



# Código Avançado

---

As seções seguintes descrevem alguns tópicos de programação avançada em Delphi e em Visual Basic para que você possa compará-los

- [Units](#)
- [Escopo \(Alcance\) de variáveis](#)
- [Desvio Condicional](#)
- [Loops](#)
- [Manipulação de String](#)
- [Arrays](#)
- [Procedures e Funções](#)
- [Arrays de Controles](#) Support
- [Object Variables](#) Support

## Units

Enquanto existem várias diferenças, as Units do Delphi são funcionalmente equivalente aos módulos no Visual Basic. Representam a unidade fundamental do código. Em VB, existe um módulo implícito associado com cada form e a habilidade de adicionar módulos em seu

projeto contendo procedures, declarações DLL e constantes e variáveis globais.

Em Delphi existe uma Unit explícita associada com cada form que contém todo o código associado com o form, incluindo as definições de classes. Isso explica porque quando você salva um projeto em Delphi, você é avisado para dar um nome a unit (\*.PAS). O form é salvo com o mesmo nome e extensão diferente (\*.DFM). Também, é possível fazer uso de várias units adicionais pelo uso da declaração uses. Diferente do Visual Basic, onde os módulos devem ser carregados dentro do projeto e então estarão completamente disponíveis para a aplicação, a disponibilidade dos recursos de uma unit em Delphi não obriga que ela seja parte do projeto - uma grande flexibilidade! Você pode acessar uma unit por outra através da inclusão da primeira na declaração uses da segunda. Como você pode ver no form onde está o código existe uma "aba" escrito "Unit1" este é o nome default para a unit. Se você clicar nesta "aba", você verá o código que já existe na unit. Na décima quinta linha existe a declaração uses com uma lista com várias unist separadas por vírgulas:

uses

SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,  
Controls,

Forms, Dialogs;

Estas são as definições default das units o qual está disponível em todo form novo. Você não precisa adicionar nenhuma destas linhas de código em seu projeto; o Delphi faz isso para você. A presença destas declarações na declaração uses faz com que elas se tornem funcionalmente disponíveis para o form. Algumas delas são importantes. Por exemplo, WinProcs e WinTypes adicionam todas as funções cruciais da API Windows. Algumas dessas como Classes, Graphics, Controls e Forms são requeridas pelo Delphi para fazer os forms trabalharem com sua aplicação. Você deve remover units desta lista default com cautela. Na maioria dos casos, você não precisará alterar esta lista porque se a funcionalidade de uma destas units não for usada, o código desta unit não será linkado a sua aplicação. Se você deseja utilizar a funcionalidade da *Unit1* dentro da *Unit2*, você precisa incluir *Unit1* na declaração uses da *Unit2*.

Como os módulos, as units podem conter procedures, declarações DLL,

tipos definidos por usuário e variáveis e constantes globais. Existem duas seções numa unit, a interface e a implementation. A interface é uma área "public" e a funcionalidade definida nesta área é disponível por toda aplicação onde a unit estiver listada na declaração uses. A implementation é uma porção da unit definida como área "private" então variáveis, constantes e funções definidas *somente* estarão disponíveis na unit. Isto será discutido mais adiante em discussão sobre scopo de variáveis e procedures abaixo:

Então, lembre-se que adicionar uma unit ao projeto é somente uma conveniência para edição e compilação. Se você quer acessar a funcionalidade de uma unit, você deve colocá-la dentro da declaração uses em seu código.

## Escopo (alcance) de variáveis

Delphi suporta praticamente todos os níveis de escopo encontrados no Visual Basic e mais. A tabela abaixo representa os diferentes níveis de escopo em Visual Basic e os correspondentes em Delphi.

<u>Visual Basic</u>	<u>Delphi</u>
<i>Local</i>	<i>Local</i>
<i>Static</i>	<i>typed constant</i>
<i>Module Level</i>	<i>Unit Level</i>
<i>Global</i>	<i>Global</i>
<i>class level</i>	<i>Object Level</i>

Em Visual Basic, se você usa a palavra chave Dim dentro de uma procedure definition, você estará definindo uma variável como local para a procedure. Igualmente se você adicionar uma variável na seção var de uma procedute em Delphi, você estará definindo uma variável local. Não existe equivalente em Delphi para as variáveis tipo static, mas uma variável static é realmente um module level variable que é usado somente numa procedure. É somente um nome de conveniência.

