The Unofficial Newsletter of Delphi Users - Issue #17 - October 1996

You know, every day I get mail from readers of UNDU, and some of the best ideas I get from these emails. Sometimes I think I have a mental block in certain areas, but things I see every day sometimes don't really register in my brain. That's why one readers comment of "Hey… why don't you put together a search tool on your web site for finding specific material in previous issues?"   Hmmm… why not indeed! I fiddle with search engines every day on the Internet and I guess it never really registered that I could do it with the newsletter. (Duhhh!!)

Anyway, give me a little time and you will soon see a search feature on the UNDU web home (which, by the way is: http://www.informant.com/undu).

On an additional thought, I have a number of good tips and articles that I had indicated in emails would appear in this issue. Not wanting to let the issue get too big, I decided to hold them until next issue. If you have submitted something, please be patient, you will probably see it soon!

- Robert

Does Windows 95 give you a Square Deal?

The Great StringList

Manipulating Regions with Delphi

Tips & Tricks

UNDU Prizes!

UNDU Subscriber List

Index of Past Issues

Where To Find UNDU

Index of Past Issues

Below is a complete index of all principle articles in past issues of the Unofficial Newsletter of Delphi Users. Provided that you have the prior issues in the same directory as this issue, you can click on any of these hotspots to go directly to that article. To return to the index, you can click on the **Back** button, or you can use the **History** list. Once you jump to one of these issues, you can navigate through the issue as you would normally, but you will need to go to the **History** list to get back to this index. There will be an updated index included in all future issues of UNDU.

Issue #1 - March 15, 1995

- What You Can Do
- Component Design
- Currency Edit Component
- Sample Application
- The Bug Hunter Report
- About The Editor
- SpeedBar And The ComponentPalette
- Resource Name Case Sensitivity
- Lockups While Linking
- Saving Files In The Image Editor
- File Peek Application

Issue #2 - April 1, 1995

- Books On The Way
- Making A Splash Screen
- Linking Lockup Revisited
- Problem With The CurrEdit Component
- Return Value of the ExtractFileExt Function
- When Things Go Wrong
- Zoom Panel Component

Issue #3 - May 1, 1995

- Articles
- Books
- Connecting To Microsoft Access
- Cooking Up Components
- Copying Records in a Table
- CurrEdit Modifications by Bob Osborn
- CurrEdit Modifications by Massimo Ottavini
- CurrEdit Modifications by Thorsten Suhr
- Creating A Floating Palette
- What's Hidden In Delphi's About Box?
- Modifications To CurrEdit

Periodicals
Progress Bar Bug
Publications Available
Real Type Property Bug
TIni File Example
Tips & Tricks
Unit Ordering Bug
When Things Go Wrong

Issue #4 - May 24, 1995

Cooking Up Components
Food For Thought - Custom Cursors
Why Are Delphi EXE's So Big?
Passing An Event
Publications Available
Running From A CD
Starting Off Minimized
StatusBar Component
TDBGrid Bug
Tips & Tricks
When Things Go Wrong

Issue #5 - June 26, 1995

Connecting To A Database
Cooking Up Components
DateEdit Component
Delphi Power Toolkit
Faster String Loading
Font Viewer
Image Editor Bugs
Internet Addresses
Loading A Bitmap
Object Alignment Bug
Second Helping - Custom Cursors
StrToTime Function Bug
The Aquarium
Tips & Tricks
What's New
When Things Go Wrong

Issue #6 - July 25, 1995

A Call For Standards
Borland Visual Solutions Pack - Review
Changing a Minimized Applications Title
Component Create - Review
Counting Components On A Form
Cooking Up Components
Debug Box Component
Dynamic Connections To A DLL
Finding A Component By Name
Something Completely Unrelated - TVHost
Status Bar Component

The Loaded Method
Tips & Tricks
What's In Print

Issue #7 - August 31, 1995

ChartFX Article
Component Cookbook
Compression Shareware Component
Corrected DebugBox Source
Crystal Reports - Review
DBase On The Fly
Debug Box Article
Faster String Loading
Formula One - Review
Gupta SQL Windows
Header Converter
Light Lib Press Release
Limiting Form Size
OLE Amigos!
Product Announcements
Product Reviews
Sending Messages
Study Group Schedule
The Beginners Corner
Tips & Tricks
Wallpaper
What's In Print

