1 Introdução

A linguagem **OpusWin** oferece amplos recursos para desenvolvimento de sistemas aplicativos em ambientes Windows e Internet, oferecendo, adicionalmente as seguintes vantagens e benefícios:

- Utilização de interfaces gráficas e recursos multimídia
- Integração nativa com os Bancos de Dados relacionais OpenBASE
- Utilização das interfaces padrão ODBC e SQL
- Implementação de recursos de programação WEB
- Acesso aos Bancos OpenBASE como objetos COM/DCOM a partir de diversas plataformas
- A linguagem OpusWin, subconjunto da tradicional OPUS, utiliza funções, comandos e opções desta poderosa linguagem

1.1 Objetivos deste manual

Este manual apresenta orientações para a correta utilização dos comandos, funções e recursos de programação introduzidos pela linguagem OpusWin. Consulte o Manual da OPUS em relação aos recursos de programação oferecidos por esta linguagem de programação.

1.2 Tópicos principais deste manual

Neste manual, agrupamos e apresentamos os recursos da OpusWin da seguinte forma:

- **Controle do ambiente de trabalho na OpusWin**
- **Gerência de DLLs na OpusWin**
- **Construção e utilização de Menus**
- **Construção e utilização de Caixas de Diálogo**
- **Construção e utilização de Caixas de Mensagem**
- **Construção e utilização de Barras de Status**
- **Construção e utilização de Barras de Ferramentas**
- **Construção e utilização de Folhas e Páginas de Propriedades**
- **Comandos e funções de acesso a Bancos de Dados**
- **Construção e utilização de Caixas de Diálogo para uso geral**
- **Funções para controle e manutenção do Registry**
- **Construção e utilização de outros comandos e funções da OpusWin**

1.3 Manuais dos produtos OpenBASE

Para obter maiores informações sobre as ferramentas OpenBASE para desenvolvimento de aplicativos consulte os seguintes manuais em nosso site www.openbase.com.br:

- Linguagem **OPUS** (ambientes Unix e Win32)
- Linguagem **OpusWin** (ambientes Win32)
- Interface visual para desenvolvimento de aplicativos OpusWin - **OpusVis**
- Sistema de impressão - **OpusRel** (ambientes Win32)
- Ambiente integrado de desenvolvimento - **OpusIDE** (ambientes Win32)
- Ambiente de Bancos de Dados - **OpenDBA** (ambientes Win32 e SQL)
- Desenvolvimento rápido de aplicativos - **OpusRAD** (ambientes Win32, VB e Delphi)

2 A linguagem OpusWin

A OpusWin é um dos produtos distribuídos pela Tecnocoop Sistemas para auxiliar na área de aplicativos com interfaces gráficas. Atualmente, são oferecidos os seguintes produtos:

- OpusWin para desenvolvimento de programas Windows
- OpusVis (Visual Opus) para desenvolvimento visual de programas OpusWin
- OPUSRel para desenho e programação de formulários e relatórios

2.1 Características e recursos da OpusWin

Os programas escritos na linguagem OpusWin são compatíveis com os programas desenvolvidos na linguagem OPUS e aderem plenamente aos padrões WIN32, possuindo, em conseqüência, as seguintes características significativas:

- Utilização dos mecanismos de multitarefa e endereçamento linear de 32 bits.
- Utilização plena da interface gráfica Windows e seus recursos, por exemplo, menus, submenus, quadros de diálogo, folhas de propriedades, botões, ícones, imagens, barras de status, barras de ferramentas, barras de tarefas, barras de progressão ... etc ...
- Utilização da janela-console (ou prompt de comandos) do Windows para aplicações com interfaces baseadas em texto, conhecidas como "console applications". Esta característica se torna importante em casos de migração de aplicativos existentes, possibilitando a compatibilidade de ambientes.
- Alto nível de compatibilidade com a linguagem OPUS
- Implementação de um conjunto totalmente novo de funções e comandos que facilitam o desenvolvimento de aplicações Windows. O objetivo deste manual é, exatamente, apresentar todo esse conjunto de funções e comandos que compõem a linguagem OpusWin

Em relação à compatibilidade da OpusWin com a linguagem OPUS, cabe esclarecer alguns pontos de extrema importância. A sua correta compreensão deverá orientar os programadores na tarefa de migrar sistemas da linguagem OPUS (interface baseada, historicamente, em plataformas não gráficas, tipo UNIX e DOS) para a linguagem OpusWin (com interface gráfica, baseada em plataformas Win32). A compreensão e observância dos itens a seguir deverão auxiliar os usuários na tarefa de equacionar e solucionar os problemas geralmente associados a qualquer processo de migração.

Assim sendo, observe atentamente os seguintes tópicos:

- Através da ***OpusWin***, a Tecnocoop Sistemas colocou a disposição de seus clientes (aqueles que utilizam a linguagem OPUS) uma **interface gráfica** padrão **Windows** e uma **estrutura de programação** baseada na utilização da API **WIN32**. Tudo isso, sem perder a facilidade de programação da linguagem **OPUS.**
- Os programas ***OpusWin*** são, efetivamente, programas Windows, utilizando aqueles recursos visuais (janelas, menus, Caixas de Diálogo, controles, barras, botões ... etc ...) que implementam novas interfaces de apresentação e comunicação com o usuário final.
- Os programas OpusWin, pelo fato de serem programas Windows, possuem uma **estrutura** e uma **lógica** específicas**,** possivelmente diferente dos programas desenvolvidos para ambientes não Windows.
- Os programas **OPUS** já existentes podem ser compilados pela ***OpusWin***, sendo, assim, **promovidos** à aplicativos Windows. Esta facilidade, porém, se aplica, principalmente, a programas que **não utilizam** recursos **dependentes** (proprietários) de plataformas específicas, por exemplo, UNIX e MS-DOS. Os programas **OPUS** que **implementam estruturas** lógicas ou **mecanismos** de **programação dependentes** de ambientes operacionais não compatíveis com o Windows, deverão ser migrados com atenção especial.
- Objetivando **facilitar a migração** de programas OPUS para programas OpusWin, comandos e funções da OPUS, que implementam interfaces com o usuário, foram reformulados para serem usados também em aplicativos Windows com interfaces gráficas. Alguns dos comandos modificados são: **@nn,nn SAY ... GET**, **SAVESCREEN( ), RETSCREEN( ), ACHOICE, MEMOEDIT, SET PRINT, PRINT, SET TRACE ON, MESSAGE** ... etc ..
- Em relação ao processo de conversão de programas **OPUS** (interface "texto", ou interface caractere) para ***OpusWin*** (interface gráfica), recomendamos fazer uma **revisão do código fonte** existente, avaliando a necessidade de adaptação ou reprogramação, visando obter os benefícios incorporados pela ***OpusWin***.
- Caso o usuário julgue viável e conveniente manter o mesmo módulo fonte para os ambientes **OPUS** e ***OpusWin*** (nem sempre aconselhável, por razões de eficiência, estrutura e tamanho dos programas, facilidade de manutenção ...etc ...), podem ser utilizados os recursos #INCLUDE, #DEFINE, #IFDEF ou #IFNDEF, controlando assim, em tempo de pre-compilação, o tipo de programa a ser produzido: programa para Windows ou programa para UNIX, por exemplo.

Os recursos de programação incorporados na ***OpusWin*** são implementados, geralmente, em forma de **controles**, **comandos** e **funções**, com seus respectivos **atributos**, **opções**, **parâmetros** e **procedimentos associados**.

2.2 Compilação de programas OpusWin

Os programas fonte a serem processados pela OpusWin possuem os sufixos **.f** ou **.fw**. Este último sufixo permite identificar melhor os programas desenvolvidos para ambientes Win32.

A execução da **OpusWin** gera programas em **C**/**C++** que serão automaticamente compilados e ligados utilizando o compilador e ligador instalados no ambiente do usuário.

A opção **$CPP**, especificada no inicio do programa fonte, informa que deverão ser gerados **programas C++ (.cpp)** que serão ligados utilizando as bibliotecas MFC assim como as bibliotecas **faccpp.lib** (**facaux.lib** na linguagem **C**) e **rotcpp.lib** (**rotaux.lib** na linguagem **C**). Um novo arquivo de include (**rotjan.h**) contém as **classes** utilizadas nos programa C++ gerados pela OpusWin.

Não informando a opção **$CPP**, o programa gerado será gerado, por opção padrão assumida, um programa em **C**.

2.3 Instalação e configuração da OpusWin

A **OpusWin** é distribuída aos clientes em duas modalidades:

- OpusWin Full sem limite de número de usuários
- OpusWin Free mono-usuário

No caso de distribuição da OpusWin Full, a mídia utilizada é CD-ROM ou disquetes 1.44MB. No caso da OpusWin Free pode ser feito um download de nosso site http://www.openbase.com.br.
Somente a primeira modalidade oferece as vantagens de suporte e manutenção por parte da equipe OpenBASE da Tecnocoop Sistemas.

Os procedimentos de instalação e desinstalação, em ambos os tipos de distribuição, são os seguintes:

- Execute o programa SETUP.EXE, do primeiro do subdirectório \DISK1, caso você tenha contratado a OpusWin Full
- Caso você tenha feito download da OpusWin Free, execute o programa (.exe) baixado de nosso site Internet
- Siga as instruções apresentadas pelo programa instalador, até completar, com sucesso, a instalação.
- O programa de instalação atualiza o Registry do Windows 9X e/ou NT, permitindo , desta maneira, estabelecer as configurações adequadas ao funcionamento da família de produtos OpenBASE, da Tecnocoop Sistemas.
- Para desinstalar a OpusWin utilize a opção “adicionar / remover programas” no Painel de Controle do Windows.
- Qualquer dúvida, pode entrar em contato com support@openbase.com.br ou pelo telefone (0XX) 21-505-8138 ou (0XX) 21 505-8138.

3 Instalação de aplicativos usando a OpusWin

A OpusWin oferece algumas funções básicas para facilitar o processo de distribuição e instalação de aplicativos desenvolvidos ou não em OpusWin.

3.1 Função MKDIR

Esta função lógica, cuja sintaxe é **MKDIR (<cad>)**, cria o diretório especificado em **<cad>** e retorna **.t.** se a operação foi realizada com sucesso.

Exemplo

```
? mkdir (“D:\Teste”)
```

3.2 Função CHDIR

Esta função lógica, cuja sintaxe é **CHDIR (<cad>)**, posiciona no diretório especificado em **<cad>** e retorna **.t.** se a operação foi realizada com sucesso.

Exemplo

```
? chdir (“D:\Teste”)
```

3.3 Função LZCOPY

Esta função lógica, cuja sintaxe é **LZCOPY (<orig>,<dest>)**, copia o arquivo especificado em **<orig>** para o arquivo **<dest>**, descomprimindo o arquivo de origem e retornando **.t.** se a operação foi realizadas com sucesso. O arquivo em **<orig>** foi criado através do utilitário COMPRESS.EXE.

Exemplo

? COMPRESS –r abc.da_ A:\abc.da_
? LZCOPY (“A:\abc.da_”,abc.dat”)

4 Gerência de DLLs na OpusWin

Utilizando a linguagem OpusWin, podemos gerar DLLs a serem chamadas a partir de:

- Programas OpusWin, utilizando os comandos WinDLL e RunDLL
- Programas desenvolvidos em outros ambientes, por exemplo, Visual Basic ou Delphi

4.1 Construção de DLLs na OpusWin

Os programas fonte OpusWin que geram DLLs possuem a seguinte estrutura e sintaxe:

```
$dll [=<nome-da-DLL>]
[prog | func | proc]  <nomeprog>  [windll | invwindll]
[parameters parm1 [,parm2, … , parmn]]
 …
[return [<num>]]
…
…
[func | proc]  <nomefunc> [noentry]
[parameters parm1 [,parm2, … , parmn]]
 …
[return [<num>]]
…
…
[func | proc]  <nomeproc> [noentry]
[parameters parm1 [,parm2, … , parmn]]
 …
[return [<num>]]
```

Observações

- O nome da DLL, a ser construída pela OpusWin, é atribuído explicitamente através do comando $DLL, no início do módulo fonte. Não especificando nome no comando $DLL, será criada uma DLL com o nome informado em <nomeprog>
- Uma DLL pode conter apenas um programa principal (“prog”). Os módulos de tipo “Prog” podem receber, opcionalmente, parâmetros cadeia
- As opções WINDLL ou INVWINDLL, associadas ao comando “prog”, especificam a criação de uma janela visível ou invisível, respectivamente.
- Uma DLL pode conter várias procedures (“proc”) e/ou functions (“func”). Cada um destes módulos recebe o nome <nomefunc> ou <nomeproc>
- Os módulos do tipo “Func” e “Proc” podem receber, opcionalmente, parâmetros cadeia
- A cláusula opcional [noentry], associada aos comandos “Proc” e “Func”, informa que esses módulos são Procedures e Functions normais de um programa OpusWin e não Entry-Points da DLL a ser construída

4.2 Utilização das DLLs geradas na OpusWin

As DLLs geradas através de programas OpusWin podem ser utilizadas em todos aqueles ambientes e plataformas que suportem esse tipo de bibliotecas dinâmicas.

4.2.1 Execução de DLLs na OpusWin

As DLLs geradas através de programas OpusWin podem ser executadas em programas OpusWin, por exemplo, através das funções RunDLL e WinDLL.

4.2.1.1 Função RunDLL

A função lógica **RunDLL** permite a execução de **DLLs**, em background, a partir de um programa ***OpusWin***, invocando rotinas dessa **DLL** e passando os parâmetros necessários. RunDLL executa em background, sem abrir janelas de diálogo Windows, chamando a rotina indicada e passando o parâmetro especificado.

A sintaxe correta é a seguinte:
f=RunDLL (<dll>,<rotina>[,<parms>)[,<tam>]]

Onde:

<dll>

Especifica o nome da DLL a ser carregada.

<rotina>

Especifica o nome da rotina (dentro da DLL) a ser executada.

<parms>

Informa, opcionalmente, os parâmetros a serem passados para a rotina da DLL.

<tam>

Se houver informação de parâmetros, deve ser especificado o tamanho dos parâmetros a serem passados para a rotina da DLL.

Exemplos

```
RunDLL ("defcom32.dll", "defcom", "define –f –desq.err esq.e")
RunDLL ("opucom32.dll", "opucom", "opus –f –dp.err p.f")
RunDLL ("dla.dll", "rot1")
RunDLL ("dla.dll", "rot2", varpar,20)
```

4.2.1.2 Função WinDLL

A função lógica Win**DLL** permite a execução de **DLLs**, a partir de um programa ***OpusWin***, invocando rotinas dessa **DLL** e passando os parâmetros necessários. WinDLL comanda a execução da DLL, abrindo uma nova janela de diálogo, chamando a rotina indicada e passando o parâmetro especificado.

A sintaxe correta é a seguinte:

```
f=WinDLL (<dll>,<rotina>[,<parms>)[,<tam>]]
```

Onde:

<dll>

Especifica o nome da DLL a ser carregada.

<rotina>

Especifica o nome da rotina (dentro da DLL) a ser executada.

<parms>

Informa, opcionalmente, os parâmetros a serem passados para a rotina da DLL.

<tam>

Se houver informação de parâmetros, deve ser especificado o tamanho dos parâmetros a serem passados para a rotina da DLL.

Exemplos

```
WinDLL ("defcom32.dll", "defcom", "define –f –desq.err esq.e")
WinDLL ("opucom32.dll", "opucom", "opus –f –dp.err p.f")
WinDLL ("dla.dll", "rot1")
WinDLL ("dla.dll", "rot2", varpar,20)
```

4.2.2 Execução de DLLs em VB

Em Visual BASIC, a declaração das funções de uma DLL apresenta a seguinte sintaxe:

```
private declare function p1 lib "<dir>\nome-da-dll"  (byval a1 as string, ta1 as integer
  [,..., byval an as string, tan as integer]) as integer
[ ...
private declare function pn lib "<dir>\nome-da-dll"  (byval b1 as string, tb1 as integer
  [,..., byval bn as string, tbn as integer]) as integer
]
```

Para cada parâmetro string é passado, na chamada, o seu tamanho máximo.

```
dim  ta1 as integer
ta1 = nn
dim  tan as integer
tan = nn
dim  tb1 as integer
tb1 = nn
dim  tbn as integer
```

```
tbn = nn
dim a1 as string * ta1
...
dim an a string * tan
dim ret1 as integer
ret1=p1(a1,ta1,...,an,tan)
dim b1 as string * tb1
...
dim bn a string * tbn
dim ret2 as integer
ret2=p2(b1,tb1,...,bn,tbn)
```

4.3 Controle de DLLs na OpusWin

Para facilitar as tarefas de desenvolvimento de sistemas e cuidar da melhor performance dos programas, a OpusWin exerce automaticamente alguns controles sobre as DLLs, conforme os seguintes critérios básicos:

- As funções lógicas LoadDLL(<exp>) e FreeDLL(<exp>) carregam e liberam a DLL indicada pela expressão cadeia <exp>.
- Ao final do programa, as DLLs não liberadas explicitamente pelo programa OpusWin, serão liberadas automaticamente.
- As funções RunDLL e WinDLL, ao serem executadas, verificam se a DLL especificada foi previamente carregada. Se não foi efetuada uma carga anterior, no programa, a DLL será carregada a DLL especificada em <exp> e liberada após completada a execução da função. Caso contrário, não será carregada nem liberada a DLL especificada em <exp>

A OpusWin permite efetuar carga de DLLs na memória assim como a liberação de seus recursos através das seguintes funções:

```
LoadDLL (<exp>)
FreeDLL (<exp>)
```

Onde:

<exp> é uma expressão cadeia que especifica o nome da DLL a ser carregada ou liberada.

Exemplo

```
If .not. LoadDLL("abc.dll")
  MessageBox ("Não carregou a DLL abc.dll")
Endif
X=RunDLL("abc.dll","parm1")
If .not. FreeDLL("abc.dll")
  MessageBox ("Não liberou a DLL abc.dll")
Endif
```

4.4 Exemplos utilizando DLLs

Nos exemplos a seguir:

- É apresentado um programa OpusWin que gera a DLL teste1.dll com duas subrotinas (ou entry-points): abre e leseq
- É apresentado um script VB carregando a DLL teste1.dll e invocando as funções abre e leseq

4.4.1 Programa OpusWin

```
$dll=teste1.dll
static prim(l)
func abre
parameters ok
database EXEMPLO 1 a 2
if dberr()=0
  ok="s"
else
  ok="n"
endif
prim=.f.
return 0

func leseq
parameters nome, idadp
database EXEMPLO 1 a 2
use PESSOA
```

```
if .not. prim
  prim=.t.
  locate
else
  continue
endif
if  eof()
  nome=""
  return 1
endif
nome=NOME.P
idadp=str(IDADE,3)
return 0
```

4.4.2 Script Visual Basic:

```
Private Declare Function abre Lib "\win\tsgbd\facil\opuwin\teste1.dll" (ByVal ok As String, ByVal tok As Integer)
As Integer
Private Declare Function leseq Lib "\win\tsgbd\facil\opuwin\teste1.dll" (ByVal nome As String, ByVal tnome As
Integer, ByVal idade As String, ByVal tidade As Integer) As Integer

Private Sub Command1_Click()
Dim ok As String * 3
Dim nome As String * 10
Dim idad As String * 3
Dim r As Integer
r = abre(ok, 3)
if  ok = "n
  msgbox …..
  exitsub
EndIf
r = leseq(nome, 10, idad, 3)
i = 0
text1.Text = ""
While r = 0 And i < 10
  text1.Text = text1.Text + "  " + nome + "  " + idad
  r = leseq(nome, 10, idad, 3)
  i = i + 1
Wend
End Sub
```

5 Gerência do ambiente de trabalho na OpusWin

Apresentamos, neste capítulo, alguns comandos e funções utilizados para gerenciar o ambiente ou área de trabalho. Após breve referência a cada um deles, apresentaremos um programa que inclui exemplos desses comandos e funções.

5.1 Controle das janelas do ambiente

O “desktop”, como seu nome indica, nada mais é do que uma área que representa o ambiente e as ferramentas de trabalho do usuário. Nela são colocados alguns objetos, tais como ícones, painéis, pastas e atalhos que permitem identificar, acessar e gerenciar os componentes do sistemas operacional, assim como disponibilizar os recursos para desenvolvimento dos aplicativos específicos de cada usuário.

As janelas constituem, basicamente, a maneira com a qual o usuário visualiza e interage com os procedimentos e as informações. Por isso, é extremamente importante que cada janela seja planejada, desenhada e implementada de maneira consistente.

As operações básicas com janelas incluem: ativar, desativar, abrir, fechar, mover, redimensionar, maximizar, minimizar, esconder, mostrar, rolar e particionar. As janelas fornecem acesso a diferentes tipos de informação e, de acordo com sua utilização, são classificadas em dois tipos: janela *principal e* janela *secundária.*

Uma janela *principal* envolve a interação com objetos através de atividades como visualização e edição. Adicionalmente, em cada janela principal, podem ser incluídas múltiplas janelas *secundárias*, de forma *a* permitir que o usuário especifique parâmetros e opções, ou seja, detalhes mais específicos em relação aos objetos e procedimentos incluídos na janela *principal*.

5.1.1 Janela principal

Uma janela *principal* consiste, usualmente, de uma moldura, que define sua extensão, um menu de controle, uma janela (ou barra) de status, barras de rolamento (quando necessário) e um título que identifica o que está sendo apresentado na janela.

Geralmente, cada janela *principal* requer um conjunto de janelas *secundárias* para complementar as atividades do usuário na janela *principal*. As janelas *secundárias* são similares às *principais* mas diferem em seu aspecto, comportamento e utilização. Exemplos de janelas secundárias são: Folhas de propriedades (*property sheets), Caixas de diálogo (Dialog boxes), paletas (palette Windows), Caixas de mensagens (message boxes)* e muitos outros.

5.1.1.1 Dimensionando a janela principal

A janela principal de um programa OpusWin possui, como dimensão inicial default, 24 linhas por 80 colunas. Estes valores podem ser modificados através de opções OpusWin, tais como **$WINLINES=<num>** e **$WINCOLS<num>**, especificadas no início do programa.

A opção **$MAXIMIZE** indica que a tela deve ser maximizada, sendo calculado automaticamente o número máximo de linhas e colunas, dependendo da resolução configurada pelo usuário.

As funções numéricas **MAXLINES()** e **MAXCOLUMNS()** retornam o número máximo de linhas e colunas na janela principal, dependendo da opção $MAXIMIZE e da resolução da tela.

Exemplo

```
$maximize
prog
? maxlines()            && retorna 38 em resolução 800x600
? maxcolumns()          && retorna 100 em resolução 800x600
```

5.1.1.2 Ocultando a janela principal

A OpusWin utiliza as opções **$NOSCREEN** e **$NOWINDOW** para determinar que a janela principal do programa não será criada.
A opção **$NOSCREEN** permite criar programas Windows cuja **janela principal não será mostrada**. Neste caso, o programa será executado em background e terminará sem avisar.
A opção **$NOWINDOW** permite criar programas Windows **sem janela principal**. Neste caso, o programa será executado em background.

Exemplo

```
$noscreen
prog
database EXEMPLO 1 a 2
use PESSOA
locate
do while !eof()
        replace IDADE with IDADE+1
        change
        continue
enddo
```

Observações

Com a opção **$NOSCREEN**, o programa não poderá receber parâmetros na execução. A opção **$NOWINDOW** permite a passagem de parâmetros, para o programa.

5.1.2 Janela secundária

Uma janela *secundária* inclui, usualmente, uma borda e uma barra de título. O usuário pode mover a janela, arrastando sua barra de título. Porém, esta janela, por ser secundária, não possui botões de maximizar e minimizar (pois essas operações não se aplicam a janelas secundárias), nem possui barra ou janela de status. Um botão “Close” pode ser incluído para fechar (“destruir”) a janela. O título da janela não inclui ícone, como vimos na janela primária. Veja na figura a seguir exemplos típicos de janelas *secundárias*.

5.2 Utilização da Barra de Status

Na linguagem ***OpusWin***, não é necessário definir, explicitamente, uma barra de status, pois ela será definida, de forma automática, sempre que for necessário. Isto acontece, por exemplo, quando utilizamos o comando **Message**, o comando GET com a opção **Message**, a opção **Message** associada a itens de menu, a opção **Message** associada a itens de ListBox e ComboBox. O comando **Set Trace On** utiliza a Barra de Status para mostrar o Trace do programa.

5.3 Utilização de Barras de Ferramentas

A OpusWin permite definir e exibir Barras de Ferramentas como um controle de janela com um ou mais botões, sendo que cada botão pode ser associado à execução de procedimentos específicos.

As Barras de Ferramentas são formas alternativas (mais rápidas, diretas e visuais) de se trabalhar de forma similar aos itens de menus.

5.3.1 Definição de Barras de ferramentas

A sintaxe geral utilizada para definir uma Barra de ferramentas é a seguinte:

```
ToolBar
   STYLE <estilo1>
  [BITMAP  <bitmap_rc> <num_rc>]
   BUTTON [<tipo> | <num>] <estilo2> [TIP <texto>] DO <proc>
EndToolBar
```

Onde:

<estilo1>

Indica o **estilo da barra de ferramentas**, podendo ser especificado como uma combinação dos estilos constantes na tabela que segue:

estilos da "ToolBar"	Descrição
TBSTYLE_ALTDRAG	Permite que o usuário troque a posição de um botão dentro da barra de ferramentas, arrastado-o enquanto permanece apertada a tecla ALT. Não especificando este estilo, o usuário deverá pressionar a tecla SHIFT enquanto arrasta o botão. Para permitir a possibilidade de arrastar os botões de uma barra de ferramentas, deve ser especificado o estilo CCS_ADJUSTABLE.
TBSTYLE_TOOLTIPS	Cria o controle "ToolTip" que pode ser usado por um aplicativo para exibir algum texto descritivo associado a cada botão da ToolBar.
TBSTYLE_WRAPABLE	Cria uma ToolBar que poderá se desdobrar em múltiplas linhas de botões, quando não é possível conter todos os botões numa única linha. O desdobramento ocorre nos separadores, se houver.
CCS_TOP	Posiciona a ToolBar no topo da janela
CCS_BOTTOM	Posiciona a ToolBar no rodapé da janela

<bitmap_rc>

Faz referência a um **"bitmap resource**", elaborado pelo usuário, contendo os ícones associados a cada um dos botões da barra de ferramentas. O Windows fornece "bitmap resources" predefinidos, cujos elementos recebem nomes padrão e podem ser utilizados conforme as necessidades.

O parâmetro **<*bitmap_rc*>** se refere a um arquivo do tipo **.bmp**, que deverá ser construído, previamente, pelo usuário. Geralmente as ferramentas IDE, por exemplo, VisualStudio, oferecem ferramentas de edição de ícones, DialogBox e ToolBars, entre outros.

<num_rc>

Especifica o identificador do **bitmap resource** referenciado em <bitmap-rc>. Cada comando **bitmap**, dentro de um mesmo programa ***OpusWin***, deve especificar diferente **<*num_rc*>.**

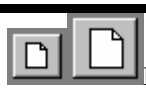
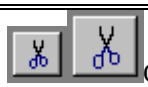
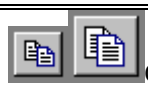
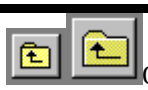
[<tipo> | <num>]

Faz referência, de forma indexada, a cada um dos **elementos de um "bitmap resource**" (parâmetro **<*bitmap_rc*>**), que contém os ícones associados a cada um dos botões. O parâmetro **<*tipo*>** é o nome atribuído a cada elemento do "bitmap resource", podendo ser especificada a palavra **SEPARATOR,** para agrupar e separar os ícones. O parâmetro **<*num*>** especifica um número seqüencial atribuído a cada elemento do "bitmap resource". O Windows fornece alguns "bitmap resources", predefinidos, cujos elementos recebem os seguintes nomes padrão:

STD_COPY	STD_PASTE	STD_CUT
STD_DELETE	STD_FILESAVE	STD_FILENEW

STD_FILEOPEN	STD_HELP	STD_REPLACE
STD_FIND	STD_UNDO	STD_REDOW
STD_PRINT	STD_PROPERTIES	STD_PRINTPRE

Alguns dos ícones do “bitmap resource” padrão do Windows são os seguintes:

---------- *botões 16X16 e 24X24* ----------		
 New	 Open	 Save
 PrintPreview	 Print	 Undo
 Redo	 Cut	 Copy
 Paste	 Delete	 Find
 Replace	 Properties	 Bold
 Italic	 Underline	 Help
 Open folder	 Writing tool	 Eraser tool

<estilo2>

Especifica o **estilo de cada botão** dentro de cada Barra de Ferramentas, e determina a maneira como o botão vai aparecer e como ele vai responder quando acionado pelo usuário, podendo ser a combinação dos seguintes estilos.

Estilo para botões da ToolBar	Descrição
TBSTYLE_BUTTON	Cria um botão, dentro da barra de ferramentas, que se comporta como um botão padrão do tipo “Push Buttons”.
TBSTYLE_CHECK	Cria um botão que alterna seu estado entre pressionado e não pressionado cada vez que o usuário “clica” nele. O botão muda a sua cor de fundo quando fica no estado “pressionado”.
TBSTYLE_CHECKGROUP e TBSTYLE_GROUP	Constrói grupos de botões em uma ToolBar, que funcionam de maneira similar aos controles do tipo “Check Buttons” numa caixa de diálogo. Cada botão acionado permanece no estado “pressionado” até que um outro botão do grupo seja acionado.
TBSTATE_ENABLED	Cria um botão que aceita ser acionado pelo usuário, ou seja, está disponível. Um botão “disabled” não aceita ser acionado e aparece “cinzento”.
TBSTYLE_SEP	Cria separadores que facilitam a visualização dos grupos de botões.

Tip <texto>

Indica que vai ser utilizado o recurso “**Tool Tip**”, ou seja, um texto, a modo de “dica”, associado a um ou mais dos botões da Barra de Ferramentas. A “dica” especificada em <text> será exibida sempre que o cursor permanecer posicionado sobre esse botão durante alguns segundos.

Do <*proc*>

Especifica o procedimento a ser executado quando esse botão for acionado.

5.3.2 Funções de controle das Barras de Ferramentas

As funções, apresentadas na tabela a seguir, permitem identificar se a “procedure” corrente foi chamada através do acionamento de um elemento da barra de ferramentas e, inclusive, qual o botão que foi “clicado” para chamar essa “procedure”:

Função	Descrição
n=Button	devolve, em variável numérica, o número identificador (opção **<num>** do comando Button) do botão "clicado" na barra de ferramentas.
f=Istoolbar	Devolve, em variável lógica, o valor TRUE (".T.") se a "procedure" corrente foi chamada a partir de um botão de barra de ferramentas.

5.3.3 Exemplos de Barra de Ferramentas

Veja a seguir um programa que utiliza os comandos e as funções das Barras de Ferramentas e um BitMap resource criado com os recursos do MS VisualStudio.

Exemplo

```
$nolib
prog
SetWindowText "OpusWin - Exemplo de ToolBar"
SetWindowIcon "world.ico"
ToolBar
  style CCS_TOP TBSTYLE_ALTDRAG TBSTYLE_TOOLTIPS
  bitmap "toolbar.bmp" 1
  Button 1 TBSTYLE_BUTTON Tip "New File ..." do proc1
  Button 2 TBSTYLE_BUTTON Tip "Open File ..." do proc1
  Button SEPARATOR TBSTYLE_SEP
  Button 4 TBSTYLE_BUTTON Tip "ShowText ..." do proc1
  Buttons 5 TBSTYLE_BUTTON Tip "Progress Bar..." do proc1
  Buttons SEPARATOR TBSTYLE_SEP
  Buttons 7 TBSTYLE_BUTTON Tip "Exit ..." do proc1
EndToolBar

proc proc1
if istoolbar() = .t.
  opt=button()
endif
varc="Exemplo de ToolBar ..."
do case
  case opt=1
    c=MessageBox(varc,"New File ...","O","I")
  case opt=2
    c=MessageBox(varc,"Open File...","O","I")
  case opt=4
    Showtext (02,10)
    texto='Isto e uma linha de texto exibida pelo comando "ShowText"'
    Showtext (02,10,texto,"Arial",1,"B/G")
  case opt=5
    c=MessageBox(varc,"Progress Bar...","O","I")
  case opt=7
    quit
endcase
return
```

5.4 Utilizando Barras de Progressão

A Barra Progressiva ou Barra de Progressão ("Progress Bar") é uma janela utilizada pelo aplicativo para visualizar e acompanhar a progressão de operações demoradas.

Na OpusWin, o recurso Progress Bar pode ser utilizado também como controle dentro de uma Janela de diálogo. Veja o Capítulo sobre Dialog Box para saber como construir e utilizar controles Progress Bar numa Janela de diálogo.

A Barra de Progressão consiste de um retângulo que é gradualmente preenchido, da esquerda para a direita, conforme a operação vai sendo executada..

5.4.1 Construção de Barras de Progressão

A tabela a seguir mostra os comandos utilizados para definir e ativar uma Barra de Progressão. Após esta tabela de referência, apresentaremos um programa de exemplo.

Comandos	Descrição dos comandos	Sintaxe dos comandos
PBCREATE	Inicia a definição de uma Barra Progressiva, atribuindo-lhe o nome **<*nome*>.** A Barra de	**PBCREATE <*nome*> <*xi,yi,xl,yl*> [*CHARS* |**

Comandos	Descrição dos comandos	Sintaxe dos comandos
	Progresso será posicionada nas coordenadas e com os tamanhos especificadas em **<*xi,yi,xl,yl*>**. Especificando a opção ***CHARS***, as coordenadas serão medidas em caracteres. Caso contrario, será utilizada como unidade de medida o padrão ***UNITS***, ou seja **unidades lógicas**.	***UNITS*]**
PBMESSAGE	Envia para a Progress Bar **<*nome*>** o comando **<*cmd*>**, com os parâmetros **<*par1*> ... <*parn*>**.	**PBMESSAGE <*nome*>** **<*cmd*> <*par1*> ... <*parn*>**
SETSTEP	Especifica o incremento (”step increment”) da Progress bar, ou seja, a quantidade (degrau) pela qual a Progress Bar vai incrementar sua posição corrente sempre que recebe o comando **STEPIT.** Por default, o **SETSTEP será 0 (zero)**.	**PBMESSAGE <*nome*>** **SETSTEP <*degrau*>**
SETRANGE	Estabelece os valores mínimo e máximo para uma Progress Bar. Por default, o mínimo é zero e o máximo é 100. O limite superior (**<max>**) pode ser inicializado ou “resetado” pelos comandos InitControl e InitControlId.	**PBMESSAGE <*nome*>** **SETRANGE <*min*> <*max*>**
STEPIT	Faz avançar a posição corrente da Barra de Progressão conforme o “step increment” previamente estabelecido. A Progress Bar é redesenhada para refletir a nova posição. Se a posição atingida exceder o valor máximo, a mesma é “resetada” de modo que o indicador de progressão é iniciado de novo.	**PBMESSAGE <*nome*>** **STEPIT**
SETPOS	Posiciona o indicador de progressão no valor especificado em <***npos***>. A Barra Progressiva é redesenhada para refletir essa nova posição.	**PBMESSAGE <*nome*>** **SETPOS <*npos*>**
DESTROY	desativa a janela que contém a Barra de Progressão cujo nome é **<nome>**	**PBMESSAGE <*nome*>** **DESTROY**

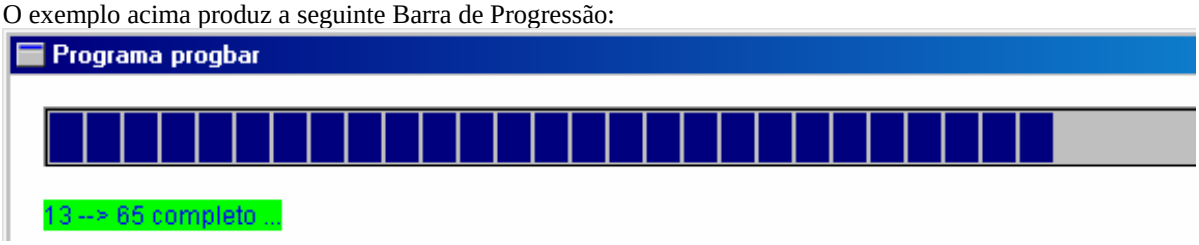
5.4.2 Exemplos de Barras de Progressão

Veja, a seguir, um exemplo de utilização de **Progress Bar**.

```
$nolib
prog
pbcreate ProgressBar 02,01,76,2     && chars
pbmessage ProgressBar "SETRANGE" 1 20
pbmessage ProgressBar "SETSTEP" 1
for i=1 to 19
  pbmessage ProgressBar "STEPIT"
  sleep 1
  temp=i*100/20
  texto=str(i,2)+" --> "+str(temp,2)+" completo ..."
  Showtext (02,04,texto,"arial",1,"B/G")
next
pbmessage ProgressBar "DESTROY"
return
```

Observação

O exemplo acima produz a seguinte Barra de Progressão:



5.5 Utilizando Histogramas

O comando **Histogram** constrói um outro tipo de Barra Progressiva, chamada **Gauge**, com o mesmo propósito e funcionamento da "Progress Bar", porém, com aparência diferente, pois preenche de modo mais contínuo e preciso a área que indica a progressão da operação.