Em Visual Basic, se você usa a palavra chave Dim em uma seção de declaração de um módulo ou form, você estará criando uma module level variable. Da mesma maneira, se você colocar uma variável numa declaração Var de uma seção Implementation de uma unit, você estará definindo uma (i.e. one which can only be seen from within that unit).

Em Visual Basic, se você usar a palavra chave Global na declarations section de um módulo, você estará definindo uma variável global. Em Delphi, você simplesmente coloca a variável na declaração var da seção

interface de uma unit e a variável estará disponível para qualquer unit que estiver com a unit da variável em sua declaração uses. O segmento de código a seguir ajuda a esclarecer:

```
unit MyUnit;  
  
interface  
  
uses  
  
WinProcs, WinTypes; {these are the units used by this unit}  
  
var  
  
globalInt:integer;  
  
globalStr:string;  
  
implementation  
  
var  
  
unitInt:Integer;  
  
unitStr:String;  
  
procedure unitProc;  
  
const  
  
staticInt:Integer = 0;  
  
staticStr:String = '';  
  
var  
  
localInt:Integer;
```

```
localStr:String;
```

```
begin
```

```
{code goes here}
```

```
end;
```

```
end.
```

Este código é uma unit completa com as seções interface e implementation. As variáveis definidas na seção interface são globais. As variáveis definidas na declaração var da unit são unit level e finalmente as definidas na declaração var da seção procedure são locais. Fique atento que variáveis unit-level criadas dentro de um form estão disponíveis através de múltiplas instâncias deste form. Em outras palavras, diferentemente do Visual Basic, estas variáveis existem somente uma por aplicação, então ainda que você pode ter múltiplas instâncias do form, variáveis unit level são compartilháveis. Um meio de contornar isso é definir um conjunto de variáveis como parte de uma nova classe de forms que você criará. Uma típica definição pode parecer com isso:

```
type
```

```
TForm1 = class(TForm)
```

```
Button1: TButton;
```

```
Button2: TButton;
```

```
private
```

```
{ Private declarations }
```

```
public
```

```
{ Public declarations }
```

```
end;
```

Você pode criar uma variável "form level" equivalente às do VB pela adição dela na seção private da definição do form. Então essa mudança parecerá com:

```
type
```

```
TForm1 = class(TForm)
```

```
Button1: TButton;
```

```
Button2: TButton;
```

```
private
```

```
{ Private declarations }
```

```
formInt: Integer;
```

```
formStr: String;
```

```
public
```

```
{ Public declarations }
```

```
end;
```

Se você não criar múltiplas instâncias do seu form em suas aplicações, você nunca saberá a diferença entre variáveis unit level ou form level. Entretanto, uma grande aplicação MDI pode começar como uma aplicação de form simples, então é melhor usar o nível mais restrito de escopo que você puder para que possa flexibilizar no futuro.

## Desvio Conditional

Existem dois tipos de desvio condicional em Delphi como no Visual Basic, `if..then` e `case`. Estas construções são similares nos dois ambientes.

### If...Then...Else

Quase todas linguagens tem alguma forma de declaração `if`. Na sua forma mais simples em Visual Basic:

```
If <condition> Then <action> Else <action>
```

o qual é exatamente como a forma simples em Delphi. A declaração

```
If chkShow.Value = True Then Text1.Visible =True Else Text1.  
Visible = False
```

em Visual Basic, parece com a seguinte em Delphi:

```
If chkShow.State = cbChecked Then Edit1.Visible := True Else  
Edit1.Visible := False;
```

A diferença aparece, quando você precisa de executar múltiplas linhas de código em uma condição. Como mostrado acima, o Object Pascal requer que o bloco de código esteja entre um par de `begin..end`. Então o seguinte código será o código em Delphi para ajustar a propriedade *Visible* de vários controles para *True* numa condição particular:

```
if chkShow.State = cbChecked then
```

```
begin
```

```
Edit1.Visible := True;
```

```
Edit2.Visible := True;
```

```
end;
```