Issue #8 - October 10, 1995

Annotating A Help System
Core Concepts In Delphi
Creating DLL's
Delphi Articles Recently Printed
Delphi Informant Special Offers
Delphi World Tour
Getting A List Of All Running Programs
How To Use Code Examples
Keyboard Macros in the IDE
The Beginners Corner
Tips & Tricks
Using Delphi To Perform QuickSorts

Issue #9 - November 9, 1995

Using Integer Fields to Store Multiple Data Elements in Tables
Core Concepts In Delphi
Delphi Internet Sites
Book Review - Developing Windows Apps Using Delphi
Object Constructors
QSort Component
The Component Cookbook
TSlideBar Component
TCurrEdit Component

The Delphi Magazine
Tips & Tricks
Using Sample Applications

Issue #10 - December 12, 1995

A Directory Stack Component
A Little Help With PChars
An Extended FileListBox Component
Application Size & Icon Tip
DBImage Discussion
Drag & Drop from File Manager
Modifying the Resource Gauge in TStatusBar
Playing Wave Files from a Resource
Review of Orpheus and ASync Professional
The Component Cookbook
Tips & Tricks
UNDU Readers Choice Awards
Using Integer Fields to Store Multiple Data Elements in Tables

Issue #11 - January 18th, 1996

Core Concepts With Delphi - Part I
Core Concepts With Delphi - Part II
Dynamic Delegation
Data-Aware DateEdit Component
ExtFileListBox Component
DBExtender Product Announcement
Dynamic Form Creation
Finding Run-Time Errors
Selecting Objects in the Delphi IDE
The Beginners Corner
The Delphi Magazine
Top Ten Tips For Delphi
The Component Cookbook
Tips & Tricks
The UNDU Awards

Issue #12 - Feburary 23rd, 1996

The Beginners Corner
Delphi Projects
Marketing Your Components
An LED Component
A 3D Progress Bar
Common Strings Functions
Checking if your application is running already
AutoRepeat for SpeedButtons
Form and Component Creation Tip
Detecting a CD-ROM Drive
Drawing Metafiles in Delphi
Shazam Review
Product Announcement - Dr. Bob's Delphi Experts
Book Review - Instant Delphi Programming
Tips & Tricks

The Component Cookbook

Issue #13 - May 1st, 1996

Core Concepts - Sorting
Delphi Information Connection
Creating Resource-Only DLL's
Quick Reports
TIFIMG Product Announcement

Issue #14 - June 1st, 1996

A 3-D Component
An Animation Component
A Bug In TGauge
The Component Cookbook
A Look At Cross Tabs
New Book - Delphi In Depth
New Book - The Revolutionary Guide to Delphi 2
Making the Enter Key Work Like the Tab Key
Jumping Straight to Form Level
Making Menu Items Work Like Radio Buttons
Modifying The System Menu
Products & Reviews
The Beginners Corner
The UNDU Awards
Tips & Tricks

Issue #15 - August 1st, 1996

UNDU - A Work In Progress…
UNDU Prizes!
The UNDU Subscriber List
Core Concepts With Delphi - Parameter Passing
Delphi Programmers Book Shelf
Component Cookbook
Tips & Tricks
How to 'Catch'Keys
Working with String Grids
Coloring Columns in a Grid
Solving a DLL problem
Reducing Memory Requirements
Creating an AutoDialer component

Issue #16 - September 1st, 1996

Menu Buttons
Core Concepts With Delphi - Enumerated Types
Extending The INI Component
Limiting Multiple Instances Of a Program in Delphi 2.0
How to Draw a Rubber-Banding Line
Marching Ants!
How to Restrict the Mouse Cursor
How to make a Color ComboBox
A Better Way to Create Menu Items
Splash Screen

Splash Screen with a Time Delay

Return to Front Page

Where To Find UNDU

When each issue of UNDU is complete, I put them in the following locations:

1. UNDUs official web site at `http://www.informant.com`. This site houses all the issues in both HTML and Windows HLP format.

2. *Borlands* Delphi forum on CompuServe (**GO DELPHI**) in the "Delphi IDE" file section. This forum will only hold the issues in Windows HLP format.
3. *Informant Communications* forum also on CompuServe (**GO ICGFORUM**) in the "Delphi 3rd Party" file section. Again, this forum will only hold issues in the Windows HLP format.