5.5.1 Construção de Histogramas

O comando **Histogram** é incluído, geralmente, dentro de uma **DialogBox** ou uma **Property Page,** especificando em **UNITS** as unidades de medida referentes às coordenadas e os tamanhos da janela do Histograma.

Sintaxe

Histogram <lim> <inc> <*xi,yi,xl,yl*>

Onde:

<lim>

Especifica o limite máximo para o Histograma. Este limite pode ser inicializado ou "resetado" através dos comandos **InitControl** ou **InitControlId.**

<inc>

Estabelece um valor de incremento para o Histograma, ou seja, a maneira como o Histograma vai progredir a partir de sua posição corrente até atingir o valor especificado em **<lim>**.

<xi,yi,xl,yl>

Informa as coordenadas e tamanhos da janela (retângulo) do Histograma, especificadas em unidades de diálogo, ou seja **Units**.

O limite <lim> pode ser modificado de maneira dinâmica durante a execução de um procedimento, conforme apresentamos no exemplo a seguir.

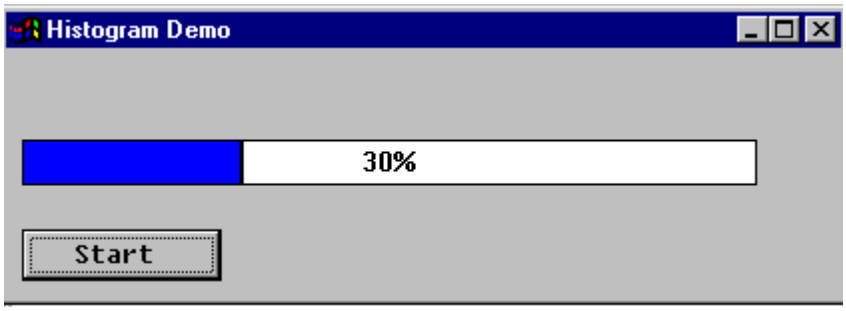
5.5.2 Exemplos de Histograma

Veja, a seguir, um exemplo de utilização de Histograma. Procure adaptá-lo a situações reais, por exemplo, um processo de cópia de arquivos, a instalação dos seus melhores softwares ... etc ...

```
$nolib
prog
b=1
lim = 50
pro = 0
func p1(l)
Dialog  08, 08, 209, 66 units
STYLE WS_OVERLAPPEDWINDOW
CAPTION "Histogram Demo"
   Histogram     lim pro 04,24,184,12
   DefPushButton "Start" b 04,48,50,14 valid p1(lim, pro)
EndDialog
return
func p1
parameters lim(n), pro(n)
maxlim=300
InitControl (1,maxlim)
for pro = 0 to maxlim step 10
  sleep 1
  RefreshControl(1)
next
return .t.
```

Observação

O exemplo acima produz o seguinte Histograma:



5.6 Comandos IDOK e Close Dialog

Os comandos **IDOK e Close Dialog** são utilizados, geralmente, dentro das funções de validação acionadas pelos controles de uma Janela de diálogo. Esses comandos terminam o Diálogo, confirmando as alterações efetuadas nos campos dessa Janela assim como nas variáveis externas ao Diálogo.

As suas funções são similares às da opção **IDOK** associada a um controle dentro de uma Janela de diálogo.

Sintaxe

```
IDOK
CLOSE DIALOG
```

Exemplo

```
Prog
Func f1(l)
Dialog 1,140,20
  EditText NULL var 1,1,10,2 valid f1(var)
EndDialog

Func f1
Parameters a
If a =  “fim”
  IDOK              && ou Close Dialog
  Return .f.
Endif
Return .t.
```

5.7 Comandos IDCANCEL e Cancel Dialog

Os comandos **IDCANCEL e Cancel Dialog** são utilizados, geralmente, dentro das funções de validação acionadas pelos controles de uma Janela de diálogo. Esses comandos cancelam o Diálogo, sem efetuar as alterações nos campos dessa Janela nem nas variáveis externas ao Diálogo.

As suas funções são similares às da opção **IDCANCEL** associada a um controle dentro de uma Janela de diálogo.

Sintaxe

```
IDCANCEL
CANCEL DIALOG
```

Exemplo

```
Prog
Func f1(l)
Dialog 1,140,29
  EditText NULL var 1,1,10,2 valid f1(var)
EndDialog

Func f1
Parameters a
If a =  “can”
  IDCANCEL          && ou Cancel Dialog
  Return .f.
Endif
Return .t.
```

5.8 Comando SetCursor

O comando **SetCursor** permite alterar o formato do cursor dentro da janela principal ou dentro de uma janela de diálogo.

Sintaxe

SetCursor (<*expc*>)

Onde:

<*expc*> é uma cadeia de caracteres tais como: "WAIT", “HAND” e "ARROW", conforme a seguir:

- A expressão "WAIT" indica que o cursor vai ser apresentado em forma da **ampulheta estilo Windows** e pode ser utilizado quando o programa vai executar processamentos demorados, convidando o usuário a esperar pacientemente, sem imaginar que o computador está "travado"
- A expressão "ARROW" faz com o cursor se apresenta na forma de **seta indicadora**
- A expressão "HAND" faz com o cursor se apresenta na forma de **mão com dedo indicador apontado**

Exemplo

```
SetCursor ("WAIT")
ShowText (22,08,”Preparando o resultado. Por favor aguarde ...”,”FixedSys”,1,”N/W”)
Pbcreate um 20,10,40,1 chars
...
...
```

5.9 Comando DispatchMessage

O comando **DispatchMessage** indica que a fila de mensagens do Windows deve ser esvaziada antes da execução de um novo comando. Surte efeito sobre os comandos OpusWin previamente codificados e pode ser utilizado para sincronizar o processamento dentro de um programa.

Sintaxe

DispatchMessage

Exemplo

```
ShowText (22,08,”Preparando o resultado. Por favor aguarde ...”,”FixedSys”,1,”N/W”)
DispatchMessage
Pbcreate um 20,10,40,1 chars
...
...
```

5.10 Comando SetWindowText

O comando **SetWindowText** permite definir o título da janela principal, a ser exibido na linha de título da mesma.

Sintaxe

SetWindowText <*expc*>

Onde:

<*expc*> é uma expressão cadeia.

5.11 Comando ShowText

O comando ShowText permite apresentar textos numa janela ou enviar um texto para impressão.

Sintaxe

```
[SET PRINT ON]
ShowText (<xi>,<yi>,<text>,<fonte>,<tam>,<cor>)
ShowText (<xi>,<yi>)
```

Onde:

<xi>

É uma expressão numérica indicando a coluna (0 a 79) onde posicionar o texto na tela. Se o valor da expressão for negativo, assume-se que a unidade de medida não é colunas, mas **pixels**.
Observe que, na impressora, 1 caracter (coluna) é igual a 36 pixels e, na tela, 1 caracter (coluna) é igual a 9 pixels.

<yi>

É uma expressão numérica indicando a linha (0 a 23) onde posicionar o texto na tela. Se o valor da expressão for negativo, assume-se que a unidade de medida não é linhas, mas **pixels**.
Observe que, na impressora, 1 caracter (linha) é igual a 65 pixels e, na tela, 1 caracter (linha) é igual a 15 pixels.

<text>

É uma expressão cadeia informando o texto a ser exibido na tela.

<fonte>

Especifica a fonte a ser utilizada, por exemplo, “Arial”, “Courier New”, ...

<tam>

É uma expressão numérica indicando o tamanho dos caracteres do texto a ser exibido, sendo que o tamanho da fonte é especificado em múltiplos do tamanho base. Se o valor da expressão for negativo, assume-se que a unidade de medida não é caracteres, mas **pixels**.
Observe que, na impressora, 1 caracter (linha) é igual a 65 pixels e, na tela, 1 caracter (linha) é igual a 15 pixels.

<cor>

Especifica as cores das **letras** e do **fundo** a serem utilizadas. A sintaxe do operando **<cor>** segue o padrão da OPUS. Recomendamos consultar os Manuais da OPUS, especialmente o comando **Set Color**, onde se encontra orientação para auxiliar na codificação correta do operando ***<cor>*.**

Exemplo

```
ShowText (02,10,”Texto texto texto ...”,”Arial”,3,”B/R”)
```

Para apagar um texto da tela, basta especificar, apenas, os operandos <xi> e <yi>. Por exemplo, o comando a seguir apaga o texto exibido em (02,10):

```
ShowText (02,10)
```

5.12 Comando @<xi>,<yi> Show

O comando **@<xi>,<yi> Show** pode ser utilizado junto com as opções **SET FONT TO** e **SET COLOR TO** (previamente especificadas) como uma alternativa para o comando ShowText.

Sintaxe

```
SET FONT TO <fonte>,<tam>
SET COLOR TO <cor>
@<xi>,<yi> Show <text>
```

Onde:

<fonte>

Especifica a fonte a ser utilizada, por exemplo, “Arial”, “Courier New”, ...

<tam>

É uma expressão numérica indicando o tamanho dos caracteres do texto a ser exibido, sendo que o tamanho da fonte é especificado em múltiplos do tamanho base. Se o valor da expressão for negativo, assume-se que a unidade de medida não é caracteres, mas **pixels**.

<cor>

Especifica as cores das **letras** e do **fundo** a serem utilizadas. A sintaxe do operando **<cor>** segue o padrão da OPUS. Recomendamos consultar os Manuais da OPUS, especialmente o comando **Set Color**, onde se encontra orientação para auxiliar na codificação correta do operando ***<cor>*.**

@<xi>,<yi>

São as coordenadas para posicionar o texto na tela. Se os valores da expressão forem negativos, assume-se que a unidade de medida não é colunas (chars), mas **pixels**.

<text>

É uma expressão cadeia informando o texto a ser exibido na tela.

Exemplo

```
Set font to "Courier",2
Set color to "n/w"
@10,10 show "alooooo mamãe …"
```

5.13 Controle de imagens e ícones

Os programas OpusWin permitem exibir imagens e icons assim como obter informações para seu controle tais como:

- Os tipos de imagens
- Largura das imagens
- Altura das imagens

5.13.1 Comando ShowImage

O comando **ShowImage** permite exibir uma imagem na janela principal.

Sintaxe

[SET PRINT ON]
ShowImage [(]<*expc*> | memo(<*item*>) [<*xi*>,<*yi*>[,<*xl*>,<*yl*>] [,<proc>[,<tip>]]] [)]

Onde:

<expc>

Informa o arquivo que contém a imagem a ser exibida. O comando ShowImage processa imagens **BMP**, **GIF** e **JPG**. Isto quer dizer que o arquivo informado deve ser do tipo **.BMP**, **.GIF** ou **.JPG**.

<item>

Especifica que a imagem a ser exibida encontra-se no item **memo <item>**.

<xi>,<yi>,<xl>,<yl>

Especifica as coordenadas e tamanhos da imagem a ser exibida na tela. As coordenadas <xl> e <yl> podem ser expressões ou variáveis.

<proc>

Especifica uma Procedure a ser executada quando for "clicada" (com o botão esquerdo do mouse) a imagem representada pelas coordenadas **<xi>,<yi>,<xl>,<yl>**. Quando o cursor se desliza sobre a área dentro do limite das coordenadas acima, ele se transforma num cursor do tipo HAND.

<tip>

Especifica um texto a ser exibido como "ToolTip" quando o cursor se desliza sobre a área dentro do limite das coordenadas da imagem.

Observações

Em relação ao comando **ShowImage**, devemos observar que:

- A especificação das coordenadas é opcional. Não sendo especificadas, a imagem será exibida na posição **0,0** em seu tamanho original
- Especificando apenas as coordenadas iniciais (**<xi>, <yi>**), a imagem será exibida na posição indicada por **<xi, yi>**, em seu tamanho original.
- Especificando as coordenadas (**<xi>, <yi>**) e os tamanhos (**<xl**, **<yl>**), a imagem será exibida na posição **xi, yi**, no tamanho **xl, yl**.
- Se **<*expc*>** for uma cadeia vazia, a área retangular correspondente será limpa.
- Se foi especificado, previamente, **SET PRINT ON**, a imagem vai para a impressora padrão. É possível modificar, dentro do programa, a impressora padrão, através do comando **SET PRINTER TO <nome da impressora>.** Fora do programa, podem ser configuradas as impressoras através do diálogo **PrintDlg()**.

5.13.2 Função ImageType()

A função cadeia ImageType() retorna o tipo de uma imagem.

Sintaxe

ImageType (<expc>)
ImageType (<arquivo>,<item>)

Utilização

A expressão cadeia **<expc>** pode ser o nome do arquivo que contém a imagem.
A expressão cadeia **<arquivo>,<item>** se refere a um campo memo onde a imagem está armazenada.
A função ImageType() retorna o tipo (.bmp, .pcx, .gif ou .jpg da imagem indicada por <expc>.

5.13.3 Função ImageWidth()

A função numérica ImageWidth() retorna a largura de uma imagem em pixels.

Sintaxe

ImageWidth (<expn>)
ImageWidth (<arquivo>,<item>)

Utilização

A expressão cadeia **<expc>** pode ser o nome do arquivo que contém a imagem.
A expressão cadeia **<arquivo>,<item>** se refere a um campo memo onde a imagem está armazenada.

Exemplo

```
? ImageWidth ("livros.jpg")
```

5.13.4 Função ImageHeight()

A função numérica ImageHeight() retorna a altura de uma imagem em pixels.

Sintaxe

ImageHeght (<expn>)
ImageHeight (<arquivo>,<item>)

Utilização

A expressão cadeia **<expc>** pode ser o nome do arquivo que contém a imagem.
A expressão cadeia **<arquivo>,<item>** se refere a um **campo memo** onde a imagem está armazenada.

Exemplo

```
Database tesmem 1 a 2
Use arqmem
…
locate …
…
? ImageHeight ("arqmem,C2"))
```

5.13.5 Comando SetWindowIcon

O comando **SetWindowIcon** serve para associar um ícone a uma janela principal, a ser exibido no canto superior esquerdo da barra de título da janela.

Sintaxe

SetWindowIcon <"arq-icon”>
Onde:

<"arq-icon”>

Especifica o arquivo que contém o Ícone a ser utilizado.

Exemplo

```
SetWindowIcon "opus.ico"
```

5.14 Comandos Message e Clear Message

O comando **Message** cria uma “status bar” (ou “status window”), colocando nela a mensagem especificada no operando **<text>**.

Sintaxe

Message <*text*>
Onde:

<text>

É uma expressão cadeia que contém a mensagem a ser exibida na barra de status. O comando **Clear Message** limpa a mensagem ativa na barra de status.

5.15 Comando RUN

O comando **RUN** permite executar programas, ou seja, iniciar processos, a partir de um programa OpusWin. Esses processos podem ser acionados de forma síncrona ou assíncrona.

Sintaxe

Run <expc> [WITH <params>] [<opc>] [NOWAIT] [POS(<xi>,<yi>,<xl>,<yl>)]
Onde:

<expc>

Especifica o nome do programa a ser executado, incluída a referência (absoluta ou relativa) ao percurso onde esse programa reside. Pode ser uma variável ou um literal.

<params>

Informa os parâmetros a serem passados para o programa que está sendo invocado pelo comando RUN.

<opc>

Especifica se a janela da aplicação vai ser mostrada quando a execução do programa for iniciada e de que maneira essa janela vai se apresentar. O operando <opc> poderá conter um dos seguintes valores:

Opção	Descrição da opção

Opção	Descrição da opção
SW_HIDE	Esconde a janela e passa a ativação para outra janela
SW_SHOWNORMAL	Ativa e mostra uma janela. Se a janela for minimizada ou maximizada, o Windows a restaura ao seu tamanho e posição inicial.
SW_NORMAL	Igual a SW_SHOWNORMAL. Opção Default.
SW_SHOWMINIMIZED	Ativa e mostra uma janela e a exibe como um ícone.
SW_SHOWMAXIMIZED	Ativa e mostra uma janela e a mostra maximizada.
SW_MAXIMIZE	Maximiza a janela ativa.
SW_SHOWNOACTIVATE	Mostra uma janela em seu tamanho e posição mais recentes. A janela atualmente ativa permanece ativa.
SW_SHOW	Ativa uma janela e a exibe em sua posição e tamanho correntes.
SW_MINIMIZE	Minimiza a janela especificada e ativa a primeira janela na lista do sistema.
SW_SHOWMINNOACTIVE	Mostra uma janela como um ícone. A janela que está ativa no momento, permanece ativa.
SW_SHOWNA	Mostra uma janela em seu status corrente. A janela atualmente ativa permanece ativa.
SW_SHOWNOACTIVATE	Mostra uma janela em seu tamanho e posição mais recentes. A janela atualmente ativa permanece ativa.

NOWAIT

Especifica que, uma vez iniciado o novo processo (iniciado pelo RUN) o controle é retornado imediatamente ao programa que emitiu o RUN, **sem esperar** (NOWAIT) a conclusão desse novo processo. Por padrão, o comando RUN inicia um processo de maneira síncrona, ou seja, o controle retorna ao programa chamador **após terminado** o programa chamado.

POS(<xi>,<yi>,<xl>,<yl>)

Especifica as coordenadas da janela a ser exibida pelo programa informado em <expc>.

5.16 Função ShellExecute

A função lógica **ShellExecute()** permite executar uma determinada ação sobre um arquivo. As ações atualmente implementadas na OpusWin são: **open** e **print.**

Sintaxe

f=ShellExecute (<expc1>, <expc2>, <expc3>, <expc4>, <show>)

Onde:

<expc1>

Indica a operação a ser executada sobre a arquivo. O código da operação pose ser precedido pela palavra wait: para especificar que o controle será retornado ao programa que invocou o “open” ou “print” quando for completada essa operação. Desta forma, é possível sincronizar as tarefas dentro de um programa.

<expc2>

Especifica o nome do arquivo a ser processado.

<expc3>

Especifica, opcionalmente, os parâmetros a serem passados para a operação a ser realizada.

<expc4>

Especifica , opcionalmente, o diretório default do arquivo a ser processado.

<show>

Especifica a maneira como vai aparecer a janela de execução. Veja tabela de opções apresentada no comando **RUN.**

Exemplo

```
xxx_name = ".\html-pb\slidview\index.htm"
f = ShellExecute ("wait:open",xxx_name,"","",SW_NORMAL)
xxx_name = ".\docum\OpenBASE World.avi"
f = ShellExecute ("open",xxx_name,"/play /close","",SW_NORMAL)
```

5.17 Comandos de animação

Os comandos e funções a seguir servem para apresentar e controlar animações dentro e fora das Janelas de Diálogo de um aplicativo OpusWin.

5.17.1 Comando PlaySound

O comando **PlaySound** serve para executar um arquivo de som, do tipo **.WAV.**

Sintaxe

PlaySound (<varc>)

Onde:

<varc>

Especifica o nome do arquivo que contém o som.

Exemplo

```
PlaySound (“hello.wav”)
```

5.17.2 Comando Animation

O comando **Animation** serve para apresentar animações, executando arquivos do tipo **.AVI.** Este comando é implementado como um controle dentro de uma Caixa de diálogo, onde o início e término de uma animação são controlados através do comando **InitControl**, geralmente codificado numa função que foi acionada através da cláusula **valid**. Veja exemplo a seguir.

Sintaxe

Animation {NULL | <caption>} <varc2> <xi>,<yi>,<xl>,<yl>
InitControl (<ncontrol>, <times>)

Onde:

{NULL | <caption>}

O conteúdo em <**caption**> (ou literal) é utilizado como título ("caption") do controle Animation.

<varc2>

Especifica o nome do **arquivo .AVI** que contém a animação a ser apresentada.

<xi>,<yi>,<xl>,<yl>

Informa as coordenadas para posicionar e dimensionar a animação a ser apresentada.

<ncontrol>

Especifica o número de controle **Animation** dentro da Caixa de diálogo.

<times>

Informa o número de vezes a ser executada a animação. O algarismo 0 (zero) indica o término da animação e o símbolo –1 (número um negativo) indica um "loop" infinito da animação, até ser acionado algum evento que finalize, como podemos ver no exemplo a seguir.

Exemplo

```
$nolib
prog
but1=1
varc="fogos.avi"
func p1(l)
Dialog  2,2,24,16
  caption "OpusWin Animation"
  style WS_POPUP WS_CAPTION WS_SYSMENU
  Animation NULL varc 0, 0, 24, 12
  DEFPUSHBUTTON "Start" but1 3, 13, 8, 02 valid p1(but1)
  PUSHBUTTON "Stop" NULL 13, 13, 8, 02 valid p1(but1)
EndDialog
end
func p1
parameters bt(n)
if bt=1
  InitControl (1, -1)
else
  InitControl (1, 0)
  quit
endif
```

return .t.

Observação

O exemplo acima produz a seguinte imagem animada:

5.17.3 Comando AniCreate

O comando AniCreate cria uma janela de animação com o nome **<nome>** nas coordenadas especificadas, conforme a seguinte sintaxe:

AniCreate <nome> <xi> <yi> <xl> <yl> [CHARS] <arquivo>

Se for especificado **CHARS**, as coordenadas são em unidades de caracteres. O arquivo **<arquivo>** (do tipo **.avi**) será exibido na janela de animação.

5.17.4 Comando AniMessage

O comando AniMessage envia uma mensagem **<mess>** para a janela de animação **<nome>** conforme a seguinte sintaxe:

AniMessage <nome> <mess>

O parâmetro **<mess>** pode ser:

PLAY	para iniciar a apresentação da animação
STOP	para terminar a apresentação da animação
DESTROY	para remover o controle de animação

5.17.5 Exemplos

AniCreate um 10,20,20,10 "search.avi"
AniMessage um "PLAY"
AniMessage um "STOP"
AniMessage um "DESTROY"

5.18 Gerência de temporizadores (timers)

A OpusWin permite definir temporizadores para controlar a chamada de procedimentos em determinados intervalos de tempo. Os comandos que implementam esta facilidade são: SetTimer e KillTimer.

5.18.1 Comando SetTimer

Serve para definir um temporizador, atribuindo-lhe um **identificador** e estabelecendo o **intervalo** de chamada de uma **procedure**.

Sintaxe

SetTimer (<id>, <interval>, <proc>)

Onde:

<id>

Especifica o **número identificador** do temporizador que está sendo definido.

<interval>

Especifica a **duração do intervalo**, expresso em quantidade de milissegundos.

<proc>

Informa a **procedure a ser invocada** a cada expiração do intervalo.

5.18.2 Comando KillTimer

Serve para destruir, ou seja desfazer, um determinado temporizador, previamente definido pelo comando SetTimer.

Sintaxe

KillTimer (<id>)

Onde:

<id>

Especifica o **número identificador** de um temporizador previamente ativado.

Exemplo

```
$nolib
prog
External p1
public i(n)
SetTimer (24,2000,p1)    && chama proc p1 a cada 2 segundos
End
proc p1
public i(n)
++i
@ 10,0 say i pic "99"
if i=5
  KillTimer (24)     && Desfaz o temporizador após 10 segundos
Endif
quit
```

5.19 Funções MousePosX() e MousePosY()

As funções numéricas MousePosX e MousePosY retornam a posição corrente do mouse em relação às coordenadas X e Y, respectivamente.

Sintaxe

MousePosX()
MousePosY()

Exemplo

```
? MousePosX()
? MousePosY()
```

5.20 Comando Wait Mouse

Este comando causa a parada do programa em execução até que seja acionado um botão do mouse.

Sintaxe

Wait Mouse

Exemplo

```
A = 1
Wait Mouse  && o programa para esperando click do mouse
B = 2
```

5.21 Comando ShowEan13

O comando **ShowEan13** exibe o código de barras correspondente a um número especificado.

Sintaxe

ShowEan13 (<xi>,<yi>,0,<yl>,<num>,0,<cor>)

Observações

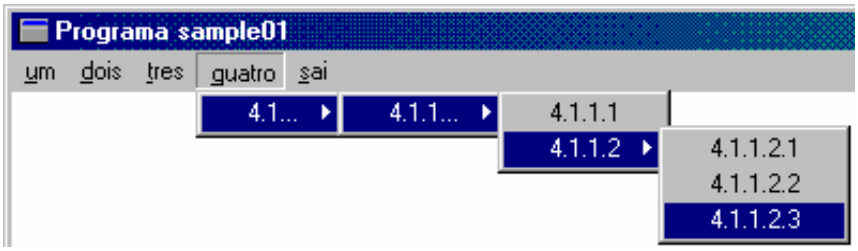
Será exibida em código de barras, na posição <xi>, <yi> com altura <yl> e cor <cor>, a representação do número <num>.
Se o valor da expressão cadeia **<num>** tiver tamanho menor que 12, o número <num> é completado com zeros à esquerda para 12. Se o tamanho for 12, um dígito verificador é calculado e colocado na 13ª posição.

Exemplo

```
ShowEan13 (10,10,0,4,"001234567890",0,"n/w")
```

6 Gerência de menus na OpusWin

Um dos componentes mais comuns da interface Windows é o menu. No padrão Windows, a barra de menu de uma janela aparece logo abaixo da sua barra de título, sendo chamada de “menu principal” ou “menu de mais alto nível”. Os itens de um “menu principal”, geralmente, acionam outros menus, chamados “submenus” ou “menus suspensos”. Usualmente, os itens nos menus e submenus acionam, por sua vez, Caixas de Diálogo que servem para apresentar e/ou obter informações assim como “disparar” procedimentos próprios e específicos de um determinado aplicativo.
A janela principal de um programa OpusWin possui 24 linhas X 80 colunas como dimensão inicial default. As opções de controle $WINLINES=<num> e $WINCOLS=<num> podem alterar essa dimensão default.
A seguir visualizamos um menu típico, incluindo vários níveis de menus secundários.



A OpusWin oferece recursos de linguagem que permitem construir e gerenciar menus, submenus e seus itens. Estas facilidades foram implementadas através de vários comandos, funções e "handles", que servem para definir e controlar menus e submenus, permitindo associar a cada um dos seus itens:

- **procedimentos**, a serem executados quando um item for acionado.
- **mensagens** associadas a cada item, a serem exibidas na barra de status**.**
- Outros submenus

A construção de menus, na OpusWin, será apresentada, a seguir, na seguinte estrutura de tópicos:

- Sintaxe geral utilizada para definir menus e submenus.
- Definição e gerenciamento dinâmico de menus e submenus
- Comandos e funções utilizados para definir menus e submenus
- Exemplos completos de menus e submenus
- Tabela dos comandos para definir menus e submenus.
- Tabelas das funções para controlar menus e submenus.

6.1 Sintaxe geral da definição de menus

Na linguagem OpusWin, podemos definir menus e submenus de forma estática e de forma dinâmica. A seguir apresentamos os comandos e as funções disponíveis na OpusWin para desenvolver aplicativos que utilizem menus e submenus.

A sintaxe global para definir menus, de forma estática, é a seguinte:

```
MENU <nome-menu>
  PAD <nome-pad> prompt <expc> [ popup <pop> | do <proc>] [message <expc>]
    BAR <num> prompt < expc > [ popup <pop> | do <proc>] [message < expc >]
      BAR1 <num> prompt < expc > [ popup <pop> | do <proc>] [message < expc >]
        BARnn <num> prompt < expc > [ popup <pop> | do <proc>] [message < expc >]
EndMenu
Activate Menu <nome-menu>
```

A sintaxe global para definir menus, de forma dinâmica, é a seguinte:

```
hm1=MakeMenu (<nome-menu>)
    hpad=MakePad (<hm1>, <nome-pad>, <nome-prompt>, <msg-pad>, <nome-proc>)
    [if hpad != 0 ... ]
    hpop1=MakePopup (<hm1>, <nome-item>, <nome-prompt>, <msg>, <nome-popup>)
       rbar=MakeBar (<hpop1>, <num-bar>, <nome-prompt>, <msg>, <nome-proc>)
       rbar=MakeBar (<hpop1>, <num-bar>, <nome-prompt>, <msg>, <nome-proc>)
       rbar=MakeBar (<hpop1>, <num-bar>, <nome-prompt>, <msg>, <nome-proc>)
    hpad=MakePad (<hm1>, <nome-pad>, <nome-prompt>, <msg>, <nome-proc>)
    hpop2=MakePopup (<hm1>, <nome-item>, <nome-prompt>, <msg>, <nome-popup>)
       rbar=MakeBar (<hpop2>, <num-bar>, <nome-prompt>, <msg>, <nome-proc>)
       rbar=MakeBar (<hpop2>, <num-bar>, <nome-prompt>, <msg>, <nome-proc>)
Activate Menu <nome-menu>
```

6.2 Definição estática de menus e submenus

Apresentamos, a seguir, os comandos utilizados para definir menus, acompanhados de explicações e exemplos específicos.
A definição e ativação de menus e submenus é uma tarefa muito fácil, na linguagem OpusWin. Muitos dos comandos, opções e parâmetros são opcionais, devendo ser utilizados apenas quando for necessário. Ao final, apresentaremos uma tabela de referência e alguns exemplos completos de menus e submenus.

6.2.1 Comando MENU

O comando MENU serve para:

- iniciar a definição de um menu principal
- identificar um menu principal, atribuindo-lhe um nome

O nome atribuído ao menu principal pode ser obtido e verificado através da função menu() e deve ser especificado nos comandos ACTIVATE e DEACTIVATE, que veremos mais adiante.

Exemplo

```
MENU Util1
```

Observações

No exemplo acima, **Util1** é o nome atribuído ao menu principal, cuja definição está sendo iniciada.

6.2.2 Comando PAD

O comando PAD serve para:

- definir os itens de um menu principal
- atribuir nomes aos itens de um menu principal
- associar rótulos, títulos, conhecidos como "captions" na linguagem visual a cada um dos itens do menu, a serem exibidos na barra de menu correspondente
- associar procedimentos a cada um dos itens do menu, a serem executados quando esses itens forem selecionados pelo mouse ou por teclas de atalho. Apenas os itens elementares de um menu podem ter procedimentos a eles associados. Ou seja, os itens que se desdobram em submenus não podem ter procedimentos associados.
- especificar mensagens associadas a cada um dos itens do menu, a serem exibidas na barra de status quando o cursor deslizar sobre o "rótulo" do item do menu.
- especificar se um item do menu é um item elementar ou se, quando acionado, vai abrir um outro menu (submenu ou menu suspenso). A opção popup <pop> é mutuamente exclusiva com a opção do <proc>.

Exemplo

```
PAD U1 prompt "&Tools" popup U1p1
? pad()  +  "  " + ? prompt()  +  "  " +  ? popup()
```

Observações

No exemplo acima, estamos definindo um item de menu principal. O nome do item é **U1**, seu rótulo é **&Tools** e, quando acionado (pelo mouse ou através de ALT-T), vai ser aberto um menu secundário, ou suspenso, chamado **U1p1.**
Podemos utilizar o recurso de teclas aceleradoras, ou atalhos, para acionar um item de menu, bastando incluir o símbolo **&** imediatamente antes do caracter escolhido. Por exemplo, no exemplo acima, escolhemos a letra T para acionar o item **U1** através do atalho ALT-T.
O nome atribuído a um item do menu principal pode ser obtido e verificado através da função pad(). O rótulo associado a um item de menu pode ser obtido e processado através da função prompt() e o nome de um menu secundário pode ser obtido e processado pela função popup().
Após a execução da segunda linha do exemplo acima, dentro do contexto do exemplo que estamos começando a codificar, será mostrado na tela a seguinte seqüência de caracteres: U1 &Tools U1p1.

6.2.3 Comando BAR

O comando **BAR** serve para:

- definir os itens de um menu secundário (ou suspenso)
- atribuir números de identificação aos itens de um menu secundário

Os operandos e parâmetros do comando BAR são idênticos aos do comando PAD, com a seguinte diferença: O comando **PAD** define itens de menu principal e o comando **BAR** define itens de menus secundários ou suspensos.
Cada item de menu, definido através de um comando **BAR,** pode abrigar, por sua vez, um outro menu suspenso, ou submenu, identificado pelo operando **popup** <*pop*> e definido pelo comando **BAR1,** subordinado de forma hierárquica ao comando **BAR.**

Exemplo

```
MENU Util1
  PAD U1 prompt "&Tools" popup U1p1
    BAR 01 prompt "&Utilities..." popup popU101 message "OpenBASE Utilities"
      BAR1 01 prompt "WinADIC";
        message "Adiciona registros aos arquivos de um Banco de Dados";
        do prupU1
```

Observações

No exemplo acima, através do comando **BAR**, estamos definindo o **item 01 (BAR 01)** do menu secundário conhecido pelo nome de **U1p1**. Quando este item for acionado, via mouse ou via CTRL-U, será aberto o menu suspenso (ou secundário) cujo nome é **popU101.** Se o mouse for apenas deslizado sobre o rótulo "&Utilities", será exibida, na janela (ou barra) de status a mensagem: "**OpenBASE Utilities**".
Não existe procedimento associado ao item 01 do menu secundário **U1p1**, pois esse mesmo item constitui, por sua vez, um outro menu secundário, de nome **popU101**, cujos itens serão especificados através do comando **BAR1**, conforme veremos a seguir.

6.2.4 Comando BARnn

O comando **BARnn** serve para:

- definir os itens de um menu secundário (ou suspenso)
- atribuir números de identificação aos itens de um menu secundário

Os operandos e parâmetros do comando **BARnn** são idênticos aos do comando **BAR.**
Cada item de menu, definido através dos comandos **BARnn**, pode abrigar, por sua vez, um outro menu suspenso, ou submenu, identificado pelo parâmetro **<*pop*>** do operando **popup**, sendo que os itens deste novo menu suspenso são definidos através de comandos **BARnn+1**.
Cada comando **BARnn** pode ter outros comandos **BARnn+1** a ele subordinados, de forma hierárquica, até o máximo de 32 níveis, como vemos no exemplo a seguir.

Exemplo

```
BAR 01 prompt "&Utilities..." popup popU101 message "OpenBASE Utilities"
    BAR1 01 prompt "WinADIC";
      message "Adiciona registros aos arquivos de um Banco de Dados";
      do prupU1
    BAR1 02 prompt "WinCODI";
      message "Apresenta informações sobre a licença do OpenBASE";
      do prupU1
```

Observações

No exemplo acima, através do comando BAR1, estamos definindo os itens 01 (BAR1 01) 02 (BAR1 02) do menu secundário conhecido pelo nome de popU101. Quando estes itens forem acionados, pressionando o mouse, será acionada a rotina prupU1. Não podem ser "passados" parâmetros (nem faz sentido neste contexto) para o procedimento especificado em <proc>.
Se o mouse for deslizado (sem "Clicar"), sobre o rótulo "WinADIC", será exibida, na janela (ou barra) de status a mensagem: "Adiciona registros aos arquivos de um Banco de Dados". A barra de status vai ser gerada, automaticamente, pela OpusWin, sem necessidade de uma definição específica por parte do usuário.
Os itens 01 e 02 do menu secundário **popU101** são itens elementares de menu, por isso não foi codificada neles a cláusula **popup <pop>**. Ou seja, a opção "**do <proc>"** não pode ser utilizada junto com a opção **"popup <pop>"**.

6.2.5 Comando EndMenu

O comando EndMenu serve para finalizar a definição de um menu principal, todos os menus secundários a ele subordinados e os seus itens.