Agora dê uma olhada no exemplo envolvendo um `else`. Você irá observar uma pequena anomalia de sintaxe neste exemplo:

```
if chkShow.State = cbChecked then

begin

Edit1.Visible := True;

Edit2.Visible := True;

end

else

begin

Edit1.Visible := False;

Edit2.Visible := False;

end;
```

Uma exceção na regra de terminar a linha de código com um ponto e vírgula é quando as linhas são imediatamente precedidas por um else. Consequentemente, você observará que a primeira declaração end não tem um ponto e vírgula, porque está imediatamente seguido por um else. Estes pares begin..end não devem ser confundidos com o complemento de um bloco if que é o EndIf. Não existe equivalente para o Endif em Delphi. Você pode, é claro, aninhar a declaração if ao longo de cada bloco de código cercado por begin..end.

### Case

A seleção com a declaração Case em Visual Basic é designada para prover uma estrutura mais elegante do que várias declarações if . A declaração case em Delphi tem uma funcionalidade similar. Veja o seguinte código em Visual Basic:

```
Select Case MyVar
```

```
Case 0
```

```
J = "Hello"
```

```
Case MY_CONST
```

```
j = "Goodbye"
```

```
Flag = False
```

```
Case 2, 3, 5 To 10
```

```
j = "How are you?"
```

```
End Select
```

traduzindo para a linguagem Object Pascal do Delphi:

```
case MyVar of
```

```
0:
```

```
J := 'Hello';
```

```
MY_CONST:
```

```
begin
```

```
j := 'Goodbye';
```

```
Flag := False;
```

```
end;
```

```
2, 3, 5..10:
```

```
j := 'How are you?';
```

Duas coisas devem ficar claras. Primeiro, o bloco de código depois da MY\_CONST requer um par begin..end e segundo, Não existe "End Select" em Delphi. A Estrutura case termina simplesmente com a última linha de código - como o case , outras construções em Object Pascal termina assim, incluindo os loops, discutidos na próxima seção.

Como em Visual Basic, você pode combinar múltiplos cases separando-os com vírgula. Um conjunto é representado por <LowerVal..HigherVal> antes da palavra to. Não existe equivalente para "Is < 500" Em lugar disso, você deve usar um range.

Quando você for usar a declaração case em Delphi, lembre-se que não pode usar strings, somente números para comparação. Se você precisar comparar uma série de strings, você terá de criar um conjunto de declarações if..then. Existe, entretanto, um tipo de dados em Delphi chamado char o qual é um caracter. Por isso ele pode ser representado como um número, isso possibilita o uso da declaração case para comparar valores char. Em outras palavras, olhe o exemplo abaixo:

```
var  
  
C:Char;  
  
begin  
  
Case C of  
  
'A':  
  
DoSomething;  
  
'B', 'D'..'G':  
  
DoSomethingElse;  
  
end;
```

Se você precisa comparar séries de strings, você precisará de fazer um conjunto de declarações if..then em vez de usar o case.

## Loops

Como em qualquer linguagem de programação, existem dois tipos diferentes de loops, determinado e indeterminado. A diferença simplesmente se você sabe o número de vezes que você quer que o loop seja executado antes que continue a execução do código. O loop determinado em Visual Basic é o For..Next, existe uma construção similar em Delphi chamada loop for. Para os loops indeterminados em VB você tem o While..Wend e o Do..Loop. Em Delphi, você tem o while e o repeat..until.

### O loop For

Como no Visual Basic, o loop for permite que você execute um bloco de código um número predeterminado de vezes. A sintaxe é muito similar. Em VB, você deve ter alguma coisa como:

```
For X = 1 To 10
```

```
A = A + X
```

```
Next
```

Enquanto em Delphi o mesmo pode ser representado por:

```
for X := 1 To 10 do
```

```
A := A + X;
```

As duas diferenças são a palavra-chave do no fim da declaração for e não usar o Next o qual define o fim do bloco de código. Em outras palavras se você tem mais de uma operação a executar ficará da seguinte forma:

```
for X := 1 To 10 do
```

```
begin
```

```
A := A + X;
```

```
Caption := IntToStr(X);
```

```
end;
```

onde o bloco begin..end define as linhas do loop.

