The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

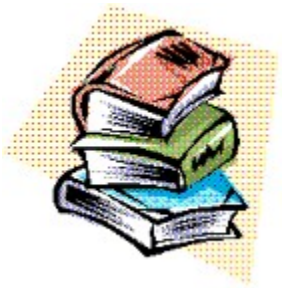
I don't know if it is my imagination, but these issues are getting bigger each month! I must say I am getting very good support for UNDU from many of the readers and it does make my work putting it together much easier. Again, my thanks for everyone's great contributions. This issue is big because there was so much good stuff to include!

Our randomly selected winner of the UNDU prize this month is **Paul Harding** for his article on Multi-Color Text in String Grids, and his prize is a copy of **Kick-Ass Delphi** by the Coriolis Group. In addition, OOPSoft, Inc has offered a special set of prizes. For this issue and next issue, they will be giving out 5 copies of their ObjectExpress package (reviewed is issue #18) to randomly chosen contributors. The five winners this month are Jim Clokey, Eric Fortier, Philip Hibbs, Gene Fowler, and Magnus Baeck. Remember, next month there will be 5 more copies give out in addition to the regular UNDU prize! We will also be looking at OOPSoft's new SQLExpress package next issue.

Also, for those of you who are interested, there is now a 16-bit version of the IniOut Property Manager mentioned in issue #18. If you are viewing this issue as a web page, you can obtain the 16-bit and 32-bit shareware versions by clicking here.

The next issue will be only 1 month away (rather than 2 for this issue). Look forward to seeing reviews on Danny Thorpe's excellent **Delphi Component Design** book, Micro-Edge's **Visual SlickEdit for Delphi**, and the **AddDict** spell checker/thesaurus for Delphi, along with tons of great tips, techniques, and components!

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997



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 #4 - May 24, 1995

Issue #5 - June 26, 1995

Issue #6 - July 25, 1995

Issue #7 - August 31, 1995

Issue #8 - October 10, 1995

Issue #9 - November 9, 1995

Issue #10 - December 12, 1995

Issue #11 - January 18th, 1996

Issue #12 - Feburary 23rd, 1996

Issue #13 - May 1st, 1996

Issue #14 - June 1st, 1996

Issue #15 - August 1st, 1996

Issue #16 - September 1st, 1996

Issue #17 - October 1st, 1996

Issue #18 - November 1st, 1996

Issue #19 - January 1st, 1997

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/undu/index.htm`. 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.

Tips & Tricks

In Issue #19 of UNDU, Paul Harding presented an article on how to determine if another application was running. Magnus Baeck shows an even simpler approach in "Is Someone Else Running" - Revisited!
Some of you may be familiar with the InputQuery function in Delphi's DIALOG.PAS unit. Wouldn't it be nice to have a InputQuery that provided a history list or prior selections? Gene Fowler shows us how with his InputQueryEx function.
Paul Harding is also back with a new tip on displaying multi-colored text in a string grid. Interestingly, this technique can be expanded into a wide number of other capabilities and even answers one of the Questions From Readers last month!

Do you need to publish Delphi/Pascal source code on a web site? It can be quite a pain having to do all that formatting by hand. But a new freeware package allows you to make it quick and painless. Check out the announcement from Pieter Polak on Converting Pascal Source to HTML.

Duncan Campbell also brings us a quick and easy database tip this month. He shows a way of improving the performance of processing large database tables.

In last issue, there was a question from readers about how to make your application look and behave correctly at differing resolutions. Check out Borland's Tech Sheet #2861 that discusses the issue very clearly.

Last issue, I discussed how to use the SHFileOperation command in the Win95 API to copy/move/delete/rename files and to add system-level undo support to these actions. Well, I left out one important thing!

Another interesting technique is presented by Eric Fortier discussing the impact of stored properties on EXE size. You can read about it in his tip on How to Make Your EXE's Lighter!

But there is a lot more! Check out the other tips in this issue: Form Aspect Ratio, Previous Instances Revisited and an alternate way of Printing Raw data to the Printer.

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

The Component Cookbook

This month, we have a couple of great new tips for you! First up is Robby Walker's "Tip Of The Day" Component. With it you can add a professional looking daily tip dialog just like Windows 95 gives you.
Emmanuel Fayet also presents a TFieldPanel component that simplifies gathering multi-field information from users. Check it out!
Also, an error crept into the TPageControl Flambe' article last month by Grahame Marsh. Fortunately, he caught it before everyone got too burnt, so he presents a quick revision this month.
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 (`RobertV@compuserve.com`) 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**.

Thats all there is to it!

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Learning To Drive

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

A common feature of many programs is the display of information relating to disc storage, either locally or over a network. These notes relate to some initial probing into the Win95 API: I give a series of utility procedures and functions which encapsulate the API calls into (I think) easier to use forms. I round off this article with an application illustrating their use.

How Many, Which?

The first question to answer is what drives are available? The API GetLogicalDrives function returns an integer representing the set of drive letters, bit 0 in the set represents drive A and so on. This set can be scanned and a TString filled with the available drive root directories:

```
procedure GetLogicalDriveList (List : TStrings);
var
  Num  : integer;
  Bits : set of 0..25;
begin
  List.Clear;
  integer (Bits) := Windows.GetLogicalDrives;
  for Num := 0 to 25 do
    if Num in Bits then
      List.Add (Char (Num + Ord('A')) + ':\')
end;
```

Using this it becomes very easy to fill, say, a combo box with a list of drives:

```
GetLogicalDriveList (ComboBox1.Items);
```

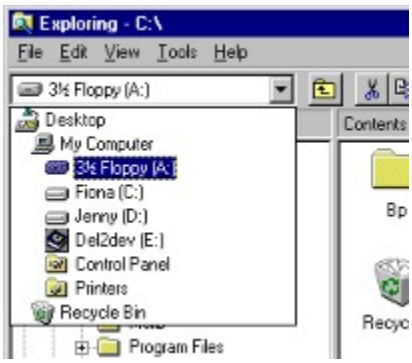
What?

The second question is "What are the drives?". The API GetDriveType returns a value which can be cast to a simple list of drive types covering floppy, fixed, removable, remote and CD-ROM.

```
type
  TDriveType = (dtUnknown, dtNoDrive, dtFloppy, dtFixed, dtNetwork, dtCDROM,
                dtRAM, dtFloppy3, dtFloppy5);

function GetDriveType (Drive : Char) : TDriveType;
begin
  Result := TDriveType (Windows.GetDriveType(PChar(Drive + ':\')))
end;
```

However, the last two, dtFloppy3 and dtFloppy5 are not returned by the API call, it only returns dtFloppy. To find out which kind of floppy takes a bit more work. I started knowing that my drive A is 3½ " floppy. I bought it, I plugged it in, I told the BIOS set up about it, even Win 95 knows the drive size:



So how can I find out? The API call GetDiskFreeSpace (more on this call later) can obviously be used to deduce a drive size, but it requires a floppy to be present in the drive, and I wanted a routine which doesn't actually spin the drive. After rummaging though the Win32 programmers reference help file, I came up with the following:

```
function FloppyDriveSize (Drive : char) : TDriveType;
type
  PDIOC_REG = ^TDIOC_Registers;
  TDIOC_Registers = record
    Reg_EBX, Reg_EDX, Reg_ECX, Reg_EAX,
    Reg_EDI, Reg_ESI, Reg_Flags : DWORD
  end;
const
  VWIN32_DIOC_DOS_INT13 = 4;
var
  H : THandle;
  R : TDIOC_Registers;
  C : DWORD;
begin
  Result := dtFloppy;
  H := CreateFile ('\\.\VWIN32', 0, 0, nil, 0, 0, 0);
  if H <> INVALID_HANDLE_VALUE then
  try
    R.Reg_EAX := $800;
    R.Reg_EDX := ord (upcase(Drive)) - Ord('A');
    if DeviceIOControl (H, VWIN32_DIOC_DOS_INT13, @R,
         SizeOf (R), @R, SizeOf (R), C, nil) and
         (R.Reg_Flags and 1 = 0) then
      if R.Reg_EBX and $FF < 3 then
         Result := dtFloppy5
      else
         Result := dtFloppy3
  finally
    CloseHandle (H)
  end
end;
```

Looks horrible doesn't it? I arrived at this code knowing that the BIOS had the value I wanted. Back in the good old days of DOS I would not have hesitated: INT 13 service 8 returns, inter alia, a value representing floppy drive size. It turns out that interrupt 13 services are available through the VWIN32 virtual device driver (VxD). To make the "interrupt" call, you first obtain a handle to the driver, the rather strange filename, "\\.\VWIN32" tells CreateFile that this is a device driver. The actual call is made using the API DeviceIOControl call. This takes the VxD handle, a set of registers for input and returns a set of registers as output. In this case I use the same variable for input and output. The input conditions required are a drive number (A = 0) in DL and the service number in AH. The call is now made to DeviceIOControl. The registers now are returned with BL representing a drive size (1 = 360K 5¼", 2 = 1.2M 5¼", 3 = 720K 3½ ", 4 = 1.44M 3½ ", 5 = 2.88M 3½ ", and so on, well, this is as far as my BIOS goes) and the carry flag (least significant bit of the flags) cleared to indicate success. It is now simple to

select either dtFloppy5 or dtFloppy3 drive type. Of course, you could split the drive type into the two 5¼" types and the many 3½ " types, but my aim was to display to the same level as Win 95. In practice you can use the utility functions something like:

```
var
  DriveType : TDriveType
....
  DriveType := GetDriveType(Drive);
  if DriveType = dtFloppy then
    DriveType := FloppyDriveSize (Drive);
```

What? A second way!

Another way of finding out about the drive types in a system is to use the Win 95 SHGetFileInfo shell function call. This function returns information about files, directories, folders and, usefully in this case, disc drives. You tell the function what drive interests you, what information you want (as a series of SHGFI_ constants) and it fills a TSHFileInfo record with the information requested. I have encapsulated the call into this procedure:

```
type
  TDriveShellInfo = record
    Icon : hIcon;
    Image : integer;
    DisplayName,
    TypeName : string
  end;

procedure GetDriveShellInfo (Drive : Char; var Info : TDriveShellInfo);
var
  SHFileInfo : TSHFileInfo;
begin
  ShGetFileInfo (PChar (Drive + ':\'), 0, SHFileInfo, SizeOf (TSHFileInfo),
     SHGFI_TYPENAME or SHGFI_DISPLAYNAME or SHGFI_SYSICONINDEX or SHGFI_ICON);
  with Info do
  begin
    Icon  := SHFileInfo.hIcon;
    Image := SHFileInfo.iIcon;
    DisplayName := SHFileInfo.szDisplayName;
    TypeName := SHFileInfo.szTypeName
  end
end;
```

The icon is a handle to the drive icon, image is the icon number in the system's image list (more on this later), display name returns the exact text that appears in the explorer combo box Drivelabel (C:) for a hard disc and 3½ Floppy (A:) for a floppy.

Volume Control

If your application needs some specific information about a particular disc, it can call the GetVolumeInformation API. This I have encapsulated as:

```
type
  TVolumeInformation = record
    VolumeName  : string;
    VolumeSerialNumber,
    MaximumComponentLength,
    FileSystemFlags : integer;
    FileSystemName : string;
  end;

function GetVolumeInformation (D : char; var V : TVolumeInformation) : boolean;
var
  O : integer;
```

```
      begin
        O := SetErrorMode (SEM_FAILCRITICALERRORS);
        try
          with V do
          begin
            SetLength (VolumeName, MAX_PATH);
            SetLength (FileSystemName, MAX_PATH);
            VolumeSerialNumber := 0;
            MaximumComponentLength := 0;
            FileSystemFlags := 0;
            Result := Windows.GetVolumeInformation (PChar (D+':\'), PChar
(VolumeName), MAX_PATH,
              @VolumeSerialNumber, MaximumComponentLength, FileSystemFlags,
              PChar (FileSystemName), MAX_PATH);
            RealizeLength (VolumeName);
            RealizeLength (FileSystemName)
          end
        finally
          SetErrorMode (O)
        end
      end;
```

VolumeName : The name of the specified volume.

VolumeSerialNumber : The volume serial number.

MaximumComponentLength : This value is the maximum length, in characters, of a filename component supported by the specified file system. A filename component is that portion of a filename between backslashes. The MaximumComponentLength is used to indicate that long names are supported by the specified file system. For example, for a FAT file system supporting long names, the function stores the value 255, rather than the previous 8.3 indicator. Long names can also be supported on systems that use the NTFS and HPFS file systems.

FileSystemFlags : These are flags associated with the specified file system. This parameter can be any combination of the following flags, with one exception: FS_FILE_COMPRESSION and FS_VOL_IS_COMPRESSED are mutually exclusive.

Value	**Meaning**
FS_CASE_IS_PRESERVED	If this flag is set, the file system preserves the case of filenames when it places a name on disk.
FS_CASE_SENSITIVE	If this flag is set, the file system supports case-sensitive filenames.
FS_UNICODE_STORED_ON_DISK	If this flag is set, the file system supports Unicode in filenames as they appear on disk.
FS_PERSISTENT_ACLS	If this flag is set, the file system preserves and enforces access control lists. For example, NTFS preserves and enforces ACLs, HPFS and FAT do not.
FS_FILE_COMPRESSION	The file system supports file-based compression.
FS_VOL_IS_COMPRESSED	The specified volume is a compressed volume; for example, a DoubleSpace volume.

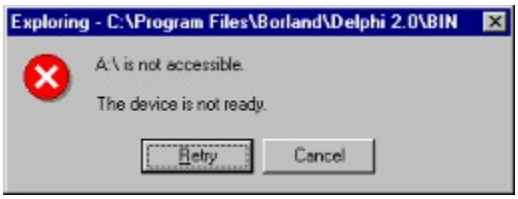
FileSystemName : The name of the file system (such as FAT, HPFS, CDFS or NTFS).

Of this data, the volume name and volume serial number is of most use. The name can be used to

identify a disc of a certain type (during say an installation process). The serial number is useful since it can be used to ensure that the punter has not, say, changed the floppy disc when writing multiple files to it.