Exemplo

```
    BAR 06 prompt "Sobre a Linguagem OpusWin..."  do prupU2
EndMenu
```

6.2.6 Comando Activate MENU

O comando **ACTIVATE MENU** serve para **ativar** um menu principal.

Exemplo

```
SetWindowIcon "world.ico"
set border on
Activate MENU Util1
```

6.2.7 Comando Deactivate MENU

O comando **DEACTIVATE MENU** serve para **desativar** um menu principal.

Exemplo

```
Deactivate MENU Util1
quit
```

6.3 Funções para controle de menus

O controle de menus e submenus é uma tarefa muito fácil, na linguagem OpusWin. As funções aqui apresentadas devem ser utilizadas apenas quando for necessário.

6.3.1 Função Menu()

A função **Menu()** serve para identificar qual o nome do menu principal selecionado pelo usuário. Caso existam vários menus primários, o programa, eventualmente, precisa saber qual desses menus está sendo processado no momento e, assim, comandar a execução dos procedimentos pertinentes.
A função **Menu()** retorna, em variável cadeia, o nome do menu principal cujos itens foram selecionados pelo usuário.

Exemplo

```
if Menu() = "Util1"
  do proc_Util1
elseif Menu() = "Util2"
  do proc_Util2
else ...
...
endif
```

6.3.2 Função Popup()

A função **Popup()** serve para identificar qual o nome do menu secundário, ou suspenso, selecionado pelo usuário. Freqüentemente, existem vários menus primários definidos no mesmo programa. Assim sendo, é necessário, eventualmente, saber qual desses menus secundários está sendo processado no momento e, assim, comandar a execução dos procedimentos pertinentes.
A função **Popup()** retorna, em variável cadeia, o nome do menu secundário cujos itens foram selecionados pelo usuário.

Exemplo

```
if  Popup() = "popU101"  .and.  bar()  =  02
  do rot_CODI
endif
```

6.3.3 Função Prompt()

A função **Prompt()** serve para obter o "rótulo" ou "caption" dos itens de um menu (principal ou secundário) de modo que o programa possa reconhecer e processar corretamente os itens do menu selecionado. A função **Prompt()** retorna, em variável cadeia, o conteúdo que foi especificado no comando **prompt,** quando da definição dos itens de um menu.

Exemplo

```
if Menu() = "Util1" .and.  Popup() = "popU101"  .and.  bar()  =  02
  ? prompt()
endif
```

Observações

O exemplo acima deverá mostrar o "rótulo" ("caption") associado ao **item 02** do menu secundário **popU101** do menu primário **Util1**.

6.3.4 Função Pad()

A função **Pad()** serve para obter o nome de um item do menu principal selecionado pelo usuário. Esta informação permite que o programa execute os procedimentos correspondentes, conforme a escolha feita pelo usuário. A função **Pad()** retorna uma variável cadeia.

Exemplo

```
if Menu() = "Util1"
  ? Pad()
endif
```

Observações

O exemplo acima deverá mostrar o nome do item selecionado pelo usuário na barra de menu principal.

6.3.5 Função Bar()

A função **Bar()** serve para obter o número identificador do item selecionado, dentro de um menu suspenso. A função **Bar()** retorna uma variável numérica, ou seja o identificador do item selecionado, dentro do menu suspenso referenciado através da função **popup()**.

Exemplo

```
if Menu() = "Util1" .and.  Popup() = "popU101"
  ? str(Bar())
endif
```

Observações

O exemplo acima deverá mostrar o número do item selecionado pelo usuário dentro do menu secundário **popU101**, subordinado ao menu principal **Util1.**

6.3.6 Função IsMenu()

A função **IsMenu()** serve para identificar se o procedimento em execução foi invocado, ou não, através da opção **do <proc>** associada aos comandos **PAD e BAR**, quando da definição do menu. A função lógica **IsMenu()** retorna **.t.** ou **.f.** dependendo se a procedure corrente foi chamada, ou não, através do acionamento de algum item de menu.
A informação obtida através da função **IsMenu()** é de grande utilidade, por exemplo, quando o mesmo procedimento é invocado via menu ou via Barra de Ferramentas, como veremos mais adiante.

Exemplo

```
Proc proc-menu
if IsMenu() = .f.  .or. menu() <> "Util1"
  return
endif
sel = bar()
do case
  case sel = 1
  case sel = 2
endcase
```

Observações

O procedimento **proc-menu** apenas trabalha quando for chamado pelo menu "Util1".

6.4 Definição dinâmica de menus e submenus

A definição dinâmica de menus é de grande utilidade, por exemplo, quando os menus, submenus e itens estão previamente cadastrados em bases de dados As funções numéricas, apresentadas a seguir, permitem definir, de maneira dinâmica, os itens de menus e submenus.

6.4.1 Função MakeMenu

A função numérica MakeMenu serve para definir um menu principal, atribuindo-lhe um nome. Esta função retorna um handle numérico (-1, em caso de erro) a ser utilizado na definição dos itens associados ao menu sendo definido.

Sintaxe

hm1=MakeMenu(<nome-menu>)

Onde:

<nome-menu>

É uma expressão cadeia que informa o nome do menu que está sendo definido.

6.4.2 Função MakePopup

A função numérica **MakePopup** serve para definir um menu suspenso (ou seja, submenu ou popup) abaixo de um menu ou item de menu, indicados estes pelo seu respectivo "handle". Esta função retorna um handle numérico (-1 em caso de erro) a ser utilizado na definição dos itens de menu associados ao Popup definido pela função MakeMenu.

6.4.2.1 Sintaxe

hpop1=MakePopup (<hm1>, <nome-item>, <nome-prompt>, <msg>, <nome-popup>)

Onde:

<hm1>

É uma expressão cadeia que informa o "handle" do menu principal ao qual pertence este menu popup.

<nome-item>

É uma expressão cadeia que informa o nome do item de um menu principal ou secundário.

<nome-prompt>

É uma expressão cadeia que especifica o "prompt", ou seja o “caption” do Popup.

<msg>

É uma expressão cadeia que especifica a mensagem associada ao Popup.

<nome-popup>

É uma expressão cadeia que especifica o nome do Popup.

6.4.3 Funções MakePad e MakeBar

As funções numéricas MakePad e MakeBar servem para definir itens de menu principal (Pad) ou itens de menu suspenso (Bar).

6.4.3.1 Sintaxe

h1=[MakePad | MakeBar] ([<hm1>|<hpop1], <nome-item>, <nome-prompt>, <msg>, <nome-proc>)

Onde:

<hm1> | <hpop1>

É uma expressão cadeia que informa o "handle" do menu principal (<Mhm1>) ou menu suspenso (<hpop1>) aos quais pertence este item de menu.

<nome-item>

É uma expressão cadeia que informa o nome do item de um menu principal ou secundário.

<nome-prompt>

É uma expressão cadeia que informa o "prompt" do Popup.

<msg>

É uma expressão cadeia que informa a mensagem associada ao Popup.

<nome-proc>

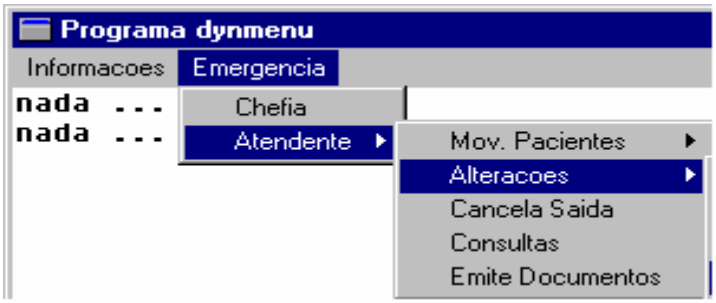
Especifica o nome da Procedure associada ao item de menu que está sendo definido. A Procedure especificada em <nome-proc> deve ser declarada, previamente, através do comando EXTERNAL. Veja o exemplo abaixo.

6.4.4 Exemplo de definição dinâmica de menus

Veja a seguir um exemplo que define dinamicamente um menu e vários submenus.

```
$nolib
prog
external proc01
decl mh1[02]=space(16)
mh1[01]="Informacoes"
mh1[02]="Emergencia"
hm1=MakeMenu("hospub")
 hinf1 = MakePad (hm1,"inf1",mh1[01],"",proc01)
 hinf2 = MakePopup (hm1,"inf2",mh1[02],"","")
  hbar1 = MakeBar (hinf2,"1","Chefia","",proc01)
  hbar3 = MakePopup (hinf2,"1","Atendente","","")
   hbar31 = MakePopup (hbar3,"1","Mov. Pacientes","","")
    hbar311 = MakeBar (hbar31,"1","Procedimentos","",proc01)
   hbar32 = MakePopup (hbar3,"2","Alteracoes","","")
    hbar321 = MakeBar (hbar32,"1","Dados Pessoais","",proc01)
    hbar322 = MakeBar (hbar32,"2","Entrada","",proc01)
    hbar323 = MakeBar (hbar32,"3","Saida","",proc01)
    hbar324 = MakeBar (hbar32,"4","Plano de Saude","",proc01)
    hbar325 = MakeBar (hbar32,"5","Acidentes de Trabalho","",proc01)
   hbar33 = MakeBar (hbar3,"3","Cancela Saida","",proc01)
   hbar34 = MakeBar (hbar3,"4","Consultas","",proc01)
   hbar35 = MakeBar (hbar3,"5","Emite Documentos","",proc01)
Activate Menu hospub
end
proc proc01
? "nada ..."
return
```

O programa acima produz a seguinte tela:



6.5 Gerenciamento dinâmico de menus e submenus

Os comandos apresentados a seguir permitem modificar, de maneira dinâmica, os itens de menus e submenus, habilitando ou desabilitando suas funções.

6.5.1 Comando EnableMenu

O Comando EnableMenu permite habilitar ou desabilitar itens de menus e submenus.

Sintaxe

EnableMenu (<pop>,<num>,<tipo>)

Onde:

<pop>

Especifica o "prompt" (ou seja o "caption", ou título) de um menu ou submenu.

<num>

Identifica o item do menu (especificado em <pop>) a ser habilitado ou desabilitado. O parâmetro <num> informa o número seqüencial do item de menu, a partir de 1, na ordem de definição dos itens desse menu.

<tipo>

Especifica o objetivo do comando EnableMenu, podendo ser uma das seguintes opções:

E[nable] para habilitar
D[isable] ou **G[rayed]** para desabilitar

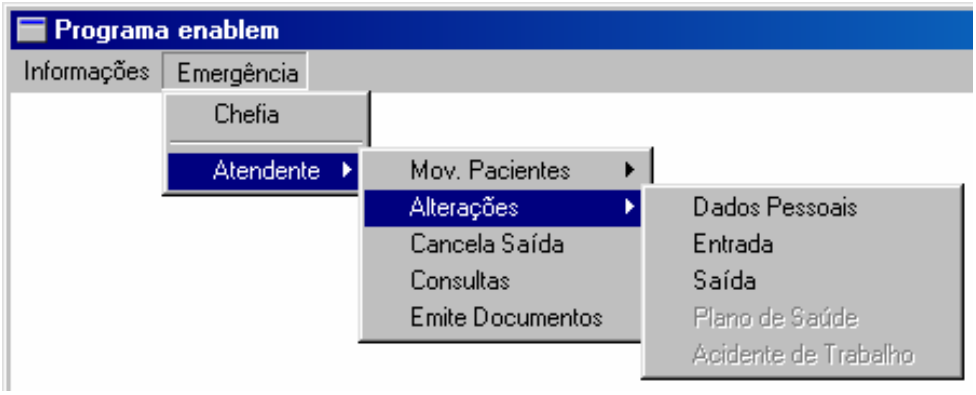
6.5.2 Exemplo do Comando EnableMenu

No exemplo que segue, são desativados (disabled) os itens "Plano de Saúde" e "Acidente de trabalho" do menu rotulado "Alterações".

```
$nolib
prog
Menu hospub
 pad  inf1 prompt "Informações" do rot
 pad  inf2 prompt "Emergência"  popup P2
  bar  1  prompt "Chefia" do rot
  bar  2  separator
  bar  3  prompt "Atendente" popup P2B3
   bar1  1  prompt "Mov. Pacientes" popup P2B3B1
    bar2  4  prompt "Procedimentos" do rot
   bar1  2  prompt "Alterações" popup P2B3B2
    bar2  1  prompt "Dados Pessoais" do rot
    bar2  2  prompt "Entrada" do rot
    bar2  3  prompt "Saída" do rot
    bar2  4  prompt "Plano de Saúde" do rot
    bar2  5  prompt "Acidente de Trabalho" do rot
   bar1  3  prompt "Cancela Saída" do rot
   bar1  4  prompt "Consultas" do rot
   bar1  5  prompt "Emite Documentos" do rot
EndMenu
Activate Menu hospub
proc rot
EnableMenu ("Alterações",4,"G")
EnableMenu ("Alterações",5,"G")
return
```

Observação

Verifique, na tela produzida pelo exemplo anterior, como os itens desabilitados são exibidos num tom meio "acinzentado".



6.5.3 Comando CheckMenu

O Comando CheckMenu permite marcar (check) ou desmarcar (uncheck) itens de menus e submenus.

Sintaxe

CheckMenu (<pop>,<num>,<tipo>)

Onde:

<pop>

Especifica o "prompt" (ou seja o "caption", ou título) de um menu ou submenu.

<num>

Identifica o item do menu (especificado em <pop>) a ser marcado ou desmarcado. O parâmetro <num> informa o número seqüencial do item de menu, a partir de 1, na ordem de definição dos itens desse menu.

<tipo>

Especifica o objetivo do comando CheckMenu, podendo ser uma das seguintes opções:

C[hecked] para marcar
U[nchecked] para desmarcar

6.5.4 Exemplo do Comando CheckMenu

No exemplo que segue, são marcados (Checked) os itens "Plano de Saúde" e "Acidente de trabalho" do menu rotulado "Alterações".

```
$nolib
prog
Menu hospub
  pad  inf1 prompt "Informações" do rot
  pad  inf2 prompt "Emergência"  popup P2
    bar  1  prompt "Chefia" do rot
    bar  2  separator
    bar  3  prompt "Atendente" popup P2B3
      bar1  1  prompt "Mov. Pacientes" popup P2B3B1
        bar2  4  prompt "Procedimentos" do rot
      bar1  2  prompt "Alterações" popup P2B3B2
        bar2  1  prompt "Dados Pessoais" do rot
        bar2  2  prompt "Entrada" do rot
        bar2  3  prompt "Saída" do rot
        bar2  4  prompt "Plano de Saúde" do rot
        bar2  5  prompt "Acidente de Trabalho" do rot
      bar1  3  prompt "Cancela Saída" do rot
      bar1  4  prompt "Consultas" do rot
      bar1  5  prompt "Emite Documentos" do rot
EndMenu
Activate Menu hospub
proc rot
CheckMenu ("Alterações",4,"C")
CheckMenu ("Alterações",5,"C")
return
```

6.6 Tabela para definição de menus e submenus

Veja, na tabela que segue, a relação dos **comandos** utilizados na definição de menus e submenus, assim como seus **operandos** e **opções**. Em seguida, serão apresentadas observações específicas, detalhando cada um dos comandos e apresentando exemplos de utilização.

Comandos e opções	Descrição dos comandos e opções
Menu <***nome***>	Inicia a definição de um menu principal, atribuindo-lhe o nome <***nome***>.
PAD <***nome***> **prompt < expc >** **[popup** <***pop***> **| do** <***proc***>**]** **[message < expc >]**	Define os **itens do menu principal**: • atribuindo-lhe o nome <*nome*>; • associando-lhe o título ("caption") < **expc**>; • informando se esse item, quando acionado, vai abrir um submenu (ou menu suspenso), cujo nome será <*pop*>; • associando o procedimento <*proc*> a ser executado quando o item correspondente for selecionado por "click" do mouse; • especificando a mensagem < **expc**> que será exibida na barra de status quando o cursor deslizar sobre o "caption" do item do menu;
BAR <***num***> **prompt < expc>** **[popup** <***pop***> **| do** <***proc***>**]** **[message < expc >]**	Define os **itens de um menu secundário,** atribuindo-lhes números de identificação. Os operandos do comando **BAR** são idênticos aos do comando **PAD**. Cada item de menu, definido através de um comando **BAR,** pode abrigar, por sua vez, um outro menu suspenso, ou submenu, identificado pelo operando **popup** <*pop*> e definido pelo comando **BAR1,** subordinado de forma hierárquica ao comando **BAR.**
BAR1 <***num***> **prompt < expc >** **[popup** <***pop***> **| do** <***proc***>**]** **[message < expc >]**	Este comando é idêntico ao comando **BAR** e a ele subordinado. Cada item de menu, definido através de um comando **BAR1,** pode abrigar, por sua vez, um outro menu suspenso, ou submenu, identificado pelo operando **popup** <pop> e definido pelo comando **BARnn,** subordinado, de forma hierárquica, ao comando **BAR1**, e assim por diante**.**
BARnn <***num***> **prompt < expc >** **[popup** <***pop***> **| do** <***proc***>**]** **[message < expc >]**	Este comando é idêntico ao comando **BARnn-1** e a ele subordinado. Cada item de menu, definido através deste comando, pode abrigar, por sua vez, um outro menu suspenso, ou submenu, identificado pelo operando **popup** <*pop*> e definido pelo comando **BARnn+1**. Cada comando **BARnn** pode ter outros comandos **BARnn+1** a ele subordinados, de forma hierárquica, até o máximo de 32 níveis**.**
EndMenu	Finaliza a definição de um menu principal.
ACTIVATE menu <***nome***>	Ativa o menu principal identificado pelo operando <***nome***> do comando MENU
DEACTIVATE Menu <***nome***>	Desativa o menu principal identificado pelo operando <***nome***> do comando MENU
ACTIVATE POPUP <***nome***> **<submenu> <botão>** **[<xi>,<yi> | mousepos]**	Ativa um menu ou submenu, nas posições indicadas (em <xi> e <yi> ou mousepos) quando acionado o botão do mouse especificado no operando <botão>.
DEACTIVATE POPUP	Desativa o popup corrente (apenas um popup pode estar ativo).

6.7 Tabela para controle de menus e submenus

As funções, apresentadas na tabela a seguir, permitem identificar os componentes de menus e submenus, de forma que o programa possa conduzir a execução dos procedimentos pertinentes. Estas funções são opcionais, devendo ser utilizadas apenas quando for conveniente.

Função	Descrição da função
c=menu( )	devolve, em variável cadeia, o nome do menu principal que foi selecionado e que está sendo processado.
c=popup( )	devolve, em variável cadeia, o nome do submenu, ou menu suspenso, que foi selecionado e está sendo processado.
c=prompt( )	devolve, em variável cadeia, o título ("caption") de um item selecionado, dentro de um menu

Função	Descrição da função
	principal ou suspenso
c=pad( )	devolve, em variável cadeia, o nome de um item selecionado, dentro de um menu principal
n=bar( )	devolve, em variável numérica, o número identificador do item selecionado dentro de um menu secundário (submenu , ou menu suspenso).
f=ismenu( )	Devolve, em variável lógica, o valor TRUE (“.t.”) se a “procedure” corrente foi chamada a partir de um comando PAD ou BAR (opção **do** <*proc*> na definição de menus e submenus).

6.8 Exemplos de menus e submenus

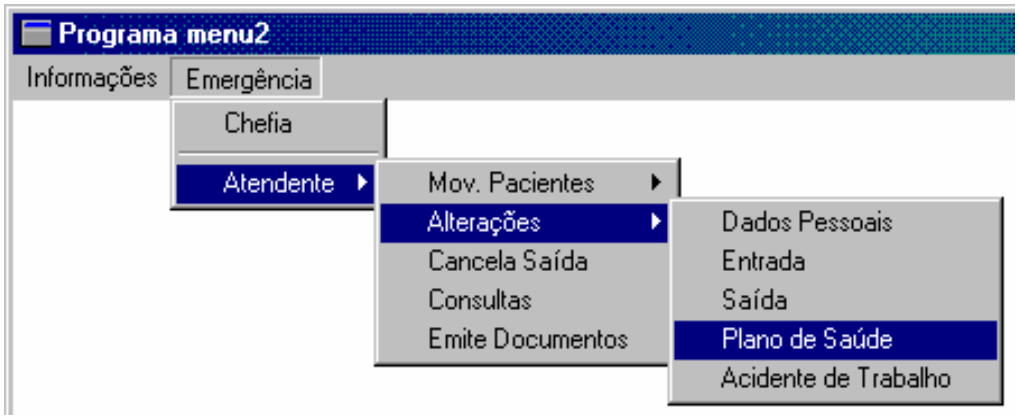
Apresentamos a seguir alguns programas com exemplos de utilização de menus e submenus.

6.8.1 Exemplo 01

Veja, a seguir, um trecho de um programa ***OpusWin*** onde está sendo definido um menu principal e vários menus secundários, ou seja, menus suspensos.

```
$nolib
prog
Menu hospub
  pad  inf1 prompt "Informações" do rot
  pad  inf2 prompt "Emergência"  popup P2
    bar  1  prompt "Chefia" do rot
    bar  2  separator
    bar  3  prompt "Atendente" popup P2B3
      bar1  1  prompt "Mov. Pacientes" popup P2B3B1
        bar2  4  prompt "Procedimentos" do rot
      bar1  2  prompt "Alterações" popup P2B3B2
        bar2  1  prompt "Dados Pessoais" do rot
        bar2  2  prompt "Entrada" do rot
        bar2  3  prompt "Saída" do rot
        bar2  4  prompt "Plano de Saúde" do rot
        bar2  5  prompt "Acidente de Trabalho" do rot
      bar1  3  prompt "Cancela Saída" do rot
      bar1  4  prompt "Consultas" do rot
      bar1  5  prompt "Emite Documentos" do rot
EndMenu
Activate Menu hospub
proc rot
return
```

A tela produzida por este programa pode ser visualizada a seguir:



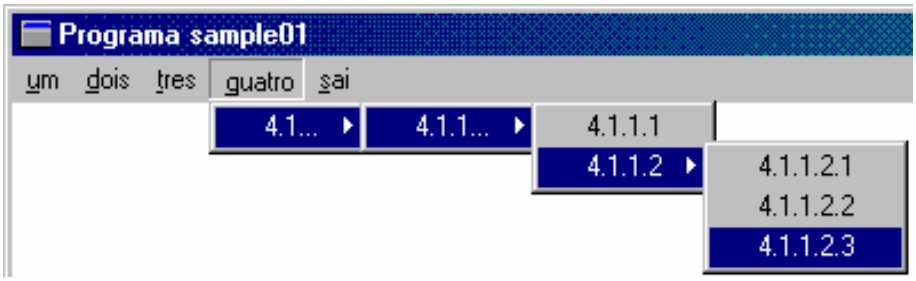
6.8.2 Exemplo 02

Veja, a seguir, um outro programa OpusWin onde está sendo definido um menu principal e vários níveis de menus secundários.

```
$nolib
prog
menu primeiro
  pad um prompt "&um" popup pad-1 message "pad 1"
    bar 11 prompt "1.1..." popup bar-11 message "1.1"
      bar1 111 prompt "1.1.1..." popup bar1-111 message "1.1.1"
        bar2 1111 prompt "1.1.1.1" do proc1 message "1.1.1.1"
        bar2 1112 prompt "1.1.1.2" do proc1 message "1.1.1.2"
```

```
      bar2 1113 prompt "1.1.1.3" do proc1 message "1.1.1.3"
    bar1 112 prompt "1.1.2..." popup bar1-112 message "1.1.2"
      bar2 1121 prompt "1.1.2.1" do proc1 message "1.1.2.1"
      bar2 1122 prompt "1.1.2.2" do proc1 message "1.1.2.2"
      bar2 1123 prompt "1.1.2.3" do proc1 message "1.1.2.3"
  bar 12 prompt "1.2..." popup bar-12 message "1.2"
    bar1 121 prompt "1.2.1..." popup bar1-121 message "1.2.1"
      bar2 1211 prompt "1.2.1.1" do proc1 message "1.2.1.1"
      bar2 1212 prompt "1.2.1.2" do proc1 message "1.2.1.2"
    bar1 122 prompt "1.2.2..." popup bar1-122 message "1.2.2"
      bar2 1221 prompt "1.2.2.1" do proc1 message "1.2.2.1"
      bar2 1222 prompt "1.2.2.2" do proc1 message "1.2.2.2"
pad dois prompt "&dois" popup pad-2 message "pad 2"
  bar 21 prompt "2.1..." popup bar-21 message "2.1"
    bar1 211 prompt "2.1.1..." popup bar1-111 message "1.1.1"
      bar2 2111 prompt "2.1.1.1" do proc1 message "2.1.1.1"
      bar2 2112 prompt "2.1.1.2" do proc1 message "2.1.1.2"
      bar2 2113 prompt "2.1.1.3" do proc1 message "2.1.1.3"
    bar1 212 prompt "2.1.2..." popup bar1-112 message "2.1.2"
      bar2 2121 prompt "2.1.2.1" do proc1 message "2.1.2.1"
      bar2 2122 prompt "2.1.2.2" do proc1 message "2.1.2.2"
      bar2 2123 prompt "2.1.2.3" do proc1 message "2.1.2.3"
  bar 22 prompt "2.2..." popup bar-22 message "2.2"
    bar1 221 prompt "2.2.1..." popup bar1-221 message "2.2.1"
      bar2 2211 prompt "2.2.1.1" do proc1 message "2.2.1.1"
      bar2 2212 prompt "2.2.1.2" do proc1 message "2.2.1.2"
    bar1 222 prompt "2.2.2..." popup bar1-222 message "2.2.2"
      bar2 2221 prompt "2.2.2.1" do proc1 message "2.2.2.1"
      bar2 2222 prompt "2.2.2.2" do proc1 message "2.2.2.2"
pad tres prompt "&tres" popup pad-3 message "pad 3"
  bar 31 prompt "3.1..." popup bar-31 message "3.1"
    bar1 311 prompt "3.1.1..." popup bar1-311 message "3.1.1"
      bar2 3111 prompt "3.1.1.1" do proc1 message "3.1.1.1"
      bar2 3112 prompt "3.1.1.2" do proc1 message "3.1.1.2"
      bar2 3113 prompt "3.1.1.3" do proc1 message "3.1.1.3"
    bar1 311 prompt "3.1.2..." popup bar1-312 message "3.1.2"
      bar2 3121 prompt "3.1.2.1" do proc1 message "3.1.2.1"
      bar2 3122 prompt "3.1.2.2" do proc1 message "3.1.2.2"
      bar2 3123 prompt "3.1.2.3" do proc1 message "3.1.2.3"
  bar 32 prompt "3.2..." popup bar-32 message "3.2"
    bar1 321 prompt "3.2.1..." popup bar1-321 message "3.2.1"
      bar2 3211 prompt "3.2.1.1" do proc1 message "3.2.1.1"
      bar2 3212 prompt "3.2.1.2" do proc1 message "3.2.1.2"
    bar1 322 prompt "3.2.2..." popup bar1-322 message "3.2.2"
      bar2 3221 prompt "3.2.2.1" do proc1 message "3.2.2.1"
      bar2 3222 prompt "3.2.2.2" do proc1 message "3.2.2.2"
pad quatro prompt "&quatro" popup pad-4 message "pad 4"
  bar 41 prompt "4.1..." popup bar-41 message "4.1"
    bar1 411 prompt "4.1.1..." popup bar1-411 message "4.1.1"
      bar2 4111 prompt "4.1.1.1" do proc1 message "4.1.1.1"
      bar2 4111 prompt "4.1.1.2" popup bar2-4111 message "4.1.1.2"
        bar3 41111 prompt "4.1.1.2.1" do proc1 message "4.1.1.2.1"
        bar3 41112 prompt "4.1.1.2.2" do proc1 message "4.1.1.2.2"
        bar3 41112 prompt "4.1.1.2.3" do proc1 message "4.1.1.2.3"
pad sair prompt "&sai" message "saindo ..." do proc1
EndMenu
Activate menu primeiro
proc proc1
if pad()="sair"
  Deactivate menu primeiro
  quit
endif
@16,01 say "Nome do menu="+menu()
@17,01 say "Item do menu principal="+pad()
@18,01 say "Nome do menu suspenso="+popup()
@19,01 say "Nome do item do menu="+prompt()
@20,01 say "Número do item no menu="+str(bar())
return
```

A figura a seguir mostra um detalhe da tela gerada pelo programa acima, onde observamos a barra do menu principal e alguns menus suspensos, ou secundários.



7 Gerência de Caixas de Diálogo na OpusWin

Após os menus e submenus, as Caixas de Diálogo são, provavelmente, os recursos mais utilizados em programas Windows. Caixas de Diálogo ou Quadros de diálogo (“DialogBox”) são janelas temporárias, criadas por uma aplicação Windows para interagir com o usuário, apresentando informações e solicitando dados adicionais, conforme as necessidades e características do aplicativo.

Usualmente, um Quadro de Diálogo inclui um ou mais controles (“controls”), que nada mais são que janelas-filhas, através das quais, por exemplo, o usuário digita um texto, seleciona opções, pressiona um botão ... etc ... conduzindo a execução dos procedimentos de uma determinada aplicação Windows.

A ***OpusWin*** oferece recursos de linguagem que permitem construir e gerenciar variados tipos de Caixas de Diálogo. Estas facilidades foram implementadas, na linguagem OpusWin, através de **comandos** e **funções,** apresentados aqui na seguinte estrutura de tópicos:

- Sintaxe geral das Caixas de Diálogo.
- Tabela dos comandos para definir Caixas de Diálogo.
- Tabela dos comandos para gerenciar dinamicamente as Caixas de Diálogo
- Apresentação dos comandos utilizados para definir e controlar as Caixas de Diálogo
- Exemplos de definição de Caixas de Diálogo

7.1 Sintaxe geral das Caixas de Diálogo

A sintaxe global utilizada na definição de Caixas de Diálogo é o seguinte:

```
DIALOG <xi>,<yi>,<xl>,<yl> [chars | units]
  [caption <título>]
  [style <dialog_style>]
  [help <arqhelp>]
  [font <fonte> [<tam>]
  [start <numc>]
  [verify <func>(<par1>, ... , <parn>)]
  [init <func>(<par1>, ... , <parn>)]
  [cancel <func>(<par1>, ... , <parn>)]]
  CheckBox  <nome> <varl> <xi>,<yi>,<xl>,<yl>  [<comum>]
  EditText  NULL <varc> <xi>,<yi>,<xl>,<yl>  [<comum>]
  RichEdit  NULL <arq> <xi>,<yi>,<xl>,<yl>  [<comum>]
  GroupBox  <nome> <varl> <xi>,<yi>,<xl>,<yl>  [<comum>]
  RadioButton  <nome> NULL <xi>,<yi>,<xl>,<yl>  [<comum>]
  DefPushButton  <nome> | BITMAP (“<bitmap>“) <varn> <xi>,<yi>,<xl>,<yl>  [<comum>]
  PushButton  <nome> | BITMAP (“<bitmap>“) NULL <xi>,<yi>,<xl>,<yl>  [<comum>]
  Ctext  <texto> NULL <xi>,<yi>,<xl>,<yl> [font (<font>,<tam>)] [ID <expc>] [ENABLE | DISABLE]
  Ltext  <texto> NULL <xi>,<yi>,<xl>,<yl> [font (<font>,<tam>)] [ID <expc>] [ENABLE | DISABLE]
  Rtext  <texto> NULL <xi>,<yi>,<xl>,<yl> [font (<font>,<tam>)] [ID <expc>] [ENABLE | DISABLE]
  VarText  NULL <texto>  <xi>,<yi>,<xl>,<yl>  [<comum>]
  ListBox  <vetc> <varn> <xi>,<yi>,<xl>,<yl> [<estilo>]  [<comum>]
  ComboBox  <vetc> <varc> <xi>,<yi>,<xl>,<yl>  [<comum>]
  Image  NULL <varc> <xi>,<yi>,<xl>,<yl> [ID <expc>] [ENABLE | DISABLE]
  Icon  <icon> <num> <xi>,<yi>,0,0
  Histogram <varn1> <varn2> <xi>,<yi>,<xl>,<yl>  [<osbserv>]
  Animation <expc> <var> <xi>,<yi>,<xl>,<yl>  [<observ1>]
  ProgressBar <varn1> <varn2> <xi>,<yi>,<xl>,<yl>  [<observ1>]
  MonthCalendar <expc> <var> <xi>,<yi>,<xl>,<yl>  [<observ1>]
  Grid  NULL | <arq> <num> <xi>,<yi>,<xl>,<yl>  [<observ1>]
  Column  <cab> <vet> | <item>  <xi>,<yi>,<xl>,<yl>
EndDialog
```

Observações importantes

O parâmetro **<comum>** na sintaxe acima engloba as seguintes opções comuns a esse grupo de controles:

[IDOK | CANCEL]
[CONTEXTid(<num>)]
[Message <texto>] | [Message <vetor>]
[tip (<texto)]
[VALID <func> (<par1>, … ,<parn>)]
[WHEN <func> (<par1>, … ,<parn>)]
[PICTURE <cad>]
[Font(<cad>,<num>)]
[Select <func> (<par1>, … ,<parn>)]
[DBLCLK <func> (<par1>, … ,<parn>)]
DISABLE | ENABLE
ID(<iden>
String(<iden>)

O parâmetro **<observ1>** na sintaxe acima indica que deverão ser consultados os detalhes específicos desse controle, que apresentaremos mais adiante.
As opções **Message, Select** e **DblClk** se aplicam apenas aos controles **ListBox** e **ComboBox**.
A opção String se aplica apenas ao controle EditText.
É importante observar que ao utilizarmos objetos do tipo **BITMAP** ou **ICON**, é criado, de forma automática, o arquivo de recursos correspondente, com o sufixo .RC. Este arquivo, que é único para cada programa e suas rotinas, será compilado pelo **resource compiler** (RC.EXE) e "ligado" com o programa principal. Desta forma, quando houver necessidade de definir BitMap ou ICONs em subrotinas, compiladas separadamente, **é necessário utilizar a opção de controle $RC, no inicio do programa principal e de todas as subrotinas que utilizam BitMap e ICONs**, conforme o seguinte exemplo, onde temos o programa Prog1.f e a subrotina Rot1.f:

```
$Library Bibxyz
$RC=prog1.RC
Prog
...
Do Rot1
$Library Bibxyz
$RC=prog1.RC
Proc
...
SetWindow Icon("World.ico")
```

7.2 Tabela de referência para Caixas de Diálogo

Os componentes de uma Caixa de Diálogo recebem, geralmente, o nome de **controles** e podem ser objetos de tipos variados, por exemplo, botões, listas, campos de entrada de dados, imagens, procedimentos ... etc ...
Após esta tabela de referência rápida, apresentaremos, de forma detalhada, cada um dos controles definidos dentro das Caixas de Diálogo.

