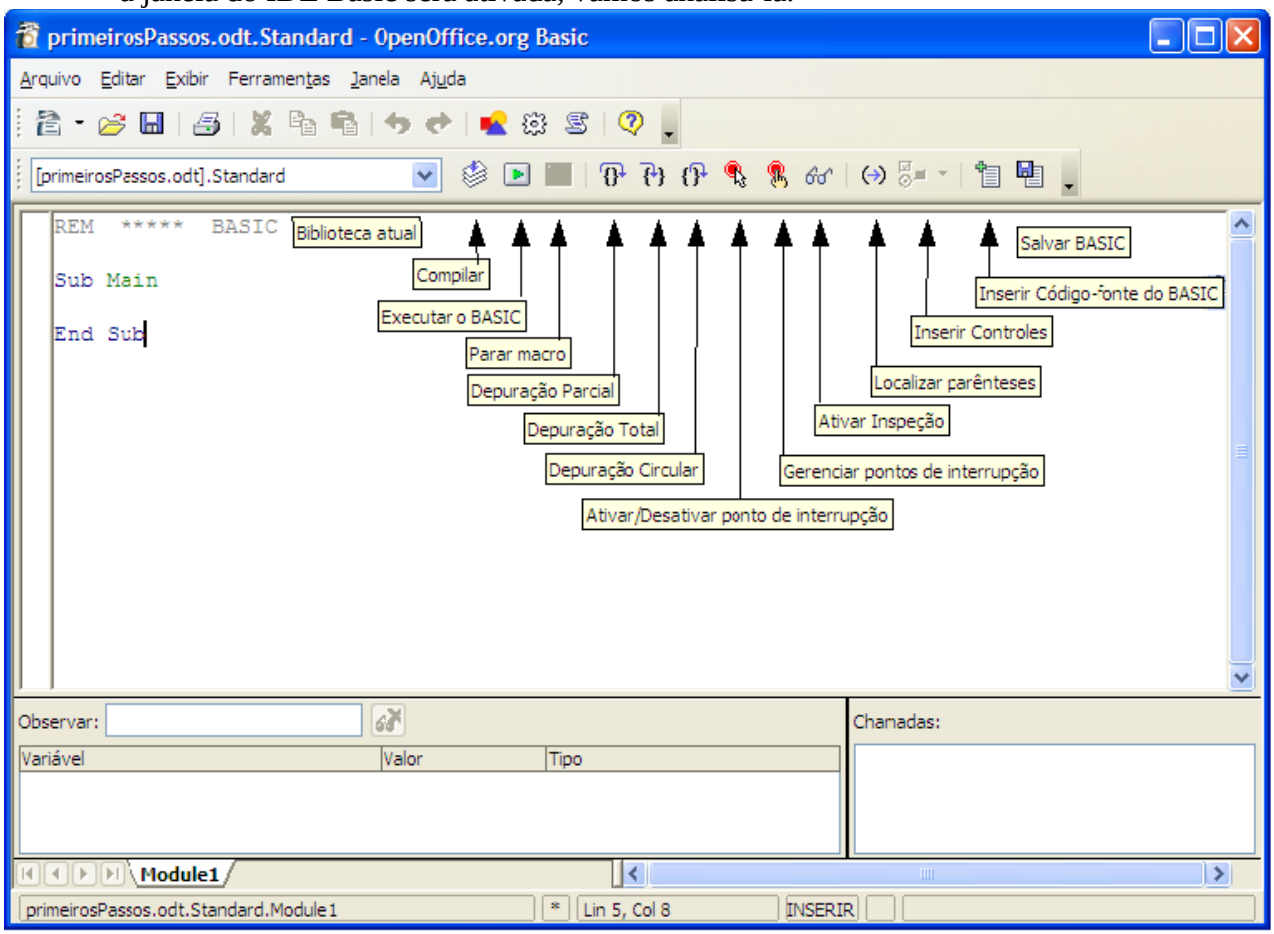
Entre as suas funcionalidades cabe ressaltar:

- Editor de texto para a edição do código fonte;
- Funções básicas de acompanhamento da execução da macro;
- Funções básicas para a depuração do código fonte;
- Editor de Caixas de Diálogo para a criação e testes de Diálogos personalizados.

Vamos analisar o IDE Basic. Crie um documento do Writer e salve-o como **primeirosPassos,** em seguida:

- selecione **Ferramentas=>Macros=>Organizar Macros=>OpenOffice.org Basic**;
- no container **primeirosPassos.odt**, selecione a biblioteca **Standard**;

- clique sobre o botão **Novo** para criar um módulo na biblioteca Standard, aceite o nome **Module1** e clique sobre **OK**;
- a janela do IDE Basic será ativada, vamos analisá-la.



Note que, ao iniciar um novo módulo, o IDE acrescenta uma linha de código comentada e uma macro Main vazia (linhas: **Sub Main** e **End Sub**).

Temos, na parte superior do IDE, a barra de menus e a barra de ferramentas com os ícones. Na parte central, temos o editor de texto e, à esquerda deste, a barra de indicação. Logo abaixo do editor, no lado esquerdo, vem a janela de inspeção e, no lado direito, a janela de chamadas. Mais abaixo temos a barra de módulos e, por fim, a barra de estado.

Na barra de ferramentas, além dos ícones já identificados, temos ainda o Catálogo de objetos, Selecionar Macros e Selecionar Módulo. Apontando o cursor do mouse para um ícone, uma pequena descrição sobre o mesmo será exibida. Vejamos a funcionalidade dos principais ícones:

Compilar: faz uma verificação da sintaxe do código fonte do módulo corrente, apenas.

Executar: verifica a sintaxe do código fonte de todos os módulos da biblioteca corrente e, se nenhum erro for encontrado, inicia a execução da primeira macro do módulo corrente. Em caso de erro, uma caixa de diálogo, com informações sobre o mesmo, será exibida. Durante a execução da macro algum erro (run-time) pode ocorrer. Para uma relação dos códigos de erros "run-time", consulte a Ajuda do OOoBasic (no IDE tecle F1).

Parar macro: finaliza a execução da macro. Útil, por exemplo, para interromper um "loop" infinito.

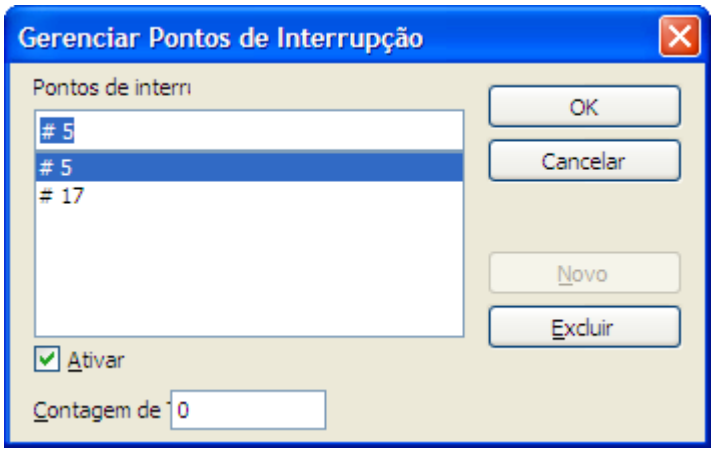
Depuração Parcial: executa uma macro comando a comando, as chamadas de rotinas são tratadas como um só comando. Durante a depuração, uma **seta amarela**, na **barra de indicação**, aponta para a linha a ser executada no próximo passo.

Depuração Total: acompanha a execução de todos os comandos da macro, inclusive os comandos das rotinas chamadas a partir da macro.

Depuração Circular: executa a macro sem acompanhar nenhum comando. Use para sair de uma depuração total e retornar a uma depuração parcial.

Ativar/Desativar ponto de interrupção: Um ponto de interrupção é um local (linha) onde a execução da macro será temporariamente interrompida. Para ativar ou desativar um ponto de interrupção, posicione o cursor na linha desejada e clique sobre este ícone. Após a definição, uma **marca circular vermelha** surge na **barra de indicação** lateral.

Gerenciar pontos de interrupção: Exibe um diálogo para incluir ou excluir pontos de interrupção. Os números das linhas com pontos de interrupcão são exibidos. O campo **Contagem de Passagens** indica que a macro será interrompida após a enésima passagem pelo ponto de interrupção, isto é desejável porque, às vezes, uma linha de código só apresenta erros quando executada mais de uma vez.



Ativar inspeção: Permite o acompanhamento, na janela de inspeção, do valor de uma variável. Para ativar uma inspeção, selecione a variável no código fonte e clique sobre este ícone. Após a definição, uma entrada com o nome da variável será acrescentada na **janela de inspeção** e o ícone **Remover Inspeção**, nesta janela, será ativado.

Localizar parênteses: Seleciona um trecho de código entre parênteses.

Inserir Controles: Ativo apenas no editor de diálogos, contém os controles das caixas de diálogo, além de outras operações relacionadas a diálogos.

Inserir código fonte do BASIC: Permite a seleção de um arquivo ASCII, contendo código fonte BASIC, para ser inserido no módulo corrente.

Salvar BASIC: Exporta o código fonte do módulo corrente para um arquivo ASCII.

Catálogo do objetos: Apresenta um navegador para os objetos do OOoBasic.

Selecionar macro: Ativa o diálogo Macros do OpenOffice.org Basic.

Selecionar módulo: Ativa o diálogo Organizador de Macros do OpenOffice.org Basic.

Na barra de indicação acompanhamos a execução da macro, os pontos de interrupção e as linhas que provocam algum erro.

Na janela de inspeção verificamos o valor e as propriedades das variáveis definidas na caixa Observar. Também, durante a execução, podemos posicionar o cursor do mouse sobre uma variável para verificar o seu conteúdo.

A janela de chamadas exibe a situação da pilha de rotinas durante a execução da macro, ou seja, a hierarquia das rotinas que estão sendo executadas no momento.

A barra de módulos deve ser usada para navegar entre módulos e caixas de diálogo.

Para se familiarizar com o IDE Basic, digite o código a seguir na macro Main, de modo que ela fique com esta aparência:

```
Sub Main
   nome = InputBox ("Digite o seu nome:")
   If (nome = "") Then
      MsgBox "Olá usuário!"
   Else
      MsgBox "Olá " & nome
   End If
End Sub
```

- Execute a macro, clicando sucessivamente sobre o ícone Depuração Parcial, note a seta amarela na barra de indicação.
- Defina um ponto de interrupção na linha **If (nome = "") Then**.
- Selecione a variável **nome** na segunda linha e clique sobre Ativar Inspeção. Observe a janela de inspeção.
- Execute a macro usando o ícone Depuração Total, observe as mudanças nas janelas de inspeção e de chamadas.
- Remova o ponto de interrupção e a Inspeção da variável nome.
- Execute a macro clicando sobre o ícone Executar.
- Verifique a funcionalidade do catálogo de objetos, do seletor de macros e de módulos.

Estes passos são suficientes para uma introdução ao IDE Basic. Naturalmente, ao escrever e depurar suas próprias macros, você aprenderá mais.

Deste ponto em diante, deixo por sua conta a criação de documentos para testar o código fonte. Sugiro que você crie documentos (Writer ou Calc) para cada capítulo, como, por exemplo, **macros_cap2.odt** para os exemplos deste capítulo. De modo geral, apenas os exemplos contendo as linhas **Sub** <nome> ... **End Sub** podem ser executados. Os outros trechos de código explicam algum conceito ou comando da linguagem.

2.3 Análise de um programa

Para explicar algumas características do OOo Basic, vamos apresentar e analisar, passo a passo, um exemplo simples de programa.

Crie um documento do Writer e salve-o como **exemplosBasic,** em seguida:

- selecione **Ferramentas=>Macros=>Organizar Macros=>OpenOffice.org Basic**;
- no container **exemplosBasic.odt**, selecione a biblioteca **Standard**;
- clique sobre o botão **Novo** para criar um módulo na biblioteca Standard, aceite o nome **Module1** e clique sobre **OK**;

A nova janela que surgiu é o editor Basic, parte do IDE Basic do OpenOffice.org. Esta janela, na sua parte central, contém as linhas:

```
REM ***** BASIC *****

Sub Main

End Sub
```

Se você não chegou neste ponto, consulte a seção *Ambiente de Desenvolvimento*, desta *Apostila*. Lá, existem informações detalhadas sobre o IDE Basic.

Vamos ao nosso exemplo. Ele solicita o nome do operador, obtém a hora do sistema e faz uma saudação apropriada. Digite o código fonte abaixo, **entre** as linhas Sub Main e End Sub:

```
Dim sNome As String    ' variável para guardar o nome
Dim sSauda As String   ' variável para guardar a saudação
Dim sHora As Integer   ' variável para guardar a hora do sistema

sHora = Hour ( Now )
sNome = InputBox ( "Digite o seu Nome:", "Caixa de entrada", "Operador" )
If ( sHora > 6 And sHora < 18 ) Then
   sSauda = "Bom Dia! "
Else
   sSauda = "Boa Noite! "
End If
MsgBox sSauda + sNome
```

Você acabou de criar uma macro chamada **Main**. Execute a macro, surge uma caixa de entrada, solicitando o seu nome, digite-o e clique sobre o botão **Ok**. Surgirá um diálogo com uma saudação, clique em **Ok** para encerrar a execução da macro. Se ocorrer algum erro, revise o código digitado e recomece.

Vamos analisar, passo a passo, o código fonte:

```
REM ***** BASIC *****
```

Esta linha contém um **comentário**. Comentários não são executados.

```
Sub Main
```

Esta linha **declara e define** a **sub-rotina** Main e deve estar acompanhada de uma linha End Sub. Entre as duas linhas deve vir o código fonte que dá funcionalidade a sub-rotina.

```
Dim sNome As String   ' variável para guardar o nome
Dim sSauda As String  ' variável para guardar a saudação
Dim sHora As Integer  ' variável para guardar a hora do sistema
```

Estas linhas **declaram e inicializam** as **variáveis** que serão usadas na macro. O **apóstrofe** marca o início de **comentários** numa linha de código.

```
sHora = Hour ( Now )
```

Aqui, chamamos duas **funções** internas do Basic. A função **Now**, obtém a hora completa do sistema (hh:mm:ss). A função **Hour** recebe o valor retornado por Now e extrai somente a hora. Em seguida, o valor obtido por Hour (a hora) é armazenado na variável sHora. Neste contexto, o sinal de igual (=) é chamado **operador de atribuição**. Tudo que estiver do lado direito do operador de atribuição recebe o nome de **expressão**, neste caso Hour (Now).

```
sNome = InputBox ( "Digite o seu Nome:", "Caixa de entrada", "Operador" )
```

Esta linha chama a função **InputBox**, do Basic, que apresenta a caixa de diálogo para lidar com a entrada do nome do operador. Note que passamos três cadeias de caracteres para esta função. Estes valores (as cadeias) são chamados de **parâmetros da função**. O primeiro parâmetro é uma mensagem para o operador, o segundo é o título do diálogo e o terceiro é o valor padrão da caixa de texto. A função InputBox retorna o conteúdo da caixa de texto se você clicar sobre o botão Ok, senão (clique sobre o botão Cancelar ou Fechar) retorna uma cadeia vazia. Após encerrar o diálogo, o valor retornado será atribuído à variável sNome.

Ainda, sobre parâmetros, podemos dizer que a função Now não têm parâmetro e que a função Hour recebe um parâmetro.

```
If ( sHora > 6 And sHora < 18 ) Then
```

Aqui, iniciamos a execução de um **comando de decisão**. Estamos dizendo ao Basic: **se** o valor armazenado na variável **sHora for maior que 6 e menor que 18 então** execute o próximo bloco de código. Os sinais > e < são chamados **operadores relacionais**. A palavra **And** é chamada de **operador lógico**. O conteúdo entre parênteses é uma **expressão composta** que será **avaliada** como Verdadeira (True) ou Falsa (False). Ela será verdadeira, se ambas as expressões (sHora > 6; sHora < 18) forem verdadeiras.

```
    sSauda = "Bom Dia! "
```

A **expressão** “Bom Dia!” será atribuída a sSauda, se a expressão da linha anterior for avaliada como verdadeira (True).

```
Else
```

Senão, a expressão foi avaliada como falsa (False), execute o próximo bloco de código.

```
    sSauda = "Boa Noite! "
```

A expressão “Boa Noite! ” é atribuída à variável sSauda.

```
End If
```

Esta linha indica o final do comando de decisão IF ... THEN ... ELSE ... END IF.

```
MsgBox sSauda + sNome
```

Esta linha chama a sub-rotina MsgBox, do Basic, que exibe uma caixa de mensagem para o operador. Ela recebe um parâmetro, neste caso a expressão sSauda + sNome. O sinal (+), neste contexto, é chamado de **operador de concatenação**, pois junta as cadeias de caracteres guardadas nas variáveis sSauda e sNome. Clique sobre o botão OK para fechar a caixa de mensagem.

```
End Sub
```

Esta linha indica o término da sub-rotina, iniciada na linha SUB Main. Não esqueça, uma linha SUB ... precisa estar associada a uma linha END SUB.

Neste tópico, você aprendeu diversos conceitos (estão em negrito) relacionados com a linguagem Basic. Se algum ponto não foi compreendido, execute novamente o exemplo (use o ícone **Depuração Parcial**) e leia o passo a passo, tentando associar o que ocorreu durante a execução com as linhas de código.

2.4 Elementos do Basic

Desta seção até o Capítulo 4 inclusive, veremos superficialmente os principais elementos da linguagem Basic. Lembro que a Ajuda do OOoBasic abrange todos os aspectos da linguagem e contém exemplos simples para todas as instruções. Para obter ajuda, basta pressionar a tecla F1, no IDE Basic.

Palavras reservadas

Uma palavra reservada é um identificador (nome) utilizado internamente pela linguagem Basic. As palavras reservadas do Basic são aquelas usadas nos seus comandos, nas suas funções internas, nas suas constantes e nos seus operadores. Abaixo, alguns exemplos:

```
BEEP    - comando do Basic
CALL    - comando do Basic
SUB     - comando do Basic
FOR     - parte de um comando do Basic
NEXT    - parte de um comando do Basic
EXIT    - parte de comandos do Basic
WHILE   - parte de comandos do Basic
DO      - parte de comandos do Basic
STR$    - função do Basic
TRIM$   - função do Basic
SQR     - função do Basic
AND     - operador do Basic
OR      - operador do Basic
PI      - constante definida pelo Basic
```

O principal sobre estas palavras é: elas não podem ser usadas como nomes para identificar variáveis, constantes ou procedimentos definidos pelo programador.

Regras para nomes

Devemos observar as seguintes regras para os nomes das nossa variáveis, constantes, sub-rotinas e funções:

```
Nomes são compostos por letras (A a Z), dígitos (0 a 9) e sublinhado (_);
Caracteres especiais e de pontuação não são permitidos ( ex: letras acentuadas, vírgula );
Nomes devem começar por uma letra do alfabeto;
Nomes não podem conter mais de 255 caracteres;
Nomes não diferenciam entre letras maiúsculas e minúsculas;
Nomes contendo espaço são permitidos, mas devem estar entre colchetes;
Palavras reservadas não podem ser usadas como nomes definidos pelo programador.
```

Exemplos:

```
Nomes válidos: ptoIni, ponto2, flag, minha_Funcao, [Minha Variavel]
Nomes inválidos: 2_ponto, minha?Variável, ºcelsius, Retângulo, BEEP
```

Comentários

Numa macro, as linhas iniciadas por apóstrofe ou pela palavra reservada REM (de REMarker) são consideradas como comentários, portanto não são processadas.

```
' Isto é um comentário
REM O código abaixo inicializa as variáveis de ambiente
```

Os comentários (com apóstrofe) podem ser colocados na frente de uma linha de código.

```
DIM coord_X As Double   ' variável para guardar a coordenada X do ponto
DIM coord_Y As Double   ' variável para guardar a coordenada Y do ponto
```

Tipos de dados internos

Inteiros (INTEGER)
Um inteiro pode conter um valor na faixa de -32768 a 32767.

Inteiros Longos (LONG)
Um inteiro longo pode conter um valor na faixa de -2.147.483.648 a 2.147.483.647.

Ponto flutuante simples (SINGLE)
Valores de precisão simples podem variar de 3.402823E38 a 1.401298E-45, para números positivos ou negativos.

Ponto flutuante duplo (DOUBLE)
Os valores de dupla precisão variam na faixa de 1.797693134862315E308 a 4.94066E-324, para valores negativos ou positivos.

Cadeias de caracteres (STRING)
Uma cadeia pode conter até 64000 caracteres (64 Kb).

Moeda (CURRENCY)
Os valores deste tipo variam de -922.337.203.685.477,5808 até 922.337.203.685.477,5807.

Booleano (BOOLEAN)
Os valores deste tipo podem ser True (verdadeiro) ou False (falso)

Data (DATE)
Podem conter valores de data e tempo armazenados num formato interno.

Variante (VARIANT)
Podem conter qualquer tipo de dado

Declaração de variáveis

Podemos declarar variáveis explicitamente com a palavra reservada DIM, do modo abaixo:

DIM nome_da_variavel AS tipo_de_dado
Ou:

DIM nome_da_variavel seguido do caractere indicador do tipo de dado

nome_da_variavel é um nome definido pelo programador, conforme as regras de nomeação.

Exemplos:

```
DIM varInteira AS INTEGER   ' Declara uma variavel inteira
DIM varInteira%             ' Declara uma variavel inteira (%)
DIM varIntLongo AS LONG     ' Declara uma variavel longa
DIM varIntLongo&            ' Declara uma variavel longa (&)
DIM varString AS STRING     ' Declara uma variavel string
DIM varString$              ' Declara uma variavel string ($)
DIM varSingle AS SINGLE     ' Declara uma variavel real simples
DIM varSingle!              ' Declara uma variavel real simples (!)
DIM varDupla AS DOUBLE      ' Declara uma variavel real dupla
DIM varDupla#               ' Declara uma variavel real dupla(#)
DIM varPreco AS CURRENCY    ' Declara uma variavel moeda
DIM varPreco@               ' Declara uma variavel moeda (@)
```

Podemos declarar uma variável, no momento do seu emprego, acrescentando o caractere de tipo ao nome da variável, como nos exemplos abaixo:

```
UmInteiro% = 1500                    ' declara um inteiro
UmLongo& = 2195678                   ' declara um inteiro longo
MeuNome$ = ´´JOSÉ DE ANDRADE´´       ' declara uma cadeia
UmRealSimples! = 15.555              ' declara um real simples
UmRealDuplo# = coordEne * sin(30)    ' declara um real duplo
```

Se o tipo de dado não for indicado a variável será assumida como Variant.

O valor de uma variável declarada será 0 para tipos numéricos, uma cadeia nula para Strings e False para Boolean.

É recomendável a declaração explícita das variáveis de um programa, bem como a escolha de nomes significativos para as mesmas. Para forçar a declaração de todas as variáveis, use a linha de comando abaixo, no início do módulo:

```
OPTION EXPLICIT
```

O OOoBasic não nos obriga a declarar explicitamente uma variável. Isto significa que, apesar de não recomendado, podemos criar uma variável sem a declaração, nestes casos o tipo de dado será assumido durante a operação de atribuição.

Constantes simbólicas

Para facilitar a leitura de uma macro, os valores literais (1000, “S”, 2.718), usados ao longo da mesma, devem ser declarados como constantes simbólicas, com o comando CONST, fora de qualquer sub-rotina ou função, assim:

```
' declaração de constantes
Const NR_PONTOS% = 1000       ' número de pontos = 1000
Const SIM$ = "S"              ' constante SIM = "S"
Const BASE_NATURAL# = 2.718   ' base log. naturais
```

Posteriormente, podemos usar estas constantes numa expressão, no lugar dos seus valores literais, por exemplo:

```
DIM coordX (NR_PONTOS) As Double    ' declara vetor com 1001 elementos
```

```
valor# = valor1 * BASE_NATURAL      ' bem mais compreensível!

If (Resposta$ = SIM) Then
   ' se resposta = "S" então
   ' execute os comandos
End If
```

Após a declaração de uma constante, o seu valor não poderá ser mudado dentro do programa.

Constantes simbólicas facilitam a manutenção do programa, pois você precisa alterar apenas o seu valor, em vez de todas as ocorrências do valor literal dentro do programa.

O OpenOffice.org Basic possui as seguintes constantes predefinidas: PI, TRUE e FALSE.

Expressões

Uma expressão é uma constante, uma variável, uma função, ou qualquer combinação destas, separadas por operadores e parênteses, escrita segundo as regras do Basic e passível de avaliação. Seguem os principais operadores encontrados em expressões.

Operador de Atribuição

O resultado de uma expressão pode ser atribuido a uma variável com o uso do operador de atribuição, o sinal de igual (=). Note que neste contexto, o sinal de igual não significa uma comparação, mas sim uma atribuição, isto é, o resultado da expressão do lado direito do sinal será atribuído à variável do lado esquerdo do sinal. Exemplos:

```
' declara as variáveis diametro e raio
Dim diametro#, raio#
' atribui o valor 2.0 ao raio ( 2.0 é uma expressão )
raio = 2.0
' atribui o resultado da expressão ( 2 * raio ) a diametro
diametro = 2 * raio
```

Operadores Aritméticos

São usados com operandos numéricos e produzem resultados numéricos. Os operandos podem ser constantes numéricas, variáveis numéricas ou funções que retornam valores numéricos. Os operadores são:

Operador	Usado para	Operandos
+	Somar expressões	2
-	Subtrair expressões	2
*	Multiplicar expressões	2
/	Dividir a primeira pela segunda expressão	2
\	Dividir expressões inteiras	2
Mod	Obter o resto de uma divisão inteira	2
^	Exponenciação	2

Exemplos:

```
' valor = 0.5 ( note que 5 e 10 são promovidos a Double )
valor# = 5/10
' cos() é uma função interna do Basic
cosTeta# = cos (1.555 )
' PI é uma constante do Basic
area# = 2 * PI * raio ^ 2
' r, a, b, c, d, e são variáveis numéricas
r = (a + b) * (c - d / e)
```

Operadores de Concatenação de Cadeias

São usados para juntar duas ou mais cadeias de caracteres.

Operador	Usado para
&	Concatenar duas ou mais cadeias (strings)
+	Concatenar duas ou mais cadeias (strings)

Exemplos:

```
preNome$ = "Menino"
sobreNome$ = "Maluquinho"
nomeCompleto$ = preNome & " " & sobreNome
```

Operadores Relacionais (Comparação)

São usados para comparar expressões e o resultado é um Booleano, True (-1) ou False (0).

Operador	Usado para
=	As expressões são iguais ?
<>	As expressões são diferentes ?
>	A primeira expressão é maior que a segunda ?
<	A primeira expressão é menor que a segunda ?
>=	A primeira expressão é maior que ou igual a segunda ?
<=	A primeira expressão é menor que ou igual a segunda ?

Na comparação de strings, maiúsculas são diferentes de minúsculas, por padrão.

Exemplos:

```
' se RioSP for > o resultado da expressão é True, senão False
resultado = ( RioSP > RioBH )
' se Raio1 for = a Raio2 o resultado é True, senão False
resultado = ( Raio1 = Raio2 )
```

Operadores Lógicos

Se usados com operandos Booleanos e/ou expressões Booleanas, resultam num valor Booleano. Se usados com operandos numéricos, executam uma operação bit a bit, resultando num valor numérico.

Operador	Usado para	Operandos
NOT	Inverte o resultado booleano	1
OR	Uma das expressões é TRUE ?	2
AND	Ambas as expressões são TRUE ?	2
XOR	Uma expressão é TRUE e a outra FALSE ?	2
EQV	Ambas são TRUE ou ambas são FALSE ?	2
IMP	Se a 1a. for TRUE a 2a precisa ser TRUE	2

Exemplos:

```
'
' resultado True, se as 2 expressões forem True
' FIM_ARQUIVO é uma constante definida anteriormente
resultado = (NrPtos < 1000 And Not FIM_ARQUIVO)
'
' resultado True se a 1a. ou a 2a. for True
' EOF() é uma função do Basic
resultado = (NrPtos < 1000 Or Not EOF())
```

Para exemplos de operação bit a bit, consulte a ajuda do OOoBasic.

Precedência dos Operadores

Uma expressão pode ser composta por mais de um operador. A ordem na qual estes operadores são avaliados chama-se precedência dos operadores, da maior para a menor temos:

( )	Parênteses
^	Exponenciacão
*, /	Multiplicação e divisão
Mod	Módulo
\	Divisão inteira
+, -	Soma e subtração
=, <>, >, <, >=, <=	Operadores relacionais
Not, And, Or, Xor, Eqv, Imp	Operadores lógicos nesta ordem

Para forçar a avaliação de uma expressão podemos, sempre, usar parênteses (maior precedência).

Exemplo:

```
valor1 = 6 + 6 / 3        ' após avaliação valor1  =  8
valor2 = ( 6 + 6 ) / 3    ' após avaliação valor2  =  4
```

2.5 Fluxo de controle da execução

A ordem de execução dos comandos, numa macro, é determinada pelos comandos de decisão (IF e SELECT CASE) e repetição (DO, FOR e WHILE) existentes no programa. Além destes, podemos usar Labels, GOTO e GOSUB, quando estritamente necessário.

Comando de Decisão If ... Then ... End If

Primeira forma do comando IF:

```
If ( expressão ) Then    ' Se expressão for True Então
   '
   ' Execute este bloco de comandos
   '
End If    ‘ Fim Se
```

Segunda forma do comando IF (com a cláusula ELSE):

```
If ( expressão ) Then       ' Se expressão for True Então
    ' Execute este bloco de comandos
Else
    ' Execute este bloco de comandos
End If      ' Fim Se
```

Terceira forma do comando IF (com a cláusula ELSE IF):

```
If (expressão) Then     ' Se expressão for True Então
    ' Execute este bloco de comandos
ElseIf (expressão) Then
    ' Execute este bloco de comandos
Else
    ' Execute este bloco de comandos
End If      ' Fim Se
```

Em todas as formas o uso dos parênteses é opcional. Podemos aninhar vários comandos If.

Exemplo da segunda forma do comando IF:

```
If (a > b) Then  ' se a for > que b, então
    maior = a    ' armazene o valor de a na variavel maior
Else             ' senão
    maior = b    ' armazene o valor de b na variavel maior
End If           ' fim se
' OU, usando IIF:
' maior = IIF ( a > b, a, b )
```

Em expressões simples, podemos usar a função IIF, como abaixo:

```
valor = IIF(expLog, exp1, exp2) ' retorna exp1 se expLog for True, senão retorna exp2
```

Exemplo da terceira forma do comando IF:

```
If (botao = Ok) Then
    mens$ = "OK pressionado"
ElseIf (botao = Cancela) Then
    mens$ = "CANCELA pressionado"
Else
    mens$ = "AJUDA pressionado"
End If
```

Comando de Decisão Select Case

Forma do comando Select Case ... End Select:

```
Select Case ( expressão_de_teste )
      Case lista_de_expressões1
            ' execute este bloco de comandos
      Case lista_de_expressões2
            ' execute este bloco de comandos
      Case Else
            ' execute este bloco de comandos
End Select   '  fim da seleção
```

A instrução Select Case avalia a expressão_de_teste somente uma vez, na entrada do comando, em seguida, compara seu resultado com a lista_de_expressões das cláusulas Case. Se houver uma coincidência, os comandos abaixo do Case serão executados. Os comandos de Case Else (que é opcional) serão executados se não houver nenhuma coincidência anterior.

A lista de expressões pode conter mais de uma expressão ou até uma faixa de valores (Ex: 10 To 20), com cada expressão separada por vírgula. As expressões devem ser do mesmo tipo de dado da expressão_de_teste. Após a execução do bloco de comandos, o controle passa para a próxima linha depois do End Select.

Exemplo de Select Case avaliando constantes:

```
Select Case tipoDesenho
       Case LINHA
            ' executa comandos para linha
       Case CIRCULO
            ' executa comandos para círculo
       Case CURVA
            ' executa comandos para curva
       Case Else
            ' avisa ao operador que o elemento é inválido
End Select
```

Exemplo de Select Case avaliando valores:

```
Select Case corElemento
       Case 0 To 2, 6, 8    ' caso cores 0, 1, 2, 6 ou 8
            ' executa este bloco de comandos
       Case 3 To 5          ' caso cores 3, 4 ou 5
            ' executa este bloco de comandos
       Case Is > 9          ' caso cor > 9
            ' executa este bloco de comandos
End Select
```

Comando de Repetição While ... Wend

Forma do comando While ... Wend:

```
While ( expressão_de_teste )    ' enquanto expressão for True
      '
      ' executa este bloco de comandos
      '
Wend                            ' fim enquanto
```

Este comando avalia a expressão_de_teste no início, se o resultado for True, o bloco de comandos será executado e o controle volta para o While para uma nova avaliação da expressão. Senão o comando após o Wend será executado. É permitido aninhar laços While ... Wend.

Exemplo de While ... Wend:

```
Sub exemplo_While_Wend
   Dim nome As String
   Dim nrTotal As Integer
   '
   nrTotal = 0                      ' inicializa contador de nomes
   nome = Space( 10 )   ' inicializa variável nome
   While (nome <> "")          ' início do laço
      ' bloco de codigo do laço
      '
      nome = InputBox("Digite um nome:")
      If nome <> "" Then
         msgBox "Você digitou: " & nome
         nrTotal = nrTotal + 1 ' incrementa nrTotal
      End If
   Wend                         ' fim do laço
   MsgBox Str$(nrTotal) + " nomes foram digitados."
End Sub
```

Neste exemplo, enquanto o operador digitar um nome o código dentro do laço será repetido.

Comando de Repetição For ... Next

Forma geral do comando For ... Next:

```
FOR Contador = valor_inicial TO valor_final STEP valor_incremento
     ' executa bloco de comandos
     IF (expressão) THEN      ' para sair do laço antes do Contador atingir o valor_final
        EXIT FOR              ' use o comando Exit For
     END IF   ' fim se
NEXT Contador                 ' aqui, Contador é opcional
```

O comando For ... Next funciona da seguinte maneira:

O valor_inicial é atribuído à variável Contador, depois ele testa o contador com o valor_final, se o resultado do teste for True, os comandos dentro do laço serão executados, em seguida Contador será incrementado com valor_incremento e um novo teste será executado, até que se obtenha um valor False para o teste, conforme as regras:

```
Para valor do incremento positivo, o teste será True se Contador <=  valor_final.
Para valor do incremento negativo, o teste será True se Contador >=  valor_final.
```

Se a cláusula STEP for omitida o valor_incremento será 1.

O teste com IF deve ser usado se quisermos sair do laço antes que Contador bata com o valor_final.

Exemplo de For ... Next:

```
' Inicializar um vetor de 100 elementos
'
Sub exemplo_For_Next
    Dim Vetor(1 To 100) As Integer
    Dim I As Integer
    '
    ' para I variando de 1 a 100 com incremento 1
    For I = 1 To 100
        ' I é usado como índice do vetor e também na expressão
        Vetor(I) = I * 2
    Next I    ' vai para a próxima avaliação de I
    ' exibe resultado da soma do 1o e último elementos
    MsgBox Str$ ( Vetor(1) + Vetor(100) )
'
End Sub
```

Aqui, definimos um vetor com 100 elementos, com o índice inicial em 1 e o final em 100 (se você não especificar o índice inicial, o OOo Basic adota o **zero**). Depois, entramos no laço e atribuímos valores (I * 2) para os elementos do vetor. Terminamos exibindo a soma dos primeiro e último elementos do vetor.

O comando For ... Next é muito eficiente e deve ser usado sempre que soubermos o número de repetições do laço.

Podemos aninhar comandos For ... Next (muito útil para operações com matrizes).

Comando de Repetição Do ... Loop

Empregado para executar um bloco de comandos um número indefinido de vezes. Este comando pode ser usado de 5 maneiras diferentes.

Primeira, laço infinito:

```
DO                  ' faça
    ' executa bloco de comandos
    ' teste para sair do laço, sem o teste o laço será infinito
    IF (expressão) THEN
        EXIT DO    ' sai do laço
    END IF
LOOP                ' retorna para o Do
```

Segunda, teste no início do comando com WHILE

```
DO WHILE (expressão)  ' faça ENQUANTO expressão for True
    ' executa bloco de comandos
LOOP                        ' retorna para o Do
```

Terceira, teste no início do comando com UNTIL:

```
DO UNTIL (expressão)  ' faça ATÉ que a expressão seja True
    ' executa bloco de comandos
```

```
LOOP                        ' retorna para o Do
```

Quarta, teste no final do comando com WHILE:

```
DO                          ' faça
    ' executa bloco de comandos
LOOP WHILE (expressão)      ' ENQUANTO expressão for True
```

Quinta, teste no final do comando com UNTIL:

```
DO                          ' faça
    ' executa bloco de comandos
LOOP UNTIL (expressão)      'ATÉ que expressão seja True
```

Em todas as maneiras podemos colocar um teste no interior do laço, para abandoná-lo com o comando EXIT DO. Note que nas formas iniciadas somente com o DO, o laço será executado pelo menos uma vez. Os Comandos Do ... Loop podem ser aninhados.

Exemplo de Do ... Loop, com o teste da expressão no início:

```
Do While ( Not Eof() )    ' faça enquanto não for Fim de Arquivo
   ' leia a linha
   ' processe a linha
Loop                      ' fim do laço
```

Exemplo de Do ... Loop, com o teste da expressão dentro do laço:

```
Do ' faça
   ' obtém nomePonto
   If IsNull (nomePonto) Then  ' se nome do ponto for nulo
      Exit Do                  ' saia do laço
   End If                      ' fim se
   ' processa o ponto
Loop                           ' fim laço
```

Comandos de Desvio Incondicional

GOTO LABEL – Desvia a execução do programa para a linha nomeada.

```
' [ bloco de comandos ]
Goto Label
' [ bloco de comandos ]
Label:
' [ bloco de comandos ]
```

GOSUB LABEL ... RETURN – Desvia a execução do programa para a linha nomeada e retorna, para a linha seguinte ao GoSub Label, ao encontrar o comando Return.

```
' [ bloco de comandos ]
Gosub Label
' [ bloco de comandos ]
Label:
' [ bloco de comandos ]
```

Return

A palavra LABEL deve ser substituída por um nome definido pelo programador. Note o sinal de dois pontos no final da linha nomeada.

Estes comandos devem ser usados quando estritamente necessário.

Comandos de Controle de Erros

Para manter um programa sob controle, precisamos tentar antecipar os erros possíveis e as ações a serem tomadas para contorná-los. Esta é a finalidade do comando ON ERROR. Ele inicia o controlador de erros e pode ser usado de várias maneiras.

On Error Goto Label => em caso de erro desvia para a linha nomeada.
On Error Goto 0 => desativa o controlador de erros e continua a execução.
On Error Resume Next => desativa o controlador e continua a execução na linha seguinte a que gerou o erro.
On Error Resume Label => desativa o controlador e continua a execução na linha nomeada.
Resume Next | Label => podem ser usados isoladamente no bloco de controle de erros.

A palavra **Label**, deve ser substituida por um nome definido pelo programador.

Após a ativação do controlador, podemos obter informações sobre o erro, através de três variáveis globais:

Err => retorna o número do erro.
Erl => retorna a linha de código que provocou o erro.
Error$ => retorna uma descrição sucinta do erro.

A escrita de código para tratamento de erros não é uma tarefa trivial. Como regra geral, procure ativar e desativar o controlador de erros dentro das rotinas e não globalmente. Veja, a seguir, um trecho de código seguindo esta estratégia.

```
Sub exemplo_OnError
   ' Em caso de erro salta para a
   ' linha tratamentoDeErros:
   On Error Goto tratamentoDeErros
   '
   ' aqui vem o código da rotina
   '
   MsgBox "Ufa! Nenhum erro."
   '
   ' desativa o controlador de erros
   On Error Goto 0
   ' sai da rotina
   Exit Sub
'
' código de tratamento de erros
tratamentoDeErros:
   MsgBox "Erro: " & Error$
   ' desativa o controlador de erros
   On Error Goto 0
End Sub
```

Note que temos dois pontos de saída da rotina (Exit Sub e End Sub) e, antes deles, desativamos o controlador de erros com o comando **On Error Goto 0**.