Tips & Tricks

This month, we have some more wonderful tips and techniques you can use in your Delphi programs. If you have a particular trick you like to use, please send it in, no matter how trivial you think it may be. Sometimes, the best tricks are the ones that are 1 or 2 lines of code!

When Delphi's smart-linker doesn't seem so smart
Cut, Copy, & Paste
A Quick Way of Setting the Tab Order
Background Bitmaps on Forms
Non-Rectangular Windows

Return to Front Page

UNDU Prizes!

Each month, I will be giving away 1 or 2 Delphi related products to a randomly chosen contributor to that issue of UNDU.

The randomly chosen winner this month is **Frank Krueger**. Frank's prize is a copy of **Delphi-In-Depth**, a new book from Osborne/McGraw Hill covering advanced Delphi 2.0 techniques.

Thanks to all the contributors for making UNDU a success. Keep those articles & tips coming!

Return to Front Page

UNDU Subscriber List

The subscriber list is a method by which I can notify the readers when a new issue is out. I will maintain a list of readers email addresses and when a new issue is released, I will fire off a batch mailing to notify everyone that it is available.

This is what you need to do to get on the subscriber list… Simply send me an email to my CompuServe address (**76416,1373**) and put the words **SUBSCRIBE UNDU** anywhere in the subject line or in the main body of the message. If you no longer wish to be notified of future issues (i.e. you are on the list and want off…) just send an email with the words **UNSUBSCRIBE UNDU**. If you are sending mail from the Internet, the address is **76416.1373@compuserve.com**

Thats all there is to it!

Return to Front Page

Background Bitmaps on Forms

by Robert Vivrette - CIS: 76416,1373

I am sure many of you have had an opportunity to see some of the visual improvements that are developing in the latest version of Microsoft's Internet Explorer. One of the more subtle effects that they do is actually quite easy to accomplish in Delphi. Look at the portion I have clipped out below:

Do you see those light gray lines running through the background? This is accomplished simply by painting a bitmap on the surface of the form before any of the controls are drawn. In Delphi, you might be inclined to throw a TImage on your form, set it to AlignClient and load a bitmap into it. While this does work in most instances, there are some problems that crop up. There is a better way!
The first step is to find a bitmap that you like and convert it into a resource file that can get linked in with your Delphi application. Simply use any text editor (or the Delphi Editor) to create a text file with a single line of text:

```
MYBITMAP BITMAP ROUGH.BMP
```

When you name the file, give it an RC extension. Let's say we name it **BMPRES.RC**. The first part of this definition is the name that will identify the resource in the program, next comes the type of resource, and then the name of the bitmap file that will be used. Now simply compile it using the Delphi resource compiler. With Delphi 1.0 the command is:

```
C:\DELPHI\BIN\BRCC BMPRESRC.RC
```

and in Delphi 2.0, the command is:

```
C:\BORLAND\DELPHI 2.0\BIN\BRCC32 BMPRESRC.RC
```

If you installed Delphi in a different place, you would substitute the appropriate path to the resource compiler.

After doing this, the resource compiler will have created a BMPRESRC.RES file. Now, in the unit for the appropriate form, you simply do the following: (Note, the entire source for this example is referenced at the bottom of this article).

1. Create a TBitmap variable in the private section of the form.

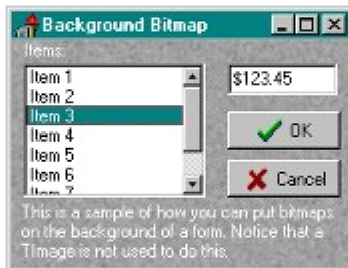
2. Include {$R BACKBMP.RES} within the body of the unit. This will tell the Delphi compiler to link your bitmap resource in with your application.
3. Add the following events for the form:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  BackgroundBitmap := TBitmap.Create;
  BackgroundBitmap.LoadFromResourceName(hInstance,'MYBITMAP');
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  BackgroundBitmap.Free;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Draw(0,0,BackgroundBitmap);
```