Em Visual Basic, você pode usar a palavra-chave Step para especificar o incremento pelo qual a variável (i.e. X) deverá ser mudada. Isso é usado para contar mais do que um em um (i.e. Step 5) ou para criar um decremento (i.e. Step -1). Enquanto não existe forma de criar um incremento maior que um em Delphi, é possível criar um contador em decremento com a palavra-chave downto. Observe o exemplo:

```
for X := 10 downto 1 do
```

```
begin
```

```
A := A + X;
```

```
Caption := IntToStr(X);
```

```
end;
```

Onde X inicia em 10 e decresce até 1. Como no Visual Basic, loops em Delphi podem ser aninhados.

### The Do Loop

A funcionalidade da construção Do..Loop em Visual Basic é provida pelo Delphi's através do loop repeat..until. Olhe o exemplo em VB:

```
Do
```

```
K = I Mod J
```

```
I = J
```

```
J = K
```

```
Loop Until J = 0
```

Observe o exemplo em Delphi

```
repeat
```

```
K := I Mod J;
```

```
I := J;
```

```
J := K;
```

```
until J = 0;
```

Neste exemplo, é importante observar que o operador de comparação em Delphi é o sinal =, e não := usado para atribuição. Outro fato que você pode observar é que não foi usado o par begin..end. Isto ocorreu porque o loop repeat..until substituiu as declarações begin e end. Este é o único loop que trabalha desta forma. Todos os outros loops são definidos na primeira linha e usam o par begin..end para múltiplas linhas de código.

### The While Loop

O loop repeat, com o loop Do, faz o teste de condição depois do código ser executado uma vez. Algumas vezes você quer testar a condição antes para que o código não seja executado nenhuma vez. Em Delphi, como no Visual Basic, isso é feito com o loop while. Siga o exemplo em VB:

```
While CanDraw = True
```

```
A = A + 1
```

```
Wend
```

### Agora em Delphi

```
while CanDraw = True do
```

```
A := A + 1;
```

Onde, como no loop for, não existe um terminador para o loop. Entretanto, se você quer executar um bloco de código, use o par begin..end. Olhe o exemplo em VB:

```
While Not Eof(1)
```

```
Line Input #1, Text
```

```
Process Text
```

```
Wend
```

Em Delphi:

```
while not Eof(InFile) do
```

```
begin
```

```
ReadLn (InFile, Text);
```

```
Process (Text);
```

```
end;
```

Como você pode ver, existem estruturas comuns entre as linguagens facilitando a transição entre elas.

## Manipulação de String

O Delphi tem um tipo de variável string, isso trás algumas restrições que você deve saber. Primeiro, o tamanho é limitado em 255 caracteres e não é usado diretamente quando se chama uma função da API requerendo um LPSTR.

Uma String Delphi pode ser usada como se usa em Visual Basic em termos de atribuição (literals cercados por aspas simples), concatenação e comparação. Existem algumas rotinas de manipulação de strings em Delphi que são similares às do VB:

### Visual Basic

*Str*

*Val*

*Len*

### Delphi

*IntStr, Str*

*StrToInt, Val*

*Length*

<i>Instr</i>	<i>Pos</i>
<i>Mid</i>	<i>Copy</i>
<i>Ucase</i>	<i>UpperCase</i>
<i>Lcase</i>	<i>LowerCase</i>
<i>Format</i>	<i>Format</i>

Uma string Delphi pode ser tratada como um array de caracteres tornando certos processos de procura e substituição mais fáceis. Em Delphi use esta sintaxe:

```
OpFlag := TButton(Sender).Caption[1];
```

para determinar o primeiro caracter de uma string numa variável do tipo char. Lembre-se que você não pode usar a declaração case com uma string mas você pode usá-la com um caracter simples. Esta declaração permite que você recupere o primeiro caracter de uam string para usar numa declaração case.