2.6 Tipos de dados avançados

Além dos tipos de dados básicos, o OOoBasic tem recursos para operações com vetores e tipos de dados definidos pelo usuário, que são estruturas de dados mais especializadas.

Vetores

Um vetor (ou matriz) contém um grupo de variáveis com características comuns e com o mesmo tipo de dado. Com uma matriz podemos usar um só nome de variável e acessar os valores de um elemento usando um índice. Uma matriz tem um limite inferior e um limite superior em cada dimensão. Os índices são valores inteiros (negativos ou positivos).

A declaração de uma matriz pode ser feita com a instrução DIM

```
DIM nome_vetor (numero_elementos) AS tipo_dado
DIM nome_matriz (nr_linhas, nr_colunas) AS tipo_dado
```

Os elementos da matriz são referenciados através dos seus índices. Por padrão, os índices começam em zero.

Seguem alguns exemplos de emprego de matrizes:

```
Dim nomeDePonto (30) AS String 'Vetor para 31 nomes de pontos (0 a 30)
Dim coordPonto3D# (30, 2)      ' Matriz de 2 dimensões com 31 linhas e
                               ' 3 colunas (X, Y e Z) do tipo Double
' Atribuindo valores
nomeDePonto(0)  = "Ponto01" ' atribui a cadeia ao 1o. elemento do vetor
nomeDePonto(30) = "Ponto31" ' atribui a cadeia ao último elemento do vetor
' Acessando valores
coordX = coordPonto3D(9,0)  ' obtém o valor da linha 10, coluna 1
coordY = coordPonto3D(9,1)  ' obtém o valor da linha 10, coluna 2
coordZ = coordPonto3D(9,2)  ' obtém o valor da linha 10, coluna 3
```

Os índices inferior e superior podem ser definidos com a cláusula TO, por exemplo:

```
Dim sNomes$(1 To 30)          ' Declara vetor para 30 (1 a 30) nomes
Dim matrizA%(1 To 5, 1 To 3)  ' Declara matriz de 5 linhas X 3 colunas
```

Podemos declarar uma matriz cujo tamanho será determinado posteriormente.

```
Dim nomePonto$( )                ' Declara um vetor de tamanho desconhecido
```

Após conhecer o número de pontos, podemos redimensionar o vetor com REDIM:

```
NumeroPontos = 100
Redim nomePonto(numeroPontos)      'redimensiona o vetor para 101 nomes (0 a 100)
```

REDIM apaga o conteúdo do vetor, se quisermos aumentar o tamanho do vetor preservando o seu conteúdo, devemos usar a cláusula PRESERVE, veja abaixo:

```
REDIM PRESERVE nomePonto(200)   ' aumenta de 100 para 200 conservando o conteúdo
```

Vetores e matrizes são estruturas de dados apropriadas para uso com as instruções de repetição (laços), sendo que a mais adequada é For ... Next. Vejamos um exemplo que soma duas matrizes 3x3.

```
Sub somaMatrizes
   ' declara as matrizes 3x3
   Dim m1(2,2) As Integer
   Dim m2(2,2) As Integer
   Dim soma(2,2) As Integer
   ' inicializa as matrizes m1 e m2
   For i = 0 To 2
      For j = 0 To 2
         m1(i,j) = i * j
         m2(i,j) = m1(i,j) * m1(i,j)
      Next j
   Next i
   ' soma m1 e m2 (poderia ser no laço anterior)
   For i = 0 To 2
      For j = 0 To 2
         soma(i,j) = m1(i,j) + m2(i,j)
      Next j
   Next i
   ' exibe o ultimo elemento de cada matriz
   print m1(2,2); m2(2,2); soma(2,2)
End Sub
```

Vetor de vetores

Nesta estrutura de dado, um vetor armazena outros vetores, que podem ter tipos de dados e tamanhos diferentes. Observe o esquema abaixo:

vetor(0) <= v1(0), v1(1), v1(2), v1(3)
vetor(1) <= v2(0), v2(1)
vetor(2) <= v3(0), v3(1), v3(2)

Aqui a variável **vetor** tem três elementos. O primeiro aponta para o vetor **v1** com quatro elementos, o segundo para **v2** com dois elementos e o terceiro para **v3** com três elementos. Os vetores v1, v2 e v3 podem ser de tipos de dados diferentes. Para acessar os elementos, devemos percorrer o vetor principal e recuperar o vetor secundário que nos interessa.

O vetor principal deve ser declarado como VARIANT. Vamos implementar a estrutura acima:

```
Sub vetorDeVetores
   ' declara o vetor principal
   Dim vetor(2) As Variant
   ' declara v1, v2 e v3
   Dim v1(3) As String
   Dim v2(1) As Integer
   Dim v3(2) As Double
   ' inicializa os vetores
   v1(0) = "José" : v1(1) = "João" : v1(2) = "Maria" : v1(3) = "Rosa"
   v2(0) = 10 : v2(1) = 20
   v3(0) = 1.64 : v3(1) = 1.75 : v3(2) = 1.82
   ' atribui os vetores secundários ao principal
   vetor(0) = v1()
   vetor(1) = v2()
   vetor(2) = v3()
   ' acessa os elementos da estrutura
   For i = LBound(vetor()) To UBound(vetor())
      ' recupera o vetor secundário
```

```
      linha = vetor(i)
      msg = ""
      ' acessa os elementos do vetor secundário
      For j = LBound(linha()) To UBound(linha())
         msg = msg & linha(j) & " "
      Next j
      ' exibe os elementos do vetor secundário
      MsgBox msg
   Next i
End Sub
```

As funções **LBound** e **UBound** retornam os limites inferior e superior de um vetor.

Tipo de dado definido pelo usuário

Esta estrutura permite a definição de um tipo de dado composto por campos. Assemelha-se a um registro de uma tabela.

A criação de um novo tipo de dado é feita com a instrução TYPE ... END TYPE, fora de qualquer procedimento, como a seguir:

```
TYPE novo_tipo
   campo1 AS <tipo_dado>
   campo2 AS <tipo_dado>
   campo3 AS <tipo_dado>
   ...
   campoN AS <tipo_dado>
END TYPE
```

Não podemos ter um vetor como um campo de dado.

Após a definição do tipo, usamos o comando DIM para criar variáveis deste tipo.

```
DIM meu_registro AS novo_tipo
DIM meus_dados(100) AS novo_tipo
```

O acesso aos campos se dá com o uso do **operador .** (ponto).

```
meu_registro.campo1 = valor
minha_var = meu_registro.campo2
meus_dados(10).campo3 = valor3
```

O operador **.** (ponto) será bastante utilizado com os objetos da API do OOo.

Como um exemplo prático, suponha que vamos registrar dados sobre clientes. Os dados são:

```
nome da pessoa: tipo String com tamanho 40
nascimento: tipo String com tamanho 8
altura: tipo Single
peso: tipo Single
```

Como temos diversas pessoas poderiamos usar vetores para manipular os dados, mas neste caso, eles seriam tratados separadamente. Então, é preferível criar um tipo de dado contendo todos os campos e, em seguida, criar um só vetor deste tipo de dado.

Veja uma possível solução:

```
' cria um novo tipo de dado
Type Cliente
   ' declara os campos
   nome AS String*40
   nasc AS String*8
   peso AS Single
   altura As Single
End Type

Sub tipoDefinidoUsuario
   ' declara uma variável para o cliente
   Dim meuCliente As Cliente
   ' inicializa o registro
   meuCliente.nome = "Regina Cavalcante"
   meuCliente.nasc = "10/05/85"
   meuCliente.peso = 61
   meuCliente.altura = 1.72
   ' exibe dados
   print meuCliente.nome; meuCliente.peso
End Sub
```

A declaração do tipo é feita fora da rotina e o acesso aos campos usa o operador ponto. Também, note a declaração de uma cadeia de comprimento fixo.

3 Organização do Programa OOoBasic

No OOo Basic, um programa é organizado em procedimentos (sub-rotinas ou funções). O ponto de entrada da macro, procedimento principal, é da responsabilidade do programador. Sub-rotinas e funções podem ser intrínsicas (vem com o OOo Basic) ou definidas pelo usuário. Uma sub-rotina não retorna nenhum valor, já uma função retorna um valor, logo elas devem ser usadas com parte de uma expressão. Um procedimento pode ser chamado dentro de outros procedimentos de uma macro.

Qualquer bloco de código passível de reutilização na macro corrente ou noutra qualquer, deve ser implementado como um procedimento definido pelo usuário, este é o critério básico. Por exemplo, se numa macro, precisarmos determinar o menor dentre três valores mais de uma vez, devemos criar uma função própria para esta tarefa.

3.1 Comandos

Um comando é uma combinação de elementos do Basic, escrito de acordo com as regras de sintaxe da linguagem. Alguns podem ocorrer em qualquer parte do programa, outros não. Os comandos são os responsáveis pela funcionalidade dos procedimentos.

Normalmente um comando cabe numa só linha, caso contrário podemos usar o **sublinhado**, para indicar que o comando continua na próxima linha. O caractere de continuação não pode ser usado dentro de uma cadeia (string) e deve ser o último na linha. Veja o exemplo:

```
Informe$ = “Esta é uma linha de comando que continua “ + _
            “na próxima linha”
```

Numa mesma linha, podemos escrever mais de um comando, usando o caractere **dois pontos**, como separador de comandos. Abaixo, temos três comandos de atribuição:

```
i = 0 : j = 1 : sinal = False
```

3.2 Sub-rotinas

Sub-rotinas devem ser definidas pelo comando SUB ... END SUB, da seguinte maneira:

```
SUB Nome_Da_Rotina ( Lista_De_Parâmetros )
'
    ' Declaração de variáveis Locais
    ' Comandos da sub-rotina
'
END SUB
```

A Lista_de_Parâmetros, são os valores, separados por vírgula, que a rotina recebe para executar o seu trabalho e devem conter a especificação de tipo. Por exemplo:

```
SUB MinhaSub (par1 As Integer, par2 As Double, par3 As String)
```

Se um dos parâmetros for uma variável da macro o seu nome na lista da sub-rotina pode (e deve) ser diferente do nome da variável na macro.

Exemplo de sub-rotina que troca os valores de duas variáveis:

```
SUB TrocaValores ( valor1 As Double, valor2 As Double)
'
   Dim temp As Double
   '
   temp = valor1
   valor1 = valor2
   valor2 = temp
   ' Note que apesar de não retornar valores, as variáveis passadas
   ' como parâmetros foram alteradas e estas alterações serão
   ' visíveis na rotina que chamar TrocaValores
'
END SUB
```

Agora vamos escrever uma macro que chama a rotina TrocaValores:

```
Sub testeTrocaValores
   ' declara as variaveis
   Dim a As Double, b As Double
   ' atribui valores
   a = 1111.11 : b = 2.22
   print a; b
   ' chama a rotina troca valores
   TrocaValores ( a, b )
   ' mostra o resultado
   print a; b
End Sub
```

O comando EXIT SUB pode ser usado dentro de uma sub-rotina para abandoná-la imediatamente.

3.3 Funções

A definição de uma função é feita entre os comandos FUNCTION ... END FUNCTION, como abaixo:

```
FUNCTION NomeFuncao (ListaParametros) As TipoRetornado

    ' declaração das variáveis locais
    ' comandos da função

    NomeFuncao = expressão_Retorno          ' NÃO ESQUEÇA!

END FUNCTION
```

Note que uma função retorna o valor de uma expressão numa variável de nome igual ao nome da função.

O comando EXIT FUNCTION pode ser usado dentro de uma funçao para abandoná-la imediatamente.

Exemplo de uma função que calcula o volume de uma esfera, a partir do seu raio:

```
FUNCTION VolumeEsfera ( raio As Double ) As Double
   Dim diametro As Double
   '
```

```
   diametro = 2 * raio
   VolumeEsfera =  (PI / 6) * diametro ^ 3
   ' NOTE: nome da função VolumeEsfera; nome da variável: VolumeEsfera
END FUNCTION

Sub testeVolumeEsfera
   Dim r As Double
   r = 5
   volume = VolumeEsfera( r )
   print "Volume: "; volume
End Sub
```

3.4 Passagem de Parâmetros

A passagem de parâmetros para sub-rotinas e funções pode ser feita de duas maneiras, por referência (padrão) ou por valor. Quando um parâmetro (variável) é passado por referência, qualquer alteração em seu conteúdo será refletida na rotina chamadora. Se a passagem for por valor, as alterações na variável serão descartadas quando o procedimento terminar a sua execução e o valor original será preservado. Matrizes são passadas por referência.

Para passar um parâmetro por valor, na definição do procedimento, use a palavra BYVAL, antes do nome do parâmetro, ou, se BYVAL omitida, coloque a variável entre parênteses, na chamada, veja abaixo:

```
SUB ImprimePonto (BYVAL cadeia$, X#, Y#)

    cadeia = Ltrim$(Rtrim$(cadeia) + ", " + Str$(X) + ", " + Str$(Y)
    Print cadeia
    ' a mudança em cadeia, será descartada no término da Sub
END SUB
```

Com relação aos parâmetros, existe, ainda, a questão da obrigatoriedade ou não da passagem. Se declarados normalmente eles são obrigatórios, se declarados com o comando OPTIONAL a passagem é opcional. Veja abaixo uma modificação da declaração anterior.

```
SUB ImprimePonto (BYVAL cadeia$, Optional X, Optional Y)
   ' agora os parâmetros X e Y são opcionais
   ' use a função IsMissing para verificar se foi passado ou não
   If IsMissing(X) Then
      MsgBox "X não foi passado"
   End If
   ' outros comandos
END SUB
```

3.5 Escopo das variáveis

O escopo tem a ver com a visibilidade de uma variável dentro da macro, ou seja, os lugares da macro onde esta variável pode ser referenciada. Quanto ao escopo, uma variável pode ser **local**, **pública** ou **global**, conforme a sua declaração.

Quando for declarada, de modo explícito ou não, dentro de uma sub-rotina ou função ela será visível apenas dentro da sub-rotina ou função, sendo portanto local.

Será pública quando declarada fora de qualquer sub-rotina ou função com os comandos PUBLIC, neste caso ela será visível por todos os procedimentos em todas as bibliotecas, mas seu valor será conservado apenas durante a execução da macro. As variáveis declaradas com DIM ou PRIVATE deveriam ser visíveis apenas no módulo, mas elas se comportam como uma variável pública.

Se for declarada fora de uma sub-rotina ou função com o comando GLOBAL, ela será visível por todas as rotinas de todas bibliotecas e, ainda, preserva o seu valor após o término da execução da macro.

Para entender os conceitos, analise o código abaixo, execute a macro **exemploEscopo** e, logo após, a macro **testaGlobal**:

```
' visivel em todas as bibliotecas, preserva
' o valor entre as chamadas
Global varGlobal As Integer
' visiveis em todas as bibliotecas, não preserva o valor
Public varPublica1 As Integer
Dim varPublica2 As Integer

Sub exemploEscopo
   inicializaVariaveis
   ' varLocal visivel apenas nesta rotina
   ' declaração implícita e não inicializada
   print "varLocal em exemploEscopo: "; varLocal
   print "varGlobal: "; varGlobal
   print "varPublica1: "; varPublica1
   print "varPublica2: "; varPublica2
End Sub

Sub inicializaVariaveis
   ' varLocal visivel apenas nesta rotina
   Dim varLocal As Integer
   varLocal = 1
   varGlobal = 2
   varPublica1 = 3
   varPublica2 = 4
   print "varLocal em inicializaVariaveis: "; varLocal
End Sub

Sub testaGlobal
   ' varGlobal preserva o valor
   print "varGlobal: ";varGlobal
   ' varPublica1 perde o valor
   print "varPublica1: "; varPublica1
End Sub
```

A declaração de uma variável pública ou global deve ser feita fora de qualquer sub-rotina ou função do módulo, procure declará-las no início do módulo.

As variáveis locais existem apenas enquanto a sub-rotina ou função, na qual elas foram declaradas, são executadas. Para contornar esta limitação, existem as variáveis estáticas. Apesar de locais, elas preservam o valor entre chamadas da rotina e são declaradas com o comando STATIC como abaixo:

```
' declaração de variável estática
Static varEstatica As Integer
```

3.6 Chamada de Procedimentos

A chamada a um procedimento depende do seu tipo, se sub-rotina ou função e, ainda, da sua localização. Como uma sub-rotina não retorna valores, ela não precisa ser usada como parte de uma expressão. Já uma Função sempre retorna um valor, logo deve ser usada numa expressão.

Procedimentos na mesma biblioteca

Formas de chamadas de sub-rotina:

```
' Usando o comando CALL com ou sem parênteses
CALL ImprimePonto (nomePonto, coordX, coordY)
CALL ImprimePonto nomePonto, coordX, coordY
' Sem o comando CALL, com ou sem parênteses.
ImprimePonto (nomePonto, coordX, coordY)
ImprimePonto nomePonto, coordX, coordY
```

Se a sub-rotina não tiver parâmetros não é preciso usar os parênteses.

Formas de chamada de funções:

```
' chama a função areaCirculo e armazena o valor da area na variavel areCirc
areCirc = areaCirculo (raio)
' chama 2 funcoes, areaCirculo ( ) e Str$ ( ) o valor retornado será atribuído a cadeia
cadeia = Str$( areaCirculo ( raio ) )
' chama 2 funcoes, Sqr ( ) e Distancia ( ) o valor retornado será atribuído a raizDist
raizDist = Sqr ( Distancia ( x1, y1, x2, y2) )
```

Exemplo de chamadas de sub-rotinas e funções:

```
Sub chamadaProcedimentos
   Call UneCadeiasSub ("Meu ", "exemplo sub 1")
   Call UneCadeiasSub "Meu ", "exemplo sub 2"
   UneCadeiasSub ("Meu ", "exemplo sub 3")
   UneCadeiasSub "Meu ", "exemplo sub 4"
   '
   Dim cad$
   cad$ = UneCadeiasFunction$ ("Meu ", "exemplo function 1")
   MsgBox cad$
End Sub
```

```
Sub UneCadeiasSub ( cad1$, cad2$ )
   MsgBox (cad1$ + cad2$)
End Sub

Function UneCadeiasFunction$ ( cad1$, cad2$ )
   UneCadeiasFunction$ = cad1$ + cad2$
End Function
```

Neste exemplo, a sub-rotina principal é chamadaProcedimentos (é a rotina que deve ser executada), ela chama a Sub UneCadeiasSub e a função UneCadeiasFunction, ambas definidas pelo programador. As quatro formas de chamada da Sub UneCadeiasSub são equivalentes.

Procedimentos em bibliotecas diferentes

Já vimos que uma macro pode ser formada por uma ou mais rotinas. As rotinas, por sua vez, são organizadas dentro de um ou mais módulos. Os módulos são organizados em bibliotecas.

Uma rotina pode chamar outra rotina de qualquer módulo em qualquer biblioteca, respeitando-se as seguintes restrições:

Uma rotina numa biblioteca do aplicativo não pode chamar outra num documento.
Uma rotina num documento não pode chamar outra de outro documento.

A biblioteca Standard do aplicativo é carregada na inicialização do OpenOffice.org. A biblioteca Standard de um documento será carregada na abertura do documento. Portanto, podemos chamar uma rotina da biblioteca Standard normalmente.

Bibliotecas diferentes da Standard devem ser carregadas para a memória antes da chamada, caso contrário o Basic não consegue localizar o procedimento referenciado. Para isto, consideramos dois casos (**note** o uso do **operador .**):

- Para carregar uma biblioteca num dos containeres do aplicativo, usamos o comando:

GlobalScope**.**BasicLibraries**.**loadLibrary (“nome_da_biblioteca”)

- Para carregar uma biblioteca no container documento, usamos o comando:

BasicLibraries**.**loadLibrary (“nome_da_biblioteca”)

Se existir qualquer ambiguidade no nome do procedimento, usamos a especificação completa do mesmo na chamada:

Nome_Modulo**.**Nome_Procedimento
Nome_Biblioteca**.**Nome_Modulo**.**Nome_Procedimento

Atenção: procure evitar ambiguidade nos nomes das bibliotecas, módulos e rotinas.

Vejamos um exemplo, que chama um procedimento na biblioteca Tools, do container do aplicativo:

```
Sub versao_OOo
   ' verifica se a biblioteca Tools está na memória
   If (Not GlobalScope.BasicLibraries.isLibraryLoaded("Tools")) Then
      ' carrega a biblioteca
      GlobalScope.BasicLibraries.loadLibrary("Tools")
   End If
```

```
    ' chama a rotina getProductName da bilioteca Tools
    print getProductName & " - " & getSolarVersion
End Sub
```

Após a carga da biblioteca, ela permanece na memória até o término da seção do OpenOffice.org.

Procedimentos em bibliotecas dinâmicas

Nos ambientes Windows, é possível a chamada de uma rotina numa biblioteca dinâmica (DLL). Para os ambientes Unix esta facilidade não foi implementada.

Antes de chamar uma rotina numa DLL ela deve ser declarada dentro do módulo OOoBasic, fora de qualquer rotina, usando a instrução DECLARE, como abaixo:

Declare Sub ooNome **Lib** "biblio.dll" **Alias** "rotinaNaDLL" (lista_parametros)

Consulte a ajuda do OOoBasic para ver um exemplo sobre este comando.

Procedimentos recursivos

Esta técnica é muito útil na solução de certos problemas. Ela permite que um procedimento chame a si mesmo repetidamente. Desde a versão 1.1.0 o OOoBasic permite, com limitações, chamadas recursivas.

Para mais detalhes, consulte a ajuda do OOoBasic ou algum texto sobre Algorítmos.

3.7 Modelo Geral de uma Macro

Sempre que possível procure colocar a sua macro dentro de uma biblioteca exclusiva. Nesta biblioteca, organize os módulos conforme a funcionalidade do código fonte.

A organização geral de um módulo pode seguir o modelo abaixo, no que for aplicável:

COMENTÁRIOS (Breve Descrição, Nome da Macro, Autor, Data, Chamada, Outros)
DECLARAÇÃO DE VARIÁVEIS PÚBLICAS (Global ...)
DECLARAÇÃO DE VARIÁVEIS PRIVADAS (Public ...)
DEFINIÇÃO DE CONSTANTES SIMBÓLICAS (Const ...)
DEFINIÇÃO DO PROCEDIMENTO PRINCIPAL (Sub ... End Sub)
DEFINIÇÃO DAS SUB-ROTINAS DA MACRO (Sub ... End Sub)
DEFINIÇÃO DAS FUNÇÕES DA MACRO (Function ... End Function)

Consulte as macros distribuídas com o OpenOffice.org, para ver como elas são organizadas.

Seguem algumas recomendações gerais para escrever um código fonte legível, bem documentado e de fácil manutenção:

Apesar de não obrigatório, declare todas as variáveis.
Evite o uso indiscriminado de variáveis Públicas e Globais.
Escolha nomes significativos para as bibliotecas, módulos, rotinas e variáveis.
Use o recurso de endentação no código fonte.
Use comentários concisos e claros para documentar o código fonte.

4 Comandos e Funções do OOoBasic

Já sabemos que o OOoBasic possue uma grande quantidade de instruções (comandos e funções). Estas instruções compõe a RTL Basic (Run-Time Library Basic).

Neste capítulo, as principais funções, dentro de cada categoria, serão apresentadas. Contudo, as funções relacionadas com os objetos UNO serão discutidas noutro contexto.

Para obter uma descrição detalhada de uma função ou comando, no IDE Basic, posicione o curso sobre o nome da mesma e pressione F1.

4.1 Interação com o Operador

São duas as instruções para a **apresentação de mensagens** ao operador.

A primeira, **MsgBox** pode ser usada como uma função (retorna valor) ou como um comando.

```
MsgBox mensagem, tipoDialogo, tituloDialogo
opcao = MsgBox ( mensagem, tipoDialogo, tituloDialogo )
```

MsgBox recebe três parâmetros: a cadeia de caracteres a ser apresentada; um valor definindo o tipo do diálogo; uma cadeia para o título do diálogo. Os dois últimos são opcionais. Consulte a ajuda para verificar os valores possíveis para tipoDialogo e, também, os valores retornados pela função.

A segunda, **Print** não retorna valor e pode receber diferentes tipos de expressões.

```
Print expressao1; cadeia; expressao2
```

Para a **entrada de dados**, temos a instrução **InputBox**, que retorna a cadeia digitada pelo operador ou nada se a entrada for cancelada.

```
Entrada = InputBox( mensagem, tituloDialogo, valorPadrao )
```

Os dois últimos parâmetros são opcionais. Lembre-se, o valor retornado é uma cadeia.

Vejamos dois exemplos simples:

```
Sub exemploInteracao
   Dim sFrase As String
   sFrase = InputBox ("Digite uma frase:", "Prezado usuário", "Padrão")
   If sFrase <> "" Then
      resposta = MsgBox( sFrase, 128 + 32 + 1, "Confirme a frase")
      If resposta = 1 Then
         MsgBox "Frase aceita"
      Else
         MsgBox "Frase recusada"
      End If
   Else
      MsgBox "A entrada foi cancelada!"
   End If
End Sub
```

```
Sub exemploPrint
```

```
   a = 20
   b = 10
   Print "Valores a = "; a; " b = "; b
End Sub
```

4.2 Instruções de tipos de dados

As funções desta seção fazem a verificação do conteúdo e a conversão de tipos de dados.

Funções de verificação

Fornecem informações sobre o tipo de dado de uma variável.

IsNumeric (var) => retorna True se var for numérica, senão False
IsDate (var) => retorna True se var for uma data, senão False
IsArray (var) => retorna True se var for um vetor, senão False
TypeName (var) => retorna o nome do tipo de dado da variável
VarType (var) => retorna um valor indicando o tipo de dado da variável

Funções de conversão

Convertem de um tipo de dado para outro.

CStr (var) => converte um valor numérico para String
CInt (var) => converte uma expressão para Integer
CLng (var) => converte uma expressão para Long
CSng (var) => converte uma expressão para Single
CDbl (var) => converte uma expressão para Double
CBool (var) => converte um valor para Boolean ou compara duas expressões
CDate (var) => converte um valor para Date
As funções acima usam as configurações locais.

Str (var) => converte um valor numérico para String
Val (var) => converte uma String para um valor numérico
Estas funções não usam as configurações locais.

A seguir, um exemplo do uso de algumas funções:

```
Sub exemploConversao
   a = 1000.00
   b = "1000.00"
   c = a + b   ' c = 2.000
   d = CStr(a)
   e = CDbl(b)
   MsgBox TypeName(c)&"-"&TypeName(d)&"-"&TypeName(e)
   f = Val("XYZ")   ' f = 0 ?
   MsgBox f
End Sub
```

O OOoBasic tenta fazer certas conversões, de modo implícito, durante a atribuição, mas evite esta prática (note a linha c = a + b).

4.3 Funções de cadeias de caracteres

O conjunto de funções e comandos da RTL Basic, que lida com cadeias de caracteres, é amplo. Seguem as principais instruções.

Funções para o conjunto de caracteres:

Asc (caractere) => retorna o valor ASCII do caractere

Chr (codigo) => retorna o caractere indicado pelo código (ASCII / Unicode)

Funções para extrair parte de uma cadeia:

Trim (cadeia) => retorna uma cadeia sem os espaços do início e fim

Left (cadeia, x) => retorna os x caracteres à esquerda de cadeia

Right (cadeia, x) => retorna os x caracteres à direita de cadeia

Mid (cadeia, inicio, x) => retorna uma subcadeia começando em início e com x caracteres

Funções de busca e substituição:

InStr (inicio, cadeia1, cadeia2, compara) => localiza cadeia2 em cadeia1, retorna a posição.

Mid (cadeia1, inicio, tamanho, texto1) => substitui texto1 em cadeia1 (é um comando)

Função de formatação (usa as configurações locais):

Format (numero, formato) => retorna numero formatado de acordo com a cadeia formato.

Funções para mudança de caixa:

UCase (cadeia) => retorna cadeia com todos os caracteres em maiúsculas

LCase (cadeia) => retorna cadeia com todos os caracteres em minúsculas

Função para o comprimento:

Len (cadeia) => retorna o comprimento de cadeia

Funções de repetição de caractere:

Space (x) => retorna uma cadeia com x espaços

String (x, caractere) => retorna uma cadeia com x caracteres

Função para comparação de duas cadeias:

StrComp (cadeia1, cadeia2, metodo) => retorna o resultado da comparação

Funções para dividir uma cadeia num vetor e juntar um vetor numa só cadeia:

Split (cadeia, cadeiaSeparadora, numero) => retorna um vetor com as cadeias divididas

Join (vetor, cadeiaSeparadora) => retorna uma cadeia com todas as cadeias do vetor

Para encerrar esta seção, vejamos um exemplo de busca e substituição numa cadeia.

```
Sub exemploSubstituicao
   cadeia = "Esta é uma cadeia de exemplo"
   pos = InStr(cadeia, "a")
   Do While (pos > 0)
      Mid (cadeia, pos, 1, "@")
      pos = InStr(cadeia, "a")
   Loop
   MsgBox cadeia
End Sub
```

4.4 Funções de Data e Hora

Eis um resumo das principais funções:

DateSerial (ano, mes, dia) => retorna o número de dias a contar de 30/12/1899
DateValue (cadeiaData) => converte a cadeiaData para um valor Date
Year (data) => retorna o ano da data
Month (data) => retorna o mês da data
Day (data) => retorna o dia da data
WeekDay (data) => retorna o dia da semana da data (valor de 1 a 7)
TimeSerial (hora, min, seg) => retorna o valor de tempo para o horário
TimeValue (cadeiaHora) => retorna o valor de tempo para o horário em cadeiaHora
Hour (horario) => retorna a hora do horário
Minute (horario) => retorna os minutos do horário
Second (horario) => retorna os segundos do horário
Date => retorna a data do sistema como uma cadeia (ou define uma data para o sistema)
Time => retorna a hora do sistema como uma cadeia
Now => retorna a data e hora do sistema como uma valor Date
Timer => retorna o número de segundos desde a meia-noite
CDateToIso (valor) => converte um valor de data serial para o formato ISO

Vamos a um exemplo que obtém o dia da semana de uma data e calcula o tempo da execução de uma operação matemática, pelo seu computador.

```
Sub exemploDataHora
   Dim lData As Long
   ' define uma data
   lData = DateValue ( "01/08/2004" )
   'lData = DateSerial ( 2004, 8, 1 )
   MsgBox CStr(lData) & " dias desde 30/12/1899."
   'cria um vetor com os nomes dos dias da semana
   nomeDiaSem = Array ("Dom","Seg","Ter","Qua","Qui","Sex","Sab")
   diaSem = WeekDay ( lData )
   MsgBox nomeDiaSem( diaSem - 1 )
   '
   ' calculo do tempo da execução de uma tarefa
   MsgBox "Clique em OK e aguarde o final da tarefa ..."
   ' obtém a hora do inicio
   h1 = TimeValue( Time() )
   'inicia a tarefa
   For i=0 To 100
       For j=1 To 5000
```

```
            tmp = Sqr ( Log ( j ) )
        Next j
    Next i
    ' obtém a hora do término
    h2 = TimeValue( Time() )
    ' obtém a diferença
    dif = h2 - h1
    ' extrai e exibe as partes
    hora = Hour(dif) : min = Minute(dif) :seg = Second(dif)
    Print hora;"h ";min;"m ";seg;"s"
End Sub
```

4.5 Funções Matemáticas

As funções matemáticas básicas estão presentes na RTL Basic. Eis um resumo:

Sin (x) => retorna o seno de x (x em radianos)
Cos (x) => retorna o cosseno de x (x em radianos)
Tan (x) => retorna a tangente de x (x em radianos)
Atn (x) => retorna o arco-tangente de x
Sqr (x) => retorna a raiz quadrada de x
Exp (x) => retorna a base dos logaritmos naturais (**e**) elevada a potência x
Log (x) => retorna o logaritmo natural de x
Fix (x) => retorna a parte inteira de x (sem arredondamento)
Int (x) => retorna o valor inteiro de x (com arredondamento)
Abs (x) => retorna o valor absoluto de x
Sgn (x) => retorna o sinal de x
Hex (x) => retorna o valor hexadecimal de x como String
Oct (x) => retorna o valor octal de x
Rnd (x) => retorna um valor pseudo-aleatório entre 0 e 1
Randomize (semente) => inicializa o gerador de números pseudo-aleatórios

Vejamos um exemplo para calcular o arco-seno e o arco-cosseno de um valor.

```
Sub exemploMatematico
    arcoSeno = (180 / PI) * Asn ( 0.5 )
    arcoCos = (180 / PI) * Acs ( 0.5 )
    print arcoSeno; arcoCos
End Sub

Function Asn( x As Double ) As Double
    Asn = Atn( x / Sqr( -x * x + 1 ) )
End Function

Function Acs( x As Double ) As Double
    Acs = Asn( Sqr( 1 - x ^ 2 ) )
End Function
```

4.6 Instruções de arquivos e diretórios

A RTL Basic oferece várias funções e comandos para arquivos e diretórios. Eis algumas.

Funções de administração

Dir (caminho, atributo) => retorna os nomes dos arquivos ou diretórios
ChDir (caminho) => alterna para o diretório
MkDir (caminho) => cria um diretorio
RmDir (caminho) => remove um diretorio
FileExists (caminho) => retorna True se o caminho existir, senão False
FileCopy (origem, destino) => copia o arquivo origem para destino
Name nomeAtual AS novoNome => renomeia o arquivo ou diretório
Kill (arquivo) => apaga o arquivo
FileLen (caminho) => retorna o tamanho do arquivo em bytes

Funções de abertura e fechamento de arquivos

Open => abre um fluxo de dados (consulte a ajuda do OOoBasic)
Close => fecha um fluxo de dados
FreeFile => retorna o próximo número de arquivo (para usar com Open)
Reset => salva os buffers e fecha todos os arquivos abertos

Funções de entrada e saída

Input #NrArq, listaVariaveis => lê dados do arquivo e armazena na listaVariaveis
Line Input #NrArq, sLinha => lê uma linha do arquivo e armazena em sLinha
Write #NrArq listaExpr => escreve a lista de expressões para o arquivo NrArq
Print #NrArq listaExpr => escreve a lista de expressões para o arquivo NrArq
Lof (nrArq) => retorna o tamanho do arquivo em bytes
Eof (nrArq) => retorna True se o ponteiro estiver no final do arquivo, senão False

Agora um exemplo de entrada e saida num fluxo de dados.

```
Sub exemploIOarquivo
   Dim iNr As Integer
   Dim sLinha As String
   Dim sFluxo As String
   Dim sMsg as String
   ' caminho do arquivo
   sFluxo = "C:\Documents and Settings\User1\My Documents\data.txt"
   ' próximo nr de arquivo livre
   iNr = Freefile
   ' abre o fluxo para saida ( cria o arquivo )
   Open sFluxo For Output As #iNr
   Print #iNr, "Linha de texto"
   Print #iNr, "Outra linha de texto"
   ' fecha o arquivo
   Close #iNr
   iNr = Freefile
   ' abre para entrada
   Open sFluxo For Input As iNr
   ' le até o fim do arquivo
   While not Eof(iNr)
```

```
      Line Input #iNr, sLinha
      If sLinha <>"" then
         sMsg = sMsg & sLinha & Chr(13)
      End If
   Wend
   Close #iNr
   Msgbox sMsg
End Sub
```

Inicialmente, abrimos data.txt **para saída**, escrevemos duas linhas de texto e fechamos o arquivo. A seguir, abrimos o mesmo arquivo **para leitura** e, no laço, fazemos a leitura das linhas para a exibição.

Algumas instruções desta categoria devem ser usadas com cautela, pois existem problemas de portabilidade. A API do OpenOffice.org tem seus próprios objetos para estas tarefas.

4.7 Funções Vetoriais

São poucas as funções para lidar com vetores.

Option Base N => define o limite inferior dos vetores como N (evite este comando)
DimArray (listaDeDimensoes) => retorna uma matriz como Variant
Array (listaDeElementos) => retorna um vetor contendo os elementos da lista
LBound (matriz, n) => retorna o limite inferior da dimensao n da matriz
UBound (matriz, n) => retorna o limite superior da dimensao n da matriz
IsArray (var) => retorna True se var for uma matriz, senão False

Analise o seguinte exemplo:

```
Sub exemploVetor
   ' cria um vetor de tamanho vazio
   vetorDeVetores = DimArray ()
   ' inicializa
   vetorDeVetores = Array ( Array ("Maria", "Regina"), _
                            Array (21, 19), Array (1.72, 1.68) )
   ' recupera o ultimo nome
   vetor = vetorDeVetores( LBound(vetorDeVetores()) )
   nome = vetor( UBound(vetor()) )
   print nome
End Sub
```

O OOoBasic carece de funções para ordenar vetores, localizar, inserir e remover elementos num vetor. Mas, a linguagem oferece recursos para o programador implementar suas próprias funções.

4.8 Instrução With ... End With

Esta instrução deve ser usada com objetos e tipos de dados estruturados. Ela facilita a digitação e a leitura do código fonte.

Eis a sintaxe da instrução:

```
WITH nomeDoObjeto
    .Atributo1 = valor1
    valor2 = .Atributo2
    [ bloco de comandos ]
END WITH
```

Note que o **atributo** é precedido do **operador ponto**. O atributo deve ser o nome de um campo, o nome de uma propriedade ou o nome de um método do objeto **nomeDoObjeto**.

Segue o nosso exemplo de tipo de dado definido pelo usuário reescrito usando With.

```
' cria um novo tipo de dado
Type Cliente
   ' declara os campos
   nome AS String*40
   nasc AS String*8
   peso AS Single
   altura As Single
End Type

Sub exemploWith
   ' declara uma variável para o cliente
   Dim meuCliente As Cliente
   ' inicializa o registro
   With meuCliente
      .nome = "Regina Cavalcante"
      .nasc = "10/05/85"
      .peso = 61
      .altura = 1.72
      ' exibe dados
      print .nome; .peso
   End With
End Sub
```

Observe que mesmo com outros comandos (print) não precisamos digitar o nome da variável.

4.9 Instruções Diversas

Relacionamos, aqui, algumas instruções que não se enquadram nas categorias anteriores:

```
IIF ( expLogica, expr1, expr2 ) => retorna expr1 se expLogica for True, senão expr2
Shell ( sAplicativo, tipoJanela, sArgs, bSinc ) => executa outro aplicativo
Wait miliSegundos => pausa na execução da macro
Environ ( nomeDaVariavelDeAmbiente ) => retorna uma cadeia com o valor da variavel
GetSolarVersion => retorna a versão (build) do OpenOffice.org
GetGUIType => retorna um valor indicando o sistema operacional
Stop => encerra a execução da macro
```

Consulte a ajuda do OOoBasic para mais informações sobre estas funções.

5 Programando Macros sem Objetos

Após três capítulos apresentando a linguagem OOoBasic, vejamos como aplicar as informações adquiridas até o momento.

O principal tipo de aplicação está relacionado com a escrita de fórmulas (funções simples) para as planilhas do Calc. Também, macros para serem disparadas por eventos como, por exemplo, na validação do conteúdo digitado numa célula ou, no Writer, num campo do tipo macro.

5.1 Funções para o Calc

Uma planilha é formada por uma coleção de células, onde cada célula tem um endereço próprio (A1, A2, B1, B2, etc). Uma extensão de células também tem o seu endereço, que é formado pelos endereços das células inicial e final (ex: A1:B2). Para mais informações sobre os esquemas de endereçamento de células e extensões de células, consulte a ajuda do Calc.

