

Programando Macros Para O OpenOffice.org

Versão **β_23_01_2006**

por

Noelson Alves Duarte

Copyright © Noelson Alves Duarte

É permitida a cópia, distribuição e / ou modificação deste documento, sob os termos da GNU Free Documentation License, Version 1.1 ou uma versão posterior publicada pela Free Software Foundation. Uma cópia da licença acompanha este documento.

Índice

<u>1 Trabalhando com Macros</u>	6
<u>1.1 Introdução</u>	6
<u>1.2 Estrutura Geral das Macros</u>	6
<u>1.3 Gravando Macros</u>	7
<u>1.4 Executando Macros</u>	8
<u>1.5 Organizando Macros</u>	8
<u>1.6 Atribuindo Macros a Eventos</u>	11
<u>1.7 Gerenciando Pacotes</u>	13
<u>2 Linguagem OpenOffice.org Basic</u>	16
<u>2.1 Componentes da linguagem</u>	16
<u>2.2 Ambiente de Desenvolvimento</u>	16
<u>2.3 Análise de um programa</u>	19
<u>2.4 Elementos do Basic</u>	22
<u>Palavras reservadas</u>	22
<u>Regras para nomes</u>	22
<u>Comentários</u>	23
<u>Tipos de dados internos</u>	23
<u>Declaração de variáveis</u>	23
<u>Constantes simbólicas</u>	24
<u>Expressões</u>	25
<u>2.5 Fluxo de controle da execução</u>	28
<u>Comando de Decisão If ... Then ... End If</u>	28
<u>Comando de Decisão Select Case</u>	29
<u>Comando de Repetição While ... Wend</u>	29
<u>Comando de Repetição For ... Next</u>	30
<u>Comando de Repetição Do ... Loop</u>	31
<u>Comandos de Desvio Incondicional</u>	32
<u>Comandos de Controle de Erros</u>	33
<u>2.6 Tipos de dados avançados</u>	33
<u>Vetores</u>	34
<u>Vetor de vetores</u>	35
<u>Tipo de dado definido pelo usuário</u>	36
<u>3 Organização do Programa OOoBasic</u>	38
<u>3.1 Comandos</u>	38
<u>3.2 Sub-rotinas</u>	38
<u>3.3 Funções</u>	39
<u>3.4 Passagem de Parâmetros</u>	40
<u>3.5 Escopo das variáveis</u>	41
<u>3.6 Chamada de Procedimentos</u>	42
<u>Procedimentos na mesma biblioteca</u>	42
<u>Procedimentos em bibliotecas diferentes</u>	43
<u>Procedimentos em bibliotecas dinâmicas</u>	44
<u>Procedimentos recursivos</u>	44
<u>3.7 Modelo Geral de uma Macro</u>	44
<u>4 Comandos e Funções do OOoBasic</u>	46
<u>4.1 Interação com o Operador</u>	46

4.2 Instruções de tipos de dados	47
Funções de verificação	47
Funções de conversão	47
4.3 Funções de cadeias de caracteres	48
4.4 Funções de Data e Hora	49
4.5 Funções Matemáticas	50
4.6 Instruções de arquivos e diretórios	50
Funções de administração	51
Funções de abertura e fechamento de arquivos	51
Funções de entrada e saída	51
4.7 Funções Vetoriais	52
4.8 Instrução With ... End With	52
4.9 Instruções Diversas	53
5 Programando Macros sem Objetos	54
5.1 Funções para o Calc	54
Funções recebendo células	54
Funções recebendo extensões de células	55
Funções Matriciais	56
5.2 Macros para Eventos	57
6 API do OpenOffice.org – Visão Geral	59
6.1 Introdução	59
6.2 UNO	59
6.3 Organização da API	60
Service	60
Interface	60
Struct	61
Exception	61
Enum	61
Constant	62
6.4 Mapeamento de dados	62
6.5 Guia de Referência da API	63
6.6 Objetos	65
Métodos do objeto	65
Propriedades do objeto	66
Analisando um objeto	67
6.7 Inspeção de objetos	69
Usando o IDE Basic	69
Usando o OOOBasic	69
Usando o XRay	70
6.8 Instruções UNO do OOOBasic	70
7 Principais objetos e interfaces	72
7.1 ServiceManager	72
7.2 Desktop	73
7.3 Coleções	74
Containeres	75
Acesso nomeado	75
Acesso indexado	76
Acesso seqüencial	77

7.4	Descritores	78
7.5	Listeners	78
7.6	Fluxos	80
	Acesso a arquivos	80
	Fluxos de entrada e saída	81
	Fluxos de texto	82
8	Trabalhando com Documentos	85
8.1	URL	85
8.2	Arquitetura FCM	86
	Objeto Frame	86
	Ativando Documentos Abertos	87
	Executando Comandos UNO	87
8.3	Filtros do OO.o	88
8.4	Descritor do Meio	88
8.5	Documentos	89
8.6	Interface XComponentLoader	91
8.7	Criando Documentos	92
8.8	Abrindo Documentos	93
8.9	Salvando e Exportando Documentos	93
8.10	Imprimindo Documentos	94
8.11	Fechando Documentos	95
8.12	Identificando documentos abertos	95
9	Documentos do Writer	97
9.1	Introdução	97
9.2	Acesso aos objetos	97
9.3	Editando texto	97
9.4	Parágrafos	98
9.5	Editando Texto usando Cursor	99
	Cursor de texto	99
	Cursor da tela	99
9.6	Formatando texto	100
9.7	Organização dos Estilos de Formatação	101
	Estilos de página	102
	Estilos de Parágrafos	102
9.8	Busca e Substituição	103
	Busca	103
	Substituição	104
9.9	Tabelas	104
	Linhas e Colunas	105
	Editando células	105
	Formatando células	105
9.10	Campos	106
	Identificando campos existentes	106
	Visitando e editando os campos inseridos	107
9.11	Cabeçalhos e Rodapés	108
9.12	Seleção	108
10	Documentos do Calc	110
11	Documentos do Draw	111

12 Documentos do Impress	112
13 Documentos do Base	113
14 Diálogos	114
15 Formulários	115
16 Mais informações	116
16.1 Na rede	116
16.2 Com o autor	117
16.3 Histórico deste documento	117
17 Créditos, Agradecimentos, Licença	118
17.1 Créditos	118
17.2 Agradecimentos	118
17.3 Licença	118

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

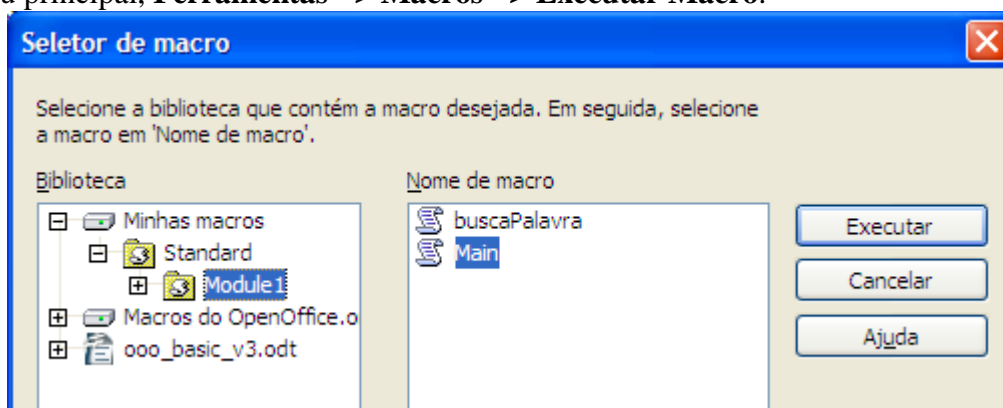
As macros do OpenOffice.org são organizadas num container do aplicativo ou do documento. As macros gravados num documento podem ser executadas apenas quando o documento estiver aberto. As macros do aplicativo podem ser executadas a qualquer momento.