Igualmente, se você precisar de criar um LPSTR para passar à uma função API, você pode construir uma numa string e passar o endereço dela, dessa forma:

```
procedure TForm1.Button1Click(Sender: TObject);  
  
var  
  
S:String;  
  
begin  
  
S := 'Hello World'#0;  
  
MessageBox (0, @S[1], 'Test', 0);  
  
end;
```

O Tipo de dado em Delphi que corresponde ao LPSTR em C é o *PChar* o qual é um ponteiro para um array de caracteres. Existem uma série de

funções desenvolvidas para manipular um *PChar* para atribuição, concatenação e comparação Procure por "String-handling routines (null-terminated)" no Help on line. O código acima implementado, usando *PChars*, ficaria desta forma:

```
procedure TForm1.Button1Click(Sender: TObject);  
  
var  
  
P:PChar;  
  
begin  
  
P := StrAlloc (256);  
  
StrPCopy (P, 'Hello World!');  
  
MessageBox (0, P, 'Test', 0);  
  
StrDispose (P);  
  
end;
```

Observe que não precisa explicitar o uso de um caracter nulo na atribuição, você pode usar *StrAlloc* para criar uma string maior de 255 caracteres e finalmente você não necessita usar o simbolo @ quando passar um *PChar* para uma função API, porque o *PChar* já é definido como um ponteiro.

## Arrays

Arrays são usadas em Delphi da mesma forma que são em Visual Basic. O código VB para criar um array é da seguinte forma:

```
Dim MyArr (10, 1 To 5) As Integer
```

a mesma definição em Delphi deve ser:

```
MyArr: array [0..10, 1..5] of Integer;
```

Em Delphi você deve definir os limites do array.

É possível passar um array para uma procedure que não conhece quantos elementos estão contidos no array de forma muito semelhante como é feito em Visual Basic. Observe o seguinte código em Visual Basic:

```
Dim MyArr (1 to 10) As Integer

Sub Set2Zero (A() As Integer)

Dim i As Integer

For i = LBound (A) to UBound (A)

A(i) = 0

Next

End Sub

Sub Command1_Click ()

Set2Zero MyArr()

End Sub
```

Em Delphi:

```
var

MyArr: array [1..10] of Integer;

procedure Set2Zero (A:array of Integer);

var

i:Integer;
```

```
begin
for i := Low(A) to Hight(A) do
A[i] := 0;
end;

procedure TForm1.Command1Click (Sender: TObject);

begin

Set2Zero (MyArr);

end;
```

Esta sintaxe aumenta a flexibilidade na criação e chamada à procedures genéricas.

## Procedures and functions

Em Delphi, como em Visual Basic, você pode criar procedures antes de realizá-las na forma de event handlers. Você vem usando subrotinas (conhecidas como *procedures* em Delphi) todas as vezes que você executa um duplo click em um objeto para criar um event handler. Uma sintaxe genérica para uma procedure em Delphi é:

```
procedure MyProc (P1:Type; P2:Type);

var {optional}

begin

{code}

end;
```

Bem similar a:

```
Sub MyProc (P1 As Type, P2 As Type)
```

```
Dim... 'optional
```

```
{code}
```

```
End Sub
```

A diferença chave é que os parametros na definição da procedure são separadas com ponto e vírgula em. E ainda, tenha em mente que a definição de variáveis locais requer a o uso da clausula Var.

As funções em Delphi também são similares às funções em VB:

```
Function MyFunc (P1 As Integer, P2 As Long) As Integer
```

```
Dim...
```

```
'Code
```

```
MyFunc = ...
```

```
End Function
```

Isso é equivalente a:

```
function MyFunc (P1:Integer; P2:LongInt):Integer;
```

```
var
```

```
begin
```

```
{Code}
```

```
MyFunc := ...
```

```
end;
```

Uma diferença fundamental é que o Delphi requer a declaração da procedure antes de usá-la, diferente do Visual Basic.

```
procedure OtherProc (P1:Integer);forward;
```

```
procedure MyProc;
```

```
begin
```

```
OtherProc (37);
```

```
end;
```

```
procedure OtherProc (P1:Integer);
```

```
begin
```

```
DoSomethingTo (P1);
```

```
end;
```