Comando	Descrição do comando	Sintaxe do comando
Dialog	Inicia a definição de uma Caixa de Diálogo, informando se a posição e o tamanho da mesma estão especificados em caracteres ("**chars**") ou em unidades de diálogo ("**units**").	**DIALOG <xy, yi, xl, yl> [chars | units]**
Caption	especifica o título, ou "rótulo" para o Quadro de Diálogo que está sendo definido.	**CAPTION <texto>**
Style	especifica os estilos da janela da Caixa de Diálogo que está sendo definida.	**STYLE <dialog_style>**
Help	informa o arquivo de Help associado aos controles da Caixa de diálogo, permitindo a utilização do recurso "Help in Context". Consulte os manuais específicos do software utilizado para a elaboração do Help.	**HELP <arqhelp>**
Font	informa o tipo e tamanho da fonte a serem utilizados na Caixa de diálogo. O tipo e o tamanho da fonte escolhida determinam as dimensões da Caixa de Diálogo assim como o número de colunas e linhas.	**FONT <nome> [<tam>]**

Comando	Descrição do comando	Sintaxe do comando
Start	Permite especificar o número seqüencial (a partir de 01) do controle, dentro da Caixa de diálogo, que, inicialmente, irá receber o foco. (Qualquer controle pode ser focalizado, dinamicamente, através do comando **FocusControl**).	**Start <numc>**
Verify	Invoca uma função lógica, passando, como parâmetros, variáveis existentes dentro e fora do Quadro de Diálogo. A função invocada retorna um valor lógico (**.t. ou .f.**). O comando **Verify** está associado a uma Caixa de diálogo como um todo, diferente do comando **Valid,** associado a um controle específico. O comando **Verify** é codificado, geralmente, antes do **EndDialog** e serve para fazer uma validação global de todas as informações da Caixa de Diálogo em questão.	**VERIFY <func>(<parm1>, ... <parmn>)**
Init	Especifica uma função que será invocada na inicialização da Caixa de diálogo, permitindo, assim, agrupar procedimentos a serem executados automaticamente sobre quaisquer dos controles da mesma.	**INIT <func>(<parm1>, ... <parmn>)**
Cancel	Especifica uma função que será invocada quando for acionada a tecla ESC ou quando for fechada a janela do diálogo teclando o ícone X que aparece no canto superior direito.	**CANCEL <func>(<parm1>, ... <parmn>)**
CheckBox	apresenta uma lista de itens, precedidos por um pequeno quadrado ("Check Box"), permitindo "checar", ou seja, **marcar *uma ou mais* opções da lista.**	**CheckBox <nome> <varl> <xi>,<yi>,<xl>,<yl> [<comum>]**
ListBox	Exibe uma lista com um ou mais itens a serem selecionados pelo usuário. A opção **<*vetmsg*>** especifica um **vetor**, cujos elementos são as mensagens associadas aos elementos correspondentes no vetor da ListBox. Essas mensagens serão exibidas na barra de status quando o elemento correspondente na ListBox for selecionado com **um** "click" do mouse. **Observações:** • Um "click" permite acionar a função associada com a opção Select, se existir. • Dois "clicks" invocam a função associada com a opção DblClk, se existir..	**ListBox <vetc> <varn> <xi>,<yi>,<xl>,<yl> [<estilo>] [<comum>]**
EditText	mostra uma área onde o usuário pode codificar uma nova informação ou modificar uma informação existente. Os estilos e as Pictures, associados a este controle, permitem um bom gerenciamento do seu conteúdo.	**EditText NULL <varc> <xi>,<yi>,<xl>,<yl> [<comum>]**
RichEdit	Permite a visualização e impressão de arquivos .TXT e .RTF. O parâmetro <arq> é a variável que contém o nome do arquivo. Os comandos PrintControl ou PrintControlId são utilizados para imprimir	**RichEdit NULL <arq> <xi>,<yi>,<xl>,<yl> [<comum>]**

Comando	Descrição do comando	Sintaxe do comando
	o arquivo visualizado pelo controle RichEdit.	
ComboBox	Através da combinação de **ListBox** e **EditBox**, o controle **ComboBox** apresenta uma lista de itens, sendo permitido selecionar, modificar um dos itens assim como incluir novas informações. A opção **<*vetmsg*>** permite associar uma mensagem para a cada elemento da ComboBox, a ser exibida na linha de status. **Observações:** • Um "click" permite acionar a função associada com a opção Select, se existir. • Dois "clicks" invocam a função associada com a opção DblClk, se existir..	**ComboBox <vetc> <varc> <xi>,<yi>,<xl>,<yl> [<comum>]**
GroupBox	Serve para agrupar vários controles numa DialogBox. A área que contém esse grupo de controles fica cercada por uma borda, sendo mostrado, na parte superior esquerda da GroupBox , o texto especificado em **<nome>.**	**GroupBox <nome> <varl> <xi>,<yi>,<xl>,<yl> [<comum>]**
RadioButton	exibe uma lista de itens, precedidos de pequenos círculos, para permitir **marcar** uma, e **apenas uma**, das opções da lista, que são mutuamente exclusivas. Os "RadioButtons" são agrupados em controles **GroupBox.**	**RadioButton <nome> NULL <xi>,<yi>,<xl>,<yl> [<comum>]**
Ctext	exibe o texto <texto> centralizado no retângulo **<*xi,yi,xl,yl*>**	**CTEXT <*texto*> *NULL* <*xi,yi,xl,yl*> *[ID <expc>] [ENABLE|DISABLE] [CONTEXTid(<num>)]* [font (<font>,<tam>)]**
Ltext	exibe o texto <texto> alinhado à esquerda no retângulo **<*xi,yi,xl,yl*>**	**LTEXT <*texto*> *NULL* <*xi,yi,xl,yl*> *[ID <expc>] [ENABLE|DISABLE] [CONTEXTid(<num>)]* [font (<font>,<tam>)]**
Rtext	exibe o texto <texto> alinhado à direita no retângulo **<*xi,yi,xl,yl*>**	**RTEXT <*texto*> *NULL* <*xi,yi,xl,yl*> *[ID <expc>] [ENABLE|DISABLE] [CONTEXTid(<num>)]* [font (<font>,<tam>)]**
VarText	exibe o conteúdo da variável cadeia <*varc*> no retângulo **<*xi,yi,xl,yl*>.** O gerenciamento da variável <*varc*> é dinâmico, podendo seu conteúdo ser "refrescado" a qualquer momento enquanto estiver ativa sua caixa de diálogo. Esta característica dinâmica constitui a diferença deste comando em relação aos comandos **Ctext, Ltext e Rtext.**	**VarText NULL <varc> <xi>,<yi>,<xl>,<yl> [<comum>]**
Icon	exibe o ícone contido no arquivo <*icon*> na posição indicada pelas coordenadas <*xi,yi*>, que será identificado pelo número <*num*>. Cada comando **icon**, dentro de um mesmo programa OpusWin, deve especificar um **<*num*>** diferente. A largura e altura devem ser especificadas com o valor zero. O controle **<*icon*>** deve ser especificado por último, após os comandos PushButton.	**Icon <icon> <num> <xi>,<yi>,0,0**
Image	exibe a imagem especificada através da variável <varc> na posição indicada pelas	**IMAGE NULL <varc> <*num*> <*xi*>,<*yi*>,<*xl*>,<*yl*>**

Comando	Descrição do comando	Sintaxe do comando
	coordenadas <***xi,yi,xl,yl***>, dentro de uma caixa de diálogo.	***[ID <expc>] [ENABLE|DISABLE]***
DefPushButton	exibe um botão de comando com o status de "pressionado". O usuário pode selecioná-lo pressionando ENTER, mesmo que esse botão não esteja "focalizado".	**DEFPUSHBUTTON** ***<nome> | BITMAP("<bitmap>") <varn><xi,yi,xl,yl> [comum]***
PushButton	exibe um botão de comando a ser pressionado pelo usuário	**PUSHBUTTON** ***<nome> | BITMAP("<bitmap>") NULL <xi,yi,xl,yl> [<comum>]***
Valid	Invoca uma função, passando como parâmetros variáveis existentes dentro e fora do ambiente do Quadro de Diálogo. A função retorna um valor lógico (.t. ou .f.). O comando **Valid** é associado aos controles do tipo **EditText, PushButton, ListBox e ComboBox.** As funções associadas a esses controles, através do comando **valid**, são executadas quando os controles são acionados para processamento.	**VALID <func>(<parm1>, ... <parmn>)**
When	A opção **when <func> (<p1>, ... <pn>),** associada aos controles do tipo **EditText e ListBox**, permite disponibilizar, condicionalmente, um controle do tipo **EditBox**, dependendo do valor retornado pela função lógica **<func>**, que poderá retornar um valor lógico (.t. ou .f.). A função associada com o comando **when** é executada quando o controle correspondente obtém o focus, ou seja, **antes** mesmo dele ser processado.	**When <func>(<parm1>, ... <parmn>)**
DblClk	A opção **DblClk <func> (<p1>, ... <pn>),** associada aos comandos **ListBox** e **ComboBox**, permite executar uma determinada função quando um dos elementos desses controles forem acionados com **duplo click** do mouse. A função lógica **<func>** pode retornar um valor lógico .t. ou .f., permitindo terminar ou não os controles **ListBox** ou **ComboBox** associados.	**DblClk <func> (<p1>, ... ,<pn>)]**
Select	A opção **Select <func> (<p1>, ... <pn>),** associada aos comandos **ListBox** e **ComboBox**, permite executar uma determinada função quando um dos elementos desses controles forem selecionados (um click do mouse). A função lógica **<func>** pode retornar um valor lógico .t. ou .f., permitindo terminar ou não os controles **ListBox** ou **ComboBox** associados.	**Select <func> (<p1>, ... ,<pn>)**
ProgressBar	O controle ProgressBar cria uma Barra de Progressão dentro de uma Janela de diálogo. Na definição do controle ProgressBar informamos: • limite a ser atingido (**<varn1>**) • valor da progressão (**<varn2>**) • As coordenadas do controle Veja mais adiante maiores detalhes deste controle.	**ProgressBar <varn1> <varn2> <xi>,<yi>,<xl>,<yl> [<observ1>]**
MonthCalendar	O controle MonthCalendar permite mostrar	**MonthCalendar <expc> <var>**

Comando	Descrição do comando	Sintaxe do comando
	um calendário, por mês, iniciando pela data informada no parâmetro **<var>**. Veja mais adiante maiores detalhes deste controle.	**<xi>,<yi>,<xl>,<yl> [<observ1>]**
Histogram	O controle Histogram constrói uma Barra Progressiva, chamada Gauge, dentro de uma DialogBox. **<varn1>** especifica o limite máximo para o Histograma. Este limite pode ser inicializado ou "resetado" através dos comandos InitControl ou InitControlId. **<varn2>** estabelece um valor de incremento para o Histograma, ou seja, a maneira como o Histograma vai progredir a partir de sua posição corrente até atingir o valor especificado. **<xi,yi,xl,yl>** Informa as coordenadas e tamanhos da janela (retângulo) do Histograma, especificadas em unidades de diálogo, ou seja Units. Veja mais adiante maiores detalhes deste controle.	**Histogram <varn1> <varn2> <xi>,<yi>,<xl>,<yl> [<osbserv>]**
Animation	O controle **Animation** serve para apresentar animações, executando arquivos do tipo **.AVI**, dentro de uma Caixa de diálogo, onde o início e término de uma animação são controlados através do comando **InitControl**, geralmente codificado numa função que foi acionada através da cláusula **valid**. Veja exemplo a seguir. **<expc>** **<expc>** é utilizado como título ("caption") do controle Animation. **<var>** Especifica o nome do **arquivo .AVI** que contém a animação a ser apresentada. **<xi>,<yi>,<xl>,<yl>** Informa as coordenadas para posicionar e dimensionar a animação a ser apresentada. Veja mais adiante maiores detalhes deste controle.	**Animation <expc> <var> <xi>,<yi>,<xl>,<yl> [<observ1>]**
Grid	Especifica uma GRID para exibir e controlar: • Elementos de vetores • Arquivos de Bancos OpenBASE • Arquivos externos Veja neste manual maiores detalhes a respeito deste controle.	**Grid NULL <num> <xi>,<yi>,<xl>,<yl> [<observ1>]** **Grid <arq> <num> <xi>,<yi>,<xl>,<yl> [<observ1>]** **Grid EXTERN <num> <xi>,<yi>,<xl>,<yl> [<observ1>]**
Column	Especifica as colunas de uma GRID que irão conter: • elementos de vetores • itens de arquivos OpenBASE • campos de arquivos externos Veja neste manual maiores detalhes a respeito deste controle.	**Column <cabn> <vetn> 0,0,<xl>,<yl> [<observ1]** **Column <cabn> <itemn> 0,0,<xl>,<yl> [<observ1]** **Column <cabn> <campon> 0,0,<xl>,<yl> [<observ1]**
EndDialog	Finaliza a definição de um Quadro de Diálogo	**EndDialog**

7.3 Tabela de comandos para Caixas de Diálogo

A ***OpusWin*** oferece alguns comandos adicionais que permitem estabelecer um tratamento mais dinâmico dos controles especificados dentro de uma DialogBox.
Estes comandos adicionais são executados, geralmente, nas funções invocadas através das cláusulas **Valid**, **When** e **Verify,** , dentro de uma Caixa de diálogo. Veja a seguir a descrição destes comandos e sua opções:

Comando	Descrição do comando	Sintaxe dos comandos
EnableControl **EnableControlId**	Habilita um determinado controle dentro da DialogBox. Os parâmetros **<num>** ou **<id>** identificam o controle: • Utilizando um número seqüencial a partir de 01, na ordem de especificação dentro da Caixa de Diálogo. • Utilizando o nome que foi atribuído ao mesmo através da opção **ID <expc>** dentro da Caixa de Diálogo.	**EnableControl (<num>)** **EnableControlId (<id>)**
DisableControl **DisableControlId**	Desabilita um determinado controle dentro da DialogBox. Os parâmetros **<num>** ou **<id>** identificam o controle: • Utilizando um número seqüencial a partir de 01, na ordem de especificação dentro da Caixa de Diálogo. • Utilizando o nome que foi atribuído ao mesmo através da opção **ID <expc>** dentro da Caixa de Diálogo.	**DisableControl (<num>)** **DisableControlId (<id>)**
MoveControl **MoveControlId**	Posiciona o controle identificado pelo número **<num>** ou pelo nome **<id>**, nas coordenadas especificadas em **<xi>,<yi>,<xl>,<yl>**.	**MoveControl (<num>, <xi>,<yi>,<xl>,<yl>)** **MoveControlId (<id>, <xi>,<yi>,<xl>,<yl>)**
DeleteControl **DeleteControlId** **RefreshContent**	Elimina, da tela, o controle identificado pelo número **<num>** ou pelo nome **<id>**. Neste comando, não é necessário informar as coordenadas do controle em questão. Exibe na tela o conteúdo atualizado de: • Um vetor **<str>** associado a controles do tipo ListBox e ComboBox. • Uma variável **<str>** associada a controles do tipo EditText, CheckBox, GroupBox e Image. O nome da variável **<str>** pode ser passado como parâmetro para a rotina que executa o comando RefreshContent ou a variável **<str>** pode ser pública.	**DeleteControl (<num>)** **DeleteControlId (<id>)** **RefreshContent (<str>)**
RefreshControl **RefreshControlId**	Exibe na tela o conteúdo atualizado do controle com número **<num> (**contado a partir de 01) ou com nome **<id>** (especificado na opção **ID),** dentro de uma DialogBox. Este comando se aplica aos controles **ListBox, EditBox, CheckBox, GroupBox, ComboBox** e **Image**.	**RefreshControl (<num>)** **RefreshControlId (<id>)**
FocusControl **FocusControlId**	Permite colocar o foco num determinado controle, dentro de uma Dialog Box, identificado pelo seu **número** ou pelo seu **nome**. Se o número do controle em **<num>** for negativo, indica que o evento KILLFOCUS, enviado pelo Windows e recebido pelo programa OpusWin, será ignorado para o controle especificado em **<id>**.	**FocusControl (<num>)** **FocusControlId (<id>)**
SelectControl **SelectControlId**	Permite selecionar, ou seja, marcar como "selected" um dos itens dos controles tipo ListBox ou ComboBox. Os parâmetros **<num>** ou **<id>** identificam o controle (ou seja, a ListBox ou	**SelectControl (<num>,<expc>)** **SelectControlId**

Comando	Descrição do comando	Sintaxe dos comandos
	ComboBox), dentro da Caixa de Diálogo. O parâmetro **<expc>** especifica o item a ser marcado como "selected" dentro do controle especificado em **<num>** ou **<id>**.	**(<id>,<expc>)**
GetControl	Esta função numérica retorna o número do controle que acionou uma função **valid**.	**nctrl = GetControl()**
GetControlId	Esta função cadeia retorna o identificador, ou nome, do controle que acionou uma função **valid**.	**cctrl = GetControlId()**
InitControl **InitControlId**	Serve para inicializar o limite máximo de uma Barra de progressão ou Histograma. Esta função recebe como parâmetros: • número (**<num>**) ou nome (**<id>**) do controle ProgressBar ou Histogram • O limite máximo desse controle ProgressBar ou Histogram	**InitControl (<num>,<lim>)** **InitControlId (<id>,<lim>)**
PrintControl **PrintControlId**	Serve para imprimir um arquivo visualizado através do controle RichEdit, especificando: • O número (**<num>**) ou nome (**<id>**) do controle RichEdit • A orientação e tamanho do papel a ser impresso: P (Portrait) L (Landscape) A (A4) L (legal) T (Letter)	**PrintControl (<num>,<par>)** **PrintControlId (<id>|<num>,<orien>)**

7.4 Definição de Caixas de Diálogo

Os comandos da linguagem ***OpusWin***, que permitem iniciar e terminar a definição de uma Caixa de Diálogo são: **Dialog** e **EndDialog**.

7.4.1 Comando DIALOG

O comando DIALOG serve para:

- iniciar a definição de um Quadro de Diálogo
- informar as coordenadas para posicionamento do Quadro de Diálogo
- especificar qual o tipo de métrica utilizada: chars ou units.

Sintaxe

DIALOG <xy, yi, xl, yl> [chars | units]

Onde:

- a expressão **<xi, yi>** especifica as coordenadas que representam a **posição inicial** da Caixa de diálogo.
- a expressão **<xl, yl>** especifica os **tamanhos** da **largura** e da **altura** da Caixa de diálogo.
- A expressão **[Chars** | **units]** informa o tipo de unidade de medida utilizado na especificação das coordenadas e tamanhos da janela da Caixa de Diálogo e de todos os controles nela incluídos.

As coordenadas e os tamanhos são parâmetros numéricos, com valor inicial a partir de zero. Podem ser especificados em "chars" (default da OpusWin) ou em "units" ("Dialog units").
Na falta de especificação, a linguagem OpusWin assume que, tanto as coordenadas que indicam a posição inicial, como os tamanhos de largura e altura das Caixas de Diálogo e dos controles, estão expressos em caracteres (opção chars).
Assim sendo, a sintaxe do comando, quando não informada a unidade de medida, assume-se como:

Dialog <coluna-inicial, linha-inicial, qtde-colunas, qtde-linhas> chars

O padrão chars da OpusWin permite maior compatibilidade com a programação OPUS para ambientes não gráficos e maior facilidade migração a partir de aplicativos com interface caractere.
A unidade de medida default é estabelecida através da opção (explícita ou implícita) [CHARS | UNITS] no comando Dialog, aplicando-se à Caixa de Diálogo como um todo, assim como a cada um dos controles nela incluídos. Porém, é possível especificar coordenadas e/ou tamanhos em unidades de medida diferentes do default estabelecido no comando Dialog, bastando especificar os valores numéricos (referentes a coordenadas e

tamanhos), precedidos do sinal negativo. Esta facilidade permite uma maior precisão e flexibilidade na especificação das unidades de medida e, em conseqüência, uma apresentação visual mais correta.

Exemplo

```
Dialog 5,10,-80,-10
```

Observação

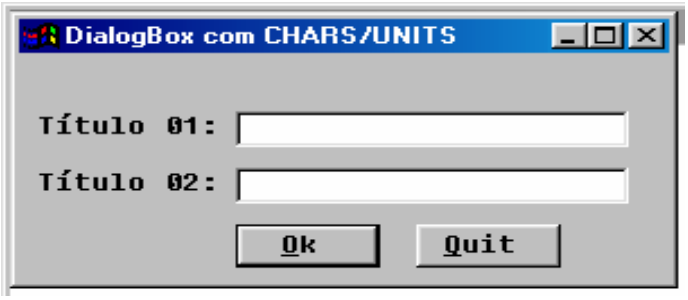
Neste exemplo, utilizamos o padrão OpusWin (**chars**). Assim sendo, a Caixa de Diálogo será posicionada na coluna 5 com linha 10 (a partir de zero), tendo 80 “Dialog units” (mais ou menos 20 colunas) de largura e 10 “Dialog units” de altura. O **tipo** e o **tamanho** da fonte especificados na opção **font** determinam, também, as dimensões reais da Caixa de Diálogo assim como o número de colunas e linhas.

Exemplo

O exemplo a seguir mostra como codificar uma Caixa de Diálogo e seus controles, utilizando a unidade de medida “**chars”**, como default, porém utilizando, também, a unidade de medida “**units”** para especificar as coordenadas e tamanhos. Após o exemplo, explicaremos melhor as unidades de medida relacionadas às Caixas de Diálogo.

```
$nolib
prog
campo1, campo2 = ""
botao = -1
Dialog 5,10,36,08
  caption "DialogBox com CHARS/UNITS"
  style WS_OVERLAPPEDWINDOW
  ltext "Título 01:" NULL 01,02,10,01
  EditText NULL campo1 12,02,-80,01
  ltext "Título 02:" null 01,04,10,01
  EditText NULL campo2 12,04,-80,01
  DefPushButton "&Ok" botao 12,06,08,-12 IDOK
  PushButton "&Quit" NULL 22,06,08,-12 IDOK
EndDialog
return
```

Veja como vai ser apresentado na tela o exemplo acima:



Observação

Quando especificada a opção “**units**”, as coordenadas e tamanhos estão baseados em **unidades de medida lógicas**, não reais. As funções gráficas e de texto do Windows operam sobre unidades lógicas, que são traduzidas em **unidades físicas** (pixels) quando o objeto é efetivamente exibido na tela. Ocorre, então, um mapeamento, executado pelo Windows. O tipo de mapeamento (existem vários) utilizado altera a maneira como as unidas lógicas ("**Dialog units**") são traduzidas em unidades físicas ("**pixels**"), que dependem, por exemplo, das características do monitor utilizado pelo usuário. Desta forma, o programa ***OpusWin*** se torna, de alguma maneira, independente das características físicas dos dispositivos de exibição utilizados, podendo ser utilizado em qualquer ambiente e configuração Windows.
Para compreender um pouco melhor este assunto, podemos estabelecer os seguintes conceitos básicos:

- Uma **unidade básica de diálogo (ubd)** é igual ao tamanho médio de um caractere na fonte padrão do Windows ("Fixed System font").
- Uma **unidade de diálogo (ud)** é igual a **1/4 de ubd**, horizontalmente, e **1/8 ubd**, verticalmente.
- A coordenada **<xi>** e o tamanho da largura **<xl>** são expressos em unidades de 0,25 (1/4) da largura média do caractere
- A coordenada **<yi>** e o tamanho da altura **<yl>** são expressos em unidades de 0,125 (1/8) da altura media de um caractere.

Exemplo

```
Dialog 20,80,160,88 units
```

Observação

No exemplo acima, usamos a opção **units**. Assim, para essa Caixa de Diálogo em particular, o canto superior esquerdo está, mais ou menos, a 5 caracteres da esquerda da área do usuário na janela principal e a 10 caracteres, mais ou menos, do alto. A caixa de diálogo tem 40 colunas de largura e 11 linhas de altura.

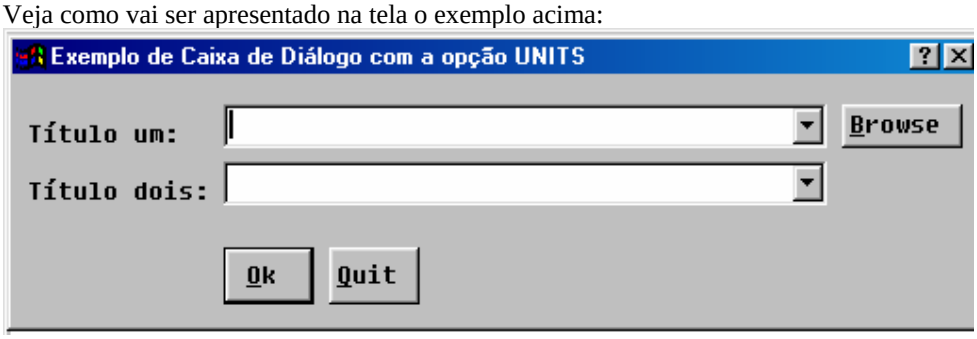
O exemplo a seguir mostra como codificar uma Caixa de Diálogo e seus controles utilizando a opção **units**, especificando as coordenadas e tamanhos em **unidades lógicas**, ou seja, "**unidades de diálogo**".

Exemplo

```
$nolib
prog
decl vetc[6]=space(40)
campo1, campo2 = ""
botao = -1
Dialog 20,40,256,70 units
  caption "Exemplo de Caixa de Diálogo com a opção UNITS"
  style WS_POPUP WS_CAPTION WS_SYSMENU DS_CONTEXTHELP
  ltext "Título um:" NULL 4,12,48,8
  ComboBox vetc campo1 56,08,160,8
  ltext "Título dois:" null 4,28,48,8
  ComboBox vetc campo2 56,24,160,8
  DefPushButton "&Ok" botao 56,48,24,16 IDOK
  PushButton "&Quit" NULL 84,48,24,16 IDOK
  PushButton "&Browse" NULL 220,08,32,12 IDOK
EndDialog
return
```

Observação

Veja como vai ser apresentado na tela o exemplo acima:



7.4.2 Comando EndDialog

O comando EndDialog serve para terminar a definição de um Quadro de Diálogo.

Sintaxe

```
EndDialog
```

7.5 Controles incluídos nas Caixas de Diálogo

A seguir apresentamos os controles a serem incluídos dentro de Caixas de Diálogo, conforme as necessidades dos usuários.

7.5.1 Controle Caption

O controle Caption informa o título da Caixa de Diálogo, que será exibido na linha de título, alinhado à esquerda.

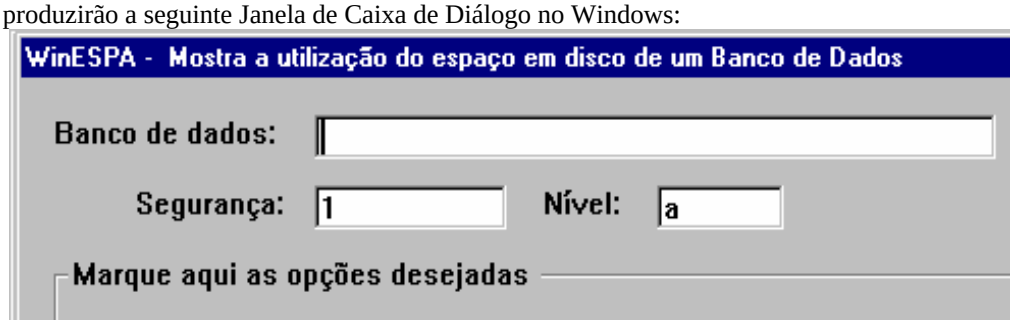
Exemplo

```
segur=1
nivel="a"
Dialog 06,02,73,13
   caption "Mostra a utilização de espaço em disco de um Banco de Dados"
   ltext texto1 NULL 02,01,15,01
   EditText NULL banco 18,01,40,01 context(10)
   ltext texto2 NULL 07,03,10,01
   EditText NULL str(segur) 18,03,10,01
   ltext texto3 NULL 32,03,06,01
   EditText NULL nivel 39,03,06,01
   GroupBox “Marque aqui as opções desejadas”  02,05,70,04
            . . .
```

```
EndDialog
```

Observação

Os comandos do exemplo acima (apenas um pequeno trecho da definição de uma Caixa de Diálogo típica) produzirão a seguinte Janela de Caixa de Diálogo no Windows:



7.5.2 Controle Style

O controle **Style** define os estilos associados à janela do Quadro de Diálogo. Os estilos jogam um papel importante na interface visual do Windows. São associados aos componentes do ambiente gráfico para definir ou modificar seu aspecto, sua aparência e seu comportamento, respondendo aos eventos do aplicativo. A utilização correta dos estilos facilita a programação, pois as funções comuns, usadas freqüentemente em ambiente gráficos e visuais, são implementadas, de alguma forma, através de combinação de estilos.
A **OpusWin** implementa os estilos do Windows através de comandos próprios ou, em alguns casos, através da especificação dos estilos do Windows dentro do programa.
Os estilos especificados no comando **Style** são estilos relacionados às janelas das Caixas de Diálogo. Estes estilos definem características da Caixa de Diálogo como um todo, assim como dos controles a ela subordinados. Por exemplo, se for especificado **Style DS_FIXEDSYS**, significa que vai ser utilizada a fonte **SYSTEM_FIXED_FONT**, tanto para a janela DialogBox como para todos os controles definidos dentro dessa DialogBox.
Os estilos **genéricos** de uma janela DialogBox, a serem especificados no comando **Style**, são apresentados na tabela a seguir. Podem ser especificados como uma combinação de uma, ou mais, das seguintes opções:

Estilo da janela DialogBox	Descrição
WS_SYSMENU	Cria uma janela que exibe o menu de controle na barra de título. O menu de controle é composto pelos botões Minimizar, Maximizar e Close
WS_CAPTION	Cria janela com barra de título
WS_MAXIMIZEBOX	Cria uma janela com o botão “Maximizar”
WS_MINIMIZEBOX	Cria uma janela com o botão “Minimizar”
WS_OVERLAPPEDWINDOW	Cria uma janela com as seguintes características: • possui um título e uma borda • usa estilo WS_SYSMENU • possui uma moldura grossa, que pode ser utilizada como “puxador” para redimensionar a janela utilizando o mouse.
WS_POPUP	Cria uma janela secundária, instantânea.
WS_EX_TOPMOST	Permite que a janela de uma Dialog Box seja exibida em primeiro plano na área de trabalho.
DS_CENTER	Centraliza a Dialog Box dentro da área de trabalho.
DS_CENTERMOUSE	Centraliza o cursor do mouse dentro da Dialog Box
DS_CONTEXTHELP	Inclui um sinal de interrogação no Menu de Controle da caixa de Diálogo, a ser utilizado pelos mecanismos de “Help in context”. O estilo DS_CONTEXTHELP não pode ser utilizado juntamente com WS_MAXIMIZEBOX ou WS_MINIMIZEBOX.
DS_3DLOOK	A janela apresenta aparência 3D
DS_FIXEDSYS	Usa a fonte SYSTEM_FIXED_FONT em vez de SYSTEM_FONT

Exemplo

```
Dialog 06,02,73,13
  caption "Mostra a utilização de espaço em disco de um Banco de Dados"
  style WS_POPUP  WS_CAPTION WS_SYSMENU DS_CENTER
EndDialog
```

7.5.3 Controle Help

O controle **Help** informa o nome do **arquivo de ajuda** (arquivo **.HLP** ou .**CHM**) que contém os tópicos de Help associados aos controles nas Caixa de Diálogo. A elaboração e associação dos tópicos de ajuda é uma tarefa muito importante, dependendo dela a correta compreensão do aplicativo e a eficiência do usuário na execução dos procedimentos. O Windows oferece o recurso de **Help in Context** para permitir solicitar ajudas específicas para cada campo (controle) na Caixa de Diálogo.
A associação dos tópicos de ajuda aos controles de uma Caixa de Diálogo possui duas etapas:

1. Elaboração, identificação e associação dos tópicos de ajuda na fase de desenvolvimento do aplicativo.
2. Acionamento dos tópicos de ajuda, por parte do usuário, na fase de execução do aplicativo.

A primeira etapa acima pode ser desenvolvida com o auxílio de recursos tais como o **Help WorkShop** e o **Html Help WorkShop**. Os tópicos recebem uma identificação para serem associados e acionados pelo aplicativo.
A segunda etapa acima pertence ao usuário final ao executar o aplicativo. Os tópicos de ajuda do Help in Context são acionados através de um dos seguintes procedimentos:

- pressionando o botão direito do mouse sobre o controle em questão e clicando sobre a frase “What’s this?” (no Windows em inglês)
- arrastando o símbolo de interrogação (?) desde o Menu de Controle (se esse símbolo ali estiver presente) até o item cujo Help deseja ser obtido.
- posicionando o cursor no item em questão e acionando a tecla F1.

1.0.0 Controle Font e cláusula Font

O controle **Font** informa o nome e o tamanho da fonte a serem utilizados dentro da Caixa de Diálogo, como um todo. A cláusula **Font** está associada a determinados controles da **DialogBox**.

Sintaxe

font <"fonte"> <tam>

Onde:

<”fonte”>

Especifica o nome completo da fonte a ser utilizada.

<tam>

Especifica o tamanho da fonte referenciada em <”fonte”>, expresso em caracteres ou em “units”. Para informar o tamanho da fonte em “units” (unidades de diálogo) basta especificá-lo em forma negativa.

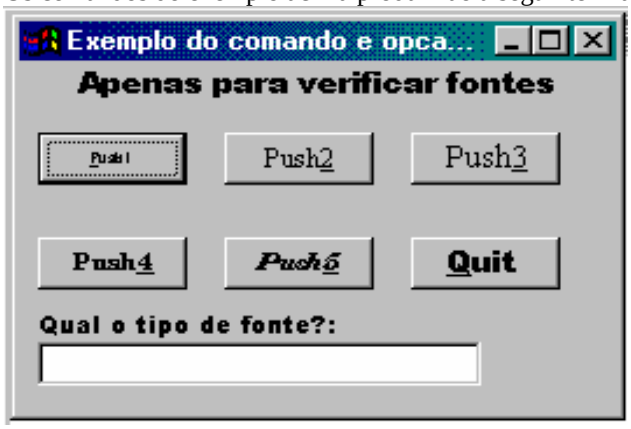
Exemplo

A seguir apresentamos um exemplo utilizando o comando e a cláusula **Font**.

```
$nolib
prog
func fver(l)
bt=-1
varc=""
Dialog 00,00,32,10
 style WS_OVERLAPPEDWINDOW
 font "AlexsHand" 18
 caption 'Exemplo do comando e opção "Font"'
 ctext "Apenas para verificar fontes" NULL 00,00,32,01 font("Arial Black", -10)
 defpushbutton "&Push1" bt 01,02,08,-12  exec fver(bt);
                font ("Times New Roman", -6)
 PushButton "Push&2"   NULL 11,02,08,-12 exec fver(bt);
                font ("Times New Roman", -8)
 PushButton "Push&3"   NULL 21,02,08,-12 exec fver(bt);
                font ("Times New Roman", -10)
 PushButton "Push&4"   NULL 01,05,08,-12 exec fver(bt);
                font ("Elephant", 1)
 PushButton "Push&5"   NULL 11,05,08,-12 exec fver(bt);
                font ("Elephant Italic", 1)
 PushButton "&Quit"    NULL 21,05,08,-12 IDOK;
                font ("Arial Black", -10)
 ltext "Qual o tipo de fonte?:" NULL 01,07,22,01 font("Arial Black", 1)
 EditText NULL varc 01,08,22,01 font("Arial Black", 1)
EndDialog
quit
func  fver
parameters bt(n)
return(.t.)
```

Observação

Os comandos do exemplo acima produzirão a seguinte DialogBox :



1.0.0 Controle Start

Permite especificar o número do controle, dentro da Caixa de diálogo, que, inicialmente, vai receber o foco. Qualquer controle pode ser focalizado, dinamicamente, através do comando **FocusControl.**

Sintaxe

Start <numc>
Onde:

<numc>

Especifica o número seqüencial do controle, iniciando de 1 (um), que receberá, inicialmente o **focus**.

1.0.0 Controle Verify

O controle **Verify,** associado a **Caixas de Diálogo** e **Folhas de Propriedades,** serve para invocar uma função lógica, passando, como parâmetros, variáveis definidas dentro do programa, (associadas, ou não, a **Caixas de Diálogo** e **Folhas de Propriedades**.
O controle **Verify** constitui um recurso de grande utilidade, pois a função lógica, por ele invocada, permite, por exemplo, validar, de forma individual e referencial, as informações fornecidas pelo usuário ou obtidas pelo programa. Se a função acionada pelo comando **Verify** retornar um valor lógico verdade (“return (.t.)”), significa que as informações validadas estão corretas. Neste caso, o Quadro de Diálogo será concluído e os dados podem ser processados. Caso a função lógica invocada pelo comando **Verify** retorne um valor lógico falso (“return(.f.)”), o Quadro de Diálogo permanece aberto e sua janela disponível na tela, permitindo, por exemplo, enviar avisos, do tipo MessageBox ou Message na barra de status, para que o usuário acerte ou complete as informações digitadas no Quadro de Diálogo.
O objetivo da função invocada pelo comando **Verify** é, obviamente, fazer a **verificação** das informações e processá-las.
O comando **Verify,** de forma diferente da cláusula **Valid**, não é associado aos controles das Caixas de Diálogo, devendo ser codificado como se fosse um dos controles da DialogBox.

Exemplo utilizando Verify numa DialogBox:

O exemplo que segue utiliza o comando **Verify** dentro de uma Caixa de Diálogo.

