Introdução ao Delphi

Parte I

Sumário

Introdução

O Delphi, fabricado pela **Borland/CodeGear** é uma ferramenta visual para programação em ambiente Windows, Orientada à Objetos, baseada na linguagem Pascal - chamada de **Object Pascal**. Portanto, podemos dizer que o Delphi é um *Visual Pascal*, já que incorpora a maior parte da sintaxe desta linguagem somada às facilidades da Orientação à Objeto.

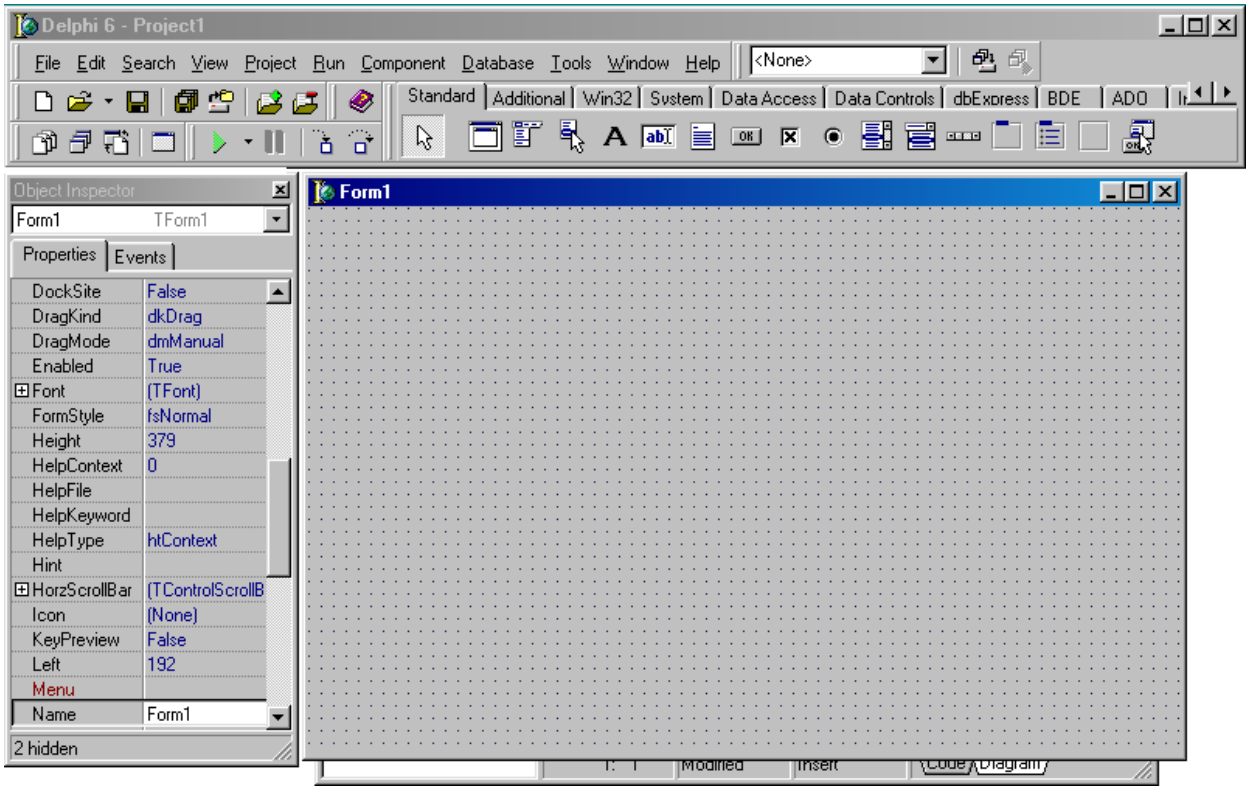


Figura 1 - Aparência do Delphi

Através deste ambiente é que se torna possível realizar todas as operações de programação e de arraste de componentes. Na página seguinte, vamos identificar cada um dos elementos mostrados acima.

Menus do software (abrir, salvar, novo projeto, etc)

"*Speed Bar*" (botões com as opções mais usadas: Salvar, Salvar Tudo, etc)

Paleta de componentes

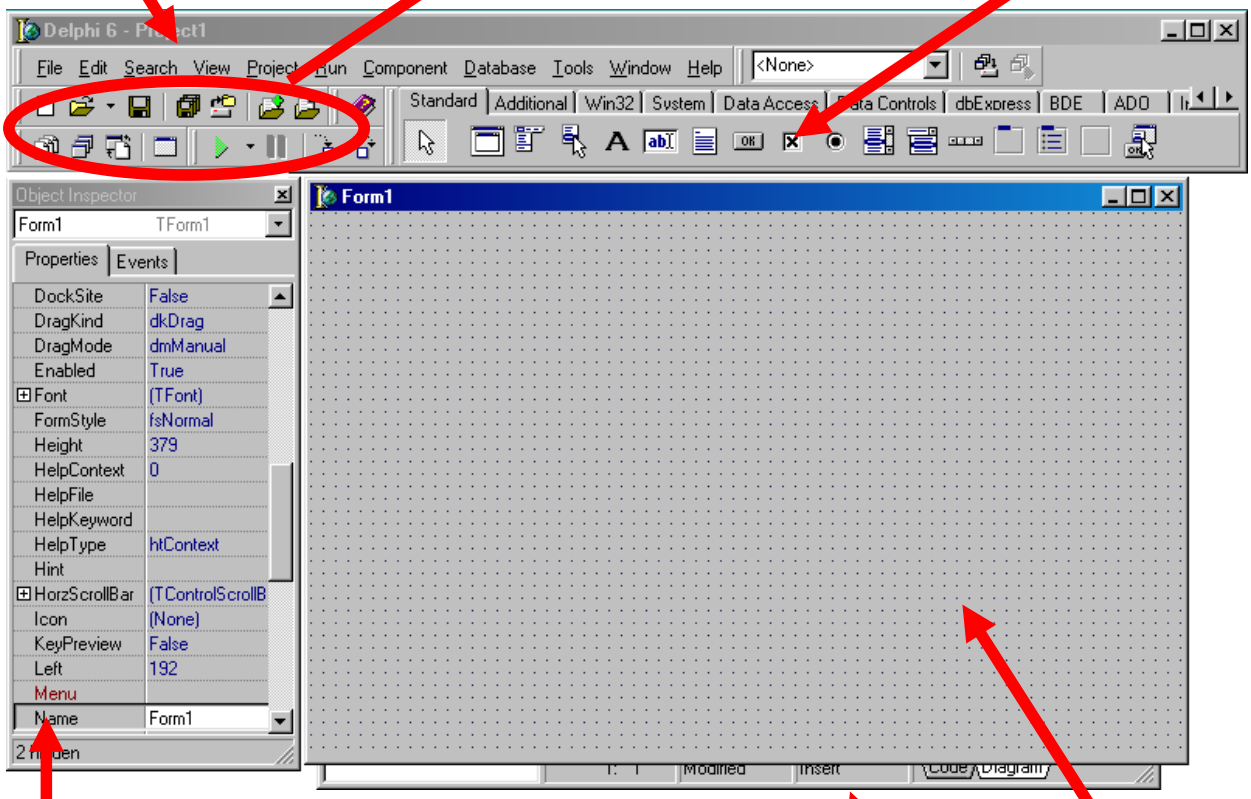


Figura 2 - Identificando componentes

"*Object Inspector*": contém Propriedades do objeto selecionado (guia "*Properties*") e eventos deste objeto ("*Events*")

Formulário a partir do qual o aplicativo será construído (também chamado de "*Form*" ou "*janela*")

Atrás do Form, página contendo os códigos da aplicação

Como você deve ter percebido, a página contendo a codificação está logo "abaixo" do formulário principal do programa. Embora ainda não tenhamos programado nada, veremos que a janela de códigos já possui "coisas" escritas. Na verdade, estas "coisas" foram geradas pelo próprio Delphi que, percebendo a existência de um formulário na aplicação, já criou códigos que o identificam perante o software. Observe:

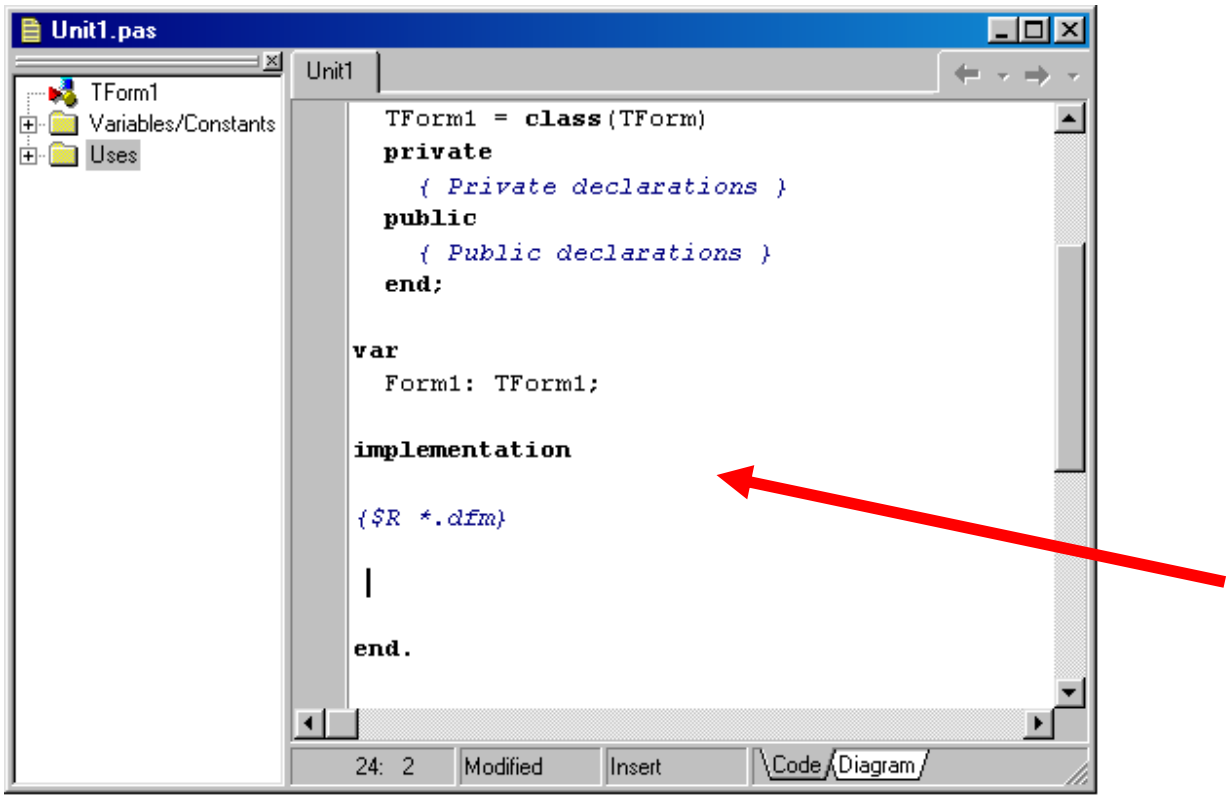


Figura 3 - Códigos gerados pelo Delphi

A codificação acima foi gerada pelo próprio Delphi. Esta "janela" que contém a codificação é chamada *Unit*. Cada aplicativo criado por você poderá ter uma ou mais *Units*, já que cada *Form* está obrigatoriamente associada a uma *Unit*. Portanto, se seu programa tiver dez formulários, ele terá a mesma quantidade de *Units*.Você começa a codificar a partir de onde a seta está. Veremos a seguir o modo como cada Unit está estruturada.

Estrutura de uma Unit

Sempre que você criar um formulário, uma Unit também será criada. Eis a estrutura de uma delas:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Tabela 1 - Estrutura de uma Unit

A primeira linha mostra o nome da Unit que, no caso, é *Unit1*. Os nomes-padrão de Unit são sugeridos pelo próprio Delphi e assumem esta seqüência: *Unit1, Unit2, Unit3*, e assim por diante. Se eu quisesse chamar de *MinhaUnit* ou de qualquer outro nome também seria possível. Quando você tiver projetos com muitas Units, é recomendável usar um nome que lhe seja mais amigável; caso contrário, aceite os nomes definidos pelo Delphi.

Em seguida vem a cláusula *Interface*. Esta linha indica que todas as variáveis e tipos definidos abaixo dela (porém antes da cláusula *Implementation*) poderão ser vistas por qualquer Unit. É diferente do que acontece com a cláusula *Implementation*. Abaixo dela, somente a própria Unit (isto é, a Unit atual) poderá acessar as variáveis e tipos ali definidos.

Após a cláusula Interface, vem a declaração *Uses*. Se você quiser que o seu aplicativo acesse uma outra Unit pré-existente no Delphi (o Delphi vem com várias Units pré-existentes) ou mesmo uma Unit que você criou através de outro formulário, coloque seu nome ao final da linha *Uses*.