Retry Cancel

Before continuing with drive utilities it is worth examining the behavior of Win 95 when their is a disc fault, say, no disc in the drive, then a retry-cancel dialog appears:



It is fairly simple to emulate this behavior by ensuring that each function that can cause an error suppresses critical errors and returns a boolean value to indicate success. You can then wrap the function in a while..do only allowing exit on success or cancel. The retry function in the utilities looks like this:

```
function DiscErrorMessage (Drive : Char) : string;
begin
  Result := Format ('%s:\ is not accessible.'#13#10#13#10+'%s',
                    [uppercase(Drive), SysErrorMessage (GetLastError)])
end;

function Retry (Drive : Char) : boolean;
begin
  Result := Application.MessageBox (
              PChar (DiscErrorMessage (Drive)),
              PChar (Application.Title),
              mb_RetryCancel or mb_IconError) = idRetry
end;
```

I have left this simple on purpose, if a more complex title is required it would be easy enough to write a more specialized retry function. One further level of digression is to look at the SysErrorMessage function. This is in the SysUtils unit. It takes am error code and returns a text message translating the code (eg error code 3 translates to "The system cannot find the path specified.". The version in SysUtils strips off the trailing full stop and CRLF. Additionally, it has no provision for additional parameters, error message 34 returns "The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.". It is simple enough to provide the parameters by re-writing the SysErrorMessage function as follows (and, for my preference not strip off the training .CRLF) :

```
function SysErrorMessageParams(ErrorCode:Integer;const Params:array of string):
string;
const
  L = 255;
begin
  SetLength (Result, L);
  FormatMessage (FORMAT_MESSAGE_FROM_SYSTEM or FORMAT_MESSAGE_ARGUMENT_ARRAY,
      nil, ErrorCode, 0, PChar(Result), L, @Params);
  RealizeLength (Result)
end;

function SysErrorMessage (ErrorCode: Integer): string;
begin
  Result := SysErrorMessageParams (ErrorCode, [''])
end;
```

This does leave you with some work to provide the parameters needed for certain error messages, but at least the error messages themselves are available, and there are, in fact, only a few that take parameters.

So, getting back to Retry-Cancel, you can wrap a call to GetVolumeInformation to show a Retry-Cancel dialog, with an error message using:

```
while not GetVolumeInformation (Drive, VolumeInfo) do
   if not Retry (Drive) then Abort;
```

Total and Free Space

Further information on a particular disc in a drive can be obtained using the GetDiskFreeSpace API call. This returns data needed to calculate the total and free space on a drive:

```
type
  TDiscFreeSpace = record
    SectorsPerCluster,
    BytesPerSector,
    NumberOfFreeClusters,
    TotalNumberOfClusters,
    TotalSpace,
    FreeSpace : integer
  end;

function GetDiscFreeSpace (Drive : char; var D : TDiscFreeSpace) : boolean;
var
  O : integer;
begin
  FillChar (D, Sizeof (TDiscFreeSpace), 0);
  O := SetErrorMode (SEM_FAILCRITICALERRORS);
  try
    with D do
    begin
      Result := Windows.GetDiskFreeSpace (PChar (Drive + ':\'),
SectorsPerCluster,
        BytesPerSector, NumberOfFreeClusters, TotalNumberOfClusters);
      FreeSpace := BytesPerSector*SectorsPerCluster*NumberOfFreeClusters;
      TotalSpace := BytesPerSector*SectorsPerCluster*TotalNumberOfClusters
    end
  finally
    SetErrorMode (O)
  end
end;
```

The returned record titles I hope are self explanatory - you won't learn much more from the Win 32 API help. Obviously the two calculated values are going to be the most useful and their difference gives the used space.

A picture is worth a thousand words

Above I used the SHGetFileInfo call to obtain information about a drive. I now will expand on this most useful function, and, in particular, look at the system image list. There are in fact two system image lists, large images and small images. You can set-up a TImageList component to contain say the large images by

```
var
  Images : TImageList;
  SHFileInfo : TSHFileInfo;
....
  Images := TImageList.Create (nil);
  Images.ShareImages := true;
  Images := ShGetFileInfo ('*.*', 0, SHFileInfo, SizeOf (TSHFileInfo),
              SHGFI_LARGEICON or SHGFI_SYSICONINDEX);
```

In this case the data put into the SHFileInfo structure is of no interest, the function itself returns a handle to a TImageList. The discutil unit obtains image lists of both the large and small icons and holds them

globally available. You can view the image list using this simple form.

Code listings:

Images1.pas - Form Code File

Images1.dfm - Form File

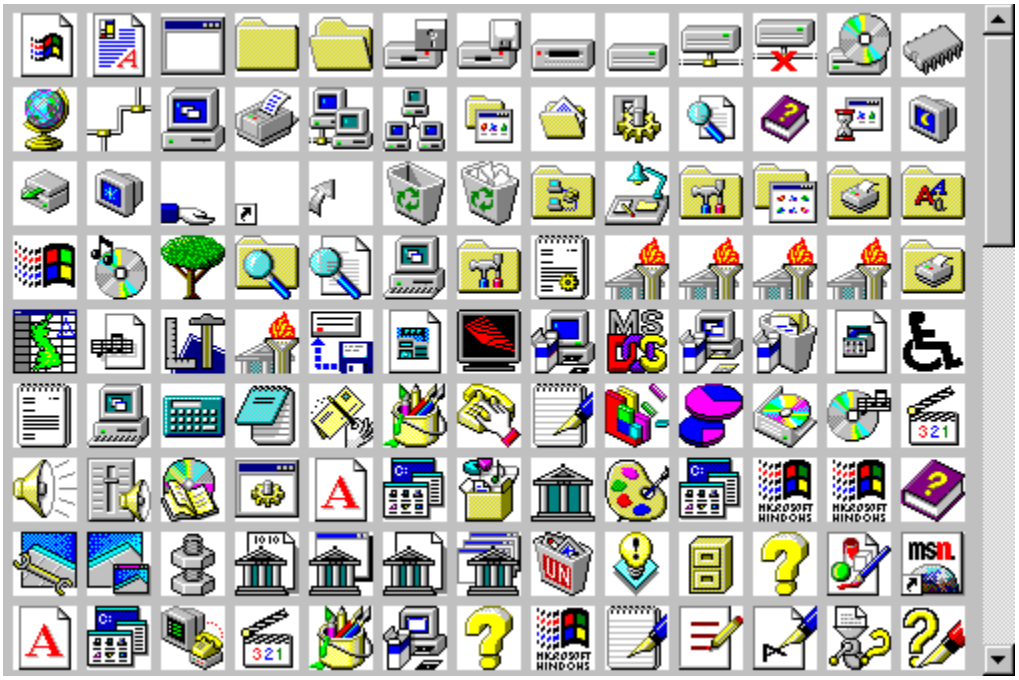
DI1.pas- Example form code file

DI1.dfm- Example form file

DI.dpr- Example project file

Also needs DiscUtils.pas

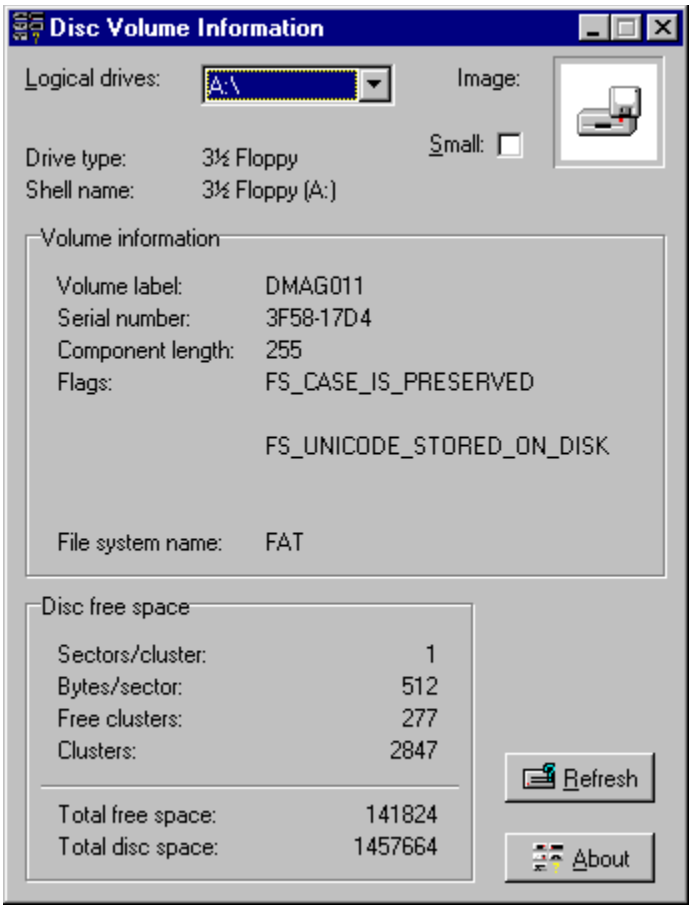
On my computer this produces a list, the top part of which looks like this:

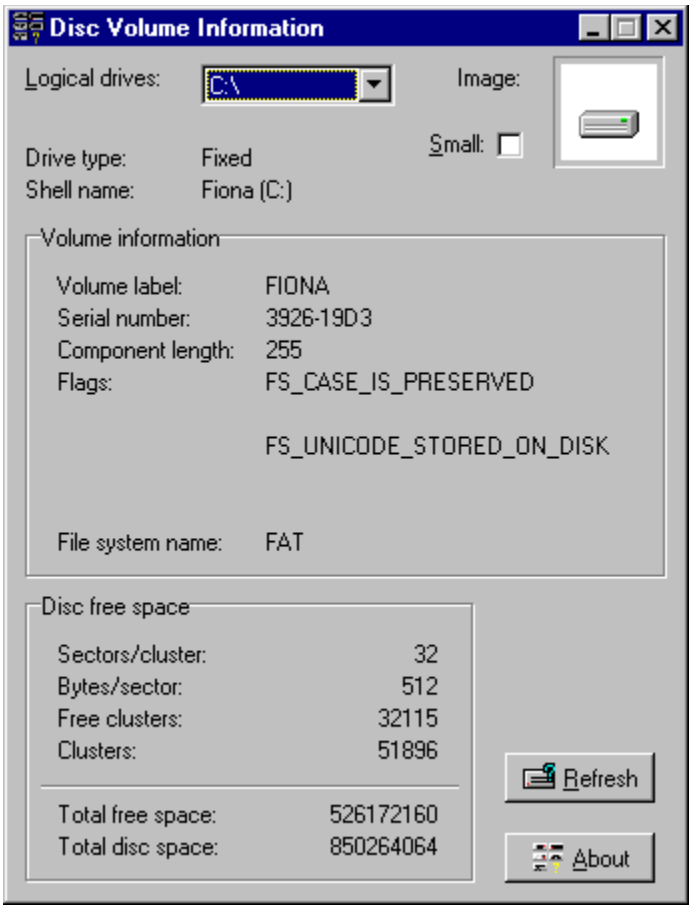


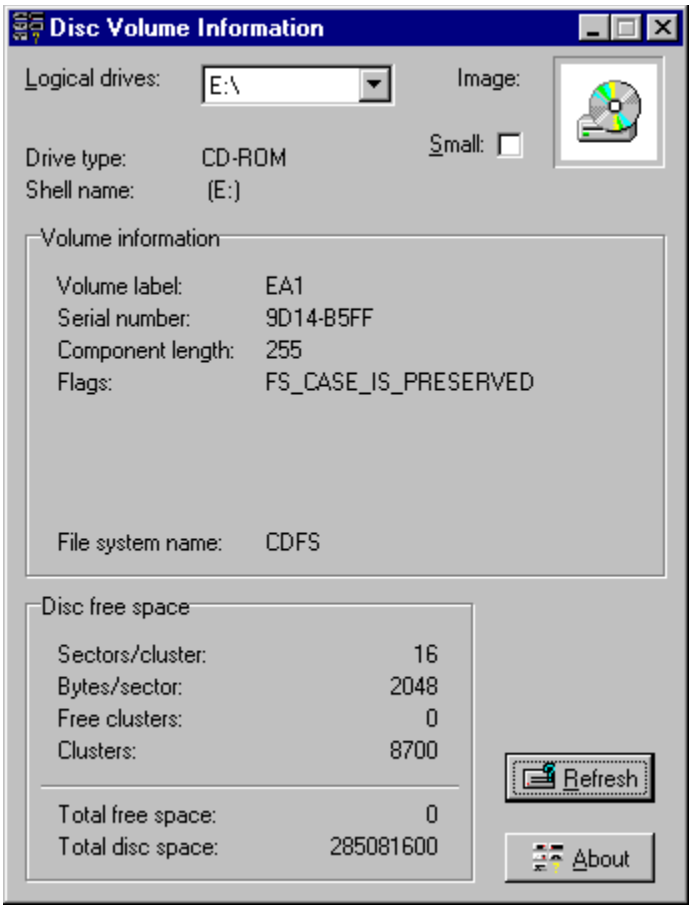
There is two thing that you can say - boy! what a good collection of icons, but also, the system icons (folders etc.) are mixed up with application icons which are machine specific. So to use these icons you must either extract them and save them yourself, or obtain the image number in the list to reference an image from the system because I don't think you can rely on the absolute location of any image being the same on any other machine. The obtaining the reference number is what I have done in the utilities for the drive images given in the top row.

Application

To illustrate all of these drive utilities I have re-written an example program given by Richter (Advanced Windows, Jeffrey Richter, Microsoft Press, ISBN 1-55615-677-4) in Delphi and added the icon information. When run it displays the available information from the utility functions. These screen shots show a floppy, hard disc and CD-ROM drive:


Disc Volume Information
Logical drives:
A:\
Image:
Small:
Drive type:
3½ Floppy
Shell name:
3½ Floppy (A:)
Volume information
Volume label:
DMAG011
Serial number:
3F58-17D4
Component length:
255
Flags:
FS_CASE_IS_PRESERVED
FS_UNICODE_STORED_ON_DISK
File system name:
FAT
Disc free space
Sectors/cluster:
1
Bytes/sector:
512
Free clusters:
277
Clusters:
2847
Total free space:
141824
Total disc space:
1457664
Refresh
About


Disc Volume Information
Logical drives:
C:\
Image:
Small:
Drive type: Fixed
Shell name: Fiona (C:)
Volume information
Volume label: FIONA
Serial number: 3926-19D3
Component length: 255
Flags: FS_CASE_IS_PRESERVED
FS_UNICODE_STORED_ON_DISK
File system name: FAT
Disc free space
Sectors/cluster: 32
Bytes/sector: 512
Free clusters: 32115
Clusters: 51896
Total free space: 526172160
Total disc space: 850264064
Refresh
About



One advantage of dynamically obtaining an icon of a drive, is that a CD_ROM drive can have an icon on it which will be substituted for the system icon:

Afterthoughts

I would predict that every programmer of Delphi has a unit where useful bits and pieces not otherwise in Delphi are collected. This will be called something like Utils or Useful. My intention is that these disc utilities are not a static set to put somewhere and used as a block together, but they should be merged into your general set of useful stuff you have in your own development area.