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.

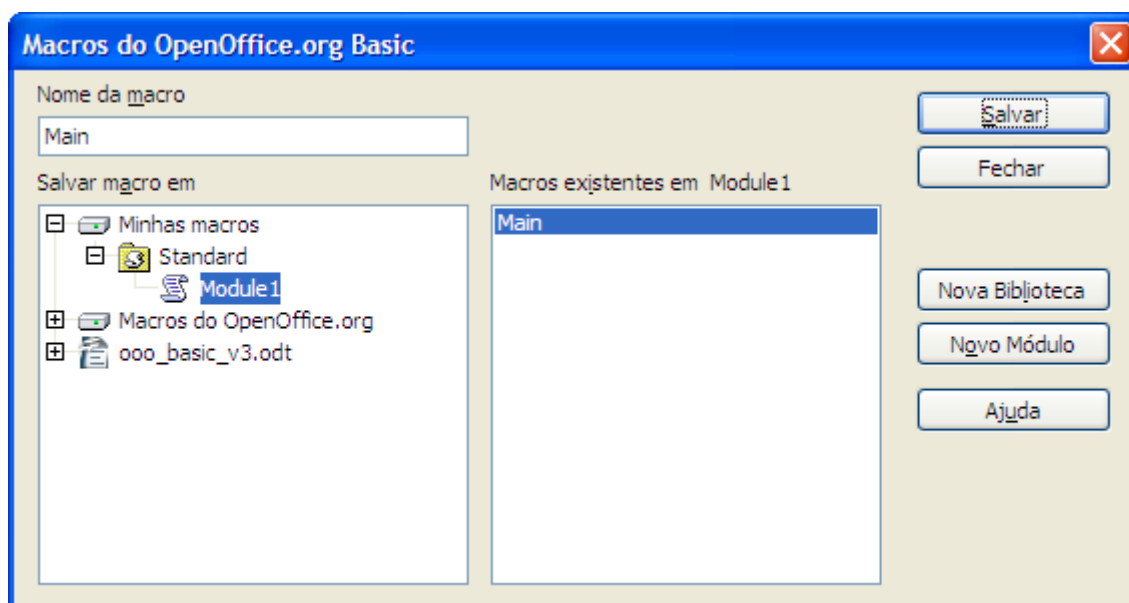
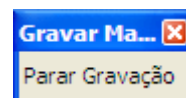
As bibliotecas Standard - do usuário ou do documento - são criadas automaticamente pelo OpenOffice.org. Ainda, o próprio OpenOffice.org gerencia, automaticamente, a carga destas bibliotecas para a memória.

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.

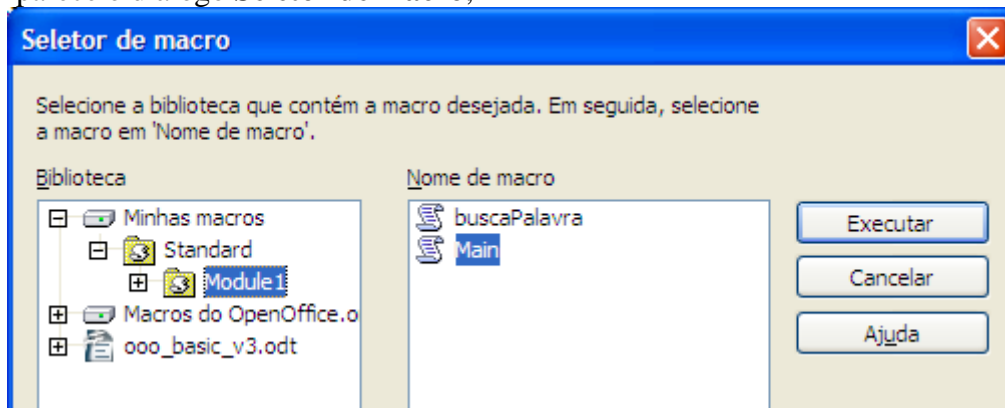
Antes de gravar a macro o usuário poderá criar uma biblioteca (botão **Nova Biblioteca**) e/ou um módulo (botão **Novo Módulo**). Mas, pela interface gráfica, uma nova biblioteca **não** pode ser criada no container **Macros do OpenOffice.org**, porque este container é protegido.

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**.