As células podem conter funções que recebem ou não algum argumento. Estes argumentos podem (ou não) ser o conteúdo de outras células; nestes casos, eles são passados como o endereço da célula ou da extensão. Eis alguns exemplos de uso de funçoes nativas do Calc, em células de uma planilha:

```
=HOJE() => retorna a data atual do sistema
=SOMA(A1:A10) => retorna a soma dos valores das células A1 até A10
=SOMA(11;22;33;44;55) => retorna a soma das parcelas (11 + 22 + 33 + 44 + 55)
=MEIO(B5;C5;D5) => extrai parte da cadeia em B5 de acordo com os valores em C5 e D5
=MEIO(“cadeia de caracteres”; 8; 2) => retorna a subcadeia “de”
```

Note que algumas funções são flexíveis com relação aos parâmetros. Infelizmente, os parâmetros em nossas funções limitam-se aos declarados na definição da função (argumentos opcionais são permitidos).

Funções recebendo células

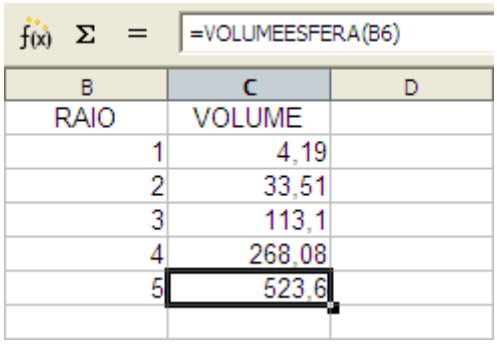
Crie um documento no Calc, salve-o como **macros_cap05.ods** e crie um novo módulo (Module1), na biblioteca **Standard** do documento. Agora, vamos escrever uma função para calcular o volume de uma esfera a partir do seu raio.

```
Function VolumeEsfera ( raio As Double ) As Double
    Dim diametro As Double
    '
    diametro = 2 * raio
    VolumeEsfera =  (PI / 6) * diametro ^ 3
End Function
```

Para usar esta função numa célula de uma planilha do documento basta digitar a fórmula:

```
= VolumeEsfera (B2)          OBS: B2 é o nome da célula contendo o raio da esfera
= VolumeEsfera ( 6,5 )             6,5 é o valor do raio da esfera ( 6.5 dá erro )
```

Observe, na figura abaixo, o resultado do uso desta função na coluna C de uma planilha do documento.



Este exemplo demonstra o caso de funções recebendo células como parâmetros. Nestes casos, o Calc atribui, automaticamente, o conteúdo das células para as variáveis declaradas como parâmetros da função.

Funções recebendo extensões de células

Existem funções que recebem como parâmetro um ou mais intervalos de células, vejamos como recuperar o conteúdo de cada célula dos intervalos.

O Calc sempre interpreta um parâmetro de função como um vetor bi-dimensional, **mesmo nos casos simples** (células). Então, a recuperação do conteúdo de cada célula da extensão resume-se a percorrer os elementos de uma matriz. Entretanto, os limites inferior e superior, desta matriz, são baseados em **um** e **não em zero** (por padrão, os limites dos vetores do OOoBasic são baseados em zero). Por fim, os índices dos elementos são relativos ao intervalo de células e não têm nenhuma relação com o endereço da célula na planilha.

O exemplo abaixo multiplica um valor por cada um dos elementos de uma extensão e retorna a soma dos produtos.

```
Function somaProduto ( valor As Double, range As Variant) As Double
   dSoma = 0
   ' percorre as linhas
   For iLin = LBound(range, 1) To UBound(range, 1)
      ' percorre as colunas
      For iCol = LBound( range, 2) To UBound(range, 2)
         ' recupera o conteudo da célula
         dCelula = range( iLin, iCol )
         dProd = valor * dCelula
         dSoma = dSoma + dProd
      Next
   Next
   somaProduto = dSoma
End Function
```

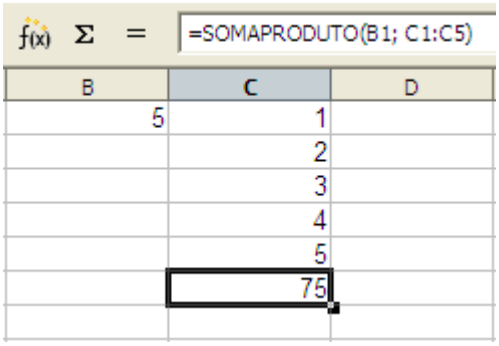
Observe que: (1) as extensões são declaradas como Variant (todos os parâmetros podem ter este tipo); (2) a primeira dimensão da extensão corresponde às **linhas** e a segunda às **colunas**.

Para usar esta função numa célula de uma planilha do documento basta digitar a fórmula:

= SomaProduto (B1; C1:C5) OBS: B1 contém o valor e C1:C5 é a extensão com o vetor
= SomaProduto (5; C1:C5) 5 é o valor e C1:C5 é a extensão com o vetor

Vetores devem ser passados numa extensão de células e o separador de argumentos é o ponto-e-vírgula.

Na figura abaixo, podemos ver o resultado de uma chamada desta função numa planilha do Calc.



Este exemplo funciona com vetores uni e bi-dimensionais. Para os primeiros, podemos eliminar um laço, mas haveria uma sobrecarga na identificação da extensão (linha ou coluna).

Atenção, ao passar o endereço de **uma** célula para uma função, o OOoBasic aceita a sintaxe de vetores uni-dimensionais como referência ao valor da mesma. Veja o código abaixo.

```
Function refCelula(cel)
   ' OBS: passando uma célula, as referências abaixo são válidas !!!
   ' cel    <<=== valor da célula
   ' cel(0) <<=== valor da célula
   ' cel(1) <<=== valor da célula
   ' cel(2) <<=== valor da célula
   '
   ' retorna o dobro do valor da célula !!!
   refCelula  =  cel(0) + cel(1)
End Function
```

No mínimo esquisito, portanto evite este tipo de referência aos valores de uma célula.

Funções Matriciais

Até agora apresentamos funções que retornam um valor para uma célula. Existem, ainda, as funções matriciais, que retornam valores para uma extensão de células. Elas diferem das anteriores apenas no valor de retorno, que deve ser uma matriz.

A seguinte função matricial percorre uma extensão e recupera os índices (linha e coluna) de cada elemento e atribui ao elemento correspondente da matriz de saída.

```
Function indicesDaExtensao(mat1 As Variant) As Variant
   ' obtém nr de linhas
   iLin = UBound(mat1(), 1)
   ' obtém nr de colunas
   iCol = UBound(mat1(), 2)
   ' dimensiona a matriz de saída
```

```
   Dim mat2(1 To iLin, 1 To iCol)
   ' percorre os elementos, extrai os indices
   ' da extensão e guarda em mat2
   For i = LBound(mat1(), 1) To UBound(mat1(), 1)
      ' percorre as colunas
      For j = LBound(mat1(), 2) To UBound(mat1(), 2)
         mat2(i, j) = "(" & CStr(i) & ", " & CStr(j) & ")"
      Next
   Next
   ' retorna a matriz
   indicesDaExtensao() = mat2()
End Function
```

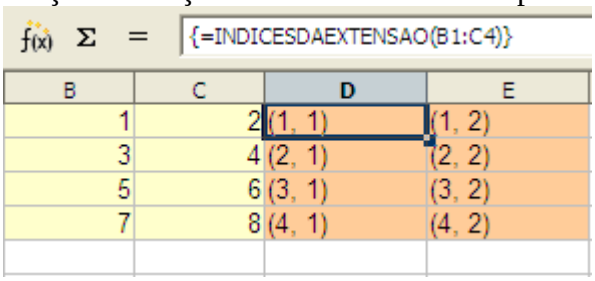
Aqui, o valor de retorno é declarado como **Variant**. Atente para o dimensionamento da matriz de saída.

Para usar esta função numa planilha, digite, por exemplo, a fórmula abaixo numa célula:

= indicesDaExtensao (B1:C4)

Após a digitação, tecle **Ctrl + Shift + Enter** para informar ao Calc que se trata de uma **fórmula matricial**. Se você teclar somente o Enter, apenas o primeiro elemento da matriz será retornado.

Eis o resultado da utilização da função matricial acima numa planilha:



Observe a aparência de uma fórmula matricial na **linha de entrada** da planilha.

Uma recomendação final, procure gravar as funções em módulos da biblioteca Standard do documento ou do container Minhas Macros, assim elas serão avaliadas corretamente na abertura do documento. É possível guardá-las em bibliotecas diferentes da Standard, mas isto requer passos adicionais.

5.2 Macros para Eventos

Para exemplificar este tópico, vamos escrever uma macro para ser disparada por um campo do tipo macro. Digite a seguinte função no container documento do Writer ou Minhas Macros:

```
Function nomeOperador() As String
   Dim sNome As String
   sNome = InputBox ("Nome do operador", "Entrada de dados", "Convidado")
   If sNome = "" Then
      sNome = "Omitido"
   End If
   nomeOperador = sNome
End Function
```

Agora, insira um campo do tipo **Executar Macro** num documento do Writer, selecionando a função acima como a macro a ser executada. A seguir, clique sobre o campo e note que a nossa função será disparada, digite um nome na caixa de entrada e o mesmo será inserido como valor do campo. Caso contrário, nas opções de configuração, em OpenOffice.org Writer – Exibir, desmarque a caixa **Códigos de Campo**.

Atenção: este procedimento não funciona corretamente no OpenOffice.org 2.0.

6 API do OpenOffice.org – Visão Geral

Neste capítulo vamos apresentar alguns conceitos sobre a API do OpenOffice.org.

Para programadores experientes e interessados numa compreensão ampla da API, recomendo a instalação da documentação, em inglês, do SDK do OpenOffice.org. O The Developer's Guide cobre todos os aspectos do desenvolvimento para o OpenOffice.org, com todas as linguagens suportadas, mas quase todos os exemplos estão na linguagem Java. O IDL Reference é um guia de referência completo para a API do OpenOffice.org, contendo informações concisas sobre objetos, métodos, propriedades, constantes, enumerações e exceções.

Para programadores iniciantes e interessados numa **compreensão ampla** da API, recomendo a leitura de algum texto introdutório sobre Orientação a Objetos, Java e componentes JavaBeans, já que os termos da API são baseados na terminologia Java.

6.1 Introdução

A API (Application Programming Interface – Interface de Programação de Aplicações) permite o acesso às funções do OpenOffice.org. Assim, os desenvolvedores podem escrever programas, usando os recursos do OpenOffice.org, de modo fácil e consistente. Suponha, por exemplo, que você está desenvolvendo uma ferramenta para o OpenOffice.org e precisa efetuar uma operação de busca num documento. Sabemos que: (1) o OOo tem uma função de busca e (2) através da API podemos acessar esta função. Então, para resolver o problema, você deve: (1) conhecer o objeto que implementa a busca; (2) definir as propriedades da pesquisa e (3) chamar o método de busca do objeto do OOo. Então, você deve lidar apenas com o item (2), pois a função de busca já existe. Sem uma API, você teria que escrever código para todas as etapas da tarefa.

Resumindo, a API é uma especificação das características programáveis do OpenOffice.org, cuja funcionalidade é baseada numa tecnologia de componentes chamada UNO.

6.2 UNO

A tecnologia UNO (Universal Network Objects – Objetos de rede universal) permite a escrita de componentes para funcionar em diferentes plataformas, usando diferentes linguagens de programação.

Os componentes UNO são definidos através de uma metalinguagem, chamada UNO IDL (Interface Definition Language), que possui os seus próprios identificadores (nomes, tipos e modificadores de dados). Após a definição, os componentes são implementados com uma linguagem suportada como C++ ou Java. A linguagem OOoBasic não permite a implementação de componentes UNO.

Cada linguagem de programação suportada pela API, comunica-se com os objetos UNO através de um módulo específico, responsável pelas tarefas básicas, como conversão de identificadores e de exceções.

Isto significa que podemos escrever um componente para o OpenOffice.org, usando a linguagem Java, num ambiente Linux. Em seguida, podemos escrever uma ferramenta que usa

este componente num ambiente Windows, usando a linguagem Python. Ainda, este componente pode ser acessado na máquina Linux a partir da máquina Windows, usando qualquer linguagem suportada pela API do OOo. A tecnologia UNO é a base para estes objetos de rede comunicarem-se através de sistemas operacionais e linguagens de programação diferentes.

6.3 Organização da API

Na especificação da API encontramos diversos elementos ou tipos. Estes elementos são organizados em módulos funcionalmente estruturados. Dentre os módulos da API, podemos citar:

com.sun.star.awt	- serviços relacionados com a interface do usuário
com.sun.star.beans	- serviços para acesso a propriedades
com.sun.star.container	- interfaces para coleções e recipientes
com.sun.star.document	- serviços para documentos do office
com.sun.star.drawing	- serviços para desenho
com.sun.star.text	- serviços para documentos texto
com.sun.star.sdb	- serviços para banco de dados
com.sun.star.sheet	- serviços para planilhas
com.sun.star.util	- serviços de utilidade diversa

Como você pode notar, o módulo raiz é **com.sun.star** e todos os outros módulos estão subordinados a ele.

Normalmente, um módulo pode conter serviços, interfaces, estruturas, enumerações, definições de tipo e grupos de constantes. Por exemplo, entre os componentes encontrados no módulo **beans**, temos:

Services	- relação de serviços do módulo beans
Interfaces	- relação de interfaces do módulo beans
Structs	- relação de estruturas do módulo beans
Exceptions	- relação de exceções do módulo beans
Enums	- relação de enumerações do módulo beans
Constant Groups	- relação de grupos de constantes do módulo beans

Um módulo não precisa conter todos estes elementos.

Vejamos, resumidamente, o significado de cada um destes elementos.

Service

Serviços, na API do OpenOffice.org, correspondem a objetos. Um serviço pode incluir outros serviços, exportar interfaces e ter propriedades. No tópico sobre objetos, apresentaremos os serviços com mais detalhes.

Interface

As interfaces contém as declarações dos métodos referentes a um dado objeto. Um serviço pode ter interfaces obrigatórias e opcionais. Os nomes das interfaces, por convenção, seguem a forma **XNomeDaInterface** (X maiúsculo seguido pelo nome com a primeira letra maiúscula). Uma mesma interface pode ser implementada de modo diferente por diferentes serviços.

Todas as interfaces UNO são derivadas da interface base XInterface. Assim, uma interface pode estender outras interfaces.

Struct

As estruturas são tipos de dados contendo campos, do mesmo tipo ou não, agrupados sob um mesmo nome. Como exemplo, vamos estudar a estrutura **com.sun.star.awt.Size**, utilizada por diversos serviços da API. Ela define o tamanho de um objeto e contém dois campos:

Struct com.sun.star.awt.Size
Width - valor do tipo longo (long), especificando a largura
Height - valor do tipo longo (long), especificando a altura

Uma estrutura pode conter outras estruturas ou objetos como campos.

Podemos declarar uma variável simples ou um vetor do tipo estrutura, como abaixo:

```
Dim vTamanho As New com.sun.star.awt.Size     ' declara uma variável simples
Dim vetorTam (2) As New com.sun.star.awt.Size ' vetor com 3 elementos
Dim vTam ( )  As New com.sun.star.awt.Size    ' declara um vetor vazio
```

O comando **As New** é utilizado em declarações de tipos de dados não suportados internamente pelo Basic. Note, ainda, a especificação completa do tipo de dado.

O acesso aos campos da estrutura se dá através do operador ponto (**.**) como em:

```
' define os valores dos campos da estrutura Size vTamanho
vTamanho.Width = 2000
vTamanho.Height = 1000
' ou, ainda
nAltura = vTamanho.Height
' acesso aos campos de um vetor de estrutura
vetorTam(0).Width = 5000
vetorTam(0).Height = 2500
```

Exception

Uma exceção é um objeto cuja finalidade é lidar com erros. Ao ocorrer uma situação inesperada, durante a execução do código, o sistema dispara a exceção. Com o OOoBasic, não podemos usar plenamente o mecanismo de exceções UNO. Contudo, o interpretador converte a exceção para um erro Basic, que pode ser tratado com o comando **On Error Goto**.

Enum

Uma enumeração contém uma relação de constantes com valores inteiros em ordem crescente, não necessariamente consecutivos. Por exemplo, a enumeração **com.sun.star.awt.PushButtonType** define tipos possíveis para um botão, são eles:

com.sun.star.awt.PushButtonType.STANDARD - comporta-se como um botão padrão
com.sun.star.awt.PushButtonType.OK - comporta-se como um botão Ok
com.sun.star.awt.PushButtonType.CANCEL - comporta-se como um botão Cancelar
com.sun.star.awt.PushButtonType.HELP - comporta-se como um botão Ajuda

Normalmente, os valores de uma enumeração são utilizados como propriedades de objetos, como abaixo:

```
oBotao1.PushButtonType = com.sun.star.awt.PushButtonType.STANDARD
```

PushButtonType é uma propriedade do objeto botão. Devemos fornecer a especificação completa para o nome da constante na enumeração.

Constant

Um grupo de constantes também contém valores para constantes relacionadas, contudo eles não são, necessariamente, ordenados.

Por exemplo, o grupo **com.sun.star.awt.MouseButton** define constantes indicativas do botão de um mouse, são elas:

com.sun.star.awt.MouseButton.LEFT	- identifica o botão esquerdo do mouse
com.sun.star.awt.MouseButton.RIGHT	- identifica o botão direito do mouse
com.sun.star.awt.MouseButton.MIDDLE	- identifica o botão central do mouse

Normalmente, o valor de uma constante é utilizado como propriedade de um objeto, como abaixo:

```
botaoMouse = EventoDoMouse.Buttons
If botaoMouse = com.sun.star.awt.MouseButton.LEFT Then
   MsgBox "Você pressionou o botão da esquerda!"
End If
```

Na estrutura MouseEvent temos um campo chamado Buttons, ao pressionar ou liberar o botão do mouse este campo receberá um dos valores do grupo de constantes MouseButton. Note que devemos fornecer a especificação completa para o nome da constante.

Atenção, ao usar constantes (enum ou constant) escreva os nomes de acordo com a sintaxe da API, pois, aqui, o OOoBasic é sensível à caixa.

6.4 Mapeamento de dados

Dissemos que os componentes UNO são definidos com uma metalinguagem que possui os seus próprios tipos de dados. As linguagens suportadas precisam mapear os tipos UNO para os seus tipos de dados. Vejamos o mapeamento de dados entre UNO e OOoBasic.

Para os tipos simples o mapeamento ocorre conforme a tabela abaixo:

UNO	OOoBasic
void	-
boolean	Boolean
byte	Integer
short	Integer
unsigned short	-
long	Long

UNO	OOoBasic
unsigned long	-
hyper	-
unsigned hyper	-
float	Single
double	Double
char	-
string	String
any	Variant
type	com.sun.star.reflection.XIdlClass

O tipo UNO **void** equivale a **nada** e, normalmente, é usado para indicar que um método nada retorna; o tipo UNO **any** equivale **qualquer tipo de dado** (por ex: objeto, interface, sequência, string, double, etc).

Muitos objetos UNO usam sequência simples <sequence <tipoDado>> e sequência de sequências <sequence <sequence <tipoDado>>>. Na linguagem OOoBasic, elas são mapeadas como vetor e vetor de vetores respectivamente, do tipo Variant e com o limite inferior zero.

Ao usar vetores com objetos UNO, sempre acrescente os parênteses na frente do nome, como abaixo:

```
umObjetoUNO.Propriedades = vetorComPropriedades()
outroObjetoUNO.setPropertyValue( "nomeProp", vetorComProp() )
```

Ao declarar objetos UNO com o Basic, use o tipo de dado Variant. Evite usar o tipo de dado Object, pois o mesmo destina-se a objetos da linguagem OOoBasic.

Os tipos de dados UNO como **struct**, **enum** e **constant** são mapeados como já demonstrado.

Na API, a especificação de um tipo (service, struct, enum, etc) é hierárquica, iniciando no módulo mais externo, passando pelos módulos internos até chegar ao nome do tipo, por ex:

```
com.sun.star.lang.ServiceManager          => service ServiceManager
com.sun.star.frame.Desktop                => service Desktop
com.sun.star.text.XMailMergeListener      => interface  XMailMergeListener
com.sun.star.beans.PropertyValue          => struct PropertyValue
com.sun.star.awt.PushButtonType.OK        => enum PushButtonType
com.sun.star.awt.MouseButton.LEFT         => constant group MouseButton
```

Esta notação deve ser usada, por exemplo, para a criação do serviço ou tipo de dado.

6.5 Guia de Referência da API

O Guia de Referência da API é parte da documentação (em inglês) distribuída com o SDK. Ele contém uma descrição completa de todos os tipos UNO. Neste documento, vou adotar uma notação simplificada do Guia de Referência.

Como a API pode ser programada usando várias linguagens de programação, na descrição dos métodos, propriedades e campos de estrutura, vou indicar os tipos de dados UNO. Assim, o manual poderá ser utilizado por programadores de todas as linguagens suportadas, observando-se apenas o mapeamento específico para cada uma delas.

Na descrição dos métodos vou indicar os seguintes elementos:

```
tipoUNORetornado nomeDoMetodo ( tipoUNO nomePar1, tipoUNO nomePar2 )
```
Por exemplo:

```
any  getPropertyValue ( string nomePropriedade )
```
o método getPropertyValue retorna um tipo UNO **any**

```
com.sun.star.util.XSearchDescriptor  createSearchDescriptor ( )
```
o método createSearchDescriptor retorna o objeto **XSearchDescriptor** (uma interface)

Com o OOoBasic isto seria declarado da seguinte maneira:

```
getPropertyValue ( nomePropriedade As String ) As Variant
```
conforme o mapeamento de dados UNO/Basic, o tipo **any** corresponde ao **Variant**

```
createSearchDescriptor ( ) As Variant
```
conforme o mapeamento de dados UNO/Basic, objetos da API correspondem ao **Variant**

Porém, o programador OOoBasic só vai precisar declarar métodos ao programar Listeners, já que, atualmente, com esta linguagem não podemos escrever componentes UNO.

As propriedades e campos de estruturas serão descritas como:

```
tipoUNO nomeDaPropriedade
tipoUNO nomeDoCampo
```
Em OOoBasic ficaria:

```
nomeDaPropriedade As tipoBasic
nomeDoCampo As tipoBasic
```

Já dissemos que, às vezes, um valor UNO pode ser retornado como uma sequência ou sequência de sequências. O Guia de Referência usa a seguinte notação:

```
Para sequência: sequence< tipoUNO >
Para sequência de sequências: sequence< sequence< tipoUNO >>
```
Por exemplo:

```
sequence< string > getElementNames ( )
```
o método getElementNames retorna uma sequência do tipo UNO string

```
sequence< sequence< any >>  getDataArray ( )
```
o método getDataArray retorna uma sequência de sequências do tipo UNO any

```
setDataArray ( sequence< sequence< any >> nomeParametro )
```
o método setDataArray recebe uma sequência de sequências do tipo UNO any

Para simplificar, nesta apostila, estes tipos serão representados como:

```
< tipoUNO >          => representa uma sequência do tipoUNO
<< tipoUNO >>        => representa uma sequência de sequências do tipoUNO
```
Por exemplo:

```
< string > getElementNames ( )
<< any >>  getDataArray ( )
setDataArray ( << any >> nomeParametro )
```

No OOoBasic, uma sequência é um vetor e uma sequência de sequência um vetor de vetores

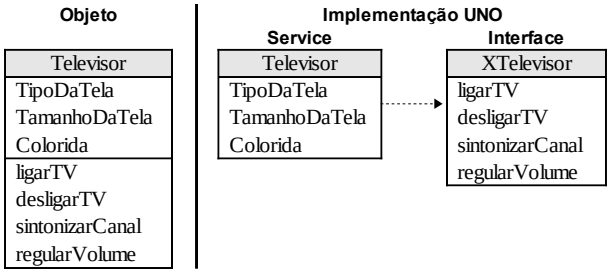
Em todos os casos, um **tipoUNO** pode representar um dos tipos básicos ou um objeto da API, por exemplo uma interface.

6.6 Objetos

Na linguagem comum, a palavra objeto é utilizada para definir um animal, um lugar ou uma coisa. Por exemplo, um aparelho de TV é um objeto com algumas características (tela, tamanho, colorido, etc) e funcionalidades (ligar, sintonizar canal, regular volume, etc).

Em programação, um objeto é uma parte de um programa com dados e comportamento próprio, os dados do objeto são as suas propriedades e o seu comportamento é definido pelos seus métodos. Estes, geralmente, executam alguma ação sobre os dados do objeto.

Eis uma possível representação da nossa TV como um objeto:



Note que os métodos são declarados na interface e as propriedades no serviço. Uma mesma interface pode ser implementada, de modo obrigatório ou opcional, por diferentes serviços.

No OpenOffice.org, todos os objetos (serviços) são registrados numa base de dados. O responsável pelo gerenciamento dos objetos é um objeto chamado **ServiceManager**. Antes de criar uma instância de um objeto, o ServiceManager verifica se o mesmo está registrado e então inicia os procedimentos para a sua criação.

Objetos são criados, normalmente, como resultado da execução de algum método ou através da atribuição de uma variável a uma propriedade do tipo objeto.

Métodos do objeto

Os métodos de um objeto são procedimentos que podem ou não receber parâmetros e retornar valores. Os parâmetros do método ou o valor retornado pelo método podem ser de qualquer um dos tipos de dados UNO e obedecem às regras de mapeamento já apresentadas para o OOoBasic.

Para chamar um método de um objeto, podemos usar a seguinte notação:

```
VariavelObjeto.nomeDoMetodo ( lista_de_argumentos )
valorRetornado = VariavelObjeto.nomeDoMetodo ( lista_de_argumentos )
```

No OOoBasic, quando um método não possui argumentos, os parênteses podem ser omitidos.

Além do nome e tipo de dado, um parâmetro pode ter um dos seguintes atributos:

```
[in] => este argumento recebe um valor que não pode ser alterado
[out] => a este argumento será atribuído um valor de saída
[inout] => este argumento recebe um valor que pode ser alterado
```

Propriedades do objeto

A linguagem IDL possui dois tipos de dados para definir uma propriedade: (1) **property** e (2) **attribute**. A principal diferença entre eles é o modo de acesso aos seus valores.

O valor de uma propriedade deve ser de um dos tipos de dados UNO e, também, obedecem às regras de mapeamento.

São representadas pela estrutura com.sun.star.beans.PropertyValue, cujos campos são:

```
string  Name => nome da propriedade
long  Handle => identificador da propriedade
any  Value => valor da propriedade ou void
short  State => indica se o valor vem do objeto, se é padrão ou se a origem é indeterminada
```

As principais características de uma propriedade são o seu nome (Name) e o valor (Value).

O grupo de constantes com.sun.star.beans.PropertyAttributes, define atributos para uma propriedade, entre eles:

```
BOUND => indica que a propriedade aceita listeners
CONSTRAINED => indica que a propriedade aceita listeners que podem vetar a ação
READONLY => indica que a propriedade é somente leitura
OPTIONAL => indica que a propriedade é opcional
```

Todos os objetos que possuem propriedades implementam pelos menos a interface **com.sun.star.beans.XPropertySet**, do módulo com.sun.star.beans, entre os seus métodos temos:

```
any getPropertyValue ( string sNomePropriedade )
```
obtém o valor de uma propriedade

```
void setPropertyValue ( string sNomePropriedade, any Valor )
```
define o valor da propriedade

```
void addPropertyChangeListener ( string sNomeProp,
                com.sun.star.beans.XPropertyChangeListener xListener )
```
cadastra um listener de mudança na propriedade (os listeners serão apresentados adiante)

```
void removePropertyChangeListener ( string sNomeProp,
                com.sun.star.beans.XPropertyChangeListener xListener )
```
remove o cadastro do listener de mudança na propriedade

Acesso aos valores das propriedades (**property**):

```
Variavel = VariavelObjeto.NomeDaPropriedade
VariavelObjeto.NomeDaPropriedade = Valor
Variavel = VariavelObjeto.getPropertyValue (“NomeDaPropriedade” )
VariavelObjeto.setPropertyValue (“NomeDaPropriedade”, Valor )
```

Acesso aos valores dos atributos (**attribute**):

```
Variavel = VariavelObjeto.getXXX ( )
VariavelObjeto.setXXX ( Valor )
```

Nos métodos getXXX / setXXX, o XXX representa o nome do atributo.

```
Variavel = VariavelObjeto.NomeDoAtributo
VariavelObjeto.NomeDoAtributo = Valor
```

Com o OOoBasic, podemos usar a notação acima para acessar atributos. Dizemos que se trata de uma **pseudo-propriedade**.

Frequentemente, precisamos criar um conjunto de propriedades para atribuir a um objeto. As propriedades são representadas pela estrutura PropertyValue e, para um conjunto de propriedades, devemos usar um vetor de estruturas PropertyValue, como abaixo:

```
Dim vetorPropriedade(2) As New com.sun.star.beans.PropertyValue
vetorPropriedade(0).Name = "nomeProp"
vetorPropriedade(0).Value = umValor
```

Analisando um objeto

Vamos analisar o serviço Spreadsheet, do módulo **com.sun.star.sheet**, que representa uma Planilha num documento do Calc. Eis um resumo deste serviço, como apresentado no guia de referência da API (IDL Reference):

```
com.sun.star.sheet.Spreadsheet:
        Inclui os serviços:
                com.sun.star.sheet.SheetCellRange
                com.sun.star.sheet.Scenario
        Exporta diversas interfaces, entre elas:
                com.sun.star.sheet.XSpreadsheet
                com.sun.star.container.XNamed
        Possui as propriedades:
                boolean IsVisible
                string PageStyle
                boolean AutomaticPrintArea (opcional)
```

Observe que Spreadsheet inclui o serviço SheetCellRange, que por sua vez inclui outros serviços e exporta outras interfaces (consulte a referência). Para o programador, isto significa que todos os métodos e propriedades dos serviços incluídos são herdados por Spreadsheet.

No exemplo acima, a interface XSpreadsheet define métodos que criam um cursor para uma extensão de células, são eles:

```
createCursor          - cria um cursor para toda a planilha
createCursorByRange   - cria um cursor para uma extensão de células
```

Um método corresponde a um procedimento da linguagem Basic.

Existe outra característica das interfaces que deve ser citada. Uma interface pode ser derivada a partir de outra (superinterface). Neste caso, ela herda os métodos da superinterface. Vejamos a hierarquia da interface XSpreadsheet:

```
com.sun.star.uno.XInterface ( interface base para todas as interfaces )
```

```
|___ com.sun.star.table.XCellRange
     |___ com.sun.star.sheet.XSheetCellRange
          |___ com.sun.star.sheet.XSpreadsheet
```

Note que XSpreadsheet é definida a partir da interface XSheetCellRange, que por sua vez é derivada de XCellRange. A conseqüência prática disto é: todos os objetos que oferecem XSpreadsheet suportam os métodos de XCellRange e XSheetCellRange.

Uma propriedade define uma característica de um objeto. Por exemplo, a propriedade IsVisible, acima, determina se a planilha a que se refere será ou não visível na interface gráfica.

O acesso aos métodos e propriedades de um objeto se dá através do operador ponto (**.**). Então, supondo que temos uma variável oPlanilha1, do tipo objeto Spreadsheet, podemos escrever:

```
oCursor = oPlanilha1.createCursor ( )  ' cria um cursor abrangendo toda a planilha
bVisivel = oPlanilha1.IsVisible        ' atribui a bVisivel o valor da propriedade IsVisible
bVisivel = thisComponent.getSheets().getByIndex(0).IsVisible
```

Note, na última linha, o acesso ao valor da propriedade a partir de outro objeto.

Para fixar os conceitos, vamos a um exemplo. Num documento do Calc, digite e execute a rotina abaixo:

```
Sub exemploPlanilha
   ' declara as variáveis da macro
   Dim oDocumento As Variant
   Dim oPlanilha As Variant
   Dim oCursor As Variant
   Dim sNome As String
   Dim bVisivel As Boolean
   ' cria uma instância do objeto documento
   oDocumento = thisComponent
   ' cria uma instância do objeto planilha
   ' com.sun.star.sheet.Spreadsheet
   oPlanilha = oDocumento.getSheets().getByIndex(0)
   ' cria uma instância do objeto cursor
   '   com.sun.star.sheet.SheetCellCursor
   oCursor = oPlanilha.createCursor()
   ' obtém o nome da planilha
   sNome = oPlanilha.Name
   ' EPA, como isto é possível ?
   ' no guia de referência não existe nenhuma propriedade Name !
   ' Na verdade, Name é um atributo obtido através do método getName().
   ' Para simplificar, o OOoBasic permite o acesso a este tipo de
   ' atributo através do seu nome (dizemos que é uma
   ' pseudo-propriedade), o correto seria escrever:
   ' sNome = oPlanilha.getName()
   '
   ' obtém o valor da propriedade IsVisible do objeto oPlanilha
   ' IsVisible é uma propriedade real. Aqui, não podemos usar getIsVisible()
   bVisivel = oPlanilha.IsVisible
   ' com propriedades podemos usar os métodos
   ' getPropertyValue() e setPropertyValue() da interface
   ' com.sun.star.beans.XPropertySet, como abaixo:
   ' bVisivel = oPlanilha.getPropertyValue("IsVisible")
   '
   ' exibe os valores das variáveis
   msgBox sNome & " - " & bVisivel
   ' exibe o nome da fonte
```

```
   msgBox "Nome da fonte: " & oPlanilha.CharFontName
   ' de novo, como isto é possível ?
   ' no guia de referência não existe nenhuma propriedade CharFontName,
   ' para o serviço <Spreadsheet>. É verdade, mas <Spreadsheet> inclui o
   ' serviço <com.sun.star.sheet.SheetCellRange>, que por sua vez inclui
   ' o serviço <com.sun.star.style.CharacterProperties> que possui uma
   ' propriedade <CharFontName>, portanto Spreadsheet herda esta propriedade.
End Sub
```

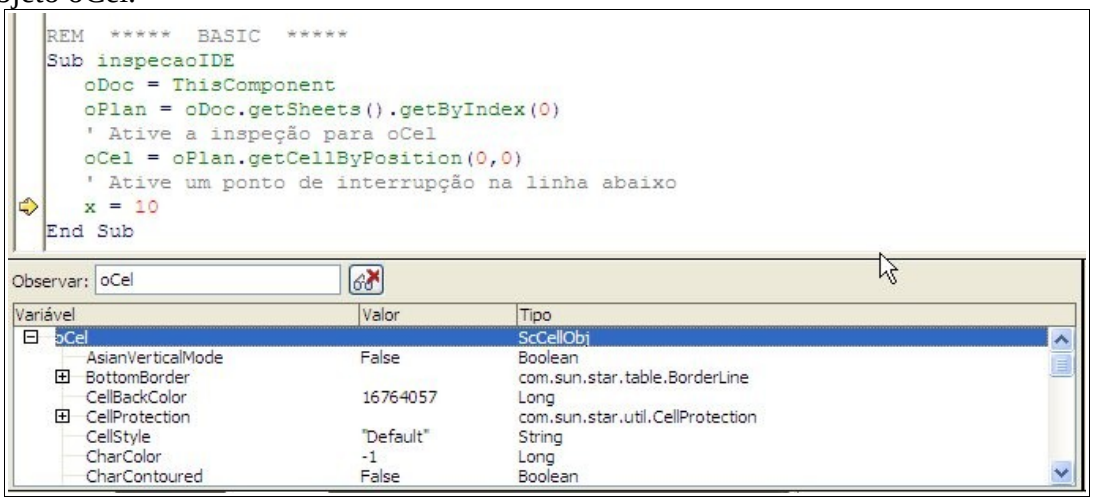
Por favor, leia atentamente os comentários.

6.7 Inspeção de objetos

Outra característica da API, é a capacidade de obter informações sobre um tipo UNO (service, interface, struct, enum, constant, etc). Isto é chamado **introspecção**. Aqui, vamos apresentar três modos simples para inspecionar tipos.

Usando o IDE Basic

A partir do OO.o 2.0, o IDE Basic permite uma análise rudimentar de tipos UNO. Isto é possível acrescentando uma variável UNO à janela de inspeção. Observe a figura abaixo: na rotina **inspeçãoIDE**, ativei a inspeção para a variável **oCel** e defini um ponto de parada na linha seguinte. Ao executar a macro, podemos observar na **janela de inspeção** as propriedades do objeto oCel.



Usando o OOoBasic

Podemos inspecionar objetos usando as propriedades abaixo, do OOoBasic:

Dbg_SupportedInterfaces
retorna as interfaces suportadas pelo objeto

Dbg_Methods
retorna os métodos do objeto

Dbg_Properties
retorna as propiedades do objeto

Eis o exemplo acima, reescrito para usar estas propriedades:

```
Sub inspecaoOOoBasic
   oDoc = ThisComponent
   oPlan = oDoc.getSheets().getByIndex(0)
   ' cria o objeto oCel
   oCel = oPlan.getCellByPosition(0,0)
   ' obtém e exibe as propriedades do OOoBasic
   sInterfaces = oCel.Dbg_supportedInterfaces
   sMetodos = oCel.Dbg_Methods
   sPropriedades = oCel.Dbg_Properties
   msgBox sInterfaces, 176, "Interfaces de uma célula"
   msgBox sMetodos, 176, "Métodos de uma célula"
   msgBox sPropriedades, 176, "Propriedades de uma célula"
End Sub
```

É confuso, mas pode-se trabalhar as cadeias para melhorar a apresentação.

Existem mecanismos poderosos na API, para inspecionar objetos. O utilitário XRayTool usa estes recursos.

Usando o XRay

Este utilitário é **indispensável** para quem pretende programar o OpenOffice.org. Foi desenvolvido por **Bernard Marcelly** e está disponível nas páginas web abaixo:

Última versão (inglês): http://www.ooomacros.org

Tradução da versão XRayTool 5.1: http://geocities.yahoo.com.br/noelsonalves/

(quando disponível, acrescentar vínculo do projeto)

Além de objetos, o XRay inspeciona campos de estruturas e valores em sequências.

Junto com o utilitário, vem um documento explicando em detalhes a sua funcionalidade.

6.8 Instruções UNO do OOoBasic

O OOoBasic possui as instruções abaixo, para lidar com objetos UNO:

GetProcessServiceManager
retorna o objeto com.sun.star.lang.ServiceManager

StarDesktop
retorna o objeto com.sun.star.frame.Desktop

ThisComponent
retorna o modelo do documento com o código ou o modelo do documento corrente, se chamado pelo aplicativo. Se não existir um documento aberto o resultado é indifinido.

CreateUnoService (sEspecificador As String) As Variant
retorna o serviço especificado (esta função usa o ServiceManager para criar o objeto)

CreateUnoStruct (sEspecificador As String) As Variant

retorna a estrutura especificada

CreateUnoDialog (Container.Biblioteca.NomeModulo) As Variant
retorna o objeto diálogo definido no local Container.Biblioteca.NomeModulo

CreateUnoListener (sPrefixo As String, sInterface As String) As Variant
cria um listener, os nomes dos métodos da interface iniciam com o prefixo

HasUnoInterfaces (oObjeto, sInterface1 As String [, sInterface2 As String]) As Boolean
objeto implementa as interfaces ? (separe os nomes das interfaces com vírgula)

EqualUnoObjects (oObjeto1, oObjeto2) As Boolean
objeto1 é igual ao objeto2 ?

IsUnoStruct (aUnoStruct) As Boolean
variável é uma estrutura UNO ?

IsObject (oObjeto) As Boolean
variável é um objeto OLE ?

ConvertToURL (sCaminhoLocal As String) As String
retorna o URL de um arquivo local

ConvertFromURL (sURL As String) As String
retorna um caminho a partir do URL

Todas estas instruções estão documentadas na ajuda do OOoBasic.