Most of these utilities are Win 95 specific, I have new computer arriving soon which, I hope will have NT 4 on it, so I will test and modify these utilities to suit. But I have no interest in backward compatibility to NT 3.5.

Finally, I am working towards a replacement unit for FileCtrl which contains Win 95 look-a-like components. Obtaining the drive information and drive icons is an important step towards this goal. But this will be a future article for UNDU…

Return to Front Page

Source for Images1.Pas

```
unit images1;

interface

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

type
  TForm1 = class(TForm)
    ScrollBox1: TScrollBox;
    PaintBox1: TPaintBox;
    procedure PaintBox1Click(Sender: TObject);
  private
  public
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

uses
  DiscUtil;

procedure TForm1.PaintBox1Click(Sender: TObject);
const
  Margin = 5;
var
  XPos,
  YPos,
  Loop : integer;
begin
  PaintBox1.Width  := ClientWidth - Margin * 4;
  PaintBox1.Height := LargeImages.Count * (LargeImages.Height + Margin) div
                        pred (PaintBox1.Width div (LargeImages.Width+Margin)) +
Margin * 2;
  XPos := Margin;
  YPos := Margin;

  for loop := 0 to LargeImages.Count - 1 do
  begin
    LargeImages.Draw (PaintBox1.Canvas, XPos, YPos, Loop);

    XPos := XPos + LargeImages.Width + Margin;
    if XPos + LargeImages.Width + Margin > PaintBox1.Width  then
    begin
      XPos := Margin;
      YPos := YPos + LargeImages.Height + Margin
    end;
  end;
end;

{$O-}

(*
//'c:\program files\borland\delphi 2.0\bin\delphi32.exe'
```

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
var
  S : TSHFileInfo;
  E,
  Z : integer;
  D1, D2,
  D, T : string;

  I : TIcon;
  L : TImageList;
  H : HResult;
  U,
  V : IShellFolder;
  K : IEnumIDList;
  W,
  Q : PItemIDList;
  R : ULONG;
type
  PA = array [0..24] of char;
var
  A : ^PA;
type
  PSSS = ^TSSS;
  TSSS = packed record
    Size : word;
    Head1,
    Head2 : byte;
    Data : TGUID
  end;

var
  X : PSSS;
begin
  H := SHGetSpecialFolderLocation (0, CSIDL_DRIVES, W);
  X := pointer (W);
  A := pointer (W);
  X^.Head1 := 31;
  Z := ShGetFileInfo (pointer(W), 0, S, SizeOf (S),
          SHGFI_DISPLAYNAME or SHGFI_SYSICONINDEX or SHGFI_PIDL);

  H := SHGetDesktopFolder (U);
  H := U.BindToObject (W, nil, IID_ISHELLFOLDER, pointer(V));

   H := V.EnumObjects (Handle, $FF, K);
   repeat
     H := K.Next (1, Q, R);
     if H = 0 then
     begin
       X := pointer (Q);
       A := pointer (Q);
       if X^.Head1 = $2E then
       begin
         X^.Head1 := X^.Head1 -15;
         Z := ShGetFileInfo (pointer(Q), 0, S, SizeOf (S),
            SHGFI_TYPENAME or SHGFI_ICON or SHGFI_DISPLAYNAME or SHGFI_SYSICONINDEX or
            SHGFI_PIDL or SHGFI_ATTRIBUTES);
       end else begin
         D1 := Char (X^.Head2) + ':\';
         Z := ShGetFileInfo (PChar(D1), 0, S, SizeOf (S),
            SHGFI_TYPENAME or SHGFI_ICON or SHGFI_DISPLAYNAME or SHGFI_SYSICONINDEX or
            SHGFI_ATTRIBUTES);
       end;
```

```
         D1 := S.szDisplayName;
         D2 := S.szTypeName;

   I := TIcon.Create;
   I.Handle := S.hIcon;
   Image1.Picture.Icon := I;
   I.Free;
   Label1.Caption := S.szDisplayName;
   Label2.Caption := S.szTypeName;
       Application.ProcessMessages;
       Sleep (1000)
     end
   until H <> 0;

   H := SHGetSpecialFolderLocation (0, CSIDL_BITBUCKET, W);
   Z := ShGetFileInfo (pointer(W), 0, S, SizeOf (S),
      SHGFI_TYPENAME or SHGFI_ICON or SHGFI_DISPLAYNAME or SHGFI_SYSICONINDEX or
      SHGFI_PIDL);
   E := GetLastError;
   Label1.Caption := S.szDisplayName;
   Label2.Caption := S.szTypeName;

   I := TIcon.Create;
   I.Handle := S.hIcon;
   Image1.Picture.Icon := I;
   I.Free;

   Beep
end;    *)

(*
type
  TSHFileInfoA = record
    hIcon: HICON;                     { out: icon }
    iIcon: Integer;                   { out: icon index }
    dwAttributes: DWORD;              { out: SFGAO_ flags }
    szDisplayName: array [0..MAX_PATH-1] of  AnsiChar; { out: display name (or path) }
    szTypeName: array [0..79] of AnsiChar;             { out: type name }
  end;  *)

end.
```

Source for Images1.dfm

```
object Form1: TForm1
  Left = 388
  Top = 252
  Width = 349
  Height = 293
  Caption = 'Form1'
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  PixelsPerInch = 96
  TextHeight = 13
  object ScrollBox1: TScrollBox
    Left = 0
    Top = 0
    Width = 341
    Height = 266
    Align = alClient
    BorderStyle = bsNone
    TabOrder = 0
    object PaintBox1: TPaintBox
      Left = 0
      Top = 0
      Width = 129
      Height = 105
      OnPaint = PaintBox1Click
    end
  end
end
```

Source for DiscUtils.Pas

```
unit DiscUtil;

interface

uses Windows, SysUtils, Controls, ShellAPI, Classes, Forms;

//--- Error handling -------------------------------------------------------

function DiscErrorMessage (Drive : Char) : string;
function FormatVolumeSerialNumber (N : integer) : string;
function Retry (Drive : Char) : boolean;
function SysErrorMessage (ErrorCode: Integer): string;
function SysErrorMessageParams (ErrorCode: Integer;
                const Params : array of string): string;

//--- Disc volume information ----------------------------------------------

type
  TVolumeInformation = record
    VolumeName  : string;
    VolumeSerialNumber,
    MaximumComponentLength,
    FileSystemFlags : integer;
    FileSystemName : string;
  end;

function GetNetworkVolumeName (D : char): string;
function GetVolumeInformation (D : char; var V : TVolumeInformation) : boolean;
function GetVolumeName (D : char) : string;

//--- System image lists ---------------------------------------------------

var
  SmallImages,
  LargeImages : TImageList;

//--- Types of disc drive --------------------------------------------------

type
  TDriveType = (dtUnknown, dtNoDrive, dtFloppy, dtFixed, dtNetwork, dtCDROM,
                dtRAM, dtFloppy3, dtFloppy5);

const
  DriveNames : array [TDriveType] of string[9] =
    ('Unknown', 'None', 'Floppy', 'Fixed', 'Network', 'CD-ROM',
     'RAM', '3½ Floppy', '5¼ Floppy');

function FloppyDriveSize (Drive : char) : TDriveType;
function GetDriveType (Drive : Char) : TDriveType;

type
  TDriveShellInfo = record
    Icon : hIcon;
    Image : integer;
    DisplayName,
    TypeName : string
  end;

procedure GetDriveShellInfo (Drive : Char; var Info : TDriveShellInfo);
```

```
//--- Information on available drives ----------------------------------------

procedure GetLogicalDriveList (List : TStrings);

type
  TDiscFreeSpace = record
    SectorsPerCluster,
    BytesPerSector,
    NumberOfFreeClusters,
    TotalNumberOfClusters,
    TotalSpace,
    FreeSpace : integer
  end;

function GetDiscFreeSpace (Drive : char; var D : TDiscFreeSpace) : boolean;

//----------------------------------------------------------------------------

implementation

//--- Internal

procedure RealizeLength(var S: string);
begin
  SetLength (S, StrLen (PChar(S)))
end;

//--- Error handling

function SysErrorMessageParams (ErrorCode: Integer; const Params : array of string):
string;
const
  L = 255;
begin
  SetLength (Result, L);
  FormatMessage (FORMAT_MESSAGE_FROM_SYSTEM or FORMAT_MESSAGE_ARGUMENT_ARRAY,
      nil, ErrorCode, 0, PChar(Result), L, @Params);
  RealizeLength (Result)
end;

function SysErrorMessage (ErrorCode: Integer): string;
begin
  Result := SysErrorMessageParams (ErrorCode, [''])
end;

function FormatVolumeSerialNumber (N : integer) : string;
begin
  Result := Format ('%X-%X', [longrec(N).hi, longrec(N).lo])
end;

function DiscErrorMessage (Drive : Char) : string;
begin
  Result := Format ('%s:\ is not accessible.'#13#10#13#10+'%s',
                    [uppercase(Drive), SysErrorMessage (GetLastError)])
end;

function Retry (Drive : Char) : boolean;
begin
  Result := Application.MessageBox (
              PChar (DiscErrorMessage (Drive)),
              PChar (Application.Title),
              mb_RetryCancel or mb_IconError) = idRetry
end;
```

```
//--- Volume Information

function GetVolumeInformation (D : char; var V : TVolumeInformation) : boolean;
var
  O : integer;
begin
  O := SetErrorMode (SEM_FAILCRITICALERRORS);
  try
    with V do
    begin
      SetLength (VolumeName, MAX_PATH);
      SetLength (FileSystemName, MAX_PATH);
      VolumeSerialNumber := 0;
      MaximumComponentLength := 0;
      FileSystemFlags := 0;
      Result := Windows.GetVolumeInformation (PChar (D+':\'), PChar (VolumeName),
MAX_PATH,
        @VolumeSerialNumber, MaximumComponentLength, FileSystemFlags,
        PChar (FileSystemName), MAX_PATH);
      RealizeLength (VolumeName);
      RealizeLength (FileSystemName)
    end
  finally
    SetErrorMode (O)
  end
end;

function GetVolumeName (D : char) : string;
var
  T : TVolumeInformation;
begin
  if GetVolumeInformation (D, T) then
    Result := T.VolumeName
  else
    Result := ''
end;

function GetNetworkVolumeName (D : char): string;
var
  L : integer;
begin
  L := MAX_PATH;
  SetLength (Result, L);
  if WNetGetConnection (PChar(D+':'#0), PChar(Result), L) = NO_ERROR then
    RealizeLength (Result)
  else
    Result := GetVolumeName(D)
end;

//--- floppy disc size determination

function FloppyDriveSize (Drive : char) : TDriveType;
type
  PDIOC_REG = ^TDIOC_Registers;
  TDIOC_Registers = record
    Reg_EBX, Reg_EDX, Reg_ECX, Reg_EAX, Reg_EDI, Reg_ESI, Reg_Flags : DWORD
  end;
const
  VWIN32_DIOC_DOS_INT13 = 4;  // Performs Interrupt 13h commands.
var
  H : THandle;
  R : TDIOC_Registers;
```

```
  C : DWORD;
begin
  Result := dtFloppy;
  H := CreateFile ('\\.\VWIN32', 0, 0, nil, 0, 0, 0);
  if H <> INVALID_HANDLE_VALUE then
  try
    R.Reg_EAX := $800;                    // service 8 in AH
    R.Reg_EDX := ord (Upcase(Drive)) - Ord('A');  // drive number in DL
    if DeviceIOControl (H, VWIN32_DIOC_DOS_INT13, @R, SizeOf (R), @R, SizeOf (R), C,
nil)
       and (R.Reg_Flags and 1 = 0) then   // clear CF indicates success
      if R.Reg_EBX and $FF < 3 then       // drive type in BL
         Result := dtFloppy5              // 1 = 360K, 2 = 1.2MB
      else
         Result := dtFloppy3              // 3 = 720K  4 = 1.44MB  5 = 2.88MB
  finally
    CloseHandle (H)
  end
end;

//--- Available drive information

function GetDiscFreeSpace (Drive : char; var D : TDiscFreeSpace) : boolean;
var
  O : integer;
begin
  FillChar (D, Sizeof (TDiscFreeSpace), 0);
  O := SetErrorMode (SEM_FAILCRITICALERRORS);
  try
    with D do
    begin
      Result := Windows.GetDiskFreeSpace (PChar (Drive + ':\'), SectorsPerCluster,
        BytesPerSector, NumberOfFreeClusters, TotalNumberOfClusters);
      FreeSpace := BytesPerSector*SectorsPerCluster*NumberOfFreeClusters;
      TotalSpace := BytesPerSector*SectorsPerCluster*TotalNumberOfClusters
    end
  finally
    SetErrorMode (O)
  end
end;

procedure GetLogicalDriveList (List : TStrings);
var
  Num  : integer;
  Bits : set of 0..25;
begin
  List.Clear;
  integer (Bits) := Windows.GetLogicalDrives;
  for Num := 0 to 25 do
    if Num in Bits then
      List.Add (Char (Num + Ord('A')) + ':\')
end;

function GetDriveType (Drive : Char) : TDriveType;
begin
  Result := TDriveType (Windows.GetDriveType(PChar(Drive + ':\')))
end;

procedure GetDriveShellInfo (Drive : Char; var Info : TDriveShellInfo);
var
  SHFileInfo : TSHFileInfo;
begin
  ShGetFileInfo (PChar (Drive + ':\'), 0, SHFileInfo, SizeOf (TSHFileInfo),
```

```
     SHGFI_TYPENAME or SHGFI_DISPLAYNAME or SHGFI_SYSICONINDEX or SHGFI_ICON);
  with Info do
  begin
    Icon  := SHFileInfo.hIcon;
    Image := SHFileInfo.iIcon;
    DisplayName := SHFileInfo.szDisplayName;
    TypeName := SHFileInfo.szTypeName
  end
end;

//--- unit start-up and close-down

function GetImageList (Option : integer) : TImageList;
var
  SHFileInfo : TSHFileInfo;
begin
  Result := TImageList.Create (nil);
  with Result do
  begin
    ShareImages := true;
    Handle := ShGetFileInfo ('*.*', 0, SHFileInfo, SizeOf (TSHFileInfo),
                  Option or SHGFI_SYSICONINDEX)
  end
end;

initialization
  LargeImages := GetImageList (SHGFI_LARGEICON);
  SmallImages := GetImageList (SHGFI_SMALLICON)
finalization
  LargeImages.Free;
  SmallImages.Free
end.
```

Source for DI1.Pas