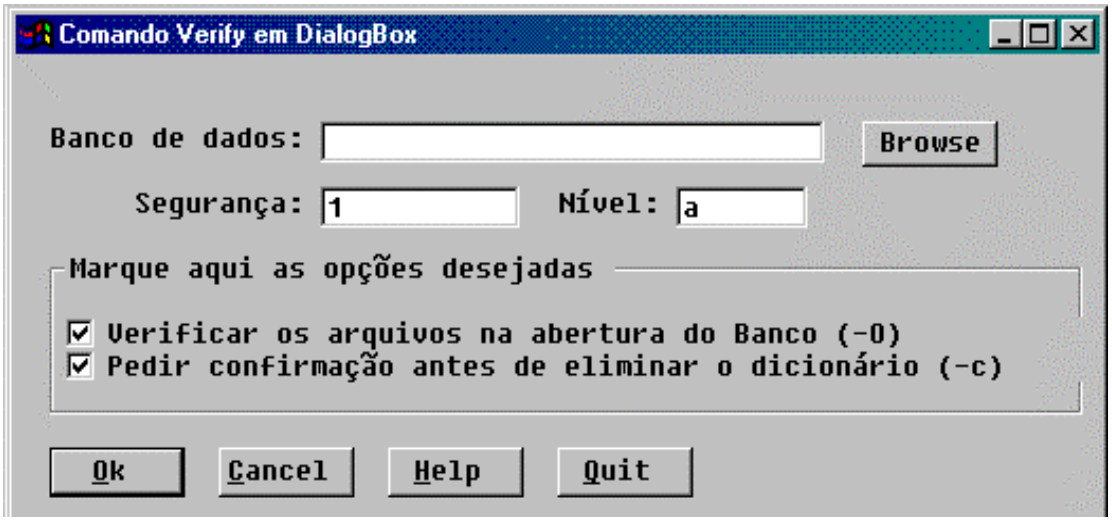
```
$nolib
prog
func vfy_dele(l)
banco,opc = ""
segur=1
nivel="a"
botao = -1
gbx1 = 1
texto1="Banco de dados:"
texto2="Segurança:"
texto3="Nível:"
tx1gb1="Marque aqui as opções desejadas"
tx1cb1="Verificar os arquivos na abertura do Banco (-O)"
tx1cb2="Pedir confirmação antes de eliminar o dicionário (-c)"
vl1, vl2 = .t.

Dialog 00,01,64,14
   caption "Comando Verify em DialogBox"
   style WS_OVERLAPPEDWINDOW
   Ltext texto1 NULL 02,02,15,01
```

```
    EditText NULL banco 18,02,28,01 ES_AUTOHSCROLL
    Ltext texto2 NULL 07,04,10,01
    EditText NULL str(segur) 18,04,10,01
    Ltext texto3 NULL 32,04,06,01
    EditText NULL nivel 39,04,06,01
    GroupBox tx1gb1 gbx1 02,06,61,05
      CheckBox tx1cb1 vl1 03,08,60,01
      CheckBox tx1cb2 vl2 03,09,60,01
      DefPushButton "&Ok" botao 02,12,08,-12 IDOK
      PushButton "&Cancel" NULL 12,12,08,-12 IDCANCEL
      PushButton "&Help" NULL 22,12,08,-12;
        valid vfy_dele (botao, banco, str(segur), nivel, vl1, vl2)
      PushButton "&Quit" NULL 32,12,08,-12 IDCANCEL
      PushButton "Browse" NULL 50,02,08,-11;
        valid vfy_dele (botao, banco, str(segur), nivel, vl1, vl2)
      Verify vfy_dele (-1, banco, str(segur), nivel, vl1, vl2)
  EndDialog
quit
func vfy_dele
parameters botao(n), banco, segur, nivel, v1(l), v2(l)
do case
  case botao=1 && ok
  case botao=3 && Help
    cmd_run="winhlp32.exe  ..\help\util.hlp"
    run cmd_run
    return .t.
  case botao=5 && Procura
    banco=GetOpenFileName("*.*","","Bancos de Dados/*.*")
    return .t.
endcase
banco = alltrim (banco)
if empty(banco)
  Message "Favor informar o Banco de Dados a ser carregado ..."
  return .f.
endif
segur = alltrim(segur)
if  len(segur) = 0
  segur = "1"
endif
nivel = alltrim(nivel)
if  len(nivel) = 0
  nivel = "a"
endif
opc = ""
opc = opc + " -b"+banco + " -s"+segur + " -n"+nivel
if v1 = .t.
  opc = opc + " " + "-c"
endif
if v2 = .f.
  opc = opc + " " + "-O"
endif
mesegur="windele " + opc
ret=MessageBox (mesegur,"Para testes ...","O","I")
return .t.
```

Observação

O programa acima produz, entre outras, a seguinte tela:



Exemplo utilizando Verify numa Property Sheet

O exemplo que segue utiliza o comando **Verify** no contexto de Property Sheets.

```
$nolib
prog
func fverify(l)
decl vetc[06]=space(10)
vetc[01]="Manga"
vetc[02]="Abacaxi"
vetc[03]="Laranja"
vetc[04]="Limão"
vetc[05]="Banana"
vetc[06]="Outros"
bt=1
varc="Outros"
varn = 6
propsheet
 caption "Exemplo de Property Sheets"
 font "Arial Narrow"
 Verify fverify (varc, varn)
 Start 2
 proppage 05,05,28,12
  caption "ComboBox"
  font "Arial Narrow"
  style WS_OVERLAPPEDWINDOW
  Ltext "Selecione da lista:" NULL 02,01,20,01
  ComboBox vetc varc 10,03,16,06 WS_VSCROLL
 endproppage
 proppage 05,05,28,12
  caption "ListBox"
  font "Arial Narrow"
  style WS_OVERLAPPEDWINDOW
  Ltext "Selecione da lista:" NULL 02,01,20,01
  ListBox vetc varn 10,03,16,06 WS_VSCROLL IDOK
 endproppage
 proppage 05,05,28,12
  caption "EditBox"
  font "Arial Narrow"
  style WS_OVERLAPPEDWINDOW
  Ltext "Escreva algo:" NULL 02,01,20,01
  EditText NULL varc 02,03,24,08;
   ES_MULTILINE ES_AUTOHSCROLL ES_AUTOVSCROLL ES_WANTRETURN
 endproppage
endpropsheet
if PropButton() = 0    && cancel
 return
endif
end
func fverify
parameters varc, varn(n)
mens="varc="+varc+",  varn="+str(varn)
```

```
ret=MessageBox(mens,"Rotina chamada por Verify...","O","I")
return .f.
```

Observação

O programa acima produz, entre outras, a seguinte tela:

1.0.0 Controle Init

Especifica uma função que será invocada na inicialização da Caixa de diálogo, permitindo, assim, agrupar procedimentos a serem executados automaticamente sobre quaisquer dos controles da mesma.

Sintaxe

Init <func>(<param1>, ...< paramn>)

Onde:

<func>

é o nome da função e <param1> até <paramn> são os parâmetros passados a essa função.

1.0.0 Controle CheckBox

O comando **CheckBox** apresenta listas de itens, precedidos por pequenas "caixas de verificação" ("Check Boxes"), permitindo **marcar** (ou seja, "checar") *uma ou mais* opções dessa lista.

A sintaxe geral deste comando é:

```
CheckBox <nome>  <varl> <xi,yi,xl,yl> [IDOK | IDCANCEL]
                 [ID <expc>] [ENABLE|DISABLE]
                 [CONTEXTid(<num>)]  [message <texto>)] [tip <tip>]
```

Onde:

- ***<nome>*** especifica o título ou texto que acompanha a pequena caixa de verificação.
- ***<varl>*** indica uma variável lógica, cujo valor será **.t.** (verdadeiro) se o controle associado foi selecionado, ou seja se a caixa de verificação correspondente foi marcada ("ticada").
- ***<xi,yi,xl,yl>*** são as coordenadas do controle (CheckBox) que está sendo definido.
- **[IDOK | IDCANCEL]** informa a maneira como as variáveis de um Quadro de Diálogo vão ser tratadas. A opção **IDOK** permite que o diálogo será terminado e a variável ***<varl>*** será atualizada. A opção **IDCANCEL** indica que o diálogo será concluído, porém, as variáveis envolvidas **não serão atualizadas.**
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<*num*>** especifica o identificador de contexto ("context identifier") associado a um determinado tópico quando o arquivo de Help foi projetado e construído.
- **<texto>** especifica uma mensagem associada ao controle, a ser exibida na barra de status (automaticamente gerada pela OpusWin) quando for selecionado o item em questão.
- **<tip>** especifica um texto a ser mostrado, como "dica" quando o mouse for deslizado sobre este controle.

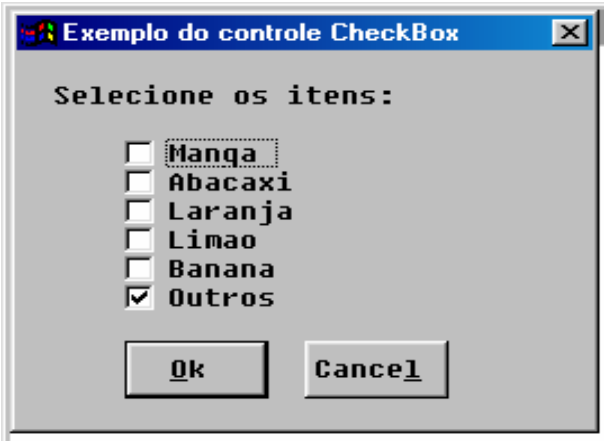
Exemplo

```
$nolib
prog
decl vet_cbx[06]=space(16)
vet_cbx[01]="Manga"
vet_cbx[02]="Abacaxi"
vet_cbx[03]="Laranja"
vet_cbx[04]="Limao"
vet_cbx[05]="Banana"
```

```
vet_cbx[06]="Outros"
bt=1
cbx6 = .t.
cbx1, cbx2, cbx3, cbx4, cbx5 = .f.
Dialog 02,02,32,13
  caption "Exemplo do controle CheckBox"
  style WS_POPUP WS_CAPTION WS_SYSMENU
  Ltext "Selecione os itens:" NULL 02,01,20,01
  CheckBox vet_cbx[01] cbx1 06,03,16,01
  CheckBox vet_cbx[02] cbx2 06,04,16,01
  CheckBox vet_cbx[03] cbx3 06,05,16,01
  CheckBox vet_cbx[04] cbx4 06,06,16,01
  CheckBox vet_cbx[05] cbx5 06,07,16,01
  CheckBox vet_cbx[06] cbx6 06,08,16,01
  DefPushButton "&Ok" bt 06,10,08,02 IDOK
  PushButton "Cance&l" NULL 16,10,08,02 IDCANCEL
EndDialog
return
```

Observação

Os comandos do exemplo acima produzirão a seguinte DialogBox:



1.0.0 Controle ListBox

O comando **ListBox** exibe uma lista de itens ou opções a serem selecionados. Normalmente, este recurso é utilizado para visualizar os dados de uma consulta.
Podem ser associadas mensagens a cada um dos elementos da ListBox. As mensagens associadas serão exibidas na barra de status quando o elemento correspondente na ListBox for selecionado com **um** "click" do mouse.
A sintaxe geral deste comando é:

```
ListBox <vetc> <varn> <xi,yi,xl,yl> <estilo> [Font (<"font">,<tam>)]
                [ID <expc>] [ENABLE|DISABLE]
                [IDOK | IDCANCEL]  [CONTEXTid(<num>)]
                [MESSAGE <vetmsg>]  [tip <tip>]
                [select <func> (<par1>, ... ,<parn>)]
                [dblclk <func> (<par1>, ... ,<parn>)]
                [valid <func>(<p1>, <pn>)]
```

Onde:

- ***<vetc>*** especifica um vetor cadeia que contém os elementos da ListBox a serem exibidos na tela do usuário.
- ***<varn>*** indica uma variável numérica que vai conter o número do item selecionado pelo usuário. **Um** "click" do mouse permite selecionar o elemento, atualizando as variáveis envolvidas. **Dois** "clicks" efetuam o processamento do elemento selecionado.
- ***<xi,yi,xl,yl>*** são as coordenadas do controle (ListBox) sendo definido
- ***<estilo>*** define os estilos associados com a **ListBox** , podendo ser a combinação dos estilos apresentados na tabela abaixo.
- **Font (<"font">,<tam>)** especifica o nome e o tamanho da fonte a ser utilizada neste controle da Caixa de Diálogo.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlId, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.

- **[IDOK | IDCANCEL]** informa a maneira como as variáveis de um Quadro de Diálogo vão ser tratadas. A opção **IDOK** permite que o diálogo será terminado e a variável **<*varn*>** será atualizada. A opção **IDCANCEL** indica que o diálogo será concluído, porém, as variáveis envolvidas **não serão atualizadas.**
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<*num*>** especifica o identificador de contexto ("context identifier") associado a um determinado tópico quando o arquivo de Help foi construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.
- **<vetmsg>** especifica um vetor de mensagens associado à ListBox. Cada mensagem será exibida na barra de status sempre que for selecionado o elemento correspondente da ListBox.
- **select <func> (<p1>, ... <pn>),** associada ao comando **ListBox**, permite executar uma determinada função quando um dos elementos da ListBox for selecionado (um click do mouse). A função lógica **<func>** pode retornar um valor **.t.** ou **.f.**).
- **dblclk <func> (<p1>, ... <pn>),** associada ao comando **ListBox**, permite executar uma determinada função quando um dos elementos da ListBox for acionado (dois clicks do mouse). A função lógica **<func>** pode retornar um valor **.t.** ou **.f.**).
- A cláusula **valid <func> (<p1>, ... <pn>),** associada ao comando **ListBox**, facilita o controle do ambiente de programação**.** Quando um dos itens da ListBox for selecionado, a função lógica referenciada pela cláusula **Valid** será acionada. Nessa função, obviamente, será efetuada uma operação de **validação**. Caso a função acionada pelo comando **Valid** retorne um valor lógico verdade ("return (.t.)"), significa que as informações validadas estão corretas. Neste caso, o Quadro de Diálogo será concluído. Caso a função invocada retornar um valor lógico falso ("return(.f.)"), o Quadro de Diálogo permanece aberto e disponível na tela, permitindo, assim, enviar avisos, do tipo MessageBox ou Message na barra de status, para que o usuário acerte ou complemente as informações digitadas.
- **<tip>** especifica um texto a ser mostrado, como "dica" quando o mouse for deslizado sobre este controle.

Os seguintes estilos podem ser especificados no comando ListBox:

Estilos da ListBox	Descrição
LBS_USETABSTOPS	Permite que uma ListBox reconheça e interprete os caracteres de tabulação quando apresentar na tela os elementos dessa ListBox. Isto permite organizar e alinhar os elementos de uma ListBox, como vemos no exemplo a seguir.
WS_VSCROLL	Permite criar janelas com barra de rolagem vertical.
WS_HSCROLL	Permite criar janelas com barra de rolagem horizontal.
DS_FIXEDSYS	Usa a fonte SYSTEM_FIXED_FONT em vez de SYSTEM_FONT

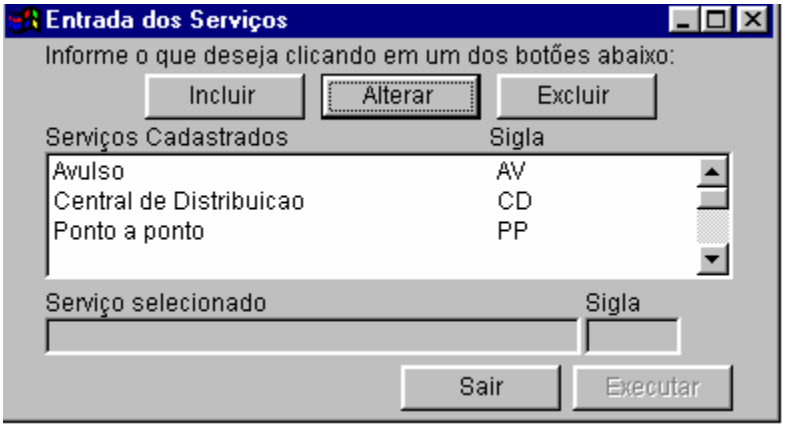
Exemplo 01

```
$library  = bibmotoca.lib
proc E_Servico
include motoca.h
*Funçoes utilizadas
func ServicoSelect(l)
func SenhaOK(l)
func SetupS(l)
func Incluir(l)
func Alterar(l)
func Remover(l)
func ExecutarServ(l)
func ValidaSiglaS(l)
func PegaServico(n)
**Verificação do direito de acesso
if !SenhaOK(5)
        return
endif
**Le os servicos para vServico e vCodServ do RefrescaTela
*Monta a Tela de entrada/modificações de Serviços
Dialog 0, 0, 192, 100 units
  caption "Entrada dos Serviços"
  style WS_OVERLAPPEDWINDOW DS_CENTER
  init SetupS()
  ListBox  vListBoxS nServico 10, 30, 172, 40 font("arial", 8);
           WS_VSCROLL LBS_USETABSTOPS select ServicoSelect()
  EditText NULL aServico  10,  74, 130, 10 font("arial", 8)
  EditText NULL aSiglaS   145, 74, 20,  10 font("arial", 8);
```

```
            ES_UPPERCASE ; valid ValidaSiglaS()
  DefPushButton "Executar" retorno 142, 87, 40, 12  font("arial", 8);
            valid ExecutarServ()
  PushButton "Sair" retorno  99,  87, 40, 12 IDOK font("arial", 8)
  PushButton "Incluir" NULL 35, 9, 40, 12 font("arial", 8) valid Incluir()
  PushButton "Alterar" NULL 79, 9, 40, 12 font("arial", 8) valid Alterar()
  PushButton "Excluir" NULL 123, 9, 40, 12 font("arial", 8) valid Remover()
  Ltext   "Informe o que deseja clicando em um dos botões abaixo:";
            NULL 10, 00, 190, 08 font("arial", 8)
  Ltext "Serviços Cadastrados" NULL 10, 22, 100, 08 font("arial", 8)
  ltext "Sigla" NULL 121, 22, 20,   08 font("arial", 8)
  Ltext "Serviço selecionado"         NULL    10,  66, 100,  08 font("arial", 8)
  ltext "Sigla"    NULL     145, 66, 20,   08 font("arial", 8)
EndDialog
if retorno = 1 && Saida
        return
endif
end
```

Observação

O exemplo acima produz a seguinte tela no Windows:



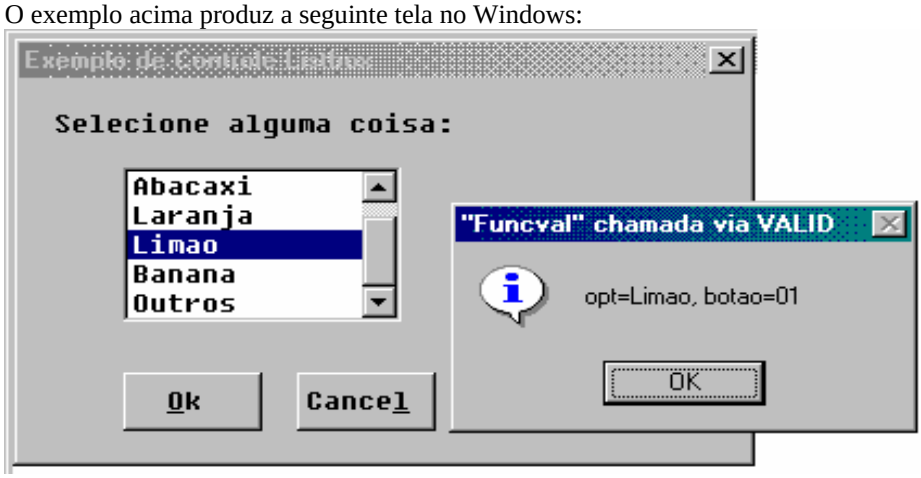
Exemplo 02

```
$nolib
prog
func funcval(l)
decl public vet_cbx[06]=space(16)
vet_cbx[01]="Manga"
vet_cbx[02]="Abacaxi"
vet_cbx[03]="Laranja"
vet_cbx[04]="Limao"
vet_cbx[05]="Banana"
vet_cbx[06]="Outros"
bt=1
opt=6
Dialog 02,02,42,13
  caption "Exemplo de Controle ListBox"
  style WS_POPUP  WS_CAPTION WS_SYSMENU
  Ltext "Selecione alguma coisa:" NULL 02,01,24,01
  ListBox vet_cbx opt 06,03,16,06 WS_Vscroll IdOk;
    valid funcval (opt, bt)
  DefPushButton "&Ok" bt 06,10,08,02 IDOK;
    valid funcval (opt, bt)
  PushButton "Cance&l" NULL 16,10,08,02 IDCANCEL;
    valid funcval (opt, bt)
EndDialog
quit
func funcval
parameters opt(n),bt(n)
if bt=2      && cancel ?
  return .t.
endif
decl public vet_cbx[06]=space(16)
mens="opt="+vet_cbx[opt]+", botao="+str(bt,2)
```

```
ret=MessageBox(mens,'"Funcval" chamada via VALID',"O","I")
return .f.
```

Observação

O exemplo acima produz a seguinte tela no Windows:



1.0.0 Controle EditText

O comando **EditText** é utilizado para construir o controle **EditBox** dentro de uma Caixa de Diálogo e mostra uma área retangular onde os dados podem ser informados (digitados), consultados ou alterados pelo usuário. Usualmente, o **EditBox** serve para entrada de dados, em cujo caso a área construída pelo comando **EditText** encontra-se vazia. Os estilos e as Pictures associados a este controle permitem um bom gerenciamento do seu conteúdo e apresentação.

A sintaxe geral deste comando é:

```
EditText NULL <varc> <xi,yi,xl,yl> <estilo> [IDOK]
                     [ID <expc>] [ENABLE|DISABLE]
                     [CONTEXTid(<num>)]
                     [PIC[ture] <editmask>] [Font (<"font">,<tam>)]  [tip <tip>]
                     [valid <func>(<p1>, <pn>)]  [when <func>(<p1>, <pn>)]
```

Onde:

- **<*varc*>** especifica uma variável cadeia para receber os dados digitados pelo usuário, podendo ter um valor inicial a ser apresentado para consulta, modificação ou eliminação.
- **<*xi,yi,xl,yl*>** são as coordenadas e tamanhos do controle EditBox.
- **<*estilo*>** define os estilos associados com a EditBox , podendo ser a combinação dos estilos constantes na tabela apresentada mais adiante.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlId, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<*num*>** especifica o identificador de contexto ("context identifier") associado a um determinado tópico quando o arquivo de Help foi construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.
- **<PICTURE <editmask>** serve para especificar uma máscara de edição da mesma maneira que é utilizada na linguagem OPUS para aplicações não gráficas. Este tópico é abordado, de maneira especial, no manual da OPUS.
- **Font (<"font">,<tam>)** especifica o nome e o tamanho da fonte a ser utilizada neste controle, dentro da Caixa de Diálogo.
- A cláusula **valid <func> (<p1>, ... <pn>),** associada ao comando **EditText,** facilita o controle do ambiente de programação. Quando a Caixa de Edição for alterada, se ela tiver associada uma clausula **Valid,** a função de validação será invocada. Se a função acionada pelo comando **Valid** retornar um valor lógico verdade ("return (.t.)"), significa que as informações validadas estão corretas, sendo processadas e concluída a Caixa de Diálogo. Caso contrário, ou seja, se a função invocada retornar um valor lógico falso ("return(.f.)"), o Quadro de Diálogo permanece aberto e disponível na tela, permitindo, assim que o usuário acerte ou complete as informações digitadas na Caixa de Edição.
- **<tip>** especifica um texto a ser mostrado, como "dica" quando o mouse for deslizado sobre este controle.

Alguns dos estilos da EditBox, disponíveis no Windows, podem ser especificados na opção **<estilo>** do comando EditText. Geralmente, os nomes destes estilos iniciam com o prefixo **ES_** (de **E**ditBox **S**tyle). Porém, os próprios recursos da **OpusWin** implementam, de formas alternativas, as características dos estilos Windows. É o caso, por exemplo, da cláusula **PICTURE.**

Caso seja necessário, os seguintes estilos Windows podem ser especificados para um controle EditBox, dentro de um programa OpusWin:

Estilos de EditBox	Descrição
ES_AUTOHSCROLL	Automaticamente faz rolagem horizontal do texto editado.
ES_AUTOVSCROLL	Automaticamente faz rolagem vertical do texto editado.
ES_CENTER	Centraliza o texto, dentro de um controle de edição com múltiplas linhas.
ES_LEFT	Alinha o texto à esquerda.
ES_LOWERCASE	Converte para minúsculos quaisquer caracteres, na medida em que são digitados dentro do controle de edição.
ES_MULTILINE	Designa um controle de edição do tipo MULTILINE. Não especificando este estilo, assume-se que o controles tem apenas uma única linha. Observações: • Quando o controle de edição "multiline" se encontra dentro de uma caixa de diálogo, a tecla ENTER aciona o botão default (ou seja, o "DefPushButton"). • Se for necessário utilizar a tecla ENTER como "carriage return" (passando para uma outra linha), deve ser especificado o estilo ES_WANTRETURN. • Se especificado o estilo ES_AUTOHSCROLL, haverá um scroll horizontal toda vez que o indicador do mouse atingir a borda direita do controle de edição. Para iniciar uma nova linha, o usuário deve pressionar a tecla ENTER.
ES_NUMBER	São aceitos apenas caracteres numéricos dentro de um controle de edição.
ES_PASSWORD	Exibe um asterisco (*) para cada caracter digitado dentro do controle de edição.
ES_READONLY	O conteúdo mostrado no controle de edição é read-only, não sendo permitido digitar nada sobre ele.
ES_RIGHT	Alinha o texto à direita dentro de um controle de edição de múltiplas linhas.
ES_UPPERCASE	Converte para maiúsculos todos os caracteres digitados dentro do controle de edição.
ES_WANTRETURN	Especifica que será inserido um "carriage return" sempre que o usuário tecla ENTER dentro do texto do controle de edição. Se não for especificado este estilo, pressionar ENTER produz o mesmo efeito que pressionar o botão default, definido em DefPushButton. Este estilo não se aplica o controles de edição com apenas uma linha de texto.
DS_FIXEDSYS	Usa a fonte SYSTEM_FIXED_FONT em vez de SYSTEM_FONT

Exemplo

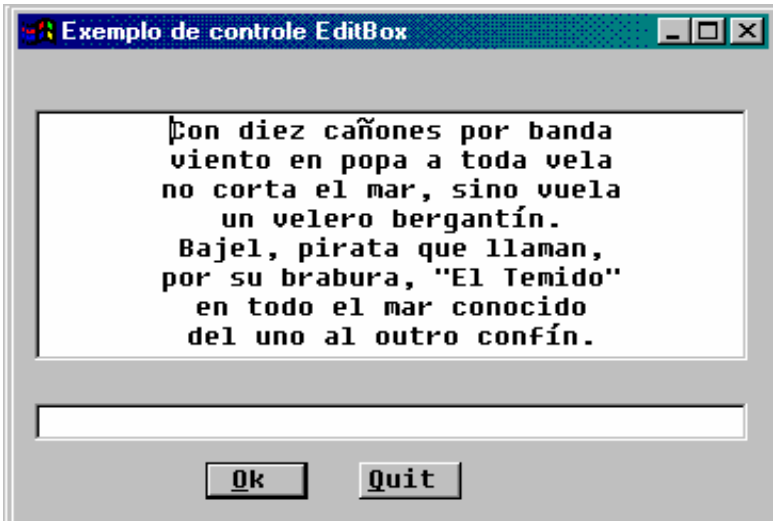
```
$nolib
$lentmp=320
prog
func vfy_geral(l)
decl vet[08]=space(40)
vet[1]="Con diez cañones por banda"
vet[2]="viento en popa a toda vela"
vet[3]="no corta el mar, sino vuela"
vet[4]="un velero bergantín."
vet[5]="Bajel, pirata que llaman,"
vet[6]='por su brabura, "El Temido"'
vet[7]="en todo el mar conocido"
vet[8]="del uno al outro confín."
private campo1(320),campo2(40)

campo1= vet[01] + chr(13) + chr(10)+ ;
      vet[02] + chr(13) + chr(10)+ ;
      vet[03] + chr(13) + chr(10)+ ;
      vet[04] + chr(13) + chr(10)+ ;
      vet[05] + chr(13) + chr(10)+ ;
      vet[06] + chr(13) + chr(10)+ ;
      vet[07] + chr(13) + chr(10)+ ;
```

```
      vet[08]
campo2=""
botao = -1
Dialog 2,2,54,16
 caption "Exemplo de controle EditBox"
 style WS_OVERLAPPEDWINDOW
 rtext "Multiline:" NULL 00,02,10,01
 EditText NULL campo1 11,02,40,-66 ;
   ES_MULTILINE ES_CENTER ES_AUTOHSCROLL ES_AUTOVSCROLL ES_WANTRETURN
 rtext "EditBox:" null 00,12,10,01
 EditText NULL campo2 11,12,40,01 ES_AUTOHSCROLL
 DefPushButton "&Ok" botao 11,14,-24,-10 IDOK
 PushButton "&Quit" NULL 20,14,-24,-10 IDCANCEL
 Verify vfy_geral (botao, campo1, campo2)
EndDialog
return
func vfy_geral
private mens(320)
parameters botao(n), campo1, campo2
if botao = 1    && ok
 mens="campo1="+trim(campo1)+ chr(10) + ;
    "campo2="+trim(campo2)
 ret=MessageBox(mens,"MessageBox no EditBox...","O","I")
 return .f.
endif
return .t.
```

Observação

O exemplo acima produz a seguinte tela no Windows:



1.0.0 Controle RichEdit

O controle RichEdit permite a visualização e impressão de arquivos **.TXT** e **.RTF**. O parâmetro **<arq>** (na sintaxe abaixo especificada) é uma variável que contém o nome do arquivo a ser visualizado ou impresso, que pode ser gerado como um relatório para a impressora utilizando o comando **SET PRINTER TO "file:<arquivo>"** antes do comando **SET PRINT ON**.
O comando EJECT (veja manual da OPUS) gera um comando **\f** a ser interpretado durante o processo de impressão do arquivo.
Os comandos **PrintControl** ou **PrintControlId** são utilizados para imprimir o arquivo visualizado pelo controle RichEdit.
A sintaxe geral do comando **RichEdit** é:

```
RichEdit NULL <arq> <xi,yi,xl,yl> <estilo> [IDOK]
                  [ID <expc>] [ENABLE|DISABLE]
                  [CONTEXTid(<num>)]
                  [PIC[ture] <editmask>] [Font (<"font">,<tam>)]  [tip <tip>]
                  [valid <func>(<p1>, <pn>)]  [when <func>(<p1>, <pn>)]
```

Onde:

- ***<arq>*** especifica o nome do arquivo a ser visualizado e, opcionalmente, impresso.

- **<*xi,yi,xl,yl*>** são as coordenadas e tamanhos do controle EditBox.
- **<*estilo*>** define os estilos associados com a EditBox , podendo ser a combinação dos estilos constantes na tabela apresentada mais adiante.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como PrintControlId, ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<*num*>** especifica o identificador de contexto ("context identifier") associado a um determinado tópico quando o arquivo de Help foi construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.
- **<tip>** especifica um texto a ser mostrado, como "dica" quando o mouse for deslizado sobre este controle.

Exemplo

```
$nolib
prog
arq="rel.txt"
func f1(l)
decl vet[08]=space(40)
…
Dialog …
…
 RichEdit NULL arq 5,5,40,20 ID("xxx")
…
…
PushButton "Imprime" NULL 5,26,7,2 valid(f1)
…
…
Func f1
PrintControlId ("xxx","PT")
return .t.
```

1.0.0 Controle ComboBox

O comando **ComboBox** é utilizado para construir listas de opções, numa **combinação** de **EditBox** com **ListBox**, permitindo, portanto, consultar, selecionar e modificar os elementos deste controle da Caixa de Diálogo. Sendo assim, o ComboBox apresenta, simultaneamente, uma **lista de seleção** (ListBox) e uma **caixa de edição** (EditBox).

A sintaxe geral deste comando é:

```
ComboBox <vetc> <varc> <xi,yi,xl,yl> <estilo> [Font (<"font">,<tam>)]
                       [ID <expc>] [ENABLE|DISABLE]
                       [MESSAGE (<vetmsg>)] [CONTEXTid(<num>)]  [tip <tip>]
                       [select <func> (<par1> ,,, <parn>)]
                       [dblclk <func> (<par1> ,,, <parn>)]
                       [valid <func>(<p1>,<pn>)]
```

Onde:

- **<*vetc*>** especifica um vetor cadeia que contém os elementos da ListBox, dentro da ComboBox.
- **<*varc*>** especifica uma variável cadeia para receber o campo digitado pelo usuário ou selecionado entre os elementos da ListBox, podendo ter um valor inicial a ser apresentado para consulta, modificação ou eliminação.
- **<*xi,yi,xl,yl*>** são as coordenadas e os tamanhos do controle ComboBox.
- **<*estilo*>** define os estilos associados com a ComboBox, podendo ser a combinação dos estilos apresentados mais adiante.
- **Font (<"font">,<tam>)** especifica o nome e o tamanho da fonte a ser utilizada neste controle.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlId, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.
- **<vetmsg>** especifica um vetor de mensagens associado ao vetor ListBox, dentro da ComboBox. Cada mensagem será exibida na barra de status sempre que for selecionado o elemento correspondente.
- **select <func> (<p1>, ... <pn>),** associada ao comando **ComboBox**, permite executar uma determinada função quando um dos elementos deste controle for selecionado (um click do mouse). A função lógica **<func>** pode retornar um valor **.t.** ou **.f.**).

- **dblclk <func> (<p1>, ... <pn>),** associada ao comando **ComboBox**, permite executar uma determinada função quando um dos elementos deste controle for acionado (dois clicks do mouse). A função lógica **<func>** pode retornar um valor **.t.** ou **.f.**).
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<*num*>** especifica o identificador de contexto ("context identifier") associado a um determinado tópico quando o arquivo de Help foi construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.
- A cláusula **valid <func> (<p1>, ... <pn>),** associada ao comando **ComboBox,** facilita o controle do ambiente de programação. Quando a Caixa de Edição (um dos componentes da ComboBox) for modificada, a função de validação **<func>** será invocada. Se esta função retornar um valor lógico verdade ("return (.t.)"), significa que as informações validadas estão corretas e serão processadas. Caso contrário, ou seja, se a função invocada retornar um valor lógico falso ("return(.f.)"), o Quadro de Diálogo permanece aberto e disponível para eventuais acertos e complementos.
- **<tip>** especifica um texto a ser mostrado, como "dica" quando o mouse for deslizado sobre este controle.

Alguns dos estilos, disponíveis no Windows, podem ser especificados na opção **<estilo>** do comando ComboBox. Geralmente, os nomes destes estilos iniciam com o prefixo **CBS_** (**ComboBox S**tyle). Porém, os próprios recursos da **OpusWin** implementam, de formas alternativas, as características dos estilos Windows. É o caso, por exemplo, da cláusula **PICTURE.**

Os seguintes estilos podem ser especificados no comando ComboBox. Observe que alguns estilos se referem a ListBox e outros a EditBox, pois, como você já sabe, o controle ComboBox é uma combinação desses outros dois controles.