```
end;
```

In the LoadFromResourceName method, make sure you use the same resource name that you defined in the RC file earlier.

Now just design your form normally. When the application is compiled and executed, you will see something like this:

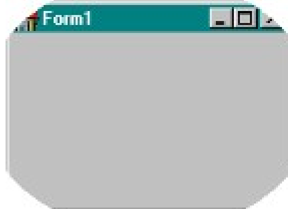


Make sure you don't use a bitmap that is overly complex. This one is borderline because the text is just getting a little difficult to read. The graphic used in the Microsoft Internet Explorer is attractive because it is subtle. In fact, you may not want to paint the entire form with the bitmap, but only a portion of it to provide an accent.

Source for Background Bitmap

Return to Tips & Tricks

Return to Front Page



Non-Rectangular Windows

by Robert Vivrette

I don't know if anyone will find this interesting, but I thought I would include it and see if there is someone out there who can find a neat use for it.

Something new for Windows 95 & NT is the procedure SetWindowRgn. This procedure allows you to define a region (ellipse, square, polygon, whatever) and then attach that region to a window. In the example above, the code generates an elliptical region and then uses the SetWindowRgn procedure to connect it to the main form. The result here is that you get a form with its corners clipped off.

Granted, it is not an amazingly useful capability, but I think there could be someone out there who might do something with this. Microsoft did create a small demo for this capability where they used the Clock program in Windows and made it appear on the screen perfectly round with no window banner or anything. The effect was quite cool as you could drag the clock around the screen and it would behave like any other form. I did think that someone could make a form that has a "torn" edge across the bottom. Might be attractive in some specialized situations.

Anyway, here is the code that achieves this effect. The Win32API help file discusses this procedure in more detail.

Source for Non-Rectangular Windows

Return to Tips & Tricks

Return to Front Page

Source For Background Bitmap

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Edit1: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    BackgroundBitmap : TBitmap;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

{$R BACKBMP.RES}

procedure TForm1.FormCreate(Sender: TObject);
begin
  BackgroundBitmap := TBitmap.Create;
  BackgroundBitmap.LoadFromResourceName(hInstance,'MYBITMAP');
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  BackgroundBitmap.Free;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Draw(0,0,BackgroundBitmap);
end;

end.
```

Return to Background Bitmaps
Return to Tips & Tricks
Return to Front Page

Source for Non-Rectangular Windows

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
    procedure FormClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    IsRound : Boolean;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormClick(Sender: TObject);
var
  R : HRgn;
begin
  if IsRound then
    begin
      SetWindowRgn(Handle,0,True);
      DeleteObject(R);
      IsRound := False;
    end
  else
    begin
      R := CreateEllipticRgn(-15,-15,Width+15,Height+15);
      SetWindowRgn(Handle,R,True);
      IsRound := True;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  IsRound := False;
end;

end.
```

Return to Non-Rectangular Windows
Return to Tips & Tricks
Return to Front Page

The Great TStringList

by Frank A. Krueger

For a long time I puzzled over the notion of building a dynamic hierarchy of user-defined objects. For some time I toyed with TLists and a bucket full of pointers. This turned out to be a complete mess. At this point I was introduced to what I now believe to be one of the most powerful aspects of Delphi; the TStringList. The TStringList has one method which makes it so great: AddObject. AddObject will accept any type of object, even user-defined types which makes it perfect for creating a hierarchy. To demonstrate the usefulness of a TStringList, we'll write the framework for a short program.

The program we are about to write will seem very simplistic and straight-forward from the user's point-of-view, but to the programmer, the program will function like a highly organized efficient dream. This program will be a very simple inventory utility which will help keep track of your furniture and pets (Really? Furniture **and** pets??), so the next time you forget your cat's name, you won't have to worry! Anyway, the first thing we must do is make up some objects to use:

```
type
  TPet = class(TObject)
    Name: string;
    Kind: string;
    Age: integer;
  end;
  TFurniture = class(TObject)
    Kind: string;
    Brand: string;
    Age: integer;
  end;
```

This is good. Now what we need is a container for these objects. Keep in mind that we want this to be a hierarchy, so we will have to make another object class. Since this article is entitled "The Great TStringList," we'll use the TStringList as the "headings." So let's define one:

```
type
  Tinventory = class(TObject)
    Pets: TStringList;
    Furniture: TStringList;
    constructor Create;
    procedure Free;
  end;
```

Now we also need a variable for this object:

```
var
  Inventory: TInventory;
```

Now let's create some functions which will allow us to use the **Inventory** variable. Because we are using user-defined objects with TStringLists, we will have to modify the constructor, and free procedures:

```
constructor TInventory.Create;
begin
```

```
  inherited Create;
  Pets := TStringList.Create;
  Furniture := TStringList.Create;
end;

procedure TInventory.Free;
begin
  Pets.Free;
  Furniture.Free;
  inherited Free;
end;
```

This too is good. Our next procedure will be used to add a pet/pet data to our hierarchy, but first a reminder. Before using any of the following functions/procedures, you MUST first create the Inventory variable (`Inventory := TInventory.Create;`). Anyway:

```
procedure AddPet(Name,Kind: string; Age: integer);
var
  P: TPet;
begin
  P := TPet.Create;
  P.Name := Name;
  P.Kind := Kind;
  P.Age := Age;
  Inventory.Pets.AddObject('P', P);
end;
```

All this procedure does is create an instance of TPet and adds that to the Inventory.Pets "array." This creates the wonderful hierarchy I was searching for. Under the heading "Inventory" we have the sub-headings "Pets," and soon to be "Furniture." Now look at the AddObject method. Notice the use of the string 'P'. AddObject requires a string argument to accompany the object, so I use a one character identifying code (P for Pets). Now our furniture adding procedure is quite similar:

```
procedure AddFurn(Kind,Brand: string; Age: integer);
var
  F: TFurniture;
begin
  F := TFurniture.Create;
  F.Kind := Kind;
  F.Brand := Brand;
  F.Age := Age;
  Inventory.Furniture.AddObject('F', F);
end;
```

All is well. We now have a nicely organized hierarchy of object! All that's left to learn is how to use them. Fortunately, this is quite simple. All you have to do is access the Objects property of your desired sub-heading. For instance:

```
Edit1.Text := TPet(Inventory.Pets.Objects[Inventory.Pets.Count-1]).Name;
```

Amazing isn't it? Much more can be achieved using the TStringList, but you will have to work on all that fun stuff without me. Well, *à plus tard,* and don't forget to free your Inventory!

Editors Note: *Another structure you may wish to use for hierarchies like this is to use the* ***TList*** *object instead of* ***TStringList****. TList also stores objects, but does not expect each object to be paired with a string value. If you don't need to associate a string with each item in the list, perhaps a TList might be a better choice. In an inventory program such as this, you might indeed wish to use the StringList and use the string associated with each item in the list to store the objects name for example. Then the associated object could hold additional information related to the item, such as size, color, brand, age, etc.*

Return to Front Page

Cut, Copy, & Paste

by Brad Evans - Eevans1@cc.curtin.edu.au

Cut, copy and paste seem to be the easiest menu items to implement. You just create the menu items and attach some code to copy the selected text to the clipboard and it is done. No worries! That is until you actually try and do it.

The Hard Way

The hard way to implement cut, copy and paste is using ActiveControl and typecasting the active control to call it's CopyToClipboard method. This wasn't too bad except the application I was developing was a MDI based and each child window has about 10 database fields. This looked so daunting for me so there had to be an easier way.

VCL To The Rescue

I checked the demo applications, online help, manuals, the Web all to no avail. So if in doubt read the VCL source to see how it is done. The VCL and Win32 help file together enabled me to find the perfect solution to my problem, Messages. If I send a message to my application I don't have to do typecasting, try…except blocks, error checking, etc.. I can just send and forget. The messages of importance are; WM_CUT WM_COPY, , and WM_PASTE.   Pretty obvious once you know about them.

So if you add the following to your menu items:

```
{For MDI based applications chnage the active control to
ActiveMDIChild.ActiveControl.Handle }

procedure TfrmMain.mniCopyClick(Sender: TObject);
begin
  SendMessage(ActiveControl.Handle, WM_COPY, 0, 0);
end;

procedure TfrmMain.mniPasteClick(Sender: TObject);
begin
  SendMessage(ActiveControl.Handle, WM_Paste, 0, 0);
end;