```
unit di1;

interface

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

type
  TDiscVolInfoForm = class(TForm)
    DrivesCB: TComboBox;
    Label1: TLabel;
    GroupBox1: TGroupBox;
    Label2: TLabel;
    DriveTypeLabel: TLabel;
    RefreshBtn: TBitBtn;
    Panel1: TPanel;
    DriveImageBitmap: TImage;
    Bevel1: TBevel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    VolumeLabelLabel: TLabel;
    SerialNumberLabel: TLabel;
    FlagsLabel1: TLabel;
    FlagsLabel2: TLabel;
    FlagsLabel3: TLabel;
    FileSystemLabel: TLabel;
    ComponentLengthLabel: TLabel;
    FlagsLabel4: TLabel;
    FlagsLabel5: TLabel;
    GroupBox2: TGroupBox;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    SectorsPerClusterLabel: TLabel;
    BytesPerSectorLabel: TLabel;
    FreeClustersLabel: TLabel;
    ClustersLabel: TLabel;
    AboutBtn: TBitBtn;
    SmallCheckBox: TCheckBox;
    Label12: TLabel;
    Bevel2: TBevel;
    Label13: TLabel;
    Label14: TLabel;
    FreeSpaceLabel: TLabel;
    DiscSpaceLabel: TLabel;
    Label15: TLabel;
    ShellNameLabel: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FillInInfo(Sender: TObject);
    procedure AboutBtnClick(Sender: TObject);
    procedure FillInImage(Sender: TObject);
  private
    DriveImage : integer;
  public
  end;
```

```
var
  DiscVolInfoForm: TDiscVolInfoForm;

implementation

{$R *.DFM}

procedure TDiscVolInfoForm.FormCreate(Sender: TObject);
begin
  GetLogicalDriveList (DrivesCB.Items);
  DrivesCB.ItemIndex := DrivesCB.Items.IndexOf ('C:\');
  FillInInfo (Sender)
end;

procedure TDiscVolInfoForm.FillInInfo (Sender: TObject);
var
  Drive : char;
  DriveType : TDriveType;
  Loop : integer;
  Info : TDriveShellInfo;
  VolumeInfo : TVolumeInformation;
  DiscFreeSpace : TDiscFreeSpace;

begin
  for loop := 0 to ComponentCount -1 do
    if Components [Loop] is TLabel then
      with Components [Loop] as TLabel do
        case Tag of
          1 : Caption := '';
          2 : Caption := '0'
        end;

  Drive := DrivesCB.Items [DrivesCB.ItemIndex][1];
  DriveType := GetDriveType(Drive);
  if DriveType = dtFloppy then
  begin
    DriveType := FloppyDriveSize (Drive);
    Screen.Cursor := crHourglass
  end;

  try
    DriveTypeLabel.Caption := DriveNames [DriveType];
    GetDriveShellInfo (Drive, Info);
    DriveImage := Info.Image;
    FillInImage (Sender);
    ShellNameLabel.Caption := Info.DisplayName;

    while not GetVolumeInformation (Drive, VolumeInfo) do
      if not Retry (Drive) then Abort;

    with VolumeInfo do
    begin
      if DriveType = dtNetwork then
        VolumeLabelLabel.Caption := GetNetworkVolumeName (Drive)
      else
        if VolumeName = '' then
          VolumeLabelLabel.Caption := '[none]'
        else
          VolumeLabelLabel.Caption := VolumeName;

      SerialNumberLabel.Caption := FormatVolumeSerialNumber (VolumeSerialNumber);
```

```
      ComponentLengthLabel.Caption := IntToStr (MaximumComponentLength);

      if FileSystemFlags and FS_CASE_IS_PRESERVED <> 0 then
        FlagsLabel1.Caption := 'FS_CASE_IS_PRESERVED';

      if FileSystemFlags and FS_CASE_SENSITIVE <> 0 then
        FlagsLabel2.Caption := 'FS_CASE_SENSITIVE';

      if FileSystemFlags and FS_UNICODE_STORED_ON_DISK <> 0 then
        FlagsLabel3.Caption := 'FS_UNICODE_STORED_ON_DISK';

      if FileSystemFlags and FS_PERSISTENT_ACLS <> 0 then
        FlagsLabel4.Caption := 'FS_PERSISTENT_ACLS';

      if FileSystemFlags and FS_VOL_IS_COMPRESSED <> 0 then
        FlagsLabel5.Caption := 'FS_VOL_IS_COMPRESSED'
      else
        if FileSystemFlags and FS_FILE_COMPRESSION <> 0 then
          FlagsLabel5.Caption := 'FS_FILE_COMPRESSION';

      FileSystemLabel.Caption := FileSystemName
    end;

    while not GetDiscFreeSpace (Drive, DiscFreeSpace) do
      if not Retry (Drive) then Abort;

    with DiscFreeSpace do
    begin
      SectorsPerClusterLabel.Caption := IntToStr (SectorsPerCluster);
      BytesPerSectorLabel.Caption := IntToStr (BytesPerSector);
      FreeClustersLabel.Caption := IntToStr (NumberOfFreeClusters);
      ClustersLabel.Caption := IntToStr (TotalNumberOfClusters);
      FreeSpaceLabel.Caption := IntToStr (FreeSpace);
      DiscSpaceLabel.Caption := IntToStr (TotalSpace)
    end

  finally
    Screen.Cursor := crDefault
  end
end;

procedure TDiscVolInfoForm.FillInImage (Sender: TObject);
var
  DriveBitmap : TBitmap;
begin
  DriveBitmap := TBitmap.Create;
  try
    if DriveImage > 0 then
      if SmallCheckBox.Checked then
        SmallImages.GetBitmap (DriveImage, DriveBitmap)
      else
        LargeImages.GetBitmap (DriveImage, DriveBitmap);
    DriveImageBitmap.Picture.Bitmap := DriveBitmap
  finally
    DriveBitmap.Free
  end
end;

procedure TDiscVolInfoForm.AboutBtnClick(Sender: TObject);
begin
  Application.MessageBox (
     #13+
     'Grahame Marsh for'#13+
```

```
     'The Unofficial Newsletter of Delphi Users'#13+
     'January 1997'#13,
     PChar (Application.Title), mb_Ok or mb_IconInformation)
end;

end.
```

Source for DI1.dfm

```
object DiscVolInfoForm: TDiscVolInfoForm
  Left = 392
  Top = 233
  ActiveControl = DrivesCB
  BorderIcons = [biSystemMenu, biMinimize]
  BorderStyle = bsSingle
  Caption = 'Disc Volume Information'
  ClientHeight = 425
  ClientWidth = 337
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object Label1: TLabel
    Left = 8
    Top = 8
    Width = 68
    Height = 13
    Caption = '&Logical drives:'
    FocusControl = DrivesCB
  end
  object Label2: TLabel
    Left = 8
    Top = 48
    Width = 51
    Height = 13
    Caption = 'Drive type:'
  end
  object DriveTypeLabel: TLabel
    Tag = 1
    Left = 96
    Top = 48
    Width = 7
    Height = 13
    Caption = 'X'
  end
  object Bevel1: TBevel
    Left = 272
    Top = 4
    Width = 56
    Height = 56
  end
  object Label12: TLabel
    Left = 224
    Top = 8
    Width = 32
    Height = 13
    Caption = 'Image:'
  end
  object Label15: TLabel
    Left = 8
    Top = 64
    Width = 55
    Height = 13
    Caption = 'Shell name:'
  end
  object ShellNameLabel: TLabel
```

```
  Left = 96
  Top = 64
  Width = 9
  Height = 13
  Caption = 'X'
end
object DrivesCB: TComboBox
  Left = 96
  Top = 8
  Width = 97
  Height = 21
  Style = csDropDownList
  ItemHeight = 13
  TabOrder = 0
  OnChange = FillInInfo
end
object GroupBox1: TGroupBox
  Left = 8
  Top = 88
  Width = 321
  Height = 177
  Caption = 'Volume information'
  TabOrder = 5
  object Label3: TLabel
    Left = 16
    Top = 24
    Width = 63
    Height = 13
    Caption = 'Volume label:'
  end
  object Label4: TLabel
    Left = 16
    Top = 40
    Width = 67
    Height = 13
    Caption = 'Serial number:'
  end
  object Label5: TLabel
    Left = 16
    Top = 56
    Width = 89
    Height = 13
    Caption = 'Component length:'
  end
  object Label6: TLabel
    Left = 16
    Top = 72
    Width = 28
    Height = 13
    Caption = 'Flags:'
  end
  object Label7: TLabel
    Left = 16
    Top = 152
    Width = 83
    Height = 13
    Caption = 'File system name:'
  end
  object VolumeLabelLabel: TLabel
    Tag = 1
    Left = 120
    Top = 24
    Width = 7
```

```
  Height = 13
  Caption = 'X'
end
object SerialNumberLabel: TLabel
  Tag = 1
  Left = 120
  Top = 40
  Width = 7
  Height = 13
  Caption = 'X'
end
object FlagsLabel1: TLabel
  Tag = 1
  Left = 120
  Top = 72
  Width = 7
  Height = 13
  Caption = 'X'
end
object FlagsLabel2: TLabel
  Tag = 1
  Left = 120
  Top = 88
  Width = 7
  Height = 13
  Caption = 'X'
end
object FlagsLabel3: TLabel
  Tag = 1
  Left = 120
  Top = 104
  Width = 7
  Height = 13
  Caption = 'X'
end
object FileSystemLabel: TLabel
  Tag = 1
  Left = 120
  Top = 152
  Width = 7
  Height = 13
  Caption = 'X'
end
object ComponentLengthLabel: TLabel
  Tag = 1
  Left = 120
  Top = 56
  Width = 7
  Height = 13
  Caption = 'X'
end
object FlagsLabel4: TLabel
  Tag = 1
  Left = 120
  Top = 120
  Width = 7
  Height = 13
  Caption = 'X'
end
object FlagsLabel5: TLabel
  Tag = 1
  Left = 120
  Top = 136
```

```
    Width = 7
    Height = 13
    Caption = 'X'
  end
end
object RefreshBtn: TBitBtn
  Left = 248
  Top = 352
  Width = 75
  Height = 25
  Caption = '&Refresh'
  TabOrder = 2
  OnClick = FillInInfo
  Glyph.Data = {
    76010000424D760100000000000076000000280000002000000010000000100
    0400000000000001000000000000000000000000000000000000000000000000
    8000008000000080800080000000800080008080000C0C0C000808080000000
    FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFFFF00333333333333
    3333333333333333333333333333333333333333333333333333333333333333
    33333FFFFFFFFFFFFFF0000000000000000888888888888888F8888888888
    88808F3F3333333FFF3F0F977777777777703F833FFF33888F3F0F7700000000
    00003F338888338F8F3F0F77777777770E003F333FFF33888F3F0F7700000000
    00003F338888338F8F3F0F77777777770E003F3333333F8388FF0F7777777777
    0E003F3333338888388F0FFFFFFFFF0000E00FFFFFFF8F88838F000000000000E
    000E088888888F888F8F33333333330E000E0333333388FFF883333333333300
    EEE00333333338888833333333333300000033333333333333333}
  NumGlyphs = 2
end
object Panel1: TPanel
  Left = 276
  Top = 8
  Width = 48
  Height = 48
  BevelOuter = bvNone
  Color = clWhite
  TabOrder = 4
  object DriveImageBitmap: TImage
    Left = 8
    Top = 8
    Width = 32
    Height = 32
    Center = True
  end
end
object GroupBox2: TGroupBox
  Left = 8
  Top = 272
  Width = 225
  Height = 145
  Caption = 'Disc free space'
  TabOrder = 6
  object Label8: TLabel
    Left = 16
    Top = 24
    Width = 75
    Height = 13
    Caption = 'Sectors/cluster:'
  end
  object Label9: TLabel
    Left = 16
    Top = 40
    Width = 63
    Height = 13
```

```
  Caption = 'Bytes/sector:'
end
object Label10: TLabel
  Left = 16
  Top = 56
  Width = 63
  Height = 13
  Caption = 'Free clusters:'
end
object Label11: TLabel
  Left = 16
  Top = 72
  Width = 40
  Height = 13
  Caption = 'Clusters:'
end
object SectorsPerClusterLabel: TLabel
  Tag = 2
  Left = 200
  Top = 24
  Width = 6
  Height = 13
  Alignment = taRightJustify
  Caption = '0'
end
object BytesPerSectorLabel: TLabel
  Tag = 2
  Left = 200
  Top = 40
  Width = 6
  Height = 13
  Alignment = taRightJustify
  Caption = '0'
end
object FreeClustersLabel: TLabel
  Tag = 2
  Left = 200
  Top = 56
  Width = 6
  Height = 13
  Alignment = taRightJustify
  Caption = '0'
end
object ClustersLabel: TLabel
  Tag = 2
  Left = 200
  Top = 72
  Width = 6
  Height = 13
  Alignment = taRightJustify
  Caption = '0'
end
object Bevel2: TBevel
  Left = 8
  Top = 96
  Width = 209
  Height = 9
  Shape = bsTopLine
end
object Label13: TLabel
  Left = 16
  Top = 104
  Width = 80
```

```
      Height = 13
      Caption = 'Total free space:'
    end
    object Label14: TLabel
      Left = 16
      Top = 120
      Width = 81
      Height = 13
      Caption = 'Total disc space:'
    end
    object FreeSpaceLabel: TLabel
      Tag = 2
      Left = 198
      Top = 104
      Width = 8
      Height = 13
      Alignment = taRightJustify
      Caption = '0'
    end
    object DiscSpaceLabel: TLabel
      Tag = 2
      Left = 200
      Top = 120
      Width = 6
      Height = 13
      Alignment = taRightJustify
      Caption = '0'
    end
  end
  object AboutBtn: TBitBtn
    Left = 248
    Top = 392
    Width = 75
    Height = 25
    Caption = '&About'
    TabOrder = 3
    OnClick = AboutBtnClick
    Glyph.Data = {
      F6000000424DF600000000000000760000002800000010000000100000000100
      0400000000008000000000000000000000000000000000000000000000000000
      8000008000000080800080000000800080008080000000C0C0C000808080000000
      FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFFFF00333333333333
      333333333333333B33338080888333333333F7077773333B3333F00000733333
      B333FFFFFFF333BBB33333333333333333338888888388888883F7FF7973F007
      0073FF7F8073F7000773FFFFFFF3FFFFFFF33333333333333333888888838888
      8883F7000773F9000003F7000973F0000003FFFFFFF3FFFFFFF3}
  end
  object SmallCheckBox: TCheckBox
    Left = 208
    Top = 40
    Width = 49
    Height = 17
    Alignment = taLeftJustify
    Caption = '&Small:'
    TabOrder = 1
    OnClick = FillInImage
  end
end
```

Source for DI1.dpr