7 Principais objetos e interfaces

Existem alguns serviços e interfaces da API que são usados em diversas situações. Neste capítulo, vamos apresentar os principais.

7.1 ServiceManager

O serviço com.sun.star.lang.ServiceManager é encarregado de gerenciar os serviços UNO. Sua principal tarefa é criar instâncias dos objetos UNO, para isto o ServiceManager procura no banco de dados o componente UNO que implementa o serviço desejado e chama o "loader" apropriado para criar o objeto (lembre-se que podemos usar várias linguagens para escrever componentes).

No OOoBasic, podemos criar uma instância do ServiceManager de duas maneiras:

```
oSvManager = GetProcessServiceManager
oSvManager = CreateUnoService ( "com.sun.star.lang.MultiServiceFactory" )
```

Este serviço pode ser suportado por diversos serviços UNO, isto significa que podemos criar um objeto a partir de outro. Por exemplo, um objeto documento do Writer pode ser usado para criar um objeto Tabela.

Entre os seus métodos estão:

Interface **com.sun.star.lang.XMultiServiceFactory**:

`XInterface createInstance ( string sEspecificador)`
retorna o objeto especificado

`XInterface createInstanceWithArguments ( string sEspecificador, < any > aArgs )`
define as propriedades aArgs do objeto, retornando-o

`< string > getAvailableServiceNames ( )`
retorna uma sequência comos nomes dos serviços que podem ser criados com o objeto

Interface **com.sun.star.lang.XMultiComponentFactory**:

```
XInterface createInstanceWithArgumentsAndContext ( string sEspecificador,
                                    < any > aArgs,  XComponentContext oContexto )
```
define as propriedades aArgs do objeto e cria o objeto especificado no contexto oContexto

```
XInterface createInstanceWithContext (  string sEspecificador,
                                                XComponentContext oContexto  )
```
cria o objeto especificado no contexto oContexto

Interface **com.sun.star.lang.XServiceInfo**:

`string getImplementationName ( )`
retorna o nome da implementação do objeto

`boolean supportsService ( string sNomeServico )`
verifica se o serviço sNomeServico é suportado pelo objeto

`< string > getSupportedServiceNames ( )`
retorna os nomes dos objetos suportados por um objeto

Outras interfaces:

`void initialize ( < any > aArgs )`
inicializa um objeto com as propriedades em aArgs

`void dispose ( )`
descarta o objeto

Eis uma rotina usando este serviço:

```
Sub exemploSvManager
   oSvManager = GetProcessServiceManager()
   ' OU:
   ' oSvManager = CreateUnoService("com.sun.star.lang.MultiServiceFactory")
   '
   ' exibe o total de serviços disponíveis para o OOo
   iTotal = UBound(oSvManager.getAvailableServiceNames()) + 1
   msgBox iTotal, 176, "Serviços disponíveis no OOo"
   ' exibe o total de serviços disponíveis para o documento
   iTotal = UBound(ThisComponent.getAvailableServiceNames()) + 1
   msgBox iTotal, 176, "Serviços disponíveis no Documento"
   '
   ' exibe o total de serviços suportados pelo ServiceManager
   iTotal = UBound(oSvManager.getSupportedServiceNames()) + 1
   msgBox iTotal, 176, "Serviços suportados pelo ServiceManager"
   ' exibe o total de serviços suportados pelo documento
   iTotal = UBound(ThisComponent.getSupportedServiceNames()) + 1
   msgBox iTotal, 176, "Serviços suportados pelo Documento"
   '
   ' cria uma instância do objeto com.sun.star.sheet.FunctionAccess
   oFuncao = oSvManager.createInstance("com.sun.star.sheet.FunctionAccess")
   ' OU: podemos dispensar o ServiceManager
   ' oFuncao = CreateUnoService ("com.sun.star.sheet.FunctionAccess")
   '
   ' exibe o nome de implementação do serviço FunctionAccess
   sNomeImpl = oFuncao.getImplementationName()
   msgBox sNomeImpl, 176, "Nome de registro de FunctionAccess"
   ' define valores a serem somados
   aDados = Array (100, 200, 300, 400, 500)
   ' chama um método do objeto oFuncao
   sResultado = oFuncao.callFunction("SUM", aDados())
   ' imprime o resultado da soma
   msgBox sResultado, 176, "Resultado da operação"
End Sub
```

Note a diferença entre serviços suportados e serviços disponíveis. Os primeiros são os serviços incluídos por um objeto, os segundos são os serviços que podem ser criados por um objeto.

7.2 Desktop

O serviço **com.sun.star.frame.Desktop** gerencia os documentos carregados e as janelas do aplicativo. Dentre outras, deve ser usado em tarefas como carregar documentos, identificar os documentos abertos e ativar ou desativar os documentos abertos.

Para criar uma instância deste objeto, podemos usar instruções específicas do OOoBasic ou o ServiceManager:

```
oDesktop = StarDesktop
```

```
oDesktop = CreateUnoService ( “com.sun.star.frame.Desktop” )
oDesktop = GetProcessServiceManager.createInstance ( “com.sun.star.frame.Desktop” )
```

A interface **com.sun.star.frame.XDesktop** define os métodos:

`XEnumerationAccess getComponents ( )`
retorna uma coleção (com.sun.star.container.XEnumerationAccess) dos componentes

`XComponent getCurrentComponent ( )`
retorna o componente (com.sun.star.lang.XComponent) corrente

`XFrame getCurrentFrame ( )`
retorna o frame (com.sun.star.frame.XFrame) do componente corrente

`boolean terminate ( )`
encerra o aplicativo ou retorna False se não for possível

O Desktop implementa a interface **com.sun.star.frame.XComponentLoader**:

`XComponent loadComponentFromURL ( args )`
retorna um componente com.sun.star.frame.XComponent, consulte o próximo capítulo

O serviço Desktop inclui o serviço com.sun.star.frame.Frame e herda todos os seus métodos e propriedades. Através deste serviço podemos efetuar operações sobre as janelas. O objeto Frame será apresentado no próximo capítulo.

O exemplo abaixo exibe o título do frame corrente:

```
Sub exemploDesktop
   ' obtém o Desktop
   oDT = StarDesktop
   ' obtém o frame corrente
   oCFrame = oDT.getCurrentFrame()
   ' exibe titulo do frame
   msgBox oCFrame.Title, 176, "Frame ativo"
End Sub
```

7.3 Coleções

Existem situações em que um objeto pode conter vários objetos de outro tipo, por exemplo:

- Documentos do Writer podem conter tabelas, vários parágrafos e desenhos;
- Documentos do Calc contém planilhas;
- Formulários e Caixas de Diálogos contém controles;
- Documentos do Draw contém desenhos;
- Apresentações contém “slides”

Containeres

Para manipular um conjunto de objetos o OO.o usa um **container**. Geralmente, um container pode ser obtido através de: (1) uma propriedade do objeto que o contém e (2) um método

getYyyy() de uma interface **XYyyySupplier**, implementada pelo objeto (aqui, **Yyyy** é o nome da coleção). Toda interface, cujo nome termina em **Supplier**, fornece uma coleção para o objeto que a implementa.

Como exemplo do primeiro caso, um documento do Calc contém um atributo **Sheets** que representa a coleção de planilhas contidas no documento, para obter esta coleção fazemos:

```
oPlanilhas = oDocumentoCalc.getSheets ( )
```

Como exemplo do segundo caso, um documento do Writer implementa uma interface **XTextTablesSupplier** que define o método **getTextTables()**, então para obter a coleção de tabelas num documento fazemos:

```
oTabelas = oDocumentoWriter.getTextTables ( )
```

Quanto ao acesso aos elementos da coleção, os containeres se classificam em três tipos: (**1**) os que usam **acesso nomeado**; (**2**) os que usam **acesso indexado** e (**3**) os que usam **acesso seqüencial**. Alguns objetos podem implementar um ou mais tipo para uma mesma coleção.

A interface base para todos os containeres é **com.sun.star.container.XElementAccess**. Eis os seus métodos:

```
boolean hasElements ( )
```
retorna True se existirem elementos na coleção, senão retorna False

```
type getElementType ( )
```
retorna o tipo do elemento ou void se existirem vários tipos

Acesso nomeado

Os containeres deste tipo implementam a interface **com.sun.star.container.XNameAccess**, cujos métodos são:

```
any  getByName ( sNome As String )
```
retorna o elemento com o nome sNome

```
< string >  getElementNames ( )
```
retorna uma sequência com os nomes de todos os elementos da coleção

```
boolean  hasByName ( string sNome )
```
retorna True se existir um elemento com o nome sNome, senão retorna False

Adicionalmente, a interface **com.sun.star.container.XNameContainer** pode ser implementada, seus métodos são:

```
void  insertByName ( string sNome, any novoElemento )
```
insere um novo elemento com o nome sNome na coleção

```
void  removeByName ( string sNome )
```
remove o elemento com o nome sNome da coleção

```
void  replaceByName ( string sNome, any novoElemento )
```
substitui o elemento com o nome sNome pelo novo elemento

Num **documento do Calc**, digite e rode a seguinte macro:

```
Sub exemploAcessoNomeado
   ' obtém o documento
   oDoc = thisComponent
   ' obtém a coleção de planilhas
   oPlanilhas = oDoc.getSheets()
   ' a coleção possui elementos ?
   If oPlanilhas.hasElements() Then
      ' a coleção tem um elemento de nome "Planilha1" ?
      If oPlanilhas.hasByName("Planilha1") Then
         ' cria uma instância do objeto Planilha1
         oPlanilha1 = oPlanilhas.getByName("Planilha1")
         ' executa alguma ação sobre o objeto
         msgBox oPlanilha1.getName() & " acessada", 176, "Acesso Nomeado"
      Else
         msgBox "Erro: planilha não existe", 176, "Acesso Nomeado"
      End If
   Else
      msgBox "Erro: não existe nenhum elemento", 176, "Acesso Nomeado"
   End If
End Sub
```

Acesso indexado

Estes containeres implementam a interface **com.sun.star.container.XIndexAccess**, com os métodos:

`any getByIndex ( long lPos )`
retorna o elemento na posição lPos

`long getCount ( )`
retorna a quantidade de elementos da coleção

A interface **com.sun.star.container.XIndexContainer** também pode ser implementada, seus métodos são:

`void insertByIndex ( long lPos, any Elemento )`
insere o elemento na posição lPos

`void removeByIndex ( long lPos )`
remove o elemento na posição lPos

`void replaceByIndex ( long lPos, any novoElemento )`
substitui o elemento na posição lPos pelo novo elemento

Num documento do Calc, digite e execute a macro abaixo:

```
Sub exemploAcessoIndexado
   ' obtém o documento
   oDoc = thisComponent
   ' obtém a coleção de planilhas
   oPlanilhas = oDoc.getSheets()
   ' a coleção possui elementos ?
   If oPlanilhas.hasElements() Then
      ' obtém o número de elementos
      iTotal = oPlanilhas.getCount()
      ' percorre todos os elementos
      For i = 0 To iTotal - 1
         ' obtém o i_ésimo elemento
```

```
         oPlanilha = oPlanilhas.getByIndex( i )
         msgBox oPlanilha.getName() & " acessada.", 176, "Acesso Indexado"
      Next i
   Else
      msgBox "Erro: não existe nenhum elemento", 176, "Acesso Indexado"
   End If
End Sub
```

Acesso seqüêncial

Estes containeres implementam a interface **com.sun.star.container.XEnumerationAccess** com o método:

com.sun.star.container.XEnumeration createEnumeration ( )
retorna uma enumeração dos elementos

Através dos métodos de **XEnumeration** podemos percorrer os elementos do container:

boolean hasMoreElements ( )
retorna True se ainda existir elementos na coleção, senão retorna False

any nextElement ( )
obtém o próximo elemento na coleção

Digite e execute o exemplo abaixo num documento do Calc:

```
Sub exemploAcessoSequencial
   ' obtém o documento
   oDoc = thisComponent
   ' obtém a coleção de planilhas
   oPlanilhas = oDoc.getSheets()
   ' cria uma enumeração dos elementos
   oEnum = oPlanilhas.createEnumeration()
   msg = "" : n = 0
   ' enquanto a coleção possuir elementos
   Do While oEnum.hasMoreElements()
      ' obtém o proximo elemento na coleção
      oPlanilha = oEnum.nextElement()
      ' executa alguma operacao sobre o objeto
      msg = msg & oPlanilha.getName() & Chr$(10)
      n = n + 1
   Loop
   ' exibe mensagem
   msg = msg & Str$(n) & " planilhas acessadas."
   msgBox msg , 176, "Acesso Seqüêncial"
End Sub
```

Note que para acessar os elementos precisamos criar o container Enumeration.

Como visto nos exemplos, existem objetos que implementam os três tipos de acesso para uma mesma coleção.

No módulo com.sun.star.container, existem diversas interfaces relacionadas com operações sobre containeres que não foram mencionadas neste tópico. Contudo, os princípios aqui apresentados são gerais.

7.4 Descritores

Existem serviços da API, cuja finalidade é definir propriedades para descrever outro objeto ou uma ação. São os descritores e os seus nomes são terminados com a palavra **Descriptor**. Além das propriedades eles podem, também, implementar interfaces.

Normalmente, o objeto que usa o descritor oferece um método para a sua criação, estes métodos possuem nomes como **createXxxxDescriptor** (onde Xxxx é o nome). Quando isto não ocorrer, podemos criá-lo usando o ServiceManager ou, apenas, definindo um vetor PropertyValue para receber as propriedades (veja o serviço MediaDescriptor no capítulo seguinte).

Como exemplo, um documento do Writer usa o objeto **SearchDescriptor** para definir as características de uma busca no documento, ele pode ser criado uma chamada ao método

```
com.sun.star.util.XSearchDescriptor  createSearchDescriptor ( )
```

como abaixo:

```
oDescritorBusca = oDocumento.createSearchDescriptor ( )
```

Entre as propriedades de SearchDescriptor, temos:

```
boolean SearchBackwards => se True, busca para o inicio do documento
boolean SearchCaseSensitive => se True, diferencia maiúsculas e minúsculas
boolean SearchWords => se True, busca palavras completas
boolean SearchStyles => se True, busca por um estilo
```

Este objeto também implementa as interfaces XSearchDescriptor e XPropertySet. Os métodos da primeira, são:

```
string getSearchString ( )
```

obtém a cadeia a ser localizada

```
void setSearchString ( string sBusca )
```

define a cadeia a ser localizada

Estas são as características gerais destes objetos e, ao longo do texto, veremos diversos exemplos usando descritores.

7.5 Listeners

Um Listener é um tipo de objeto cuja função é receber uma chamada a um de seus métodos sempre que ocorrer um determinado evento em outro objeto (gerador do evento).

Para isto acontecer, o Listener precisa se cadastrar no objeto gerador e, quando cessar o interesse no evento, remover o cadastro. Para tal, o objeto gerador implementa alguma interface com métodos nomeados como **addYyyyListener** para o cadastro e **removeYyyyListener** para remover o cadastro (Yyyy é o nome do evento). Esta interface pode definir outros métodos relacionados ao evento.

Um Listener ou um Handler consiste de uma interface específica que define métodos para executar alguma ação na ocorrência do evento. Estas interfaces têm o nome na forma **XYyyyListener** ou **XYyyyHandler** (onde Yyyy é o nome do evento). Os métodos devem ser

definidos pelo programador e, no caso do handler, devem retornar um valor booleano indicando se o evento deve ou não ser consumido. Atenção, todos os métodos da interface devem ser definidos, inclusive os das interfaces base.

Estes métodos recebem um parâmetro contendo informações sobre o evento (o programador apenas nomeia o parâmetro e o objeto gerador preenche a estrutura com as informações). Os eventos são representados por alguma estrutura que, por sua vez, é baseada na estrutura:

com.sun.star.lang.EventObject, com o campo

com.sun.star.uno.XInterface Source
contém informações sobre o objeto que disparou o evento

Vejamos um resumo para implementar um listener:

a) definir os métodos da interface do listener
b) criar uma instância do objeto listener
c) cadastrar o listener no objeto gerador do evento
d) posteriormente, remover o cadastro do listener no objeto gerador do evento

Para o item a), no OOoBasic, os nomes de todos os métodos da interface devem começar com o mesmo prefixo seguido pelo caractere _. Por exemplo:

```
' Aqui, umListener_ é o prefixo
' nomeDoMetodo é o nome do método como definido na interface.
' Este esquema deve ser usado apenas com o OOoBasic
Sub umListener_nomeDoMetodo ( oEvento )
   ' código
End Sub
```

Note que o método recebe um argumento, aqui oEvento, para as informações do evento.

Para o item b) usamos a instrução CreateUnoListener como abaixo:

meuListener = CreateUnoListener ("umListener_", "especificadorDaInterface")
Para o item c) devemos chamar o método **addYyyyListener** do objeto gerador:

oObjeto.addYyyyListener (meuListener)
Para o item d) devemos chamar o método **removeYyyyListener** do objeto gerador:

oObjeto.removeYyyyListener (meuListener)

Agora, vamos escrever uma macro contendo um listener para o evento **objeto modificado** de uma célula. Digite o código abaixo num documento do Calc:

```
Global oListener As Variant
Global oCelula As Variant

Sub cadastraListener
   ' obtém o documento do Calc
   oDoc = thisComponent
   ' obtém a 1a. planilha
   oPlan = oDoc.getSheets().getByIndex(0)
   ' obtém a celula A5
   oCelula = oPlan.getCellByPosition (0, 4)
   ' cria uma instancia do objeto listener
   ' NOTA: aqui, XList é o prefixo usado na definição dos
```

```
    '       nomes dos métodos da interface (veja abaixo)
   oListener = CreateUnoListener ("XList_", "com.sun.star.util.XModifyListener")
    ' cadastra o listener no objeto gerador ( a celula )
    oCelula.addModifyListener (oListener)
    msgBox "Listener cadastrado", 176, "Exemplo de Listener"
End Sub

Sub removeListener
   oCelula.removeModifyListener (oListener)
   msgBox "Listener removido", 176, "Exemplo de Listener"
End Sub

'_____________________________________________________________________
' Interface com.sun.star.util.XModifyListener
'_____________________________________________________________________

' metodo modified() de com.sun.star.util.XModifyListener
Sub XList_modified ( oEvento )
   ' acessa o objeto gerador do evento
   valorAtual = oEvento.Source.getFormula()
   msgBox "Valor atual: " & valorAtual, 176, "Exemplo de Listener"
End Sub

' metodo disposing() de com.sun.star.lang.XEventListener
' TODOS os listeners são derivados desta interface.
Sub XList_disposing ( oEvt )
   ' nada a fazer, definicao obrigatoria
End Sub
```

Após executar a rotina **cadastraListener**, todas as vezes que você editar a célula A5, a rotina **modified** será executada. Para remover o Listener execute a rotina **removeListener**.

Note que declaramos duas variáveis globais. Isto, porque a rotina removeListener é executada de modo independente do código de cadastro e ela precisa destes objetos.

7.6 Fluxos

A API tem os seus próprios objetos para lidar com fluxos e pastas, isto aumenta a portabilidade e independência da linguagem de programação.

Acesso a arquivos

O objeto **com.sun.star.ucb.SimpleFileAccess** possui os métodos a seguir:

Na interface **com.sun.star.ucb.XSimpleFileAccess2**:

`void writeFile ( string sURL, com.sun.star.io.XInputStream dados )`
sobrescreve o conteúdo do arquivo, se não existir será criado

Na interface **com.sun.star.ucb.XSimpleFileAccess**:

`void copy ( string urlOrigem, string urlDest )`
copia o arquivo de origem para o destino

`void move ( string urlOrigem, string urlDest )`
move o arquivo da origem para o destino

`void kill ( string sURL )`

remove o URL, mesmo que seja um diretório não vazio !

`boolean isFolder ( string sURL )`
retorna True se URL for um diretório

`boolean isReadOnly ( string sURL )`
retorna True se URL for um diretório

`void setReadOnly ( string sURLArq, boolean bFlag )`
se bFlag True ativa read-only, senão reseta, o processo precisa ter os direitos

`void createFolder ( string sUrlNovaPasta )`
cria uma nova pasta

`long getSize ( string sUrlArquivo )`
retorna o tamanho do arquivo

`string getContentType ( string sUrlArquivo )`
retorna uma cadeia identificando o tipo do arquivo

`com.sun.star.util.DateTime getDateTimeModified ( string sUrlArquivo )`
retorna a data da última modifcação do arquivo

`< string > getFolderContents ( string sUrlPasta, boolean bFlag )`
retorna um vetor com os nomes dos arquivos. Se bFlag True, inclue subdiretórios

`boolean exists ( string sUrlArquivo )`
retorna True se o URL existir, senão False

`com.sun.star.io.XInputStream openFileRead ( string sUrlArquivo )`
retorna um fluxo para leitura, se possível

`com.sun.star.io.XInputStream openFileWrite ( string sUrlArquivo )`
retorna um fluxo para escrita, se possível

`com.sun.star.io.XInputStream openFileReadWrite ( string sUrlArquivo )`
retorna um fluxo para leitura e escrita, se possível

`void setInteractionHandler ( com.sun.star.task.InteractionHandler oHandler )`
define um tratador para outras operações

Além deste objeto básico, no módulo **com.sun.star.io**, existem outros serviços e interfaces relacionadas com a entrada e saída de dados.

Fluxos de entrada e saída

A interface base para operações de entrada em fluxos é **XInputStream**. Seguem os métodos:

`long readBytes ( < byte > aDados, long nTotalBytes )`
retorna os dados na sequência aDados e um valor longo indicando o total de bytes lidos

`long readSomeBytes ( < byte > aDados, long nTotalBytes )`
retorna os dados na sequência aDados e um valor indicando o total de bytes lidos

`long skipBytes ( long nBytes )`
salta o próximos nBytes

`long available ( )`
retorna o número de bytes disponíveis

```
void  closeInput ( )
```
fecha o fluxo de entrada

A interface base para operações de saída em fluxos é **XOutputStream**. Seguem os métodos:

```
void  writeBytes ( < byte >  aDados )
```
escreve a sequência aDados no fluxo

```
void  flush ( )
```
descarrega no fluxo os dados existentes nos buffers

```
void  closeOutput ( )
```
fecha o fluxo de saída

Fluxos de texto

Na API, existem fluxos específicos para operações sobre determinados tipos de dados. Aqui, veremos apenas os que lidam com texto.

O objeto **TextInputStream** lida com a entrada de texto e implementa duas interfaces:

A interface **XActiveDataSink**, com os métodos:

```
void  setInputStream ( com.sun.star.io.XInputStream oFluxoEntrada )
```
encaixa o fluxo de entrada no objeto

```
com.sun.star.io.XInputStream  getInputStream (  )
```
obtém o fluxo encaixado no objeto

E a interface **XTextInputStream** com os métodos:

```
string  readLine ( )
```
retorna a linha de texto, os caracteres são codificados de acordo com setEncoding

```
string readString ( < string > sCaracteresDelim, boolean bRemoveDelim )
```
lê texto até encontrar um delimitador ou EOF. Retorna uma cadeia sem delimitadores

```
boolean  isEOF ( )
```
retorna True se o final de arquivo for encontrado, senão False

```
void setEncoding ( string sConjunto )
```
define o conjunto de caracteres usado na codificação

O objeto **TextOutputStream** lida com a saída de texto e implementa duas interfaces:

A interface **XActiveDataSource** com os métodos:

```
void setOutputStream ( com.sun.star.io.XOutputStream  oFluxoSaida )
```
encaixa o fluxo de saída no objeto

```
com.sun.star.io.XOutputStream  getOutputStream (  )
```
obtém o fluxo encaixado no objeto

E a interface **XTextOutputStream** com os métodos:

```
void  writeString ( string sCadeia )
```
escreve a cadeia no fluxo. Se preciso, os delimitadores devem ser acrescentados

```
void setEncoding ( string  sConjunto )
```
define o conjunto de caracteres usado na codificação

Agora, vamos reescrever o nosso exemplo anterior, usando os recursos da API:

```
Sub exemploFluxoTexto
   ' ALTERE o url para o seu sistema
   sFluxo = ConvertToURL ("c:\OOoTestes\testeIO.txt")
   ' cria o objeto de acesso a arquivos
   oSFA = createUnoService ("com.sun.star.ucb.SimpleFileAccess")
   ' se arquivo existir, sai
   If oSFA.exists(sFluxo) Then
      msgBox "Arquivo já existe, saindo...", 176, "Erro:"
      Exit Sub
   End If
   ' abre o arquivo para escrita
   oFSaida = oSFA.openFileWrite ( sFluxo )
   ' cria o objeto de saida de texto
   oFSTexto = createUnoService ("com.sun.star.io.TextOutputStream")
   ' atribui o fluxo aberto ao objeto de saida de texto
   oFSTexto.setOutputStream (oFSaida)
   ' escreve duas linhas, note o delimitador CR/LF
   oFSTexto.writeString("Linha de texto" & Chr$(13) & Chr$(10))
   oFSTexto.writeString("Outra linha de texto")
   ' fecha o fluxo de saida de texto
   oFSTexto.closeOutput ( )
   '
   msgBox "Saída concluída", 176, "Exemplo de saída"
   ' abre o arquivo para leitura
   oFEntrada = oSFA.openFileRead ( sFluxo )
   ' cria o objeto de entrada de texto
   oFETexto = createUnoService ("com.sun.star.io.TextInputStream")
   ' atribui o fluxo aberto ao objeto de entrada de texto
   oFETexto.setInputStream (oFEntrada)
   ' le o conteudo do arquivo
   Do While ( NOT oFETexto.isEOF() )
      sLinha = oFETexto.readLine( )
      msg = msg & sLinha & Chr(13)
   Loop
   ' fecha o fluxo de entrada de texto
   oFETexto.closeInput ( )
   ' exibe linhas lidas
   msgBox msg, 176, "Exemplo de entrada"
End Sub
```

Eis um resumo dos passos necessários para usar fluxos de entrada e saída:

criar o objeto SimpleFileAccess para acessar arquivos
obter um fluxo, abrindo o arquivo para entrada e / ou saída
criar o objeto de fluxo de dados apropriado ao tipo de operação (textos, objetos, etc)
encaixar o fluxo aberto no objeto
executar a operação desejada
fechar o objeto

Analise o exemplo acima, identificando cada um dos passos.

8 Trabalhando com Documentos

Neste capítulo você aprenderá os conceitos gerais e as operações básicas de programação de documentos do OpenOffice.org.

8.1 URL

O OO.o usa a definição de URL em todas as referências a recursos (documentos, comandos, etc). Um URL simples é formado por: **protocolo://domínio/caminho/recurso**, como em:

```
http://www.empresa.com.br/setor/pagina.html
ftp://user:senha@empresa.com.br/setor/arquivo.sxc
ftp://user@empresa.com.br/setor/arquivo.sxc
file://host/c|/documentos/relatorio.sxw
file:///c|/documentos/relatorio.sxw
```

Existem também URLs específicos do OO.o, como:

```
private:factory/swriter   => URL para criar um novo documento do writer
private:resource/toolbar/standardbar  => URL para acessar um recurso
.component:DB/DataSourceBrowser  => URL do navegador de banco de dados
.uno:About  => URL do comando UNO About (Sobre)
```

Na API, temos a estrutura **com.sun.star.util.URL** com campos para representar as partes de um URL e, para analisar e montar URLs, o objeto **com.sun.star.util.URLTransformer** com a interface XURLTransformer.

Eis um exemplo de análise de um URL usando a API:

```
Sub exemploURL
   ' cria a estrutura
   Dim umURL As New com.sun.star.util.URL
   ' define o URL
   umURL.Complete = "http://www.sitio.com.br:2006/docs/manual.html#cap1"
   ' cria o objeto de análise
   oURLT = CreateUnoService("com.sun.star.util.URLTransformer")
   ' analisa
   oURLT.parseStrict(umURL)
   ' obtém os campos após análise
   msg = "Mark....: " & umURL.Mark & Chr$(10) & _
         "Name....: " & umURL.Name & Chr$(10) & _
         "Path....: " & umURL.Path & Chr$(10) & _
         "Port....: " & umURL.Port & Chr$(10) & _
         "Server..: " & umURL.Server & Chr$(10) & _
         "Protocol: " & umURL.Protocol & Chr$(10)
   ' exibe
   msgBox msg, 176, "Exemplo de campos de URL:"
   msgBox umURL.Main, 176, "Campo Main de URL:"
End Sub
```

Existem outros campos na estrutura URL e outros métodos no objeto URLTransformer.

8.2 Arquitetura FCM

Ao trabalhar com o OpenOffice.org, lidamos com três aspectos do aplicativo: (**1**) seu sistema de quadros e janelas (menu, barra de ferramentas, janela de edição, etc), (**2**) dispositivos de interação com o aplicativo (mouse, teclado, monitor, etc) e (**3**) os dados referentes aos nossos documentos (parágrafos, figuras, gráficos, desenhos, etc).

A API usa a **arquitetura FCM** (Frame :: Controller :: Model) para isolar estes aspectos em camadas independentes. Assim, um documento aberto no OO.o consiste de:

- um modelo (**model**) para os dados e métodos de manipulação do documento. Estes métodos podem alterar os dados independentemente das outras camadas;
- um ou mais controladores (**controllers**) para a interação visual com o documento. O controlador "percebe" as alterações no modelo e cuida da sua representação visual;
- um quadro (**frame**) por controlador. O quadro é o vínculo entre o controlador e o sistema de janelas e, também, o responsável pelo tratamento dos comandos recebidos.

Para o programador, o importante é saber que cada camada é representada por um objeto e que cada um deve ser usado para um dado fim.

Para recuperar ou editar dados, usamos o objeto **modelo** do documento:

```
oDoc = oDesktop.getCurrentComponent ()
oDoc = oDesktop.loadComponentFromURL ( args )
oDoc = ThisComponent
```

qualquer destas instruções cria um modelo do documento (aqui, oDoc).

Ao lidar com a visualização (ex: selecionar dados), usamos o **controlador** do documento:

```
oControlDoc = oDoc.getCurrentController ()
oControlDoc = oDesktop.getCurrentFrame().getController()
```

qualquer destes comandos retorna o objeto controlador do documento (aqui, oControlDoc)

Ao operar com janelas e encaminhamento de comandos, usamos o **frame** do documento:

```
oFrameDoc = oDesktop.getCurrentFrame ()
oFrameDoc = oControlDoc.getFrame ()
```

qualquer destes comandos cria o objeto frame do documento (aqui, oFrameDoc)

Note que todos os objetos são obtidos a partir do objeto Desktop e que é possivel obter um objeto a partir do outro.

Objeto Frame

O objeto **Frame** é suportado pelo serviço Desktop e implementa as seguintes interfaces:

Interface **com.sun.star.frame.XFrame**, entre seus métodos temos:

```
com.sun.star.awt.XWindow  getContainerWindow ( )
```

retorna a janela principal do frame

```
com.sun.star.awt.XWindow  getComponentWindow ( )
```

retorna a janela do componente

`com.sun.star.frame.XController getController ( )`
retorna o objeto controlador

Interface **com.sun.star.frame.XFramesSupplier**, com os métodos:

`com.sun.star.frame.XFrames getFrames ( )`
retorna um container com os frames

`com.sun.star.frame.XFrame getActiveFrame ( )`
retorna o frame ativo

`void setActiveFrame ( com.sun.star.frame.XFrame oFrame )`
define o frame ativo

Ativando Documentos Abertos

Vejamos como ativar os documentos abertos no OpenOffice.org, usando o objeto Frame:

```
Sub exemploFrame
   ' obtém o Desktop
   oDT = StarDesktop
   ' obtém os quadros
   oFrames = oDT.getFrames()
   ' visita os quadros
   For i = 0 To oFrames.getCount() - 1
      ' obtém o quadro
      oFrame = oFrames(i)
      ' obtém a janela principal do frame
      oJanela = oFrame.getContainerWindow()
      ' define o foco nesta janela
      oJanela.toFront()
      ' ou: oJanela.setFocus()
      ' aguarda 2 segundos
      wait 2000
   Next i
End Sub
```

Abra alguns documentos no OO.o e execute a macro acima.

Executando Comandos UNO

Os comandos do OO.o são identificados por um URL e pelas suas propriedades. Estes comandos podem ser executados, a partir de uma macro, para simular uma ação do usuário na interface gráfica.

O método **queryDispatch** do Frame retorna um objeto com informações sobre o comando UNO, o qual pode ser encaminhado usando o método **dispatch**. Vamos simular, via código, a execução do comando Sobre (Ajuda => Sobre).

```
Sub exemploComandoUNO
   ' cria a estrutura
   Dim cmdUNO As New com.sun.star.util.URL
   ' define URL do comando
   cmdUNO.Complete = ".uno:About"
   ' cria o objeto de análise
   oURLT = CreateUnoService("com.sun.star.util.URLTransformer")
```

```
    ' analisa
    oURLT.parseStrict(cmdUNO)
    ' Frame é o objeto que trata os comandos UNO
    oFrame = thisComponent.getCurrentController().getFrame()
    ' obtém um objeto representando o comando About
    ' [ com.sun.star.frame.XDispatch ]
    oDispatcher = oFrame.queryDispatch( cmdUNO, "", 0)
    ' encaminha o comando
    msgBox "Ok para diálogo Sobre", 176, "Executar comando:"
    oDispatcher.dispatch( cmdUNO, Array() )
End Sub
```

Note que este comando UNO não possui propriedades, então usamos um vetor vazio.

Existe outro modo de execução de comandos UNO, que é utilizado pelo gravador de macros. Como exercício, grave uma macro para o comando Sobre e compare os dois códigos.

Nos próximos capítulos, veremos outros exemplos usando os objetos da arquitetura FCM. Para informações detalhadas, consulte o The Developer's Guide.

8.3 Filtros do OO.o

O OO.o usa diversos filtros de conversão para abrir, salvar e exportar documentos. Eles estão relacionados nos arquivos de configuração, da pasta TypeDetection, sob o diretório de instalação do OO.o. Eis alguns filtros:

“MS Word 97” => Para arquivos do MS Word 97
“MS Word 2003 XML” => Para arquivos XML do MS Word 2003
“Rich Text Format” => Para arquivos RTF no Writer
“writer_pdf_Export” => Para salvar arquivos do Writer como PDF
“Star Writer 5.0” => Para arquivos do StarOffice Writer 5.0
“MS Excel 95” => Para arquivos do MS Excel 5.0 / 95
“MS Excel 97” => Para arquivos do MS Excel 97 / 2000 / XP
“MS Excel 2003 XML” => Para arquivos XML do MS Excel 2003
“Rich Text Format (StarCalc)” => Para arquivos RTF no Calc
“Text – txt – csv (StarCalc)” => Para arquivos ASCII no Calc
“dBase” => Para tabelas do dBase

Os filtros podem precisar de propriedades específicas, dependendo da conversão desejada.

Ao longo do texto, vamos apresentar exemplos usando filtros do OO.o.

8.4 Descritor do Meio

Ao carregar, salvar e exportar documentos, precisamos definir algumas propriedades referentes à operação. Para tal, usamos o serviço **com.sun.star.document.MediaDescriptor**. Vejamos algumas propriedades deste descritor:

boolean AsTemplate => cria um novo documento ou abre o modelo para edição
string FilterName => filtro usado para abrir / salvar o documento
string FilterOptions => opções adicionais do filtro
boolean ReadOnly => somente leitura ou leitura / gravação
boolean Hidden => oculta o documento na interface
boolean Unpacked => ao salvar não compacta o documento

string Password => senha de proteção do documento

Um vetor da estrutura com.sun.star.beans.PropertyValue pode ser usado para definir as propriedades do descritor, da seguinte maneira:

```
Dim descMedia(2) As New com.sun.star.beans.PropertyValue
descMedia(0).Name  = "AsTemplate"
descMedia(0).Value = False
descMedia(1).Name  = "FilterName"
descMedia(1).Value = "MS Excel 97"
descMedia(2).Name  = "ReadOnly"
descMedia(2).Value = False
```

Posteriormente, este vetor será passado como argumento de algum método.

8.5 Documentos do Office

Todos os documentos do OpenOffice.org são baseados no objeto **OfficeDocument**, representado pelo serviço com.sun.star.document.OfficeDocument.

As propriedades do objeto OfficeDocument são:

boolean AutomaticControlFocus => foco no controle do formulário
boolean ApplyFormDesignMode => ativa o modo de desenho de formulário

Este objeto implementa as interfaces abaixo, com os respectivos métodos:

Interface **com.sun.star.frame.XModel**

string getURL ( )
retorna o URL do documento

< com.sun.star.beans.PropertyValue > getArgs ( )
retorna uma sequência com os argumentos correntes do descritor do meio (MediaDescriptor)

void lockControllers ( )
bloqueia os controladores do documento (ex: impede atualização da visão)

void unlockControllers ( )
desbloqueia os controladores

boolean hasControllersLocked ( )
retorna True se os controladores estiverem bloqueados

com.sun.star.frame.XController getCurrentController ( )
retorna o controlador corrente

com.sun.star.uno.XInterface getCurrentSelection ( )
retorna a seleção corrente (o tipo de objeto depende do conteúdo selecionado)

void dispose ( )
descarta o objeto (pode ser usado para fechar documentos em versões antigas)

Interface **com.sun.star.util.XModifiable**

boolean IsModified ( )
retorna True se o documento foi modificado

```
void  setModified ( boolean bFlag )
```
define o estado do flag de alteração do documento

Interface **com.sun.star.frame.XStoreable**

```
boolean  hasLocation ( )
```
retorna True se o documento está armazenado fisicamente

```
string  getLocation ( )
```
retorna o URL do documento

```
boolean  isReadonly ( )
```
retorna True se somente leitura

```
void  store ( )
```
salva o documento, deve ser usado **apenas** com arquivos existentes (já armazenados)

```
void  storeAsURL ( string sURL, < com.sun.star.beans.PropertyValue >  descMedia )
```
salva o documento no URL, segundo propriedades do descMedia

```
void storeToURL ( string sURL, < com.sun.star.beans.PropertyValue >  descMedia )
```
salva o documento no URL, segundo propriedades do descMedia (use para exportar)

Interface **com.sun.star.view.XPrintable**

```
< com.sun.star.beans.PropertyValue >  getPrinter ( )
```
retorna uma sequência de propriedades do descritor da impressora

```
setPrinter ( < com.sun.star.beans.PropertyValue >  vProp )
```
define uma impressora conforme as propriedades

```
print ( < com.sun.star.beans.PropertyValue >  vProp )
```
imprime o documento de acordo com as propriedades

Interface **com.sun.star.document.XDocumentInfoSupplier**

```
com.sun.star.document.XDocumentInfo  getDocumentInfo ( )
```
retorna o objeto DocumentInfo

Entre as informações de **DocumentInfo**, temos:

```
string Author => autor do documento
com.sun.star.util.DateTime  CreationDate  => data de criação do documento
string Title => título do documento
string Description => descrição do documento
string MIMEType => tipo MIME do documento
com.sun.star.lang.Locale  Language => linguagem do documento
string  ModifiedBy => modificado por
com.sun.star.util.DateTime  ModifyDate => data da modificação
string  PrintedBy => impresso por
com.sun.star.util.DateTime  PrintDate => data da impressão
string  Template => modelo no qual o documento está baseado
com.sun.star.util.DateTime  TemplateDate => data da última atualização doc / template
```

A estrutura com.sun.star.util.DateTime possui os campos:

```
Year – Month – Day – Hours – Minutes – Seconds – HundredthSeconds
```

O objeto OfficeDocument implementa outras interfaces relacionadas com controle de eventos, estado da visão do documento, etc.