Estilos da ComboBox	Descrição
CBS_AUTOHSCROLL	Automaticamente, faz rolagem horizontal do texto. Não especificando este estilo, somente será tratado o texto que cabe dentro dos limites do retângulo ComboBox.
CBS_DROPDOWN	Similar ao estilo CBS_SIMPLE, exceto que a Lista de Seleção não é exibida a não ser que o usuário a selecione. Permite digitar na Caixa de Edição qualquer dado diferente dos elementos da Lista de Seleção.
CBS_LOWERCASE	Converte para minúsculas quaisquer caracteres colocados em maiúsculo dentro do controle de edição de uma ComboBox.
CBS_SIMPLE	Não permite digitar dados na Caixa de Edição. Exibe sempre a Lista de Seleção, sendo que o elemento correntemente selecionado é mostrado dentro na Caixa de edição.
CBS_UPPERCASE	Converte para maiúsculos quaisquer caracteres minúsculos colocados no controle de edição de uma ComboBox.
DS_FIXEDSYS	Usa a fonte SYSTEM_FIXED_FONT em vez de SYSTEM_FONT

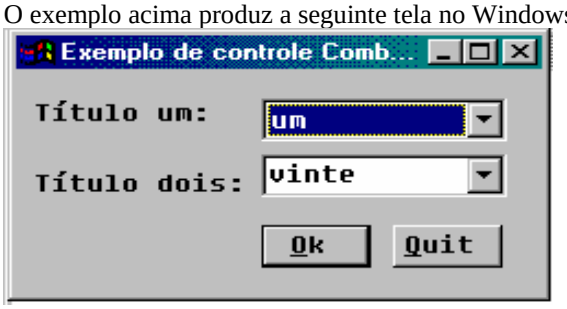
Exemplo

```
$nolib
prog
decl vetc[6]=space(16)
vetc[01]="um"
vetc[02]="dois"
vetc[03]="tres"
vetc[04]="quatro"
vetc[05]="cinco"
vetc[06]="Outros"
func funcval(l)
campo1, campo2 = ""
botao = -1
Dialog 08,08,120,64 units
  caption "Exemplo de controle ComboBox"
  style WS_OVERLAPPEDWINDOW
  Ltext "Título um:" NULL 4,08,48,8
  ComboBox vetc campo1 56,08,56,48 IDOK CBS_AUTOHSCROLL CBS_SIMPLE
  Ltext "Título dois:" null 4,28,48,8
  ComboBox vetc campo2 56,24,56,48 IDOK CBS_AUTOHSCROLL CBS_DROPDOWN
  DefPushButton "&Ok" botao 56,44,25,12 IDOK;
    valid funcval (campo1, campo2, botao)
  PushButton "&Quit" NULL 86,44,25,12 IDCANCEL
EndDialog
return
func funcval
```

```
parameters campo1, campo2, botao(n)
RefreshControl(2)
mens="botao="+str(botao)+", "+campo1+" - "+campo2
ret=MessageBox(mens,"MessageBox na ComboBox...","O","I")
return .t.
```

Observação

O exemplo acima produz a seguinte tela no Windows:



1.0.0 Controle GroupBox

O comando **GroupBox** é utilizado para **agrupar logicamente** outros controles dentro de uma área que fica cercada por uma borda e cujo título aparece na borda superior, alinhado à esquerda.
A sintaxe geral deste comando é:

GROUPBOX <nome> <varn> <xi,yi,xl,yl>
[ID <expc>] [ENABLE|DISABLE]
[IDOK | IDCANCEL] [CONTEXTid(<num>)]

Onde:

- **<*nome*>** especifica o título da GroupBox que é exibido na borda superior, alinhado à esquerda. A opção **<nome>** pode ser uma variável cadeia ou um literal.
- **<*varn*>** especifica uma variável numérica que vai receber o número seqüencial da opção selecionada pelo usuário, a partir de 01, na ordem de especificação no comando RadioButton. Veja o exemplo apresentado junto com o comando RadioButton, mais adiante.
- **<*xi,yi,xl,yl*>** são as coordenadas e tamanhos do controle (GroupBox).
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlId, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.
- **[IDOK | IDCANCEL]** informa a maneira como as variáveis de um Quadro de Diálogo vão ser tratadas. Por exemplo, a opção **IDOK** permite que, terminado o diálogo, sejam **atualizadas as variáveis** envolvidas, ou seja, **<*varn*>,** em nosso caso. A opção **IDCANCEL**, porém, indica que as variáveis envolvidas **não serão atualizadas**, ao término do diálogo.
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<*num*>** especifica o identificador de contexto (“Context identifier”) associado a um determinado tópico quando o arquivo de Help foi construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.

Exemplo

Veja o exemplo que acompanha a explicação do comando RadioButton.

1.0.0 Controle RadioButton

O comando **RadioButton** serve para exibir um grupo de itens, ou opções, precedidos de pequenos círculos (parecidos aos botões dos antigos rádios, daí o nome de RadioButton), utilizados para marcar, ou seja, selecionar, uma e apenas uma das opções, que são mutuamente exclusivas. Os botões de rádio são agrupados dentro de controles GroupBox, sendo neste comando onde é especificada a variável numérica que informa, ao programa, a opção escolhida, ou seja, o botão que foi marcado.
A sintaxe geral deste comando é:

RadioButton <nome> NULL <xi,yi,xl,yl>
[IDOK | IDCANCEL]
[ID <expc>] [ENABLE|DISABLE]
[CONTEXTid(<num>)] [tip <tip>]
[valid <func>(<p1>,<pn>)] [CONTEXTid(<num>)]
[font (<font>,<tam>)]

Onde:

- **<*nome*>** especifica o nome do item RadioButton, podendo ser uma variável cadeia ou um literal alfanumérico.

- ***<xi,yi,xl,yl>*** são as coordenadas e tamanhos da área do controle GroupBox, onde são agrupados outros controles.
- **[IDOK | IDCANCEL]** informa a maneira como as variáveis de um Quadro de Diálogo vão ser tratadas. Por exemplo, a opção **IDOK** permite que, terminado o diálogo, sejam **atualizadas as variáveis** envolvidas, ou seja, ***<varn>,*** em nosso caso. A opção **IDCANCEL**, porém, indica que as variáveis envolvidas **não serão atualizadas**, ao término do diálogo.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlId, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<*num*>** especifica o identificador de contexto ("Context identifier") associado a um determinado tópico quando o arquivo de Help foi construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.
- A cláusula **valid <func> (<p1>, ... <pn>)**, associada aos botões de pressionar**,** oferece alguns recursos importantes de programação, especialmente no que se refere à validação dos dados digitados e as opções escolhidas pelo usuário. Quando um desses botões for acionado, a função de validação, a ele associada, será invocada.
  Caso a função acionada pelo comando **Valid** retorne um valor lógico **verdade** ("**return (.t.)**"), significa que as informações foram validadas e podem ser processadas.
  Caso contrário, ou seja, se a função invocada retornar um valor lógico **falso** ("**return(.f.)**"), quer dizer que as informações estão incorretas ou incompletas. Nestes casos, o Quadro de Diálogo ainda permanece aberto e disponível na tela, permitindo enviar avisos (utilizando, por exemplo, Message Box , mensagem na Barra de Status ...) para que o usuário possa acertar as informações e opções escolhidas.
- **Font (<"font">,<tam>)** especifica o nome e o tamanho da fonte a ser utilizada neste controle da Caixa de Diálogo.
- **<tip>** especifica um texto a ser mostrado, como "dica" quando o mouse for deslizado sobre este controle.

Exemplo

A seguir, veja um exemplo de uma lista de itens RadioButton, dentro de uma GroupBox. A variável numérica **gbx**, especificada no controle GroupBox, receberá o número seqüencial que identifica a opção marcada pelo usuário, ou seja, o botão por ele acionado.

```
$nolib
prog
func funcval(l)
decl vetc[06]=space(16)
vetc[01]="Manga"
vetc[02]="Abacaxi"
vetc[03]="Laranja"
vetc[04]="Limao"
vetc[05]="Banana"
vetc[06]="Outros"
bt=1
gbx=6
Dialog 02,02,40,15
 caption "Exemplo de GroupBox"
 style WS_POPUP  WS_CAPTION WS_SYSMENU
 GroupBox "Selecione um dos itens:" gbx 02,01,32,12
 RadioButton vetc[01] NULL 04,03,16,01
 RadioButton vetc[02] NULL 04,04,16,01
 RadioButton vetc[03] NULL 04,05,16,01
 RadioButton vetc[04] NULL 04,06,16,01
 RadioButton vetc[05] NULL 04,07,16,01
 RadioButton vetc[06] NULL 04,08,16,01
 DefPushButton "&Ok" bt 04,10,08,02 IDOK
 PushButton "Cance&l" NULL 14,10,08,02 IDOK
 Verify funcval (bt, gbx)
EndDialog
if bt=2     && cancel ?
 quit
endif
end
func funcval
parameters bt(n),gbx(n)
if bt=2   && botao cancel pressionado ?
 return .t.
```

```
endif
mens="opt="+str(gbx,2)+", botao="+str(bt,2)
ret=MessageBox(mens,"comando VERIFY chamando funcval","O","I")
return .f.
```

Observação

No exemplo acima, assumimos que o usuário marcou o botão número 03, ou seja, o terceiro item da lista, na ordem de especificação no programa, que é o item "Laranja". O exemplo acima produzirá, quando executado, a seguinte tela no Windows:

1.0.0 Controles Ltext, Rtext e Ctext

Os comandos **Ltext, Rtext** e **Ctext** são controles que servem para exibir blocos de texto, alinhados, respectivamente, à esquerda, à direita ou ao centro.
A sintaxe geral desses comandos é:

```
LTEXT <texto> NULL <xi,yi,xl,yl>
   [ID <expc>] [ENABLE|DISABLE]
   [CONTEXTid(<num>)] [Font (<"font">,<tam>)]
RTEXT <texto> NULL <xi,yi,xl,yl>
   [ID <expc>] [ENABLE|DISABLE]
   [CONTEXTid(<num>)] [Font (<"font">,<tam>)]
CTEXT <texto> NULL <xi,yi,xl,yl>
   [ID <expc>] [ENABLE|DISABLE]
   [CONTEXTid(<num>)] [Font (<"font">,<tam>)]
```

Onde:

- **<texto>** informa o bloco de texto a ser exibido. Poderá ocupar várias linhas, desde que sejam incluídos os controles CR ou LF necessários. Porém, o texto **<texto>** possui tamanho máximo de 80 caracteres.
- ***<xi,yi,xl,yl>*** são as coordenadas e tamanhos do controle que está sendo definido.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlId, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<num>** especifica o identificador de contexto ("Context identifier") associado a um determinado tópico quando o arquivo de Help foi construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.
- **Font (<"font">,<tam>)** especifica o nome e o tamanho da fonte a ser utilizada neste controle da Caixa de Diálogo.

Exemplo

A seguir, veja um exemplo utilizando o controle Ctext.

```
$nolib
prog
bt=1
linha1="Com diez cañones por banda"
linha2="viento en popa a toda vela"
linha3="no corta el mar, sino vuela"
linha4="un velero bergantín."
linha5="Bajel, pirata que llaman,"
```

```
linha6='por su brabura, "El Temido"'
linha7="en todo mar conocido"
linha8="del uno al outro confín."
Dialog 2,2,32,14
 caption "Exemplo de Ctext"
 style WS_POPUP WS_CAPTION WS_SYSMENU DS_CONTEXTHELP
 ctext linha1 NULL 02,01,30,01
 ctext linha2 NULL 02,02,30,01
 ctext linha3 NULL 02,03,30,01
 ctext linha4 NULL 02,04,30,01
 ctext linha5 NULL 02,05,30,01
 ctext linha6 NULL 02,06,30,01
 ctext linha7 NULL 02,07,30,01
 ctext linha8 NULL 02,08,30,01
 DefPushButton "&Quit" bt 11,11,08,02 IDCANCEL
EndDialog
quit
```

Observação

Execute o programa acima para visualizar os resultados.

1.0.0 Controle VarText

O comando **VarText** exibe na tela o conteúdo da uma variável cadeia, sendo dinâmico o seu gerenciamento. Isto significa que a variável em questão poderá ter o seu conteúdo atualizado ("refrescado") enquanto estiver ativa a sua Caixa de Diálogo. Esta característica dinâmica constitui a diferença do **VarText** em relação aos comandos **Ctext, Ltext e Rtext.**

A sintaxe geral deste comando é:

```
VarText NULL <varc> <xi,yi,xl,yl>
  [ID <expc>] [ENABLE|DISABLE]
  [CONTEXTid(<num>)] [Font (<"font">,<tam>)]
```

Onde:

- **<varc>** especifica a variável a ser exibida e atualizada dinamicamente.
- ***<xi,yi,xl,yl>*** são as coordenadas e tamanhos da variável **<varc>.**
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlId, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<*num*>** especifica o identificador de contexto ("Context identifier") associado a um determinado tópico quando o arquivo de Help foi construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.
- **Font (<"font">,<tam>)** especifica o nome e o tamanho da fonte a ser utilizada neste controle da Caixa de Diálogo.

Exemplo

A seguir, veja um exemplo utilizando o controle **VarText**.

```
$nolib
prog
func fvar(l)
bt = 1
ivet = 1
decl vet[08]=space(32)
vet[1]="Con diez cañones por banda"
vet[2]="viento en popa a toda vela"
vet[3]="no corta el mar, sino vuela"
vet[4]="un velero bergantín."
vet[5]="Bajel, pirata que llaman,"
vet[6]='por su brabura, "El Temido"'
vet[7]="en todo el mar conocido"
vet[8]="del uno al outro confín."
varc=vet[01]
Dialog 02,02,38,07
 caption "Exemplo de VarText"
 style WS_POPUP WS_CAPTION WS_SYSMENU
 Vartext NULL varc 02,02,32,01
 DefPushButton "&Ok" bt 02,04,08,02;
  valid fvar (varc, vet, ivet)
```

```
  PushButton "&Quit" NULL 12,04,08,02 IDCANCEL
EndDialog
if bt = 2   && Quit
  quit
endif
return
func fvar(l)
parameters varc, vet[], ivet(n)
++ivet
if ivet>8
  varc="Acabó la poesía. Ahora: Quit!"
  return .t.
endif
varc = vet[ivet]
return .t.
```

Observação

Execute o programa acima e pressione o botão OK para visualizar os resultados. Quando solicitado, pressione o botão Quit.

1.0.0 Controle Icon

O comando **Icon** exibe um ícone na posição indicada pelas coordenadas <***xi,yi***>, e passa a ser identificado pelo número <***num***>. A largura e altura <***xl,yl***> devem ser especificadas com o valor **zero**, aceitando os tamanhos padrão do Windows. O controle **<*icon*>** deve ser único para cada DialogBox e deve ser **especificado por último**, após todos os outros controles da Caixa de Diálogo. Quando existam várias DialogBox e vários comandos **Icon**, cada um deles deve possuir um identificador (**<*num*>*)* diferente.
A sintaxe geral deste comando é:

```
Icon <arq-ico> <num> <xi,yi,0,0>
      [ID <expc>] [ENABLE|DISABLE]
```

Onde:

- **<arq-ico>** especifica o nome do arquivo, tipo **.ico**, que contém o ícone a ser exibido.
- **<num>** especifica um número identificador do recurso, ou seja do ícone
- <***xi,yi,0,0***> são as coordenadas para posicionar o ícone na tela.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlID, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.

Exemplo

A seguir, veja um exemplo de uma Caixa “About”, onde apresentamos um ícone.

```
$nolib
prog
xtext="OpenOPUS - OPUS for Windows"
botao=1
Dialog 0,0,35,08
  Font "Arial Black" 08
  style WS_POPUP DS_MODALFRAME
  ctext xtext NULL 1,2,32,1
  ctext "Copyright (c) Tecnocoop Sistemas" NULL 1,3,32,1
  ctext "1997,  Rio de Janeiro  -  BRAZIL" NULL 1,4,32,1
  defpushbutton "&Ok" botao 14,6,6,-12 IDOK
  icon "world.ico" 1 14,0,0,0
EndDialog
quit
```

Observação

O exemplo acima produzirá a seguinte tela no Windows:

1.1.1 Controle Image

O comando **Image** exibe uma imagem dentro de uma Caixa de Diálogo. A imagem será posicionada nas coordenadas indicadas com os tamanhos informados, recebendo o identificador numérico <num>.
A sintaxe geral deste comando é:

```
Image NULL <varc> <num> <xi,yi,xl,yl>
      [ID <expc>] [ENABLE|DISABLE]
```

Onde:

- **<varc>** especifica o nome do arquivo (**.bmp**, **.gif** ou **.jpg**) que contém a imagem a ser exibida.
- **<num>** especifica um número identificador do controle, ou seja da imagem.
- **<xi,yi,xl,yl>** são as coordenadas e tamanhos para posicionar e exibir corretamente a imagem na tela.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlID, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.

Exemplo

A seguir, veja um exemplo utilizando o comando **Image**.

```
$nolib
prog
func mostra_bmp(l)
set sigquit off
bt=1
var_com   = space(25)
varnom    = space(30)
varc      = space(100)
Dialog 00,00,40,23
  caption "Exemplo do comando Imagem"
  style WS_POPUP  WS_CAPTION  WS_SYSMENU
  image NULL varc  02,01,36,16
  DefPushButton    "Imagem&1"  bt 02,20,10,02  valid mostra_bmp(varc,bt)
  PushButton       "Imagem&2"  bt 14,20,10,02  valid mostra_bmp(varc,bt)
  PushButton       "&Sair"     bt 26,20,08,02  IDCANCEL
EndDialog
return
func mostra_bmp
parameters varc(c),bt(n)
if bt = 1
  varc="\windows\clouds.bmp"
elseif bt = 2
  varc="\windows\forest.bmp"
else
  varc=space(100)
endif
return .t.
```

1.0.0 Controle Cancel

O comando Cancel especifica uma função que será invocada quando for acionada a tecla ESC ou quando for fechada a janela do diálogo teclando o ícone X qua aparece no canto superior direito.
O evento Cancel, acionado por este controle, não invoca as funções valid associadas aos controles dentro de uma Caixa de diálogo.

Sintaxe

```
Cancel <func> (<par1>, ... <parn>)
```

1.0.0 Controles [Def]PushButton

Os comandos **[Def]PushButton** servem para especificar botões que, quando pressionados, acionam procedimentos. Podem ser especificados vários botões, dentro da mesma Caixa de Diálogo, sendo que um deles, e apenas um, deve ser o **botão padrão**, definido através de **DefPushButton**. Os outros botões são definidos através do comando **PushButton.** O botão **default** (aquele definido através de **DefPushButton)** é exibido na tela com o status de "pressionado", podendo ser acionado, também, através da tecla ENTER, mesmo que esse botão não esteja "focalizado".
A sintaxe geral deste comando é:

```
[Def]PushButton <nome> |  BITMAP("<bitmap>")
                       <varn> | NULL <xi,yi,xl,yl>
                         [IDOK | IDCANCEL]
                         [ID <expc>] [ENABLE|DISABLE]
                         [CONTEXTid(<num>)] [Font (<"font">,<tam>)] [tip <tip>]
                         [valid <func>(<p1>,<pn>)]
```

Onde:

- **<nome>** especifica o nome do botão, ou seja, o rótulo que aparece dentro do botão. Podem ser utilizadas as teclas aceleradoras, ou atalhos, para acionar os botões de pressionar (PushButtons). Para isso, basta marcar com o símbolo "&" um dos caracteres do rótulo do botão e utilizar a combinação ALT+caractere marcado com "&".
- **<varn>** especifica uma variável numérica que vai receber um número identificador do botão pressionado. Este número corresponde aos **[Def]PushButtons**, a partir de 01, na ordem em que eles foram especificados dentro da DialogBox.
  A cláusula **<varn>** é obrigatória na definição **DefPushButton**, sendo opcional em caso de **PushButton**. Neste caso, utiliza-se a cláusula NULL, pois a variável numérica **<varn>** serve para todos os **PushButtons** definidos na DialogBox, abaixo de um **DefPushButton.**
- **BITMAP("<bitmap>")** informa o nome de um arquivo, com a extensão **.BMP**, que contém uma imagem a ser incluída dentro do botão de pressionar. O tamanho e as coordenadas estão especificados em **<xi,yi,xl,yl>.**
- **NULL** é especificado nos PushButton, no lugar de **<varn>** do DefPushButton.
- **<xi,yi,xl,yl>** especifica as coordenadas e tamanhos do botões.
- **[IDOK | IDCANCEL]** informa a maneira como as variáveis de um Quadro de Diálogo vão ser tratadas. Por exemplo, a opção **IDOK** permite que, terminado o diálogo, sejam **atualizadas as variáveis** envolvidas, ou seja, ***<varn>*,** em nosso caso. A opção **IDCANCEL**, porém, indica que as variáveis envolvidas **não serão atualizadas**, ao término do diálogo.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlID, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.
- ***CONTEXTid(<num>)*** identifica um tópico de Help associado contextualmente (Help in Context) ao controle em questão. O parâmetro **<*num*>** especifica o identificador de contexto ("Context identifier") associado a um determinado tópico quando o arquivo de Help foi construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.
- **Font (<"font">,<tam>)** especifica o nome e o tamanho da fonte a ser utilizada neste controle da Caixa de Diálogo.
- A cláusula **valid <func> (<p1>, ... <pn>)**, associada aos botões de pressionar**,** oferece alguns recursos importantes de programação, especialmente no que se refere à validação dos dados digitados e as opções escolhidas pelo usuário. Quando um desses botões for acionado, a função de validação, a ele associada, será invocada.
  Caso a função acionada pelo comando **Valid** retorne um valor lógico **verdade** ("**return (.t.)**"), significa que as informações foram validadas e podem ser processadas.
  Caso contrário, ou seja, se a função invocada retornar um valor lógico **falso** ("**return(.f.)**"), quer dizer que as informações estão incorretas ou incompletas. Nestes casos, o Quadro de Diálogo ainda permanece aberto e disponível na tela, permitindo enviar avisos (utilizando, por exemplo, Message Box , mensagem na Barra de Status ...) para que o usuário possa acertar as informações e opções escolhidas.
- **<tip>** especifica um texto a ser mostrado, como "dica" quando o mouse for deslizado sobre este controle.

Exemplo

A seguir, veja um exemplo utilizando diversos **[Def]PushButton.**

```
$nolib
prog
func mostra_bmp(l)
bt=1
var_com   = space(25)
```

```
varnom    = space(30)
varc      = space(100)
Dialog 00,00,40,23
  caption "Exemplo do comando Imagem"
  style WS_POPUP  WS_CAPTION WS_SYSMENU
  image NULL varc  02,01,36,16
  DefPushButton    "Imagem&1"  bt 02,20,10,02  valid mostra_bmp(varc,bt)
  PushButton       "Imagem&2"  bt 14,20,10,02  valid mostra_bmp(varc,bt)
  PushButton   BITMAP ("clouds.bmp") NULL 02,20,10,02  valid mostra_bmp(varc,bt)
  PushButton       BITMAP ("forest.bmp") NULL 14,20,10,02  valid mostra_bmp(varc,bt)
  PushButton       "&Sair"     bt 26,20,08,02  IDCANCEL
EndDialog
return
func mostra_bmp
parameters varc(c),bt(n)
if bt = 1
  varc="\windows\clouds.bmp"
elseif bt = 2
  varc="\windows\forest.bmp"
else
  varc=space(100)
endif
return .t.
```

1.0.0 Controle ProgressBar

O controle ProgressBar serve para criar uma Barra de Progressão dentro de uma Janela de diálogo. A OpusWin permite também construir Barras de Progressão fora de DialogBox.

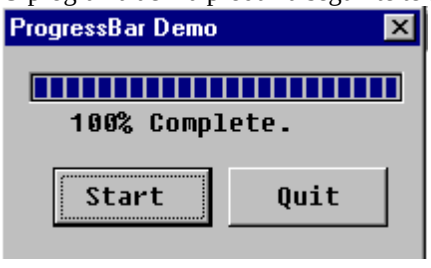
Numa definição de controle ProgressBar devemos informar:

- limite final a ser atingido pela progressão
- valor numérico da progressão
- As coordenadas e tamanhos do controle ProgressBar

Exemplo

```
$nolib
prog
b=1
Public pro(n), lim(n), varc
lim=100
pro=0
varc=""
func Incremen(l)
Dialog 36,39,104,56 units
  Style DS_MODALFRAME WS_POPUP WS_CAPTION WS_SYSMENU
  Caption "ProgressBar Demo"
  ProgressBar lim pro 12,12,188,16
  Vartext NULL varc 16,16,108,16
  DefPushButton "Quit" b 56,32,40,16 IDOK
  PushButton "Start" NULL 12,32,40,16 valid Incremen()
EndDialog
End
Func Incremen
Public pro(n), lim(n), varc
for pro = 0 to lim step 10
  RefreshControl (1)
  varc=str(pro)+"% Complete."
  RefreshControl (2)
  sleep 1
next
return .t.
```

O programa acima produz a seguinte tela:



1.0.0 Controle MonthCalendar

O controle MonthCalendar permite mostrar um calendário mensal, iniciando pela data informada, dentro de uma janela de diálogo.

Sintaxe

MonthCalendar <expc> <var> <xi>,<yi>,<xl>,<yl>
[IDOK | IDCANCEL]
[ID <expc>] [ENABLE|DISABLE]
[valid <func> (<par1>, ... ,<parn>)]
[font (<font>,<tam>)]

Onde:

- **<expc>** é uma expressão cadeia contendo o título ("caption") deste controle. Se for especificado NULL, a própria data será o caption do controle MonthCalendar.
- **<var>** é uma variável tipo cadeia, contendo a data inicial do Calendário sendo construído.
- **<xi>,<yi>,<xl>,<yl>** são as coordenadas e tamanhos do controle MonthCalendar. Se <xl>,<yl> são valores menores do que as dimensões do controle ou contém os valores 0,0 serão ignorados os valores <xl>,<yl>.
- **[IDOK | IDCANCEL]** informa a maneira como as variáveis de um Quadro de Diálogo vão ser tratadas. Por exemplo, a opção **IDOK** permite que, terminado o diálogo, sejam **atualizadas as variáveis** envolvidas, ou seja, ***<varn>*,** em nosso caso. A opção **IDCANCEL**, porém, indica que as variáveis envolvidas **não serão atualizadas**, ao término do diálogo.
- **ID <expc>** atribui um nome ao controle. Este nome identifica o controle e pode ser utilizado nos comandos tais como RefreshControlId, EnableControlId, SelectControlId ... etc ... **<expc>** é uma expressão cadeia.
- **[ENABLE | DISABLE]** indica que o controle deve ser habilitado ou desabilitado na inicialização da DialogBox. O valor default é ENABLE.
- **Font (<"font">,<tam>)** especifica o nome e o tamanho da fonte a ser utilizada neste controle da Caixa de Diálogo.

Observações

- A seleção da data é feita por duplo click
- As dimensões do controle dependem da fonte estabelecida
- Comando RefreshControl (<num>) pode ser utilizado para enviar uma nova data.

Exemplo

```
$nolib
Prog
Func p1(l)
Bt1=1
Dat="01/09/98"
Dialog 5,5,120,152 units
  Style WS_DLGFRAME WS_SYSMENU
  MonthCalendar "OpusWin MonthCalendar" dat 10,10,100,110
  DefPushButton "Ok" bt1 10,120,16,14 valid p1(dat)
EndDialog
Quit
Func p1
Parameters a
If ctod(a) > ctod("30/09/98")
  A=MessageBox("Invalid Date")
  Return (.f.)
Endif
Return (.t.)
```

O exemplo acima produz a seguinte tela:

1.0.0 Controle GRID

O controle GRID serve para construir "grids" dentro das Caixas de Diálogo para **exibir** e **editar**:

- Elementos de vetores
- Itens de arquivos em Bancos de Dados OpenBASE
- Campos existentes em arquivos externos

1.0.0.0 Elementos de vetores

Em uma Caixa de Diálogo pode ser especificada uma GRID para edição de elementos de vetores.

Sintaxe

GRID NULL <num> <xi>,<yi>,<xl>,<yl> [font(<font>,<tam>)] [DTS_ROWNUMBERS | DTS_EDIT]
COLUMN <cab1> <vet1> 0,0,<xl>,<yl> [picture <pic>] [valid <fun>]
…
…
COLUMN <cabn> <vetn> 0,0,<xl>,<yl> [picture <pic>] [valid <fun>]

Onde:

<num>	**número de colunas associadas ao GRID**
<cab.>	**cabeçalho da coluna**
<vet.>	**vetor associado à coluna**
<pic>	**picture a editar valor**
<fun>	**função a validar valor**

Observações

- A função numérica GRIDROW() indica a linha (ou seja, elemento do vetor) que está sendo validada.
- O estilo DTS_ROWNUMBERS indica que serão usados números de linhas.
- O estilo DTS_EDIT indica que as linhas e células são editáveis.
- A função RefreshControl[ID] permite a "re-exibição" dos elementos do(s) vetor(es)

Exemplos

…
…
Grid NULL 2 1,1,40,10
Column "Vetor 1" vet1 0,0,20,10
Column "Vetor 2" vet2 0,0,20,10
…
…

1.0.0.0 Itens de arquivos OpenBASE

Em uma Caixa de Diálogo pode ser especificada uma GRID para edição de arquivos de Bancos OpenBASE.

Sintaxe

GRID <arq> <num> <xi>,<yi>,<xl>,<yl> [font(<font>,<tam>)]
[DTS_ROWNUMBERS | DTS_EDIT | DTES_INSKEY | DTES_DELKEY]
COLUMN <cab1> <item1> 0,0,<xl>,<yl> [picture <pic>] [valid <fun>]
…
…
COLUMN <cabn> <itemn> 0,0,<xl>,<yl> [picture <pic>] [valid <fun>]

Onde:

<arq>	**nome do arquivo OpenBASE a editar**
<num>	**número de colunas associadas ao GRID**
<cab.>	**cabeçalho da coluna**
<item.>	**item associado à coluna**
<pic>	**picture a editar valor**
<fun>	**função a validar valor**

Observações

- O estilo DTS_ROWNUMBERS indica que serão usados números de linhas.
- O estilo DTS_EDIT indica que os registros e itens das células são editáveis.
- O estilo DTES_INSKEY indica que registros podem ser incluídos.
- O estilo DTES_DELKEY indica que registros podem ser removidos.
- A função RefreshControl[ID] permite a "re-leitura" dos registros do arquivo.

Exemplos

```
…
…
Grid "PESSOA" 2 1,1,40,10
Column "Nome" NOME_P 0,0,20,10
Column "Idade" IDADE 0,0,20,10
…
…
```

1.0.0.0 Campos de arquivo externo

Em uma Caixa de Diálogo pode ser especificada uma GRID para edição de arquivos externos.

Sintaxe

```
GRID EXTERN <num> <xi>,<yi>,<xl>,<yl> [font(<font>,<tam>)]
      [DTS_ROWNUMBERS | DTS_EDIT | DTES_INSKEY | DTES_DELKEY]
COLUMN <cab1> <campo1> 0,0,<xl>,<yl> [picture <pic>] [valid <fun>]
…
…
COLUMN <cabn> <campon> 0,0,<xl>,<yl> [picture <pic>] [valid <fun>]
```

Onde:

<num>	**número de colunas associadas ao GRID**
<cab.>	**cabeçalho da coluna**
<campo.>	**vetor associado à coluna**
<pic>	**picture a editar valor**
<fun>	**função a validar valor**

Observações

- O arquivo externo deve estar definido, através do comando USE, antes do diálogo e deve ser do tipo **S**, **R** ou **I** (registros de tamanho fixo) - Veja manual da OPUS para maiores detalhes a respeito dos tipos de arquivos externos
- O estilo DTS_ROWNUMBERS indica que serão usados números de linhas.
- O estilo DTS_EDIT indica que os registros e itens das células são editáveis.
- O estilo DTES_INSKEY indica que registros podem ser incluídos.
- O estilo DTES_DELKEY indica que registros podem ser removidos.
- A função RefreshControl[Id] permite a "re-leitura" dos registros do arquivo.

Exemplos

```
Use *
Arqtes R len(12)
Nome U10
Idade N2
enduse
…
…
Grid EXTERN 2 1,1,40,10
Column "Nome" Nome 0,0,20,10
Column "Idade" Idade 0,0,20,10
…
…
```

1.0 Opções associadas aos controles das Caixas de Diálogo

Apresentamos a seguir algumas das opções associadas aos controles definidos em Caixas de Diálogo. Veja os tópicos dedicados a cada controle para obter maiores detalhes.

1.0.0 Opção Valid

A opção **Valid,** associada aos controles do tipo **[Def]PushButton, ListBox, ComboBox e EditBox**, serve para invocar uma função lógica, passando para ela, opcionalmente, como parâmetros, as variáveis associadas aos controles da Caixa de Diálogo ou quaisquer outros itens fora da DialogBox.
O objetivo da função invocada pela cláusula **Valid** é, obviamente, fazer a **validação** das informações e processá-las corretamente, retornando ao processamento da Caixa de Diálogo. Se a função acionada retornar um valor lógico verdade ("return (.t.)"), o Quadro de Diálogo será encerrado. Porém, se a função invocada retornar um valor lógico falso ("return(.f.)"), o Quadro de Diálogo permanece aberto e disponível na tela, permitindo, assim, enviar avisos, do tipo MessageBox ou Message na barra de status, para que o usuário tome as devidas providências, por exemplo, corrigir ou completar um dado que foi digitado incorretamente.
A cláusula **Valid** poderá ser utilizada, também, para **executar procedimentos** associados a eventos como, por exemplo, acionar um PushButton ou selecionar o item de uma lista.

Exemplo

O exemplo que segue executa a função lógica "**mostra_bmp**" quando acionados os botões "Imagem1" e "Imagem2", mostrando uma figura para cada botão acionado. Uma Caixa de Mensagem informa o arquivo que contém a imagem a ser mostrada. Compile este exemplo e modifique-o conforme você desejar.

```
$nolib
prog
func mostra_bmp(l)
set sigquit off
bt=1
varc = ""
Dialog 00,00,40,22
  caption "Exemplo do comando Valid"
  style WS_OVERLAPPEDWINDOW
  image NULL varc 02,01,36,16
  DefPushButton "Imagem&1" bt 02,20,10,-12 valid mostra_bmp(varc,bt)
  PushButton "Imagem&2" bt 14,20,10,-12 valid mostra_bmp(varc,bt)
  PushButton "&Sair" bt 26,20,08,-12 IDCANCEL
EndDialog
quit
func mostra_bmp
parameters varc(c),bt(n)
if bt = 1
  varc="\windows\clouds.bmp"
  c=MessageBox(varc,"Vou mostrar a imagem em...","O","I")
elseif bt = 2
  varc="\windows\forest.bmp"
  c=MessageBox(varc,"Vou mostrar a imagem em...","O","I")
else
  varc=""
endif
return .t.
```

1.0.0 Opção When

A opção **when <func> (<p1>, ... <pn>),** associada ao comando **EditText**, permite disponibilizar, condicionalmente, um controle do tipo **EditBox**, dependendo do valor retornado pela função lógica **<func>.** Esta opção pode ser utilizada, com proveito, para controle de entrada de dados e preenchimento de formulários eletrônicos.

Exemplo

```
$nolib
prog
cbnom,cbsex,cbend,cbtel=.f.
func fval(l)
func mbox(l)
varnom,varsex,varend,vartel = ""
botao=1
Dialog 00,00,36,14
  style WS_OVERLAPPEDWINDOW
  font "Courier New"
```

```
  ltext "Marque campos a preencher:" NULL 01,00,26,01
  checkbox "Nome" cbnom 05,02,10,01
  checkbox "Sexo" cbsex 05,03,10,01
  checkbox "Endereço" cbend 05,04,10,01
  checkbox "Telefone" cbtel 05,05,10,01
  Rtext "Nome:" NULL 02,07,10,01
  EditText NULL varnom 12,07,20,1  when fval(cbnom)
  Rtext "Sexo:" NULL 02,08,10,01
  EditText NULL varsex 12,08,01,1  pic "!" when fval(cbsex)
  Rtext "Endereco:" NULL 02,09,10,01
  EditText NULL varend 12,09,20,1  pic rep("!",40) when fval(cbend)
  Rtext "Fone:" NULL 02,10,10,01
  EditText NULL vartel 12,10,10,1  when fval(cbtel)
  DefPushButton "&OK" botao 12,12,06,-12;
    valid mbox(botao,varnom,varsex,varend,vartel)
  PushButton "&Reset" botao 20,12,06,-12;
    valid mbox(botao,varnom,varsex,varend,vartel)
  PushButton "&Quit" NULL 28,12,06,-12 IDCANCEL
EndDialog
if botao = 2
  quit
endif
return
func  fval
parameters cbx(l)
return cbx
func mbox
parameters botao(n),varnom,varsex,varend,vartel
if botao = 2
  varnom,varsex,varend,vartel = ""
  ret=MessageBox ("Vou limpar as variáveis ...","","O","I")
  RefreshControl (6)
  RefreshControl (8)
  RefreshControl (10)
  RefreshControl (12)
  return (.t.)
endif
mens="nome="+varnom+chr(10)+;
     "sexo="+varsex+chr(10)+;
     "end.="+varend+chr(10)+;
     "tel.="+vartel
ret=MessageBox (mens,"Para você ver as variáveis...","O","I")
return (.t.)
```

Observações

No exemplo do programa acima, são apresentadas Caixas de Verificação, uma para cada campo a ser preenchido. O programa permite preencher apenas aqueles campos cuja CheckBox correspondente foi marcada. A seguir, podemos visualizar a tela do programa:

1.0 Comandos e funções de controle de Caixas de Diálogo

Apresentamos a seguir os comandos e funções freqüentemente utilizados para gerenciar as caixas de Diálogo.