```
program di;

uses
  Forms,
  di1 in 'di1.pas' {DiscVolInfoForm},
  DiscUtil in 'DiscUtil.pas';

{$R *.RES}

begin
  Application.Initialize;
  Application.Title := 'Disc Volume Information';
  Application.CreateForm(TDiscVolInfoForm, DiscVolInfoForm);
  Application.Run;
end.
```

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Delphi Books & Periodicals

by Jim Clokey - jclokey@mtgbcs.mt.lucent.com

Listed below are all English language Delphi books and periodicals of which I am aware. This list was last updated on March 1st, 1997

Authors, publishers and others are invited to suggest books and periodicals for inclusion and to provide information and comments. I will read all email and include [with credit] those comments which I use. Unless you specifically request otherwise, your email address will be included when you are given credit. Please send your comments and suggestions to Jim Clokey

The views expressed in this document are solely those of the author and do not represent the views of his current or any past employer.

If you want the short version, here are my recommendations. If you want to know more about me, click here.

Books

Beginning Delphi 2; Peter Wright

Borland Delphi How-To; Gary Frerking, Nathan Wallace, Wayne Niddery

Borland's Official No Nonsense Guide To Delphi 2; Michelle M. Manning

Building Internet Applications with Delphi 2; Davis Chapman

Delphi 32 Bit Programming Secrets; Tom Swan with Jeff Cogswell

Delphi - A Developers' Guide; Bill Todd & Vince Kellen, with Ray Novak and Brad Saenz

Delphi By Example; Blake Watson

Delphi Client Server Developer's Guide; Joseph D. Booth

Delphi Database Development; Ted Blue, John Kaster, Greg Leif, Loren Scott

Delphi Developer's Guide; Xavier Pacheco, Steve Teixeira

Delphi In Depth; Cary Jensen, Loy Anderson, Joseph Fung, Ann Lynworth, Mark Ostroff, Martin Rudy, Robert Vivrette

Delphi Nuts and Bolts; Gary Cornell

Delphi Power Toolkit for Windows; Harold Davis

Delphi Programming Explorer; Neil Rubenking

Delphi Programming for Dummies; Neil Rubenking

Delphi Programming Problem Solver; Jeff Duntemann, Jim Mischel, Don Taylor

Delphi Super Bible; Paul B. Thurrott, Gary Brent, Richard Bagadazian, Steve Tendon

Delphi Unleashed; Charles Calvert

Delphi 2 Developer's Guide; Xavier Pacheco, Steve Teixeira

Delphi 2 Developer's Solutions; Nathan Wallace, Steve Tendon

Delphi 2 Unleashed; Charles Calvert

Delphi 3 Unleashed; Charles Calvert

Developing Custom Delphi Components; Ray Konopka

Developing Custom Delphi 3 Components; Ray Konopka

Developing With Delphi; Edward C. Weber, J. Neal Ford, Christopher R. Weber

Developing Windows Applications Using Delphi; Paul Penrod

Foundations of Delphi Development for Windows 95; Tom Swan

Instant Delphi Programming; Dave Jewell

Mastering Delphi; Mark Cantu

Mastering Delphi 2; Mark Cantu

Peter Norton's Guide to Delphi 2; Peter Norton, John Paul Mueller

Programming Delphi Custom Components; Fred Bulback

(The) Revolutionary Guide to Delphi 2; Brian Long, Bob Swart, Ewan McNab, Dave Jewell, Arjan Jansen, etc

Secrets of Delphi 2; Ray Lischner

Teach Yourself Database Programming with Delphi in 21 Days; Nathan and Ori Gurewich

Teach Yourself ... Delphi; Devra Hall

Teach Yourself Delphi in 21 Days; Andrew J. Wozniewicz, Namir Shammas & Tom Campbell

(Special Edition) Using Delphi; Jon Matcho, David R. Faulkner et al.

Periodicals

Delphi Aquarium

Delphi Developer

Delphi Informant

The Delphi Magazine

Unofficial Newsletter for Delphi Users

Return to Front Page

Beginning Delphi 2

Author(s):	Peter Wright
Publisher:	WROX Press
Copyright Date:	1995
ISBN:	1-874416-74-5
Extras:	Disk
Price (US$):	$36.95
Level:	Beginner
Comments:	I like the WROX Press books because they tend to be information heavy and they also have a layout which appeals to me. I have several Delphi protégés and this is the first book I have them study.
Acquisition Priority:	1 [If you are a Beginner], 3 {All Others]

Borland Delphi How-To

Author(s): Gary Frerking, Nathan Wallace, Wayne Niddery

Publisher: Waite Group Press

Copyright Date:

ISBN: 1-57169-019-0

Extras: CD-ROM

Price (US$): $39.95

Level:

Comments:

Acquisition Priority:

Borland's Official No Nonsense Guide to Delphi 2

Author(s):	Michelle M. Manning
Publisher:	SAMS Publishing
Copyright Date:	1996
ISBN:	0-672-30871-1
Extras:	None
Price (US$):	$25.00
Level:	Beginner
Comments:	If you have never used Delphi and are a novice with the IDE and its drag and drop User Interface development method, this book is for you. Written by an ex-Borland QA Engineer, it covers the IDE in enough depth to get you started and is short enough to actually read cover-to-cover.
Acquisition Priority:	1 [If you are a Beginner], 4 [All Others]

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Building Internet Applications with Delphi 2

Author(s):	Davis Chapman
Publisher:	Que
Copyright Date:	1996
ISBN:	0-7897-0732-2
Extras:	CD-ROM
Price (US$):	$49.99
Level:	Intermediate to Advanced
Comments:	
Acquisition Priority:	

Delphi 32 Bit Programming Secrets

Author(s):	Tom Swan with Jeff Cogswell
Publisher:	IDG Books
Copyright Date:	1996
ISBN:	1-56884-690-8
Extras:	Floppy Disk
Price (US$):	$44.99
Level:	Advanced
Comments:	I have just started reading this book [scanned it all and read first chapter]. My first impressions are that it is a very advanced book ... it starts off with a discussion of Object Pascal which rapidly gets very deep. Although! highly technical, the writing is lucid and well-informed. I suspect this will become a classic along the lines of the Calvert and Pacheco/Teixeira books. Not for the faint-hearted but definitely worth your time and effort.
Acquisition Priority:	1

Delphi - A Developer's Guide

Author(s): Bill Todd, Vince Kellen with Ray Novak, Brad Saenz

Publisher: M & T Books

Copyright Date:

ISBN: 1-55851-455-4

Extras: CD-ROM

Price (US$): $44.95

Level:

Comments:

Acquisition Priority:

Delphi By Example

Author(s): Blake Watson

Publisher: QUE

Copyright Date:

ISBN: 1-56529-757-1

Extras: none

Price (US$): $29.99

Level:

Comments:

Acquisition Priority:

Delphi Client Server Developer's Guide

Author(s): Joseph D. Booth

Publisher: M & T Books

Copyright Date: 1997

ISBN: 1-55851-492-9

Extras: CD-ROM

Price (US$): $44.95

Level:

Comments:

Acquisition Priority:

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Delphi Database Development

Author(s):	Ted Blue, John Kaster, Greg Leif, Loren Scott
Publisher:	M & T Books
Copyright Date:	1996
ISBN:	1-55851-469-4
Extras:	CD-ROM
Price (US$):	$44.95
Level:	Advanced
Comments:	This is really a reference book for the Delphi Database components, the BDE and Interbase Server. It provides documentation that is not readily available elsewhere, such as, the full BDE API in Pascal (rather than C/C++) with Error codes, the full Interbase SQL reference and a host of other reference information. This is not a book to read cover to cover, unless you also read a dictionary that way. The book also covers alternatives to the BDE.
Acknowledgment:	My thanks to Alan Gauld for this review. Larry Bradshaw says "If you are finding the Delphi BDE call documentation a bit sparse or have had some difficulty translating the C++ version into Pascal, we recommend this book. Did you know that there was a TSession.CloseDatabase method ? Or that the Delphi 1.0 documentation was incorrect for the method and that rather than take a string it takes a parameter of type TDatabase? If you had this book you could find that on page 139, or through the excellent index ! and cross reference. I cannot praise this book highly enough. It does not simply reiterated the Borland docs, but rather adds significant value with code samples for each method or procedure documented. It is technical, but if you do Delphi 1 or 2 database applications or BDE work, this is a must have reference. My hat's off to the authors.
Acquisition Priority:	1 [Developers working on database applications] / 3 {All Others]

Delphi Developer's Guide

Author(s): Xavier Pacheco, Stave Teixeira

Publisher: SAMS

Copyright Date:

ISBN: 0-672-30704-9

Extras: CD-ROM

Price (US$): $49.99

Level:

Comments:

Acquisition Priority:

Delphi In Depth: Power Techniques from the Experts

Author(s):	Cary Jensen, Loy Anderson, Joseph Fung, Ann Lynworth, Mark Ostroff, Martin Rudy, Robert Vivrette
Publisher:	Osborne McGraw-Hill
Copyright Date:	1996
ISBN:	0-07-882211-4
Extras:	CD-ROM
Price (US$):	$42.95
Level:	Advanced
Comments:	One of the best Delphi books. Some reviews have indicated problems with the mix of writing styles that comes with any book whose chapters are written by different authors. The strength and depth of the material makes this a very minor concern. Should be read cover to cover by the advanced developer.
Acquisition Priority:	2

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Delphi Nuts and Bolts: for Experienced Programmers

Author(s): Gary Cornell

Publisher: Osborne McGraw Hill

Copyright Date:

ISBN: 0-07-882203-3

Extras: None

Price (US$): $24.95

Level:

Comments:

Acquisition Priority:

Delphi Power Toolkit for Windows

Author(s): Harold Davis

Publisher: Ventana

Copyright Date:

ISBN: 1-56604-292-5

Extras: CD-ROM

Price (US$): $49.95

Level:

Comments:

Acquisition Priority:

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Delphi Programming Explorer

Author(s): Jeff Duntemann, Jim Mischel, Don Taylor

Publisher: The Crinoline Group

Copyright Date:

ISBN: 1-883577-25-X

Extras: Floppy Disk

Price (US$): $39.99

Level:

Comments:

Acquisition Priority:

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Delphi Programming for Dummies

Author(s):	Neil Rubenking
Publisher:	IDG Books
Copyright Date:	
ISBN:	1-56884-200-7
Extras:	None
Price (US$):	$19.99
Level:	Beginning
Comments:	Larry Bradshaw says "If you don't have any other Delphi book, we recommend Rubenking's book. This book is easy to read, targeted to the not-so-technical audience and yet covers some programming topics which are not to be found in either the Borland documentation or most other references. It is cheap but an excellent starter book for Delphi. Keep it handy though; we think you will continue to refer to it (as we do).
Acquisition Priority:	

Delphi Programming Problem Solver

Author(s): Neil Rubenking

Publisher:

Copyright Date:

ISBN: 1-56884-795-5

Extras:

Price (US$): $

Level:

Comments: Ahto Tanner [ahto@estpak.ee] commended this book to me with the comment that it is a how-to book with code samples that solve real-world problems.

Acquisition Priority: 3

Delphi Super Bible

Author(s): Paul B. Thurrott, Gary Brent, Richard Bagadazian, Steve Tendon

Publisher:

Copyright Date:

ISBN: 1-57169-027-1

Extras:

Price (US$): $54.99

Level:

Comments:

Acquisition Priority:

Delphi Unleashed

Author(s): Charles Calvert

Publisher: SAMS Publishing

Copyright Date:

ISBN: 0-672-30499-6

Extras: CD-ROM

Price (US$): $45.00

Level:

Comments:

Acquisition Priority:

Delphi 2 Developer's Guide

Author(s):	Xavier Pacheco and Steve Teixeria
Publisher:	SAMS Publishing
Copyright Date:	1996
ISBN:	0-672-30914-9
Extras:	CD-ROM
Price (US$):	$59.99
Level:	Intermediate to Advanced
Comments:	An excellent book ... One of the few "must haves". It is marred by several errors ... the authors would like to know if you discover any errors. This book clearly belongs on your bookshelf not just as a reference but as something to read cover to cover.
Acquisition Priority:	1

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Delphi 2 Developer's Solutions: Cutting Edge How-Tos for the Professional Developer

Author(s):	Nathan Wallace & Steve Tendon
Publisher:	Waite Group Press
Copyright Date:	1996
ISBN:	1-57169-071-9
Extras:	CD-ROM
Price (US$):	$59.99
Level:	Intermediate to Advanced
Comments:	I always purchase How-To books and am always disappointed in the purchase ... UNTIL I have a question and find the answer neatly worked out. This is one of the better how-to books. The code examples are well done. I think the! basic issue in determining if you should purchase this book is whether there is one solution in the book that you can use today. In fact, I now tend to go to my local bookstore when I have a problem and look through the chapters of a how-to book, if the! book solves the problem or at least points me in the right direction, I buy it.
Acquisition Priority:	3 [The highest rating I would give to a how-to book]

Delphi 2 Unleashed

Author(s):	Charles Calvert
Publisher:	SAMS Publishing
Copyright Date:	1996
ISBN:	0-672-30858-4
Extras:	CD-ROM which includes much of the first edition accessible with Adobe Acrobat Reader
Price (US$):	$59.99
Level:	Advanced
Comments:	DELPHI UNLEASHED, by the same author, was one of the first Delphi books published ... not surprising since the author was a member of the Borland Delphi development team. This book, surpasses the original in its depth. It delves into the innards of Delphi in a way that is unmatched by any other author.
Acquisition Priority:	1

Delphi 3 Unleashed

Author(s):	Charles Calvert
Publisher:	SAMS Publishing
Copyright Date:	
ISBN:	0-672-31015-5-4
Extras:	CD-ROM which includes much of the first edition accessible with Adobe Acrobat Reader
Price (US$):	$59.99
Level:	
Comments:	
Acquisition Priority:	

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Developing Custom Delphi Components

Author(s):	Ray Konopka
Publisher:	Coriolis Group Books
Copyright Date:	1996
ISBN:	1-883577-47-0
Extras:	CD-ROM
Price (US$):	$39.99
Level:	Advanced
Comments:	Excellent, well written, well organized, easy to understand and usable examples, good learning tool. Since I assume that virtually every advanced developer will be building components [in fact, this is one of the tests of being! an advanced developer] this is an absolutely necessary book.
Acquisition Priority:	1

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Developing Custom Delphi 3 Components

Author(s): Ray Konopka

Publisher:

Copyright Date:

ISBN: 1-57610-112-6

Extras:

Price (US$): $49.99

Level: Advanced

Comments: Excellent, well written, well organized, easy to understand and usable examples, good learning tool. Since I assume that virtually every advanced developer will be building components [in fact, this is one of the tests of being! an advanced developer] this is an absolutely necessary book.

Acquisition Priority: 1

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Developing with Delphi: Object Oriented techniques

Author(s):	Edward C. Weber, J. Neal Ford, Christopher R. Weber
Publisher:	Prentice Hall PTR
Copyright Date:	1996
ISBN:	0-13-378118-6
Extras:	Floppy Disk
Price (US$):	$29.95
Level:	Beginning to Intermediate
Comments:	This is very much a book like Mastering Delphi and similar works ... with a big BUT. The approach here is object-oriented and it discusses the Delphi IDE and Object Pascal from that viewpoint. It is not as complete in its cove! rage of Delphi features as the longer books, but it is a good introduction to the object-oriented nature of Delphi.
Acquisition Priority:	2

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Developing Windows Applications Using Delphi

Author(s): Paul Penrod

Publisher: John Wiley & Sons

Copyright Date:

ISBN: 0-471-11017-5

Extras: None

Price (US$): $29.95

Level:

Comments:

Acquisition Priority:

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Foundations of Delphi Development for Windows 95

Author(s): Tom Swan

Publisher: IDG Books

Copyright Date:

ISBN: 1-56884-347-X

Extras: CD-ROM

Price (US$): $39.99

Level:

Comments:

Acquisition Priority:

Instant Delphi Programming

Author(s): Dave Jewell

Publisher: WROX Press

Copyright Date:

ISBN: 1-874416-57-5

Extras: Floppy Disk

Price (US$): $24.95

Level:

Comments:

Acquisition Priority:

Mastering Delphi

Author(s):	Marco Cantu
Publisher:	Sybex
Copyright Date:	1995
ISBN:	0-7821-1739-2
Extras:	CD-ROM
Price (US$):	$49.99
Level:	All
Comments:	This is a 1450 page tome that tries to cover all of Delphi. I bought it when it first came out, read 3 chapters, skimmed 6 more and have not touched it since. I do not think a library needs more than one of this type of book and then only when starting with Delphi. I would select either this one [or a similar book] based on personal preference for layout and writing style.
Acquisition Priority:	2

Mastering Delphi 2

Author(s): Marco Cantu

Publisher: Sybex

Copyright Date:

ISBN:

Extras:

Price (US$): $

Level:

Comments: Alan Gauld says "I got this as an extra to the Delphi Developer's Guide book. It has paid for itself in several ways ... e.g.: the Media Player "! Play" method does not work properly and the book gives a work around which involves sending a MouseDown / MouseUp combination to the right screen coordinates using PostMessage. Its a good beginner all purpose guide and alternative to the manuals.

Acquisition Priority:

Peter Norton's Guide to Delphi 2

Author(s):	Peter Norton and John Paul Mueller
Publisher:	SAMS Publishing
Copyright Date:	1996
ISBN:	1-672-30898-3
Extras:	CD-ROM
Price (US$):	$49.99
Level:	Beginner to Intermediate
Comments:	This is a 750 page tome that tries to cover much of Delphi. For my development work, some of the topics can be useful. I do not like the code listings in blue type since they are hard to copy for ease of reference and hard to ! read when working at the computer. It does have a very good tear out page in front which provides an easy reference to Delphi Keyboard Shortcuts, Windows 95 Keyboard Shortcuts, Form Design Keys and Mouse Movement Keys in the Editor as well as some Handy! Editor Tidbits. Larry Bradshaw says "An excellent reference for all manner of technical menu items such as Help files, Install sets, OLE Automation, Creation Order, Delphi on a LAN. I like the overview of the Windows 95 Server Architecture on page 508. We use this book, oddly, when no other reference will suffice ... and it has become one of the look-there-first references in our office.
Acquisition Priority:	3

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Programming Delphi Custom Components

Author(s): Fred Bulback

Publisher:

Copyright Date:

ISBN:

Extras:

Price (US$): $

Level:

Comments:

Acquisition Priority:

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

(The) Revolutionary Guide to Delphi 2

Author(s):	Brian Long, Bob Swart, Ewan McNab, Dave Jewell, Arjan Jansen, etc.
Publisher:	WORX Press
Copyright Date:	
ISBN:	1-874416-67-2
Extras:	CD-ROM
Price (US$):	$49.95
Level:	
Comments:	
Acquisition Priority:	

Secrets of Delphi 2

Author(s):	Ray Lischner
Publisher:	Waite Group
Copyright Date:	1996
ISBN:	1-57169-026-3
Extras:	CD-ROM
Price (US$):	$49.99
Level:	Advanced
Comments:	This book [along with several others] is one of the most valuable books for the advanced Delphi developer. It is packed with information you can find nowhere else. It is the only place you can find complete descriptions for component messages and Delphi streams. Acknowledgment: My thanks to John M. Miano for the review.
Acquisition Priority:	1 [Advanced Developers] / 3 [Others]

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Teach Yourself Database Programming with Delphi in 21 Days

Author(s): Nathan & Ori Gurewich

Publisher: SAMS Publishing

Copyright Date:

ISBN: 0-672-30851-7

Extras: CD-ROM

Price (US$): $39.99

Level:

Comments:

Acquisition Priority:

Teach Yourself ... Delphi

Author(s): Devera Hall

Publisher: MIS Press

Copyright Date:

ISBN: 1-55828-390-0

Extras: Floppy Disk

Price (US$): $27.95

Level:

Comments:

Acquisition Priority:

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Teach Yourself Delphi in 21 Days

Author(s):	Andrew J. Wozniewicz, Namir Shammas, Tom Campbell
Publisher:	SAMS Publishing
Copyright Date:	
ISBN:	0-672-30997-1
Extras:	None
Price (US$):	$29.99
Level:	Beginning
Comments:	Based on comments from Richard Tefler [richard.telfer@gecm.com] ... I have been using this to learn Delphi 1 and have so far done the first 7 days plus day 21 (DLLs). It comes from the SAMS Borland Press series so presumably Borland endorses it. The book does what it claims to do ... teaches you Delphi in a finite time. Day manes that ... most of a work day not a couple of hours. Some of the descriptions could be shortened by referring back to earlier ones but the book generally gives a longer and more detailed explanation to aid clarity.
Acquisition Priority:	1 [If you are a Beginner]

(Special Edition) Using Delphi

Author(s):	Jon Matcho, David R. Faulkner et al.
Publisher:	QUE
Copyright Date:	1995
ISBN:	1-56529-823-3
Extras:	None
Price (US$):	$29.99
Level:	
Comments:	
Acquisition Priority:	

Delphi Informant

Frequency:	Monthly
Extras:	Web Site with all code available for download
Price (US$):	$49.95 per year
Address:	Informant Communications Group Inc., 10519 E. Stockton Blvd., Suite 142, Elk Grove, Ca 95624-9704
URL:	http://www.informant.com
Comments:	Excellent, slightly less technical than Delphi Magazine and a bit more informal in its writing style.
Acquisition Priority:	1

(The) Delphi Magazine

Frequency:	Monthly
Extras:	Disk with all code from each issue included in magazine
Price (US$):	$140.00 per year
Address:	The Delphi magazine, iTEC, 9a London Road, Bromley, Kent BR1 1BY, England
U.S. Address:	The Delphi magazine [USA], RR1, Box 6020, Waterbury Center, VT 05677
Comments:	Excellent and technical, probably the best in the industry from a technical viewpoint. Tight, crisp writing style.
Acquisition Priority:	1

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Unofficial Newsletter for Delphi Users

Frequency:	Irregular but about one issue every 6 weeks. Electronically distributed.
Extras:	Available in Help format or HTML format.
Price (US$):	$FREE ... Yes Free
URL:	http://www.informant.com/undu/index.htm (Use this to download)
Comments:	Published, edited and mostly written by Robert Vivrette. Not as many articles per issue as the commercial magazines. Articles are short and address specific issues in a thorough manner.
Acquisition Priority:	1

RECOMMENDED DELPHI BOOKS & PERIODICALS

The Delphi newsgroups continually have postings asking for recommendations on Delphi books and periodicals. This set of pages is an effort to provide a comprehensive and annotated list of all books and periodicals relevant to the Delphi developer. The list will be updated until all current books and periodicals are included and then as necessary when new books and periodicals are published.

For the ADVANCED Delphi Developer:

Books

Code Complete

Delphi 32 Bit Programming Secrets

Delphi 2 Developer's Guide

Delphi In Depth

Delphi Unleashed 2

Developing Custom Delphi Components

Secrets Of Delphi 2

Periodicals

Unofficial Newsletter for Delphi Users

Delphi Magazine, The

Delphi Informant

For the BEGINNING to INTERMEDIATE Delphi Developer:

Books

Guidelines for Enterprise-Wide GUI Design

Beginning Delphi 2

Borland's Official No Nonsense Guide to Delphi 2

Periodicals

Unofficial Newsletter for Delphi Users

Delphi Informant

Return to Front Page

About Jim Clokey

As a Senior Software Engineer, I have been working with Delphi since version 1.0 [beta 3]. I work as an on-site consultant for organizations involved with designing and building major client-server and desktop applications.

In addition to writing Delphi code, my expertise is in application architecture, GUI design, standards development, test design development and project management.

Contact Information