Outra diferença importante é que os parametros são passados por valor (default) nas procedure e funções do Delphi, enquanto são passadas por referência em Visual Basic. Delphi dá a você a flexibilidade de escolher o que for melhor para suas necessidades. Para definir um parametro como referência, use a palavra chave var na definição da procedure, como no exemplo abaixo:

```
procedure MyProc(var P1 as Integer);
```

```
begin
```

```
end;
```

Escopo das procedures e funções é similar ao escopo das variáveis. Todas as procedures definidas na seção implementation de uma unit são locais

para aquela unit. Todas que são declaradas na seção interface (todas são implementadas na seção implementation) são disponibilizadas para todas as units inclusas na clausula uses.

O seguinte código é um unit completa chamada VBFUNC que contém uma função QBColor desenvolvida para imitar a funcionalidade de outra função chamada VB.

```
unit VBFUNC;

interface

uses Graphics; {location of TColor def}

function QBColor (n:Integer):TColor; {mentioning here makes
it public}

implementation

function QBColor (n:Integer):TColor;

var

C:TColor;

begin

case n of

0: C := 0;

1: C := 8388608;

2: C := 32768;

3: C := 8421376;

4: C := 128;
```

```
5: C := 8388736;  
6: C := 32896;  
7: C := 12632256;  
8: C := 8421504;  
9: C := 16711680;  
10: C := 65280;  
11: C := 16776960;  
12: C := 255;  
13: C := 16711935;  
14: C := 65535;  
15: C := 16777215;  
  
end;  
  
QBColor := C;  
  
end;  
  
end.
```

## Arrays de Controles

Enquanto o conceito de array de controles não existe em Delphi, sua funcionalidade (e muito mais) estão disponíveis para você. Existem dois motivos para se criar um array de controles em Visual Basic. O primeiro é

quando múltiplos controles compartilham o mesmo código. Isso pode ser realizado pela simples atribuição do mesmo event handler, como o evento `OnClick`, para os múltiplos componentes de um form Delphi. O segundo é quando você quer criar vários componentes "on the fly". Estes dois casos são mais do que possíveis em Delphi.

Você pode, por exemplo, criar um form com três botões. Em Visual Basic, eles todos deveriam ter o mesmo nome `B` e diferentes índices. Poderia então escrever o seguinte código:

```
Sub B_Click (Index As Integer)
```

```
Caption = B(i).Caption
```

```
End Sub
```

Se você quer executar a mesma tarefa em Delphi, você deverá criar três botões e dar a eles nomes como `B_1`, `B_2` e `B_3`. Você deverá criar um event handler para um deles chamado `B` e atribuí-lo aos event handler dos outros dois. O event handler deverá parecer com isso:

```
procedure TForm1.BClick(Sender:TObject);
```

```
begin
```

```
Caption := TButton(Sender).Caption;
```

```
end;
```

Deve ficar claro que o parâmetro "Sender" refere-se ao componente o qual ativou o event handler. Exatamente como você usa o parâmetro `Index` do Visual Basic para diferenciar entre os controles que chamaram o event handler, você usa o *Sender* para determinar a origem do evento em Delphi.

O tipo de dado do *Sender* é simplesmente *TObject* o qual pode ser atribuído a qualquer coisa em Delphi. Entretanto, um button, um text box e um check box podem compartilhar o mesmo event handler em Delphi, algo que não é possível em Visual Basic. O único preço desta flexibilidade é que você deve dizer ao compilador como tratar este *TObject* "lançando-

o" a um objeto mais baixo na hierarquia. A sintaxe deste lançamento é muito parecida com uma função onde você simplesmente usa o nome da classe como o nome da função e o *TObject* com o parâmetro simples. Isso diz ao Delphi para tratar temporariamente o *TObject* como um *TButton*. Você poderia continuar o lançamento desta variável conforme sua necessidade. Se você quiser um código um pouco mais elegante, você pode atribuir a uma variável local o tipo *TButton* desta forma:

```
procedure TForm1.BClick (Sender:TObject);  
  
var  
  
Source:TButton;  
  
begin  
  
Source := TButton(Sender)  
  
Caption := Source.Caption;  
  
end;
```

### Dynamic Control Arrays

Outra razão para usar um array de controles em Visual Basic é para a criação de controles "on the fly" baseado na informação de run-time. Em VB Você deve criar um template e então construir cópias deste template. Você não tem este tipo de limitação em Delphi. Você pode criar qualquer objeto "on the fly" e atribuir event handlers a eles dinamicamente.