A cláusula *Type* armazena os *tipos* usados pelo programa e é preenchida automaticamente pelo Delphi. Por exemplo, ao criar um novo formulário, a linha **TForm1 = class (TForm)** aparece imediatamente, identificando que aquele formulário pertence à classe de objetos chamada *TForm*. Na verdade, de acordo com os princípios de orientação a objetos, **TForm1** é uma classe derivada

da "*classe-pai*" **TForm -** o que implica dizer que **TForm1** tem as mesmas características de **TForm**. Sempre que eu arrastar um objeto da **Paleta de Componentes** (também chamada de **VCL – Visual Component Library**) para o formulário, o Delphi criará uma linha para ele dentro da cláusula *Type*.

A cláusula *Var* armazena as variáveis globais, que podem ser acessadas por qualquer função daquela Unit. O Delphi cria automaticamente uma variável chamada **Form1 : TForm1**, para permitir a manipulação do formulário recém-criado, onde Form1 é o nome da variável e **TForm1** é o tipo desta variável.

Por fim, a cláusula *Implementation* que, como já disse, informa à Unit que somente ela própria poderá acessar as variáveis e tipos ali definidos.

Propriedades de um Objeto

"O Delphi é uma ferramenta essencialmente visual e apresenta diversos recursos para alterar as características de uma aplicação visualmente. Os componentes são adicionados numa Form clicando-se no ícone do componente desejado, na *Paleta de Componentes*, e clicando-se na Form, na posição desejada. Caso o componente não esteja na posição correta, pode-se arrastá-lo com o botão esquerdo do mouse pressionado para a posição desejada.

Uma vez na Form, pode-se alterar as características do componente selecionando-o, dando um clique sobre ele e usando o *Object Inspector*. Ao selecionar um componente, o Object Inspector modifica-se para mostrar as suas características, ou seja, suas propriedades, na parte inferior da janela.

A parte superior é uma *Combobox*, caixa que se abre ao clicar no botão com a seta para baixo, e que contém todos os componentes da Form atual, que está selecionada. Escolher um componente desta Combobox é equivalente a selecionar o componente, clicando sobre ele. Uma vez selecionado, pode-se inspecionar suas propriedades ou mesmo alterá-las.

A maneira de alteração depende do tipo de propriedade escolhida: se for um string ou um número, basta teclá-lo no quadro correspondente. Por exemplo, seleciona-se a Form e, defronte à propriedade Caption, tecla-se o novo título da janela. Note que, à medida que vai se teclando, o título da janela altera-se simultaneamente"[1]. Observe a Figura 4.

Eventos

"Uma das coisas complicadas para quem vem da programação DOS é o conceito de Eventos. A programação DOS é linear, o programa é ativo, isto é, ele chama as rotinas que deseja executar, quando isso é necessário. Já na programação Windows, o programa é essencialmente passivo, isto

[1] SONINNO, 2000

é, ele é chamado quando algo acontece. Por isso, ele deve estar preparado para tudo, em qualquer ordem.

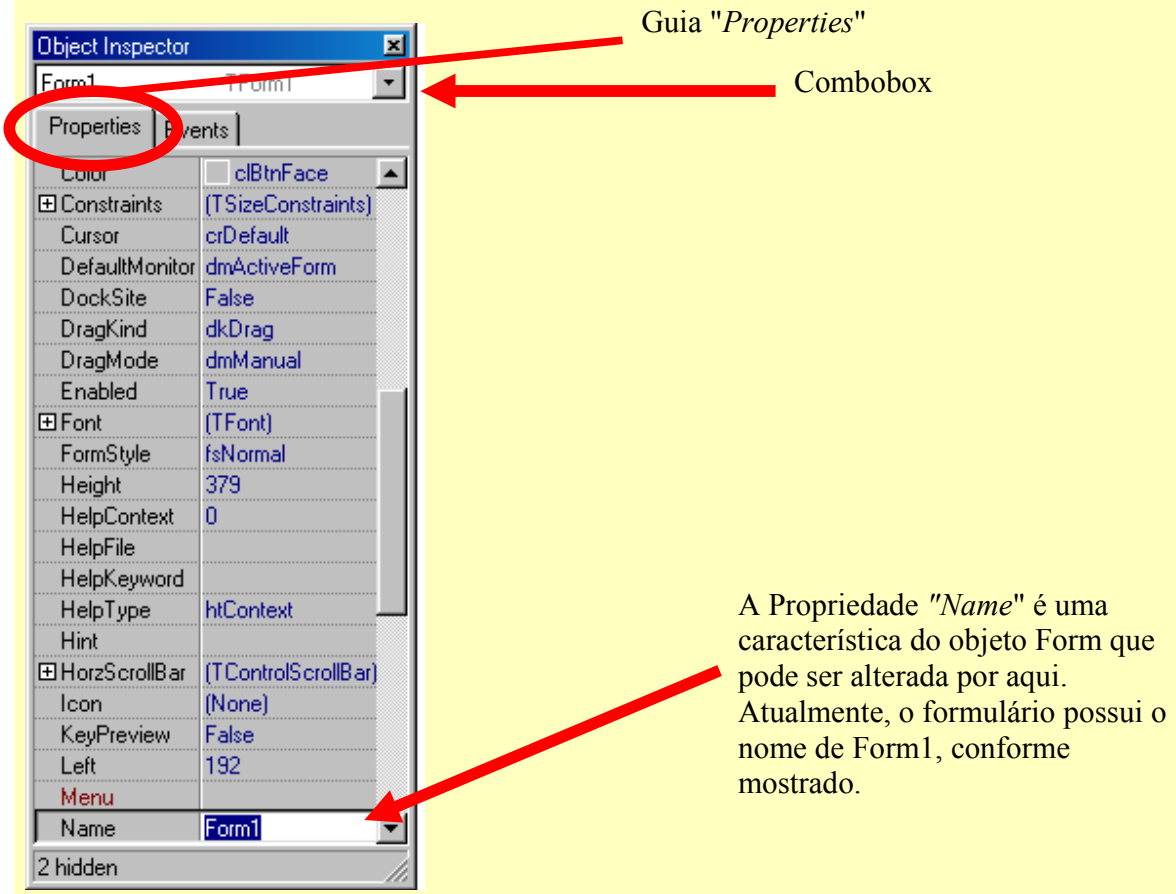


Figura 4 - Visão do Object Inspector

Isto seria semelhante à maneira usual de trabalho: você está executando uma tarefa, quando toca o telefone. Deve-se ter uma rotina para atender ao telefone. Este telefonema é do chefe, chamando em seu escritório. Você interrompe a tarefa que estava sendo executada, vai ao escritório do chefe, retorna e retoma a tarefa.

Com o Windows, acontece coisa semelhante: o programa está executando uma tarefa (normalmente esperando que algo aconteça), quando o usuário pressiona o botão do mouse ou uma tecla. Ele deve ter uma rotina para tratar este botão do mouse ou tecla, executando alguma tarefa especial, caso necessário.

O Delphi facilita muito este tratamento de 'interrupções' no processamento, com os eventos dos componentes. Na segunda guia do *Object Inspector* são mostrados os eventos disponíveis para este

componente. Cada um desses eventos é acionado quando é executada uma ação. Por exemplo, quando se clica com o botão do mouse sobre um Form é ativado seu evento **OnClick**. Se houver algum código associado a este evento, ele é executado neste momento.

Dando-se um clique duplo no evento que se deseja tratar, abre-se o editor de código, onde se coloca o código que tratará deste evento"[2]. Observe a ilustração a seguir:

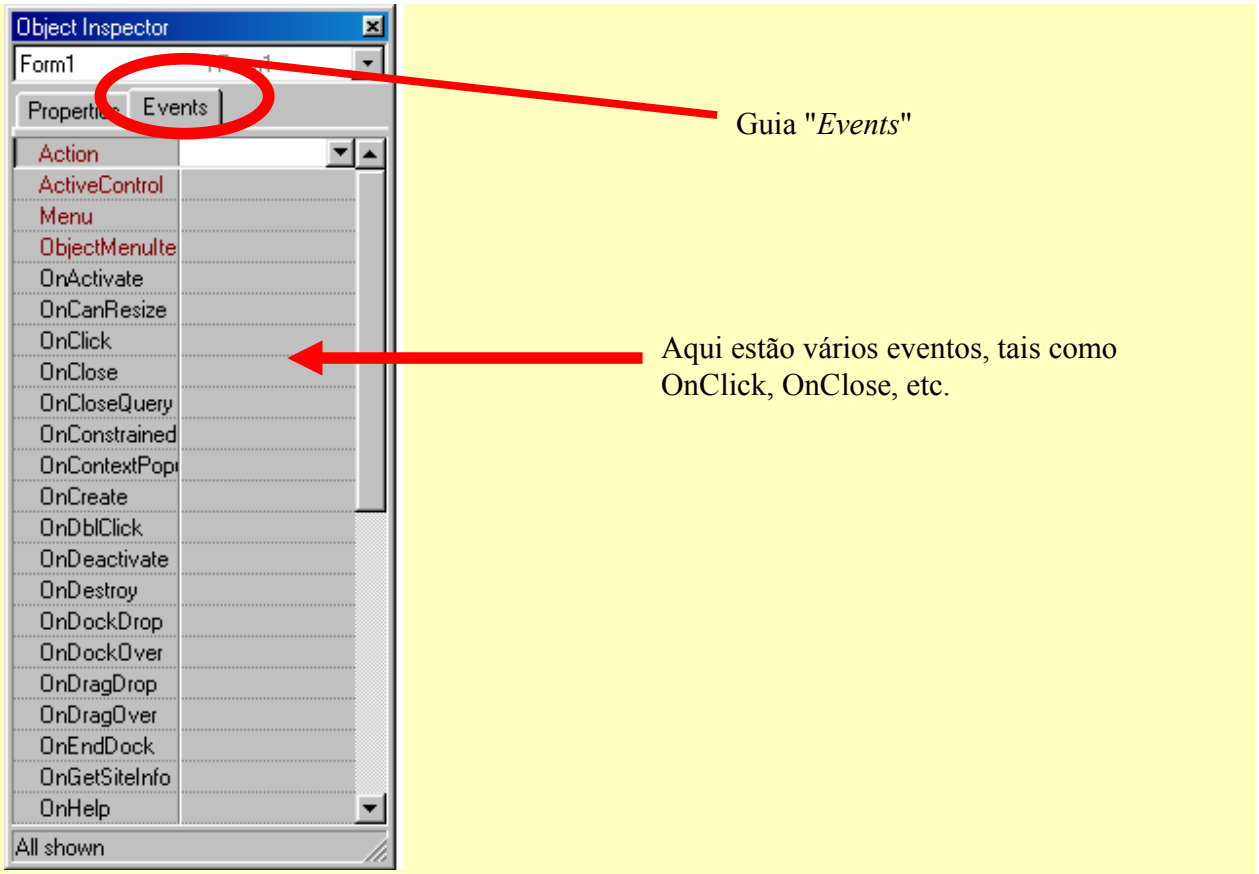


Figura 5 - Eventos no Object Inspector

Por exemplo, digamos que você queira que assim que o formulário **Form1** for clicado com o mouse, apareça a mensagem "*Isto foi um clique*". Para tanto, no *Object Inspector*, dê dois cliques no campo em branco logo à frente do **evento** *OnClick*. O seguinte código é gerado automaticamente:

[2] SONINNO, 2000

```
procedure TForm1.FormClick(Sender: TObject);
begin

end;
```

Agora, entre o *Begin* e o *End*, digite:

```
ShowMessage('Isto foi um clique');
```

O código ficará assim:

```
procedure TForm1.FormClick(Sender: TObject);
begin
 ShowMessage('Isto foi um clique');
end;
```

Procedure é o nome dado a uma determinada rotina do Delphi. Veremos mais sobre rotinas adiante.