```
Permanent Email: master@pipeline.com
Current Assignment: jclokey@mtgbcs.mt.lucent.com
Voice - Current Assignment: 908-957-2607
Voice Mail: 610-670-7787
Pager: 800-675-0271 [enter area code and number and the # key]
Fax - Current Assignment: 908-957-5604
```


Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

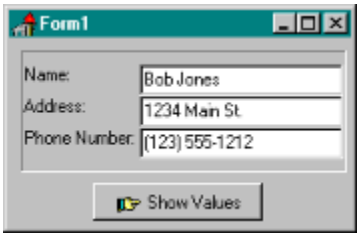
How To Be Outstanding in Your Field with TFieldPanel

by Emmanuel Fayet - 100333.2250@compuserve.com

If you need a quick way to display or collect an address or other data, you can now drop a TFieldPanel on a form and initialize the Fields property with the names of the fields you want. For example:

```
Name
Address
Phone Number
```

Now run the project and you will get a form with those field names, and an edit box after each. It might look something like this after typing in some sample values:



When you need to retrieve the data that has been entered, you just look back at the Fields property. The format of each field is <field name>=<fieldvalue>. The button click method in this sample application shows how it is done:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
  i : integer;
  s : string;
begin
  s:= '';
  With FieldPanel1.Fields do
    for i:= 0 to Count-1 do
      s:= s + Strings[i] + #13;
  ShowMessage(s);
end;
```

I hope you find TFieldPanel useful in your applications. If you improve it, please let us know. I see two natural extensions of the component: a DB version that will read and write fields from a database blob field, and an other version that will use formatted edit controls (date, money, ..)