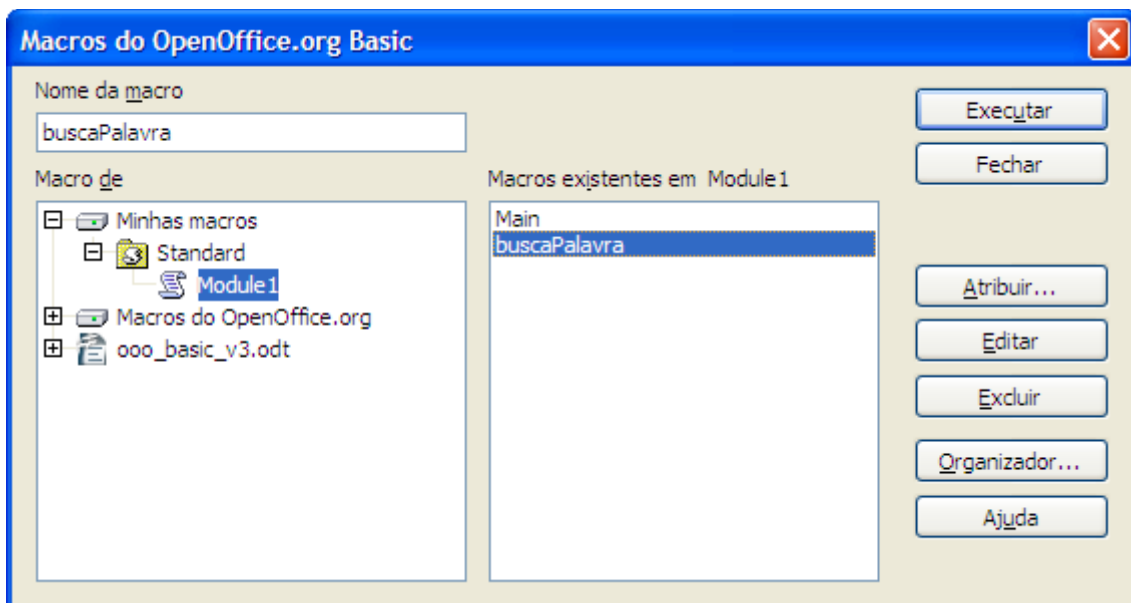
Este procedimento deve ser usado para rodar qualquer macro do OpenOffice.org. Contudo, podemos rodar macros a partir da linha de comando (útil para scripts). Veja alguns exemplos:

```
> 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:

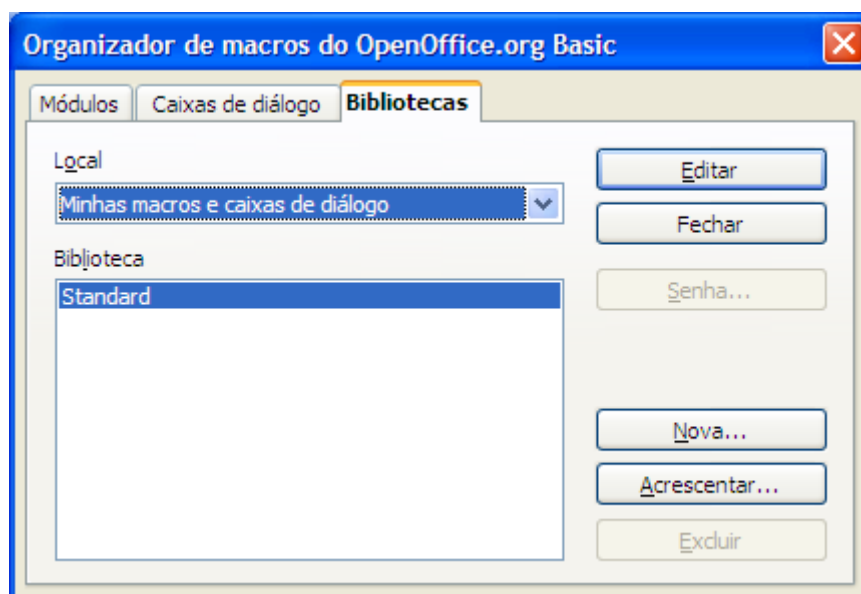
Botão Executar: outro modo para executar a macro selecionada. Aqui, apenas as macros do OOOBasic estão disponíveis. Com o diálogo **Seletores de macros**, podemos executar macros escritas em qualquer uma das linguagens suportadas pelo OpenOffice.org.

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.

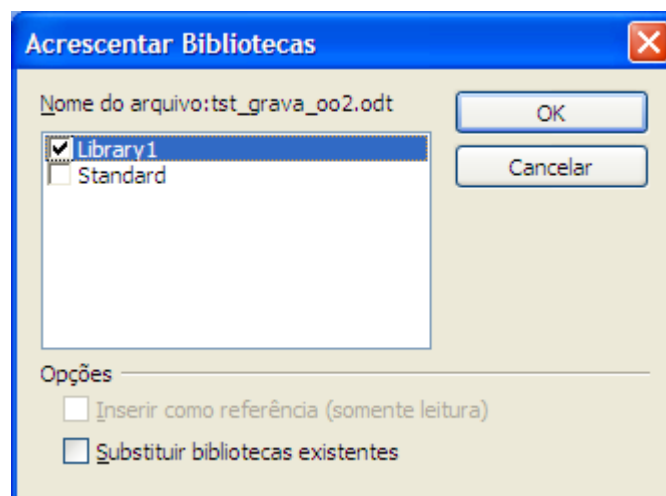
Botão Organizador: ativa o diálogo **Organizador de Macros do OpenOffice.org Basic**. Este diálogo permite operações sobre módulos, caixas de diálogos e bibliotecas. Clique sobre o botão para ativar o diálogo e vamos analisá-lo.



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.

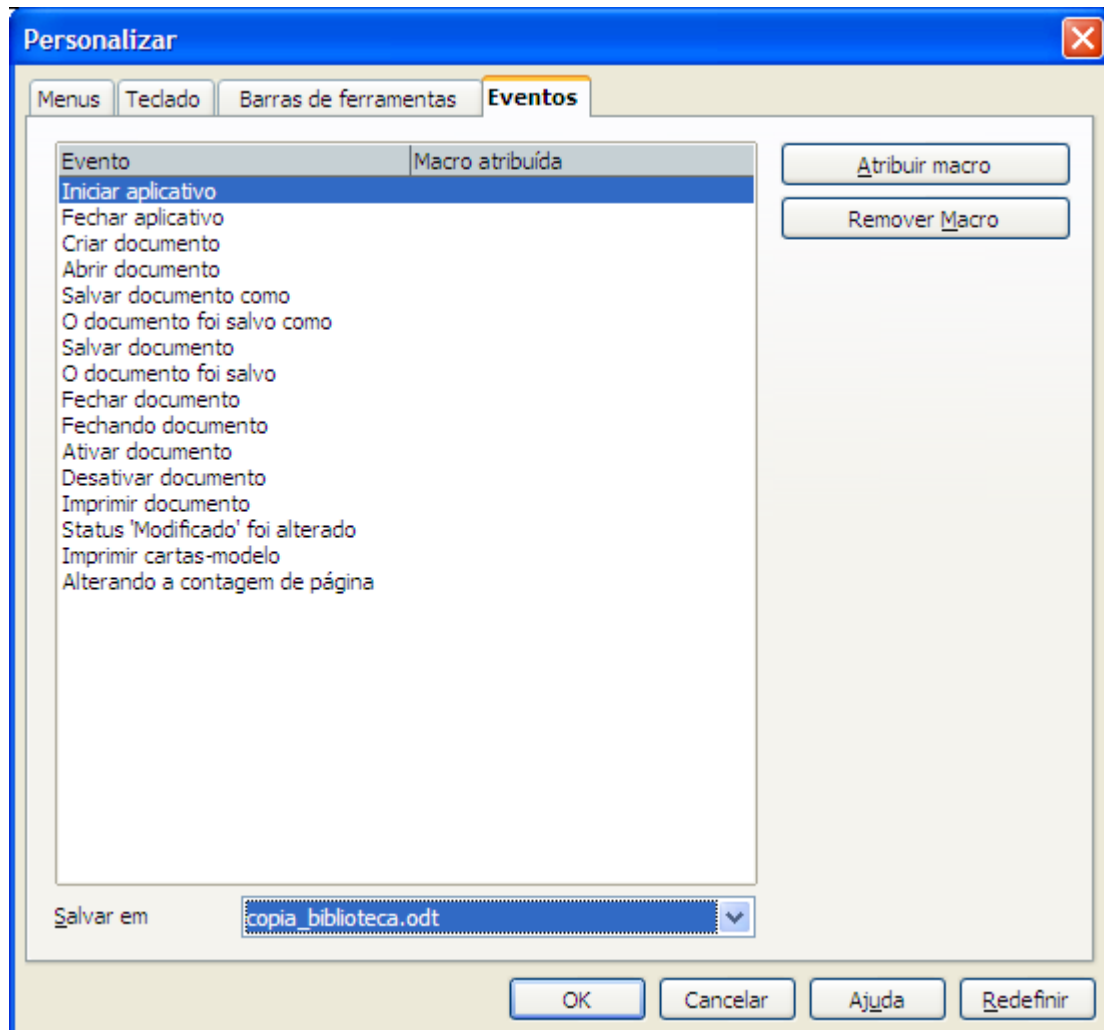
Finalmente, as bibliotecas do container **Macros e caixas de diálogo do OpenOffice.org** são gravadas como um subdiretório, com o nome igual ao da biblioteca, subordinado ao diretório <OOo_Install>/share/basic. As bibliotecas do container **Minhas macros e caixas de diálogo** são gravadas, também, em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

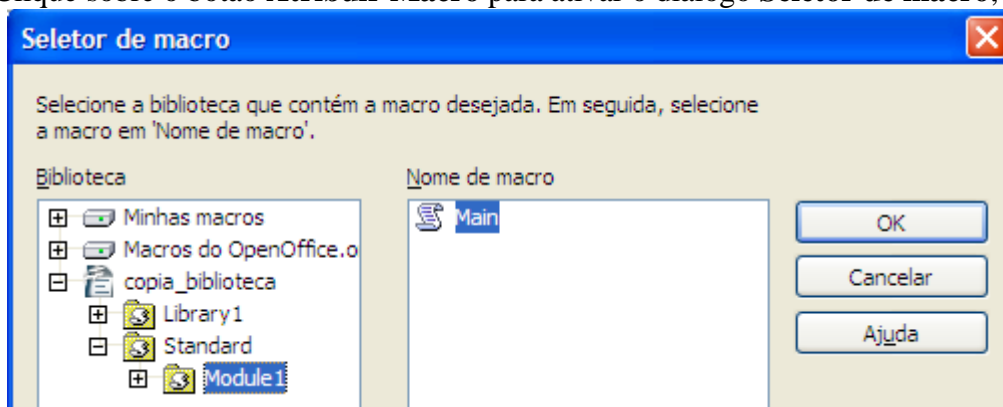
Um evento do aplicativo ocorre para todo o OpenOffice.org e um evento do documento ocorre somente no documento. Por exemplo, o evento Fechar Documento está disponível para o aplicativo e para um documento. Uma macro atribuída a este evento do aplicativo será executada quando qualquer documento do OpenOffice.org 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**;



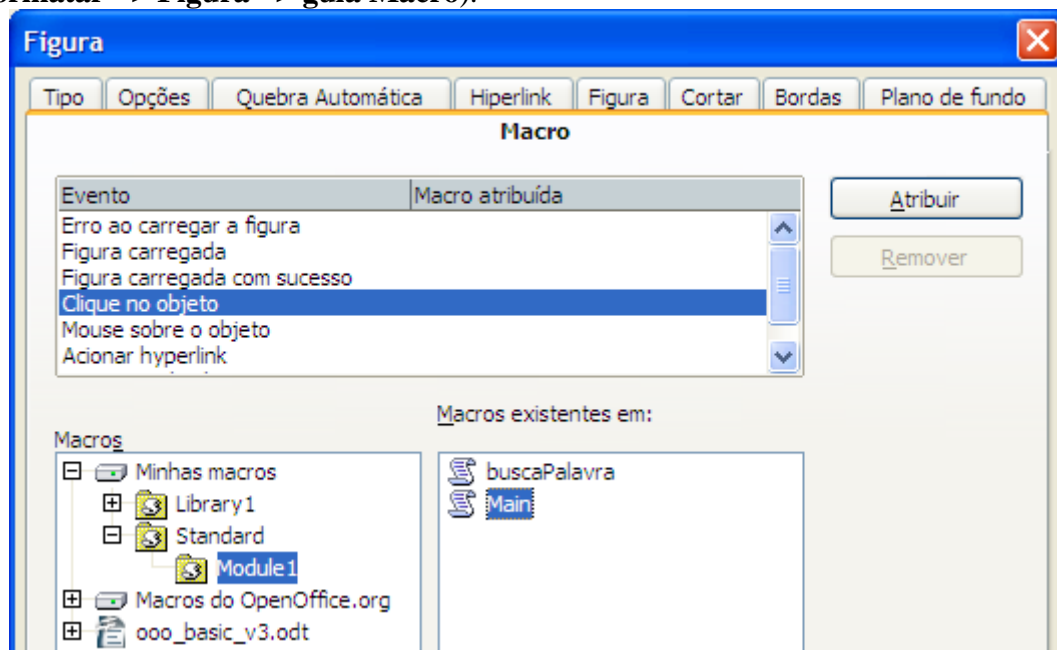
- Selecione o módulo e a macro e clique sobre o botão **OK**. Note que o URL da macro será acrescentada na coluna **Macro atribuída**, ao lado do evento selecionado.

- 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
```