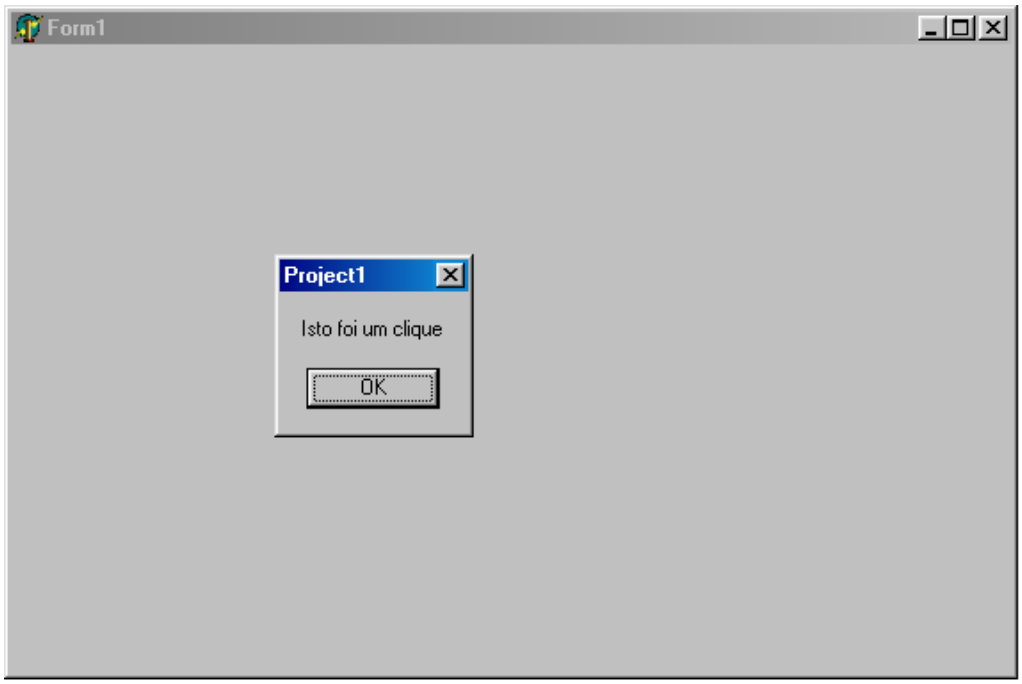


Figura 6 - O seu aplicativo em execução

Para conferir este código em funcionamento, compile e execute seu programa pressionando a tecla **F9** do teclado ou vá até o menu **Run** e selecione também a opção *Run*. Ao fazer isso, o Delphi pedirá a você que salve seu *projeto* (cada aplicativo construído em Delphi é encarado por ele como um *projeto*). Primeiramente o Delphi pedirá a você que salve a Unit, que será um arquivo de extensão **.pas* contendo o código do aplicativo. Você pode dar o nome que quiser a este arquivo. Se preferir, aceite a sugestão do Delphi e deixe como *Unit1*.

Em seguida, deve-se salvar o projeto como um todo. Código e projeto não são armazenados no mesmo arquivo e, sabendo disso, o Delphi pede que você digite um nome ao seu projeto. O nome dado a ele será o nome do executável. Por exemplo: se você salvar seu projeto como *Teste*, o Delphi, ao compilar seu programa, gerará um arquivo chamado *Teste.exe*. O projeto, antes da compilação, gera um arquivo de extensão **.dpr*, que servirá de base para a criação do executável.

Tendo compilado seu aplicativo (clicando em *Run* > *Run* ou pressionando *F9*), um arquivo executável de extensão ***.EXE** aparecerá no mesmo diretório onde os arquivos **.pas* e **.dpr* estão armazenados e seu aplicativo será executado. Para compilar sem rodar o aplicativo, clique no menu **Project** e selecione **Compile Project1**[3] ou pressione **CTRL+F9**.

Com seu aplicativo em execução, vamos testá-lo: dê um clique em qualquer área da janela do formulário e note que a mensagem que você digitou irá aparecer (Figura 6).

Toda vez que você alterar o código de sua aplicação ou acrescentar novos objetos em seu formulário, recomenda-se salvar seu projeto novamente. Basta, para isso, acionar a opção **File > Save** ou **File > Save All** no menu do Delphi. Esta última opção salva tanto código quanto projeto de uma só vez; a opção **File > Save** salva apenas a janela ativa - se a janela ativa for a do código, este será salvo num arquivo **.pas*; se for a do projeto (a do formulário), o projeto será salvo num arquivo **.dpr*.

Tanto o arquivo **.pas* quanto o **.dpr* são, na verdade, arquivos puramente textos e podem ser abertos até mesmo no **Bloco de Notas** (*Notepad*) do Windows.

[3] *Atenção*: ao invés de *Project1*, o Delphi usará o nome que você deu ao seu projeto; isso quer dizer que se seu projeto chama-se Teste, a opção vista será *Compile Teste*.

Principais Propriedades de um Form

Todos os objetos possuem certas propriedades, estejam elas visíveis no Object Inspector ou não. Conheçamos aqui algumas das propriedades mais importantes de um formulário.

Propriedade Name

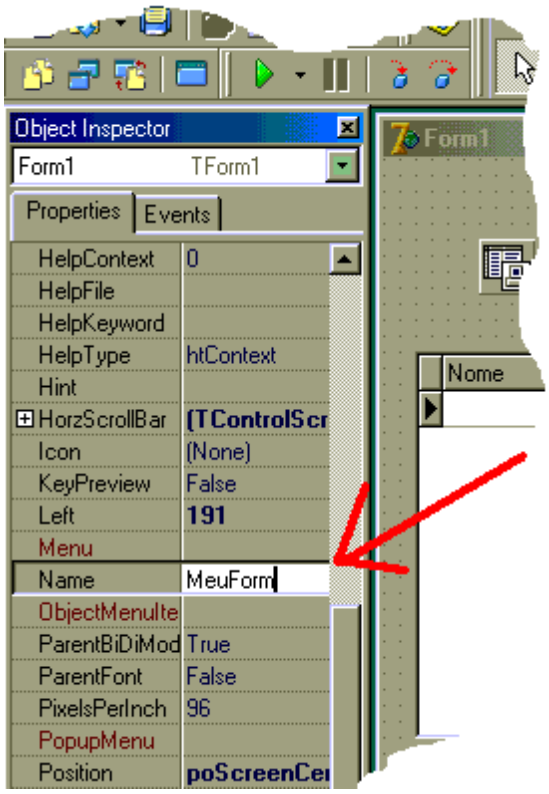


Figura 7 - Propriedade Name

Esta é uma propriedade comum à todos os objetos. Na propriedade **Name**, dá-se o nome desejado a um componente, nome pelo qual ele poderá ser referenciado em código. É comum atribuirmos nomes que nos "digam algo" sobre aquele objeto, ou sobre o contexto no qual ele se encontra em nossa aplicação.

Por exemplo, o formulário principal de minha aplicação poderia ser chamado de **FrmPrincipal**. É mais fácil lembrar-se de um nome como **FrmPrincipal** do que, simplesmente, **Form1** (ou ***Form2***, ***Form3***, etc), principalmente se sua aplicação contiver vários formulários.

Para alterar a propriedade *Name*, basta selecionar o formulário desejado e, no Object Inspector, buscar por Name. Na coluna da direita, digite o nome desejado.

Na figura que ilustra esta propriedade, o programador decidiu dar ao seu formulário o nome de **MeuForm**.

A partir desse momento, nas menções que ele desejar fazer ao seu formulário ele usará *MeuForm*.

Propriedade Caption

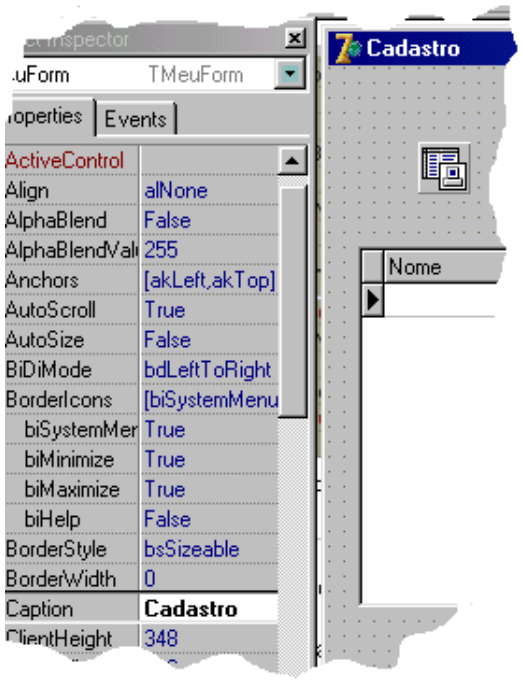


Figura 8 - Propriedade Caption

A propriedade **Caption** de um Form é responsável pelo texto exibido na barra de títulos da janela, conforme mostra a figura ao lado.

Ao alterar-se a propriedade **Caption** e teclar ENTER, a barra de títulos já assume o novo valor.

Outros componentes, como o **Label**, também possuem esta propriedade para exibir textos (não necessariamente numa barra de títulos).

Propriedade BorderIcons

Esta propriedade diz ao aplicativo quais ícones deverão ser mostrados na barra de títulos da janela. A propriedade **biSystemMenu**, quando alterada para *TRUE*, mostra uma **cortina de menu** quando o ícone da janela do aplicativo é clicado, além do botão com um **"X"** (equivalente ao botão *Fechar* dos aplicativos Windows) no lado direito.

Já **biMinimize** exibe o botão de **Minimizar Janela** (aquele com um sinal de "*menos*" (**-**) no lado direito da barra do aplicativo). E **biMaximized** mostra o botão **Maximizar / Restaurar** Janela.

A propriedade **biHelp**, mesmo quando **TRUE**, aparecerá apenas se o tipo de formulário for **bsDialog** ou quando as opções **biMinimized** e **biMaximized** forem **FALSE**. Nestes casos, um ponto de interrogação aparecerá no lugar destes ícones.

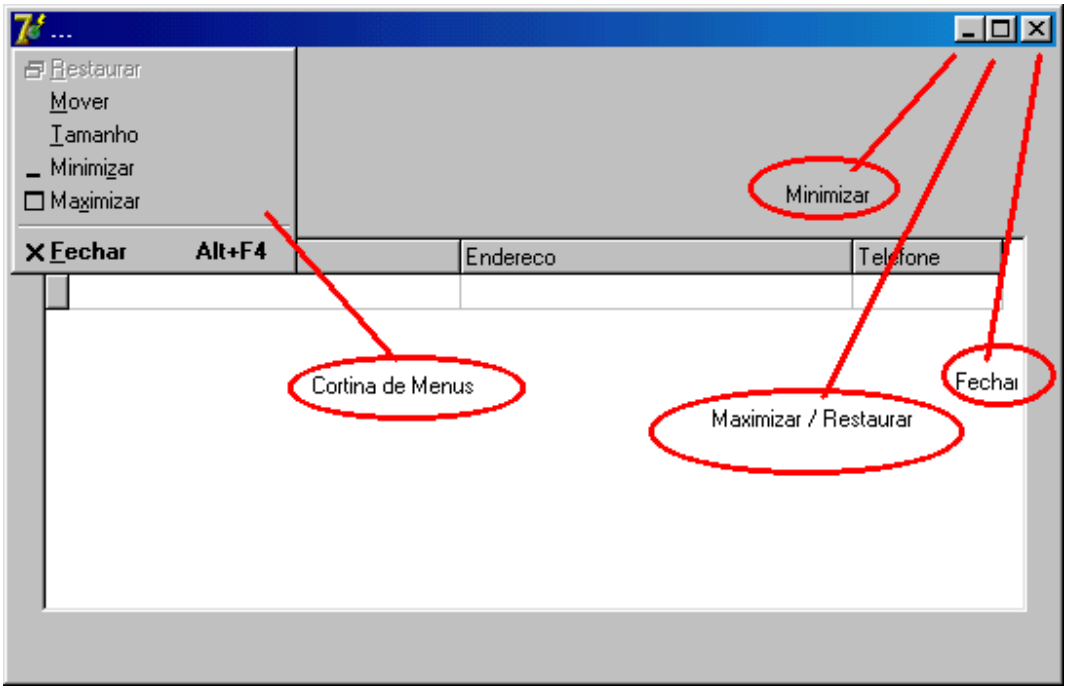


Figura 9 - Ícones do formulário

Propriedade BorderStyle

Esta propriedade controla o tipo de "*bordas*" do formulário. Por exemplo, a opção **bsSizeble** permite que o usuário redimensione o formulário com o mouse, selecionando uma das pontas e arrantando-a. Já a opção **bsSize** é idêntica na aparência, mas impede este tipo de redimensionamento.

A opção **bsDialog** exibe um formulário com aspecto mais simples, sem os botões da janela (maximizar, fechar, etc) e sem o ícone do aplicativo na ponta esquerda. E **bsNone** nem mesmo exibe a barra de títulos.

Propriedade Color

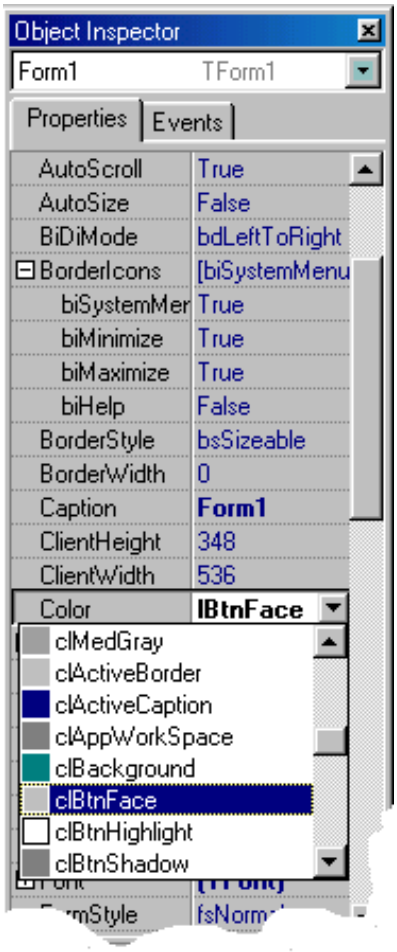


Figura 10 - Propriedade Color

Comum não somente aos Forms, como também a vários outros componentes, a propriedade Color permite que se altere a cor do objeto. Por exemplo, para que seu formulário fique branco, mude a cor dele para **clWhite**. O padrão é **clBtnFace** (que assume um tom de cinza). Observe que, além das cores comuns (**clRed, clGreen, clWhite** ...) há outras com denominações meio "estranhas", tais como **clActiveCaption, clBtnText, clBtnFace**, etc.