Source to TFieldPanel

Return to The Component Cookbook

Return to Front Page

Source for TFldPanel

```
unit Fldpanel;

interface

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

type
  TFieldPanel = class(TPanel)
  private
    FFields: TStrings;
    FFieldsName: TStrings;
    FFieldsValue: TStrings;
    Fedits: TStrings;
    Flabels: TStrings;
    FScrollBar: TScrollBar;
    FTopIndex: integer;
    procedure Display;
    procedure SetTopIndex(const Value: integer);
    procedure ScrollBarScroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos:
Integer);
  protected
    procedure CreateWnd; override;
    procedure SetFields( value: TStrings );
    procedure WMSize(var Message: TWMSize); message WM_SIZE;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Fields: TStrings read FFields write SetFields;
  end;

  TFieldPanelEdit = class(TEdit)
  private
     procedure CNKeyDown(var Message: TWMKeyDown); message CN_KEYDOWN;
     procedure CMEnter(var Message: TCMEnter); message CM_ENTER;
     procedure CMExit(var Message: TCMExit); message CM_EXIT;
     procedure UpdateValue;
  end;

procedure Register;

implementation

procedure TFieldPanelEdit.UpdateValue;
begin
  with TFieldPanel(Parent) do
  begin
     FFieldsValue.Strings[FTopIndex+self.Tag]:= self.Text;
     FFields.Strings[FTopIndex+self.Tag]:= FFieldsName.Strings[FTopIndex+self.Tag] +
'=' + self.Text;
  end;
end;

procedure TFieldPanelEdit.CMExit(var Message: TCMExit);
begin
  UpdateValue;
end;
```

```
procedure TFieldPanelEdit.CMEnter(var Message: TCMEnter);
begin
  with TFieldPanel(Parent) do
     FScrollBar.Position:= FTopIndex+self.Tag;

  inherited;
end;

procedure TFieldPanelEdit.CNKeyDown(var Message: TWMKeyDown);
begin
   with TFieldPanel(Parent) do
   begin
      if (Message.CharCode = VK_TAB) and (GetKeyState(VK_SHIFT) >=0) then
      begin
         if (self.TabOrder=FLabels.Count-1) and
            (FLabels.Count+FTopIndex<FFields.Count) then
         begin
             UpdateValue;
             SetTopIndex(FTopIndex+1);
             FScrollBar.Position:= FTopIndex+self.Tag;
             Message.Result:=1;
             Exit;
         end;
      end;

      if (Message.CharCode = VK_TAB) and (GetKeyState(VK_SHIFT) <0) then
      begin
         if (self.TabOrder=0) and (FTopIndex<>0) then
         begin
            UpdateValue;
            SetTopIndex(FTopIndex-1);
            FScrollBar.Position:= FTopIndex+self.Tag;
            Message.Result:= 1;
            Exit;
         end;
      end;
   end;

   inherited;
end;

constructor TFieldPanel.Create(AOwner: TComponent);
begin
   inherited Create(aOwner);
   ControlStyle:= ControlStyle - [csAcceptsControls, csSetCaption];

   bevelOuter:= bvLowered;
   caption:= '';

   FFields:= TStringList.Create;
   FFieldsName:= TstringList.Create;
   FFieldsValue:= TStringList.Create;

   Flabels:= TStringList.Create;
   Fedits:= TStringList.Create;
   FScrollBar:= TScrollBar.Create(self);
   FTopIndex:= 0;
end;

procedure TFieldPanel.CreateWnd;
begin
   inherited CreateWnd;
```

```
   Display;
end;

destructor TFieldPanel.Destroy;
begin
   FFields.Free;
   FFieldsName.Free;
   FFieldsValue.Free;

   Flabels.Free;
   Fedits.Free;
   FScrollBar.Destroy;

   inherited Destroy;
end;

procedure TFieldPanel.SetFields( value: TStrings );
begin
   FFields.Assign(Value);
   Display;
end;

procedure TFieldPanel.WMSize(var Message: TWMSize);
begin
   if (csDesigning in ComponentState) then
      Display;
end;

procedure TFieldPanel.SetTopIndex(const Value: integer);
var i: integer;

begin
   if (Value>=0) and (FTopIndex<>value) and (FLabels.Count+Value-1<FFields.Count) then
   begin
      FTopIndex:= Value;
      for i:=0 to FLabels.Count-1 do
        TLabel(FLabels.Objects[i]).Caption:= FFieldsName.Strings[i+FTopIndex]+':';

      for i:=0 to FEdits.Count-1 do
        TFieldPanelEdit(FEdits.Objects[i]).Text:= FFieldsValue.Strings[i+FTopIndex];
   end;
end;

procedure TFieldPanel.ScrollBarScroll(Sender: TObject; ScrollCode: TScrollCode; var
ScrollPos: Integer);
begin
   SetTopIndex(ScrollPos);
end;

procedure TFieldPanel.Display;
var Label1: TLabel;
    Edit1: TFieldPanelEdit;
    i, p, widthLabel, maxLabel, leftLabel, topLabel, heightLabel, heightInterLabel,
    widthScrollBar: integer;
    bShowScrollBar: boolean;

begin
   while FLabels.Count<>0 do
   begin
     (FLabels.Objects[0] as TLabel).Destroy;
     FLabels.Delete(0);
   end;
   while FEdits.Count<>0 do
```

```
   begin
     (FEdits.Objects[0] as TFieldPanelEdit).Destroy;
     FEdits.Delete(0);
   end;

   FFieldsName.Clear;
   FFieldsValue.Clear;
   for i:=0 to FFields.Count-1 do
   begin
      p:= pos('=',FFields.Strings[i]);
      if p<>0 then
      begin
         FFieldsName.Add( copy( FFields.strings[i], 1, p-1) );
         FFieldsValue.Add( copy( FFields.strings[i], p+1, length(FFields.Strings[i])-
p) );
      end
      else
      begin
         FFieldsName.Add( FFields.strings[i] );
         FFieldsValue.Add( '' );
         FFields.strings[i]:=FFields.strings[i]+'=';
      end;
   end;

   heightInterLabel:= 8;
   heightLabel:= Canvas.TextHeight('W');
   widthLabel:= 0;
   for i:=0 to FfieldsName.Count-1 do
      if Canvas.TextWidth(FFieldsName.Strings[i]+':')>widthLabel then
         WidthLabel:=Canvas.TextWidth(FfieldsName.Strings[i]+': ');

   leftLabel:= 2;
   topLabel:= heightInterLabel+1;
   widthScrollBar:= GetSystemMetrics(SM_CXVSCROLL);
   maxLabel:= (Height - topLabel) div (heightLabel + topLabel);
   if maxLabel>FfieldsName.count then maxLabel:= FFieldsName.count;

   if (maxLabel<FFieldsName.count) and (FFieldsName.count>0) then bShowScrollBar:=
true else bShowScrollBar:= false;
   if bShowScrollBar then
   begin
     FScrollBar.Parent:= self;
     FScrollBar.Kind:= sbVertical;
     FScrollBar.Width:= widthScrollBar;
     FScrollBar.Height:= Height-1;
     FScrollBar.left:= width-FScrollBar.width-1;
     FScrollBar.top:= 1;
     FScrollBar.Min:= 0;
     FScrollBar.Max:= FfieldsName.Count-1;
     FScrollBar.OnScroll:= ScrollBarScroll;
     FScrollBar.Visible:= true;
   end
   else
   begin
     FScrollBar.Parent:= self;
     FScrollBar.left:= 0;
     FScrollBar.top:= 0;
     FScrollBar.Width:= 0;
     FScrollBar.Height:= 0;
     FScrollBar.Visible:= false;
   end;

   for i:=0 to maxLabel-1 do
```

```
   begin
     label1:= TLabel.Create(self);
     label1.Parent:= self;
     label1.Left:= leftLabel;
     label1.Top:= topLabel;
     label1.Caption:= FfieldsName.Strings[i]+':';
     FLabels.AddObject(FfieldsName.Strings[i], Label1);

     Edit1:= TFieldPanelEdit.Create(self);
     Edit1.Parent:= self;
     Edit1.Left:=leftLabel+widthLabel;
     Edit1.Top:=topLabel;
     if not bShowScrollBar then Edit1.Width:= Width-((2*LeftLabel)+widthLabel)
                           else Edit1.Width:= Width-((2*LeftLabel)
+widthLabel+widthScrollBar);
     Edit1.TabStop:= true;
     Edit1.TabOrder:= i;
     Edit1.Tag:= i;
     Edit1.Text:= FFieldsValue.Strings[i];
     FEdits.AddObject(FfieldsName.Strings[i], Edit1);

     topLabel:= topLabel + heightLabel + heightInterLabel;
   end;
end;

procedure Register;
begin
   RegisterComponents('Examples', [TFieldPanel]);
end;

end.
```

Keeping Form Aspect Ratio

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

In UNDU Issue #17 I showed how I limited a form to a square shape by intercepting the Win 95 WM_Sizing message. I was distracted enough by the usefulness of this to write a component which, when placed on a form, limits the form's shape to a given aspect ratio.

To create a component which influences the form on which it is placed you have to intercept the form's messages in the component. Here is the shell code for this.

You are simply inserting a new WndProc for the original WndProc. The only twist is the need to use MakeObjectInstance to convert a method pointer (which can't be passed to the API) into a pointer (which can be).

The two example components which use this shell are TAspect which can control a forms shape and TMinMax which can control the minimum and maximum size a form can be sized to under a variety of conditions. You need to find WM_Sizing and WM_GetMinMaxInfo in the API help file to understand how these components work and what the properties are for.

Here's two palette bitmaps for you to clip and use in your DCR file:

`TAspect:` 

`TMinMax:` 

Have Fun!

Source for Aspect.pas

Source for MinMax.pas

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

A Bit More About Previous Instances

by Robert Vivrette - RobertV@compuserve.com

In Issue #16 of UNDU, I presented an article about limiting multiple instances of a program in Delphi

It turns out there is one additional issue that needs addressing… Using the source in issue #16, if you run an application, then minimize it, then try to launch it again, it restores the original copy (as it should). However, you can not then re-minimize the application. It just ignores you.

The problem stems from the fact that there is a hidden application window floating around. The applications MainForm is a child of this window. When you restore just the main form, the application still thinks it's minimized. Then when you click on the minimize button of the main form, the application says "forget it... I am already minimized". The solution is that the application needs to be restored instead of the main form being restored. When you are searching for a second instance of the application, you really should be looking for the application window and not the main form window. However, it can be done the latter way also. Here is a modified example of the DPR source that shows how this can be done to make the technique work correctly.

```
program Project0;

uses
  Windows,
  Forms,
  Unit0 in 'Unit0.pas' {Form1};

var
  Handle1 : LongInt;
  Handle2 : LongInt;

{$R *.RES}

begin
  Application.Initialize;
  Handle1 := FindWindow('TForm1',nil);
  if handle1 = 0 then
    begin
      Application.CreateForm(TForm1, Form1);
      Application.Run;
    end
  else
    begin
      {Obtain handle to owner of Main Form. This is the application window}
      Handle2 := GetWindow(Handle1,GW_OWNER);
      {Hide application window to avoid zoom effect}
      ShowWindow(Handle2,SW_HIDE);
      {Restore application window}
      ShowWindow(Handle2,SW_RESTORE);
      {Set Main Form as foreground window}
      SetForegroundWindow(Handle1);
    end;
end.
```

Note that we first find the Main form window, then use GetWindow to find its owner. Then we send the restore to that window, and then set the main form as the foreground window. When the restore goes to the application window, it restores the main form. I am sending a hide to the application first to avoid a zoom effect from the Win95 task bar. If you comment out the line with SW_Hide, you will see what I mean.

Also keep in mind that this technique works correctly only outside of the Delphi IDE. When you try to run an application from the IDE, the design-time copy of the main form is still around and Windows see's that

as another instance of the program according to our test.

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Borland's Solution to Form Resolution

Last issue, I posted a reader question concerning form resolution. The question centered around the fact that you would want your application to look essentially the same regardless of what the screen resolution is and/or the system font size (small fonts vs large fonts).

Although I got quite a few responses on this issue, the one that I think explains the issues best is a tech sheet that Borland itself put out. It covers the issues in some depth so it will help readers see a little more about all the factors that come into play.

Some of the solutions that were sent in went to great lengths to change the form's font size to be the same visual size regardless of the setting of large fonts and small fonts. Big mistake in my opinion. A key reason behind the use of Small fonts vs. Large fonts is that a user might be visually impaired and writing code to force a font to be a specific height despite the system font size setting could cause problems. It is better to design the interface such that differing resolutions and font size have as little impact on the program (and user) as possible.

Anyway, here is a complete reprint of Borland's tech sheet #2861… You can find all of their tech sheets on the CompuServe Delphi forum in the "From Borland" file section.

#2861 - Form display with different screen resolutions.

When designing forms, it is sometimes helpful to write the code so that the screen and all of its objects are displayed at the same size no matter what the screen resolution is. Here is some code to show how that is done:

```
implementation
const
  ScreenWidth: LongInt = 800; {I designed my form in 800x600 mode.}
  ScreenHeight: LongInt = 600;

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  scaled := true;
  if (screen.width <> ScreenWidth) then
  begin
    height := longint(height) * longint(screen.height) div ScreenHeight;
    width := longint(width) * longint(screen.width) div ScreenWidth;
    scaleBy(screen.width, ScreenWidth);
  end;
end;
```

Then, you will want to have something that checks to see that the font sizes are OK. You can iterate over each child control's font to adjust its size as necessary. This can be done as follows:

```
type
  TFooClass = class(TControl); { needed to get at protected }
                               { font property }

var
  i: integer;
begin
  for i := ControlCount - 1 downto 0 do
    TFooClass(Controls[i]).Font.Size :=
        (NewFormWidth div OldFormWidth) *
        TFooClass(Controls[i]).Font.Size;
end;
```

Note: The following are issue to bear in mind when scaling Delphi applications (forms) on different screen resolutions:

Decide early on in the form design stage whether you're going to allow the form to be scaled or not. The advantage of not scaling is that nothing changes at runtime. The disadvantage of not scaling is that nothing changes at runtime (your form may be far too small or too large to read on some systems if it is not scaled).

If you're NOT going to scale the form, set Scaled to False. Otherwise, set the Form's Scaled property to True.

Set AutoScroll to False. AutoScroll = True means 'don't change the form's frame size at runtime' which doesn't look good when the form's contents do change size.

Set the form's font to a scaleable TrueType font, like Arial. MS San Serif is an OK alternate, but remember that it is still a bitmapped font. Only Arial will give you a font within a pixel of the desired height. NOTE: If the font used in an application is not installed on the target computer, then Windows will select an alternative font within the same font family to use instead. This font may not match the same size of the original font any may cause problems.

Set the form's Position property to something other than poDesigned. poDesigned leaves the form where you left it at design time, which for me always winds up way off to the left on my 1280x1024 screen - and completely off the 640x480 screen.

Don't crowd controls on the form - leave at least 4 pixels between controls, so that a one pixel change in border locations (due to scaling) won't show up as ugly overlapping controls.

For single line labels that are alLeft or alRight aligned, set AutoSize to True. Otherwise, set AutoSize to False.

Make sure there is enough blank space in a label component to allow for font width changes - a blank space that is 25% of the length of the current string display length is a little too much, but safe. (You'll need at least 30% expansion space for string labels if you plan to translate your app into other languages) If AutoSize is False, make sure you actually set the label width appropriately. If AutoSize is True, make sure there is enough room for the label to grow on its own.

In multi-line, word-wrapped labels, leave at least one line of blank space at the bottom. You'll need this to catch the overflow when the text wraps differently when the font width changes with scaling. Don't assume that because you're using large fonts, you don't have to allow for text overflow - somebody else's large fonts may be larger than yours!

Be careful about opening a project in the IDE at different resolutions. The form's PixelsPerInch property will be modified as soon as the form is opened, and will be saved to the DFM if you save the project. It's best to test the app by running it standalone, and edit the form at only one resolution. Editing at varying resolutions and font sizes invites component drift and sizing problems.

Speaking of component drift, don't rescale a form multiple times, at design time or a runtime. Each rescaling introduces roundoff errors which accumulate very quickly since coordinates are strictly integral. As fractional amounts are truncated off control's origins and sizes with each successive rescaling, the controls will appear to creep northwest and get smaller. If you want to allow your users to rescale the form any number of times, start with a freshly loaded/created form before each scaling, so that scaling errors do not accumulate.

Don't change the PixelsPerInch property of the form, period.

In general, it is not necessary to design forms at any particular resolution, but it is crucial that you review their appearance at 640x480 with small fonts and large, and at a high-resolution with small fonts and large before releasing your app. This should be part of your regular system compatibility testing checklist.

Pay close attention to any components that are essentially single-line TMemos - things like TDBLookupCombo. The Windows multi-line edit control always shows only whole lines of text - if the control is too short for its font, a TMemo will show nothing at all (a TEdit will show clipped text). For such components, it's better to make them a few pixels too large than to be one pixel too small and show not text at all.

Keep in mind that all scaling is proportional to the difference in the font height between runtime and design time, NOT the pixel resolution or screen size. Remember also that the origins of your controls will be changed when the form is scaled - you can't very well make components bigger without also moving them over a bit.

Return to Tips & Tricks

Return to Front Page

Speeding up Processing of Large Tables

by Duncan Campbell ("Dunk") - duncan@tvl.com

Recently I have been writing an app, working with large tables (10,000+ Records) containing many fields (50+). I had a section of my program that required me to loop through a large number of records, updating them as I went. For various reasons, the TQuery component was not appropriate for this situation, and I was very dismayed by how slow the TTable seemed to be processing. I traced the speed problems down to the fact that I had a number of calculated fields that were being re-calculated far too often, and came up with a "quick and dirty" method of speeding things up, that you may find useful.
First of all, I create a global variable -gPROCESS_OK - of type Boolean which I initialize as TRUE.