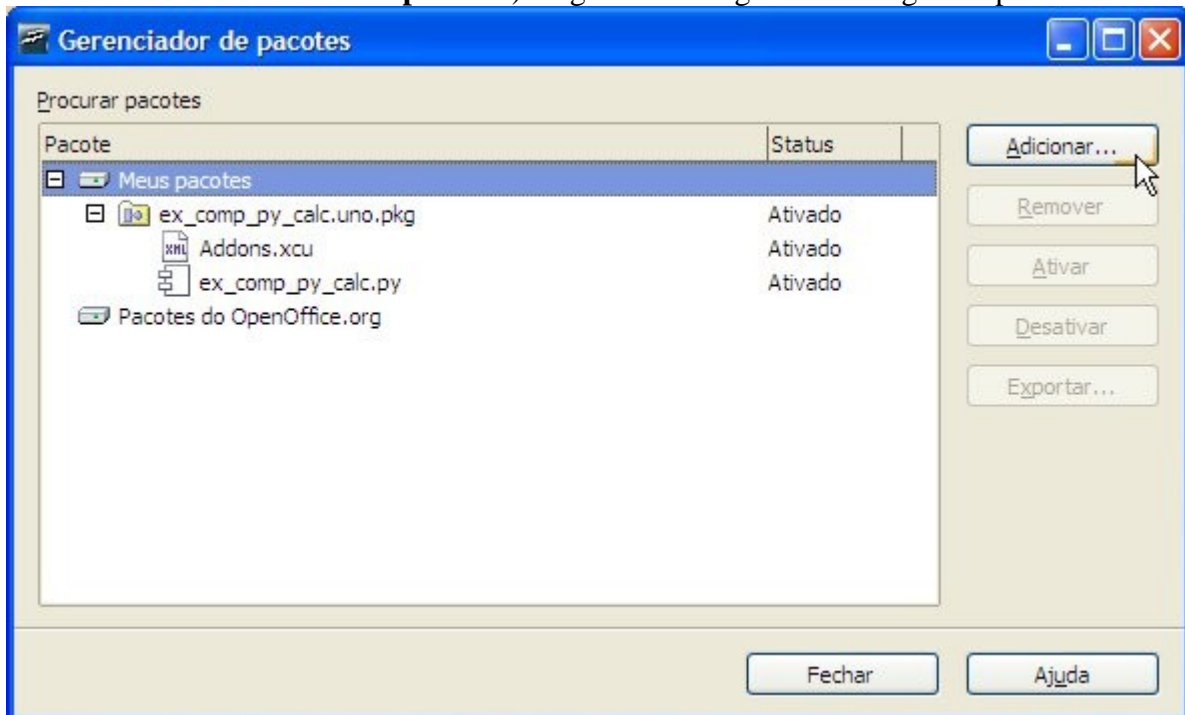
A versão 2.0 do OpenOffice.org traz um novo gerenciador de pacotes, o programa **unopkg.exe**, também localizado no diretório **<ooo_install>/program**. Evite usar o pkgchk nesta versão, pois o unopkg reconhece os pacotes das versões anteriores. Para ver as suas opções, digite na linha de comando do 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** (Beginner's All-purpose Symbolic Instruction Code) 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 disso, 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 pré-definidas 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 possui um Ambiente de Desenvolvimento Integrado (IDE Basic) que oferece as funções básicas para a programação de macros OOoBasic.

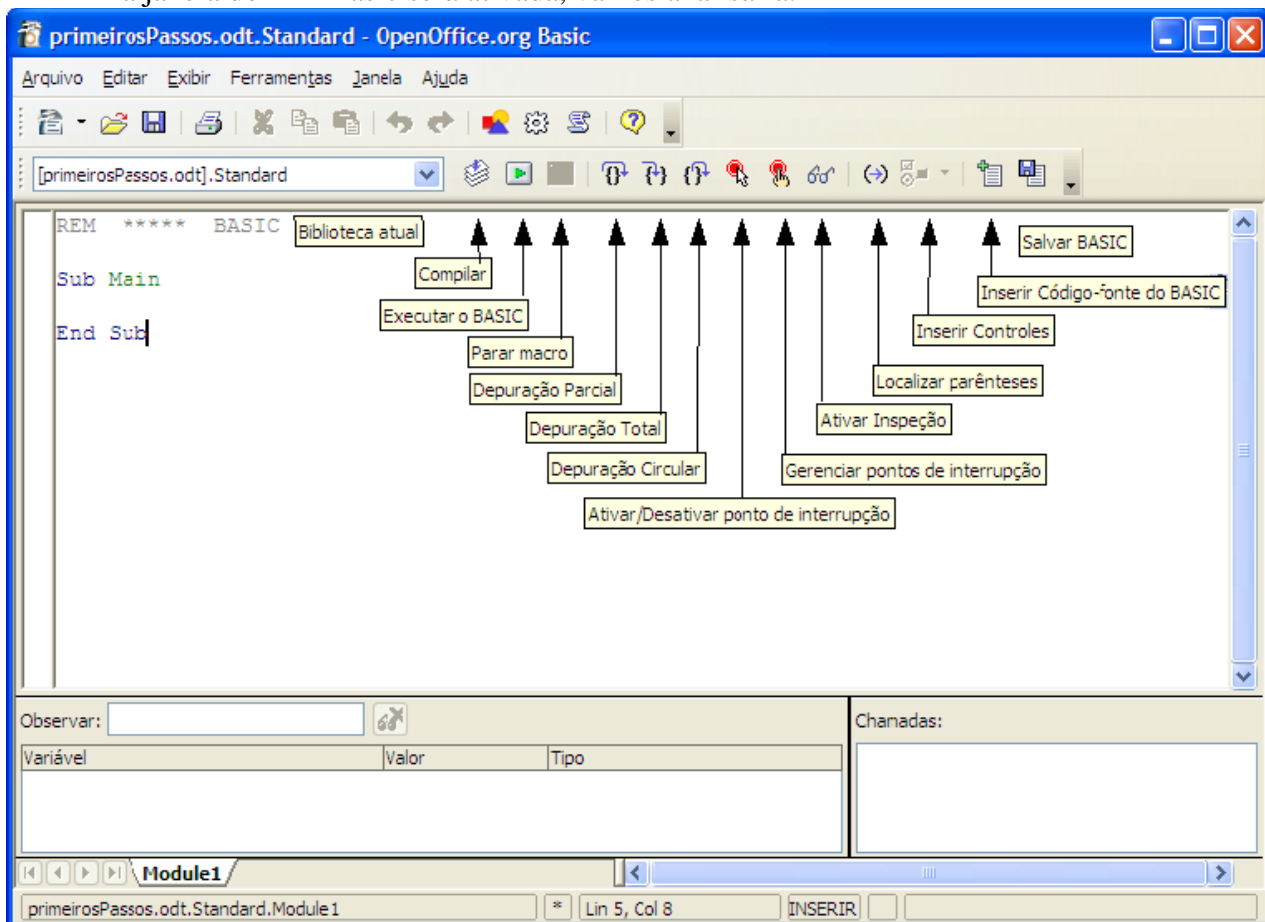
Entre as suas funcionalidades cabe ressaltar:

- 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 OoBasic (no IDE tecla F1).

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

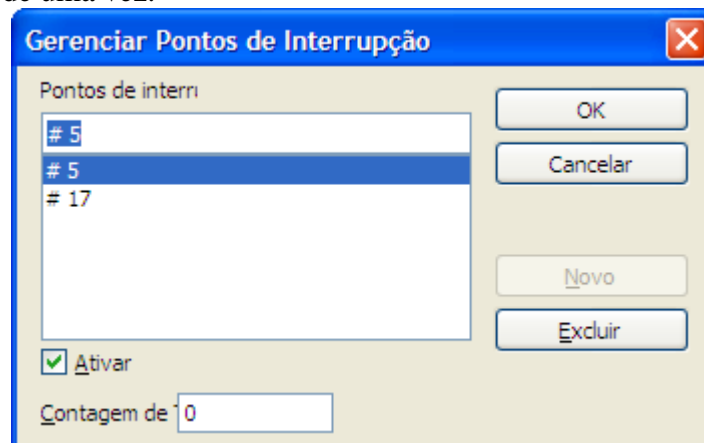
Depuração Parcial: executa uma macro comando a comando, as chamadas de rotinas são tratadas como um só comando. Durante a depuração, uma **seta amarela**, na **barra de indicação**, aponta para a linha a ser executada no próximo passo.

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 interrupção são exibidos. O campo **Contagem de Passagens** indica que a macro será interrompida após a *n*ésima 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, °celsius, 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 pré-definidas: 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 atribuído 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 Raiol for = a Raio2 o resultado é True, senão False
resultado = ( Raiol = 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
^	Exponenciação
*, /	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
```

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
  ' comandos a serem executados
```

```
' 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 código 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 substituída 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 OoBasic 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
  Next i
End Sub

```

```
' 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 poderíamos 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 intrínsecas (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 variáveis
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 função 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**:

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

Sub exemploEscopo
  inicializaVariaveis
  ' varLocal visível 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 visível 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 chamada `Procedimentos` (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 getProductNome da biblioteca Tools
print getProductNome & " - " & 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 Algoritmos.

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 possui uma grande quantidade de instruções (comandos e funções). Estas instruções compõem 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 diretório
 Rmdir (caminho) => remove um diretório
 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 saída 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 saída ( 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:

```
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 funções 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.

The screenshot shows a spreadsheet with a formula bar at the top containing the function `=VOLUMEESFERA(B6)`. Below the formula bar is a table with three columns: B, C, and D. Column B is labeled 'RAIO' and contains values 1, 2, 3, 4, and 5. Column C is labeled 'VOLUME' and contains values 4,19, 33,51, 113,1, 268,08, and 523,6. The cell containing 523,6 is highlighted with a black border.

B	C	D
RAIO	VOLUME	
1	4,19	
2	33,51	
3	113,1	
4	268,08	
5	523,6	

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.

B	C	D
5	1	
	2	
	3	
	4	
	5	
		75

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:

B	C	D	E
1	2	(1, 1)	(1, 2)
3	4	(2, 1)	(2, 2)
5	6	(3, 1)	(3, 2)
7	8	(4, 1)	(4, 2)

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 contêm 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`

<code>Width</code>	- valor do tipo longo (long), especificando a largura
<code>Height</code>	- 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:

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

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

```
oBotaol.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

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

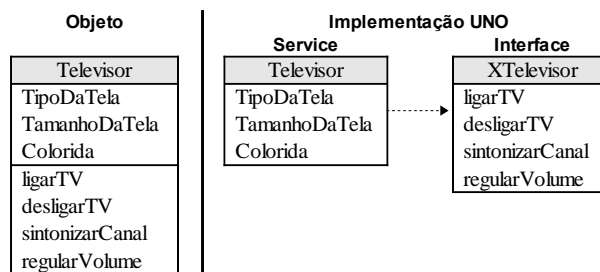
Em todos os casos, um **tipoUNO** pode representar um dos tipos básicos ou um objeto da API, por exemplo uma interface.

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:

Name => nome da propriedade

Handle => identificador da propriedade

Value => valor da propriedade ou void

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

Os objetos com propriedades suportam a interface **`com.sun.star.beans.XPropertySet`**, entre os seus métodos temos:

```
any getPropertyValue ( string sNomePropriedade )
```

obtem 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
```

Analizando 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 consequência 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
```

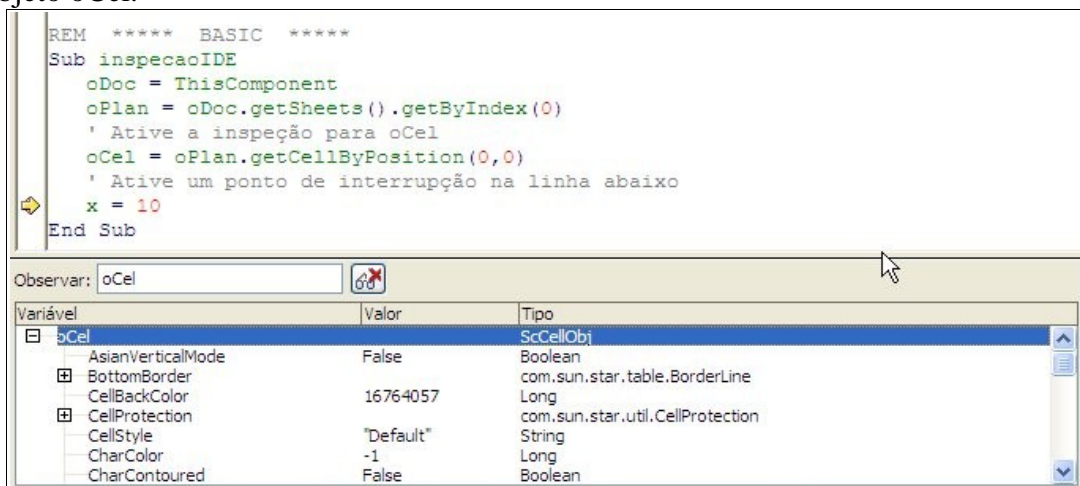
Por favor, leia atentamente os comentários.

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 propriedades 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 Marcellly** e está disponível nas páginas web abaixo:

Última versão (inglês): <http://www.oocomacs.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 indefinido.

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 com os 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
iTTotal = UBound(oSvManager.getAvailableServiceNames()) + 1
msgBox iTTotal, 176, "Serviços disponíveis no OoO"
' exibe o total de serviços disponíveis para o documento
iTTotal = UBound(ThisComponent.getAvailableServiceNames()) + 1
msgBox iTTotal, 176, "Serviços disponíveis no Documento"
'
' exibe o total de serviços suportados pelo ServiceManager
iTTotal = UBound(oSvManager.getSupportedServiceNames()) + 1
msgBox iTTotal, 176, "Serviços suportados pelo ServiceManager"
' exibe o total de serviços suportados pelo documento
iTTotal = UBound(ThisComponent.getSupportedServiceNames()) + 1
msgBox iTTotal, 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 exhibe o título do frame corrente:

```
Sub exemploDesktop
' obtém o Desktop
oDT = StarDesktop
' obtém o frame corrente
oCFrame = oDT.getCurrentFrame()
' exhibe 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 sequencial**. 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.getByNamed("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.hasMoreElements() 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 seqüencial

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$(10)
        n = n + 1
    Loop
    ' exhibe mensagem
    msg = msg & Str$(n) & " planilhas acessadas."
    msgBox msg , 176, "Acesso Seqüencial"
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 containers 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:

```
oDescriptorBusca = oDocumento.createSearchDescriptor ( )
```

Entre as propriedades de `SearchDescriptor`, temos:

```
boolean SearchBackwards => se True busca para o início 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 ( )
```

obtem 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

Vejam os resumos 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 célula 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 célula )
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 getDateModified (string sUrlArquivo)`

retorna a data da última modificação do arquivo

`< string > getFolderContents (string sUrlPasta, boolean bFlag)`

retorna um vetor com os nomes dos arquivos. Se bFlag True, inclui 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 ( )
```

obtem 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
oSaida = oSFA.openFileWrite ( sFluxo )
' cria o objeto de saída de texto
oFSTexto = createUnoService ("com.sun.star.io.TextOutputStream")
' atribui o fluxo aberto ao objeto de saída de texto
oFSTexto.setOutputStream (oSaida)
' 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 saída 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 conteúdo 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)
' exhibe
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 possível 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 getWindow ()
retorna a janela principal do frame
```

```
com.sun.star.awt.XWindow getComponentWindow ()
retorna a janela do componente
```

```
com.sun.star.frame.Controller getController ()
```

retorna o objeto controlador

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

```
com.sun.star.frame.FrameContainer getFrames ()
```

retorna um container com os frames

```
com.sun.star.frame.Frame getActiveFrame ()
```

retorna o frame ativo

```
void setActiveFrame ( com.sun.star.frame.Frame oFrame )
```

define o frame ativo

Ativando Documentos Abertos

Vejam os 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

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 ()` As Boolean

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 `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:\000exemplos\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, definidas no serviço

com.sun.star.view.PrintOptions, cujas propriedades 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 responsabilidade 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 título 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

RASCUNHO DO CAPÍTULO (NÃO REVISAR)

9.1 Introdução

Conteúdo de um documento texto

Parágrafos com texto

Estilos de formatação (caracteres, parágrafos, páginas, etc)

Tabelas, gráficos, desenhos, campos, etc

Definições de configuração do documento

9.2 Acesso aos objetos

Vários objetos do documento podem ser nomeados, indexados ou enumerados. Conforme o objeto, podemos usar o acesso nomeado, indexado ou enumerado. Alguns objetos devem estar ancorados num parágrafo, num caractere ou numa página.

Os objetos gráficos são desenhados na página de desenho do documento

Os principais serviços estão no módulo **com.sun.star.text**

TextDocument – TextRange – Text – TextCursor – TextTable – TextField

9.3 Editando texto

Obter o modelo do documento

Obter o texto a editar

Método getText ()

Usar os métodos da interface XTextRange

getString () As String

setString (sStr As String)

getStart () As XTextRange

getEnd () As XTextRange

Exemplo

Crie um documento e escreva uma macro para editar texto (observe as limitações desta edição)

```
Sub editaTexto  
oDoc = ThisComponent
```

```
oTxt = oDoc.getText()
sStr = "Este é o texto "
oTxt.setString ( sStr )
MsgBox " teclé Ok "
oTxtRng = oTxt.getEnd()
sStr = "a inserir no documento."
oTxtRng.setString ( sStr )
End Sub
```

9.4 Parágrafos

Acesso aos parágrafos do documento

Formam uma coleção que pode ser enumerada

Principais serviços

com.sun.star.text.Paragraph

com.sun.star.text.ParagraphEnumeration

com.sun.star.text.TextPortion

com.sun.star.text.TextPortionEnumeration

Exemplo

Preparar um documento e escrever uma macro para percorrer os seus parágrafos. Em seguida, alterar a macro para obter porções destes parágrafos.

```
Sub obtemParagrafos
oDoc = ThisComponent
Enum = oDoc.getText().createEnumeration()
While Enum.hasMoreElements()
oTxt = Enum.nextElement()
If oTxt.supportsService("com.sun.star.text.Paragraph") Then
MsgBox oTxt.getString()
End If
Wend
End Sub
```

```
Sub obtemPartesDoParagrafo
oDoc = ThisComponent
Enum = oDoc.getText().createEnumeration()
While Enum.hasMoreElements()
oTxt = Enum.nextElement()
If oTxt.supportsService("com.sun.star.text.Paragraph") Then
Enum2 = oTxt.createEnumeration()
While Enum2.hasMoreElements()
oTxtParte = Enum2.nextElement()
MsgBox oTxtParte.getString()
Wend
End If
Wend
End Sub
```

9.5 Editando Texto usando Cursor

Movendo-se pelo texto

Cursor de texto

Cursor de texto (com.sun.star.text.TextCursor)

XTextCursor – XWordCursor – XSentenceCursor – XParagraphCursor

Podem ser criados diversos cursores

Um cursor de texto pode ser expandido (seleção)

Cursor da vista é um objeto TextViewCursor

Criando um cursor de texto (XSimpleText)

createTextCursor () As TextCursor

createTextCursorByRange (oTxtRng) As TextCursor

insertString (oTxtRng, sStr, bFlag)

insertControlCharacter (oTxtRng, controlChar, bFlag)

Nota: TextCursor é também um TextRange

Principais métodos do objeto Cursor

goLeft (nChar, bSel) As Boolean - goRight (nChar, bSel) ...

gotoStart(bSel) – gotoEnd (bSel) – gotoRange(oRng, bSel)

collapseToStart() – collapseToEnd() – isCollapsed()

Verificar os métodos das outras interfaces no Guia de Referência

Cursor da tela

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.

oCV = oDoc.getCurrentController().getViewCursor()

Alguns métodos disponíveis:

screenDown – screenUP – jumpToPage (iNr) – getPage ()

goDown (iLin, bSel) – goUp (iLin, bSel) – gotoEndOfLine (bSel)

gotoStartOfLine (bSel) – IsAtEndOfLine – IsAtStartOfLine

Criando um cursor de texto a partir do cursor da vista

oCTxt = oTxt.createTextCursorByRange (oCV.getStart())

Exemplo

Escrever uma macro para editar texto usando um cursor

```

Sub exemploCursor
Dim mProp()
oDesktop = StarDesktop
oDoc = oDesktop.loadComponentFromURL("private:factory/swriter", "_
    "_blank", 0, mProp())
oTexto = oDoc.getText()
oCursor = oTexto.createTextCursor()
sStr = "Este é o texto do primeiro parágrafo do documento"
quebraParagrafo = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
With oTexto
    .insertString(oCursor, sStr, False)
    .insertControlCharacter(oCursor, quebraParagrafo, False)
    .insertString(oCursor, "Este é o segundo parágrafo ", False)
    .insertControlCharacter(oCursor, quebraParagrafo, False)
End With
' insere uma nova página
oCursor.gotoStartOfParagraph(True)
oCursor.breakType = com.sun.star.style.BreakType.PAGE_BEFORE
oTexto.insertString(oCursor, "Estou numa nova página.", False)
' move o cursor sem expansão
oCursor.gotoStart(False)
oCursor.gotoNextWord(False)
oCursor.gotoNextWord(False)
oCursor.gotoNextWord(False)
' move o cursor expandindo, i.é, seleciona "texto do"
oCursor.gotoNextWord(True)
oCursor.gotoNextWord(True)
' substitui o texto entre o início e o final do cursor
oTexto.insertString(oCursor, "", True)
' usando o cursor da vista
oCV = oDoc.getCurrentController().getViewCursor()
oCV.jumpToPage(1)
oCV.goDown (1, True)
End Sub