1.0.0 Comandos EnableControl e EnableControlId

Os comandos **EnableControl** e **EnableControlId** habilitam determinados controles (previamente desabilitados) tornando-os disponíveis dentro de uma DialogBox.

Sintaxe

EnableControl (<num>)
EnableControlId (<id>)

Onde:
<num> identifica o controle a ser ativado, ou seja, disponibilizado. Todos os controles recebem um número seqüencial, a partir de 01, na ordem de especificação dentro da Caixa de Diálogo.
<id> especifica o nome atribuído ao controle, pela opção **ID <expc>,** quando da definição do mesmo dentro de um DialogBox.

1.0.0 Comandos DisableControl e DisableControlId

Os comandos **DisableControl** e **DisableControlId** desativam determinados controles dentro de uma DialogBox. O parâmetro **<num>** identifica o controle através de um número seqüencial, a partir de 01, na ordem de especificação dentro da Caixa de Diálogo.

Sintaxe

DisableControl (<num>)
DisableControlId (<id>)

Onde:
<num> identifica o controle a ser desativado. Todos os controles recebem um número seqüencial, a partir de 01, na ordem de especificação dentro da Caixa de Diálogo.
<id> especifica o nome atribuído ao controle, pela opção **ID <expc>,** quando da definição do mesmo dentro de um DialogBox.

1.0.0 Comandos MoveControl e MoveControlId

Os comandos **MoveControl** e **MoveControlId** deslocam o controle identificado pelo número **<num>** ou pela identificação **<id>** e o posiciona nas coordenadas especificadas em **<xi>,<yi>,<xl>,<yl>**.

Sintaxe

MoveControl (<num>,<xi>,<yi>,<xl>,<yl>)
MoveControlId (<id>,<xi>,<yi>,<xl>,<yl>)

Onde:
<num> identifica o controle a ser deslocado. Todos os controles recebem um número seqüencial, a partir de 01, na ordem de especificação dentro da Caixa de Diálogo.
<id> especifica o nome atribuído ao controle, pela opção **ID <expc>,** quando da definição do mesmo dentro de um DialogBox.
<xi>,<yi>,<xl>,<yl> especificam as coordenadas e os tamanhos para posicionar o controle que está sendo movido de lugar.

1.0.0 Comandos DeleteControl e DeleteControlId

Os comandos **DeleteControl e DeleteControlId** eliminam da tela o controle identificado pelo número **<num>** ou pela identificação **<id>**.

Sintaxe

DeleteControl (<num>)
DeleteControlId(<id>)

Onde:
<num> identifica o controle a ser deslocado. Todos os controles recebem um número seqüencial, a partir de 01, na ordem de especificação dentro da Caixa de Diálogo.
<id> especifica o nome atribuído ao controle, pela opção **ID <expc>,** quando da definição do mesmo dentro de um DialogBox.

1.0.0 Comando RefreshContent

O comando **RefreshContent** exibe na tela o conteúdo atualizado dos vetores que foram associados aos controles **ListBox** ou **ComboBox**. Este comando permite repintar na tela o conteúdo dos controles acima referenciados, desde que seus correspondentes vetores sejam previamente preenchidos ou atualizados no programa.

Sintaxe

RefreshContent (<str>)

Onde:

<str> identifica um vetor associado a ListBox ou ComboBox, assim como, também, uma variável associada a controles do tipo **EditText**, **CheckBox**, **GroupBox** e **Image**. O nome da variável pode ser passado como parâmetro ou a variável pode ser pública.

1.0.0 Comandos RefreshControl e RefreshControlId

Os comandos **RefreshControl** e **RefreshControl Id** exibem na tela o conteúdo atualizado do controle **<num>,** contado a partir de 01 dentro de uma DialogBox, ou do controle identificado por **<id>**. Este comando se aplica a **ListBox, EditBox, CheckBox, GroupBox, ComboBox e Image**, sendo similar ao comando RefreshContent. Se **<num>** é igual a –1 (um negativo) todos os controles do diálogo serão refrescados.

Sintaxe

RefreshControl (<num>)
RefreshControlId (<id>)

Onde:

<num> identifica o controle a ser atualizado. Todos os controles recebem um número seqüencial, a partir de 01, na ordem de especificação dentro da Caixa de Diálogo.
<id> especifica o nome atribuído ao controle, pela opção **ID <expc>,** quando da definição do mesmo dentro de um DialogBox.

1.0.0 Comandos SelectControl e SelectControlId

Os comandos **SelectControl** e **SelectControlId** permitem selecionar, ou seja, marcar como “selected” algum dos itens dos controles ListBox ou ComboBox.

Sintaxe

SelectControl (<num>,<expc>)
SelectControlId (<id>,<expc>)

Onde:

<num> identifica o controle a ser processado. Neste caso, pode ser uma ListBox ou uma ComboBox. Todos os controles recebem um número seqüencial, a partir de 01, na ordem de especificação dentro da Caixa de Diálogo.
<id> especifica o nome atribuído ao controle, pela opção **ID <expc>,** quando da definição do mesmo dentro de um DialogBox.
<expc> especifica o item a ser marcado como “selected” , dentro dos controles ListBox ou ComboBox especificados em **<num>**.

1.0.0 Comandos FocusControl e FocusControlId

Os comandos **FocusControl** e **FocusControlId** permitem "focalizar" um controle, ou seja, posicionar nele o cursor dentro de uma janela de diálogo.

Sintaxe

FocusControl (<num>)
FocusControlId (<id>)

Onde:

<num> Identifica o controle a ser processado. Todos os controles recebem um número seqüencial, a partir de 01, na ordem de especificação dentro da Caixa de Diálogo. Se **<num>** for um número negativo, indica que a mensagem KILLFOCUS, enviada pelo Windows para o programa OpusWin, será ignorada. Veja exemplo a seguir.
<id> especifica o nome atribuído ao controle, pela opção **ID <expc>,** quando da definição do mesmo dentro de um DialogBox.
A OpusWin considera que a digitação de um campo do tipo EditText foi concluída quando o "foco" sai desse campo, ou seja quando o Windows envia para o programa OpusWin o aviso de ocorrência de KILLFOCUS. Em certas ocasiões, por exemplo quando foi associada uma função (via comando **valid)** a um controle EditText, e conveniente ignorar essas mensagens KILLFOCUS que o Windows gera, principalmente quando nada foi digitado no campo EditText. Por esta razão, quando o número do controle especificado em **<num>** for negativo, o programa OpusWin ignora o evento KILLFOCUS de um EditText.
No exemplo abaixo, quando o cursor sai (KillFocus) do campo "varcep", a função "varcep" será acionada. Se nada for digitado no campo *varcep* (por exemplo, passando-se o cursor para outro campo ou botão), será disparado pelo Windows o evento KILLFOCUS para o campo *varcep*. Este evento é recebido e interpretado pelo programa OpusWin. Como o campo *varcep* não está correto (pois nada foi digitado nessa campo), a mensagem "Cep inválido” seria emitida uma mensagem e o foco será posicionado em outro controle. Caso não fosse codificado o comando FocusControl (-4), o evento KILLFOCUS no campo *varcep* seria interpretado pelo programa OpusWin e a mensagem "Cep inválido" seria mostrada indefinidamente enquanto o campo CEP não fosse preenchido.

Exemplo

```
$nolib
prog
vartel = space(15)
varcep = space(10)
bt =1
function fval(l)
Dialog 00,00,79,23
      style WS_POPUP DS_MODALFRAME WS_CAPTION WS_SYSMENU
      ltext "Digite O Telefone:" NULL 05,05,25,01
      EditText NULL vartel 31,05,10,01
      ltext "Digite O Cep:"  NULL 05,07,25,01
      EditText NULL varcep    30,07,05,01   exec fval(varcep)
      defpushbutton "&OK"   bt 05,20,08,02
      PushButton    "&Sair" bt 20,20,08,02  IDOK
EndDialog
if bt = 2
  return
endif
return
*Funcao Para Validar Numero CEP
func fval
parameters varcep
FocusControl(-4)
if empty (varcep)
   m = MessageBox("Cep Invalido")
   FocusControl(6)
endif
return .t.
```

1.0.0 Funções GetFocusControl() e GetNextFocus()

As funções numéricas **GetFocusControl()** e **GetNextFocus()** retornam o número do controle, dentro de uma Janela de diálogo, para o qual o foco será transferido. Veja as observações no tópico a seguir.

1.0.0 Funções GetFocusControlId() e GetNextFocusId()

As funções cadeia **GetFocusControlId()** e **GetNextFocusId()** retornam o identificador (nome) do controle, dentro de uma Janela de diálogo, para o qual o foco será transferido.
A OpusWin considera que a digitação de um campo do tipo **EditText** foi concluída quando o "foco" sai desse campo, ou seja quando o Windows envia para o programa OpusWin o aviso de ocorrência de KILLFOCUS. Em certas ocasiões, por exemplo quando foi associada uma função (via comando **valid)** a um controle EditText, e conveniente ignorar essas mensagens KILLFOCUS que o Windows gera, principalmente quando nada foi digitado no campo EditText.

Exemplo

No exemplo abaixo, imaginemos que o usuário acionou o botão **Sair**. Neste caso, o foco saiu do **campo** <**varcep**> e a **função** <**varcep**> será acionada. A função associada ao botão Sair verifica que o NextFocus é efetivamente o botão de saída e termina a Janela de diálogo.
Caso não fosse utilizada a função GetNextFocus() , o evento KILLFOCUS no campo <**varcep**> seria interpretado pelo programa OpusWin e a mensagem "Cep inválido" seria mostrada indefinidamente enquanto o campo CEP não fosse preenchido.
Veja a seguir o código básico do exemplo:

```
$nolib
prog
vartel = space(15)
varcep = space(10)
bt =1
function fval(l)
Dialog 00,00,79,23
      style WS_POPUP DS_MODALFRAME WS_CAPTION WS_SYSMENU
      ltext "Digite O Telefone:" NULL 05,05,25,01
      EditText NULL vartel 31,05,10,01
      ltext "Digite O Cep:"  NULL 05,07,25,01
      EditText NULL varcep    30,07,05,01   valid fval(varcep)
      defpushbutton "&OK"   bt 05,20,08,02
      PushButton    "&Sair" bt 20,20,08,02  IDOK valid fval(varcep)
EndDialog
```

```
if bt = 2
  return
endif
return
*Funcao Para Validar Numero CEP
func fval
parameters varcep
n = GetFocusControl()              && ou n = Get NextFocus()
if n = 6
   Cancel Dialog                   && ou IDCANCEL
endif
if empty (varcep)
   m = MessageBox("Cep Invalido")
endif
return .t.
```

1.0.0 Funções GetControl e GetControlId

As funções **GetControl** e **GetControlId** permitem obter o número ou nome do controle que acionou uma função **valid** a ele associada.

Sintaxe

<varnum> = GetControl ( )
<varchar> = GetControlId ( )

Onde:
<varnum> identifica uma variável numérica onde será retornado o número relativo do controle que acionou a função **corrente** através da cláusula **valid.**
<varchar> identifica uma variável cadeia onde será retornado o nome controle que acionou a função **corrente** através da cláusula **valid.**
Geralmente, as funções **GetControl( )** e **GetControlId()** são utilizadas dentro de uma função (acionada por **valid**) para determinar qual foi o controle que acionou a função corrente. Desta forma, podemos associar a vários controles o mesmo bloco de procedimentos, diferenciando de acordo com o controle específico que acionou esse bloco de procedimentos.

Exemplo

A seguir apresentamos um exemplo que utiliza várias das opções de gerência dinâmica dos controles das Caixas de Diálogo, acima descritos.

1.0.0 Exemplos de comandos de controle de DialogBox

```
$nolib
prog
func fcontrol(l)
declare vetnom[6]=space(10)
vetnom[1]= "Ana"
vetnom[2]= "Beatriz"
vetnom[3]= "Carla"
vetnom[4]= "Denise"
vetnom[5]= "Eliane"
vetnom[6]= "Fábio"
varnom = "Ana"
varnum = 2
bt = 1
varnome= "EditText"
Dialog 00,00,40,16
  Font “Times New Roman” 10
 caption "Gerenciamento dinâmico da ListBox"
 style WS_OVERLAPPEDWINDOW
 ltext "ComboBox:" NULL   01,01,10,01
 ComboBox vetnom  varnom 01,02,10,06  WS_VSCROLL
 ltext "ListBox:"  NULL   21,01,10,01
 ListBox vetnom  varnum 21,02,10,06  WS_VSCROLL
 ltext "Nome:" NULL 01,10,5,01
 EditText NULL varnome 07,10,10,1
 DefPushButton "&Refresh"  bt 01,12,08,-12  Valid fcontrol(vetnom,bt)
 PushButton "&Enable"   NULL 11,12,08,-12 Valid fcontrol(vetnom,bt)
 PushButton "&Disable"  NULL 21,12,08,-12 Valid fcontrol(vetnom,bt)
 PushButton "&Move"     NULL 01,14,08,-12 Valid fcontrol(vetnom,bt)
 PushButton "&Delete"   NULL 11,14,08,-12 Valid fcontrol(vetnom,bt)
 PushButton "&Select"   NULL 21,14,08,-12 Valid fcontrol(vetnom,bt)
```

```
  PushButton "Sa&i"       NULL 31,14,08,-12 IDCANCEL
EndDialog
if bt = 6
  quit
endif
Return
func  fcontrol
parameters vetnom[],bt(n)
if bt = 1
  vetnom[6]="Pepe"
  RefreshControl(2)
  RefreshContent(vetnom) && Altera o ListBox  e ComboBox
elseif bt = 2
  EnableControl(5)
  EnableControl(6)
elseif bt = 3
  DisableControl(5)
  DisableControl(6)
elseif bt = 4
  MoveControl(5,14,08,05,01)
  MoveControl(6,14,09,10,01)
elseif bt = 5
  DeleteControl(5)
  DeleteControl(6)
else
  SelectControl(2,"Carla")
  SelectControl(4,"Denise")
endif
return(.t.)
```

Observação

Veja a tela produzida pelo exemplo acima:

1.0.0 Comandos InitControl e InitControlId

Os comandos **InitControl** e **InitControlId** servem para inicializar o limite máximo de uma Barra de progressão ou Histograma. Esta função recebe como parâmetros o número (**<num>**) ou nome (**<id>**) do controle ProgressBar ou Histogram assim como o limite superior desse controle ProgressBar ou Histogram

Sintaxe

```
InitControl <num>,<lim>)
InitControlId (<id>,<lim>)
```

Onde:

<num> identifica o número do controle do tipo ProgressBar ou Histogram, cujo limite superior deseja ser "resetado" ou inicializado.

<id> especifica nome do controle do tipo ProgressBar ou Histogram, cujo limite superior deseja ser "resetado" ou inicializado.

<lim> especifica o valor do limite superior de uma Barra de Progressão de um Histograma.

1.0.0 Funções CHEXW, CHEXMW e CHEXMEW

As funções lógicas **Chexw, Chexmw e Chexmew** permitem validar, mostrar e editar itens EditText dentro de uma janela de diálogo. Possuem as mesmas características e regras de utilização das funções Chex, Chexm e Chexme (da linguagem OPUS). Veja Manual da OPUS para maiores detalhes.

A função **Chexw( )** pode ser chamada para validar um campo do tipo **EditText**.

Sintaxe

CHEXW (<chave>,<exp>)

Exemplo

```
EditText NULL v1 1,1,10,2 valid chexw("NOME_p",v1)
```

A função **Chexmw( )** pode ser chamada para validar um campo do tipo **EditText**, colocando o valor de outro item num campo do tipo VarText.

Sintaxe

CHEXMW (<chave>,<exp>,<item>,<vartext>)

Exemplo

```
EditText NULL v1 1,1,10,2 valid chexmw("NOME_p",v1, "IDADE",v2)
VarText NULL v2  1,4,5,2
```

A função **Chexmew( )** pode ser chamada para validar um campo do tipo **EditText**, colocando o valor de outro item num campo do tipo **VarText**, editado conforme uma máscara especificada.

Sintaxe

CHEXMEW (<chave>,<exp>,<item>,<vartext>,<mask>)

Exemplo

```
EditText NULL v1 1,1,10,2 valid chexmew("NOME_p",v1, "IDADE",v2,"XXX.XX")
VarText NULL v2  1,4,5,2
```

1.0.0 Comandos PrintControl e PrintControlId

Os comandos PrintControl e PrintControlId servem para imprimir um arquivo visualizado através do controle **RichEdit**, especificando:

- O número (**<num>**) ou nome (**<id>**) do controle **RichEdit**
- A orientação e tamanho do papel a ser impresso:
  - **P** (Portrait) ou **L** (Landscape)
    - **A** (A4)
    - **L** (legal)
    - **T** (Letter)

Sintaxe

PrintControlId (<id>,<orien>)

Exemplo

```
…
Dialog …
…
 RichEdit NULL arq 5,5,40,20 ID("xxx")
…
PushButton "Imprime" NULL 5,26,7,2 valid(f1)
…
Func f1
PrintControlId ("xxx","PT")
return .t.
```

1 Gerência de Caixas de Mensagens na OpusWin

Uma Caixa de Mensagem (ou MessageBox) é uma janela que exibe uma mensagem e aguarda uma confirmação por parte do usuário.

Em geral, o propósito de uma Caixa de Mensagens é informar ao usuário que algum evento ocorreu. Porém, é muito freqüente utilizar Caixas de Mensagens para permitir a escolha entre várias alternativas. Por exemplo, uma forma comum de Caixa de Mensagens solicita ao usuário escolher entre **A**nular, **R**epetir ou **I**gnorar. Nos exemplos de Caixas de Diálogo, apresentamos algumas MessageBox.

1.0 Definição de Caixas de Mensagem

Na ***OpusWin***, as Caixas de Mensagens foram implementadas como funções, cuja sintaxe de codificação é o seguinte:

ret=MessageBox (<*exp1*> [,<*exp2*> [,<*exp3*> [,<*exp4*>]]])

Onde:

<***exp1***> especifica o texto da mensagem.

<***exp2***> especifica o título da Caixa de Mensagem. Se <exp2> for uma cadeia vazia, a Caixa de Mensagem fica sem título.

<***exp3***> indica o tipo de providência a ser tomada pelo usuário e poderá ser uma das seguintes opções:

opção	*significado*
"O"	OK
"OC"	OKCANCEL
"YN"	YESNO
"YNC"	YESNOCANCEL
"RC"	RETRYCANCEL
"ARI"	ABORTRETRYIGNORE

<***exp4***> indica o tipo de ícone a ser mostrado na MessageBox, podendo ser uma das opções:

opção	*tipo de ícone*
"E"	EXCLAMATION
"I"	INFORMATION
"H"	HAND
"S"	STOP
"Q"	QUESTION

<ret> é a variável de retorno da função **MessageBox,** podendo ser um dos seguintes valores:

retorna ...	*quando pressionado ...*
"O"	o botão OK
"C"	o botão CANCEL
"Y"	o botão YES
"N"	o botão NO
"R	o botão RETRY
"A"	o botão ABORT
"I"	o botão IGNORE

1.0 Controle de Caixas de Mensagem

As Caixas de mensagem podem ser controladas de diversas maneiras. Quando se deseja apresentar uma Caixa de Mensagem em primeiro plano, pode ser usada a opção **SET TOPMOST ON | OFF**.

Quando esta opção estiver ligada (ON), indica que as MessageBox subseqüentes terão o estilo TOPMOST (SYSTEMMODAL). Por exemplo:

```
Set topmost ON
R=MessageBox("Alooooooo ….")
```

1.0 Exemplos de Caixas de Mensagem

```
c=MessageBox("Texto da Mensagem","Título da Caixa de Mensagem","O","I")
```

Observação

Veja outros exemplos de Caixas de Mensagens nos programas apresentados junto com os comandos das Caixas de Diálogo.

1 Utilização de Folha de Propriedades

Folhas de propriedades (*"Property Sheets")* são janelas que permitem ao usuário visualizar e/ou modificar as características ou propriedades (daí o nome) de um ou mais conjuntos de itens relacionados. Geralmente, uma *folha de propriedades* contém uma coleção de fichas chamadas **páginas de propriedades** *("Property Pages")*, ou seja, fichas que são selecionadas através de suas respectivas guias (abas).

As folhas de propriedades são extensamente utilizadas quando há necessidade de agrupar, de maneira organizada, várias *Caixas de Diálogo.*

1.0 Definição de Folhas de Propriedades

A seguir, relacionamos os comandos e opções utilizados para definir *folhas* e *páginas* de propriedades.

Comando	Descrição do comando	Sintaxe
Caption	informa o título de toda a Folha de Propriedades (quando especificado antes da primeira página) ou o título de cada página (quando especificado dentro da Property Page). O título da Folha é mostrada na primeira linha, alinhado à esquerda. O título de cada página e exibido em sua própria guia.	**CAPTION <texto>**
PropSheet	inicia a definição de uma Property Sheet.	**PropSheet**
EndPropSheet	finaliza a definição de uma Property Sheet	**EndPropSheet**
PropPage	inicia a definição de uma Property Page, dentro de uma Property Sheet.	**PropPage <xi>,<yi>,<xl>,<yl>**
EndPropPage	finaliza a definição de uma Property Page dentro de uma Property Sheet	**EndPropPage**
Start	Especifica qual a página (PropPage) que deverá ficar em primeiro plano, na frente de todas as páginas da Property Sheet. O operando **<expn>** informa o número seqüencial, relativo às PropPages, conforme a ordem de especificação dentro do programa, a partir de 0 (zero).	**START <expn>**
PropButton( )	Serve para saber qual o botão da Property Sheet que foi acionado pelo usuário. Esta função retorna o valor **"1"** quando o usuário acionou o **botão OK** ou valor **"0"** quando o usuário acionou o **botão CANCEL**. Veja exemplo mais adiante.	**if=PropButton( ) = n** onde: **n** é **"1" (OK)** ou **"0" (CANCEL)**
GetPage( )	Serve para saber qual a página (PropertyPage) dentro de uma PropertySheet que teve um controle acionado pelo usuário. Esta função numérica retorna o número, a partir de 1, da página que acionou (através da opção **valid,** por exemplo) a função que está sendo executada no momento.	**Npage = GetPage()**

1.0 Observações sobre as de Folhas de Propriedades

Em cada Folha de propriedades, são incluídos, de forma automática, os botões **OK**, **CANCEL** e, dependendo do aplicativo, o botão **APPLY.** O botão **OK serve para aplicar as modificações** pendentes e **fechar a janela** da Property Sheet. O botão **Apply** serve para a**plicar as modificações** pendentes, **deixando aberta a janela** da Property Sheet. O botão **cancel** serve para descartar **as modificações** pendentes e **fechar a janela** da Property Sheet. Não podem ser canceladas ou desfeitas alterações previamente aplicadas.
Não são permitidos os comandos **[Def]PushButton** na definição da Folha de Propriedades. Porém, esses botões podem ser utilizados dentro das páginas de propriedades que são verdadeiras Caixas de Diálogo.
Após o comando EndPropPage, pode ser utilizada a função **PropButton( )** para gerenciar corretamente o fluxo dos procedimentos de acordo com o botão acionado (OK ou CANCEL).

1.0 Exemplo de Folhas de Propriedades

A seguir, vejamos um exemplo utilizando Property Sheets.

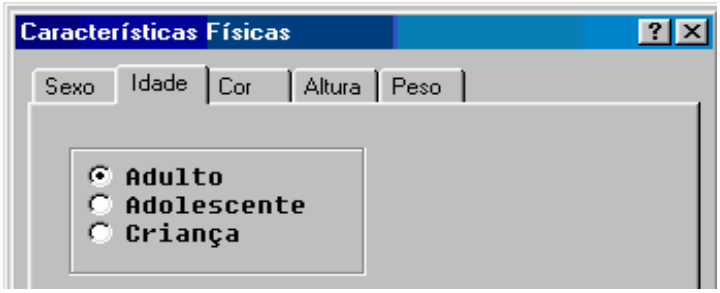
```
$nolib
prog
func fverify(l)
sex=1
idade=1
cor=1
altu=1
peso=1
propsheet
  caption "Características Físicas "
```

```
  font "Arial Narrow"
  verify fverify (sex,idade,cor,altu,peso)
  Start 0
  proppage 00,00,16,08
    caption "Sexo"
    style WS_POPUP WS_CAPTION  DS_CENTER
    GroupBox "" sex 02,01,18,04
    RadioButton "&Homem"  NULL 03,02,16,01
    RadioButton "&Mulher" NULL 03,03,16,01
  endproppage
  proppage 00,00,16,08
    caption "Idade"
    style WS_POPUP  WS_CAPTION DS_CENTER
    GroupBox "" idade 02,01,18,05
    RadioButton "Adulto     "  NULL 03,02,16,01
    RadioButton "Adolescente"  NULL 03,03,16,01
    RadioButton "Criança    "  NULL 03,04,16,01
  endproppage
  proppage 00,00,16,08
    caption "Cor"
    style WS_POPUP WS_CAPTION  DS_CENTER
    GroupBox "" cor 02,01,18,06
    RadioButton "Branco"  NULL 03,02,16,01
    RadioButton "Preto "  NULL 03,03,16,01
    RadioButton "Pardo "  NULL 03,04,16,01
    RadioButton "Outra "  NULL 03,05,16,01
  endproppage
  proppage 00,00,16,08
    caption  "Altura"
    style WS_POPUP WS_CAPTION  DS_CENTER
    GroupBox "" altu 02,01,18,05
    RadioButton  "Alto " NULL 03,02,16,01
    RadioButton "Médio"  NULL 03,03,16,01
    RadioButton "Baixo"  NULL 03,04,16,01
  endproppage
  proppage 00,00,16,08
    caption "Peso"
    style WS_POPUP   DS_CENTER
    GroupBox "" peso 02,01,18,05
    RadioButton "Gordo" NULL 03,02,16,01
    RadioButton "Médio" NULL 03,03,16,01
    RadioButton "Magro" NULL 03,04,16,01
  endproppage
endpropsheet
if PropButton() = 0    && cancel
  return
endif
return

func fverify
parameters sex(n),idade(n),cor(n),altu(n),peso(n)
mens="sexo="+str(sex)+chr(10)+;
    "idade="+str(idade)+chr(10)+;
    "cor="+str(cor)+chr(10)+;
    "altu="+str(altu)+chr(10)+;
    "peso="+str(peso)
ret=MessageBox(mens,"Rotina chamada por Verify...","O","I")
return .f.
```

Observação

Veja como aparece uma das páginas (recortada) da Folha de Propriedades definida no programa acima:



1 Gerência de Bancos de Dados na OpusWin

Agrupamos aqui alguns comandos e funções relacionados com o processamento de Bancos de Dados.

1.0 Comando DataTable

O comando **DataTable** da ***OpusWin*** permite **consultar e modificar** os arquivos de um Banco de Dados.

1.0.0 Definição de DataTable

O recurso **DataTable** utiliza matrizes para representar os registros (linhas) e os itens (colunas). Podem ser selecionados os registros (linhas) e os itens (colunas) que desejam ser exibidos. As teclas **INSERT** e **DELETE** podem ser utilizadas para **INCLUIR** e **EXCLUIR** registros.

Sintaxe

```
query <explog>
DataTable  [itens(<item1>, <item2> ..., <itemn>)]
    [window(<xi>,<yi>,<xl>,<yl>)]
    [headers (<expc1>, <expc2>, ..., <expcn>)]
    [query <expc>]
    [valid ([<func>(<par1>, ..., <parn>)], ..., [<func>(<par1>, ..., <parn>)]]
           [NULL                          ], ..., [NULL                          ]
```

Onde:

query <explog>

O comando **query <explog>,** antes do comando **DataTable**, indica uma **expressão lógica,** sendo necessário para indicar os registros a serem exibidos. Caso se deseje exibir todos os registros, deve ser especificado **query .t.**

itens(<item1>, ... ,<itemn>)

A lista **itens**(<***item1***>, ... ,<***itemn***>) é opcional. Não sendo especificada a opção **itens**, serão exibidos todos os itens de cada registro, conforme foram definidos no esquema do Banco de Dados. Para arquivos de bancos OpenBASE, a lista de itens é opcional. Em se tratando de arquivos externos, a lista de itens é obrigatória.

window(<*xi*>,<*yi*>,<*xl*>,<*yl*>)

A opção **window** especifica, opcionalmente, as coordenadas da matriz. O padrão assumido é **window(0,0,79,23)**.

headers (<expc1>, ..., <expcn>)

A opção **headers** especifica os títulos a serem exibidos como cabeçalhos nas colunas dos itens. A especificação de **headers** é opcional. Quando não for especificado, serão exibidos os nomes dos itens constantes no esquema do banco. Quando especificado, **<expc1>, ..., <expcn>** deve ser uma lista de literais, contendo um cabeçalho para cada item do registro.

query <expc>

A opção **query <expc>,** dentro do comando **DataTable**, é uma expressão relacional e serve para selecionar os registros do arquivo, utilizando a mesma sintaxe do comando **SELECIONE** do utilitário "**GERAL**", hoje mais conhecido como **Query**. Este comando Query foi implementado por razões de compatibilidade e facilidade de migração da OPUS e do antigo utilitário GERAL.

valid ([<func>(<par1>, ..., <parn>)], ..., [<func>(<par1>, ..., <parn>)]
[NULL   ], ..., [NULL   ]

A opção **valid** especifica as funções lógicas a serem chamadas em caso de alteração nas células da **DataTable.** Esta cláusula **valida** a entrada de dados em cada célula da matriz. Cada uma das funções se referem, respectivamente, a cada um dos itens declarados na cláusula **itens.** Caso não se deseje validar um determinado item, deve-se especificar a cláusula **NULL**, dentro da opção valid, na respectiva posição. As funções referenciadas no valid são lógicas e, caso retornem falso, não é gravado no arquivo o valor digitado dentro da célula.

O controle **GRID** (dentro de Caixas de Diálogo) permite efetuar funções similares ao DataTable, porém oferece maiores recursos. Veja maiores detalhes neste manual.

1.0.0 Exemplos de DataTable

Veja a seguir um pequeno exemplo que mostra como utilizar o recurso **DataTable**.

```
$nolib
prog
Mens=""
func fnome(l)
func fida(l)
SetWindowText("Exemplo do DataTable")
Showtext(01,00,"Cadastro de Pessoas:","Arial Black",1,"")
Showtext(01,08,"Tecle Esc para sair","Arial Black",1,"")
DataBase .\EXEMPLO 1 a 2
use PESSOA
query .t.
DataTable itens(NOME_P,IDADE) window(01,01,24,06);
        headers("Nome","Idade");
        valid(fnome(NOME_P),fida(IDADE))
return
func fnome
parameters wnome(c)
Mens='Nome="'+NOME_P + '"'
m = MessageBox(Mens,"","O","I")
return(.f.)
func fida
parameters idade(n)
Mens='Idade="'+str(IDADE) + '"'
m = MessageBox(Mens,"","O","I")
return(.f.)
```

Observação

O comando DataTable também pode ser utilizado com arquivos externos, como mostrado neste exemplo:

```
use *
abc r len(20)
cod    n3
nome u17
enduse
DataTable itens(cod,nome)
```

1.0 Comando DataDialog

O comando **DataDialog** oferece uma ferramenta poderosa e cheia de recursos, pois constrói, de forma automática, uma Caixa de Diálogo, uma Barra de navegação e uma lógica completa para processamento de arquivos do Banco de Dados.

1.0.0 Definição de DataDialog

O comando **DataDialog** gera uma Caixa de Diálogo com os seguintes controles:

- Os itens do arquivo conforme especificado em **itens(<*item1*>, <*item2*> ..., <*itemn*)**
- A Barra de Navegação que permite executar as tarefas normais de processamento de dados, quais sejam, ler, excluir, incluir, alterar, pesquisar e navegar.
- O botão OK

Sintaxe do comando DataDialog:

```
DataBase <banco> <segurança> <nivel> <modo>
[use <arquivo>]
DataDialog [file(<arquivo>)] [itens(<item1>, <item2> ..., <itemn>)]
    [window(<xi>,<yi>,<xl>,<yl>)]
    [headers (<cab1>, <cab2>, ..., <cabn>)]
```

Onde:

DataBase <Banco>

Informa o Banco cujos arquivos vão ser processados.

use <arquivo>

Prepara o arquivo <arquivo> para processamento.

file(<arquivo>)

Informa o nome de arquivo a ser processado pelo comando **DataDialog**.

itens(<item1>, ... ,<itemn>)

A lista **itens**(<***item1***>, ... ,<***itemn***>) é opcional. Não sendo especificada a opção **itens**, serão processados todos os itens de cada registro, conforme definido no esquema do Banco de Dados.

window(<*xi*>,<*yi*>,<*xl*>,<*yl*>)

A opção **window** especifica, opcionalmente, as coordenadas e tamanhos da Caixa de Diálogo onde serão colocados todos os controles necessários. Não havendo especificação deste parâmetro, a OpusWin constrói uma Dialog Box na posição e tamanhos adequados.

headers (<expc1>, ..., <expcn>)

A opção **headers** especifica os títulos a serem associados aos itens relacionados na opção **itens**. A especificação de **headers** é opcional. Quando não for especificado, serão exibidos os nomes dos itens do arquivo constantes no esquema do Banco de Dados.

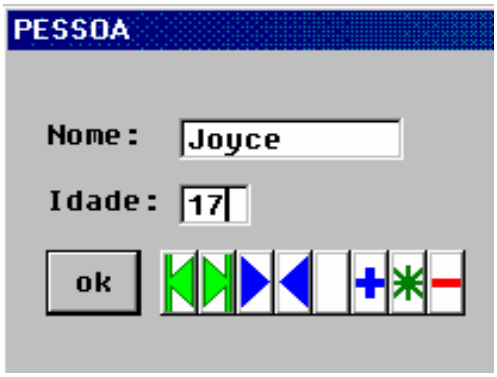
1.0.0 Exemplo de DataDialog

Veja, a seguir, como é fácil codificar um programa (completo !) que utiliza **DataDialog.**

```
prog
DataBase .\exemplo 1 a 2
use PESSOA
DataDialog file("PESSOA") itens(NOME_P,IDADE) headers("Nome:","Idade:")
```

Observação

Na figura a seguir (recortada), mostramos os campos e a Barra de Navegação gerados pelo programa acima.



Na figura acima, produzida pelo programa acima (que nós recortamos) são exibidos os campos do arquivo PESSOA, do Banco de Dados EXEMPLO.
Na parte de baixo da figura, é mostrada um Barra de Navegação com as seguintes funções associadas aos ícones, pela ordem de apresentação, da esquerda para a direita: **Primeiro Registro, Último Registro, Próximo Registro, Registro Anterior, Limpar campos na tela, Incluir Registro, Alterar Registro e Excluir Registro.**

1 Diálogos e comandos de uso geral

Agrupamos aqui uma variedade de comandos e diálogos a serem utilizados com freqüência nos programas OpusWin.

1.0 Diálogos GetOpenFileName / GetSaveFileName

A função **GetOpenFileName** constrói uma Caixa de Diálogo e permite que o usuário selecione a unidade, diretório e nome de um arquivo, ou conjunto de arquivos, a serem abertos. A função **GetSaveFileName** constrói uma Caixa de Diálogo e permite que o usuário informe a unidade, diretório e nome de um arquivo a ser gravado.

Ambas as funções devolvem o nome completo de um arquivo, inclusive o seu percurso completo, ou seja, unidade, pasta e sub-pasta onde ele se encontra ou onde ele vai ser gravado esse arquivo.

Sintaxe

```
GetOpenFileName [(<expc1>, <expc2>, <expc3>)[,<expc4>]]
GetSaveFileName [(<expc1>, <expc2>, <expc3>)[,<expc4>]]
```

Onde:

<expc1>

Informa, opcionalmente, um nome de arquivo

<expc2>

Informa, opcionalmente, o nome do diretório onde reside o arquivo ou onde o mesmo vai ser gravado. Sendo <**expc2**> uma cadeia vazia (“”), assume-se que foi informado o diretório corrente.