procedure TfrmMain.mniCutClick(Sender: TObject);
begin
  SendMessage(ActiveControl.Handle, WM_Cut, 0, 0);
end;
```

You can forget about cut, copy & paste forever. It will just work. You don't have to worry if the current control does not support copy & paste it will just ignore the error and continue. All in one line of code (well three but who's counting). I knew Delphi could do it… I just didn't know it was that easy.

About the Author

Brad Evans is a Programmer for Realtime Computing in Perth Western Australia.   Brad develops leasing applications full time using Borland Delphi 2.0. You can reach Brad through e-mail at Eevans1@cc.curtin.edu.au.

Return to Tips & Tricks

Return to Front Page

Does Windows 95 give you a Square Deal?

By Grahame Marsh - grahame.s.marsh@corp.courtaulds.co.uk

Recently I was developing an application which manipulated digital raster maps of the UK. I had bought these maps as 4000x4000 pixel, 256 color windows bitmaps, so you can see these bitmaps are large, occupying 16,001,078 bytes of disc storage each! They are very detailed, each map covering an area 20km square at a scale of 1:50,000.

But I digress… One of the functions I incorporated into the application was to view this map as a thumbnail using a TImage occupying the whole of a form (Align property set to alClient) with the Stretch property set to true. This worked fine, except the user could resize the screen to a non-square shape, seriously distorting the map. I therefore set out to force the image to be square. Two window message handlers later on the form and I had my solution:

The first window message is a new one incorporated into Windows 95 - this is WM_SIZING. It is sent repeatedly *while* a window is being dragged. The Delphi **Messages** unit does not contain a message type, so I wrote my own in the same style:

```
type
  TWMSizing= packed record
    Msg    : cardinal;
    Edge   : integer;
    Rect   : PRect;
    Result : longbool
  end;
```

The edge parameter (wParam) can take one of eight values depending on the edge or corner being dragged. The windows unit contains these values:

```
WMSZ_LEFT
WMSZ_RIGHT
WMSZ_TOP
WMSZ_TOPLEFT
WMSZ_TOPRIGHT
WMSZ_BOTTOM
WMSZ_BOTTOMLEFT
WMSZ_BOTTOMRIGHT
```

The Rect parameter (lParam) is a pointer to a TRect with the screen coordinates of the drag rectangle. The drag rectangle can be changed too by changing the values in this structure. It now becomes straightforward to write the code: if the user drags, say, the left edge, then the application must move the bottom edge to maintain height equal to width and so on:

```
procedure TSquareForm.WMSizing(var Msg: TWMSizing);
begin
  with Msg, Rect^ do
  begin
    case Edge of
      WMSZ_BottomRight,
      WMSZ_Bottom : Right := Left + (Bottom - Top);

      WMSZ_BottomLeft,
```

```
      WMSZ_Right  : Bottom := Top + (Right - Left);

      WMSZ_TopRight,
      WMSZ_Left   : Top := Bottom - (Right - Left);

      WMSZ_TopLeft,
      WMSZ_Top    : Left := Right - (Bottom - Top)
    end;
    Result := true;
  end
end;
```

But this does not take into account the case when the user hits the maximize button. The WMSizing message is not sent. In this case we must use the **WMGetMaxMinInfo** message which is sent when the size or position is about to change. So we only need force the width to be equal to the height (or the other way if height has more pixels than width) to force the effect wanted:

```
procedure TSquareForm.WMGetMinMaxInfo(var Msg: TWMGetMinMaxinfo);
begin
  with Msg.MinMaxInfo^.ptMaxSize do
    if X > Y then
      X := Y
    else
      Y := X
end;
```

The square-only effect can be achieved other ways, for instance, you can use the OnResize event to modify a form to square after it has been resized by the user to a non-square shape.   But in this case the user drags the form to a shape, when the mouse is released the form snaps back to another shape, and possibly an unwanted size. This behavior is decidedly non-intuitive.

The application was written for Windows 95 only and I have not investigated how to get exactly the same behavior under Windows 3.1. Obviously, the absence of the WMSizing message makes this more difficult.

I suppose the next step will be to encapsulate the behavior in a component, but I have a help file to write next and I must not get side-tracked!

Return to Front Page

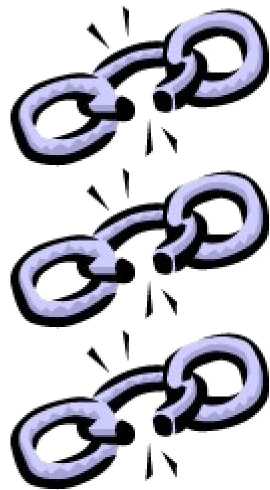
Source for Square Deal