Cada aplicativo possui os objetos abaixo representando os seus documentos, todos incluem o objeto OfficeDocument.

```
Writer.....: com.sun.star.text.TextDocument
Calc.....: com.sun.star.sheet.SpreadsheetDocument
Presentation.....: com.sun.star.presentation.PresentationDocument
Draw.....: com.sun.star.drawing.DrawingDocument
Base.....: com.sun.star.sdb.OfficeDatabaseDocument
HTML.....: com.sun.star.text.WebDocument
Formulário XML: tem como base um documento do Writer
```

Um pequeno exemplo, para exibir algumas características do documento ativo.

```
Sub exemploOfficeDocument
   oDoc = thisComponent
   ' exibe algumas características
   msgBox oDoc.ApplyFormDesignMode, 176, "Modo de desenho ?"
   msgBox oDoc.IsModified(), 176, "Foi modificado ?"
   msgBox oDoc.hasLocation(), 176, "Documento foi armazenado ?"
   ' obtém as informações do documento
   oDocInfo = oDoc.getDocumentInfo()
   msgBox oDocInfo.Author, 176, "Autor do Documento"
   Dim sData As String
   Dim sHora As String
   ' obtém a estrutura
   vData = oDocInfo.CreationDate
   ' converte e exibe dados
   sData = DateSerial(vData.Year, vData.Month, vData.Day)
   sHora = TimeSerial(vData.Hours, vData.Minutes, vData.Seconds)
   msgBox "Data: " & sData & " Hora: " & sHora, 176, "Data da Criação"
End Sub
```

8.6 Interface XComponentLoader

A interface **com.sun.star.frame.XComponentLoader**, implementada pelo objeto Desktop, possui um método para carregar componentes:

```
com.sun.star.lang.XComponent  loadComponentFromURL (
                                  string sURL,
                                  string sNomeFrame,
                                  long iFlagBusca,
                                  <com.sun.star.beans.PropertyValue> descMedia  )
```

vejamos o significado de cada parâmetro:

```
sURL => URL do componente
            Eis os URLs privados para criar novos documentos:
                Writer  => “private:factory/swriter”
                Calc    => “private:factory/scalc”
                Draw    => “private:factory/sdraw”
                Impress => “private:factory/simpress”

sNomeFrame => nome do quadro de destino, se não existir será criado.
            Os nomes abaixo são reservados pelo OO.o:
                “_blank” => cria um novo frame no desktop
                “_default” => similar ao anterior, comportamento difere
```

```
                    "_self" => usa o próprio frame, se possível
                    "_parent" => usa o frame pai, se possível
                    "_top" => usa o frame no topo, se possível
                    "_beamer" => usado com componentes simples do OO.o (sem modelo)

iFlagBusca => define como buscar um frame, use 0 (zero) se definir sNomeFrame

descMedia => descritor do meio, podemos usar um vetor vazio.
```

Nota: consulte a documentação para uma descrição completa dos parâmetros

As ferramentas para executar todas as **tarefas básicas** sobre documentos foram apresentadas, nos próximos itens veremos alguns exemplos destas operações.

8.7 Criando Documentos

Para criar novos documentos, devemos:

```
obter o objeto Desktop
definir o URL de novo documento (private:factory/ + tipoDocumento)
      ou: para criar a partir de um modelo, passe o URL do modelo
definir as propriedades do descritor do meio, se necessário
definir o quadro, para um novo frame use "_blank"
definir o flag de busca do frame, quando este não for definido; senão use 0 (zero)
chamar o método loadComponentFromURL passando os parâmetros
```

Para exemplificar, vamos criar dois novos documentos um do Writer e outro do Calc:

```
Sub novoDocWriter
   Dim oProp() As New com.sun.star.beans.PropertyValue
   oDesk = StarDesktop
   sUrl = "private:factory/swriter"
   oDoc = oDesk.loadComponentFromURL ( sUrl,"_blank",0,oProp() )
End Sub
```

```
Sub novoDocCalc
   sUrl = "private:factory/scalc"
   oDoc = StarDesktop.loadComponentFromURL (sUrl, "_blank", 0, Array())
End Sub
```

Nota: a função **Array** retorna um vetor vazio, nestes casos podemos omitir a declaração.

8.8 Abrindo Documentos

O procedimento para abrir documentos é similar ao da criação, mas o URL aponta para o documento a ser carregado. Pela GUI, salve um novo documento do Writer e adapte a rotina a seguir para carregá-lo. Antes de executar a macro, feche o documento.

```
Sub carregaDoc
   ' adapte o URL para o seu sistema
   sUrl = ConvertToURL ("C:\prog_ooo\testeAbrir.sxw")
   oDoc = StarDesktop.loadComponentFromURL (sUrl, "_blank", 0, Array())
```

```
End Sub
```

8.9 Salvando e Exportando Documentos

Os passos são:

obter o objeto documento a ser salvo, usando por exemplo:
- ThisComponent
- loadComponentFromURL
- getCurrentComponent

definir o URL
definir as propriedades do descritor de mídia
Chamar um dos métodos da interface com.sun.star.frame.XStorable, lembrando que:
- store( ) => sobrescreve o arquivo, **nunca** usar com novos documentos
- storeAsURL (sURL, descMedia) => similar ao Salvar Como
- storeToURL (sURL, descMedia)=> não altera o original (use para exportar documentos)

Os métodos abaixo também podem ser usados para obter um maior controle:
isModified – isReadonly – hasLocation – getLocation

Eis um exemplo para salvar o documento ativo:

```
Sub salvaDocAtivo
   Dim oProp() As New com.sun.star.beans.PropertyValue
   oDesk = StarDesktop
   ' obtém o modelo do documento
   oDoc = oDesk.getCurrentComponent()
   ' o documento foi modificado ?
   If (oDoc.isModified()) Then
      If (oDoc.hasLocation() And (Not oDoc.isReadOnly())) Then
         ' documento já existe e não é somente leitura,
         ' então sobrescreve:
         oDoc.store()
      Else
         ' é um novo documento, precisamos do URL:
         ' adapte para o seu sistema
         sURL = ConvertToURL ( "c:\novo_doc_ooo" )
         oDoc.storeAsURL(sURL, oProp())
      End If
   Else
      MsgBox "O documento não foi modificado."
   End If
End Sub
```

O exemplo seguinte exporta um documento do writer para PDF:

```
Sub exportaPDF
   ' precisamos no mínimo da propriedade Filtro
   Dim aProp(0) As New com.sun.star.beans.PropertyValue
   oDoc = thisComponent
   ' adapte para seu sistema
   sUrl = "c:\OOOexemplos\tst_PDF.pdf"
   sUrl = ConvertToURL ( sUrl )
   ' define o filtro de exportação
   aProp(0).Name = "FilterName"
   aProp(0).Value = "writer_pdf_Export"
   ' para exportar use storeToURL
```

```
   oDoc.storeToURL ( sUrl, aProp() )
End Sub
```

8.10 Imprimindo Documentos

O serviço OfficeDocument, através da interface **com.sun.star.view.XPrintable**, dispõe dos métodos abaixo para lidar com impressões:

< com.sun.star.beans.PropertyValue > getPrinter ( )
retorna o descritor da impressora atual, cujas principais propriedades são:

```
string Name  => nome da fila
com.sun.star.view.PaperOrientation  PaperOrientation => orientação do papel
com.sun.star.awt.Size  PaperSize => tamanho do papel em centésimos de mm
boolean  isBusy  => a fila está ocupada (imprimindo) ?
com.sun.star.view.PaperFormat  PaperFormat => formato do papel (A3=0, A4=1, etc)
```

void setPrinter (< com.sun.star.beans.PropertyValue > Descritor)
define uma nova impressora para o documento (reformata o documento)

void print (< com.sun.star.beans.PropertyValue > vOpcoes)
imprime o documento segundo as opções de impressão, conforme PrintOptions.

As opções do serviço **com.sun.star.view.PrintOptions**, são:

```
short  CopyCount => número de cópias
string  FileName => nome do arquivo a imprimir
boolean  Collate => agrupar cópias
string  Pages => cadeia com as páginas a imprimir ( por ex: “1; 3; 5-8; 12” )
boolean  Wait => se True, aguarda término da impressão
```

O exemplo abaixo retorna o descritor da impressora definida para o documento:

```
Sub obtemDescritorImpressora
   oDoc = ThisComponent
   ' obtém o descritor
   oDescImpr = oDoc.getPrinter()
   iNrEstruturas% = UBound(oDescImpr)
   sMsg = ""
   ' percorre as propriedades
   For n = 0 To iNrEstruturas%
      sMsg=sMsg + oDescImpr(n).Name + Chr$(10)
   Next n
   MsgBox sMsg, 176, "Propriedades da Impressora"
   ' obtém e exibe o valor da propriedade PaperFormat
   ' segundo a enum <com.sun.star.view.PaperFormat>
   ' A3=0; A4=1; Letter/Carta=5; ...
   MsgBox oDescImpr(2).Value, 176, "Valor de PaperFormat"
End Sub
```

Ao imprimir documentos, observe como definir algumas opções de impressão:

```
   Dim aOpcoes(0) As New com.sun.star.beans.PropertyValue
   aOpcoes(0).Name = "Pages"
```

```
   aOpcoes(0).Value = "1;3;5-8;12"
   thisComponent.print ( aOpcoes() )
```

8.11 Fechando Documentos

A partir do OO.o versão 1.1.x, os documentos suportam a interface com.sun.star.util.XCloseable, com o método:

void close (boolean bFlag)
Fecha o documento, use True para repassar a responsabiblidade para outros processos que podem impedir o fechamento.

Em versões antigas do OO.o, devemos usar o método dispose ( ) da interface XComponent. Antes de fechar o documento, salve as alterações.

Segue um exemplo:

```
Sub fechaDocAtivo
   oDoc = ThisComponent
   If HasUnoInterfaces(oDoc, "com.sun.star.util.XCloseable") Then
     oDoc.close(True)
   Else
     oDoc.dispose()
   End If
End Sub
```

8.12 Identificando documentos abertos

Já sabemos que o objeto Desktop é o encarregado da manutenção dos componentes carregados no OO.o. Entretanto, 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 apenas dois são documentos. Para identificar os documentos abertos precisamos lidar com este detalhe. Segue um exemplo:

```
Sub exibeComponentes
   ' enumera os componentes carregados
   oComponentes = StarDesktop.getComponents().createEnumeration()
   n = 0
   ' visita cada um deles
   Do While (oComponentes.hasMoreElements())
      oComp = oComponentes.nextElement()
      ' nem todos componentes são modelos
      If HasUnoInterfaces(oComp, "com.sun.star.frame.XModel") Then
         ' obtém o titulo da janela
         sTituloJanela = oComp.getCurrentController().getFrame().Title
         ' identifica o tipo de componente
         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.presentation.PresentationDocument") Then
            sTipo = "Presentation"
         ElseIf oComp.SupportsService("com.sun.star.drawing.DrawingDocument") Then
            sTipo = "Draw"
         ElseIf oComp.SupportsService("com.sun.star.text.WebDocument") Then
            sTipo = "HTML"
         ElseIf oComp.SupportsService("com.sun.star.formula.FormulaProperties") Then
            sTipo = "Math"
```

```
         ElseIf oComp.SupportsService("com.sun.star.script.BasicIDE") Then
            sTipo = "Basic"
         Else
            sTipo = "Desconhecido"
         End If
         msg = msg & sTipo & ": -> "& sTituloJanela & Chr$(10)
      Else
         ' componente nao tem um modelo (ex: navegador de fonte de dados)
         msg = msg & "Componente sem modelo" & Chr$(10)
      End If
      n = n + 1
   Loop
   MsgBox msg, 176, "Componentes carregados: " + Str$(n)
End Sub
```

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.

9 Documentos do Writer

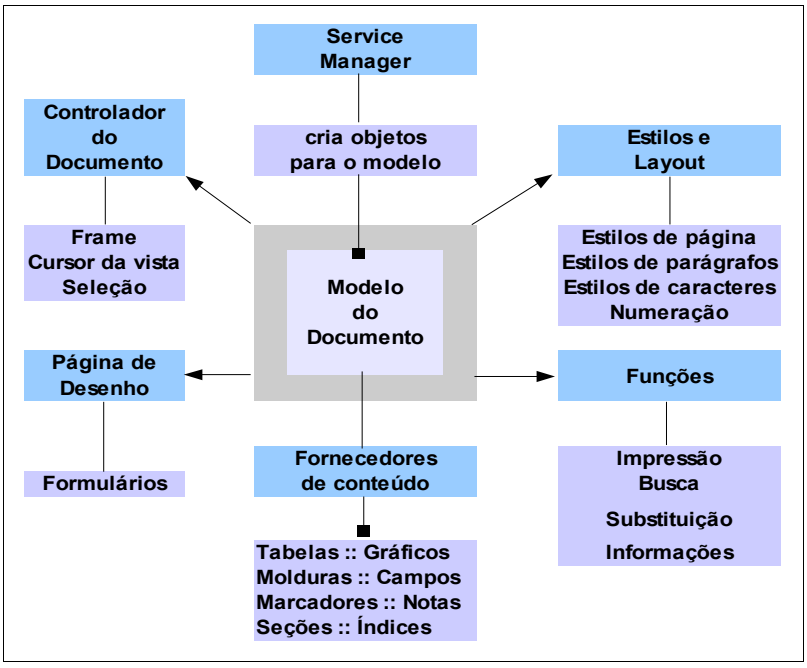
Neste capítulo, veremos como escrever macros para documentos do Writer usando a API.

9.1 Introdução

Podemos considerar um documento do Writer como um repositório de dados. Estes dados são representados por diferentes tipos de objetos, cada um com suas próprias características. Seguem os principais objetos encontrados em documentos de texto:

Parágrafos no documento ou como parte de outro objeto, por exemplo num quadro ou tabela
Informações sobre os estilos de formatação (caracteres, parágrafos, páginas, etc)
Definições de configuração do documento
Tabelas contendo texto, valores ou gráficos
Formulários e controles como botões e caixas de texto, entre outros
Gráficos e desenhos como linhas, polígonos e símbolos
Campos de vários tipos, como: função, documento e banco de dados

Eis uma representação visual incompleta, adaptada do **The Developers Guide**, da organização de um documento texto.



Os objetos que usam uma forma de desenho são colocados sobre a **página de desenho** do documento. Em algumas operações, o acesso a objetos deste tipo (por ex: **formulários**) se dá através do objeto página de desenho.

Normalmente, os objetos num documento são armazenados como uma coleção, alguns devem estar ancorados num parágrafo, num caractere ou numa página. Conforme o objeto, podemos usar o acesso nomeado, indexado ou enumerado, como já explicado.

Os principais serviços encontram-se no módulo **com.sun.star.text**. Seguem alguns dos serviços básicos para operar com documentos de texto:

```
com.sun.star.text.TextDocument => representa o modelo do documento
com.sun.star.text.TextRange => representa uma extensão de texto no documento
com.sun.star.text.Text => representa todo o conteúdo de texto do documento
com.sun.star.text.TextCursor => representa um cursor para navegar pelo documento
com.sun.star.text.TextTable => representa uma tabela no documento
com.sun.star.text.TextField => representa um campo no documento
com.sun.star.text.Paragraph => representa um parágrafo no documento
```

9.2 Documento Texto

Um documento do Writer possui dois objetos principais o modelo e o controlador.

Modelo

O modelo de um documento texto é representado pelo serviço **TextDocument** que inclui o serviço **com.sun.star.text.TextGenericDocument**, pois outros documentos usam este serviço (HTML e Documento Mestre). Este, por sua vez, inclui o serviço OfficeDocument.

Entre as propriedades de TextGenericDocument, temos:

```
com.sun.star.lang.Locale CharLocale => identifica a localização do documento
long WordCount => [read-only] palavras existentes no documento
long CharacterCount => [read-only] caracteres existentes no documento
long ParagraphCount => [read-only] parágrafos existentes no documento
string WordSeparator => caracteres separadores para a contagem de palavras
```

Todas as propriedades são opcionais. A estrutura Locale possui os campos abaixo:

```
string Language => identifica a linguagem (ISO-639)
string Country => identifica o país (ISO-3166)
string Variant => para definir informações específicas do produto
```

O objeto TextGenericDocument implementa várias interfaces, contendo métodos para recuperar objetos existentes no documento ou para executar alguma operação sobre o documento. Seguem algumas destas interfaces e os principais métodos:

com.sun.star.text.XTextDocument:

```
com.sun.star.text.XText  getText ( )
```

retorna o objeto com.sun.star.text.Text, que contém o texto principal do documento. Texto em quadros e células de tabelas não estão incluídos, para obter o texto destes objetos, este método deve ser chamado a partir dos mesmos.

```
void  reformat ( )
```

reformata o documento

com.sun.star.util.XRefreshable:

```
void  refresh ( )
```

reabastece dados de uma fonte de dados ligada ao documento

```
void  addRefreshListener ( com.sun.star.util.XRefreshListener  xListener )
```

```
void   removeRefreshListener (  com.sun.star.util.XRefreshListener  xListener )
```
adiciona ou remove o cadastro de um listener XRefreshListener, cujo método é:

```
void   refreshed ( com.sun.star.lang.EventObject  oEvento )
```
chamado durante o evento refresh

A interface **com.sun.star.text.XPagePrintable** lida com a impressão de múltiplas páginas numa única folha, os seus métodos são:

```
< com.sun.star.beans.PropertyValue >   getPrintPageSettings ( )
void   setPrintPageSettings ( < com.sun.star.beans.PropertyValue >   vProp )
```
obtém ou define propriedades para impressão de múltiplas páginas, entre as quais temos:

```
short PageRows => número de linhas por páginas
short PageColumns => número de colunas por página
boolean IsLandscape => se True imprime como paisagem, senão como retrato
long LeftMargin => margem esquerda da página
long RightMargin => margem direita da página
long TopMargin => margem superior da página
long BottomMargin => margem inferior da página
long HoriMargin => margem entre as linhas
long VertMargin => margem entre as colunas
```

```
void  printPages ( < com.sun.star.beans.PropertyValue >   vOpcoes )
```
imprime as páginas, as opções são as mesmas de com.sun.star.view.PrintOptions

As interfaces com.sun.star.util.XSearchable e com.sun.star.util.XReplaceable lidam com a pesquisa e substituição de texto no documento. Além destas, existem as interfaces **fornecedoras de conteúdo**, nomeadas como **XYyyySupplier**, que lidam com o acesso a coleções de objetos do documento (XFootnotesSupplier, XBookmarksSupplier, XTextSectionsSupplier, etc). Algumas serão apresentadas em seções próprias.

Segue um exemplo de acesso aos dados de um documento:

```
Sub exemploDocumentoTexto
   ' obtém o modelo do documento
   oDoc = thisComponent
   ' obtém a linguagem do documento
   vLocal = oDoc.CharLocale
   ' exibe o conteúdo dos campos
   msg = vLocal.Language & " " & vLocal.Country & " " & vLocal.Variant
   msgBox msg, 176, "Linguagem do documento"
   ' obtém o texto do documento
   oTexto = oDoc.getText()
   ' exibe as interfaces do objeto
   msgBox oTexto.Dbg_SupportedInterfaces()
   ' exibe o texto (cadeia) no objeto
   msgBox oTexto.String, 176, "Texto do documento"
End Sub
```

Antes de rodar a macro, digite algo no documento.

Controlador

Todo modelo possui pelo menos um controlador, responsável pela apresentação visual dos dados. O serviço com.sun.star.text.TextDocumentView permite o acesso ao cursor da visão, às definições da janela de exibição e ao estado da seleção. Eis algumas das suas propriedades:

```
long PageCount => número de páginas do documento
long LineCount => número de linhas do documento
```

Entre as suas interfaces temos:

com.sun.star.view.XViewSettingsSupplier, com o método:

```
com.sun.star.beans.PropertySet  getViewSettings ( )
```
obtém as propriedades (com.sun.star.text.ViewSettings) da janela de exibição, entre elas:

```
boolean  ShowRulers => se True, exibe as réguas
boolean  ShowHoriRuler => se True, exibe a régua horizontal
boolean  ShowVertRuler => se True, exibe a régua vertical
boolean  ShowHoriScrollBar => se True, exibe a barra de rolagem horizontal
boolean  ShowVertScrollBar => se True, exibe a barra de rolagem vertical
short  ZoomType => define o tipo do “zoom”
short  ZoomValue => define um valor (percentual) para o “zoom”
boolean  ShowHiddenText => se True, exibe texto oculto
boolean  ShowHiddenParagraphs => se True, exibe parágrafos ocultos
boolean  ShowTables => se True, exibe as tabelas
boolean  ShowGraphics => se True, exibe os gráficos
```

com.sun.star.view.XTextViewCursorSupplier, com o método:

```
com.sun.star.text.XTextViewCursor  getViewCursor ( )
```
retorna o cusor da visão, isto é, o cursor que o usuário vê na janela de exibição

Todos os controladores de documentos do OOffice.org incluem o serviço com.sun.star.view.OfficeDocumentView, com a interface com.sun.star.view.XSelectionSupplier e seus métodos:

```
boolean  select ( any oObjeto)
```
seleciona o objeto na janela de exibição, se não for possível retorna False

```
any  getSelection ( )
```
retorna um objeto ou uma coleção com os objetos selecionados

```
void addSelectionChangeListener (XSelectionChangeListener oListener)
```
cadastra um listener de mudança na seleção, com.sun.star.view.XSelectionChangeListener

```
void removeSelectionChangeListener ( XSelectionChangeListener oListener )
```
remove o cadastro do listener de mudança na seleção

Para acelerar a execução do código, podemos bloquear a atualização da visão, usando os métodos de XModel:

```
void  lockControllers ( )
```
bloqueia os controladores do documento (impede atualização da visão)

```
void  unlockControllers ( )
```
desbloqueia os controladores

```
boolean  hasControllersLocked ( )
```
retorna True se os controladores estiverem bloqueados

Vejamos um exemplo usando o objeto controlador:

```
Sub exemploControlador
   ' obtem o modelo
```

```
   oDoc = ThisComponent
   ' obtém o controlador
   oControlador = oDoc.getCurrentController()
   ' exibe o número de linhas
   msgBox Str(oControlador.LineCount), 176, "Número de Linhas:"
   ' obtém as propriedades da janela de exibição
   oViewSettings = oControlador.getViewSettings()
   ' exibe o estado da régua vertical
   msgBox oViewSettings.ShowVertRuler, 176, "Exibição da régua vertical"
End Sub
```

9.3 Edição Básica

Existem três objetos diretamente envolvidos na edição do conteúdo de um documento: Text, TextRange e TextContent. Segue uma apresentação de cada um deles.

Objeto Text

O objeto com.sun.star.text.Text representa o conteúdo textual de outro objeto. Por exemplo, o conteúdo de um ou mais parágrafos, de uma célula numa tabela ou de uma moldura. Por si só ele é bastante limitado, mas deve ser usado para inserir ou remover conteúdo no documento, criar objetos **TextRange** e **TextCursor** para a formatação ou navegação pelo documento.

Para obter este objeto podemos chamar o método getText ( ) como em:

```
oTexto = thisComponent.getText ( )
```

oTexto é um objeto com.sun.star.text.Text contendo as propriedades:

```
com.sun.star.beans.PropertyValues  StartRedline
com.sun.star.beans.PropertyValues  EndRedline
```

PropertyValues é uma sequência da estrutura PropertyValue com várias propriedades.

O objeto Text implementa várias interfaces, entre as quais:

com.sun.star.text.XText, contendo os métodos:

```
void insertTextContent (XTextRange oPos, XTextContent oCont, boolean bFlag)
```

insere oCont na posição oPos. Se bFlag for True o conteúdo de oPos será substituído

```
void removeTextContent ( XTextContent oCont )
```

remove o conteúdo de oCont do objeto

Nota: os métodos acima não devem ser usados com parágrafos.

Métodos herdados de **com.sun.star.text.XSimpleText**

```
com.sun.star.text.XTextCursor  createTextCursor ( )
```

cria um cursor de texto para navegar pelo objeto

```
com.sun.star.text.XTextCursor createTextCursorByRange ( XTextRange  oExtensao )
```

cria um cursor de texto abrangendo a extensão

```
void  insertString ( XTextRange  oPos, string sCadeia, boolean bSubst )
```

insere a cadeia na posição oPos. Se bSubst for True substitui o conteúdo de oPos

```
void  insertControlCharacter ( XTextRange  oPos, short iChar, boolean bSubst )
```

insere o caractere na posição oPos. Se bSubst for True substitui o conteúdo de oPos

Um caracter de controle cumpre alguma função no texto, estão definidos no grupo de constantes **com.sun.star.text.ControlCharacter**:

```
PARAGRAPH_BREAK => inicia um novo parágrafo
LINE_BREAK => inicia uma nova linha no mesmo parágrafo
HARD_HYPHEN => hífen que impede a hifenização nesta posição
SOFT_HYPHEN => define uma posição para a  hifenização
HARD_SPACE => impede a hifenização entre duas palavras
APPEND_PARAGRAPH => adiciona um novo parágrafo
```

A *quebra de página* é uma propriedade do parágrafo (não há caractere de controle)

Nota1: todos os métodos insertXXX usam um indicador (TextRange) da posição

Nota2: XText herda, também, os métodos com.sun.star.text.XTextRange.

com.sun.star.text.XTextRangeCompare, cujos métodos são:

```
short  compareRegionStarts ( XTextRange oExt1, XTextRange oExt2 )
```

compara o início de oExt1 com o início de oExt2

```
short  compareRegionEnds ( XTextRange oExt1, XTextRange oExt2 )
```

compara o final de oExt1 com o final de oExt2.

Em ambos os casos o valor retornado pode ser:

```
 1: se oExt1 antes de oExt2
 0: se a posição for a mesma
-1: se oExt1 depois de oExt2
```

com.sun.star.text.XTextRangeMover, contendo o método:

```
void  moveTextRange ( XTextRange oConteudo, short nParagrafos )
```

move o conteúdo nParagrafos para frente ou para trás

Ainda, a interface **com.sun.star.container.XEnumerationAccess** e seu método **createEnumeration** permite a enumeração de um objeto Text. Por exemplo, para percorrer parágrafos e porções de parágrafos (consulte o item *Parágrafos*).

Objeto TextRange

O objeto com.sun.star.text.TextRange representa uma parte de um objeto Text e nos oferece mais flexibilidade para lidar com o texto. Podemos, por exemplo, formatá-lo ou navegar pelo seu conteúdo. Este objeto é utilizado, também, como âncora para outros objetos.

Entre os serviços incluídos, temos CharacterProperties e ParagraphProperties, que definem uma série de propriedades para a formatação de caracteres e parágrafos (consulte, adiante, o tópico *Formatação*). Por conta destes serviços, TextRange também implementa as interfaces XPropertySet e XPropertyState.

A interface XTextRange oferece os métodos abaixo:

```
com.sun.star.text.XText  getText ( )
```

retorna o objeto Text do TextRange (note que TextRange também tem o método getText)

```
string  getString ( )
```

retorna a cadeia de caracteres do objeto

`void setString ( string sCadeia )`
define uma cadeia de caracteres para o objeto, substitui todo o conteúdo do objeto

`com.sun.star.text.XTextRange getStart ( )`
retorna uma extensão (TextRange) indicando o início do objeto

`com.sun.star.text.XTextRange getEnd ( )`
retorna uma extensão (TextRange) indicando o final do objeto

Um TextRange possui algumas propriedades que nos permite identificar em qual objeto ele está localizado:

TextField => retorna o objeto campo de texto do TextRange
TextFrame => retorna o objeto quadro do TextRange
TextSection => retorna o objeto seção do TextRange
TextTable => retorna o objeto tabela do TextRange

Em todos os casos, retorna um **objeto vazio** se o texto não se referir a um deles

Já conhecemos os elementos básicos para editar texto, então vamos a um exemplo:

```
Sub editaTexto
    ' obtém o objeto modelo do documento
   oDoc = ThisComponent
   ' obtém o conteúdo textual do modelo
   ' ( um objeto Text )
   oTxt = oDoc.getText()
   ' define uma cadeia
   sStr = "Esta é a primeira linha do primeiro parágrafo"
   ' insere a cadeia no objeto Text
   ' (XText herda setString de XTextRange)
   oTxt.setString ( sStr )
   ' para usar os métodos insertXXX precisamos de um objeto
   ' TextRange indicando a posição da inserção
   ' obtém um no final do texto ( XText herda getEnd de XTextRange )
   oTxtRng = oTxt.getEnd()
   ' insere um caractere de controle (LINE_BREAK) no final
   quebraLinha = com.sun.star.text.ControlCharacter.LINE_BREAK
   oTxt.insertControlCharacter ( oTxtRng, quebraLinha, False )
   ' objeto TextRange tem propriedades para formatação
   oTxtRng.CharColor = RGB (255,10,10)
   sStr = "Esta é a segunda linha do primeiro parágrafo"
   ' insere texto no TextRange
   oTxtRng.setString ( sStr )
   'muda a cor
   oTxtRng.CharColor = RGB(0,0,255)
   ' insere um caractere de controle (PARAGRAPH_BREAK) no final
   quebraPara = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
   ' os métodos insertXXX são do objeto Text, por isto antes de usá-lo
   ' recupere o objeto Text de TextRange
   oTxtRng.getText().insertControlCharacter(oTxtRng.getEnd(), quebraPara, False)
   sStr = "Este é o segundo parágrafo"
   ' insere texto usando insertString
   oTxtRng.getText().insertString(oTxtRng.getEnd(), sStr, False)
End Sub
```

Os passos usados numa edição básica são:

obter o objeto contendo o texto a ser editado
chamar o método getText ( )
usar um dos métodos de XText para editar o conteúdo, lembrando que:

setString substitui todo conteúdo do objeto (inclusive tabelas, molduras, etc)
insertXXX precisa de um TextRange indicando a posição da inserção

Agora, identifique cada um deles no exemplo acima.

Objeto TextContent

Já vimos que um documento pode conter quadros, tabelas, desenhos e campos, que devem ser ancorados sobre o conteúdo textual. Estes objetos podem conter seus próprios objetos Text, além de outras características. Todos suportam o serviço com.sun.star.text.TextContent, que é utilizado para inserir e remover os mesmos do documento.

Eis as propriedades de TextContent:

com.sun.star.text.TextContentAnchorType AnchorType
define o modo como o texto será preso ao objeto com.sun.star.drawing.Text

< com.sun.star.text.TextContentAnchorType > AnchorTypes
define o tipo de âncora do conteúdo de texto

Os valores da enum **com.sun.star.text.TextContentAnchorType** são:

AT_PARAGRAPH => objeto ancorado no canto superior esquerdo do parágrafo
AS_CHARACTER => objeto ancorado como um caractere (altera altura da linha)
AT_PAGE => objeto ancorado na página (posição não altera na edição)
AT_FRAME => objeto ancorado num quadro de texto
AT_CHARACTER => objeto ancorado num caractere (posição altera na edição)

com.sun.star.text.WrapTextMode TextWrap
define se o texto está numa forma e como ele vai envolver a mesma

Os valores da enum **com.sun.star.text.WrapTextMode** são:

NONE => texto não flui em torno do objeto
THROUGHT => fluxo do texto ignora o objeto
PARALLEL => flui a esquerda e direita do objeto
DYNAMIC => depende da formatação
LEFT => flui do lado esquerdo do objeto
RIGHT => flui do lado direito do objeto

A interface **com.sun.star.text.XTextContent** tem os métodos:

void attach (XTextRange oConteudo)
embute o objeto num texto (este método não foi implementado ?? confirmar)

XTextRange getAnchor ( )
retorna a extensão aonde o objeto está ancorado

Métodos herdados de **com.sun.star.lang.XComponent**

void dispose ( )
libera todos os recursos alocados pelo objeto

void addEventListener (com.sun.star.lang.XEventListener oListener)
cadastra um listener para o objeto

void removeEventListener (com.sun.star.lang.XEventListener oListener)
remove o cadastro do listener para o objeto

Vejamos um exemplo de edição de TextContent (insere uma tabela e uma moldura):

```
Sub editaTextContent
   ' obtém o objeto modelo do documento
   oDoc = ThisComponent
   ' obtém o objeto Text do documento
   oTxt = oDoc.getText()
   ' cria um objeto Tabela com o ServiceManager
   oTabela = oDoc.createInstance("com.sun.star.text.TextTable")
   ' inicializa o objeto tabela (tres colunas e tres linhas)
   oTabela.initialize( 3, 3 )
   ' obtém a posicao da insercao
   oTxtRng = oTxt.getEnd()
   ' insere a tabela
   oTxt.insertTextContent (oTxtRng, oTabela, False)
   ' cria uma moldura (objeto TextFrame)
   oMoldura = oDoc.createInstance("com.sun.star.text.TextFrame")
   ' cria uma estrutura Size para dimensionar a moldura
   vDim = CreateUNOStruct ("com.sun.star.awt.Size")
   vDim.Width = 10000
   vDim.Height = 5000
   ' define o tamanho da moldura
   oMoldura.setSize (vDim)
   ' define o tipo de ancora
   tipoAncora = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
   oMoldura.setPropertyValue("AnchorType", tipoAncora)
   ' insere o objeto TextFrame no documento
   oTxt.insertTextContent( oTxtRng.getEnd(), oMoldura, False)
   ' obtem o objeto Text da Moldura
   oTxtMoldura = oMoldura.getText()
   ' insere uma cadeia no objeto texto da moldura
   oTxtMoldura.setString("Texto da moldura!")
End Sub
```

Analise cuidadosamente o código, vários conceitos são exemplificados. Procure identificar os passos da edição.

9.4 Edição Avançada

Para navegar pelo conteúdo do documento, a API usa os objetos cursor de texto, cursor da visão e cursor de tabela. Nesta seção veremos os dois primeiros, o último será apresentado posteriormente.

Cursor de Texto

Com este objeto podemos acessar o conteúdo de outro objeto (por ex: Text ou TextRange), de modo independente da representação visual do mesmo. Vários cursores podem ser criados para um mesmo objeto ou para diferentes objetos. Um cursor tem um início e um final e a sua extensão pode variar, abrangendo parte ou todo o conteúdo do objeto. Finalmente, este cursor não tem nenhuma ligação com o cursor visível na tela (cursor da visão).

Para criar um cursor de texto, chamamos um dos métodos abaixo, da interface XSimpleText:

`com.sun.star.text.XTextCursor createTextCursor ( )`
cria um cursor de texto para navegar pelo objeto

```
com.sun.star.text.XTextCursor  createTextCursorByRange ( XTextRange  oExtensao )
```
cria um cursor de texto abrangendo a extensão

O serviço **com.sun.star.text.TextCursor** inclui o serviço **TextRange** (que permite a formatação de caracteres e parágrafos) e implementa as seguintes interfaces:

Interface **com.sun.star.text.XTextCursor**:

```
boolean  goLeft ( short nChar, boolean bSel )
```
move o cursor n_caracteres para a esquerda, se bSel True seleciona os n_caracteres

```
boolean  goRight ( short nChar, boolean bSel )
```
move o cursor n_caracteres para a direita, se bSel True seleciona os n_caracteres

```
void  gotoStart( boolean bSel )
```
move o cursor para o início, se bSel True seleciona

```
void  gotoEnd ( boolean bSel )
```
move o cursor para o final, se bSel True seleciona

```
void  gotoRange( XTextRange oRng, boolean bSel )
```
move o cursor para o TextRange, se bSel True seleciona

```
void  collapseToStart( )
```
coincide o final do cursor com o seu início

```
void  collapseToEnd( )
```
coincide o início do cursor com o seu final

```
boolean  isCollapsed( )
```
retorna True se o início e o final do cursor coincidirem

Interface **com.sun.star.text.XWordCursor**:

```
boolean  isStartOfWord ( )
```
retorna True se o cursor estiver no início de uma palavra

```
boolean  isEndOfWord ( )
```
retorna True se o cursor estiver no início de uma palavra

```
boolean  gotoNextWord ( boolean bSel )
```
move o cursor para a próxima palavra, retorna False se não for possível

```
boolean  gotoPreviousWord ( boolean bSel )
```
move o cursor para a palavra anterior, retorna False se não for possível

```
boolean  gotoStartOfWord ( boolean bSel )
```
move o cursor para o início da palavra, retorna False se não for possível

```
boolean  gotoEndOfWord ( boolean bSel )
```
move o cursor para o final da palavra, retorna False se não for possível

Nota: Em todos os casos, se bSel for True o cursor será estendido.

Interface **com.sun.star.text.XSentenceCursor**:

```
boolean isStartOfSentence ( )
```
retorna True se o cursor está no início de uma sentença

```
boolean isEndOfSentence ( )
```
retorna True se o cursor está no final de uma sentença

```
boolean gotoNextSentence ( boolean  bSel )
```

move o cursor para a próxima sentença, retorna False se não for possível

`boolean gotoPreviousSentence ( boolean bSel )`
move o cursor para a sentença anterior, retorna False se não for possível

`boolean gotoStartOfSentence ( boolean bSel )`
move o cursor para o início da sentença, retorna True se foi possível

`boolean gotoEndOfSentence ( boolean bSel )`
move o cursor para o final da sentença, retorna True se foi possível

Nota: Em todos os casos, se bSel for True o cursor será estendido.

Interface **com.sun.star.text.XParagraphCursor**:

`boolean isStartOfParagraph ( )`
retorna True se o cursor está no início de um parágrafo

`boolean isEndOfParagraph ( )`
retorna True se o cursor está no final de um parágrafo

`boolean gotoNextParagraph ( boolean bSel )`
move o cursor para o próximo parágrafo, retorna False se não for possível

`boolean gotoPreviousParagraph ( boolean bSel )`
move o cursor para o parágrafo anterior, retorna False se não for possível

`boolean gotoStartOfParagraph ( boolean bSel )`
move o cursor para o início do parágrafo, retorna True se foi possível

`boolean gotoEndOfParagraph ( boolean bSel )`
move o cursor para o final do parágrafo, retorna True se foi possível

Nota: Em todos os casos, se bSel for True o cursor será estendido.

Interface **com.sun.star.document.XDocumentInsertable**:

`void insertDocumentFromURL ( string sURL, < PropertyValue > vOpcoes )`
insere o documento definido pelo URL, conforme as opções, na posição do cursor

Interface **com.sun.star.util.XSortable** (consulte o tópico *Ordenando Texto*):

`< com.sun.star.beans.PropertyValue > createSortDescriptor ( )`
retorna um descritor de ordenação, as propriedades dependem do objeto

`void sort ( )`
ordena o conteúdo segundo as propriedades do descritor

Um cursor de texto tem, ainda, à sua disposição, todas as propriedades e métodos do objeto TextRange.

Vejamos um exemplo:

```
Sub exemploCursorTexto
   ' obtém o aplicativo
   oDesktop = StarDesktop
   ' carrega um novo documento
   oDoc = oDesktop.loadComponentFromURL("private:factory/swriter",_
          "_blank", 0, Array())
   ' cria uma instância do objeto Text do documento
   oTexto = oDoc.getText()
   ' cria um cursor de texto para o objeto Text
```

```
   oCursor = oTexto.createTextCursor()
   sStr = "Este é o texto do primeiro parágrafo do documento"
   quebraParagrafo = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
   ' insere dados no objeto Text ( um cursor é um TextRange )
   With oTexto
      .insertString(oCursor, sStr, False)
      .insertControlCharacter(oCursor, quebraParagrafo, False)
      .insertString(oCursor, "Este é o segundo parágrafo ", False)
      .insertControlCharacter(oCursor, quebraParagrafo, False)
   End With
   ' move para o início do parágrafo, selecionando
   oCursor.gotoStartOfParagraph(True)
   ' insere uma quebra de página (é uma propriedade do parágrafo)
   oCursor.breakType = com.sun.star.style.BreakType.PAGE_BEFORE
   ' insere texto na nova página
   oTexto.insertString(oCursor, "Estou numa nova página.", False)
   ' move o cursor para o início do texto, sem expansão
   oCursor.gotoStart(False)
   ' avança as três primeiras palavras
   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
   ' usando um método de XTextRange
   MsgBox oCursor.getString(), 176, "Conteúdo do cursor"
   oCursor.setString("")
   MsgBox oCursor.getString(), 176, "Conteúdo do cursor"
   ' move para o próximo parágrafo sem selecionar
   oCursor.gotoNextParagraph(False)
   ' seleciona todo o parágrafo
   oCursor.gotoEndOfParagraph(True)
   ' formata a cor dos caracteres,
   ' usando uma propriedade de TextRange
   oCursor.CharColor = RGB(0,0,255)
End Sub
```

Note que com um TextCursor você pode mover-se pelos dados e, ainda, editá-los.