<expc3>

Informa, a maneira de **filtro**, os tipos de arquivos a **abrir** ou **salvar**. Esta informação é especificada em duas partes, aos pares, separadas por “/” (barra): a primeira parte contém a descrição e a segunda os tipos de arquivos. Podem ser especificados vários tipos de arquivos, separados por “;” (ponto e vírgula). Podem ser especificados vários pares, separados por “/” (barra).

<expc4>

Especifica, opcionalmente, o texto a ser mostrado no título da Caixa de Diálogo.

Exemplo

```
arq1=GetOpenFileName (“*.txt”,””,”arquivos texto/*.txt”)
arq1=GetOpenFileName (“*.txt”,””,”arquivos texto/*.txt”,”Selecione o nome do arquivo …”)
```

Nos exemplo acima:

- Vai ser apresentada uma caixa de diálogo, cujo título é: **Selecione o nome do arquivo …** , incluindo uma lista de todos os arquivos terminados em **.txt** residentes no diretório corrente.
- O usuário pode selecionar um dos arquivos da lista.
- A variável **arq1** conterá o nome, inclusive com o percurso completo, do arquivo selecionado pelo usuário.

Exemplo

```
arq2=GetSaveFileName (“p.f”,”\dir\trab”,”fontes/*.f;*.fon/textos/*.txt”)
```

No exemplo acima:

- Vai ser apresentada uma caixa de diálogo com o arquivo **p.f**, no diretório **\dir\trab**.
- O usuário pode selecionar o arquivo.
- A variável **arq2** conterá o nome, inclusive com o percurso completo, do arquivo selecionado pelo usuário.

Observações

Ao final do tópico “Caixas de Diálogo de uso geral”, apresentamos um exemplo com a utilização destes recursos.

1.0 Diálogo ChooseFont

O comando **ChooseFont** constrói uma Caixa de Diálogo e permite que o usuário selecione o **nome** de uma fonte e seu respectivo **tamanho**.

Sintaxe

ChooseFont (<varc>, <tam>)
Onde:

<varc>

Especifica a variável cadeia onde o comando ChooseFont vai retornar o nome da Fonte selecionado pelo usuário.

<varn>

Especifica a variável numérica onde o comando ChooseFont vai retornar o tamanho da Fonte escolhido pelo usuário.

1.0 Diálogo PrintDlg

A função lógica **PrintDlg()** apresenta uma Caixa de Diálogo que permite configurar a impressão e as características da Impressora.

Sintaxe

```
f=PrintDlg()
```

Dentro dos programas ***OpusWin***, pode ser utilizado o comando **SET PRINTER TO <expc>** para informar a impressora a ser subseqüentemente utilizada. O nome especificado em **<expc>** é o mesmo que aparece no diálogo **PrintDlg()**, bastando informar as primeiras n posições de forma a evitar qualquer ambigüidade.

1.0 Exemplos de comandos e diálogos de uso geral

Os exemplos a seguir mostram a utilização de alguns **comandos e diálogos de uso geral**.

1.0.0 Exemplo 01

```
$nolib
prog
public arquivo(255),bt(n)
arquivo=""
menu sample03
 pad e1 prompt "&File" popup p1
  bar 01 prompt "&New";
   message "Edit new file ...";
   do proc1
  bar 02 prompt "&Open...";
   message "Open file for Edit...";
   do proc1
  bar 03 separator
  bar 04 prompt "&Showtext...";
   message "Show a sample text message...";
   do proc1
  bar 05 prompt "&Progress Bar...";
   message "Show a sample Progress bar...";
   do proc1
  bar 06 separator
  bar 07 prompt "E&xit";
   message "End of program...";
   do proc1
 pad e2 prompt "&Help" popup p2
  bar 01 prompt "About OpusWin...";
   do proc2
EndMenu
*
SetWindowText "OpusWin - SAMPLE03.F"
SetWindowIcon "world.ico"
set border on
set color to "GR+/B,B/GR+"

Activate menu sample03
PlaySound ("musica.wav")

ToolBar
 style CCS_TOP TBSTYLE_ALTDRAG TBSTYLE_TOOLTIPS
 bitmap "toolbar.bmp"   1
 Buttons 1 TBSTYLE_BUTTON Tip "New File ..." do proc1
 Buttons 2 TBSTYLE_BUTTON Tip "Open File ..." do proc1
 Buttons SEPARATOR TBSTYLE_SEP
 Buttons 4 TBSTYLE_BUTTON Tip "ShowText ..." do proc1
 Buttons 5 TBSTYLE_BUTTON Tip "Progress Bar..." do proc1
 Buttons SEPARATOR TBSTYLE_SEP
 Buttons 7 TBSTYLE_BUTTON Tip "Exit ..." do proc1
 Buttons SEPARATOR TBSTYLE_SEP
 Buttons 9 TBSTYLE_BUTTON Tip "About OpusWin ..." do proc2
EndToolBar

proc proc1
public arquivo(255),bt(n)
if istoolbar() = .t.
 opt=button()
else
 opt=bar()
endif

do case
 editor="notepad "
 case opt=1     && new
  Run editor SW_NORMAL

 case opt=2     && Open
  bt=2         && cancel
  do open_file
  if bt=1       && ok: abrir arquivo
   if len(alltrim(arquivo)) > 0
```

```
      cmd_run = editor + arquivo
      Run cmd_run
    else
      Run editor
    endif
  elseif bt=3    && browse
    arquivo=GetOpenFileName("*.f","","Fonte/*.f;*.fon/Texto/*.txt/Todos/*.*")
    bt=2         && cancel
    do open_file
    if bt=1      && ok: abrir arquivo
      cmd_run = editor + arquivo
      Run cmd_run
    endif
  endif

 case opt=4
  Showtext (02,10)
  texto='Isto e uma linha de texto exibida pelo comando "ShowText"'
  Showtext (02,10,texto,"Arial",1,"B/G")

 case opt=5
  Showtext (02,10)
  pbcreate progress_bar 02,12,76,2     && chars
  pbmessage progress_bar "SETRANGE" 1 20
  pbmessage progress_bar "SETSTEP" 1
  for i=1 to 19
    pbmessage progress_bar "STEPIT"
    sleep 1
    temp=i*100/20
    texto=str(i,2)+" --> "+str(temp,2)+" completo ..."
    Showtext (02,10,texto,"arial",1,"B/G")
  next
  pbmessage progress_bar "DESTROY"

 case opt=7
  Deactivate menu sample03
  quit
endcase
return

proc proc2
if istoolbar() = .t.
 opt=button()
else
 opt=bar()
endif

if opt = 1 .or. opt=9
 xtext="OpusWin - OPUS for Windows"
 botao=1
 Dialog 20,05,36,08
  style WS_POPUP DS_MODALFRAME
  ctext xtext NULL 02,02,32,01
  ctext "Copyright (c) Tecnocoop Sistemas" NULL 02,03,32,01
  ctext "1997,  Rio de Janeiro  -  BRAZIL" NULL 02,04,32,01
  DefPushButton "&Ok" botao 13,06,08,02 IDOK
  icon "world.ico" 1 15,00,0,0
 EndDialog
endif

proc open_file
public arquivo(255),bt(n)
Dialog 10,05,61,06
 caption "OpusWin - OPUS for Windows"
 style WS_POPUP WS_CAPTION WS_SYSMENU
 help "C:\openbhlp\wopushlp.hlp"
 ltext "File:" NULL 01,01,05,01 CONTEXT(1)
 EditText NULL arquivo 06,01,52,01 IDOK CONTEXT(1)
 DefPushButton "&Edit" bt 06,04,08,02 IDOK CONTEXT(1)
 PushButton "E&xit" NULL 16,04,08,02 IDOK
```

```
  PushButton "&Browse" NULL 26,04,08,02 IDOK CONTEXT(1)
EndDialog
return
```

1.0.0 Exemplo 02

O exemplo apresentado a seguir utiliza as Caixas de Diálogo de uso geral.

```
$nolib
prog
public arquivo(255),bt(n)
arquivo=""
menu Dialogos
 pad e1 prompt "&File" popup p1
  bar 01 prompt "&New";
   message "Edit new file ...";
   do proc1
  bar 02 prompt "&Open...";
   message "Open file for Edit...";
   do proc1
  bar 03 prompt "&Font";
   message "End of program...";
   do proc1
  bar 04 prompt "E&xit";
   message "End of program...";
   do proc1
EndMenu
SetWindowText "OpusWin - Dialogos de uso geral"
SetWindowIcon "world.ico"
Activate Menu Dialogos
PlaySound ("musica.wav")
end
proc proc1
public arquivo(255),bt(n)
opt=bar()
varc=""
varn=0
do case
 editor="notepad "
 case opt=1     && new
  Run editor SW_NORMAL
 case opt=2     && Open
  bt=2        && cancel
  do open_file
  if bt=1      && ok: abrir arquivo
   if len(alltrim(arquivo)) > 0
    cmd_run = editor + arquivo
    Run cmd_run SW_NORMAL
   else
    Run editor SW_NORMAL
   endif
  elseif bt=3   && browse
   arquivo=GetOpenFileName("*.f","","Fonte/*.f;*.fon/Texto/*.txt/Todos/*.*")
   bt=2        && cancel
   do open_file
   if bt=1      && ok: abrir arquivo
    cmd_run = editor + arquivo
    Run cmd_run SW_NORMAL
   endif
  endif
 case opt=3
  ChooseFont (varc,varn)
  mens='Fonte escolhida="'+varc+'"' + chr(10) + ;
      'Tamanho da Fonte="'+str(varn)+'"'
  ret=MessageBox(mens,"Apenas um aviso ...","O","I")
 case opt=4
  Deactivate menu Dialogos
  quit
endcase
return
proc open_file
public arquivo(255),bt(n)
```

```
Dialog 01,01,40,06
  caption "OpusWin - OPUS for Windows"
  style WS_POPUP DS_MODALFRAME WS_CAPTION WS_SYSMENU
  help "..\help\ctxhlp.hlp"
  ltext "File:" NULL 01,01,05,01 CONTEXT(1)
  EditText NULL arquivo 06,01,32,01 ES_AUTOHSCROLL IDOK CONTEXT(1)
  DefPushButton "&Edit" bt 06,04,08,-12 IDOK CONTEXT(1)
  PushButton "E&xit" NULL 16,04,08,-12 IDOK
  PushButton "&Browse" NULL 26,04,08,-12 IDOK CONTEXT(1)
EndDialog
return
```

1 Gerência de impressão na OpusWin

Agrupamos aqui os comandos e funções relacionados com as tarefas de impressão em aplicativos OpusWin. Consulte as informações do produto **OPUSRel** para obter maiores detalhes sobre os recursos de impressão da OpusWin.

1.0 SET Print ON / OFF

A opção **SET Printer ON** especifica que os comandos a seguir são direcionados para a impressora, indicando, opcionalmente, a orientação da impressão.

Sintaxe

SET PRINT ON | OFF [Landscape | Portrait]

Onde:

Landscape indica impressão horizontal da página e **Portrait** indica impressão no sentido vertical.

1.0 SET Printer TO

A opção **SET Printer TO** especifica o destino dos documentos a serem impressos pelo programa OpusWin e permite informar:

1. O **nome da impressora** para a qual serão enviados os documentos a imprimir:

Sintaxe

SET Printer to "<printer-name>"

Onde:

<printer-name> é o nome da impressora conforme exibido pela função PrintDlg(). São comparadas apenas os primeiros n caracteres, onde n é o tamanho da expressão <printer-name>.

Exemplo

SET Printer to "Canon"

Onde:

"Canon" é o nome da impressora que foi obtido previamente através da função PrintDlg().

2. O **nome de um arquivo** onde será armazenado o relatório para consulta e/ou impressão posteriores. Esse arquivo pode existir ou não.

Sintaxe

SET Printer to "file:<file-name>"

Onde:

<file-name> é o nome de um arquivo novo ou já existente, precedido pela constante **"file:".**

Exemplo

SET Printer to "file:relat01.txt".

1.0 SET Orientation TO

A opção **SET Orientation TO** permite configurar a orientação da página a ser impressa.

Sintaxe

SET Orientation TO <expc>

Onde:

<expc> pode ser "**L**" (Landscape) para impressão horizontal da página ou "**P**" (Portrait) para indicar impressão no sentido vertical.

1.0 Função PrintDev( )

Dentro dos programas ***OpusWin***, pode ser utilizada a função cadeia **PrintDev( )** para obter o nome da impressora padrão a ser utilização subseqüentemente. A impressora padrão é, usualmente, informada ou modificada através da opção **Printers** do **Control Panel** ou **My Computer**.
Por exemplo, o comando **default_printer = PrintDev( )** colocará na variável **default_printer** o nome da impressora padrão associada com seu computador.
Observe que esta função fornece também os mesmos serviços do comando SET PRINTER TO, explicado anteriormente.

1.0 Exemplo completo de impressão em OpusWin

```
prog
bt1 =1
func fimp(l)
Dialog 10,6,40,5 chars
 style WS_POPUP DS_MODALFRAME WS_CAPTION WS_SYSMENU
 caption "Escolha a Opção de Saída"
 defpushbutton "Impressora" bt1 2,2,12,2;
  valid fimp (bt1);
  font("Arial Negrito",-08)
 PushButton "Video" NULL 17,2,10,2;
  valid fimp (bt1);
  font("Arial Negrito",-08)
 PushButton "Sair" NULL 29,2,10,2 IDOK ;
    font("Arial Negrito",-08)
EndDialog
if bt1= 3
 return
endif
return
*************************************************************************
func fimp
parameters bt1(n)
mess=MessageBox("Confirma impressao?","Tabela de Contratados:","YN","Q")
if mess = "N"
   return .t.
endif
if bt1 = 1    && Foi Feita a Opção na Dialog de se direcionar a saída para Impressora
  set print on  PORTRAIT
else          && Foi Feita a opção na Dialog de se direcionar a saída para Arquivo
  set printer to "file:abc.txt"
endif
set print on

  if  bt1 = 1
     xlin = -6631 && Medida em pixels( 1 char = -65 pixels) que ultrapassa a linha 66
  else
     xlin = 51
  endif
  xpag= 0

  database Exemplo 1 a 2
  select a
  use Arq01
  locate
  do while .not. eof()
   if bt1 = 1 && Foi Feita a Opcao na Dialog de se Direcionar a Saida Para Impressora
    if xlin < -6630 && Medida em pixels( 1 char = -65 pixels) que equivale a linha 66
      if xpag > 0
        eject  && Necessário Para se Mudar de Pagina(Quebra de Pagina)
      endif
      ************
      do CABECALHO with bt1,xpag
      ************
      xlin = -715   **Equivale a linha  11(-65 X 11) onde será iniciada a impressão após o cabeçalho
    endif
   else    && Impressão em tele
    if xlin > 50
      ************
```

```
        do CABECALHO with bt1,xpag
        ************
        xlin = 06
      endif
     endif
     wmat=    ConMatric
     wnome=   ConNome
     if bt1 = 1   && Saida na Impressora
        Showtext(00,xlin,wmat,"Arial",-60,"B/R")
        Showtext(08,xlin,wnome,"Arial",-60,"B/R")
     else         && Saida em Arquivo
        @xlin,01 say wmat
        @xlin,13 say wnome
     endif
      if bt1 = 1   && Saida na Impressora
        xlin = xlin - 65  && equivale a acan‡o de linha ( xlin = xlin + 1)
      else         && Saida em Arquivo
        ++ xlin
      endif
      select a
      continue
   enddo
set print off
if bt1 = 2
 run "c:\arquiv~1\acessó~1\wordpad.exe c:\OpusWin\ibge\abc.txt"
endif
return .t.
**********************
static proc CABECALHO
**********************
parameters bt1(n),xpag(n)
vardata = date()
varhora = time()
++ xpag
wpag= str(xpag,3)
if bt1 = 1
  Showtext(01,01,"EMPRESA XPTO","Arial",-60,"B/R")
  Showtext(25,01,"SISTEMA DE ADMINISTRAÇÃO DE PESSOAL","Arial",-60,"B/R")
  Showtext(63,01,vardata,"Arial",-60,"B/R")
  Showtext(70,01,varhora,"Arial",-60,"B/R")
  Showtext(01,02,"EXEMPLO","Arial",-60,"B/R")
  Showtext(20,02,"RELATóRIO DE FUNCIONARIOS","Courier",-70,"B/R")
  Showtext(63,02,"PAG:","Arial",-60,"B/R")
  Showtext(68,02,wpag,"Arial",-60,"B/R")
  Showtext(00,05,"Matríula","Arial",-60,"B/R")
  Showtext(08,05,"Nome","Arial",-60,"B/R")
else
  @01,01 say "EMPRESA XPTO"
  @01,20 say "SISTEMA DE ADMINISTRAÇÃO DE PESSOAL"
  @01,63 say vardata
  @01,74 say varhora   + chr(13) + chr(10)
  @02,01 say "EXEMPLO"
  @02,20 say "RELATÓRIO DE FUNCIONARIOS"
  @02,63 say "PAG:"
  @02,74 say wpag  + chr(13) + chr(10)
  @05,01 say "Matrícula"
  @05,13 say "Nome"
endif
return
```

1 Controle do Registry na OpusWin

Apresentamos, neste capítulo, os **comandos e as funções** relacionadas com o **Registry**, base de dados unificada onde são organizadas, de forma hierárquica e segura, as informações relativas aos componentes de software e hardware do sistema e dos aplicativos, assim como os perfis dos usuários e o ambiente de segurança.
Na primeira parte deste manual foram abordados alguns tópicos básicos em relação ao assunto deste capítulo. Volte nessa parte, se necessário, para esclarecer alguns conceitos básicos como, por exemplo, chave, subchave, etc ...

1.0 RegInitKey

Esta função numérica retorna o “handle” de uma determinada “chave” do Registry.

Sintaxe

<hnd>=RegInitKey (<chave>)

onde:

<hnd>

Especifica o “handle” retornado pela função RegInitKey(), associado à chave **<chave>**.

<chave>

Especifica uma determinada chave, dentro da estrutura do Registry, que poderá ser uma das seguintes:

- **“HKEY_CURRENT_USER”**
- **”HKEY_LOCAL_MACHINE”**
- **”HKEY_USERS”**
- **“HKEY_CURRENT_CONFIG”**
- **”HKEY_CLASSES_ROOT”**

1.0 RegOpenKey

Esta função numérica retorna o “handle” de uma determinada sub-chave, hierarquicamente dependente da chave associada a uma chave do Registry que foi obtida através da função **RegInitKey()**.

Sintaxe

<hnd>=RegOpenKey (<hnd1>,<subchave>)

Onde:

<hnd>

Especificada o “handle” retornado pela função RegOpenKey(), associado a **<sub-chave>**.

<hnd1>

Especificada o “handle” associado à chave imediatamente superior a esta **<sub-chave>**.

<sub-chave>

Especifica o nome de uma determinada sub-chave, hierarquicamente dependente da chave associada ao handle <hnd1>.

1.0 RegCreateKey

Esta função numérica cria a sub-chave <sub-chave>, dentro da chave associada com o “handle”<*hnd1*>, e retorna em <*hnd*>, o “handle” da <sub-chave> criada.

Sintaxe

hnd=RegCreateKey (<hnd1>,<subchave>)

1.0 RegSetKey

Esta função lógica inclui, na chave associada ao “handle” **<*hnd*>**, o nome <nome> e o valor <valor>.

Sintaxe

f=RegSetKey (<hnd>,<nome>,<valor>)

1.0 RegQueryKey

Esta função cadeia retorna o valor associado ao nome<nome>, dentro da chave ou sub-chave associadas ao “handle” **<*hnd*>**.

Sintaxe

c=RegQueryKey (<hnd>,<nome>)

1.0 RegCloseKey

Esta função lógica fecha a chave ou sub-chave associadas ao “handle” <hnd>.

Sintaxe

f=RegCloseKey (<hnd>)

1.0 PutReg [U]ser e PutRegT [U]ser

Estas funções lógicas incluem uma "subkey" abaixo de uma determinada "Key" do Registry, cujo nome é: **HKEY_CURRENT_USER\Software\Tecnocoop Sistemas\OpenBASE\Vn.m**.

Sintaxe

f=PutReg ("[U]ser",<chave>, <nome>, <valor>)
f=PutRegT ("[U]ser",<chave>,<nome>,<valor>,<taman>)

Onde:

"[U]ser"

Especifica que as funções **Putreg / PutRegT** deverão processar a "Key" **HKEY_CURRENT_USER\Software\Tecnocoop Sistemas\OpenBASE\Vn.m.**

<chave>

Especifica a subkey a ser incluída, possuindo os seguintes atributos (valores):

- o **nome** da "subkey" é **<nome>**
- e o **conteúdo** (valor) da "subkey" é **<valor>**

<taman>

Na função PutRegT o parâmetro **<nome>** não é uma cadeia, por isso especifica-se o tamanho de **<valor>**.

Exemplo

```
result=PutReg ("U","Código do usuário", "Código", "12345")
if result = .f. then ...
```

1.0 PutReg [M]achine e PutRegT [M]achine

Estas funções lógicas incluem uma "subkey" abaixo de uma determinada "Key" do Registry, cujo nome é: **HKEY_LOCAL_MACHINE\Software\Tecnocoop Sistemas\OpenBASE\Vn.m**.

Sintaxe

f=PutReg ("[M]achine",<chave>, <nome>, <valor>)
f=PutRegT ("[M]achine",<chave>,<nome>,<valor>,<taman>)

Onde:

"[M]achine"

Especifica que as funções **Putreg / PutRegT**deverão processar a "Key" **HKEY_LOCAL_MACHINE\Software\Tecnocoop Sistemas\OpenBASE\Vn.m.**

<chave>

Especifica a subkey a ser incluída, que passará a ter os seguintes atributos (valores):

- o **nome** da "subkey" é **<nome>**
- e o **conteúdo** (valor) da "subkey" é **<valor>**

<taman>

Na função PutRegT o parâmetro **<nome>** não é uma cadeia, por isso especifica-se o tamanho de **<valor>**.

Exemplo

```
result= PutReg("M","Configuration","Config", "xxxxxx?????xxxxxx")
if result = .f. then ...
```

1.0 GetReg [U]ser e GetRegT [U]ser

Estas funções lógicas obtém o conteúdo de uma determinada "subkey" , dentro da "Key" **HKEY_CURRENT_USER\Software\Tecnocoop Sistemas\OpenBASE\Vn.m**.

Sintaxe

f=GetReg ("[U]ser",<chave>, <nome>,<valor>)
f=GetRegT ("[U]ser",<chave>,<nome>,<valor>,<taman>)

Onde:

"[U]ser"

Especifica que as funções **GetReg /GetRegT** deverão processar a "Key" **HKEY_CURRENT_USER\Software\Tecnocoop Sistemas\OpenBASE\Vn.m.**

<chave>

Especifica a subkey a ser extraída, cujo atributo (valor) **nome** é **<nome>**

<valor>

Especifica a variável que vai receber o **conteúdo** do valor <**nome**> da subkey <**chave**>.

<taman>

Na função GetRegT o parâmetro <**nome**> não é uma cadeia, por isso especifica-se o tamanho de <**valor**>.

Exemplo

```
result=GetReg (“U”,“Código do usuário”, “Código”,content)
if content = “12345”  then ...
```

1.0 GetReg [M]achine e GetRegT [M]achine

Estas funções lógicas obtém o conteúdo de uma determinada “subkey” , dentro da “Key” **HKEY_LOCAL_MACHINE\Software\Tecnocoop Sistemas\OpenBASE\Vn.m**.

Sintaxe

f=GetReg (“[M]achine”,<chave>,<nome>,<valor>)
f=GetRegT (“[M]achine”,<chave>,<nome>,<valor>,<taman>)
Onde:

“[M]achine”

Especifica que as funções **GetReg / GetRegT** deverão processar a “Key” **HKEY_LOCAL_MACHINE\Software\Tecnocoop Sistemas\OpenBASE\Vn.m.**

<chave>

Especifica a subkey a ser extraída, cujo atributo (valor) **nome** é <**nome**>

<valor>

Especifica a variável que vai receber o **conteúdo** do valor <**nome**> da subkey <**chave**>.

<taman>

Na função GetRegT o parâmetro <**nome**> não é uma cadeia, por isso especifica-se o tamanho de <**valor**>.

Exemplo

```
f=GetReg (“M”,“Configuration”, “Config”,content)
if content= “12345”  then ...
```

1.0 Exemplos de Funções do Registry

Abaixo seguem alguns exemplos das funções apresentadas acima, onde são efetuadas operações no Registry do Windows.

1.0.0 Exemplo 01

Neste exemplo:

- São colocadas informações de configuração na chave HKEY_LOCAL_MACHINE
- São obtidas, de volta, as informações de configuração previamente gravadas

```
prog
a =  REGINITKEY("HKEY_LOCAL_MACHINE")
b =  REGOPENKEY(a,"SOFTWARE")
c =  REGOPENKEY(b,"Microsoft")
d =  REGOPENKEY(c,"Windows")
e =  REGOPENKEY(d,"CurrentVersion")
f =  REGOPENKEY(e,"RunServices")
g =  RegCreateKey(f,"OpenBASE")
gg = REGOPENKEY(f,"OpenBASE")
h =  RegSetKey(gg,"bdsgbd","bdsgbd.exe")
i =  RegQueryKey(gg,"bdsgbd")
j =  RegCloseKey(gg)
k =  PutReg(“M”,"Configuration”,”Config”, “xxxxxx??????xxxxxx")
l =  GetReg(“M”,"Configuration”,”Config”,content)
return
```

1.0.0 Exemplo 02

Neste exemplo, usamos as informações referentes a produtos instalados e comandamos a remoção dos mesmos.

```
$nolib
$lentmp=256
prog

public StatusMens
StatusMens = "Press Help / Readme Buttons or use ;
              Help in Context for additional information."
Tit_janela="OpenBASE DEMO Version for Windows - ;
              Tecnocoop Sistemas - Brazil"
SetWindowText Tit_janela
SetWindowIcon "openb.ico"
set border on

PlaySound (".\docum\musica.wav")

do faz_dlg

proc faz_dlg
func vfy_geral(l)
botao = 1
vl1, vl2, vl3, vl5 = .f.
gbx1=1
public StatusMens

txt01 = "OpenServer.. DBMS Service and DEMO Data Bases"
txt02 = "OpenOPUS.... Developer language for Windows ;
         and WEB Applications"
txt03 = "OpenDLLs.... Interface modules for RAD Tools"
txt05 = "TSQLServer.. TSQL Server and interactive Query"

message StatusMens

Dialog 02,01,73,11
  Font "Times New Roman" 1
  caption " Welcome to OpenBASE Products Family !"
  style WS_POPUP  WS_CAPTION WS_SYSMENU ;
        DS_FIXEDSYS DS_CONTEXTHELP
  help ".\docum\install.hlp"

  GroupBox ' Select package to be Uninstalled ;
             and press "UnInstall" ' gbx1 01,01,71,09
    CheckBox txt01 vl1  03,03,68,01  CONTEXT(11)
    CheckBox txt02 vl2  03,04,68,01  CONTEXT(12)
    CheckBox txt03 vl3  03,05,68,01  CONTEXT(13)
    CheckBox txt05 vl5  03,06,68,01  CONTEXT(15)
    Defpushbutton "&UnInstall" botao 03,08,10,-12 IDOK
    PushButton "&Readme" NULL 15,08,10,-12 IDOK
    PushButton "&Help" NULL 27,08,10,-12 IDOK
    PushButton "&CD-Info" NULL 39,08,10,-12 IDOK
    PushButton "&Quit" NULL 51,08,10,-12 IDOK
    verify vfy_geral (botao, vl1, vl2, vl3, vl5)
EndDialog

quit

func vfy_geral
parameters botao(n), vl1(l), vl2(l), vl3(l), vl5(l)
private content(203)
public StatusMens

do case
  case botao = 2    && Readme
    cmd_run="notepad.exe .\readme.txt"
    run cmd_run
    message StatusMens
    return .f.
  case botao = 3    && Help
    cmd_run="notepad.exe .\install.txt"
    run cmd_run
    Message StatusMens
```

```
    return .f.
  case botao = 4    && CD-Info
    cmd_run="notepad.exe .\cd-info.txt"
    run cmd_run
    Message StatusMens
    return .f.
  case botao = 5  && Quit
    quit
endcase

hk=RegInitKey("HKEY_LOCAL_MACHINE")
h=RegOpenKey(hk,"SOFTWARE")
h1=RegOpenKey(h,"Microsoft")
h2=RegOpenKey(h1,"Windows")
h3=RegOpenKey(h2,"CurrentVersion")
h4=RegOpenKey(h3,"Uninstall")

if vl1
  h5=RegOpenKey(h4,"UOpenServerDemo97")
  varc=""
  varc=RegQueryKey(h5,"UninstallString")
  if !empty(varc)
    run varc SW_SHOWNORMAL
    DisableControl (2)
  else
    ret=MessageBox("OpenServer not installed ...", ;
         "OpenBASE DEMO UnInstall","O","I")
  endif
endif

if vl2
  h5=RegOpenKey(h4,"UOpenOPUS")
  varc=""
  varc=RegQueryKey(h5,"UninstallString")
  if !empty(varc)
    run varc SW_SHOWNORMAL
    DisableControl (3)
  else
    ret=MessageBox("OpenOPUS not installed ...", ;
         "OpenBASE DEMO UnInstall","O","I")
  endif
endif

if vl3
  h5=RegOpenKey(h4,"UOpenDLLs")
  varc=""
  varc=RegQueryKey(h5,"UninstallString")
  if !empty(varc)
    run varc SW_SHOWNORMAL
    DisableControl (4)
  else
    ret=MessageBox("OpenDLLs not installed ...", ;
         "OpenBASE DEMO UnInstall","O","I")
  endif
endif

if vl5
  h5=RegOpenKey(h4,"UTSQLDemo97")
  varc=""
  varc=RegQueryKey(h5,"UninstallString")
  if !empty(varc)
    run varc SW_SHOWNORMAL
    DisableControl (5)
  else
    ret=MessageBox("TSQLServer not installed ...", ;
         "OpenBASE DEMO UnInstall","O","I")
  endif
endif

Message StatusMens
```

return .f.

1 Outros comandos e funções da OpusWin

Neste capítulo apresentaremos alguns comandos e funções não abordados ate aqui, que possuem características especiais.

1.0 GetClient( )

Esta função cadeia obtém o nome do cliente, armazenado no Registry do Windows e/ou no arquivo "FacPrint".

Exemplos

```
? GetClient( )
Cliente = GetClient( )
```

1.0 GetVersion( )

Esta função cadeia obtém a versão do ***Open*BASE**, armazenado no Registry do Windows e/ou no arquivo "FacPrint". Por exemplo, o comando **? GetVersion( )**, retorna **Vn.m**, onde **n.m** informa a versão e release do produto OpenBASE.

1.0 GetDirectory( )

Esta função cadeia obtém o nome da pasta (ou diretório) default onde se encontram localizados os Bancos de Dados ***Open*BASE.** Esta informação é obtida a partir do Registry do Windows (ou mesmo do arquivo "FacPrint").
Por exemplo, o comando **? GetDirectory( )** poderá retornar, por exemplo:

- a cadeia **C:\USR\TSGBD\TSDIC** ou
- qualquer outro diretório que tenha sido estabelecido como default pelo usuário através do programa de configuração WinCNFG.EXE (ou BDCNFG).

1.0 Comando GET

O comando **GET**, da linguagem **OPUS**, foi reformulado para ser utilizado, também, em aplicativos ***OpusWin***. A sintaxe do comando GET (na ***OpusWin*)** é compatível com a sintaxe desse mesmo comando na **OPUS**, tendo sido adicionadas, apenas, algumas opções, como veremos a seguir. Sedo o objetivo deste manual apresentar apenas os recursos da linguagem **OpusWin,** recomendamos a leitura dos manuais da **OPUS** para uma explicação completa e detalhada do comando **@ ... SAY** e **@ ... GET**.

Por sua natureza, este comando deve ser utilizado na ***OpusWin*** apenas por razões práticas de compatibilidade quando da migração de aplicativos para ambiente gráfico Windows, a partir de aplicativos não gráficos, geralmente, UNIX e aplicações console.

1.0.0 Sintaxe do comando GET

```
@<lin>,<col> [SAY <expc>] GET <var> [Picture <picture>]
        [Valid <expl>] [Message <msg> ]
        [CONTEXTid <expn>]
        [InitMess <expc>] ...
Read
```

Onde:

@<*lin*>,<*col*>

Especifica as coordenadas da janela de GET. Os tamanhos serão calculados automaticamente pela ***OpusWin***. Os comandos **Set Border** e **Set Color** determinam detalhes de apresentação da janela principal e das janelas de GET. Veja o exemplo a seguir e consulte os Manuais da **OPUS** para maiores detalhes.

Valid <*expl*>

Especifica uma expressão lógica. Se a expressão é "**true**", assume-se que o campo <***var***> está corretamente digitado. Caso contrário, o conteúdo de <***var***> será tido como inválido, sendo exibida, na Barra de Status, a mensagem especificada em <***msg***> da cláusula Message.

Message <msg>

Especifica uma mensagem a ser exibida na barra de status quando a expressão lógica <***expl***> for **"false**", ou seja o conteúdo do campo sendo processado é inválido.

CONTEXTid <expn>

Especifica um tópico de Ajuda associado com um determinado elemento da janela GET. O parâmetro <**expn**> especifica o identificador de contexto ("context identifier") associado a um determinado tópico quando o arquivo

de Help foi projetado e construído. Consulte os manuais específicos do software utilizado para a elaboração do Help.

InitMess <expc>

Indica a mensagem a ser exibida na Barra de Status, imediatamente antes da execução do comando GET.

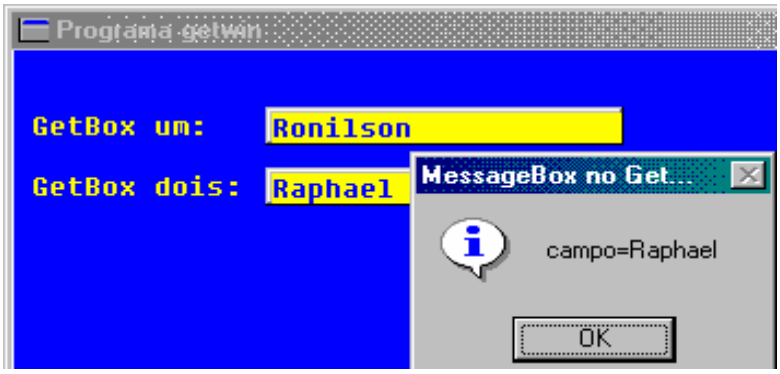
1.0.0 Exemplo do comando GET

Veja a seguir um pequeno exemplo de utilização do comando GET.

```
$nolib
prog
func vfy_geral(l)
campo1, campo2 = ""
botao = -1
set color to "GR+/B,B/GR+"
set border on
@02,01 SAY "GetBox um:   " GET campo1 Picture REP ("X",20);
        valid vfy_geral (campo1);
        Message "Digite o campo1 ...";
        InitMess "Exemplo de Comando Get ...";
        CONTEXTid (11)
@04,01 SAY "GetBox dois: " GET campo2 Picture REP ("X",20);
        valid vfy_geral (campo2);
        Message "Digite o campo2 ...";
        InitMess "Exemplo de Comando Get ...";
        CONTEXTid (11)
Read
resp=""
@08,01 SAY "Tecle ESC para Sair ..." GET resp
Read
return
func vfy_geral
parameters campo
if empty(campo)
  return .f.
endif
mens="campo="+trim(campo)
ret=MessageBox(mens,"MessageBox no Get...","O","I")
return .t.
```

Observações

Veja a tela gerada pelo programa acima. Recortamos o resultado para focalizar apenas as janelas GET.



1.0 Função GetDiskUsage()

A função cadeia GetDiskUsage() retorna a percentagem do espaço ocupado dentro do <device> especificado como parâmetro na chamada dessa função.

1.0.0 Sintaxe da função GetDiskUsage()

```
Perc_lilvre= GetDiskUsage()
```

1.0.0 Exemplo de GetDiskUsage()

```
? GetDiskUsage("C:\")
? GetDiskUsage("/dev/hda3")     && ambiente OPUS (Unix)
```