```
--------- MYFORM.PAS complete unit ---------

unit MyForm;

interface

uses
  Windows, Messages, Classes, Forms, ExtCtrls, Controls;

type
  TWMSizing= packed record
    Msg    : cardinal;
    Edge   : integer;
    Rect   : PRect;
    Result : longbool
  end;

type
  TSquareForm = class(TForm)
    Image : TImage;
  private
    procedure WMSizing (var Msg : TWMSizing); message WM_Sizing;
    procedure WMGetMinMaxInfo (var Msg : TWMGetMinMaxInfo); message
WM_GetMinMaxInfo;
  public
  end;

var
  Square: TSquareForm;

implementation

{$R *.DFM}

procedure TSquareForm.WMSizing (var Msg : TWMSizing);
begin
  with Msg, Rect^ do
  begin
    case Edge of
      WMSZ_BottomRight,
      WMSZ_Bottom : Right := Left + (Bottom - Top);

      WMSZ_BottomLeft,
      WMSZ_Right  : Bottom := Top + (Right - Left);

      WMSZ_TopRight,
      WMSZ_Left   : Top := Bottom - (Right - Left);

      WMSZ_TopLeft,
      WMSZ_Top    : Left := Right - (Bottom - Top)
    end;

    Result := true
  end
end;

procedure TSquareForm.WMGetMinMaxInfo (var Msg : TWMGetMinMaxinfo);
begin
  with Msg.MinMaxInfo^.ptMaxSize do
    if X > Y then
```

```
      X := Y
    else
      Y := X
end;

end.

------ MYFORM.DFM  as text ------

object SquareForm: TSquareForm
  Left = 375
  Top = 237
  Width = 400
  Height = 400
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  PixelsPerInch = 96
  TextHeight = 13
  object Image: TImage
    Left = 0
    Top = 0
    Width = 392
    Height = 373
    Align = alClient
    Stretch = True
  end
end
```

Return to Square Deal

Return to Front Page



Smart-Linking Issues

by Frank M. DeBlanc - CIS: 70713,640

I always thought that when a unit is not used in the program but is in the **USES** clause, then that unit will **not** be used. This is because Delphi's smart-linker only links in units that are needed by the program. Trying to find a bug in one of my programs I found something interesting. If you have the unit **ToCtrl** (Delphi 1.0) in your uses clause but do not use it, Delphi will not smart-link it out but think that you are using it.

I experimented with two projects. The only difference between the two projects is that one had the **ToCtrl** unit in the uses clause and the other project did not have the **ToCtrl** unit in its uses clause.

I compiled both projects and it was noted that when used on another computer the program without the **ToCtrl** in the uses clause ran fine. The program with the **ToCtrl** in the uses clause did not run but gave the error message: *"Could not fine BOLE16D.DLL"*. Even though I did not reference the unit in my program Delphi did not smart-link it out.

Editors Comments: *This is one I have wanted to point out in the past, but it always slipped my mind. The reason this occurs is because some units have initialization code that gets executed anytime it is compiled in with the program. In such cases, it doesn't matter that you are not using specific objects or routines in the unit. Rather, the compiler sees that there is code in the* ***Initialization*** *section (or the* ***Begin****…****End*** *block at the end of a unit) and links in everything necessary for this code.*

This can be quite a "gotcha!" in some cases. Say for example you may be developing a new component, and you decide to make a version of it that is data-aware. You think that since they are essentially the same component, they should be in the same unit. Don't do it! Since you have references to some of the database units (DB.PAS, DBCTRLS.PAS, etc), the compiler will link in code to initialize the BDE. As a result, the BDE will be required for you to use even the non-data-aware version of your component because the database-related units have initialization code that fires up the Borland Database Engine.

Return to Tips & Tricks

Return to Front Page

A Quick Way of Setting the Tab Order

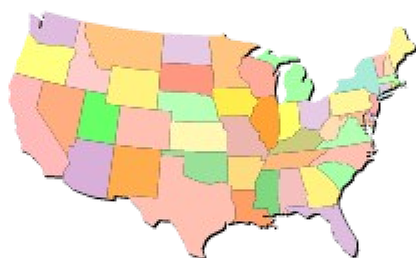
by Robert Vivrette - CIS: 76416,1373

Sometimes we don't think about it too much. Or we forget about it altogether! The tab order of our controls on a form is generally overlooked until someone tries to use the application without the mouse. Then watch the fur fly!!!

Each control on a form that is capable of receiving the input focus has a TabOrder property. When the form first gains control, the component with the lowest numbered TabOrder will initially have focus. Then, when the tab key is pressed, focus shifts to the control with the next highest number and so on.

Most developers will use the Tab Order menu option on the right-click menu in the Delphi IDE. With this, Delphi allows you to shuffle around the component names to indicate their tab order.
Sometimes however, doing it this way can be a bit complex, particularly if you have many controls on the form, and the list is difficult to move through. *Here is an alternate way I like to use at times:*
What I do is look at the form and figure out which component I want to be **LAST** in the tab order. I select it and then key in a **zero** ("0") in its TabOrder. Then I go to the control I want to be second from the last and also give it a zero for its TabOrder. I continue doing this until I get to the component I want to be first in the tab order and I also give it a zero for its TabOrder. The Delphi IDE renumbers all the TabOrder values every time one is changed. Because I keep typing in a zero value for the TabOrder, it keeps pushing down all the components that I had keyed in prior. At the end, the list is sorted exactly the way I want!
Return to Tips & Tricks

Return to Front Page



Manipulating Regions with Delphi

by Don Monroe

This article deals with a set of API calls that are usually encapsulated in Controls/Components to create HotSpots on graphics images. Unfortunately, these Controls/Components usually need an image predefined to setup the HotSpots. The API calls themselves are easily found in the Windows API help by searching for the word 'REGION'. The search results in a treasure trove of functionality that was until now hidden.

While I was writing a random map based game in Delphi, I ran into a wall. How could I determine which continent the user was currently working with, in which province, district, region, city, town, village, and so on?

I proceeded to run the CIS and AOL gauntlet looking in vain for help with my problem. I looked in the Help file for anything detailing HotSpots and found nothing. But when I searched for *Region* I knew I had found the answer.

The first thing you have to do is declare a global variable of type HRgn. Here I call it **MyRegion**.