Cursor da Visão

O serviço **com.sun.star.text.TextViewCursor** representa o **cursor visível** na janela de exibição do documento e possui facilidades de navegação e edição. Deve ser usado para paginação e operações sobre linhas de texto.

Para criar este objeto, devemos fazer:

```
oCursorVisao = thisComponent.getCurrentController().getViewCursor()
```

Este objeto suporta, também, as interfaces XTextCursor e XTextRange. Podemos criar um cursor de texto a partir do cursor da visão, como abaixo:

```
oCursorTexto = oText.createTextCursorByRange ( oCursorVisao.getStart() )
```

Entre as interfaces de TextViewCursor, temos:

com.sun.star.text.XTextViewCursor com os métodos:

```
boolean  isVisible ( )
```
retorna True se o cursor estiver visível

```
void  setVisible ( boolean bFlag )
```
se bFlag for True, o cursor fica visível, se False invisível

```
com.sun.star.awt.Point  getPosition ( )
```
retorna as coordenadas do cursor, relativas ao canto superior esquerdo da primeira página

A estrutura **com.sun.star.awt.Point** possui os campos:

```
long X => coordenada X
long Y => coordenada Y
```

com.sun.star.view.XScreenCursor com os métodos:

```
boolean  screenDown ( )
boolean  screenUp ( )
```
avança ou retrocede uma página da tela

com.sun.star.text.XPageCursor com os métodos:

```
boolean jumpToFirstPage ( )
```
salta para a primeira página

```
boolean jumpToLastPage ( )
```
salta para a última página

```
boolean jumpToPage( short  nPag )
```
salta para a página nPag

```
short getPage ( )
```
retorna o número da página corrente

```
boolean jumpToNextPage ( )
```
salta para a próxima página

```
boolean jumpToPreviousPage ( )
```
salta para a página anterior

```
boolean jumpToEndOfPage ( )
```
salta para o final da página corrente

```
boolean jumpToStartOfPage ( )
```
salta para o início da página corrente

com.sun.star.view.XViewCursor com os métodos:

```
boolean  goDown (short nLinhas, boolean  bSel)
boolean  goUp (short nLinhas, boolean  bSel)
```
move n_linhas abaixo ou acima, se bSel for True seleciona

```
boolean  goLeft (short nChar, boolean  bSel)
boolean  goRight (short nChar, boolean  bSel)
```
move n_caracteres para a esquerda ou direita, se bSel for True seleciona

com.sun.star.view.XLineCursor com os métodos:

```
void  gotoStartOfLine (boolean   bSel)
void  gotoEndOfLine (boolean   bSel)
```
move para o início / final da linha, se bSel for True seleciona

```
boolean  isAtStartOfLine ( )
boolean  isAtEndOfLine ( )
```
retorna True se estiver no início / final da linha

Vejamos um exemplo usando o cursor da visão:

```
Sub exemploCursorVisao
   ' obtém o aplicativo
   oDesktop = StarDesktop
   ' carrega um novo documento
   oDoc = oDesktop.loadComponentFromURL("private:factory/swriter",_
          "_blank", 0, Array())
   ' obtém o controlador
   oControlador = oDoc.getCurrentController()
   ' obtém o cursor da visão
   oCVisao = oControlador.getViewCursor()
   sStr = "Este é o texto do primeiro parágrafo do documento"
   quebraParagrafo = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
   ' cursor da visão pode usar métodos de XTextRange
   ' insere uma cadeia
   oCVisao.setString( sStr )
   MsgBox "Cadeia inserida"
   ' cursor da visão pode usar métodos de XTextCursor
   ' remove seleção (após inserir a cadeia ela fica selecionada)
   oCVisao.collapseToEnd()
   ' obtém objeto Text, precisamos dele para usar os métodos
   ' de XText e XSimpleText ( cursor da visão não implementa
   ' estas interfaces )
   oTxt = oCVisao.getText()
   oTxt.insertControlCharacter(oCVisao, quebraParagrafo, False)
   ' move uma linha abaixo sem selecionar
   oCVisao.goDown( 1, False )
   ' insere uma cadeia
   oCVisao.setString("Este é o segundo parágrafo ")
   ' move para o final da linha sem selecionar
   oCVisao.gotoEndOfLine(False)
   oTxt.insertControlCharacter(oCVisao, quebraParagrafo, False)
   ' insere uma quebra de página (é uma propriedade do parágrafo)
   oCVisao.breakType = com.sun.star.style.BreakType.PAGE_BEFORE
   ' agora, o cursor está na nova página
   ' insere texto
   oCVisao.setString("Estou numa nova página")
   MsgBox "Nova página"
   ' salta para a primeira página
   oCVisao.jumpToFirstPage()
   ' obtém e exibe o número da página atual
   MsgBox Str(oCVisao.getPage()), 176, "Página atual"
   ' move para o fim da linha selecionando
   oCVisao.gotoEndOfLine(True)
   ' formata caracteres
   oCVisao.CharColor = RGB (250, 50, 50)
   ' remove a seleção
   oCVisao.gotoStartOfLine(False)
End Sub
```

Analise o exemplo, observando os métodos herdados de outras interfaces.

9.5 Formatação

O OOffice.org oferece recursos avançados para formatar o conteúdo dos seus documentos. A formatação pode ocorrer de duas maneiras: (**1**) de modo direto ou (**2**) usando estilos. Na primeira, selecionamos o objeto e alteramos as propriedades desejadas, por exemplo selecionar uma palavra e alterar a cor da fonte. Na segunda, após selecionar o objeto, aplicamos um dos estilos de formatação predefinidos (para vê-los, selecione *Formatar => Estilos e Formatação*, no menu principal).

Ao aplicar formatação, lembre-se que pode existir interdependência entre propriedades.

Formatando Caracteres

O serviço **com.sun.star.style.CharacterProperties** define as propriedades para formatar caracteres, seguem as principais:

```
string CharFontName => define o nome da fonte (pode ter mais de um nome)
string CharFontStyleName => contém o nome do estilo da fonte
short CharFontFamily => define a família da fonte (com.sun.star.awt.FontFamily)
```

Eis as constantes (e valores) de com.sun.star.awt.FontFamily:

```
DONTKNOW (0)  => família desconhecida
DECORATIVE (1)  => família de fontes decorativa
MODERN (2)  => família de fontes moderna
ROMAN (3)  => família de fontes roman com serifa
SCRIPT (4)  => família de fontes de script
SWISS (5)  => família de fontes roman sem serifa
SYSTEM (6)  => família de fontes do sistema
```

```
short CharFontCharSet => define a codificação da fonte (com.sun.star.awt.CharSet)
```

Eis algumas constantes (e valores) de com.sun.star.awt.CharSet:

```
DONTKNOW (0)  => conjunto de caracteres desconhecido
ANSI (1)  =>  conjunto de caracteres ANSI
MAC (2)  =>  conjunto de caracteres MAC
IBMPC_437 (3)  =>  conjunto de caracteres IBMPC_437
IBMPC_860 (5)  =>  conjunto de caracteres IBMPC_860
SYSTEM (9)  =>  conjunto de caracteres SYSTEM
SYMBOL (10)  =>  conjunto de caracteres SYMBOL
```

```
short CharFontPitch => define a densidade horizontal da fonte (com.sun.star.awt.FontPitch)
```

Eis as constantes (e valores) de com.sun.star.awt.FontPitch:

```
DONTKNOW (0)  => densidade horizontal desconhecida
FIXED (1)  =>  densidade horizontal fixa
VARIABLE (2)  =>  densidade horizontal variável
```

```
com.sun.star.util.Color CharColor => define o valor da cor da fonte
```

o tipo com.sun.star.util.Color é um **typedef** para um valor longo [long].

Cores são compostas por quatro bytes ARGB (alpha/red/green/blue: 0xAARRGGBB). Componentes iniciais podem ser omitidos se forem zero (red: 0xFF0000 – green: 0xFF00 – blue: 0xFF). Com o OOoBasic, use a função RGB para definir cores.

```
float CharHeight => define a altura dos caracteres em pontos
short CharUnderline => define o valor para o caractere de sublinhado (awt.FontUnderline)
```

Eis algumas constantes (e valores) de com.sun.star.awt.FontUnderline:

```
NONE (0)  => nenhum sublinhado
SINGLE (1)  =>  linha simples no sublinhado
DOUBLE (2)  =>  linha dupla no sublinhado
DOTTED (3)  =>  linha pontilhada no sublinhado
DASH (5)  =>  linha tracejada no sublinhado
DASHDOT (7)  =>  linha com traços e pontos no sublinhado
WAVE (10)  =>  linha ondulante no sublinhado
```

```
float CharWeight => define o valor do peso da fonte (com.sun.star.awt.FontWeight)
```

Eis algumas constantes (e valores) de com.sun.star.awt.FontWeight:

```
THIN (50.00)  => peso de 50%
LIGHT (75.00)  => peso de 75%
NORMAL (100.00)  => peso de 100%
BOLD (150.00)  => peso de 150%
ULTRABOLD (175.00)  => peso de 175%
BLACK (200.00)  => peso de 200%
```

```
com.sun.star.awt.FontSlant CharPosture => define a postura (inclinação) da fonte
```

Eis alguns valores da **enum** com.sun.star.awt.FontSlant:

```
NONE  => sem postura
OBLIQUE  => postura obliqua
ITALIC  => postura itálica
REVERSE_OBLIQUE  => postura obliqua reversa
REVERSE_ITALIC  => postura itálica reversa
```

```
com.sun.star.util.Color CharBackColor => [opcional] define o valor da cor de fundo
com.sun.star.lang.Locale CharLocale => contém o valor do conjunto de caracteres
com.sun.star.util.Color CharUnderlineColor => define o valor da cor para o sublinhado
boolean CharUnderlineHasColor => se True, o sublinhado será colorido
boolean CharHidden => [opcional] se True os caracteres não são exibidos
```

Vejamos um exemplo de formatação de caracteres:

```
Sub exemploFormatandoCaracteres
   ' obtém o aplicativo
   oDesktop = StarDesktop
   ' carrega um novo documento
   oDoc = oDesktop.loadComponentFromURL("private:factory/swriter",_
          "_blank", 0, Array())
   ' cria uma instância do objeto Text do documento
   oTexto = oDoc.getText()
   ' cria um cursor de texto para o objeto Text
   oCursor = oTexto.createTextCursor()
   ' define a fonte
   oCursor.CharFontName = "Verdana"
   ' define a altura da fonte em pontos
   oCursor.CharHeight = 20.00
   ' insere uma cadeia que será formatada
   oCursor.setString("Fonte Verdana Height 20 "
   oCursor.collapseToEnd()
   ' define a altura da fonte em pontos
```

```
   oCursor.CharHeight = 14.00
   ' define o peso, note a referencia a constante
   ' não aconselhável mas pode-se usar o valor da mesma (150.00)
   oCursor.CharWeight = com.sun.star.awt.FontWeight.BOLD
   oCursor.setString (" BOLD ")
   oCursor.collapseToEnd()
   oCursor.CharFontName = "Times New Roman"
   oCursor.CharWeight = com.sun.star.awt.FontWeight.NORMAL
   oCursor.CharPosture = com.sun.star.awt.FontSlant.ITALIC
   oCursor.CharUnderline = com.sun.star.awt.FontUnderline.DOTTED
   ' salva a cor
   Dim lCor As Long
   lCor = oCursor.CharColor
   ' use a função RGB para definir cores
   oCursor.CharColor = RGB (255, 0, 0)
   oCursor.setString( " ITÁLICO VERMELHO SUBLINHADO" )
   oCursor.collapseToEnd()
   ' restaura a cor e a postura
   oCursor.CharColor = lCor
   oCursor.CharUnderlineHasColor = False
   oCursor.CharUnderlineColor = lCor
   oCursor.CharPosture = com.sun.star.awt.FontSlant.NONE
   oCursor.setString( " NORMAL SUBLINHADO" )
End Sub
```

Formatando Parágrafos

O serviço **com.sun.star.style.ParagraphProperties** contém diversas propriedades para formatar parágrafos, seguem algumas:

com.sun.star.style.ParagraphAdjust ParaAdjust => define o tipo de alinhamento

eis os valores da enum com.sun.star.style.ParagraphAdjust:

LEFT => ajustamento na margem esquerda
RIGHT => ajustamento na margem direita
BLOCK => justificado, exceto a última linha
CENTER => centralizado
STRETCH => justificado, inclusive a última linha

com.sun.star.style.LineSpacing ParaLineSpacing => define o espaço entre as linhas

a estrutura com.sun.star.style.LineSpacing contém os campos:

short Mode => define o modo do espaçamento [com.sun.star.style.LineSpacingMode]
short Height => define a altura da linha, conforme o campo Mode

eis as constantes (e valores) de com.sun.star.style.LineSpacingMode:

PROP (0) => o valor da altura é proporcional
MINIMUM (1) => o valor da altura é mínimo
LEADING (2) => o valor da altura é a distância para a linha anterior
FIX (3) => o valor da altura é fixo

com.sun.star.util.Color ParaBackColor => define a cor de fundo
short ParaLastLineAdjust => define o ajustamento da última linha
long ParaLeftMargin => margem esquerda em 1/100 mm
long ParaRightMargin => margem direita em 1/100 mm
long ParaTopMargin => margem superior em 1/100 mm
long ParaBottomMargin => margem inferior em 1/100 mm

```
boolean ParaLineNumberCount => se True inclue o parágrafo na numeração de linha
long ParaLineNumberStartValue => valor inicial para a numeração de linha
string ParaStyleName => [opcional] nome do estilo do parágrafo
string PageDescName => insere uma quebra de página antes e aplicando o estilo de página
short PageNumberOffset => na quebra, define um novo valor para a numeração da página
com.sun.star.style.BreakType BreakType => [opcional] tipo de quebra aplicada
```

Nas quebras de página, o estilo da página em uso será mantido (veja *PageDescName*)

Os valores da enum com.sun.star.style.BreakType são:

```
NONE => sem quebra de coluna ou página
COLUMN_BEFORE => quebra de coluna antes
COLUMN_AFTER => quebra de coluna depois
COLUMN_BOTH => quebra de coluna antes e depois
PAGE_BEFORE => quebra de página antes
PAGE_AFTER => quebra de página depois
PAGE_BOTH => quebra de página antes e depois
```

Note que as quebras de página e coluna são propriedades do parágrafo.

```
long ParaFirstLineIndent => [opcional] define a endentação da primeira linha
short ParaVertAlignment => [opcional] define o alinhamento vertical [ParagraphVertAlign]
```

as constantes (e valores) de com.sun.star.text.ParagraphVertAlign são:

```
AUTOMATIC (0) => texto horizontal e a 90º pela linha base, a 270º centralizado
BASELINE (1) => alinhamento pela linha base
TOP (2) => alinhamento pelo topo
CENTER (3) => alinhamento pelo centro
BOTTOM (4) => alinhamento pela base
```

Agora, vamos usar algumas destas propriedades num exemplo:

```
Sub exemploFormatandoParagrafo
   ' obtém o aplicativo
   oDesktop = StarDesktop
   ' carrega um novo documento
   oDoc = oDesktop.loadComponentFromURL("private:factory/swriter",_
          "_blank", 0, Array())
   ' cria uma instância do objeto Text do documento
   oTexto = oDoc.getText()
   ' cria um cursor de texto para o objeto Text
   oCursor = oTexto.createTextCursor()
   ' define propriedades deste parágrafo (usa duas maneiras)
   oCursor.setPropertyValue("ParaStyleName", "Título")
   oCursor.setPropertyValue("ParaBackColor", RGB(150, 100, 50))
   oCursor.ParaAdjust = com.sun.star.style.ParagraphAdjust.CENTER
   oCursor.setString ("Primeiro parágrafo")
   MsgBox "Pausa"
   ' insere quebra de paragrafo
   quebraParagrafo = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
   oTexto.insertControlCharacter(oCursor.getEnd(), quebraParagrafo, False)
   ' move o cursor
   oCursor.gotoNextParagraph(False)
   ' define o nome do estilo deste parágrafo
   oCursor.ParaStyleName = "Padrão"
   ' cria uma estrutura e define seus campos
   espLinha = CreateUnoStruct("com.sun.star.style.LineSpacing")
   espLinha.Mode = com.sun.star.style.LineSpacingMode.FIX
   espLinha.Height = 1000  ' 1000/100 mm = 10 mm = 1 cm
   ' define altura da linha (usa a estrutura)
   oCursor.setPropertyValue("ParaLineSpacing", espLinha)
```

```
    ' insere texto
    sStr = ""
    For i = 0 To 10
       sStr = sStr & "Texto no segundo parágrafo. "
    Next i
    oCursor.setString(sStr)
    ' insere um parágrafo
    oTexto.insertControlCharacter(oCursor.getEnd(), quebraParagrafo, False)
    oCursor.gotoNextParagraph(False)
    ' insere uma quebra de página antes do parágrafo e
    ' altera o estilo da página para Envelope
    oCursor.setPropertyValue("PageDescName", "Envelope")
    oCursor.setString("Nova página com estilo Envelope")
End Sub
```

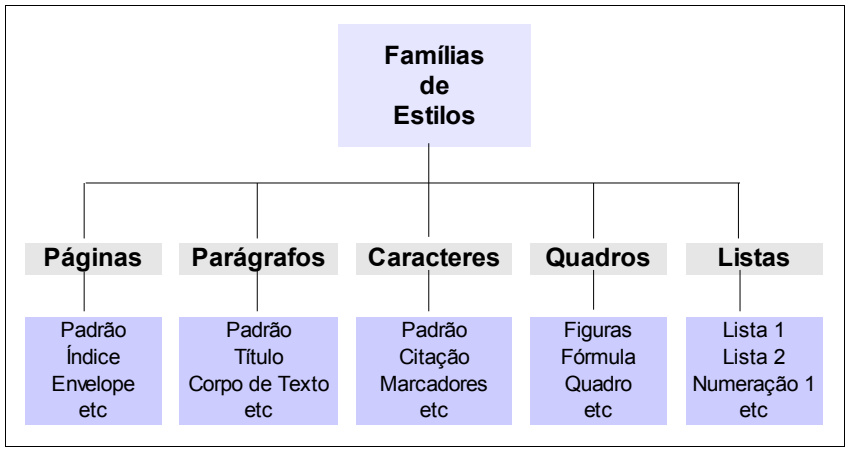
Analise o código e observe como as estruturas são definidas e usadas.

9.6 Estilos de Formatação

No módulo **com.sun.star.style** estão os principais serviços relacionados com os estilos.

Organização dos estilos

O OOffice.org organiza seus estilos de formatação em famílias. Cada família contém estilos predefinidos (os estilos padrão do OO.o e os definidos pelo usuário). Segue um diagrama da organização do estilos.



O serviço com.sun.star.style.StyleFamilies é um container para as famílias de estilos, cada família é uma coleção com.sun.star.style.StyleFamily e cada estilo da coleção suporta o serviço com.sun.star.style.Style. Com a exceção dos estilos de página e de numeração (listas), os estilos podem ser hierárquicos, isto é, possuir um estilo base vinculado.

O serviço Style possui as propriedades:

```
boolean IsPhysical => [readonly] se True, o estilo foi criado fisicamente
```

Nota: os estilos embutidos no OO.o não são criados

```
string FollowStyle => [opcional] nome do próximo estilo (só para algumas famílias)
string DisplayName => [readonly] nome exibido na interface gráfica
boolean IsAutoUpdate => define se o estilo será atualizado automaticamente (veja abaixo)
```

Nota: se um objeto formatado com o estilo for reformatado, o estilo será alterado

A interface com.sun.star.style.XStyle possui os métodos:

`boolean isUserDefined ( )`
identifica um estilo como definido pelo usuário

`boolean isInUse ( )`
determina se o estilo está sendo usado no documento

`string getParentStyle ( )`
define o nome do estilo base (apenas famílias hierárquicas)

`void setParentStyle ( string sEstiloBase )`
define o nome do estilo base (apenas famílias hierárquicas)

`string getName ( )`
`void setName ( string sNome )`
para obter / definir um nome (são métodos da interface base XNamed). Os nomes e a categoria dos estilos padrão do OOffice.org não podem ser alterados.

Ainda, Style suporta as interfaces XPropertySet, XMultiPropertySet e XMultiPropertyStates.

Acessando os estilos

A interface com.sun.star.style.XStyleFamiliesSupplier, implementada pelo modelo do documento, tem o método:

`com.sun.star.container.XNameAccess getStyleFamilies ( )`
retorna uma coleção com todas as famílias de estilos do documento

A seguir, podemos usar os métodos de acesso a coleções para obter uma determinada família e os estilos existentes na mesma.

O próximo exemplo exibe características dos estilos de cada família:

```
Sub exemploAcessoEstilos
   ' obtém a coleção de famílias de estilos
   oFamilias = thisComponent.getStyleFamilies()
   ' obtém os nomes de cada família
   sNomesFamilias = oFamilias.getElementNames()
   ' visita cada família
   For i = LBound(sNomesFamilias) To UBound(sNomesFamilias)
      ' obtém a família
      oFamilia = oFamilias.getByName(sNomesFamilias(i))
      ' obtém os nomes dos estilos da família
      sNomesEstilos = oFamilia.getElementNames()
      msg = ""
      ' visita cada estilo
      For j = LBound(sNomesEstilos) To UBound(sNomesEstilos)
         ' obtém o objeto estilo (com.sun.star.style.Style)
         oEstilo = oFamilia.getByName(sNomesEstilos(j))
         With oEstilo
            ' nome programático
            sNome = .getName()
            ' nome exibido na gui
            sGUI = .DisplayName
            ' criado fisicamente ?
            bCriado = .IsPhysical
```

```
         End With
         msg = msg & sNome & " = " & sGUI & " : " & bCriado & Chr$(10)
      Next j
      ' exibe uma mensagem
      MsgBox msg, 176, sNomesFamilias(i)
   Next i
End Sub
```

Analise o código e a saída para entender o mecanismo de estilos do OOffice.org.

Criando estilos de página

O serviço com.sun.star.style.PageStyle representa um estilo de página e inclui os serviços Style e PageProperties, o primeiro já foi apresentado, vejamos o segundo:

O serviço **com.sun.star.style.PageProperties** define várias propriedades para a formatação de páginas, seguem algumas:

```
com.sun.star.util.Color BackColor => define a cor de fundo
string BackGraphicURL => URL do gráfico para o fundo da página
string BackGraphicFilter => nome do filtro para o gráfico do fundo da página
GraphicLocation BackGraphicLocation => define o local do gráfico do fundo da página
```

eis os valores da enum com.sun.star.style.GraphicLocation (arredores):

```
NONE => local não assinalado
LEFT_TOP => no canto superior esquerdo
MIDDLE_TOP => no meio da margem superior
RIGHT_TOP => no canto superior direito
LEFT_MIDDLE => no meio da margem esquerda
MIDDLE_MIDDLE => no centro
RIGHT_MIDDLE => no meio da margem direita
LEFT_BOTTOM => no canto inferior esquerdo
MIDDLE_BOTTOM => no meio da margem inferior
RIGHT_BOTTOM => no canto inferior direito
AREA => escalado para preencher toda a área
TILED => disposto como ladrilhos sobre o objeto
```

```
long LeftMargin => margem esquerda da página
long RightMargin => margem direita da página
long TopMargin => margem superior da página
long BottomMargin => margem inferior da página
boolean IsLandscape => se True, página em paisagem
com.sun.star.style.PageStyleLayout PageStyleLayout => define a disposição da página
```

eis os valores da enum com.sun.star.style.PageStyleLayout:

```
ALL => estilo usado pelas páginas direita e esquerda
LEFT => estilo usado apenas pelas páginas esquerdas
RIGHT => estilo usado apenas pelas páginas direitas
MIRRORED => usado pelas páginas esquerdas e espelhado nas páginas direitas
```

```
long Width => define a largura da página
long Height => define a altura da página
com.sun.star.awt.Size Size => tamanho do papel da página
```

eis os campos da estrutura com.sun.star.awt.Size:

```
long Width => define a largura do objeto
long Height => define a altura do objeto
```

```
long HeaderLeftMargin => margem esquerda do cabeçalho
long HeaderRightMargin => margem direita do cabeçalho
long HeaderBodyDistance => distância entre o cabeçalho e o texto
long HeaderHeight => altura do cabeçalho
boolean HeaderIsOn => se True, ativa o cabeçalho
com.sun.star.text.XText HeaderText => objeto Text do cabeçalho
com.sun.star.text.XText HeaderTextLeft => objeto Text do cabeçalho das páginas esquerdas
com.sun.star.text.XText HeaderTextRight => objeto Text do cabeçalho das páginas direitas
long FooterLeftMargin => margem esquerda do rodapé
long FooterRightMargin => margem direita do rodapé
long FooterBodyDistance => distância entre o rodapé e o texto
long FooterHeight => altura do rodapé
boolean FooterIsOn => se True, ativa o rodapé
com.sun.star.text.XText FooterText => objeto Text do rodapé
com.sun.star.text.XText FooterTextLeft => objeto Text do rodapé das páginas esquerdas
com.sun.star.text.XText FooterTextRight => objeto Text do rodapé das páginas direitas
```

Além destas, existem propriedades relacionadas com as bordas, gráficos, notas de rodapé, etc. Para uma descrição detalhada, consulte a documentação do SDK.

Ainda, o serviço com.sun.star.text.TextPageStyle acrescenta propriedades, para documentos texto, referentes a notas de rodapé, rodapé e cabeçalho.

Já vimos como alterar o estilo da página, inserindo uma quebra. Agora, veremos como criar e aplicar um novo estilo de página:

```
Sub criandoEstiloPagina
   ' obtém o aplicativo
   oDesktop = StarDesktop
   ' carrega um novo documento
   oDoc = oDesktop.loadComponentFromURL("private:factory/swriter",_
          "_blank", 0, Array())
   ' obtém a coleção de famílias de estilos
   oFamilias = oDoc.getStyleFamilies()
   ' obtém a família de estilos de página
   oPaginas = oFamilias.getByName("PageStyles")
   If oPaginas.hasByName("Minha_Pagina_Pessoal") Then
      msgBox "Estilo já existe", 176, "Erro"
   Else
      ' cria uma instancia do novo estilo de página
      oNovoEstiloPagina = oDoc.createInstance("com.sun.star.style.PageStyle")
      ' acrescenta aah família de estilos de página do documento
      oPaginas.insertByName("Minha_Pagina_Pessoal", oNovoEstiloPagina)
      ' altera algumas propriedades
      oNovoEstiloPagina.LeftMargin = 3000
      oNovoEstiloPagina.RightMargin = 3000
      oNovoEstiloPagina.TopMargin = 3000
      oNovoEstiloPagina.BottomMargin = 3000
      oNovoEstiloPagina.IsLandscape = True
      ' obtém o cursor de texto
      oTexto = oDoc.getText()
      oCursor = oTexto.createTextCursor()
      ' aplica o novo estilo na página corrente
      ' observe que aqui, PageDescName não insere uma quebra!
      oCursor.PageDescName = "Minha_Pagina_Pessoal"
   End If
End Sub
```

Note que o novo estilo será acrescentado à coleção de estilos do documento.

Criando estilos de parágrafos

O serviço com.sun.star.style.ParagraphStyle deve ser usado para criar novos estilos de parágrafos. Ele inclui os serviços ParagraphProperties e Style, já apresentados.

Eis algumas das propriedades de ParagraphStyle:

long ParaLeftMarginRelative => define se a margem esquerda é relativa à do estilo base
long ParaRightMarginRelative => define se a margem direita é relativa à do estilo base
long ParaTopMarginRelative => define se a margem superior é relativa à do estilo base
long ParaBottomMarginRelative => define se a margem inferior é relativa à do estilo base
short Category => define a categoria do estilo [style.ParagraphStyleCategory]

constantes (e valores) de com.sun.star.style.ParagraphStyleCategory:

TEXT (0) => estilos usados em texto comum
CHAPTER (1) => estilos usados em títulos
LIST (2) => estilos usados em numeração e listas
INDEX (3) => estilos usados em índices
EXTRA (4) => estilos usados em áreas específicas (cabeçalho, rodapé, etc)
HTML (5) => estilos usados no suporte de HTML

Vejamos um exemplo:

```
Sub criandoEstiloParagrafo
   ' obtem o modelo
   oDoc = ThisComponent
   ' cria uma instancia do estilo
   oNovoEstilo = oDoc.createInstance("com.sun.star.style.ParagraphStyle")
   ' obtem a coleção de estilos de parágrafos
   oParagrafos = oDoc.getStyleFamilies.getByName("ParagraphStyles")
   If oParagrafos.hasByName( "Meu_Novo_Estilo" ) Then
      MsgBox "Estilo já existe.", 176, "Erro"
   Else
      ' insere o novo parágrafo
      oParagrafos.insertByName ( "Meu_Novo_Estilo", oNovoEstilo )
      'define algumas propriedades
      oNovoEstilo.CharFontName = "Arial"
      oNovoEstilo.CharHeight = 20
      oNovoEstilo.ParaAdjust = com.sun.star.style.ParagraphAdjust.CENTER
      ' cria um crusor de texto
      oTexto = oDoc.getText()
      oCursor = oTexto.createTextCursor()
      oCursor.gotoStart(False)
      oCursor.gotoEndOfParagraph(True)
      oCursor.setString("Um Novo Estilo de parágrafo!"
      ' aplica o novo estilo
      oCursor.ParaStyleName = "Meu_Novo_Estilo"
   End If
End Sub
```

Note que o novo estilo reside apenas no documento aonde foi criado.

De modo geral, os passos para criar novos estilos são:

obter as famílias de estilos
obter a família do tipo de estilo a ser criado
criar uma instância do objeto xxxStyle

inserir o objeto na coleção
definir as propriedades do novo objeto
Procure identificá-los nos exemplos anteriores.

9.7 Numerando Tópicos

Através da API podemos numerar parágrafos e tópicos.

Numerando parágrafos

Para numerar e marcar parágrafos, o serviço ParagraphProperties tem as propriedades:

```
short NumberingLevel => define o nível de numeração do parágrafo
com.sun.star.container.XIndexReplace NumberingRules => define as regras de numeração
short NumberingStartValue => define o valor inicial da numeração, se inicia uma nova
boolean ParaIsNumberingRestart => define se a numeração recomeça
string NumberingStyleName => define um estilo predefinido de numeração
boolean NumberingIsNumber => retorna True se a numeração é numérica sem símbolo
```

As principais são NumberingRules e NumberingLevel.

O serviço **com.sun.star.text.NumberingRules** define as propriedades:

```
boolean IsAbsoluteMargins => define se as margens são relativas ao nível anterior
boolean IsAutomatic => define se as regras são automaticamente criadas
boolean IsContinuousNumbering => define se a numeração dos níveis é contínua ou separada
string Name => [readonly] nome das regras de numeração
boolean NumberingIsOutline => define se é numeração de estrutura de tópicos
```

NumberingRules contém, ainda, as propriedades para os níveis de numeração. Cada nível têm o mesmo conjunto de propriedades, definidas em **com.sun.star.text.NumberingLevel,** eis as principais:

```
com.sun.star.style.HorizontalAlignment Adjust => define o alinhamento
```

a enum HorizontalAlignment tem os valores:

```
LEFT => alinhamento horizontal na margem esquerda do objeto
CENTER => alinhamento horizontal centralizado
RIGHT => alinhamento horizontal na margem direita do objeto
```

```
short NumberingType => define o tipo de numeração
```

algumas constantes (e valores) de com.sun.star.style.NumberingType:

```
CHARS_UPPER_LETTER (0) => numeração em letras maiúsculas (A, B, C, ... )
CHARS_LOWER_LETTER (1) => numeração em letras minúsculas (a, b, c, ... )
ROMAN_UPPER (2) => numeração em números romanos maiúsculos (I, II, III, ... )
ROMAN_LOWER (3) => numeração em números romanos minúsculos (i, ii, iii, ... )
ARABIC (4) => numeração em números arábicos (1, 2, 3, ... )
CHAR_SPECIAL (6) => usa um caractere especial de uma fonte definida
BITMAP (8) => numeração exibida como uma figura bitmap
```

```
short ParentNumbering => número do nível superior a incluir na númeração corrente
string Prefix => prefixo inserido na frente do símbolo
string Suffix => prefixo inserido na após o símbolo
string CharStyleName => nome do estilo de caractere usado para o símbolo
```

```
short StartWith => define o valor inicial da numeração
long LeftMargin => define a margem esquerda para a numeração
long SymbolTextDistance => distância entre o símbolo e o texto
long FirstLineOffset => define o deslocamento da primeira linha
```

Vejamos um exemplo usando algumas destas propriedades:

```
Sub numerandoParagrafos
   Dim i As Integer, j As Integer
   ' obtém o aplicativo
   oDesktop = StarDesktop
   ' carrega um novo documento
   oDoc = oDesktop.loadComponentFromURL("private:factory/swriter",_
          "_blank", 0, Array())
   ' cria as regras de numeração ( contém regras para 10 níveis )
   oNumeracao = oDoc.createInstance("com.sun.star.text.NumberingRules")
   ' percorre os níveis que nos interessa
   For i = 0 To 2
      ' obtém uma CÓPIA das propriedades do nível
      vProp = oNumeracao.getByIndex(i)
      ' percorre as propriedades de cada nível
      For j = 0 To UBound(vProp())
         ' localiza a propriedade: Tipo de Numeração
         If vProp(j).Name = "NumberingType" Then
            ' altera o tipo de numeração de acordo com o nível
            Select Case i
              Case 0  ' nível 0 ( letras maiúsculas: A B C ... )
               vProp(j).Value = com.sun.star.style.NumberingType.CHARS_UPPER_LETTER
              Case 1  ' nível 1 ( letras minúsculas: a b c ... )
               vProp(j).Value = com.sun.star.style.NumberingType.CHARS_LOWER_LETTER
              Case 2  ' nível 2 ( números arábicos: 1 2 3 ... )
               vProp(j).Value = com.sun.star.style.NumberingType.ARABIC
            End Select
            ' atualiza as propriedades do nível, por que ?
            ' vProp é uma cópia das propriedades originais, a alteração que
            ' fizemos afetou apenas vProp e não as propriedades de oNumeracao
            oNumeracao.replaceByIndex(i, vProp())
         End If
      Next j
   Next i
   ' cria uma instância do objeto Text do documento
   oTexto = oDoc.getText()
   quebraPara = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
   ' cria um cursor de texto para o objeto Text
   oCursor = oTexto.createTextCursor()
   ' atribui as regras de numeração ao TextCursor
   oCursor.setPropertyValue ("NumberingRules", oNumeracao)
   ' insere conteudo textual
   For i = 0 To 4
      j = IIF (i < 3, i, 4 - i)
      oCursor.setString ("Nível de numeração " & Str(j))
      ' define o nível da numeração deste parágrafo
      oCursor.setPropertyValue("NumberingLevel", j)
      oTexto.insertControlCharacter(oCursor.getEnd(), quebraPara, False)
      oCursor.gotoNextParagraph(False)
   Next i
   oCursor.setString ("Nível de numeração " & Str (j))
End Sub
```

Analise o código, os comentários e o documento com a numeração. Veja como criar as regras, como alterar e atualizar a propriedade de um nível, como atribuir as regras e definir o nível para um dado parágrafo.

Numerando capítulos

Para obter os dados de numeração dos capítulos de um documento, usamos a interface XChapterNumberingSupplier, do modelo do documento, e seu método:

com.sun.star.container.XIndexReplace getChapterNumberingRules ( )
retorna uma coleção com as regras de numeração do documento

Além das propriedades já apresentadas para a numeração de parágrafos, temos o mesmo conjunto de propriedades, para cada nível, com uma propriedade adicional:

string HeadingStyleName => contém o nome do estilo de parágrafo usado no nível

Segue um exemplo de numeração de capítulos num documento:

```
Sub numerandoCapitulos
   ' carrega um novo documento
   sURL = "private:factory/swriter"
   oDoc = StarDesktop.loadComponentFromURL(sURL, "_blank", 0, Array())
   ' obtém as regras de numeração de capítulos
   ' também retorna 10 NumberingLevels
   oNumCap = oDoc.getChapterNumberingRules()
   ' define valores iniciais de propriedades
   margem = 700
   tipo = com.sun.star.style.NumberingType.ARABIC
   ' percorre os níveis que nos interessa,
   ' alterando algumas propriedades
   For i = 0 To 2
      margem = margem + 100
      estilo = "Título " & Trim(Str(i))
      vet() = alteraPropNivel(oNumCap, "NumberingType", tipo, i)
      ' atualiza as regras de numeração
      oNumCap.replaceByIndex(i, vet())
      vet() = alteraPropNivel(oNumCap, "LeftMargin", margem, i)
      oNumCap.replaceByIndex(i, vet())
      vet() = alteraPropNivel(oNumCap, "HeadingStyleName", estilo, i)
      oNumCap.replaceByIndex(i, vet())
   Next i
   ' obtém o texto do documento
   oTexto = oDoc.getText()
   quebraPara = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
   ' cria um cursor de texto para o objeto Text
   oCursor = oTexto.createTextCursor()
   ' define os estilos de parágrafos a usar
   sEst() = Array("Título 1", "Corpo de texto", "Título 2", _
                  "Corpo de texto", "Título 3", "Corpo de texto", _
                  "Título 2", "Corpo de texto", "Título 3", "Corpo de texto")
   ' insere conteudo textual
   For i = 0 To 9
      oCursor.setString (sEst(i))
      ' define o estilo do parágrafo
      oCursor.setPropertyValue("ParaStyleName", sEst(i))
      oTexto.insertControlCharacter(oCursor.getEnd(), quebraPara, False)
      oCursor.gotoNextParagraph(False)
   Next i
   ' faz o cursor abranger todo texto
   oCursor.gotoStart(False)
   oCursor.gotoEnd(True)
   ' atribui as regras de numeração ao TextCursor
```

```
   oCursor.setPropertyValue ("NumberingRules", oNumCap)
End Sub
'_____________________________________________________________________________
'
Function alteraPropNivel ( oOriginal, sNome, valor, nivel ) As Variant
   ' obtém uma CÓPIA das propriedades do nível
   vProp = oOriginal.getByIndex(nivel)
   ' percorre as propriedades do nível
   For j = 0 To UBound(vProp())
      ' localiza a propriedade
      If vProp(j).Name = sNome Then
         ' altera o valor da propriedade
          vProp(j).Value = valor
      End If
      alteraPropNivel = vProp()
   Next j
End Function
```

Exceto pelo uso de uma função auxiliar e das propriedades LeftMargin, HeadingStyleName e ParaStyleName este exemplo é similar ao anterior.

9.8 Parágrafos

O conteúdo de um objeto Text pode ser enumerado. Isto é útil, por exemplo, para identificar os parágrafos e as tabelas existentes no objeto.

O serviço **com.sun.star.text.Paragraph** representa um parágrafo e inclui, dentre outros, os serviços ParagraphProperties, CharacterProperties, TextContent e TextTable, este último apenas quando o parágrafo referir-se a uma tabela.

Entre as suas interfaces temos XEnumerationAccess e seu método createEnumeration, para criar uma coleção contendo os objetos TextPortion.