```

9.6 Formatando texto

Tipos de formatação de texto

Formatação direta

Formatação com estilos pré-definidos

Formatando caracteres

Propriedades no serviço com.sun.star.style.CharacterProperties

CharFontName – CharPosture – CharWeight – CharHeight

Formatando parágrafos

Propriedades no serviço com.sun.star.style.ParagraphProperties

ParaAdjust – ParaStyleName – ParaLeftMargin – ParaRightMargin

Exemplo

Acrescentar formatação na macro do exemplo anterior

```

Sub exemploFormatacao
  Dim mProp()
  oDesktop = StarDesktop
  oDoc = oDesktop.loadComponentFromURL("private:factory/swriter", "_
    "_blank", 0, mProp())
  oTexto = oDoc.getText()
  oCursor = oTexto.createTextCursor()
  sStr = "Este é o texto do primeiro parágrafo do documento"
  quebraParagrafo = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
  With oTexto
    .insertString(oCursor, sStr, False)
    .insertControlCharacter(oCursor, quebraParagrafo, False)
    .insertString(oCursor, "Este é o segundo parágrafo ", False)
    .insertControlCharacter(oCursor, quebraParagrafo, False)
  End With
  oCursor.gotoStartOfParagraph(True)
  oCursor.breakType = com.sun.star.style.BreakType.PAGE_BEFORE
  oTexto.insertString(oCursor, "Estou numa nova página.", False)
  oCursor.gotoStart(False)
  oCursor.gotoNextWord(False)
  oCursor.gotoNextWord(False)
  oCursor.gotoNextWord(False)
  oCursor.gotoNextWord(True)
  oCursor.gotoNextWord(True)
  oTexto.insertString(oCursor, "", True)
  ' código para formatação
  oCursor.gotoNextParagraph(False)
  oCursor.gotoNextWord(False)
  oCursor.gotoNextWord(False)
  oCursor.gotoNextWord(False)
  oCursor.gotoNextWord(True)
  oCursor.CharPosture = com.sun.star.awt.FontSlant.REVERSE_ITALIC
  oCursor.CharColor = RGB(255,0,0)
  ' aplica estilo de parágrafo
  oCursor.gotoNextParagraph(FALSE)
  oCursor.paraStyleName = "Heading"
End Sub

```

9.7 Organização dos Estilos de Formatação

- + Famílias de Estilos
- + Estilos de Páginas (Padrão – Índice – Primeira Página – etc)
- + Estilos de Parágrafos (Título1 – Padrão – Corpo do texto – etc)
- + Estilos de Caracteres (Padrão – Citação – Definição – etc)
- + Estilos de Molduras (Moldura – Rótulos – etc)
- + Estilos de Numeração (Lista 1 – Numeração 1 – etc)

Acessando as famílias

getStyleFamilies () => Coleção com todas as famílias

Métodos de manipulação de coleções (container)

Exemplo

Escrever uma macro para exibir os nomes das famílias de estilos

```
Sub exibeFamíliasDeEstilos
  oFamEstilos = ThisComponent.getStyleFamilies()
  sNomes = oFamEstilos.getElementNames()
  msg = ""
  For i=LBound(sNomes) To UBound(sNomes)
    msg = msg & sNomes(i) & Chr(13)
  Next i
  MsgBox ( msg, 0, "Famílias de Estilos" )
End Sub
```

Estilos de página

Propriedades definidas no serviço com.sun.star.style.PageStyle

LeftMargin – RightMargin – TopMargin – BottomMargin

IsLandscape – HeaderIsOn – FooterIsOn – BackColor

Estilos de Parágrafos

Propriedades no serviço com.sun.star.style.ParagraphStyle

Inclui os serviços ParagraphProperties e Style

Name – CharFontName – CharHeight – ParaAdjust

Exemplos

Macro para exibir as propriedades do estilo de página padrão

Escrever uma macro para criar e aplicar um estilo de parágrafo

```
Sub paginaPadrao
  oFamEstilos = ThisComponent.getStyleFamilies()
  oEstilosPag = oFamEstilos.getByName("PageStyles")
  If oEstilosPag.hasByName("Standard") Then
    oPadrao = oEstilosPag.getByName("Standard")
    Print oPadrao.Size.Width;" - ";oPadrao.Size.Height
  Else
    MsgBox "Estilo Standard não encontrado!"
  End If
End Sub
```

```
Sub novoEstiloParagrafo
  oDoc = ThisComponent
  oNovoEstilo = oDoc.createInstance("com.sun.star.style.ParagraphStyle")
  oFPara = oDoc.getStyleFamilies.getByName("ParagraphStyles")
  oFPara.insertByName("Meu_Novo_Estilo", oNovoEstilo)
  oNovoEstilo.CharFontName = "Arial"
  oNovoEstilo.CharHeight = 20
  oNovoEstilo.ParaAdjust = com.sun.star.style.ParagraphAdjust.CENTER
  oTexto = oDoc.getText()
```

```
oCursor = oTexto.createTextCursor()
oCursor.gotoStart(False)
oCursor.gotoEndOfParagraph(True)
oCursor.ParagraphStyleName = "Meu_Novo_Estilo"
End Sub
```

9.8 Busca e Substituição

Busca

Serviço com `sun.star.util.SearchDescriptor`

`SearchBackwards` – `SearchCaseSensitive` – `SearchWords`

Interface com `sun.star.util.XSearchDescriptor`

`getSearchString` – `setSearchString`

Interface com `sun.star.util.XSearchable`

`createSearchDescriptor`

`FindAll` – `FindFirst` – `FindNext`

Exemplos

Localizar todas as ocorrências de uma palavra num texto.

Buscar e alterar a cor das ocorrências de uma palavra num texto.

```
Sub buscaTexto
oDoc = ThisComponent
oBusca = oDoc.createSearchDescriptor()
sPalav = InputBox ( "Qual palavra ?" )
oBusca.setSearchString(sPalav)
oResultado = oDoc.findAll(oBusca)
MsgBox oResultado.getCount()
End Sub
```

```
Sub alteraTexto
oDoc = ThisComponent
oBusca = oDoc.createSearchDescriptor()
sPalav = InputBox ( "Qual palavra ?" )
oBusca.setSearchString(sPalav)
oResultado = oDoc.findFirst(oBusca)
Do Until IsNull ( oResultado )
oResultado.CharColor = RGB (255, 0, 0)
oResultado = oDoc.findNext (oResultado, oBusca)
Loop
End Sub
```

Substituição

Serviço com.sun.star.util.ReplaceDescriptor

Inclui o serviço SearchDescriptor

Interface com.sun.star.util.XReplaceDescriptor

getReplaceString – setReplaceString

Interface com.sun.star.util.XReplaceable

createReplaceDescriptor – replaceAll

Exemplo

Escrever uma macro para substituir as letras minúsculas de uma dada palavra por letras maiúsculas.

```
Sub trocaCaixa
  oDoc = ThisComponent
  oBusca = oDoc.createReplaceDescriptor()
  sPalav = InputBox ( "Qual palavra ?" )
  oBusca.setSearchString(sPalav)
  oBusca.setReplaceString(UCase(sPalav))
  nRes = oDoc.replaceAll(oBusca)
  Print nRes
End Sub
```

9.9 Tabelas

Criar o objeto Tabela

createInstance (“com.sun.star.text.TextTable”)