```
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs;

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

var
  Form1: TForm1;
  MyRegion: HRgn;

implementation

{$R *.DFM}
```

Now, in the FormCreate event we will create a region. See below:

```
procedure TForm1.FormCreate(Sender: TObject);
Var
  Points : Array[0..4] of TPoint; {Needed for the API Call}
begin
  Points[0].X := 10; Points[0].Y := 10;
```

```
  Points[1].X := 50; Points[1].Y := 10;
  Points[2].X := 70; Points[2].Y := 100;
  Points[3].X := 60; Points[3].Y := 150;
  Points[4].X := 90; Points[4].Y := 110;
  MyRegion := CreatePolygonRgn(Points,5,Winding);
end;
```

The region can be created using different shapes. There is usually just a little in the way of parameter changes to consider. Listed below are the different Region types.

CreateEllipticRgn *CreateEllipticRgnIndirect*
CreatePolygonRgn *CreatePolyPolygonRgn*
CreateRectRgn *CreateRectRgnIndirect*
CreateRoundRectRgn

Now that we have created the region, for our example, we need to display it. This step may not be needed for your application. It is only shown for testing purposes. In the FormPaint event add the following:

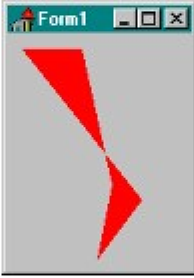
```
procedure TForm1.FormPaint(Sender: TObject);
Var
  B : Boolean;
begin
  Canvas.Brush.Color := clRed;
  B := PaintRgn(Canvas.Handle,Region);
end;
```

The PaintRgn function fills the specified Region with the current Brush.Color. Now if you run the example you should see your region. Right now it doesn't do anything except show the region. But we will fix that next.

The heart of creating a region on any surface is to find out where the user is… i.e. inside or outside a given region. Hence, the purpose of the PtInRegion Function. By passing this function the Region to check, and the X,Y coordinates of the Mouse, you can find out where the user is clicking. In the FormMouseUp event (or FormMouseDown if you prefer) add the following code:

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  If PtInRegion(Region,X,Y) then InvertRgn(Canvas.Handle,Region);
end;
```

This **IF** statement calls another function called InvertRgn. This function inverts the color of the region so in this case when the user clicks inside the region, the color of the region changes.



There is only one last thing to discuss and that is removing the region from memory when we are through with it. The DeleteObject function, while not listed with the Region Functions, handles the job for us. In the FormDestroy Event Place the following:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  DeleteObject(Region);
end;
```

When the form is destroyed the Region is removed from memory. This step is VERY important. If you fail to release the Regions then the memory used be them will be lost until Windows is rebooted. Make sure you delete the regions before when your done. Better safe than sorry.

Return to Front Page