O método createEnumeration, do objeto Text, retorna uma coleção de parágrafos (inclusive tabelas, mas não enumera os parágrafos dentro da tabela).

Vejamos um exemplo de identificação dos parágrafos:

```
Sub exemploParagrafos
   ' obtém o modelo do documento
   oDoc = ThisComponent
   ' obtém Text e cria uma coleção
   Enum = oDoc.getText().createEnumeration()
   ' percorre os elementos da coleção
   While Enum.hasMoreElements()
      ' obtem o objeto da coleção
      oTxt = Enum.nextElement()
      ' pode ser que algum objeto da coleção não seja
      ' um parágrafo, então precisamos nos certificar
      If oTxt.supportsService("com.sun.star.text.Paragraph") Then
         ' é um parágrafo
         MsgBox oTxt.getString()
      End If
   Wend
End Sub
```

Neste exemplo identificamos os parágrafos do objeto Text do documento (rode a macro num documento com 2 ou 3 parágrafos). Podemos, também, enumerar um TextRange.

Partes de um parágrafo

O conteúdo do objeto Paragraph também pode ser enumerado, isto é útil, por exemplo, para identificar texto com a mesma formatação, campos e outros objetos dentro do parágrafo.

O serviço **com.sun.star.text.TextPortion** inclui TextRange e representa uma parte do parágrafo com os mesmos atributos. Entre as suas propriedades, temos:

```
short ControlCharacter => retorna o caractere de controle, se um foi identificado
XTextContent Bookmark => retorna o objeto Bookmark, se for deste tipo
boolean IsCollapsed => define se a porção resume-se apenas a um ponto
string TextPortionType => uma cadeia de caracteres indicando o tipo da porção de texto
```

eis os principais nomes de tipo de texto:

```
Text => identifica conteúdo do tipo cadeia de caracteres
TextField => identifica um campo de texto
TextContent => se existe objeto gráfico ancorado como caractere
Footnote => indica uma nota de rodapé ou nota final
ControlCharacter => indica um caractere de controle
ReferenceMark => identifica uma marca de referência
DocumentIndexMark => identifica uma marca de índice
Bookmark => indica um marcador
Redline => identifica uma alteração no texto
```

Segue um exemplo, acessando as partes dos dois primeiros parágrafos de um documento:

```
Sub exemploPartesDoParagrafo
   ' cria um cursor de texto
   oCur = thisComponent.getText().createTextCursor()
   ' move o cursor e seleciona os dois primeiros parágrafos
   oCur.collapseToStart()
   oCur.gotoNextParagraph(True)
   oCur.gotoEndOfParagraph(True)
   ' cria a enumeração
   Enum = oCur.createEnumeration()
   ' percorre os elementos
   While Enum.hasMoreElements()
      ' obtém o objeto
      oTxt = Enum.nextElement()
      ' verifica se é um parágrafo
      If oTxt.supportsService("com.sun.star.text.Paragraph") Then
         ' é parágrafo, enumera o seu conteúdo
         Enum2 = oTxt.createEnumeration()
         ' percorre os elementos
         While Enum2.hasMoreElements()
            ' acessa o objeto e identifica o seu tipo
            oTxtParte = Enum2.nextElement()
            MsgBox oTxtParte.TextPortionType & "/" & oTxtParte.getString()
         Wend
      End If
   Wend
End Sub
```

Antes de rodar a macro, prepare os dois parágrafos iniciais com texto diferentemente formatado, um campo Autor e uma pequena figura ancorada como caractere.

9.9 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 cadeias, números, fórmulas, gráficos 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 os principais.

Criando tabelas

O serviço com.sun.star.text.TextTable inclui TextContent e implementa várias interfaces com métodos para executar diversas ações sobre as suas células. Entre as principais temos:

Interface **com.sun.star.text.XTextTable** com os métodos:

```
void initialize ( long  nLins, long  nCols )
```
define o número de linhas e colunas da tabela (chamar antes de inserir a tabela)

```
com.sun.star.table.XTableRows getRows ( )
```
retorna a coleção de linhas da tabela

```
com.sun.star.table.XTableColumns getColumns ( )
```
retorna a coleção de colunas da tabela

```
com.sun.star.table.XCell getCellByName ( string  sNomeCel )
```
retorna uma célula da tabela, conforme o nome da mesma

```
< string > getCellNames ( )
```
retorna uma sequência de cadeias com os nomes de todas as células da tabela

```
com.sun.star.text.XTextTableCursor createCursorByCellName ( string  sNomeCel )
```
cria um objeto TextTableCursor na célula nomeada

Interface **com.sun.star.container.XNamed** com os métodos:

```
string getName ( )
void setName ( string sNome )
```
se você não definir um nome, o OO.o se encarrega da nomeação. Os nomes devem ser únicos no container e não devem possuir espaços.

Interface **com.sun.star.table.XCellRange** com os métodos:

```
XCell getCellByPosition ( long nCol, long nLin )
```
retorna o objeto com.sun.star.table.Cell de acordo com a sua posição

```
XCellRange getCellRangeByPosition (long nEsq, long nSup, long nDir, long nInf )
```
retorna o objeto com.sun.star.table.CellRange defindo pelos cantos superior e inferior

```
XCellRange getCellRangeByName ( string  sNome )
```

retorna o objeto com.sun.star.table.CellRange de acordo com o seu nome

Interface **com.sun.star.table.XAutoFormattable**, com os métodos:

```
void autoFormat ( string  sNomeAutoFormat )
```
aplica uma autoformatação. Verifique os nomes disponíveis e o aspecto, inserindo uma tabela num documento e selecionando *Tabela* | *Auto Formatar* no menu principal do OO.o.

Interface **com.sun.star.util.XSortable** com os métodos:

```
< com.sun.star.beans.PropertyValue > createSortDescriptor ( )
```
cria um descritor de ordenação

```
void sort ( < com.sun.star.beans.PropertyValue >  vDescritor )
```
ordena de acrodo com o descritor

Interface **com.sun.star.sheet.XCellRangeData** com os métodos:

```
< < any > > getDataArray ( )
```
retorna uma sequência de sequências com os dados da extensão de células

```
void setDataArray ( < < any > >  vDados )
```
atribui os dados na sequência de sequências às células da extensão

A interface com.sun.star.chart.XChartDataArray lida com os dados de um gráfico e será apresentada posteriormente.

Eis as propriedades de com.sun.star.text.TextTable:

```
long LeftMargin => margem esquerda da tabela
long RightMargin => margem direita da tabela
long BottomMargin => margem inferior da tabela
long TopMargin => margem superior da tabela
long Width => largura absoluta da tabela
long BackColor => cor do fundo da tabela
boolean BackTransparent => define se a cor do fundo é transparente
com.sun.star.style.BreakType BreakType => quebra aplicada no início da tabela
boolean ChartColumnAsLabel => a primeira coluna contém os títulos dos eixos do gráfico
boolean ChartRowAsLabel => a primeira coluna contém os títulos dos eixos do gráfico
com.sun.star.text.HoriOrientation HoriOrient => [short] orientação horizontal
```

Eis as constantes (e valores) de com.sun.star.text.HoriOrientation:

```
NONE (0) => nenhum alinhamento (use quando definir as margens da tabela)
RIGHT (1) => o objeto será alinhado no lado direito
CENTER (2) => o objeto será alinhado no centro
LEFT (3) => o objeto será alinhado no lado esquerdo
FULL(6) =>  o objeto usará todo o espaço ( somente para tabelas )
LEFT_AND_WIDTH (7) => o deslocamento da esquerda e a largura serão definidos
```

```
boolean IsWidthRelative => define se o valor relativo da largura é válido
boolean KeepTogether => se True, previne quebras entre esta tabela e o próximo parágrafo
string PageDescName => insere uma quebra de página antes e define o nome do estilo
short PageNumberOffset => o novo valor do número de página, se tiver uma quebra
short RelativeWidth => define a largura relativa da tabela
boolean RepeatHeadline => define se a primeira linha será repetida em cada página
boolean Split => se False, evita que a tabela seja distribuída sobre duas páginas
com.sun.star.table.TableBorder TableBorder => descrição das bordas da tabela
```

eis os campos da estrutura com.sun.star.table.TableBorder:

```
BorderLine TopLine => define os atributos da linha superior
boolean IsTopLineValid => define se TopLine deve ser usada
```

```
BorderLine BottomLine => define os atributos da linha inferior
boolean IsBottomLineValid => define se BottomLine deve ser usada
BorderLine LeftLine => define os atributos da linha da esquerda
boolean IsLeftLineValid => define se LeftLine deve ser usada
BorderLine RightLine => define os atributos da linha da direita
boolean IsRightLineValid => define se RightLine deve ser usada
BorderLine HorizontalLine => define os atributos das linhas horizontais
boolean IsHorizontalLineValid => define se HorizontalLine deve ser usada
BorderLine VerticalLine => define os atributos das linhas verticais
boolean IsVerticalLineValid => define se VerticalLine deve ser usada
short Distance => define a distância entre a linha e o conteúdo
boolean IsDistanceValid => define se Distance deve ser usada
```

campos da estrutura com.sun.star.table.BorderLine:

```
com.sun.star.util.Color Color => cor da linha da borda
short InnerLineWidth => largura da linha interna quando dupla
short OuterLineWidth => largura de uma linha simples ou da externa quando dupla
short LineDistance => distância entre as linhas
```

```
ShadowFormat ShadowFormat => define os atributos do sombreamento
```

eis os campos da estrutura com.sun.star.table.ShadowFormat:

```
com.sun.star.table.ShadowLocation  Location => posição do sombreamento
short  ShadowWidth => define a largura do sombreamento
com.sun.star.util.Color  Color => define a cor do sombreamento
boolean  IsTransparent => define se é ou não transparente
```

```
short TableColumnRelativeSum => soma das larguras das colunas
< TableColumnSeparator > TableColumnSeparators => define a largura das colunas
```

em tabelas com células mescladas ou divididas a sequência é vazia. Eis os campos da estrutura com.sun.star.text.TableColumnSeparator:

```
short Position => define a posição do separador (relativa à margem esquerda)
boolean IsVisible => define se o separador será visível ( células mescladas )
```

```
string BackGraphicFilter => nome do filtro do arquivo gráfico do fundo
com.sun.star.style.GraphicLocation BackGraphicLocation => posição do gráfico do fundo
string BackGraphicURL => URL do arquivo gráfico do fundo
```

Após apresentar os métodos e propriedades do objeto TextTable, eis as etapas para criar e inserir uma tabela num documento:

```
criar o objeto, chamando um dos métodos do ServiceManager
inicializar a tabela (definir o número de linhas e colunas)
definir, se preciso, um nome sem espaço para a tabela, chamando setName ( )
inserir a tabela no documento, chamando insertTextContent ( )
alterar as propriedades da tabela
```

Vejamos um exemplo:

```
Sub criandoTabela
   ' cria um documento vazio
   sURL = "private:factory/swriter"
   oDoc = StarDesktop.loadComponentFromURL(sURL,"_blank", 0, Array())
   ' cria uma tabela (propriedades com valores padrão)
   oTabela = oDoc.createInstance("com.sun.star.text.TextTable")
```

```
    ' define o número de linhas e colunas
    oTabela.initialize ( 6, 4 )
    ' insere a tabela no documento
    oTxt = oDoc.getText()
    oCur = oTxt.createTextCursor()
    oTxt.insertTextContent (oCur, oTabela, False)
    '
    ' define algumas propriedades da tabela:
    ' margens superior, esquerda e direita
    oTabela.TopMargin = 2000  ' 2 cm
    oTabela.LeftMargin = 2000
    oTabela.RightMargin = 2000
    ' horientação horizontal (margens dependem de NONE)
    oTabela.HoriOrient = com.sun.star.text.HoriOrientation.NONE
    ' repete o cabeçalho nas páginas seguintes
    oTabela.RepeatHeadline = True
    ' cria uma estrutura para o tipo da linha da borda
    vLinhaBorda = createUNOStruct("com.sun.star.table.BorderLine")
    ' define os campos da estrutura linha da borda
    vLinhaBorda.Color = RGB(255,0,0) ' vermelho
    vLinhaBorda.InnerLineWidth = 20  ' 0,2 mm
    vLinhaBorda.OuterLineWidth = 20  ' 0,2 mm
    vLinhaBorda.LineDistance = 100   ' 1 mm
    ' cria uma estrutura para a borda da tabela
    vBordasTabela = createUNOStruct("com.sun.star.table.TableBorder")
    ' define valores para os campos da borda externa da tabela,
    ' note que os mesmos valores são aplicados a todas as bordas.
    ' as bordas internas permanecem com o valor padrão
    vBordasTabela.LeftLine = vLinhaBorda
    vBordasTabela.IsLeftLineValid = True
    vBordasTabela.RightLine = vLinhaBorda
    vBordasTabela.IsRightLineValid = True
    vBordasTabela.TopLine = vLinhaBorda
    vBordasTabela.IsTopLineValid = True
    vBordasTabela.BottomLine = vLinhaBorda
    vBordasTabela.IsBottomLineValid = True
    oTabela.TableBorder = vBordasTabela
    ' altera a largura padrão das colunas:
    MsgBox "Note a largura padrão", 176, "Altera largura das colunas"
    ' obtém cópia dos valores atuais dos separadores
    vSeparador = oTabela.TableColumnSeparators
    ' altera a posição dos dois primeiros
    vSeparador(0).Position = 1000
    vSeparador(1).Position = 6000
    ' copia de volta na propriedade
    oTabela.TableColumnSeparators = vSeparador()
End Sub
```

Analise o uso de propriedades complexas (por ex: TableBorder é uma estrutura contendo campos do tipo estrutura) e procure identificar cada passo no código fonte. Como exercício, tente definir ou alterar outras propriedades.

Linhas e Colunas

O serviço **com.sun.star.text.TableRows** representa a coleção de linhas numa tabela e implementa a interface **com.sun.star.table.XTableRows**, com os métodos:

`void insertByIndex ( long  nIndice, long  nLinhas )`
insere n_linhas na posição indicada pelo índice

void removeByIndex (long nIndice, long nLinhas)
remove n_linhas na posição indicada pelo índice

XTableRows herda os métodos de XIndexAccess e XElementAccess.

Para obter a coleção de linhas, chame o método getRows do objeto tabela:

```
oLinhas = oTabela.getRows ( )
```

O serviço **com.sun.star.text.TableColumns** representa a coleção de colunas da tabela e implementa a interface **com.sun.star.table.XTableColumns** com os métodos:

void insertByIndex (long nIndice, long nColunas)
insere n_colunas na posição indicada pelo índice

void removeByIndex (long nIndice, long nColunas)
remove n_colunas na posição indicada pelo índice

XTableColumns também é derivada de XIndexAccess e XElementAccess.

Para obter a coleção de colunas, chame o método getColumns do objeto tabela:

```
oColunas = oTabela.getColumns ( )
```

Para obter uma linha chamamos o método getByIndex na coleção de linhas:

```
oLinha = oLinhas.getByIndex ( nIndice )
```

O serviço **com.sun.star.text.TextTableRow** representa uma linha da tabela, implementa a interface XPropertySet e possui uma série de propriedades, entre as quais:

com.sun.star.util.Color BackColor => define a cor de fundo do objeto
boolean BackTransparent => Se True, a cor definida em BackColor não é visível
string BackGraphicURL => define um URL para o gráfico do fundo
string BackGraphicFilter => define o nome do filtro para o gráfico do fundo
com.sun.star.style.GraphicLocation BackGraphicLocation => posição do gráfico do fundo
TableColumnSeparator TableColumnSeparators => descreve os separadores na linha
long Height => define a altura da linha (depende de IsAutoHeight)
boolean IsAutoHeight => Se True, a altura da linha depende do conteúdo das células
boolean IsSplitAllowed => [opcional] Se True, a linha pode ser dividida nas quebras

Em tabelas de texto, não temos, ainda, como obter uma coluna da coleção TableColumns.

Em tabelas com células mescladas ou dividivas, os métodos para inserir e remover linhas e colunas não funcionam.

Segue um exemplo de acesso às linhas e colunas de uma tabela.

```
Sub exemploLinhasColunas
   ' cria um documento vazio
   sURL = "private:factory/swriter"
   oDoc = StarDesktop.loadComponentFromURL(sURL,"_blank", 0, Array())
   ' cria uma tabela (propriedades com valores padrão)
   oTabela = oDoc.createInstance("com.sun.star.text.TextTable")
   ' define o número de linhas e colunas
   oTabela.initialize ( 2, 2 )
   ' insere a tabela no documento
```

```
    oTxt = oDoc.getText()
    oCur = oTxt.createTextCursor()
    oTxt.insertTextContent (oCur, oTabela, False)
    MsgBox "Ok para alterar", 176, "Linhas e Colunas"
    ' obtém a coleção de linhas e colunas
    oLinhas = oTabela.getRows()
    oColunas = oTabela.getColumns()
    ' insere 3 linhas e duas coluna
    oLinhas.insertByIndex(1, 3)
    oColunas.insertByIndex(1, 2)
    ' obtém a penúltima linha
    oLinha = oLinhas.getByIndex( oLinhas.getCount() - 2 )
    ' altera a sua altura e cor de fundo
    oLinha.setPropertyValue("IsAutoHeight", False)
    oLinha.setPropertyValue("Height", 2000)
    oLinha.setPropertyValue("BackColor", CLng ("&HAABBCC"))
End Sub
```

Agora, altere as propriedades para inserir um pequeno gráfico de fundo numa linha.

Células

Ao operar sobre as células de uma tabela, podemos fazê-lo de dois modos: (**1**) usando uma extensão de células e (**2**) usando células individuais. Na tabela abaixo, temos duas extensões e uma célula em destaque.

A1 (0, 0)	B1 (1, 0)	C1 (2, 0)	D1 (3, 0)
A2 (0, 1)	B2 (1, 1)	C2 (2, 1)	D2 (3,1)
A3 (0, 2)	B3 (1, 2)	C3 (2, 2)	D3 (3,2)
A4 (0, 3)	B4 (1, 3)	C4 (2, 3)	D4 (3,3)

O endereço de uma célula ou extensão pode ser expresso como uma cadeia de caracteres ou pelos números da coluna e linha; para as partes em destaque temos:

Célula... : D3 ou (3, 2) onde: 3 número da coluna e 2 número da linha
Extensão: A2**:**B3 ou (0, 1, 1, 2) onde: 0, 1 célula inicial e 1, 2 célula final
Extensão: C1**:**C4 ou (2, 0, 2, 3) onde: 2, 0 célula inicial e 2, 3 célula final
Note que os índices começam em zero e o primeiro valor é o da coluna.

Em tabelas complicadas o esquema de endereçamento muda. Observe isto mesclando e dividindo algumas células numa tabela, depois posicione o cursor sobre as mesmas e verifique na barra de estado do Writer os seus endereços.

Vejamos as características de cada um destes objetos.

Extensão de células (objeto CellRange)

Uma extensão de células é representada pelo serviço **com.sun.star.text.CellRange**, que inclui os serviços CharacterProperties e ParagraphProperties.

Entre as suas interfaces temos XCellRange, XCellRangeData, XSortable e XPropertySet. Ainda, XChartData e XChartDataArray para lidar a montagem de gráficos.

As propriedades de CellRange são:

com.sun.star.util.Color BackColor => define a cor de fundo do objeto

```
boolean BackTransparent => Se True, a cor definida em BackColor não é visível
string BackGraphicURL => define um URL para o gráfico do fundo
string BackGraphicFilter => define o nome do filtro para o gráfico do fundo
com.sun.star.style.GraphicLocation BackGraphicLocation => posição do gráfico do fundo
boolean ChartColumnAsLabel => define se a 1a. coluna será tratada como títulos dos eixos
boolean ChartRowAsLabel => define se a 1a. linha será tratada como títulos dos eixos
long NumberFormat => contém o formato numérico
long TopMargin => margem superior da célula
long BottomMargin => margem inferior da célula
```

Note que podemos ter uma extensão abrangendo uma só célula.

Para obter uma extensão chamamos getCellRangeByName ou getCellRangeByPosition do objeto tabela, como abaixo:

```
oExtensao = oTabela.getCellRangeByName("A2:B3")
```

ou, ainda:

```
oExtensao = oTabela.getCellRangeByPosition(0, 1, 1, 2)
```

Célula (objeto Cell)

Uma célula individual é representada pelo serviço **com.sun.star.text.Cell**, que inclui o serviço **com.sun.star.text.CellProperties**.

As propriedades de CellProperties são:

```
string CellName => contém o nome da célula
com.sun.star.util.Color BackColor => define a cor de fundo do objeto
boolean BackTransparent => Se True, a cor definida em BackColor não é visível
string BackGraphicURL => define um URL para o gráfico do fundo
string BackGraphicFilter => define o nome do filtro para o gráfico do fundo
com.sun.star.style.GraphicLocation BackGraphicLocation => posição do gráfico do fundo
long NumberFormat => contém o formato numérico
com.sun.star.table.BorderLine LeftBorder => linha da borda esquerda
com.sun.star.table.BorderLine RightBorder => linha da borda direita
com.sun.star.table.BorderLine TopBorder => linha da borda superior
com.sun.star.table.BorderLine BottomBorder => linha da borda inferior
long LeftBorderDistance => distância da borda esquerda
long RightBorderDistance => distância da borda direita
long TopBorderDistance => distância da borda superior
long BottomBorderDistance => distância da borda inferior
XNameContainer UserDefinedAttributes => contém atributos definidos pelo usuário
XTextSection TextSection => seção da tabela se a mesma estiver contida numa
boolean IsProtected => célula protegida contra escrita ou não
short VertOrient => orientação vertical do texto desta célula
```

O serviço Cell implementa a interface **com.sun.star.table.XCell** com os métodos:

```
string getFormula ( )
```

retorna uma cadeia com a fórmula da célula

```
void setFormula ( string sFormula )
```

define uma fórmula para a célula

```
double getValue ( )
```

retorna o valor da célula

```
void setValue ( double nValor )
```

define um valor para a célula

com.sun.star.table.CellContentType getType ( )
retorna o tipo do conteúdo da célula, da enum com.sun.star.table.CellContentType:

```
EMPTY => célula vazia
VALUE => célula contém um valor constante
TEXT => célula contém texto
FORMULA => célula contém uma fórmula
```

long getError ( )
retorna o código de erro das células contendo fórmula, senão retorna zero

Cell implementa também as interfaces XPropertySet e XText, além de outras.

Para obter uma célula, chamamos os métodos getCellByName ou getCellByPosition da tabela ou da extensão. O endereço da célula é relativo ao objeto que a contém.

```
oCelula = oTabela.getCellByName ( “D3” )
oCelula = oTabela.getCellByPosition ( 3, 2 )
oCelula = oExtensão.getCellByPosition ( 0, 0 )
```

o último comando retorna a primeira célula da extensão (não da tabela)

Vejamos um exemplo sobre os dois objetos.

```
Sub exemploCelulas
   ' cria um documento vazio
   sURL = "private:factory/swriter"
   oDoc = StarDesktop.loadComponentFromURL(sURL,"_blank", 0, Array())
   ' cria uma tabela
   oTabela = oDoc.createInstance("com.sun.star.text.TextTable")
   ' define o número de linhas e colunas
   oTabela.initialize ( 4, 4 )
   ' insere a tabela no documento
   oTxt = oDoc.getText()
   oCur = oTxt.createTextCursor()
   oTxt.insertTextContent (oCur, oTabela, False)
   ' cria uma extensão de células
   oExt = oTabela.getCellRangeByName("A2:B3")
   ' altera a cor de fundo
   oExt.setPropertyValue("BackColor", RGB(50,70,150))
   ' cria uma extensão de células
   oExt = oTabela.getCellRangeByPosition(2,0,2,3)
   ' altera a cor de fundo
   oExt.setPropertyValue("BackColor", RGB(50,100,100))
   ' cria uma célula
   oCel = oTabela.getCellByPosition(3,2)
   ' altera a cor de fundo
   oCel.setPropertyValue("BackColor", RGB(50,100,150))
   ' define um vetor
   vDados = Array("A", "B", "C", "D")
   ' visita as células da tabela
   For i = 0 To 3
      For j = 0 To 3
         ' obtem a célula na coluna i, linha j
         oCel = oTabela.getCellByPosition(i,j)
         sCont = vDados(i)&Trim(Str(j+1))&" (" & Str(i)&","&Str(j)&")"
         ' atribui conteudo para a célula
         oCel.setString( sCont )
      Next j
   Next i
End Sub
```

Cursor da tabela

O serviço **com.sun.star.text.TextTableCursor** é um cursor de tabela e devemos usá-lo para navegar pelas células e para formatar o conteúdo das mesmas. Ele inclui os serviços ParagraphProperties e CharacterProperties. Implementa as interfaces XPropertySet e **com.sun.star.text.XTextTableCursor**, com os métodos:

string getRangeName ( )
retorna o nome da extensão de células

boolean gotoCellByName (string sNome, boolean bExp)
move o cursor para a célula nomeada, se bExp for True move selecionando

boolean goLeft (short nCont, boolean bExp)
move o cursor para a esquerda

boolean goRight (short nCont, boolean bExp)
move o cursor para a direita

boolean goUp (short nCont, boolean bExp)
move o cursor para cima

boolean goDown (short nCont, boolean bExp)
move o cursor para baixo

void gotoStart (boolean bExp)
move o cursor para a primeira célula

void gotoEnd (boolean bExp)
move o cursor para a última célula

boolean mergeRange ( )
mescla a extensão de células do cursor

boolean splitRange (short nCont, boolean bHoriz)
divide cada célula da extensão em nCont células

Agora, um exemplo demonstrando o emprego do objeto cursor de tabela.

```
Sub exemploCursorTabela
   sURL = "private:factory/swriter"
   oDoc = StarDesktop.loadComponentFromURL(sURL,"_blank", 0, Array())
   oTab = oDoc.createInstance("com.sun.star.text.TextTable")
   oTab.initialize ( 5, 3 )
   oTxt = oDoc.getText()
   oCur = oTxt.createTextCursor()
   oTxt.insertTextContent (oCur, oTab, False)
   Dim oCelula As Object
   oCelula = oTab.getCellByName("A1") ' nome em maiúsculas
   oCelula.setString("Coluna A")
   oCelula = oTab.getCellByName("B1")
   oCelula.setString("Coluna B")
   oCelula = oTab.getCellByName("C1")
   oCelula.setString("Coluna C")
   ' cria um cursor de tabela na primeira célula
   Dim oCurTab As Object
   oCurTab = oTab.createCursorByCellName(oTab.CellNames(0))
   ' move para o início sem selecionar
   oCurTab.gotoStart(False)
```

```
   ' move 2 células a direita, selecionando
   oCurTab.goRight(2, True)
   oCurTab.ParaBackColor = CLng( "&HAABBCC" )
   MsgBox oCurTab.getRangeName(), 176, "Método do cursor"
   Dim sNomes() As Variant
   Dim sNomeCelula As String
   sNomes = Array("A","B","C")
   ' visita as células e insere conteúdo
   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
            oCelula.setValue(i% + 1)
         Else
            oCelula.setString(sNomeCelula)
         End If
      Next j%
   Next i%
   ' insere uma fórmula numa célula
   oCelula = oTab.getCellByName("C5")
   oCelula.setFormula("sum <C2:C4>")
   oCurTab.gotoCellByName("A5", False)
   oCurTab.goRight(1, True)
   ' mescla células
   oCurTab.mergeRange()
   oCurTab.CharColor = CLng( "&H0000FF" )
   oCelula = oTab.getCellByName("A5")
   oCelula.setString("Total")
End Sub
```

Note como se dá a atribuição de uma fórmula para uma célula.

Visitando as tabelas do documento

O método getTextTables da interface XTextTablesSupplier retorna uma coleção com as tabelas existentes no documento. Depois, usamos o acesso nomeado ou indexado para obter cada tabela da coleção. Segue um exemplo.

```
Sub visitaTabelas
   ' obtém a coleção de tabelas no documento
   oTabelas = thisComponent.getTextTables()
   ' visita cada tabela da coleção
   For i = 0 To oTabelas.getCount() - 1
      ' usando o acesso indexado
      oTabela = oTabelas.getByIndex(i)
      MsgBox oTabela.getName()
   Next i
End Sub
```

9.10 Campos

O OOffice.org usa uma grande variedade de tipos de campos. Estes campos caem em duas categorias: (1) simples e (2) dependentes. Os primeiros contém os seus próprios dados e os segundos obtém seus dados de um campo principal (master).

Os campos simples suportam o serviço com.sun.star.text.TextField, que inclui TextContent e exporta as interfaces XPropertySet e XTextField, esta última com o método:

```
string getPresentation ( boolean  bComando )
```
retorna o conteúdo do campo ou o seu comando, se bComando for True

XTextField possui também os métodos de XTextContent.

Os campos dependentes suportam o serviço com.sun.star.text.DependentTextField, que inclui TextField e exporta a interface XDependentTextField, com os métodos:

```
void attachTextFieldMaster ( com.sun.star.beans.XPropertySet oCampoMaster )
```
liga um campo master a este campo

```
com.sun.star.beans.XPropertySet getTextFieldMaster ( )
```
retorna o campo master ligado ao campo

Os serviços de cada tipo de campo encontram-se no módulo com.sun.star.text.textfield, seguem alguns:

```
com.sun.star.text.textfield.PageCount => [simples] campo total de páginas
com.sun.star.text.textfield.PageNumber => [simples] campo número da página
com.sun.star.text.textfield.DateTime => [simples] campo data hora
com.sun.star.text.textfield.DropDown => [simples] campo de lista de entrada
com.sun.star.text.textfield.Database => [dependente] campo de banco de dados
com.sun.star.text.textfield.SetExpression => [dependente] campo de definição de expressão
com.sun.star.text.textfield.User => [dependente] campo do usuário
```
As propriedades dos campos variam conforme o tipo do campo, por exemplo, PageCount tem atributos diferentes de DateTime.

Os campos dependentes precisam de um campo master, os quais estão relacionados no módulo com.sun.star.text.FieldMaster, eis alguns:

```
com.sun.star.text.FieldMaster.Database => campo principal de banco de dados
com.sun.star.text.FieldMaster.SetExpression => campo principal de definição
com.sun.star.text.FieldMaster.User => campo principal do usuário
```
Estes campos suportam o serviço com.sun.star.text.TextFieldMaster e, aqui também, as propriedades variam segundo o tipo do campo.

Eis alguns nomes de propriedades encontrados em ambas as categorias:

```
string Name => nome do campo
double Value => valor do campo, se numérico
string Content => conteúdo do campo
string CurrentPresentation => apresentação na interface gráfica
com.sun.star.util.DateTime DateTimeValue => valor da data hora
short NumberFormat => formatação numérica
short NumberingType => tipo de numeração
```

Para informações detalhadas sobre os tipos de campos e suas propriedades, consulte o *The Developer's Guide*.

Criando campos

Para criar e inserir campos num documento, devemos:

criar o(s) objeto(s) com.sun.star.text.textfield.Tipo e com.sun.star.text.FieldMaster.Tipo
definir as propriedades do(s) objeto(s)
ligar os campos dependentes, se for o caso
inserir o campo no documento

Segue um exemplo:

```
Sub insereCampos
   sURL = "private:factory/swriter"
   oDoc = StarDesktop.loadComponentFromURL(sURL,"_blank", 0, Array())
   ' cria um objeto campo data
   oCampoData = oDoc.createInstance("com.sun.star.text.TextField.DateTime")
   ' insere o campo data
   oDocTexto = oDoc.getText()
   oDocTexto.setString("Campo Data/Hora: ")
   oDocTexto.insertTextContent ( oDocTexto.getEnd(), oCampoData, False )
   ' cria um Campo do Usuario
   oCampoUser = oDoc.createInstance("com.sun.star.text.TextField.User")
   ' cria o campo Master associado ao Campo do Usuario
   oMaster = oDoc.createInstance("com.sun.star.text.FieldMaster.User")
   ' define as propriedades do Campo Master
   oMaster.setPropertyValue ("Name", "Idade")
   oMaster.setPropertyValue ("Value", 20)
   ' para uma cadeia:
   ' oMaster.Content = "Conteudo de texto"
   '
   ' liga os dois campos ( User e Master )
   oCampoUser.attachTextFieldMaster (oMaster)
   ' cria um cursor e posiciona no final do documento
   oCursor = oDocTexto.createTextCursor()
   oCursor.gotoEnd(false)
   ' insere uma quebra de paragrafo
   qPara = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
   oDocTexto.insertControlCharacter(oCursor, qPara, false)
   oCursor.gotoEnd(False)
   oCursor.setString("Campo do usuário: ")
   ' insere o campo do usuario na posicao do cursor
   oDocTexto.insertTextContent(oDocTexto.getEnd(), oCampoUser, False)
End Sub
```

Agora, como exercício, insira mais alguns campos no documento.

Identificando e editando campos

O modelo do documento suporta a interface XTextFieldsSupplier com os métodos:

`com.sun.star.container.XEnumerationAccess getTextFields( )`
retorna a coleção com.sun.star.text.TextFields com todos os campos do documento

`com.sun.star.container.XNameAccess getTextFieldMasters( )`
retorna a coleção com.sun.star.text.TextFieldMasters de campos masters no documento

A coleção com.sun.star.text.TextFields implementa as interfaces XEnumerationAccess e XRefreshable, esta última com os métodos:

`void refresh ( )`
reabastece os dados do objeto

`void addRefreshListener ( com.sun.star.util.XRefreshListener oListener )`
cadastra um listener para o objeto

`void removeRefreshListener ( com.sun.star.util.XRefreshListener oListener )`
remove o cadastro do listener

com.sun.star.util.XRefreshListener possui o método:

`void refreshed ( com.sun.star.lang.EventObject oEvento )`
chamado quando ocorre o evento

A coleção com.sun.star.text.TextFieldMasters implementa a interface XNameAccess. Os nomes dos campos principais, usados no acesso nomeado, seguem o padrão:

`com.sun.star.text.FieldMaster.TipoDoCampo.NomeDoCampo`
para o campo User da macro anterior temos: com.sun.star.text.FieldMaster.User.Idade

Vejamos uma macro que identifica e edita dados dos campos do exemplo anterior:

```
Sub editaCampos
   oDoc = thisComponent
   ' obtém a coleção e cria uma enumeração
   enumCampos = oDoc.getTextFields().createEnumeration()
   ' percorre os campos
   Do While (enumCampos.hasMoreElements())
      oCampo = enumCampos.nextElement()
      msgBox oCampo.getPresentation(False)
   Loop
   '
   ' obtém a coleção de campos masters
   Masters = oDoc.getTextFieldMasters()
   sNomes = Masters.getElementNames()
   ' percorre os campos masters
   For i = LBound(sNomes) To UBound(sNomes)
      oMaster = Masters.getByName(sNomes(i))
      msgBox oMaster.InstanceName
   Next i
   ' edita campo do usuario
   Masters.getByName("com.sun.star.text.FieldMaster.User.Idade").Value = 200
   ' apos edicao atualizar campos
   oDoc.getTextFields().refresh()
End Sub
```

Note que, além dos campos que inserimos, existem outros criados, por padrão, juntamente com o documento.

9.11 Cabeçalhos e Rodapés

Os cabeçalhos e rodapés estão diretamente ligados ao estilo da página em uso e suas principais propriedades foram apresentadas na descrição do serviço PageProperties.

Agora, que dominamos o uso de campos, vamos editar os dados do rodapé de um documento.

```
Sub editandoRodape
   oDoc = ThisComponent
   ' obtém estilos de página
   oEstPagina = oDoc.StyleFamilies.getByName("PageStyles")
   ' obtém página padrão
   oPagPadrao = oEstPagina("Standard")
```

```
    ' ativa o rodapé
    oPagPadrao.FooterIsOn = True
    ' obtém o objeto texto do rodapé
    oTxtRodape = oPagPadrao.FooterText
    ' cria um cursor e insere texto
    oCursorRodape = oTxtRodape.createTextCursor()
    oCursorRodape.setString("Texto no rodapé da página padrão - Pág: ")
    oCursorRodape.collapseToEnd()
    ' cria um campo Número de Página
    oNrPagina = oDoc.createInstance("com.sun.star.text.TextField.PageNumber")
    ' define algumas propriedades do campo
    oNrPagina.NumberingType = com.sun.star.style.NumberingType.ARABIC
    oNrPagina.SubType = com.sun.star.text.PageNumberType.CURRENT
    ' insere o campo
    oTxtRodape.insertTextContent(oCursorRodape, oNrPagina, False)
End Sub
```

Para ativar e editar dados do cabeçalho usamos a mesma abordagem, alterando apenas as propriedades.

9.12 Desenhos

A API do OpenOffice.org possui diversos serviços e interfaces para operações com desenhos num documento, através de uma macro. Nesta seção, veremos como usar alguns destes serviços e interfaces.

O serviço com.sun.star.text.Shape, contém muitas propriedades relacionadas aos objetos de desenho, seguem algumas:

```
short AnchorPageNo => número da página aonde o objeto está ancorado
TextContentAnchorType AnchorType => tipo da âncora do objeto
short HoriOrient => orientação horizontal
short VertOrient => orientação vertical
long LeftMargin => margem esquerda
long RightMargin => margem direita
long TopMargin => margem superior
long BottomMagin => margem inferior
```

O serviço com.sun.star.drawing.Shape, incluído em com.sun.star.text.Shape, contém diversas propriedades adicionais, como Name, que permite definir um nome para o desenho, e implementa, entre outras, as interfaces XPropertySet, XComponent e com.sun.star.drawing.XShape, com os métodos:

```
com.sun.star.awt.Point getPosition ( )
void setPosition ( com.sun.star.awt.Point Ponto )
```

usados para obter / definir a posição do objeto

```
com.sun.star.awt.Size getSize ( )
void setSize ( com.sun.star.awt.Size Tamanho )
```

usados para obter / definir o tamanho do objeto

a estrutura com.sun.star.awt.Size, contém os campos:

```
long Width => define a largura do objeto
long Height => define a altura do objeto
```

Através da interface com.sun.star.lang.XMultiServiceFactory, criamos o objeto de desenho usando o método:

```
com.sun.star.uno.XInterface createInstance ( string sNomeObjeto )
```
cria uma instância do objeto especificado

No módulo **com.sun.star.drawing**, encontramos diversos serviços para a criação de objetos de desenho. Estes serviços contém, além de propriedades comuns, outras específicas. Eis alguns deles:

```
com.sun.star.drawing.EllipseShape => usado para desenhar círculos e elipses
com.sun.star.drawing.RectangleShape => usado para desenhar retângulos
com.sun.star.drawing.LineShape => usado para desenhar linhas
```

A interface com.sun.star.drawing.XShapes define métodos para adicionar e remover desenhos do documento, através da sua página de desenho:

```
void add ( com.sun.star.drawing.XShape oDesenho )
void remove ( com.sun.star.drawing.XShape oDesenho )
```

A interface com.sun.star.text.XText também define os métodos abaixo, já apresentados, para adicionar e remover desenhos do documento:

```
void insertTextContent (XTextRange oPos, XTextContent oCont, boolean bFlag)
```
insere oCont na posição oPos. Se bFlag for True o conteúdo de oPos será substituído

```
void removeTextContent ( XTextContent oCont )
```
remove o conteúdo de oCont do objeto

A interface com.sun.star.drawing.XDrawPageSupplier define o método abaixo para obter a página de desenho do documento:

```
com.sun.star.drawing.XDrawPages getDrawPage ( )
```
retorna uma coleção de DrawPages (no Writer, somente uma página)

Para a manipulação de desenhos no Writer e Calc, o OpenOffice.org utiliza os mecanismos básicos de desenho do Draw. Uma das principais diferenças, é que no Draw existem várias páginas de desenho, enquanto no Writer existe apenas uma.