No exemplo em VB, ARRAY.FRM no diretório \vb\samples\controls \controls.mak, existe um botão de opção chamado optButton com o index 0 que usa um template na procedure cmdAdd\_Click:

```
Sub cmdAdd_Click ()  
  
If MaxId = 0 Then MaxId = 1  
  
If MaxId > 8 Then Exit Sub
```

```
MaxId = MaxId + 1
```

```
Load OptButton(MaxId)
```

```
OptButton(MaxId).Top = OptButton(MaxId - 1).Top + 400
```

```
OptButton(MaxId).Visible = True ' Display new button.
```

```
OptButton(MaxId).Caption = "Option" & MaxId + 1
```

```
End Sub
```

A rotina em Delphi compatível seria:

```
procedure TForm1.cmdAddClick(Sender: TObject);
```

```
var
```

```
rbOld, rbNew: TRadioButton;
```

```
begin
```

```
if nCtl = 0 then
```

```
nCtl := 1
```

```
else
```

```
Inc(nCtl);
```

```
if nCtl < 16 then begin
```

```
rbOld := TRadioButton(FindComponent ('optButton_' + IntToStr  
(nCtl - 1)));
```

```
rbNew := TRadioButton.Create(Self);
```

```
rbNew.Parent := Self;

rbNew.SetBounds (rbOld.Left, rbOld.Top + rbOld.Height * 2,
rbOld.Width, rbOld.Height);

rbNew.Caption := 'Option' + IntToStr(nCtl);

rbNew.Name := 'optButton_' + IntToStr(nCtl);

rbNew.OnClick := optButton_0Click;

end;

end;
```

Neste código inicialmente se parece com o exemplo em Visual Basic:

1 Configurar o *rbOld* como o último componente criado.

Isto é executado com uma função muito útil chamada `FindComponent` o qual pega o valor de uma string. Tente isso no VB!

1 *rbNew* é criado

2 *rbNew* é colocado no form

3 *rbNew* é movido para um novo lugar

4 *rbNew's* caption é configurado

5 *rbNew's* name é configurado!

6 *rbNew's OnClick* handler é dinamicamente atribuido para *optButton\_0Click* handler!

O resultado é exatamente o mesmo que no código do Visual Basic. O *optButton\_Click* handler em VB parece com:

```
Sub optButton_Click (Index As Integer)
```

```
picDisplay.BackColor = QBColor(Index + 1)
```

```
End Sub
```

Em Delphi:

```
procedure TForm1.optButton_0Click(Sender: TObject);
```

```
var
```

```
i:Integer;
```

```
rb:TRadioButton;
```

```
begin
```

```
rb := TRadioButton(Sender);
```

```
Caption := rb.Name;
```

```
i := StrToInt (Copy (rb.Name, 11, 2));
```

```
shpDisplay.Brush.Color := QBColor (i);
```

```
end;
```

A grande diferença é a ausência do parâmetro *Index*, vocês estão extraíndo o último caracter do caption para pegar o valor do index. Esta é apenas uma técnica para Array de controles. Por exemplo, a propriedade *Tag* em Delphi é um long integer maior que uma string. Então as duas rotinas podem ser vistas como:

```
procedure TForm1.cmdAddClick(Sender: TObject);
```

```
var
```

```
rbOld, rbNew: TRadioButton;

begin

if nCtl = 0 then

nCtl := 1

else

Inc(nCtl);

if nCtl < 16 then begin

rbOld := TRadioButton(FindComponent ('optButton_' + IntToStr
(nCtl -1)));

rbNew := TRadioButton.Create(Self);

rbNew.Parent := Self;

rbNew.SetBounds (rbOld.Left, rbOld.Top + rbOld.Height * 2,
rbOld.Width, rbOld.Height);

rbNew.Caption := 'Option' + IntToStr(nCtl);

rbNew.Name := 'optButton_' + IntToStr(nCtl);

rbNew.Tag := QBColor (nCtl); {this is new}

rbNew.OnClick := optButton_0Click;

end;

end;
```

```
procedure TForm1.optButton_0Click(Sender: TObject);  
  
begin  
  
shpDisplay.Brush.Color := TRadionButton(Sender).Tag;  
  
end;
```