Interface com.sun.star.text.XTextTable

initialize (nLinhas, nColunas)

getCellNames – getCellByName – getRows – getColumns

createCursorByCellName

Inserir a tabela no documento

insertTextContent (oCursor, oTabela, bFlag)

Exemplo

Escrever uma macro para inserir uma tabela num documento

```
Sub insereTabela
  oDoc = StarDesktop.getCurrentComponent()
  oTab = oDoc.createInstance("com.sun.star.text.TextTable")
  oTab.initialize ( 5, 3 )
  oTxt = oDoc.getText()
  oCur = oTxt.createTextCursor()
  oTxt.insertTextContent ( oCur, oTab, False)
End Sub
```

Linhas e Colunas

Interfaces com.sun.star.text.XTableRows e XTableColumns

insertByIndex – removeByIndex – hasElements

getByIndex – getCount – getElementType

Extensões de células

Interface com.sun.star.table.XCellRange

getCellByPosition – getCellRangeByPosition – getCellRangeByName

Interface com.sun.star.text.XTextTableCursor

getRangeName – gotoCellByName – mergeRange – SplitRange

goLeft – goRight – goUp – goDown – gotoStart – gotoEnd

Editando células

Interfaces com.sun.star.table.XCell / XText

getFormula – set Formula (strFormula)

getValue – setValue (nValor)

Formatando células

Serviço com.sun.star.table.CellProperties

Serviço com.sun.star.text.TextTableCursor

Exemplo

Escrever uma macro para inserir e editar uma tabela num documento

```
Sub editaTabela
oDoc = StarDesktop.getCurrentComponent()
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")
oCelula.setString("Coluna A")
oCelula = oTab.getCellByName("B1")
oCelula.setString("Coluna B")
oCelula = oTab.getCellByName("C1")
oCelula.setString("Coluna C")
Dim oCurTab As Object
oCurTab = oTab.createCursorByCellName(oTab.CellNames(0))
oCurTab.gotoStart(False)
```

```

oCurTab.goRight(2, True)
oCurTab.paraStyleName = "Heading"
MsgBox oCurTab.getRangeName()
Dim sNomes() As Variant
Dim sNomeCelula As String
sNomes = Array("A","B","C")
For i% = 1 To 4
  For j% = 1 To 3
    sNomeCelula = sNomes(j%-1)+ Str$(i%+1)
    oCelula = oTab.getCellByName(sNomeCelula)
    If (j% - 1 = 2) Then
      oCelula.setValue(i% + 1)
    Else
      oCelula.setString(sNomeCelula)
    End If
  Next j%
Next i%
oCelula = oTab.getCellByName("C5")
oCelula.setFormula("sum <C2:C4>")
oCurTab.gotoCellByName("A5", False)
oCurTab.goRight(1, True)
oCurTab.mergeRange()
oCelula = oTab.getCellByName("A5")
oCelula.setString("Total")
End Sub

```

9.10 Campos

Alguns tipos de campos (módulo com.sun.star.text.textfield)

Simples: PageCount – PageNumber – DateTime – DropDown

Serviço com.sun.star.text.TextField.Tipo

Campos do tipo: Database – SetExpression – User

Necessitam de com.sun.star.text.FieldMaster.Tipo

Criar os objetos (TextField.Tipo e FieldMaster.Tipo)

createInstance (strTipoDoCampo)

Definir as propriedades(conforme o Tipo de Campo)

Name : Value : Content : CurrentPresentation : DateTimeValue

Ligar campos compostos e Inserir o campo no documento

oCampo.attachTextFieldMaster (oMaster)

insertTextContent (oCursor, oCampo, bFlag)

Identificando campos existentes

Obter as coleções de campos

getTextFields() => XEnumerationAccess (todos os campos)

Métodos refresh() e createEnumeration()

getTextFieldMasters() => XNameAccess (campos masters)

getElementNames()

getByName (strNomeCampoMaster)

Nome de campo Master segue o padrão:

com.sun.star.text.FieldMaster.Tipo.NomeCampo

Campo Master a partir do TextField

getTextFieldMaster () => retorna o campo master associado

Exemplos

Macro para inserir um campo do Usuário e um campo Data

```
Sub insereCampos
oDoc = ThisComponent
' cria um objeto campo data
oCampoData = oDoc.createInstance("com.sun.star.text.TextField.DateTime")
' insere o campo data
oDocTexto = oDoc.getText()
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
oDocTexto.insertControlCharacter(oCursor, _
com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, false)
' insere o campo do usuario na posicao do cursor
oDocTexto.insertTextContent(oDocTexto.getEnd(), oCampoUser, False)
End Sub
```

Visitando e editando os campos inseridos

```
Sub visitaCampos
oDoc = thisComponent
' percorre os campos
enumCampos = oDoc.getTextFields().createEnumeration()
Do While (enumCampos.hasMoreElements())
oCampo = enumCampos.nextElement()
msgBox oCampo.getPresentation(False)
Loop
'
```

```

' percorre os campos masters
Masters = oDoc.getTextFieldMasters()
sNomes = Masters.getElementNames()
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

```

9.11 Cabeçalhos e Rodapés

Inserindo um campo Nr de Página no rodapé da página

```

Sub insereCampoNoRodape
Doc = StarDesktop.getCurrentComponent()
'cria um campo Numero de Pagina
NrPagina = Doc.CreateInstance("com.sun.star.text.TextField.PageNumber")
' define propriedades
NrPagina.NumberingType = com.sun.star.style.NumberingType.ARABIC
NrPagina.SubType = com.sun.star.text.PageNumberType.CURRENT
' obtém estilos de pagina
EstPagina = Doc.StyleFamilies.getByName("PageStyles")
' obtém pagina padrao
PagPadrao = EstPagina("Standard")
' ativa o rodape
PagPadrao.FooterIsOn = True
' obtém o texto do rodape
TxtRodape = PagPadrao.FooterText.Text
' cria um cursor
CursorRodape = TxtRodape.createTextCursor()
' insere o campo
TxtRodape.insertTextContent(CursorRodape, NrPagina, False)
End Sub

```

9.12 Seleção

Objeto controlador do documento

```
oDocView = oDoc.getCurrentController ()
```

Seleção atual

```
oSel = oDoc.getCurrentSelection ()
```

```
oSel = oDocView.getSelection ()
```

O objeto retornado depende da seleção, pode ser:

TextRanges – TextTableCursor – drawing.Shape

Selecionando objetos

bResult = oDocView.select (oObjeto)

Exemplos

Escrever uma macro para obter o texto selecionado

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

Escrever uma macro para selecionar texto na interface gráfica

```
Sub selecionaTexto
  oDoc = ThisComponent
  oTxt = oDoc.getText()
  oDocView = oDoc.getCurrentController()
  If oDocView.select( oTxt ) Then
    MsgBox "Texto selecionado"
  Else
    MsgBox "Erro na seleção"
  End If
End Sub
```

10 Documentos do Calc

REESCREVER aproveitando material

11 Documentos do Draw

ESCREVER ?? demanda quase nula ??

12 Documentos do Impress

ESCREVER ?? demanda quase nula ??

13 Documentos do Base

REESCREVER aproveitando material (vai dar trabalho BASE tá cheio de BUGS)

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.ooforum.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	23/01/2006	Adotado os tipos de dados UNO na descrição dos métodos e propriedades
β	18/01/2006	Foi incluído o capítulo 8 Capítulo 08 – 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

É permitida a cópia, distribuição e / ou modificação deste documento, sob os termos da GNU Free Documentation License, Version 1.1 ou uma versão posterior publicada pela Free Software Foundation. Uma cópia da licença acompanha este documento, consulte o arquivo **FDL.TXT**. Se você não recebeu uma cópia deste arquivo, por favor informe ao autor ou ao “webmaster” do site que disponibilizou este documento.

Copyright © 2005 Nelson Alves Duarte.