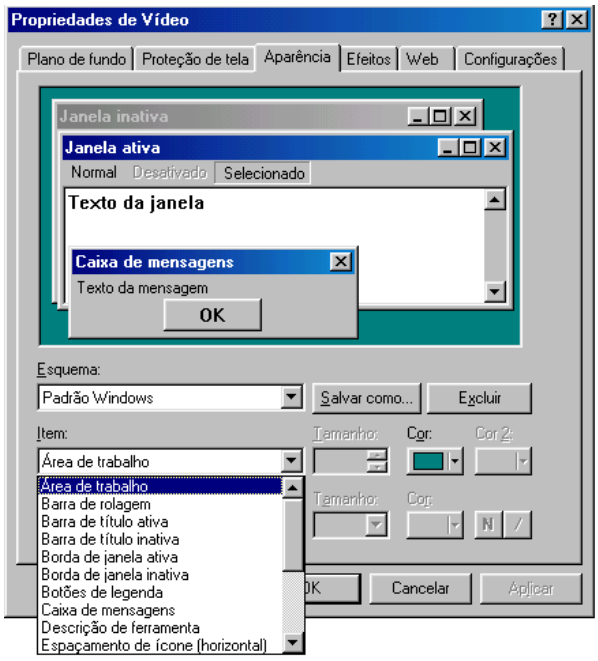


Figura 11 - Esquema de cores do Windows

Boa parte destas cores "estranhas" assume tons de cinza ou tons derivados do azul, salvo algumas exceções. Mas o que significam tais cores?

A diferença delas para as demais é que estas mudam conforme o usuário modifica o **Esquema de Cores do Windows**. Por exemplo, se você escolher a cor **clActiveCaption** para um componente de sua aplicação (cuja cor padrão é **azul-marinho**), quando o usuário alterar o esquema de cores e deixar todas as barras de títulos **cor-de-rosa**, o seu componente também ficará cor-de-rosa. Aliás, é este o significado de **clActiveCaption**: **Barras de Títulos** (ou **de Componentes**) **ativa**.

Seguindo o mesmo raciocínio, **clBtnFace** indica a cor-padrão dos objetos num aplicativo, tais como formulário e botões; **clActiveBorder** indica a cor das bordas de uma janela ativa; clInactiveBorder indica a cor das bordas de uma janela inativa; **clInactiveCaption** indica barra de títulos inativa, e assim por diante.

Propriedade Enabled

Indica se o componente está habilitado ou não. Se o componente estiver configurado para FALSE, ele estará desabilitado, o que significa que não será possível selecioná-lo com o mouse. Esta propriedade está presente em diversos componentes. O valor padrão para **Enabled** é TRUE.

Propriedade Font

Comum também a outros componentes, permite definir a fonte do texto do formulário, seu tamanho e cor, além de estilos como **Negrito, Itálico, Sublinhado** e **Tachado** (***StrikeOut***).

Propriedade Position

Position indica a posição do Formulário perante as dimensões da tela. O padrão é **poDesigned**, o que indica que o Form ficará, quando o aplicativo estiver sendo executado, exatamente na mesma posição em que ele se encontrava durante o seu desenvolvimento. Isso é ruim, já que, se o mesmo encontrava-se bem no canto direito da tela, ele permanecerá lá quando o software estiver rodando.

O mais desejável é que se use a opção **poScreenCenter**, para que, ao ser executado, ele assuma o centro da tela.

WindowState

Indica qual o estado da janela do formulário: se configurado para **wsMaximized**, o formulário será maximizado quando seu aplicativo estiver rodando; se **wsMinimized**, ele será executado minimizado; se **wsNormal** (que é o valor padrão), ele será executado exatamente nas dimensões em que foi criado.

Desenvolvendo a Primeira Aplicação

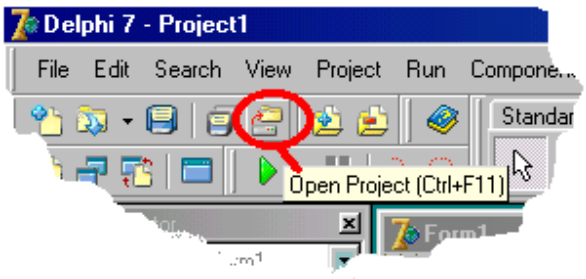


Figura 12 - Ícone para abrir um projeto já existente

Para desenvolver seu primeiro aplicativo, inicie o Delphi (caso ele ainda não tenha sido iniciado). Para isso, clique no menu **INICIAR** do Windows e, em **PROGRAMAS**, escolha **Borland Delphi 7** (ou **Borland Delphi 6**, dependendo da versão que você possui). Por padrão, o Delphi já inicia com uma nova Unit e um novo formulário. Como vamos desenvolver um aplicativo a partir do zero, então podemos mantê-los.

Caso quisesse continuar o desenvolvimento de um aplicativo já começado anteriormente, bastava você acionar o menu **File** e escolher **Open Project**. Na caixa de diálogo que se abriria, você deveria selecionar o arquivo ***.dpr** correspondente. Você também poderia atingir o mesmo objetivo clicando no ícone **Open Project** ou pressionando **CTRL+F11**.

Figura 13 - Escolhendo um Button

Aqui, vamos exercitar alguns eventos e propriedades do Delphi. Coloque um botão (**Button**, da paleta **Standard**) no seu formulário e mude sua propriedade **Caption** para **Sair**.

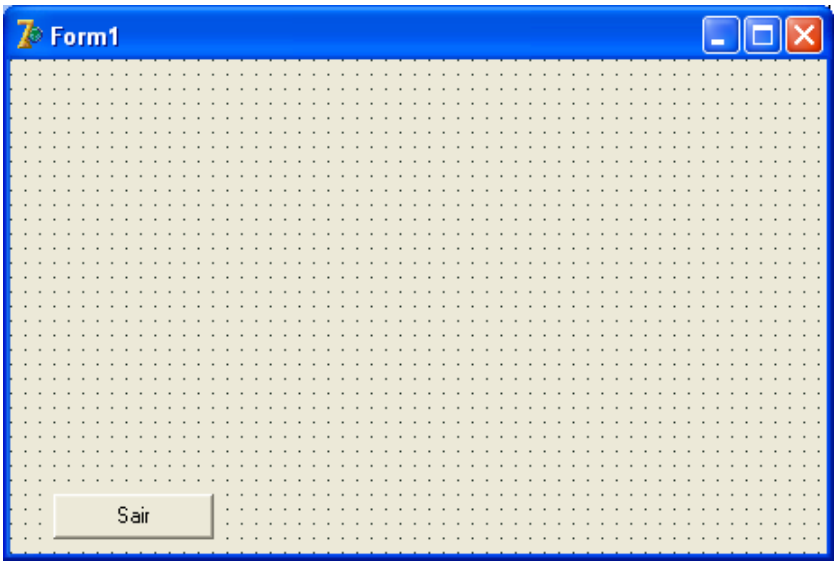


Figura 14 - Aparência da aplicação

Seu formulário deve ficar com a aparência da Figura 14. Vamos codificar seu evento **OnClick** para que, ao ser clicado, o formulário se feche, fazendo com que seu aplicativo seja encerrado. Para codificá-lo, dê dois cliques sobre o botão **Sair** ou selecione, na guia Events do Object Inspector o evento OnClick. Dês dois cliques no espaço em branco à sua frente e digite, entre **begin** e **end**, o comando *Close,* conforme mostrado abaixo:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Close;
end;
```

Execute seu aplicativo, pressionando **F9** no teclado. Ao clicar sobre seu botão Sair, seu programa é encerrado.

Vamos alterar também a propriedade **Caption** do formulário para **Meu programa** (Figura 15)– afinal, nunca se viu a barra de títulos de um software com o nome de Form1.

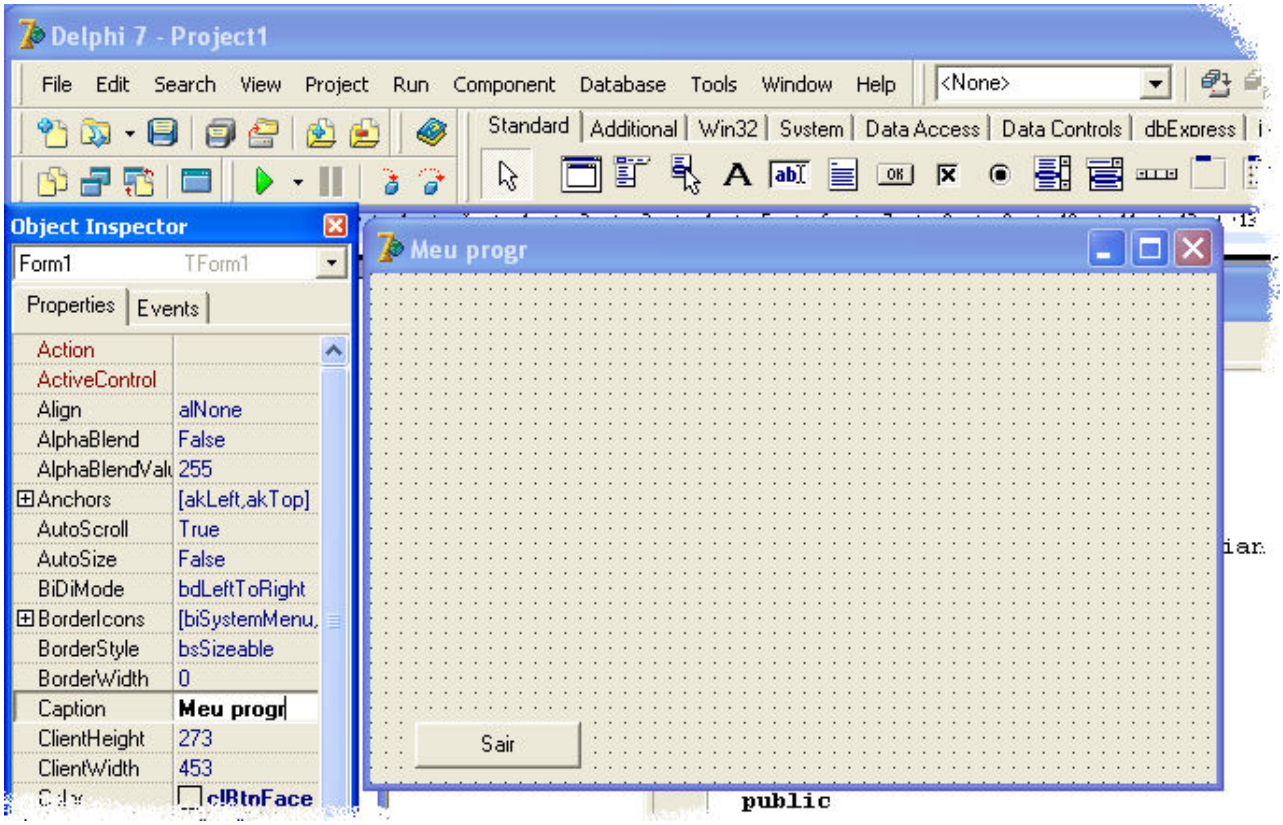


Figura 15 - Alterando o Caption do form

Variáveis, Operações, Comandos e Rotinas

Antes de começarmos qualquer outro código, é melhor que estejamos habituados com algumas particularidades usadas pelo Delphi, tais como os tipos de variáveis suportadas, como utilizar as operações básicas, etc. Abaixo, estão listados alguns dos tipos mais comuns usados pelo Delphi:

Tipo	O Que é	Exemplo
Integer	Número Inteiro	5
Double	Número real (isto é, com ponto-flutuante)	5,45
Bool	Booleano	TRUE/FALSE
Char	Caracter	'c'
String	String	'isto é uma string'

Há outros, como o *Extended*, que excerce a mesma função do tipo *Double*, porém com suporte a uma quantidade maior de algarismos; *ShortInt* realiza as mesmas operações inteiras que o tipo *Integer,* porém com capacidade menor de armazenamento; e assim por diante.

Vamos passar agora às operações realizadas pelo Delphi:

Operação	O Que é	Exemplo
+	Adição / Concatenação de Strings	5+5
-	Subtração	5-2
/	Divisão	4/2
*	Multiplicação	5*4
< >	Diferença	If A<>5 then ...
=	Igualdade / Comparação	If A=5 then ...
:=	Atribuição	A:=5

O Comando If

Passemos agora aos comandos do Delphi. A seguir, um exemplo de como usar a instrução **If:**