No exemplo acima, o valor atual da cor é armazenado na propriedade Tag, por isso que todas rotinas optButton\_0Click precisam ser configuradas com a mesma cor da propriedade Tag do Sender.

Finalmente, os elementos do array de controle são removidos de maneira similar ao Visual Basic:

```
Sub cmdDelete_Click ()  
  
If MaxId = 1 Then Exit Sub ' Keep first two buttons.  
  
Unload OptButton(MaxId) ' Delete last button.  
  
MaxId = MaxId - 1 ' Decrement button count.  
  
OptButton(0).SetFocus ' Reset button selection.  
  
End Sub
```

Em Delphi:

```
procedure TForm1.cmdDeleteClick(Sender: TObject);  
  
var  
  
rb:TRadioButton;  
  
begin
```

```
if nCtl > 0 then

begin

rb := TRadioButton (FindComponent ('optButton_' + IntToStr
(nCtl)));

rb.Destroy;

Dec(nCtl);

end;

end;
```

Outro exemplo de array de controles pode ser encontrado no exemplo CALC.DPR no diretório [VBSAMPL].

## Object Variables

Object variables é um novo conceito para o Visual Basic, embora se tenha visto como parâmetro em procedures no Delphi e VB. Outra maneira de manipular um conjunto de controles como um array no VB é criando um array de type control e colocá-lo no Form\_Load como:

```
Dim T (1 To 10) As TextBox

Sub Form_Load ()

Dim c As Integer

Dim i As Integer

i = 1

For c = 0 to Form1.Controls.Count -1

If TypeOf Form1.Controls (c) is TextBox then
```

```
Set T(i) = Form1.Controls(c)
```

```
i = i + 1
```

```
end if
```

```
Next
```

```
End Sub
```

**Pode-se, então, "andar" pelo array e configurar as propriedades de todos os controles para um certo valor, exatamente como um array de controle. O mesmo exemplo em Delphi:**

```
var
```

```
i, c:Integer;
```

```
begin
```

```
i := 1;
```

```
for c := 0 to form1.componentcount -1 do
```

```
if Form1.Components[c] Is TEdit then
```

```
begin
```

```
T[i] := TEdit(Form1.Components[c]);
```

```
Inc (i);
```

```
end;
```

```
end;
```

**Object variables são também usado em Visual Basic para criar múltiplas**

instâncias em forms usando o New keyword. Veja o exemplo em VB:

```
Sub cmdNextInstance_Click ()  
  
Dim NextInstance As New frmMain  
  
NextFormNum = NextFormNum + 1  
  
NextInstance.Caption = "Instance # " & NextFormNum  
  
NextInstance.Left = Left + (Width \ 10)  
  
NextInstance.Top = Top + (Height \ 10)  
  
NextInstance.Show  
  
End Sub
```

Funcionalidade similar em Delphi:

```
procedure TfrmMain.cmdNewInstanceClick(Sender: TObject);  
  
var  
  
F:TfrmMain;  
  
begin  
  
F := TfrmMain.Create(Application);  
  
Inc(NextFormNum);  
  
F.Caption := 'Instance #' + IntToStr(NextFormNum);  
  
F.Left := Left + (Width div 10);  
  
F.Top := Top + (Height div 10);
```

```
F.Visible := True;
```

```
end;
```

Com excessão do método .Create, este código é praticamente identico ao código usado em Visual Basic.



*Delphi, Delphi 2.0, Borland International, and the Delphi logos are either trademarks or registered trademarks of Borland International.*

*All other trademarks are the sole property of their respective owners.*