Vejamos como inserir desenhos num documento, através de uma macro que usa alguns dos métodos e propriedades apresentados.

```
Sub criaDesenho
   Dim oTamanho As New com.sun.star.awt.Size
   Dim oPonto As New com.sun.star.awt.Point

   sURL = "private:factory/swriter"
   oDoc = StarDesktop.loadComponentFromURL(sURL,"_blank", 0, Array())
   oTexto = oDoc.getText()
   oCursor = oTexto.createTextCursor()
   ' insere dois novos parágrafos no final do texto
   oCursor.gotoEnd(False)
   oTexto.insertControlCharacter(oCursor, _
      com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
   oTexto.insertControlCharacter(oCursor, _
      com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
   oCursor.gotoPreviousParagraph(False)
```

```
   ' cria um objeto retângulo e um objeto elipse
   oRetang = oDoc.createInstance("com.sun.star.drawing.RectangleShape")
   oElipse = oDoc.createInstance("com.sun.star.drawing.EllipseShape")
   ' define o tamanho do retângulo e elipse
   oTamanho.Height = 4000
   oTamanho.Width = 10000
   oRetang.setSize(oTamanho)
   oRetang.Name = "Ret_1"
   oTamanho.Height = 3000
   oTamanho.Width = 6000
   oElipse.setSize(oTamanho)
   oElipse.Name = "Eli_1"
   ' define a posição do retângulo a direita da elipse
   oPonto.X = 6100
   oPonto.Y = 0
   oRetang.setPosition (oPonto)
   ' define a âncora no parágrafo
   oRetang.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
   oElipse.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
   ' insere na posição do cursor de texto
   oTexto.insertTextContent(oCursor,oElipse,FALSE)
   oTexto.insertTextContent(oCursor,oRetang,FALSE)
   oPosRet = oRetang.getPosition()
   MsgBox Str$(oPosRet.X)+" - "+Str$(oPosRet.Y)
   ' cria uma linha
   oLinha = oDoc.createInstance( "com.sun.star.drawing.LineShape" )
   ' define o tamanho da linha
   oTamanho.Height = 30
   oTamanho.Width = 6000
   oLinha.setSize (oTamanho)
   ' define a posição da linha
   oPonto.X = 6000
   oPonto.Y = 10000
   oLinha.setPosition (oPonto)
   ' obtém a página de desenho do documento
   oPagDes = oDoc.getDrawPage()
   ' adiciona a linha na primeira página
   oPagDes.add (oLinha)
End Sub
```

Começamos inserindo parágrafos no final do texto, a seguir criamos dois objetos de desenho, um retângulo e uma elipse, definimos a suas propriedades e usamos o método insertTextContent ( ) para adicionar os objetos no documento. Depois, criamos uma linha, obtemos a página de desenho do documento e adicionamos a linha com o método add ( ).

Para encerrar esta seção, vejamos um exemplo que identifica os objetos de desenho existentes no nosso documento. Crie a macro abaixo num documento contendo desenhos, molduras e gráficos. Em seguida execute e observe a saída.

```
Sub obtemDesenhos
   oDoc = ThisComponent
   ' obtém a página de desenho
   oPage = oDoc.getDrawPage()
   ' obtém o número de objetos na página de desenho
   nNrDes = oPage.getCount()
   sMsg = ""
   ' visita cada objeto
   For i = 0 To nNrDes - 1
      oShape = oPage.getByIndex ( i )
      ' obtem nome se definido na criação
```

```
      sMsg = sMsg + oShape.getShapeType() + chr$(10)
   Next i
   MsgBox sMsg + Str$(nNrDes), 176, "Desenhos"
End Sub
```

Note que outros objetos, baseados no serviço Shape, como molduras, também são identificados. O método getName ( ), somente retorna o nome, se o objeto foi nomeado durante a criação do desenho.

9.13 Busca e Substituição

A API oferece vários recursos para pesquisa e substituição de texto em documentos.

Busca

O modelo do documento, através da interface com.sun.star.util.XSearchable, possui métodos relacionados com a localização de texto. São eles:

`com.sun.star.util.XSearchDescriptor createSearchDescriptor ( )`
cria o objeto com.sun.star.util.SearchDescriptor para descrever os atributos da busca

`com.sun.star.uno.XInterface findFirst ( XSearchDescriptor oDescritorBusca )`
retorna um objeto (por ex: um TextRange) indicando a primeira ocorrência

`XInterface findNext ( XInterface oPosInicial, XSearchDescriptor oDescritorBusca )`
retorna um objeto (por ex: um TextRange) indicando a próxima ocorrência

`com.sun.star.container.XIndexAccess findAll ( XSearchDescriptor oDescritorBusca )`
retorna uma coleção de objetos com todas as ocorrências do descritor

O serviço com.sun.star.util.SearchDescriptor traz algumas propriedades relacionadas com o busca, entre as quais temos:

```
boolean SearchBackwards => se True, busca para o início do documento
boolean SearchCaseSensitive => se True, diferencia maiúsculas de minúsculas
boolean SearchWords => se True, busca por palavras completas
boolean SearchRegularExpression => se True, busca usando expressões regulares
boolean SearchStyles => se True, busca por estilos de formatação
```

Além destas, existem as propriedades para pesquisa por similaridade.

SearchDescriptor implementa a interface com.sun.star.util.XSearchDescriptor e XPropertySet, os métodos da primeira são:

`string getSearchString ( )`
retorna a cadeia a ser localizada

`void setSearchString ( string sCadeia )`
define o padrão a ser localizado, por ex: cadeia, estilo ou expressão regular

Existem alguns objetos do documento que são insensíveis a operação de busca.

Segue um pequeno exemplo sobre localização de texto:

```
Sub buscaTexto
   ' obtém o modelo do documento
   oDoc = ThisComponent
```

```
   ' cria um descritor de busca
   oBusca = oDoc.createSearchDescriptor()
   ' solicita o padrão
   sPalav = InputBox ( "Qual palavra ?" )
   ' define o que buscar
   oBusca.setSearchString(sPalav)
   ' localiza todas as ocorrências
   oResultado = oDoc.findAll(oBusca)
   ' percorre cada uma
   For i = 0 To oResultado.getCount() - 1
      ' obtém a i_ésima ocorrência
      oOcorr = oResultado.getByIndex(i)
      ' verifica se está dentro de um quadro ou tabela
      If ( IsEmpty(oOcorr.TextFrame) = False ) Then
         MsgBox "Dentro de um quadro", 176, "Busca texto"
      ElseIf ( IsEmpty(oOcorr.TextTable) = False ) Then
         MsgBox "Dentro de uma tabela", 176, "Busca texto"
      End If
   Next i
End Sub
```

O próximo exemplo demonstra o uso dos métodos findFirst e findNext:

```
Sub alteraTexto
   oDoc = ThisComponent
   oBusca = oDoc.createSearchDescriptor()
   sPalav = InputBox ( "Qual palavra ?" )
   oBusca.setSearchString(sPalav)
   ' localiza a primeira ocorrência
   oResultado = oDoc.findFirst(oBusca)
   Do Until IsNull ( oResultado )
       ' altera a cor do resultado
       oResultado.CharColor = RGB (255, 0, 0)
       ' localiza a próxima ocorrência
       oResultado = oDoc.findNext (oResultado, oBusca)
   Loop
End Sub
```

Aqui, a posição inicial para findNext é a ocorrência anterior. Contudo, podemos usar qualquer TextRange como ponto de partida.

Substituição

O modelo do documento também implementa a interface com.sun.star.util.XReplaceable, cujos métodos são:

`com.sun.star.util.XReplaceDescriptor createReplaceDescriptor ( )`
retorna o serviço com.sun.star.util.ReplaceDescriptor, um descritor de substituição

`long replaceAll ( XReplaceDescriptor oDescritor )`
substitui todas as ocorrências conforme o descritor, retorna o total de substituições

O serviço com.sun.star.util.ReplaceDescriptor herda os métodos e propriedades de SearchDescriptor e implementa a interface com.sun.star.util.XReplaceDescriptor, com os métodos:

`string getReplaceString ( )`
retorna a cadeia de substituição

`void setReplaceString ( string sCadeia )`

define a cadeia de substituição

A seguir, um exemplo que substitui as letras minúsculas de uma dada palavra por letras maiúsculas.

```
Sub trocaCaixa
   ' obtém o modelo
   oDoc = ThisComponent
   ' cria um descritor de substituição
   oBusca = oDoc.createReplaceDescriptor()
   ' define a palavra a localizar
   sPalav = InputBox ( "Qual palavra ?" )
   oBusca.setSearchString(sPalav)
   ' define a nova palavra
   oBusca.setReplaceString(UCase(sPalav))
   ' substitui todas as ocorrências
   nRes = oDoc.replaceAll(oBusca)
   MsgBox "Total: "&Str(nRes), 176, "Substituir"
End Sub
```

9.14 Ordenação

A ordenação de texto é possível em documentos do Writer. Para tal, os objetos TextCursor, TextTable e CellRange implementam a interface:

com.sun.star.util.XSortable com os métodos:

```
< com.sun.star.beans.PropertyValue > createSortDescriptor ( )
```
cria um descritor de ordenação com.sun.star.text.TextSortDescriptor2

```
void sort ( < com.sun.star.beans.PropertyValue >  vDescritor )
```
ordena de acordo com o descritor

O serviço TextSortDescriptor2 possui as propriedades:

```
boolean IsSortInTable => define se uma tabela ou uma seleção de parágrafos será ordenada
char Delimiter => define o caractere separador de colunas numa seleção de parágrafos
```

Também, inclui com.sun.star.table.TableSortDescriptor2, cujas propriedades são:

```
boolean IsSortColumns => define se as colunas ou as linhas serão ordenadas
long MaxSortFieldsCount => [readonly] contém o número máximo de campos ( 3 ) do descritor
< TableSortField > SortFields => define uma sequência de campos para ordenar
```

a estrutura com.sun.star.table.TableSortField contém os campos:

```
long Field => índice da linha / coluna a ordenar
boolean IsAscending => se True, ordenação crescente
boolean IsCaseSensitive => define se diferencia maiúsculas de minúsculas
com.sun.star.lang.Locale CollatorLocale => definições locais para comparação de texto
string CollatorAlgorithm => algoritmo usado para comparar texto
TableSortFieldType FieldType => tipo do conteúdo do campo
```

eis as constantes da enum com.sun.star.table.TableSortFieldType:

```
AUTOMATIC => o tipo do campo será determinado automaticamente
NUMERIC => o tipo do campo é numérico
ALPHANUMERIC => o conteúdo do campo é texto
```

O exemplo abaixo ordena os dez primeiros parágrafos de um documento.

```
Sub ordenaTexto
   oDoc = ThisComponent
   ' obtém o objeto texto
   oTexto = oDoc.getText()
   ' cria um cursor de texto
   oCursor = oTexto.createTextCursor()
   ' seleciona os 10 primeiros parágrafos
   oCursor.collapseToStart()
   For i = 0 To 8
      oCursor.gotoNextParagraph(True)
   Next i
   ' cria o descritor de ordenação
   oDescritor = oCursor.createSortDescriptor()
   ' define as propriedades que nos interessa
   oDescritor(0).Value = False    ' IsSortInTable
   oDescritor(1).Value = 32       ' Delimiter
   oDescritor(2).Value = False    ' IsSortColumns
   ' ordena o texto do objeto cursor
   oCursor.sort(oDescritor())
End Sub
```

O próximo exemplo ordena uma tabela de um documento do Writer.

```
Sub ordenaTabela
   ' obtém a 1a. tabela
   oTab = ThisComponent.getTextTables().getByName("Tabela1")
   ' cria o descritor de ordenação
   oDescritor = oTab.createSortDescriptor()
   ' define os campos a ordenar (aqui, 1a. coluna)
   Dim vCampo(0) As New com.sun.star.table.TableSortField
   vCampo(0).Field = 0
   vCampo(0).IsAscending = True
   vCampo(0).IsCaseSensitive = False
   vCampo(0).FieldType = com.sun.star.table.TableSortFieldType.ALPHANUMERIC
   ' define as propriedades que nos interessa
   oDescritor(0).Value = True       ' IsSortInTable
   oDescritor(1).Value = 32         ' Delimiter
   oDescritor(2).Value = False      ' IsSortColumns
   oDescritor(4).Value = vCampo()   ' SortFields
   ' ordena a tabela
   oTab.sort(oDescritor())
End Sub
```

9.15 Seleção

Às vezes pode ser útil selecionar objetos ou obter um objeto selecionado na interface gráfica. A apresentação visual dos dados é feita pelo objeto controlador do documento e as interfaces XModel e XSelectionSupplier possuem métodos para lidar com seleções.

Para obter o(s) objeto(s) selecionado(s), devemos:

chamar o método getCurrentSelection do modelo do documento, ou
obter o controlador do documento e chamar o seu método getSelection
O objeto retornado depende do conteúdo selecionado, podendo ser, por exemplo:

TextTableCursor => se células de uma tabela estiverem selecionadas

TextFrame => se uma moldura estiver selecionada
ShapeCollection => se tiver um ou mais desenhos selecionados
TextRanges => uma coleção de TextRange para seleção de texto, sendo:
se nada selecionado, um TextRange com a posição do cursor
seleção simples, um TextRange com o texto selecionado
seleção múltipla, um TextRange para o cursor e para cada texto selecionado

Para selecionar objetos devemos:

chamar o método select do controlador, passando o objeto como parâmetro
o método select retorna True se o objeto for selecionado ou False em caso contrário

Seguem dois exemplos simples, um para cada caso.

```
Sub processaSelecao
   ' obtém o modelo
   oDoc = ThisComponent
   ' obtém a seleção ( método de XModel )
   oSel = oDoc.getCurrentSelection()
   ' percorre a coleção de TextRange
   For i = 0 To oSel.Count-1
      ' obtém o TextRange e altera a cor
      oCurTxt = oSel(i)
      oCurTxt.CharColor = RGB(255,0,0)
   Next i
   MsgBox "Total: " & Str(oSel.Count), 176, "Seleção"
End Sub
```

Antes de executar a macro, selecione partes de texto no documento.

Este exemplo seleciona texto na interface gráfica:

```
Sub selecionaTexto
   ' obtém o modelo do documento
   oDoc = ThisComponent
   ' obtém o objeto Text do modelo
   oTxt = oDoc.getText()
   ' obtém o objeto controlador
   oDocView = oDoc.getCurrentController()
   ' tenta selecionar o objeto Text
   If oDocView.select( oTxt ) Then
      MsgBox "Texto selecionado"
   Else
      MsgBox "Erro na seleção"
   End If
End Sub
```

Os princípios destes exemplos aplicam-se a todos os documentos do OOffice.org.

10 Documentos do Calc

NÃO REVISAR ESTE CAPÍTULO ==> É RASCUNHO

10.1 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, gráficos, dados de uma fonte externa e objetos OLE.

As funções básicas já abordadas no capítulo *Trabalhando com Documentos*, podem ser usadas também com documentos do Calc.

Os principais serviços encontram-se nos módulos **com.sun.star.table** e **com.sun.star.sheet**. O primeiro oferece os serviços básicos e o segundo serviços mais especializados.

10.2 Documento

Aqui, também temos os objetos modelo e controlador do documento. Vejamos cada um.

Modelo

O serviço com.sun.star.sheet.SpreadsheetDocument representa o modelo de um documento do Calc, contendo atributos e uma ou mais planilhas. Ele inclui o serviço OfficeDocument e possui as propriedades:

```
XNamedRanges NamedRanges => coleção de extensões nomeadas no documento
XDatabaseRanges DatabaseRanges => coleção de extensões de dados no documento
XLabelRanges ColumnLabelRanges => coleção de extensões de colunas no documento
XLabelRanges RowLabelRanges => coleção de extensões de linhas no documento
XNameAccess SheetLinks => coleção de vínculos de planilhas no documento
XAreaLinks AreaLinks => coleção de vínculos de áreas no documento
XNameAccess DDELinks => coleção de vínculos DDE no documento
```

Entre as interfaces implementadas por SpreadsheetDocument, temos:

Interface com.sun.star.sheet.XSpreadsheetDocument, com o método:

```
com.sun.star.sheet.XSpreadsheets getSheets ( )
```

retorna a coleção com.sun.star.sheet.Spreadsheets com as planilhas do documento

Interface com.sun.star.util.XProtectable, cujos métodos são:

```
void protect ( string  sSenha )
```
ativa a proteção do documento

```
void unprotect ( string  sSenha )
```
remove a proteção do documento

```
boolean isProtected ( )
```
retorna True se o documento estiver protegido

Interface com.sun.star.sheet.XCalculatable e seus métodos:

```
void calculate ( )
```
recalcula todas as células com erros

```
void calculateAll ( )
```
recalcula todas as células

```
boolean isAutomaticCalculationEnabled ( )
```
retorna True se o cálculo automático estiver ativo

```
void enableAutomaticCalculation ( boolean bFlag )
```
se bFlag for True, ativa o cálculo automático, senão desativa

Interface com.sun.star.document.XActionLockable, para controlar a atualização automática do conteúdo das células, através dos métodos:

```
boolean isActionLocked ( )
```
retorna True se pelo menos um bloqueio existe

```
void addActionLock ( )
```
incrementa o contador de bloqueio

```
void removeActionLock ( )
```
decrementa o contador de bloqueio

Além destas, as seguintes interfaces são implementadas:

com.sun.star.lang.XMultiServiceFactory => para criar diversos objetos de planilhas
com.sun.star.frame.XModel => métodos para o modelo, derivada de XComponent
com.sun.star.style.XStyleFamiliesSupplier => famílias de estilos (PageStyles e CellStyles)
com.sun.star.drawing.XDrawPagesSupplier => acessa as páginas de desenho do documento
com.sun.star.document.XLinkTargetSupplier => acessa a coleção de vínculos do documento
XDocumentAuditing => com um método para repor os indicadores de auditoria
XConsolidatable => contém métodos para consolidação de dados
XGoalSeek => contém um método para atingir metas
com.sun.star.util.XNumberFormatsSupplier => acessa a coleção de formatos numéricos

A coleção com.sun.star.sheet.Spreadsheets contém as folhas de planilha do documento. Já vimos que podemos usar o acesso nomeado, indexado e sequêncial para obter uma folha.

Através da interface XSpreadsheets podemos inserir, mover e copiar folhas:

```
void insertNewByName ( string sNome, short iPosicao )
```
insere uma nova planilha na posição

```
void moveByName ( string sNome, short iDestino )
```

move a planilha para o destino

`void copyByName ( string sPlanilhaOrigem, string sDestino, short iPosicao )`
copia a planilha de origem para destino na posição definida

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

```
Sub exemploDocumentoCalc
   ' cria documento do Calc
   sUrl = "private:factory/scalc"
   oDoc = StarDesktop.loadComponentFromURL(sUrl, "_blank", 0, Array())
   ' exibe a propriedade casas decimais padrão do documento
   MsgBox oDoc.StandardDecimals, 176, "Casas Decimais"
   ' obtém a coleção de planilhas existentes no documento
   oPlanilhas = oDoc.getSheets()
   sMsg = ""
   ' visita cada planilha da coleção
   For n=0 To oPlanilhas.Count - 1
      ' usando o acesso indexado
      oPlanilha = oPlanilhas.getByIndex(n)
      sMsg = sMsg + oPlanilha.getName() + Chr$(13)
   Next n
   MsgBox sMsg + Str$(oPlanilhas.Count), 176, "Planilhas:"
   ' usando o acesso nomeado
   If oPlanilhas.hasByName("Planilha2") Then
      oPlanilha = oPlanilhas.getByName("Planilha2")
      MsgBox oPlanilha.PageStyle, 176, "Estilo de página"
   End If
   ' insere uma folha no início
   MsgBox "Planilha4 no início", 176, "Inserindo planilha"
   oPlanilhas.insertNewByName("Planilha4",0)
   ' move para o final
   MsgBox "Planilha4 no final", 176, "Movendo planilha"
   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.

Controlador

As operações sobre a vista do documento são da responsabilidade do objeto controlador, que pode ser obtido com uma chamada ao método getCurrentController de XModel:

```
oControlador = oDocumento.getCurrentController ( )
```

O controlador é representado pelo serviço com.sun.star.sheet.SpreadsheetView, que implementa as interfaces:

Interface com.sun.star.sheet.XSpreadsheetView, com os métodos:

`com.sun.star.sheet.XSpreadsheet getActiveSheet ( )`
obtém a folha de planilha ativa (objeto com.sun.star.sheet.Spreadsheet)

`setActiveSheet ( com.sun.star.sheet.XSpreadsheet oPlanilha )`
define a folha de planilha ativa

Interface com.sun.star.sheet.XViewFreezable, que permite congelar colunas e linhas:

```
boolean hasFrozenPanes ( )
```
retorna True se tiver algum painel congelado

```
void freezeAtPosition ( long  nColuna, long  nLinha )
```
congela a divisão do painel na coluna e linha

Interface com.sun.star.sheet.XActivationBroadcaster para o evento de ativação de planilhas:

```
void addActivationEventListener ( XActivationEventListener  oListener )
```
cadastra um listener do evento de ativação de planilha

```
void removeActivationEventListener( XActivationEventListener  oListener )
```
remove o cadastro do listener do evento de ativação de planilha

A interface com.sun.star.sheet.XActivationEventListener possui o método:

```
activeSpreadsheetChanged ( ActivationEvent oEvento )
```
a estrutura com.sun.star.sheet.ActivationEvent possui o campo:

```
XSpreadsheet ActiveSheet => objeto representando a planilha ativa
```

Além destas, temos as interfaces:

```
XViewSplitable => permite a divisão da vista
XRangeSelection => permite a seleção interativa de uma extensão de células
com.sun.star.container.XIndexAccess => acessa a coleção de painéis da vista
com.sun.star.container.XEnumerationAccess => cria uma enumeração dos painéis
com.sun.star.view.XSelectionSupplier => acessa a seleção corrente, que pode ser:
    SheetCell - SheetCellRange - SheetCellRanges - Shape - Shapes
```

O serviço SpreadsheetView inclui com.sun.star.sheet.SpreadsheetViewSettings, que possui, dentre outras, as propriedades:

```
boolean ShowFormulas => controla a exibição das fórmulas ou dos seus resultados
boolean ShowZeroValues => ativa a exibição de valores zero
boolean ShowNotes => define a exibição de marcas para as notas de uma célula
boolean HasVerticalScrollBar => ativa a barra de rolagem vertical
boolean HasHorizontalScrollBar => ativa a barra de rolagem horizontal
boolean HasSheetTabs => ativa as abas de tabulação das folhas
boolean HasColumnRowHeaders => ativa os cabeçalhos de linhas e colunas
boolean ShowGrid => abilita a exibição da grade
com.sun.star.util.Color GridColor => define a cor da grade
boolean ShowHelpLines => define a exibição das linhas ao mover objetos
boolean ShowAnchor => define a exibição do símbolo de âncora ao selecionar objetos
boolean ShowPageBreaks => define a exibição das quebras de página
short ShowObjects => define a exibição de objetos embutidos
short ShowCharts => habilita a exibição de gráficos
short ShowDrawing => habilita a exibição de desenhos
short ZoomValue => define o valor do zoom (apenas para ZoomType igual a BY_VALUE)
DocumentZoomType ZoomType => define o tipo de zoom do documento
```

constantes (e valores) de com.sun.star.view.DocumentZoomType:

```
short OPTIMAL (0) => exibição ideal da seleção corrente
short PAGE_WIDTH (1) => largura da página ajustada na vista
short ENTIRE_PAGE (2) => define se uma página completa será exibida
short BY_VALUE (3) => ajuste relativo, definido em ZoomValue
short PAGE_WIDTH_EXACT (4) => a largura da página é ajustado na vista, incluindo o final
```

SpreadsheetView inclui também com.sun.star.sheet.SpreadsheetViewPane, que define as interfaces:

Interface com.sun.star.sheet.XViewPane, cujos métodos são:

`long getFirstVisibleColumn ( )`
retorna a primeira coluna visível

`void setFirstVisibleColumn ( long  nColuna )`
define a primeira coluna visível

`long getFirstVisibleRow ( )`
retorna a primeira linha visível

`void setFirstVisibleRow ( long nLinha )`
define a primeira linha visível

`com.sun.star.table.CellRangeAddress getVisibleRange ( )`
retorna o endereço da área visível

Interface com.sun.star.sheet.XCellRangeReferrer com o método:

`com.sun.star.table.XCellRange getReferredCells ( )`
retorna a extensão de células visível (sem acessar o modelo do documento)

Segue um exemplo demonstrando operações com a vista do documento:

```
Sub exemploControlador
   ' cria documento do Calc
   sUrl = "private:factory/scalc"
   oDoc = StarDesktop.loadComponentFromURL(sUrl, "_blank", 0, Array())
   ' obtém o controlador do documento
   oVista = oDoc.getCurrentController()
   ' define algumas propriedades
   MsgBox "Cabeçalhos e Grade", 176, "Ajustando propriedades"
   oVista.HasColumnRowHeaders = False
   oVista.ShowGrid = False
   ' obtém a folha de planilha ativa
   oFolha = oVista.getActiveSheet()
   MsgBox oFolha.Name, 176, "Folha ativa:"
   ' ativa uma nova planilha
   If oDoc.getSheets().hasByName("Planilha3") Then
      oPlan = oDoc.getSheets().getByName("Planilha3")
      oVista.setActiveSheet ( oPlan )
      MsgBox "Planilha3 ativada", 176, "Ativando planilha"
   End If
   ' altera a linha e coluna visível
   oVista.HasColumnRowHeaders = True
   oVista.ShowGrid = True
   ' coluna U, linha 50
   oVista.setFirstVisibleColumn ( 20 )
   oVista.setFirstVisibleRow ( 49 )
End Sub
```

Note que no Calc não temos o cursor da visão do documento.

Edição Básica

Antes de apresentar os principais objetos de um documento do Calc, vejamos como se dá a edição básica de células numa planilha.

Eis os passos necessários para editar células de uma planilha:

obter a planilha onde se encontram as células a editar
obter a extensão contendo as células a serem editadas, este passo é opcional
obter a célula a editar
definir o conteúdo da célula

Os métodos para solucionar cada passo já foram apresentados. Para o primeiro, usamos getSheets com o acesso nomeado ou indexado. Para os outros, podemos usar os métodos de XCellRange e XCell já apresentados na seção sobre *Tabelas*, ou seja:

Interface **com.sun.star.table.XCellRange** com os métodos:

```
XCell getCellByPosition ( long nCol, long nLin )
XCellRange getCellRangeByPosition (long nEsq, long nSup, long nDir, long nInf )
XCellRange getCellRangeByName ( string  sNome )
```

A posição de uma célula, dentro de uma extensão, é relativa ao início da extensão.

Interface **com.sun.star.table.XCell** com os métodos:

```
string getFormula ( )
void setFormula ( string sFormula )
double getValue ( )
void setValue ( double nValor )
com.sun.star.table.CellContentType getType ( )
long getError ( )
```

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
   ' cria um novo documento do Calc
   sUrl = "private:factory/scalc"
   oDoc = StarDesktop.loadComponentFromURL(sUrl, "_blank", 0, Array())
   ' define os títulos das colunas
   sTitCol = Array ("CATEGORIA","MATERIAL","QUANT","P. UNIT","P. TOTAL")
   ' obtém a primeira planilha no documento
   oPlanilha = oDoc.getSheets().getByIndex(0)
   ' visita as células A1, B1, C1 e D1
   For i = 0 To 4
       ' cria um objeto célula
       oCelula = oPlanilha.getCellByPosition(i, 0)
       ' escreve o título da coluna na célula
       oCelula.setString(sTitCol(i))
   Next i
   ' visita as células da extensão A2:D7
   ' percorre as linhas
   For i = 1 To 6
      ' percorre as colunas
      For j = 0 To 4
```

```
         ' obtém a célula e define o seu
         ' conteúdo conforme a coluna
         oCelula = oPlanilha.getCellByPosition(j, i)
         If (j = 0) Then
            ' atribui uma cadeia para a célula
            If i < 4 Then
               oCelula.setString("A" + Str$(i))
            Else
               oCelula.setString("A" + Str$(i-3))
            End If
         ElseIf (j = 1) Then
            ' atribui uma cadeia para a célula
            oCelula.setString("Material de construção " + Str$(i))
         ElseIf (j = 2) Then
            ' atribui um valor para a célula
            oCelula.setValue(i * j)
         ElseIf (j = 3) Then
            ' atribui um valor para a célula
            oCelula.setValue(100 * Rnd())
         Else
            sLinha = Trim$(Str$(i+1))
            ' atribui uma fómula (multiplica valores de células)
            sFormula = "=C" + sLinha + " * " + "D" + sLinha
            ' atribui a formula à célula
            oCelula.setFormula(sFormula)
         End If
      Next j
   Next i
   ' obtém a célula A8
   oCelula = oPlanilha.getCellByPosition(0, 7)
   ' atribui uma cadeia à célula
   oCelula.setString("CUSTO TOTAL")
   ' define uma fórmula
   sFormula = "=Sum(E1:E6)"
   ' obtém a célula D8
   oCelula = oPlanilha.getCellByPosition(4, 7)
   ' atribui a fórmula ao conteúdo da celula
   oCelula.setFormula(sFormula)
End Sub
```

Analise o código e a saída identificando cada um dos passos da edição.

10.3 Folhas da Planilha

O serviço com.sun.star.sheet.Spreadsheet representa uma Folha de Planilha num documento do Calc e inclui os serviços SheetCellRange e Scenario. Eis as suas propriedades:

```
boolean IsVisible => determina se a folha de planilha será visível ou não
string PageStyle => determina o estilo de página da folha de planilha
boolean AutomaticPrintArea => [opcional] define se a área de impressão da folha é automática
```

Além destas, existem as propriedades herdadas de SheetCellRange.

Podemos executar uma série de operações sobre uma folha de planilha. Para isto, Spreadsheet implementa diversas interfaces, entre as quais:

A interface sheet.XSpreadsheet contém métodos para a criação de um cursor de célula:

```
com.sun.star.sheet.XSheetCellCursor createCursor ( )
```

cria um cursor para navegar pelas células da planilha (com.sun.star.sheet.SheetCellCursor)

```
XSheetCellCursor createCursorByRange ( XSheetCellRange oExtensao )
```
cria um cursor numa extensão de células (com.sun.star.sheet.SheetCellCursor)

A interface com.sun.star.container.XNamed acessa o nome da folha de planilha.

```
string getName ( )
void setName( string  sNome )
```
obtém / define o nome da folha de planilha

A interface XDataPilotTablesSupplier acessa a coleção de tabelas dinâmicas.

```
com.sun.star.sheet.XDataPilotTables getDataPilotTables ( )
```
retorna a coleção de tabelas dinâmicas

A interface XScenariosSupplier acessa a coleção de cenários.

```
com.sun.star.sheet.XScenarios getScenarios ( )
```
retorna a coleção de cenários

A interface XSheetAnnotationsSupplier acessa a coleção de anotações.

```
com.sun.star.sheet.XSheetAnnotations getAnnotations ( )
```
retorna a coleção de anotações

A interface com.sun.star.drawing.XDrawPageSupplier acessa a página de desenho da folha. No Calc, cada folha tem a sua própria página de desenho.

```
com.sun.star.drawing.XDrawPage getDrawPage ( )
```
retorna a página de desenho da folha de planilha

A interface com.sun.star.table.XTableChartsSupplier acessa a coleção de objetos gráficos.

```
XTableCharts getCharts ( )
```
retorna a coleção de objetos gráficos

A interface XCellRangeMovement com métodos para mover células dentro do documento.

```
void insertCells ( com.sun.star.table.CellRangeAddress vEnd, CellInsertMode nModo )
```
insere as células

```
void removeRange ( CellRangeAddress vEnd, CellDeleteMode nModo )
```
remove a extensão de células

```
void moveRange (CellAddress vDestino, CellRangeAddress  vOrigem )
```
move a extensão de origem para o endereço de destino

```
void copyRange ( CellAddress vDestino, CellRangeAddress  vOrigem )
```
copia a extensão definida em origem para o endereço de destino

Estes métodos recebem, como parâmetros, os tipos UNO abaixo:

a estrutura com.sun.star.table.CellRangeAddress contém os campos:

```
short Sheet => índice da planilha com a extensão de células
long StartColumn => índice da coluna esquerda da extensão
long StartRow => índice da linha superior da extensão
long EndColumn => índice da coluna direita da extensão
long EndRow => índice da linha inferior da extensão
```

a estrutura com.sun.star.table.CellAddress contém os campos:

```
short Sheet => índice da planilha com a célula
long Column => índice da coluna da célula
```

```
long Row => índice da linha da célula
```

eis as constantes da enum com.sun.star.sheet.CellInsertMode:

```
NONE => nenhuma célula será movida
DOWN => as células abaixo da célula inserida serão movidas para baixo
RIGHT => as células a direita da célula inserida serão movidas para a direita
ROWS => linhas completas abaixo da inserção serão movidas para baixo
COLUMNS => colunas completas a direita da inserção serão movidas para a direita
```

as constantes da enum com.sun.star.sheet.CellDeleteMode são:

```
NONE => nenhuma célula será movida
UP => células abaixo da deleção serão movidas para cima
LEFT => células a direita da deleção serão movidas para a esquerda
ROWS => linhas completas abaixo da deleção serão movidas para cima
COLUMNS => colunas completas a direita da deleção serão movidas para a esquerda
```

A interface XPrintAreas acessa as definições de área de impressão da planilha.

```
< com.sun.star.table.CellRangeAddress > getPrintAreas ( )
```
retorna um vetor com os endereços das áreas de impressão

```
void setPrintAreas ( < com.sun.star.table.CellRangeAddress >  vAreaImpressao )
```
define as áreas de impressão, recebe um vetor com os endereços das extensões

```
boolean getPrintTitleColumns ( )
void setPrintTitleColumns ( boolean  bFlag )
```
obtém / define o estado da repetição de colunas na impressão (se bFlag True, ativa)

```
com.sun.star.table.CellRangeAddress getTitleColumns ( )
void setTitleColumns ( com.sun.star.table.CellRangeAddress  vColunasTitulo )
```
obtém / define as colunas que devem ser repetidas durante a impressão

```
boolean getPrintTitleRows ( )
void setPrintTitleRows ( boolean  bFlag )
```
obtém / define o estado da repetição de linhas na impressão (se bFlag True, ativa)

```
com.sun.star.table.CellRangeAddress getTitleRows ( )
void setTitleRows ( com.sun.star.table.CellRangeAddress  vLinhasTitulo )
```
obtém / define as linhas que devem ser repetidas durante a impressão

A interface XSheetPageBreak acessa as quebras de páginas da planilha.

```
< com.sun.star.sheet.TablePageBreakData > getColumnPageBreaks ( )
```
retorna um vetor com as quebras de coluna

```
< com.sun.star.sheet.TablePageBreakData > getRowPageBreaks ( )
```
retorna um vetor com as quebras de linha

```
void removeAllManualPageBreaks ( )
```
remove todas as quebras manuais

a estrutura com.sun.star.sheet.TablePageBreakData contém os campos:

```
long Position => índice da linha ou coluna da quebra
boolean ManualBreak => True se a quebra for manual, False se automática
```

A interface XSheetLinkable contém métodos para vinculação de dados.

```
SheetLinkMode getLinkMode ( )
void setLinkMode ( SheetLinkMode  nModo )
```
obtém / define o modo do vínculo

```
string getLinkUrl ( )
void setLinkUrl ( string  sUrl )
```
obtém / define o URL do vínculo

```
string getLinkSheetName ( )
void setLinkSheetName ( string  sNomePlanilha )
```
obtém / define o nome da folha de planilha vinculada

```
void link ( string sUrl, string sNomePlanilha, string sNomeFiltro,
            string  sOpcoesFiltro, SheetLinkMode  nModo )
```
ativa um vínculo conforme os argumentos

a enum com.sun.star.sheet.SheetLinkMode possui as constantes:

```
NONE => folha não será vinculada
NORMAL => valores e fórmulas serão copiados
VALUE => somente os valores serão copiados
```

As interfaces abaixo também são implementadas:

com.sun.star.util.XProtectable contém métodos para a proteção da planilha.

com.sun.star.sheet.XSheetOutline => acessa as definições de estrutura de tópicos (esquema)

com.sun.star.sheet.XSheetAuditing => funções para auditoria (detetive)

O objeto Spreadsheet pode usar os métodos das interfaces dos serviços incluídos, Scenario e SheetCellRange.

Exemplos:

Movimentando dados

Vinculando dados

10.4 Extensão de células (SheetCellRange)

11 Documentos do Draw

ESCREVER ?? demanda quase nula ??

12 Documentos do Impress

ESCREVER ?? demanda quase nula ??

13 Documentos do Base

REESCREVER

14 Diálogos

REESCREVER aproveitando material

15 Formulários

ESCREVER

16 Mais informações

ATUALIZAR INFORMAÇÕES

16.1 Na rede

OpenOffice.org : O lar do OpenOffice.org

http://www.openoffice.org

http://www.openoffice.org.br

Projeto de Documentação do OpenOffice.org:

http://documentation.openoffice.org

OpenOffice.org for Developers http://website.openoffice.org/developer

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

- OpenOffice.org 1.0.2 – Developer ' s Guide (~ 12 Mb)
- Manual de Referência da API do OpenOffice.org (~ 8 Mb)
- StarOffice 6.0 – Basic Programmer ' s Guide (~ 1 Mb)
- StarOffice 5.2 – Programmer ' s Tutorial (~ 1 Mb)
- Exemplos do Developer ' s Guide (~ 1 Mb)

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

Documentação, fóruns (inclusive sobre Macros) e notícias do OpenOffice.org.

OOOForum.Org http://www.oooforum.org

Fóruns do OpenOffice.org , inclusive sobre Macros e API.

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.

EDV – Systeme Kienlein http://kienlein.com/pages/oo.html

Contém o DevGuide, com exemplos do Developer's Guide e o **módulo Inspect** uma excelente ferramenta para análise de objetos. Outro achado é o DbMonitor, um aplicativo OOo Basic completo, para acesso a Bancos de Dados.

16.2 Com o autor

Aceito solicitações para correções e inclusão de novos tópicos. Qualquer contribuição será muito bem recebida, neste caso, os créditos serão dos contribuidores.

Se você tiver alguma dúvida sobre o OpenOffice.org Basic, talvez eu possa lhe ajudar. Por favor, antes de um contato direto, poste a sua dúvida num fórum apropriado, assim você estará contribuindo para o aumento do "know-how" da comunidade.

Se você desejar entrar em contato comigo, envie uma mensagem para:

noelsonduarte@globo.com

16.3 Histórico deste documento

Versão	Data	Alteração
β	18/03/2006	Conclusão do Capítulo 9 – Documentos do Writer
β	23/01/2006	Adotado os tipos de dados UNO na descrição dos métodos e propriedades
β	18/01/2006	Foi incluído o Capítulo 8 – Trabalhando com Documentos
β	09/01/2006	Foram incluídos os capítulos 6 e 7: Capítulo 07 – Principais Objetos e Interfaces Capítulo 06 – API do OpenOffice.org Visão Geral
β	20/12/2005	Liberado documento com os 5 primeiros capítulos, para receber críticas e sugestões da comunidade: Capítulo 05 – Programando Macros sem Objetos Capítulo 04 – Comando e Funções do OOoBasic Capítulo 03 – Organização do Programa OOoBasic Capítulo 02 – Linguagem OpenOffice.org Basic Capítulo 01 – Trabalhando com Macros

17 Créditos, Agradecimentos, Licença

ATUALIZAR INFORMAÇÕES

17.1 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>

17.2 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 os seus trabalhos, contribuem para o crescimento do OpenOffice.org.

Aos coordenadores do **OpenOffice.org – Projeto Brasil.**

17.3 Licença