```
If A=5 then
 ShowMessage('É cinco')
 Else
  ShowMessage('Não é cinco');
```

O comando acima diz que se **A** for igual a **cinco**, então deve ser mostrada na tela a mensagem "*É cinco"*; do contrário, exibe-se "*Não é cinco*". Normalmente, toda instrução do Delphi termina com *ponto-e-vírgula* (;) - como é o caso da última linha. A exceção é a instrução **Else**. A linha que antecede o **Else** NÃO DEVE CONTER ponto-e-vírgula.

Note também que as strings são delimitadas por aspas simples (′) e não aspas duplas (″). Outro exemplo:

```
If A=5 then
 ShowMessage('É cinco');
```

Aqui, a instrução terminou com ponto-e-vírgula porque não houve a presença da instrução **Else**. Observe este outro exemplo, agora com mais de uma linha numa das condições:

```
If A=5 then
 Begin
  ShowMessage('É cinco');
  B:=10;
  F:=23;
End
Else
 ShowMessage('Não é cinco');
```

Aqui, se A for cinco, então deve-se exibir a mensagem "*é cinco*" na tela e, além disso, a variável B deverá ser atribuída de 10 e F de 23. Como tudo isso está relacionada à mesma condição (isto é, para A=5), então foi necessária a delimitação deste trecho com um **Begin** e um **End**. Note que, como já disse, antes de **Else** não se usa ponto-e-vírgula na linha imediatamente anterior: por esse motivo, **End** ficou sem o ponto-e-vírgula. Isso, porém, não se aplicou às demais linhas.

O Comando For

A sintaxe deste comando, usado para repetir uma instrução diversas vezes, é simples:

```
For B:=1 to 10 do
 A:=A+1;
```

Acima, a variável A deve ser adicionada de 1 até que B seja igual a 10. Isto que dizer que, a cada repetição, B tem uma unidade adicionada a seu valor. Quando B for igual a 10, o *loop*[4] é finalizado. Também pode-se decrescer a variável, assim:

```
For B:=10 downto 2 do
 A:=A+1;
```

O código acima somará 1 à variável A até que B seja igual a 2. A cada repetição, B tem uma unidade subtraída de seu valor, até que se atinja o valor 2. Note que o comando **For** incrementa ou decrementa números de um em um, nunca de dois em dois ou de três em três.

O Comando While

Para decrementar ou incrementar valores maiores que 1, você pode usar o comando **While**. Veja o exemplo:

```
A:=1;
While A < 10 do
 Begin
  ShowMessage('Menor que Dez');
  A:=A+2;
 End;
```

Inicialmente, A recebeu valor 1. O looping acima será executado até que A seja igual a 10 e, a cada repetição, a mensagem "*Menor que Dez*" é mostrada na tela e A é incrementado de duas unidades. Este procedimento cessará quando A atinjir o valor 10.

Note que, como a cada repetição mais de uma instrução deve ser executada (adicionar duas unidades a A e mostrar mensagem), então foi-se necessário delimitar estas ações com os comandos **Begin** e **End**. Observe também o ponto-e-vírgula ao final de cada instrução.

Repeat

"O loop *while* pode não ser executado nenhuma vez, caso a condição de teste não seja satisfeita, porque o programa nem entra no loop. Quando se quer obrigar a execução do procedimento de loop ao menos uma vez e testar a condição ao final, pode-se usar o **repeat**"[5]. Observe o exemplo a seguir:

```
K:=0
Repeat
 A:=A*3;
```

[4] *Loop*: repetição seqüencial de comandos
[5] SONINNO, 2000

```
K:=K+1;
Until K > 10;
```

O que ocorrerá no código acima será o seguinte: K, inicialmente, vale 0 (zero). O comando **repeat** obriga a execução, ao menos uma vez, da operação indicada - no caso, multiplicar o valor de A por 3 e atribuí-lo a A. A cada repetição, K tem uma unidade adicionada ao seu valor e, quando este atingir um número maior que 10, a execução é encerrada.

Procedures e Functions

Rotinas são seqüências de códigos cuja finalidade é executar alguma tarefa. Podem ser divididas em dois tipos:

- "**Function (função)** - pode receber opcionalmente um ou mais parâmetros e devolver à rotina chamadora um valor, que pode ser utilizado em qualquer expressão.

- **Procedure (procedimento)** - esse tipo de rotina pode receber opcionalmente um ou mais parâmetros, mas não pode retornar nenhum valor à rotina chamadora"[6].

"Uma função, em Delphi, seja ela interna à própria linguagem ou desenvolvida pelo usuário, é como sua similar na matemática: ela opera nos valores passados como parâmetros (se houver algum) e devolve um novo valor.

Para declararmos uma função, devemos utilizar a palavra-chave *Function*. Seguindo essa palavra-chave vem o nome da função propriamente dita e, opcionalmente, uma lista de parâmetros entre parênteses. Ao final da declaração da função, temos o tipo do dado a ser retornado, separado por dois pontos (:)"[7]. A seguir, um exemplo de função:

```
Function MultiplDez (n: Integer): Integer;
Begin
  MultiplDez := n *10;
End;
```

A função acima tem por objetivo multiplicar qualquer número informado por 10. Ela recebe um parâmetro inteiro e devolve o resultado à função.

O número passado fica "guardado" na variável n, que é multiplicada por 10 e seu resultado atribuído ao nome da função novamente (o que ocorre na linha *MultiplDez := n *10*).

Para chamar a função e passar um número a ela, no corpo do seu programa, digite algo semelhante a:

```
MultiplDez(2);
```

[6] ALVES, 1997
[7] ALVES, 1997

O resultado disso, 20, será retornado à função.

"Um procedimento nada mais é que uma rotina desenvolvida especificamente para executar uma determinada tarefa, que pode ser genérica ou não. Ele não retorna qualquer valor à rotina chamadora.

Você declara um procedimento através da palavra-chave **Procedure,** seguida pelo nome do procedimento e um par de parênteses, que pode conter ou não uma lista de parâmetros. Para finalizar o corpo do procedimento, utilize a palavra-chave **End**"[8]. Na seqüência, um exemplo de *procedure*:

```
Procedure MultiplDez (n: Integer);
Begin
 n:= n *10
End;
```

Note que esta procedure faz a mesma coisa que a função anterior; no entanto, precisamos usar a variável *n* - poderia ser outra - para capturar o resultado (através da expressão *n:=n*10*), já que procedures não retornam valores, o que invalidaria o uso de algo como `MultiplDez := n * 10`, como foi usado na função.

Para passar um valor à procedure, chame-a no corpo do programa dessa forma:

```
MultiplDez (3);
```

O número 3 passado à função será multiplicado por 10 e *ShowMessage* exibirá *30* na tela.

Conversão de Valores no Delphi

Observe a procedure a seguir:

```
Procedure MultiplDez (n: Integer);
Begin
 n:= n *10
 ShowMessage(InttoStr(n));
End;
```

O procedimento acima usa o **ShowMessage** (que é uma procedure interna do Delphi) para exibir o resultado de *n* após a multiplicação *n * 10*. Sabe-se que *n* é um número inteiro que o usuário passará à procedure quando quiser executá-la.

Mas note a existência de um "comando" diferente junto ao *ShowMessage*: **InttoStr**. Vamos entender isso agora.

[8] ALVES, 1997

Alguns comandos no Delphi são tipicamente *strings*; outros, inteiros, e assim por diante. *ShowMessage* é um comando (uma procedure pronta do Delphi, na verdade) que aceita que apenas *strings* sejam passadas a ela. Como a intenção era exibir o número inteiro resultante da multiplicação, foi-se necessário usar a função **InttoStr,** que converte valores numéricos inteiros em strings. Isso tornou possível a exibição do resultado.

O nome desta função (**InttoStr)** é uma abreviação de **Integer to String**. Conversões como esta são muito comuns nos programas Delphi. Alguns objetos, como um **Edit** (veja Figura 16), também apenas trabalham com strings.

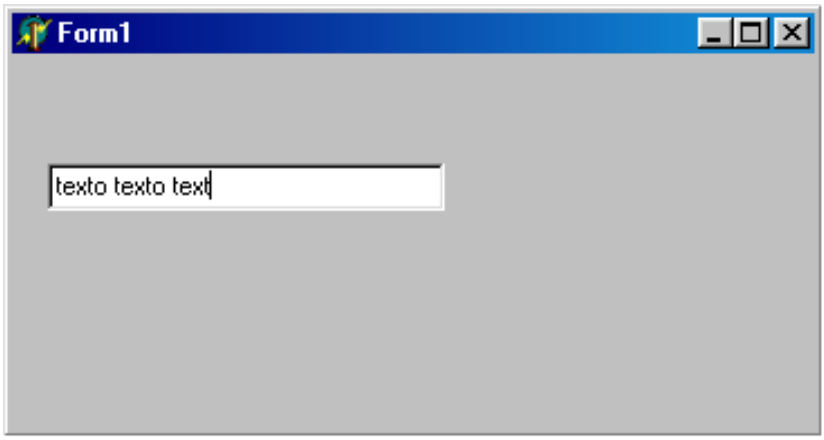


Figura 16 - Um campo *Edit*

Atenção: quando dizemos que o campo **Edit** aceita apenas textos, não significa que você não possa digitar números ali; significa apenas que, para o Delphi, aquele número será tratado como string e não como variável numérica. Isso implica dizer que, se você quiser realizar uma operação matemática com o número ali digitado, uma conversão será necessária. A conversão de *String para Inteiro* é feita usando-se o comando **StrtoInt**. Portanto, se houvéssemos digitado um número no campo *Edit1* e quiséssemos passá-lo a uma variável inteira *I*, a linha de comando seria essa:

```
I:=StrtoInt(Edit1.Text);
```

Isto significaria que estaríamos pegando o conteúdo do campo *Edit1* e passando à variável **i**. "*Text*", neste caso, é uma propriedade do componente Edit que armazena seu conteúdo. Esta propriedade consta da lista de **Propriedades** do Object Inspector para este e outros componentes semelhantes.

Desenvolvendo a Segunda Aplicação

Para desenvolver seu segundo aplicativo, supondo que o anterior ainda esteja na tela, feche-o primeiro clicando no menu **File** e, depois, em **Close All**. Inicie um novo programa clicando em **File**, depois em **New > Application**. Surgirá um formulário novo e uma nova Unit.

Vamos adicionar alguns componentes ao nosso formulário. Se você tiver dificuldades em encontrar os componentes aqui mencionados, pare com o mouse sobre um deles, um por vez, e o próprio Delphi lhe dará a descrição do mesmo.

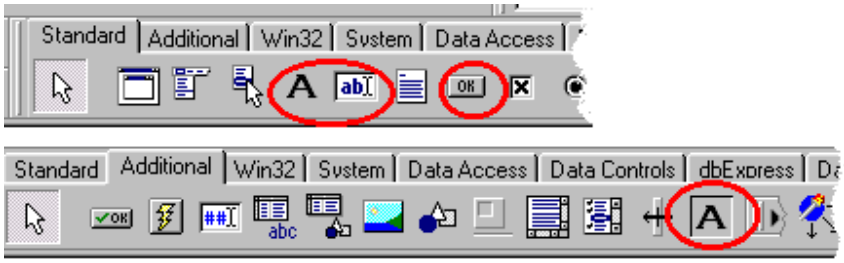


Figura 17 – Componentes a utilizar: Label, Edit, Button (Standard) e StaticText (Additional)

Da paleta **Standard**, coloque um componente **Label**, um **Edit** e dois **Buttons**. Da paleta **Additional**, coloque um **StaticText** (Figura 17). Ajuste-os em seu Form para que fiquem semelhantes ao mostrado na Figura 18.

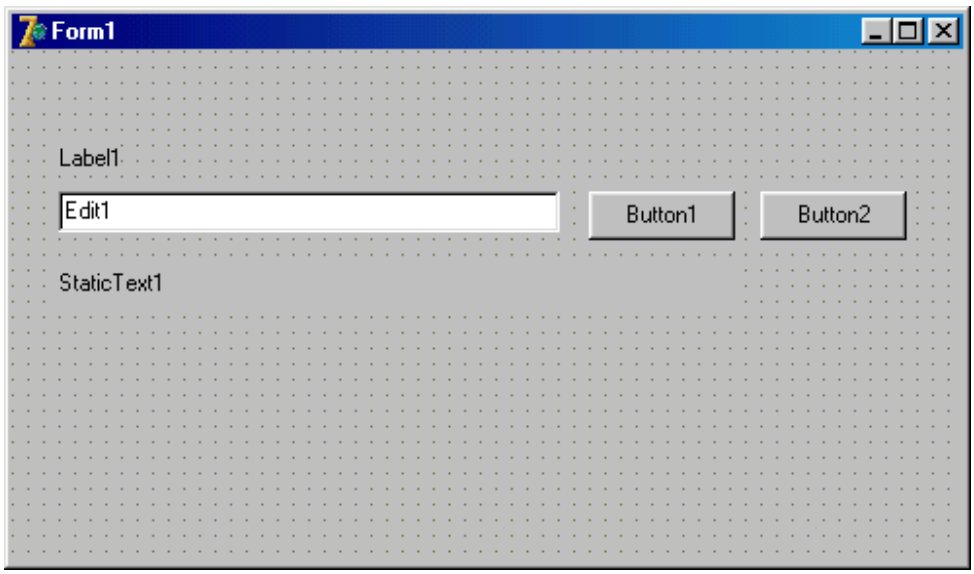


Figura 18 -Segundo aplicativo

Este nosso primeiro aplicativo servirá para multiplicar qualquer número inserido no **Edit** (chamado, por padrão, **Edit1**) por **oito** (**8**). É uma aplicação básica, apenas para que você se acostume com alguns comandos e componentes.

Vamos, antes de iniciar a codificação, alterar algumas propriedades destes componentes. O nosso **Label** não deve exibir, por exemplo, o texto **Label1** no formulário. Por padrão, cada componente adicionado assume como nome a classe a qual ele pertence (no caso, ele é um Label, que serve para exibir textos descritivos) mais um número (aqui, número 1, pois é o primeiro Label que colocamos no Form). Se adicionássemos mais um Label, ele assumiria o nome de Label2, e assim por diante. Também por padrão, os componentes que exibem textos também mostram, em sua propriedade Caption (ou propriedade Text, no caso do Edit), o nome do componente. É por isso que vemos na tela o texto **Edit1, Label1** e **StaticText1**. Estas propriedades serão alteradas por nós no devido momento. Comecemos por modificador a propriedade **Caption** de **Label1**.

É de se esperar que tenhamos na propriedade **Caption** um pequeno texto explicativo que diga ao usuário como utilizar o software que estamos criando. Um texto mais adequado seria **"Digite abaixo um número e pressione OK para multiplicá-lo por Oito"** (sem as aspas). Faça isto então.

Selecione o componente **Edit1**. A propriedade **Text** do **Edit1** não deverá conter valor algum, já que será o usuário quem informará um número. Neste caso, selecione esta propriedade e apague o texto, usando as teclas **DEL** ou **BACKSPACE** de seu teclado.

Selecione, agora, o componente **Button1** e mude seu nome (propriedade **Name**) para **BtnOK**. Mude seu **Caption** para **OK**. Selecione, depois, **Button2** e mude seu nome para **BtnSair** e seu **Caption** para **Sair**.

Por fim, selecione **StaticText1** (que será o local onde o resultado de nossa multiplicação aparecerá) e mude a propriedade **AutoSize** para **FALSE** - isso fará com que o componente tenha largura fixa e que o mesmo não seja redimensionado sempre que o resultado de uma multiplicação for exibido. Mude também a propriedade **BorderStyle** para **bsSunken** – isso fará uma pequena borda aparecer em volta do **StaticText1**. E, por fim, deixe seu **Caption** em branco, bastando apagar o texto que ali se encontra.

É melhor também reajustar a largura e altura de seu formulário, para que não fique tão grande em relação aos componentes nele inseridos. Deixe-o com 131 pixels de altura e 465 pixels de largura - basta, para isso, alterar os valores das propriedades **Height** (altura) e **Width** (largura). Reajuste a disposição dos componentes, se preciso. Seu aplicativo deve estar semelhante ao mostrado na Figura 19.

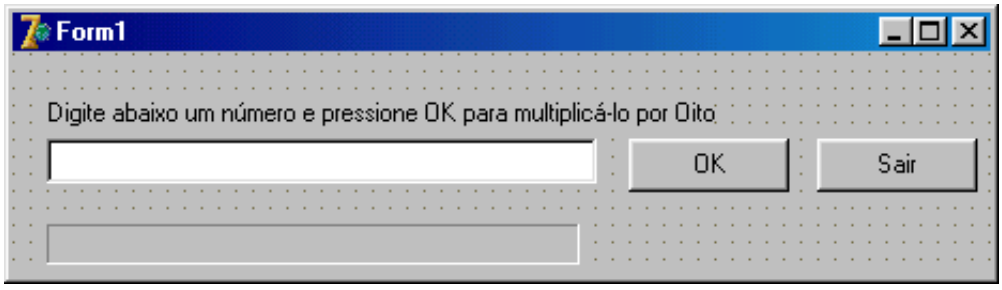


Figura 19 - Aparência final do aplicativo

É conveniente também mudar o título que aparece na **Barra de Títulos** de seu aplicativo: "Form1" não é um nome agradável para nenhuma aplicação. Selecione a propriedade **Caption** do formulário e digite qualquer coisa mais elucidativa, como "**Multiplicação por Oito"** (sem as aspas).

Antes de continuar, é conveniente salvar o trabalho já feito. Para salvar tanto o formulário quanto a Unit, acione **File > Save All** - você também poderá usar o botão Save All, da **Speed Bar** do Delphi (Figura 20). Como ainda não demos nome nem para a Unit e nem para nosso arquivo de projeto *(*.dpr*), o Delphi nos pede que isso seja feito agora.

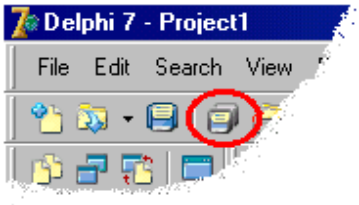


Figura 20 - Botão Save All (Delphi 7)

Nosso aplicativo é pequeno e provavelmente terá apenas uma Unit; então, aceite a sugestão do Delphi e salve a Unit como **Unit1.pas**. O formulário, no entanto, deve ter o nome do executável. Salve-o como **Mult8.dpr**. Assim, nosso aplicativo, depois de compilado, terá o nome **Mult8.exe**.

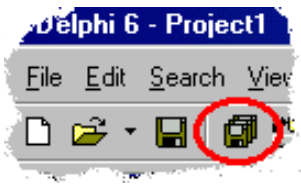


Figura 21 - Botão Save All (Delphi 6)

Agora vamos codificar nosso aplicativo. Quando o botão **OK** for clicado, o que deverá ser feito? O valor contido em **Edit1** deverá ser multiplicado por 8 e exibido no campo **StaticText1**. Vamos fazer isso então.

Para que o nosso aplicativo responda ao clique do botão **OK**, devemos editar o evento **OnClick** do botão. Você pode fazer isso selecionando o **BtnOK** e, no **Object Inspector**, na guia **Events**, efetuando dois cliques na área em branco logo à frente do evento **OnClick**. Ou pode, simplesmente, clicar duas vezes diretamente sobre o botão **BtnOK**. Uma procedure para o evento **OnClick** será gerada automaticamente.

Temos, então, o seguinte código para o evento OnClick até agora:

```
procedure TForm1.BtnOKClick(Sender: TObject);
begin

end;
```

Acrescente códigos à procedure para que a mesma fique conforme abaixo:

```
procedure TForm1.BtnOKClick(Sender: TObject);
begin

 StaticText1.Caption := Edit1.Text * 8 ;

end;
```

Entenda o que foi feito. Estamos pegando o número contido no **Edit1** (para isso, estamos lendo a propriedade **Text** do Edit) e multiplicando por 8, e já atribuindo, ao mesmo tempo, o resultado a **StaticText1**, em sua propriedade **Caption** - propriedade que irá nos mostrar na tela o resultado.

No entanto, se você pressionar CTRL+F9 para compilar o software, o Delphi apontará um erro:

```
[Error] Unit5.pas(32): Incompatible types: 'String' and 'Integer'.
```

Isso significa que estamos tentando realizar uma operação de multiplicação utilizando tipos que não são compatíveis: **string** e **inteiro** (integer). Mas onde está a incompatibilidade? Lembra-se de que já vimos que, algumas propriedades são, por si só, strings, outras integer, etc? Pois bem, **Edit1.Text** e **StaticText1.Caption** fazem referência a propriedades puramente texto dos objetos Edit e StaticText, respectivamente. É impossível, portanto, atribuir-lhes uma operação matemática qualquer que seja! Para que o código mencionado acima funcione, é necessário fazer a conversão entres os tipos. Assim:

```
procedure TForm1.BtnOKClick(Sender: TObject);
begin

 StaticText1.Caption := StrtoInt(Edit1.Text) * 8 ;

end;
```

Com o código acima, estamos convertendo o valor contido em **Edit1.Text** (que, até então, embora o usuário deva ter digitado um número, é considerado puro texto para o Delphi) em inteiro através da função **StrtoInt**. Portanto, agora estamos realizando uma multiplicação somente com números.

Mas há outro detalhe: ao atribuir o resultado a **StaticText**, para que o mesmo possa ser exibido na tela por ele, é preciso torná-lo texto de novo (para que a propriedade **Caption** possa armazenar o resultado em formato texto).

Altere, então, novamente o código e deixe-o conforme segue:

```
procedure TForm1.BtnOKClick(Sender: TObject);
begin

 StaticText1.Caption := InttoStr(StrtoInt(Edit1.Text) * 8) ;

end;
```

Com isso, estamos dizendo que todo o conteúdo da multiplicação deve ser convertido de **Inteiro** para **String** de novo – para isso, usamos **InttoStr**. Pressione agora **CTRL+F9** e compile seu aplicativo. Para executá-lo, pressione **F9** e faça o teste. Informe um número qualquer e pressione **OK**. O resultado correto aparecerá no campo próximo do rodapé do aplicativo. Ainda é preciso codificar o botão **Sair**. Dê dois cliques sobre ele para editar o evento **OnClick** do botão **BtnSair**. Implemente o código abaixo:

```
procedure TForm1.BtnSairClick(Sender: TObject);
begin
 Form1.Close ;
end;
```

Com **Form1.Close,** estamos dando ordens para que o formulário seja fechado quando receber um clique no botão Sair.

Note que foi feita uma referência ao nome do formulário: **Form1.Close**. Isso porque a **propriedade Name** deste Form (se você manteve o nome proposto pelo Delphi) **é Form1**. Portanto, estamos dizendo para que seja fechado o **Form1** e não outro formulário qualquer. Como

só há um Form em nossa aplicação, poderíamos omitir a palavra Form1 e deixarmos nosso código apenas assim:

```
procedure TForm1.BtnSairClick(Sender: TObject);
begin
  Close ;
end;
```

O código acima teria o mesmo efeito. Não apenas porque temos só um formulário, mas porque o botão Sair **faz parte de Form1** – por isso é possível omití-lo. Já se quiséssemos fechar um segundo formulário a partir de Form1, então sim, seria necessário o caminho completo: **Form2.Close**. Com isso, Form1 fecharia Form2.

Por outro lado, se você alterar o nome do Form de **Form1** para **FrmPrincipal**, seria necessário modificar a linha **Form1.Close** para **FrmPrincipal.Close** – ou simplesmente omitir o seu nome, deixando apenas **Close** (conforme **Erro! Fonte de referência não encontrada.**).

Caso você tenha mais de um formulário em seu aplicativo, recomenda-se, então, a utilização de **Application.Terminate** no lugar de *close*.

Usando o Application.MessageBox

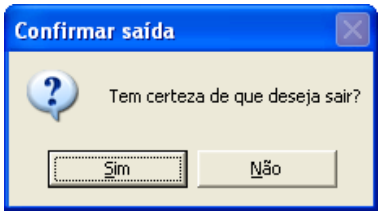


Figura 22 - Confirmação de saída

É comum que os softwares, ao receberem um clique no botão "Sair" perguntem ao usuário se ele realmente deseja fazer isso. Você pode obter esse recurso usando a função Application.MessageBox, conforme mostrado abaixo:

```
procedure TForm1.BtnSairClick(Sender: TObject);
begin

  if application.messagebox('Tem certeza de que deseja
sair?','Confirmação de saída',36) = IdYes then
 Close ;

end;
```

Sempre que se clica no botão "Sim", o Windows envia um mensagem do tipo IdYes ao aplicativo; quando se clica em "Não", envia-se uma mensagem IdNo. Portanto, se compararmos o resultado da função Application.Messagebox com IdYes, saberemos se esse botão foi clicado pelo usuário e, então, poderemos pedir ao formulário que seja fechado.

Eliminando o Delphi de Fundo

Você deve ter notado que, enquanto executa seu software, os menus e janelas do Delphi permanecem ao fundo da tela. É possível fazer com que o Delphi "desapareça" durante a execução de seu aplicativo, permitindo que você tenha uma visão melhor do mesmo. Páre a execução de seu aplicativo e vá ao menu **Tools** do Delphi. Nele, escolha **Environment Options**.

Na guia **Preferences**, procure pela opção que diz **Minimize On Run.** Habilite-a, clicando uma vez sobre ela.

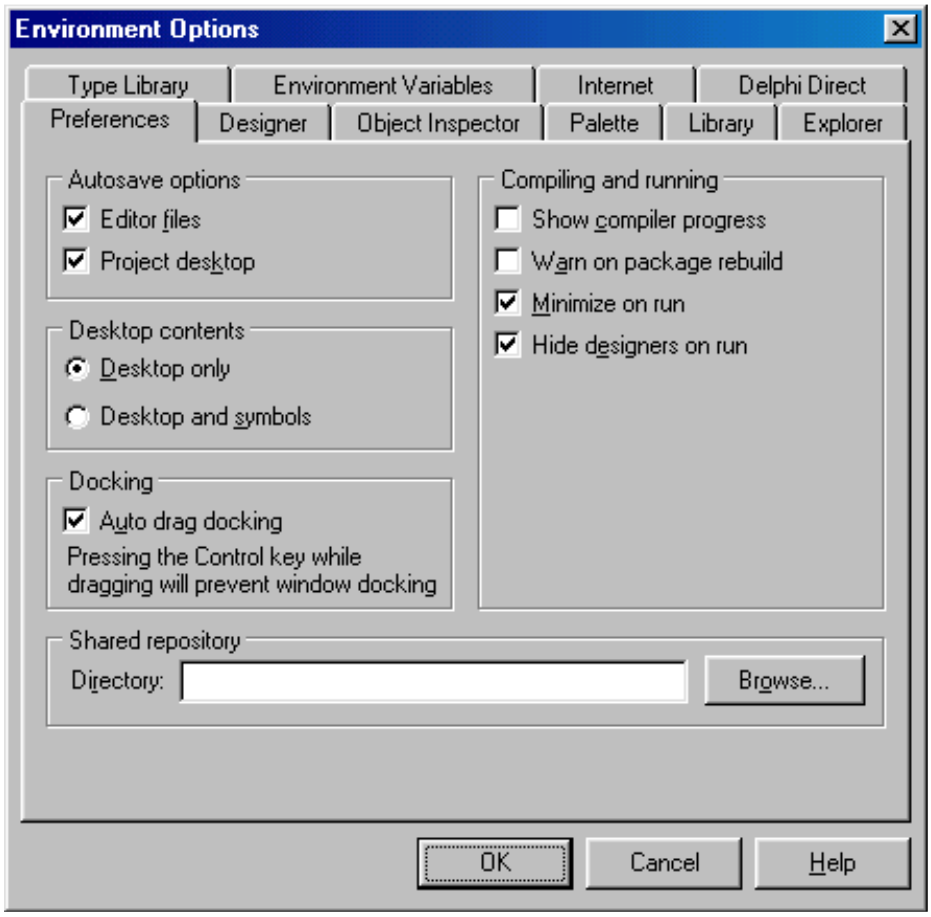


Figura 23 - Minimizar o Delphi durante a execução do aplicativo

Pressione **OK** e retorne ao seu aplicativo. Pressione F9 novamente e sinta a diferença. O Delphi permanece minimizado durante a execução de seu projeto. Apenas quando você o interrompe o Delphi retorna à tela.

Naquele mesmo item de menu (Tools > Environment Options > Preferences) há outras opções interessantes. Você já deve ter notado que o Delphi, ao ser iniciado, sempre abre um projeto novo automaticamente. Mas você pode solicitar a ele que abra sempre o último projeto com o qual você trabalhou.Para isso, marque a opção onde lê-se **Project Desktop**. Esta opção salva também o arranjo das janelas e a disposição das mesmas. Se, além disso, você desejar que sempre que seu projeto for executado (através de **Run > Run** ou via **F9**) o Delphi salve as modificações feitas no código-fonte, basta marcar também a opção **Editor Files** (recomendável).

Desenvolvendo a Terceira Aplicação

Vamos criar agora outro aplicativo. Se você ainda não salvou o aplicativo anterior, salve-o agora acionando **File > Save All**. Crie um novo aplicativo, acionando **File > New > Application**. Surge na tela um Form e, atrás, uma Unit. Coloque no Form, da paleta **Standard**, um componente **Edit**, um **Memo** e, da paleta **Additional**, um botão **BitBtn** - ele é o primeiro componente da esquerda para a direita. Ajuste os componentes para que o formulário se pareça com o mostrado a seguir:

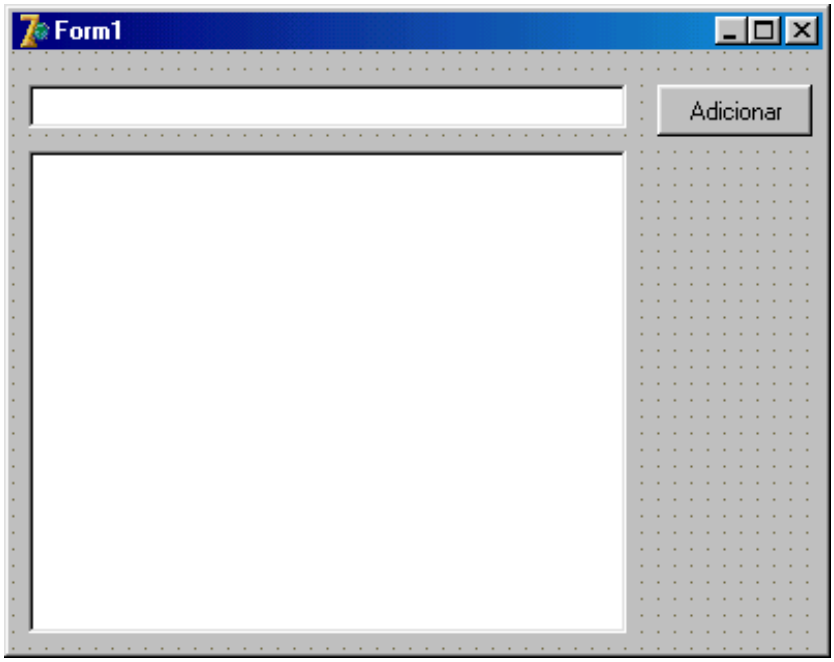


Figura 24 - Terceiro aplicativo

Limpe o texto contido em **Edit1.Text** (ou seja, propriedade **Text** do **Edit1**); edite a propriedade **Caption** do botão **BitBtn1** e mude-a para **Adicionar**. Mude também o nome do botão, de **BitBtn1**, para **BtnAdicionar.** E, no **Memo**, procure a propriedade **ReadOnly** e deixe-a como **TRUE** – isso impedirá que se digite diretamente no campo Memo. Procure também a propriedade **Lines**, selecionando o valor **TStrings** (logo à frente) e clique no botão **...** . Na janela que se abre, vá teclando BACKSPACE em seu teclado até apagar a linha contendo as strings de **Memo1**. Pressione **OK** para retornar ao formulário.

O objetivo desta aplicação é que o usuário digite qualquer coisa no campo **Edit** e, ao pressionar **Adicionar**, o texto digitado seja acrescentado ao **Memo1**. Como o usuário deverá usar o botão Adicionar para colocar textos no Memo1, tivemos que ativar a propriedade **ReadOnly** (deixando-a como TRUE). Se deixássemos como FALSE, o aplicativo perderia sua função, já que o texto

poderia ser editado diretamente no Memo, sem a necessidade de se passar pelo Edit e pelo botão Adicionar.

Façamos nossa codificação agora: dê dois cliques no botão **Adicionar** (ou, na guia Events, dê dois cliques no evento **OnClick**) para que tenhamos acesso ao evento **OnClick**. Lá, digite o seguinte:

```
procedure TForm1.BtnAdicionarClick(Sender: TObject);
begin
 Memo1.Lines.add(Edit1.Text);
end;
```

Note que bastou uma linha para que o software ganhasse funcionalidade. Na linha acima, estamos dizendo que queremos adicionar uma linha em Memo1 (*Memo1.Lines.add*) e que a mesma deverá conter o texto digitado em **Edit1.Text**. Um Memo, ao contrário de Labels e Edits, podem ter mais de uma linha. Por isso, sempre que quisermos adicionar uma linha, devemos usar **Lines.Add**.

Caso desejássemos apenas adicionar o texto, sempre continuando na mesma linha – e deixando a quebra de linhas por conta do próprio Memo -, o código poderia ser este:

```
procedure TForm1.BtnAdicionarClick(Sender: TObject);
begin
 Memo1.Text := Memo1.Text + Edit1.Text ;
end;
```

Isso faz com que apenas adicionemos o texto do Edit ao Memo, sem perder o texto já adicionado anteriormente (por isso Memo1.Text deve ser igual a ele mesmo mais o conteúdo de Edit1.Text). Se, ao invés de **`Memo1.Text := Memo1.Text + Edit1.Text`** tivéssemos digitado **`Memo1.Text := Edit1.Text`**, o texto adicionado anteriormente sempre se perderia.

Salve agora seu aplicativo: use File > Save All e dê nomes à Unit (se desejar mudar o nome sugerido) e ao aplicativo - algo como **Adiciona.dpr**. Execute-o pressionando **F9**. Ao fazer isso, você estará compilando e, logo em seguida, rodando seu software. Neste momento, o Delphi gerou um arquivo chamado **Adiciona.Exe**, que contém seu aplicativo.

O Botão BitBtn

O botão BitBtn tem características interessantes quando comparado ao seu semelhante Button. Se você observar o Object Inspector cuidadosamente, verá, ao menos, duas propriedades que valem a pena conhecer: **Kind** e **Glyph**. Kind serve para indicar um tipo pré-definido para o botão. Por exemplo, se desejo que meu botão seja um botão de OK, basta escolher o valor bkOK e teremos uma tela como a seguinte:

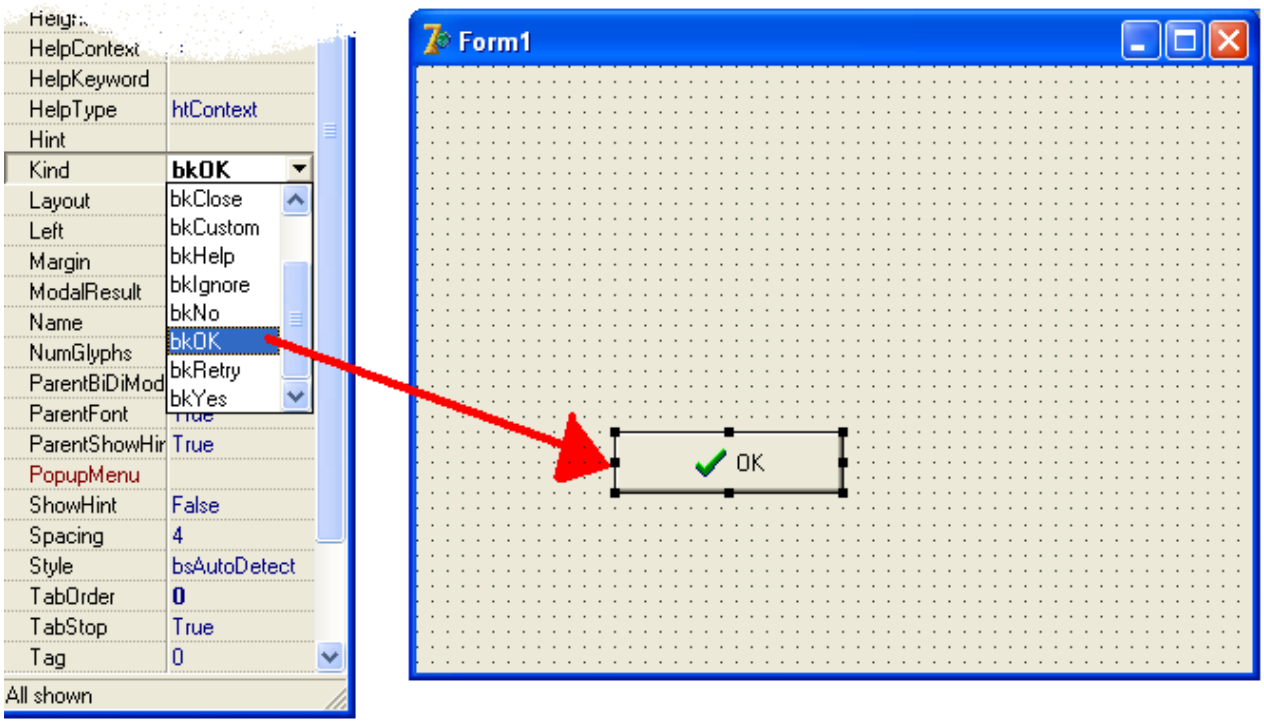


Figura 25 - Botão OK

Embora o Caption do botão já assuma o texto "OK", você poderá alterá-lo para o texto que quiser. Também poderá modificar a pequena figura ("Glyph") que aparece à esquerda, selecionando a propriedade **Glyph** e, depois, clicando em … . Uma nova caixa de diálogo irá aparecer. Clique em **Load.** Navegue pelas pastas do sistema até encontrar a pasta **Arquivos Comuns**, normalmente localizada em **C:\Arquivos de Programas\Arquivos Comuns.** Nela, acesse **Borland Shared \ Images \ Buttons.** Ali, você terá uma grande quantidade de pequenas imagens para ilustrar seu botão.

Usando um Timer

Vamos criar agora aplicativo que utiliza um **Timer**. O objetivo do Timer é contar os tic-tacs do relógio do sistema a intervalos pré-determinados pelo usuário.

Por padrão, conta-se a cada 1.000 milésimos de segundo (ou 1 segundo). Para o exemplo a seguir foi adicionado um Timer (paleta **System**) ao formulário e uma **StatusBar** (Paleta **Win32**). O aplicativo irá mostrar na StatusBar (que é uma barra de status típica de aplicativos Windows). A barra de status é feita de pequenos painéis (ou Panels), conforme ilustra a figura a seguir.

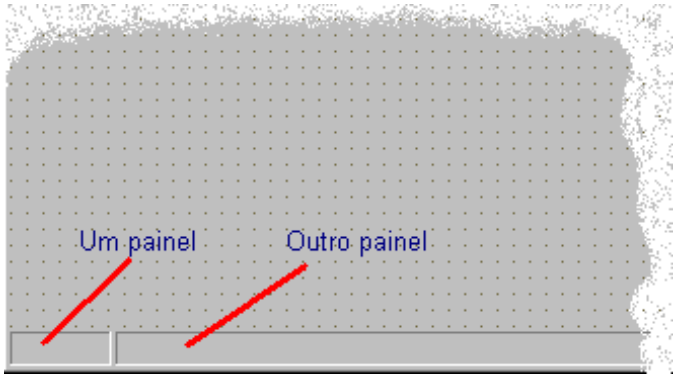


Figura 26 - Painéis da Barra de Status (StatusBar)

Para que estes painéis apareçam na StatusBar, é necessário adicioná-los.

Dê um clique duplo no componente e, na pequena caixa que se abre, clique duas vezes no **botão Add New** (o mesmo que aparece circulado na Figura 27).

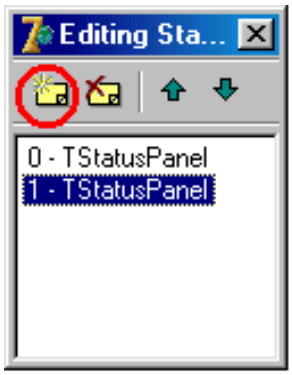


Figura 27 - Botão Add New em destaque

Dois objetos do tipo **TStatusPanel** serão criados. Clique no primeiro e, no **Object Inspector**, mude a propriedade **Width** do Painel para 200. Isso irá aumentar sua largura para que os dados possam caber corretamente.

Todo o restante do trabalho será feito no evento OnTimer do componente Timer. Para criar este evento, dê um clique duplo sobre o Timer e digite os códigos a seguir:

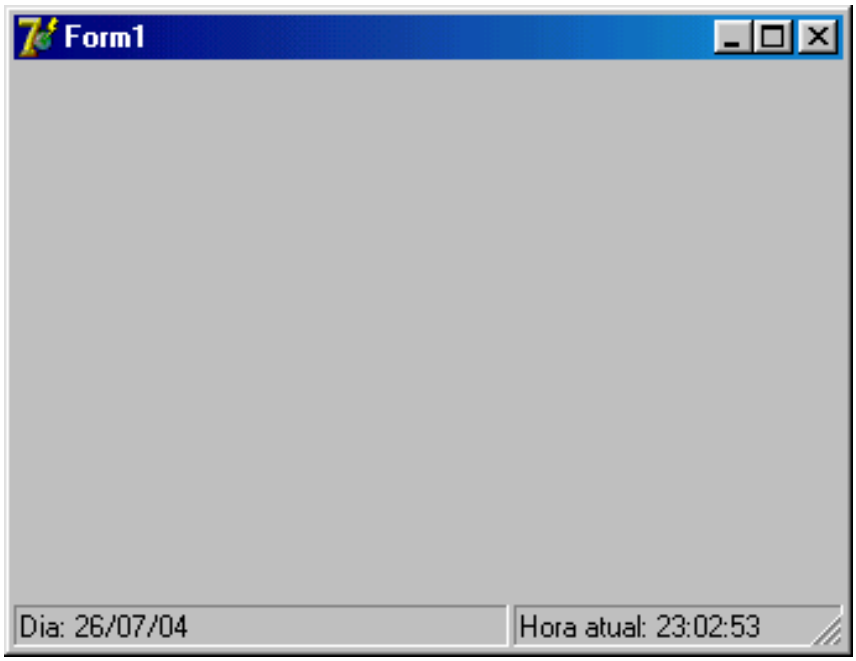
```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
 StatusBar1.Panels[0].Text := DatetoStr(Date);
 StatusBar1.Panels[1].Text := TimetoStr(Time)
end;
```

O procedimento acima mostrará a data no primeiro painel e a hora no segundo. Note que para acessar os painéis mencionamos cada um deles como elementos de uma matriz. A propriedade **Panels** contém todos os painéis e, assim, o **painel 0** (zero) é o primeiro, **1** é o segundo, etc. E, para cada um deles, usamos a propriedade **Text,** que irá mostrar o texto com a data e a hora.

Poderíamos melhorar o exemplo acima da seguinte forma:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
 StatusBar1.Panels[0].Text := 'Dia: ' +DatetoStr(Date);
 StatusBar1.Panels[1].Text := 'Hora atual: ' +TimetoStr(Time)
end;
```

Colocando os textos '**Dia:'** e **'Hora atual:'** antes de cada função (**Time** e **Date** são funções do Delphi que capturam a data e hora atual do sistema) faremos com que o Delphi exiba este texto antes da data e da hora. O resultado deste exemplo é mostrado abaixo:



Chamando Um Aplicativo Externo

Para executar programas externos à nossa aplicação, pode-se utilizar a função WinExec. Vamos supor que desejamos executar a calculadora do Windows. Para tanto, basta proceder assim:

```
WinExec('calc.exe',SW_SHOWNORMAL);
```

Note que devemos informar o nome do executável em primeiro lugar e, depois, a maneira como a janela do aplicativo deverá aparecer. SW_SHOWNORMAL indica que a calculadora será chamada e executada com sua janela nas dimensões padrão do aplicativo. Mas há outras opções que você pode utilizar:

- SW_SHOWMINIMIZED ➔ indica que aplicativo chamado deverá ser executado minimizado.
- SW_SHOWMAXIMIZED ➔ indica que aplicativo chamado deverá ser executado maximizado.
- SW_SHOWNORMAL ➔ exibe o aplicativo chamado com seu tamanho de janela normal.

Há casos, no entanto, em que apenas informar o executável não será suficiente. Para chamar o Internet Explorer, por exemplo, é necessário informar o caminho completo de pastas nas quais ele se encontra, até o executável. Ficaria assim:

```
WinExec('c:\Arquivos de Programas\Internet
Explorer\iexplore.exe',SW_SHOWNORMAL);
```

Também é possível executar o Internet Explorer fazendo-o abrir já numa página específica:

```
WinExec('c:\Arquivos de Programas\Internet Explorer\iexplore.exe
www.terra.com.br',SW_SHOWNORMAL);
```

O comando acima obriga o Internet Explorer a ser carregado já com o site do Terra. No entanto, há um inconveniente relacionado ao WinExec: a própria Microsoft não recomenda sua utilização – provavelmente prevendo que o mesmo deverá ser retirado das futuras bibliotecas do Windows. Em seu lugar, a Microsoft recomenda o uso de **ShellExecute** ou de **CreateProcess**. Vamos usar aqui, então, o ShellExecute:

```
ShellExecute(Handle, 'open', 'calc.exe',nil,nil,SW_SHOWNORMAL) ;
```

A função acima também abre a calculadora. Apenas é necessário entendermos melhor os parâmetros da função:

- **Handle** ➔ é um manipulador de janelas. O Windows precisa saber que programa (ou que janela) chamou o aplicativo da calculadora. Passando *Handle* como parâmetro estamos indicando que o nosso aplicativo é quem fez o chamado.

- **Open** ➔ indica o que desejamos fazer com o aplicativo chamado. Neste caso, desejamos que ele seja "aberto" - daí o uso de *open*.

- **Calc.exe** ➔ indica o nome do programa a ser executado.

- **nil** ➔ Os dois parâmetros seguintes são parâmetros adicionais e que não são realmente necessários no momento. Para ignorá-los, passamos **nil** para ambos.

- **SW_SHOWNORMAL** ➔ O último parâmetro refere-se ao tamanho da janela do aplicativo, do mesmo modo como é usado na função *WinExec*.

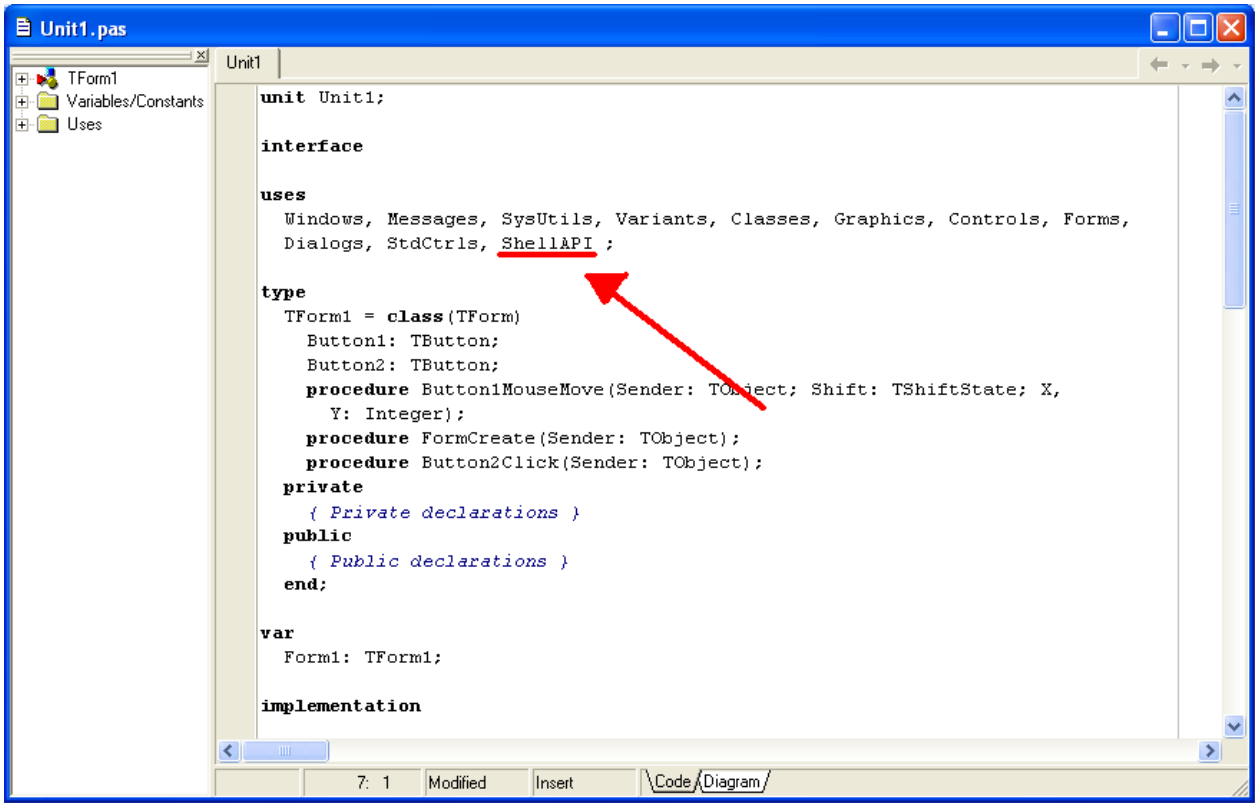


Figura 28 - Unit 'ShellAP' adicionada manualmente

O único detalhe aqui é que, para usarmos a função ShellExecute, precisamos de uma "biblioteca" (que, na verdade, é uma unit também) do Delphi chamada *ShellAPI*. Para adicioná-la ao seu programa, basta localizar a cláusula **uses** da sua **Unit** e proceder conforme mostrado abaixo:

Há outras vantagens do comando ShellExecute em relação ao WinExec. Você pode abrir um arquivo chamando-o diretamente, sem precisar especificar a qual programa do Windows ele está associado. Por exemplo, o código abaixo abre diretamente um site na Internet com seu navegador-padrão:

```
ShellExecute(Handle, 'open', 'www.terra.com.br',nil,nil,SW_SHOWNORMAL)
```

Outro exemplo:

```
ShellExecute(Handle, 'open', 'c:\Carta.doc',nil,nil,SW_SHOWNORMAL) ;
```

O código acima abre o Microsoft Word já com o arquivo **Carta.doc** carregado na tela.

Referência Bibliográfica

(*ALVES, 1997*) ALVES, Willian Pereira, "Delphi 3 - Curso Prático", Editora Érica, São Paulo-SP, 1997.

(*SONINNO, 2000*) SONINNO, Bruno, "Desenvolvendo Aplicações com Delphi 5", Makron Books, São Paulo-SP, 2000.