In the "OnCalcFields" of my TTable, I put the following code:

```
If gPROCESS_OK then
  {code for calculated fields}
```

Then when I want to do my large process, I perform the following steps:

```
with myTable do
  try
    {this stops the user from noticing that anything is going on}
    DisableControls;
    {this turns OFF the calculated fields}
    gPROCCESS_OK := False;
    {perform process}
  finally
    {Turn calculated fields back on}
    gPROCESS_OK := TRUE;
    {and re-enable the data-aware controls}
    EnableControls;
  end;
```

As you can see, at the beginning of the try..finally block, I first disable all data-aware controls, and turn off calculated fields processing. I then perform my loop before finally turning back on the calculated fields and re-enabling data-aware controls.

I have found this method to be very useful in increasing performance when large numbers of records need to be processed.

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

SHFileOperation Revisited

by Robert Vivrette - RobertV@compuserve.com

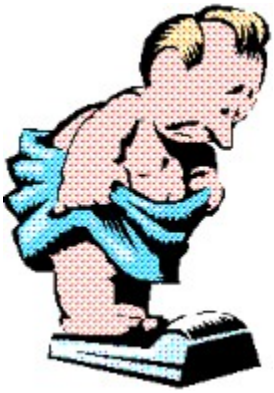
Last issue, I discussed how to use the SHFileOperation command in the Win95 API to copy/move/delete/rename files and to add system-level undo support to these actions.

However, I left out one little issue in the discussion. When you are assigning values to the pFrom and pTo portions of the record structure, they need to by terminated with two nulls (#0) rather than just a single null. These two record fields are used to specify source and destination file names, and they allow you to input more than one filename on each. To use more than a single file name, you must make them a single string, each terminated by a single null value, and then the entire string terminated with an additional null. That way, the operation system knows where each file name ends and also knows when it has reached the end of all the names. Even if you are using only a single name, it must end with two nulls or you get all sorts of odd behaviors.

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997



How To Make Your EXE's Lighter!

by Eric Fortier - cfortier@clic.net

This quick tip is a way to trim down the size of you applications, and in my case, these changes enabled me to cut down over 260K from one executable! I think everyone will benefit from this.
If you're like me, you build your own components, and add plenty of properties. What you should know is that the properties of your component are stored in a file which is linked into the executable you create, the *.DFM file. So, with careful programming you can have less properties stored in those DFM files! How you ask? Well, VERY simple! consider this property:

```
published
  property MyCount: integer read FMyCount write FMyCount;
```

This property will always be saved in the DFM file. There are two ways to have Delphi think before saving it, to see if it's really necessary. These ways are through the DEFAULT and STORED directives. First, if your MyCount property is almost always zero, you can use this definition instead:

```
published
  property MyCount: integer read FMyCount write FMyCount DEFAULT 0;
```

and add this to the create method of your component:

```
constructor xx.Create...
begin
  inherited Create(AOwner);
  FMyCount:=0;
  ...
end;
```

This way, Delphi will check to see if the property is set to zero, and if so, it will not be saved to the DFM file and it will "default" to 0 next time it loads.

You can also explicitly state if you want the property stored or not with the STORED keyword:

```
published
  property MyCount: integer read FMyCount write FMyCount STORED True;
      {or}
  property MyCount: integer read FMyCount write FMyCount STORED IsStored;
      (with IsStored being a procedure)
```

This way, you'll cut the size of your DFM file, and your executable. I cut almost all my DFM size in half, and stripped more than 260K off my executable! Remember to load and save back the .DFM file for the changes to appear!

There are also other ways to cut down the size of things. Another way, as suggested by another user, is to make property names smaller. This has to be used with care however, because you might name a property "BACK1" and not remember what it is after a while.

This same user also suggest to move the property to the PUBLIC declaration module of the component, thus removing it entirely from the DFM file. This too has the side effect from removing it from the Object Inspector, so you have to know it's there if you want to use it.

After using defaults for your properties, you might want to use the Delphi code editor to load a .DFM file to take a look. There might be "residual" properties which are cluttering the file. By removing the unwanted, orphaned properties (left over after the "Default" procedure) you will take out another slice off those DFM, in the case of some components, I was able to scrape off 10-20 K.

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Revision for TPageControl

by Grahame Marsh - Grahame.S.Marsh@corp.courtaulds.co.uk

An error crept into the destroy method of the TExPageControl in the article in issue #19 of UNDU. I didn't check to see if the FGlyphs property had been set before setting the OnChange property to nil to remove the link. It seems that I had only tested the component with it linked to a TImage which was my main reason for writing the component in the first place. Sorry, to anyone who suffered the great pile of Access Violation errors you get with this bug! There must be a moral to this story somewhere.

Anyway, here is what you need to change the destructor to solve the problem:

```
destructor TExPageControl.Destroy;
begin
  if Assigned (FGlyphs) then
    FGlyphs.OnChange := nil;
  FCanvas.Free;
  inherited Destroy;
end;
```

Return to The Component Cookbook

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Displaying Multi-Colored Text in a String Grid

by Paul Harding - 100046.2604@compuserve.com

Here is a tip you might find useful… namely how to display multi-colored text in a string grid.
As you may already know, if you wish to display text in a string grid, you can set the text of an individual cell using: StringGrid1.Cells[Col,Row] However, you can also display the text in each cell of the grid in different colors, and for this tip I'll just demonstrate a simple way to alter the color of a cell's text just by clicking on it.
Originally, I was going to demonstrate the technique by just making positive numbers black and negative numbers red. However, the technique is far more powerful than this and there is a far simpler way of showing whether a number in a cell is positive or negative: namely using the OnDrawCell event to look at the cell's value and decide its color based off whether it is positive and negative.

But there is quite a bit more in this technique. What you can do, is use the string grid's Objects property to store a color (or anything else) with each cell. Every cell on a grid has the ability to store a pointer to an object. So, using this, we can actually store a number, by typecasting the number into a pointer like this: Pointer(56745) and we can get the number back by typecasting the pointer like this: LongInt(myPtrVariable);

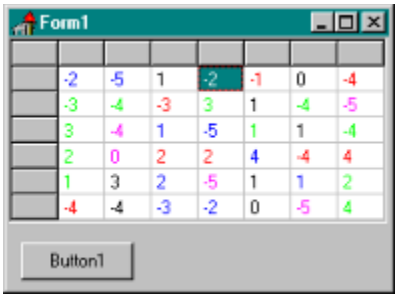
Now, since colors are just LongInts, we can do this: Pointer(clRed), and we are halfway there.

To access the pointer to the object in each cell, we just do this:

```
StringGrid1.Cells[col,row] := '7';
StringGrid1.Objects[col,row] := Pointer(clLime);
```

Now we have stored the text '7' in the cell, and the "value" of the color Lime Green in the cells' object pointer. When we come to draw the cell, we just use the pointer value to get the color we need.

The sample project demonstrates a string grid, and a button dropped onto a form. When the button is clicked, the string grid gets filled with a random selection of numbers. Initially, all of the cell's Objects properties are saved as Pointer(clBlack). But when you double-click on a cell, it randomly changes it to one of 4 other colors.



When the grid draws its own cells, it has to know what color to use. Using the OnDrawCell event, simply retrieve the color from the object pointer, and use it to draw the text, and hey presto, you have a grid of colored numbers!
We have cheated really, because we are not storing a "proper" object in the Objects property of the grid. If we HAD stored objects there, they would need to be freed up before the grid gets destroyed. Since we haven't really stored objects in the grid, we'll just finish off by setting all our pointers back to nil in the

form's OnClose event.
Obviously this technique is a lot more powerful than just displaying random numbers in different colors. Keep in mind that the Objects property for each cell can hold pretty much anything. The color we store could indicate specific states for the cell. For example, in an accounting situation you might have numbers that need authorization. You could mark them by changing their color to red and when the user selects that cell and clicks on a "Authorize" button, the cell color changes to black. Also, since the Objects property is storing pointers, you could keep pointers to some other object. Perhaps a bitmap that should be used along with the number, or maybe even a pointer to some explanatory text.

The possibilities are endless!

Source Code For This Project

Return to Tips & Tricks

Return to Front Page

Printing Raw Data

by Philip Hibbs - 101621,1264

Reading UNDU#19, I was interested to see the solution for sending raw data to the printer. I myself had this problem, but resolved it another way. I used the Windows API function SpoolFile like this:

```
iRet := SpoolFile('Generic / Text Only','LPT1',pcTitle,pcTempName);
```

where pcTempName is the name of the file to be sent to the printer. Note that this file will automatically be deleted by the spooler, so be careful!

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Converting Delphi Source files to HTML

Announcement by Pieter Polak - PP@coas.com

Do you need to publish Delphi/Pascal source code on your web site? It can be quite a pain having to do all that formatting by hand. But to simplify this job, we have created a tool which convert Pascal source files into HTML with full syntax highlighting. This is a free tool, and can be found on http://www.coas.com/pas2html or can de downloaded from the BDELPHI32 forum on Compuserve.

Currently I am working on a new version which will be available as a Delphi expert as well, so you can run the conversion from within the IDE. In addition to the CGI version, an ISAPI version will become available as well (to increase performance on Windows/NT servers running IIS). Also a version with a graphical (windows) interface will become available within a short time.

Try it out and let me know what you think!

Return to Tips & Tricks

Return to Front Page

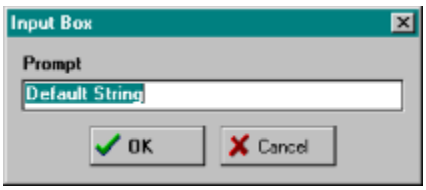
The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997

Adding a History List to an InputQuery Box

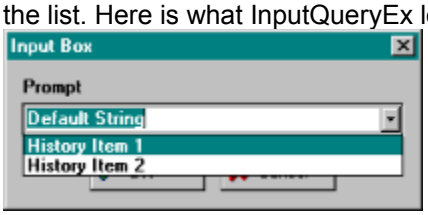
by Gene Fowler - acorioso@ccnet.com

InputQueryEx is an InputQuery with a history list so you can recycle, or simply cycle through, past inputs. This quick user-input collector that keeps its past, is a very useful thing to have on hand, whatever you call it. This code was created for Delphi 1.0, although I expect that it wouldn't be too different in Delphi 2.0.

For those of you who are unfamiliar with InputQuery, this is what it looks like:



As you can see, it is just a simple input box that allows the user to specify a string value. The enhanced InputQueryEx function, replaces the Edit box with a Combobox and allows you to specify history items for the list. Here is what InputQueryEx looks like:



If you want to see an InputQueryEx among a number of InputQuerys, grab ftp.coriolis.com/pub/Controls/dbdxpand.zip and unzip it. Two "hand-held" outrigger editors for Database Desktop, MemoEdit and PictEdit will tumble out. If that's awkward but you have Kick Ass Delphi (Coriolis) on a near-by shelf, the pictures are on pages 318-9 and the editors are on the disk.
InputQueryEx Source

The following is some sample code showing the use of InputQueryEx:

```
      procedure TForm1.Button1Click(Sender: TObject);
      var
        NewString: string;
        ClickedOK: Boolean;
        History : TStringList;
      begin
        NewString := 'Default String';
        Label1.Caption := NewString;
        History := TStringList.Create;
        History.Add('History Item 1');
        History.Add('History Item 2');
        ClickedOK := InputQueryEx('Input Box', 'Prompt', NewString, History);
        if ClickedOK then                              { NewString contains new input
string }
          Label1.Caption := 'The new string is ''' + NewString + '''';
        History.Free;
      end;
```

Obviously, this example doesnt really do anything with the history list, so you would probably want to make the History string list a global variable and add new strings to it as you go. Even better, you would probably want to save it to an INI file or the Registry if appropriate.

Return to Tips & Tricks

Return to Front Page

"Is Someone Else Running?" - Revisited!

Editors Note: *In Issue #19 of UNDU, Paul Harding presented an article on how to determine if another application was running. This article was an extension of the technique used to determine if an application already had an instance running (which was from issue #12). As you will see, the tip by Magnus Baeck below shows a much simpler way to see if another (different) application is running. However, this tip will* ***not*** *work as a way to limit multiple instances of the same program.*

by Magnus Baeck - baeck@swipnet.se

There is a much easier solution to the 'Is Someone Else Running?' article in UNDU 19. Instead of checking class names, why not just check is a certain EXE is running? The following code does the trick with both D1 and D2:

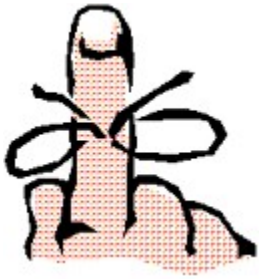
```
function IsModuleRunning(ModuleName: string): Boolean;
{$IFDEF VER80}
var
   S: array [0..127] of Char;
{$ENDIF}
begin
{$IFDEF VER80}
   StrPCopy(S, ModuleName);
   IsModuleRunning := GetModuleHandle(S) <> 0;
{$ELSE}
   IsModuleRunning := GetModuleHandle(PChar(ModuleName)) <> 0;
{$ENDIF}
end;
```

The ModuleName parameter can also indicate a DLL. This method would of course fail if there were two modules with the same name, but that would have to be very unusual.

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #20 - March 1997



Tip Of The Day

by Robby Walker - RobertAWalker@msn.com

You load up Windows 95 and the first thing you see is the welcome screen (if you haven't disabled it). The welcome screen looks like this:

Wow! What a cool idea… So, how do I use it in my own applications? The best way is to develop a tip of the day component. Of course another advantage is that you learn a lot by developing it. You become more familiar with string lists and file access, and also learn how to wrap a form into a component. All of these are key Delphi concepts.

There are two things needed for a tip of the day component: the form which displays the tips and the component which displays the form. Let's start with the form.

There few key pieces that we will mainly discuss are the tip label, the "Show Tips at Startup" checkbox, and the buttons. When the form is created, it loads a tip file into a TStringList called Tips. The tip file format is very simple. The first line of the file is a True or False to indicate whether to show the tips or not. Then after that is simply a list of all the tips, one tip per line. An example tip file might look like this:

```
True
You can get context-sensitive help by clicking on the help button.
Buttons captions that are gray indicate the button has been disabled.
Don't take candy from strangers.
Chocolate ice cream tastes really good.
```

When the first string is read, if it is 'True', then the show at startup checkbox is checked. If the string is anything else, then the show at startup checkbox is cleared. Then, the first tip is displayed in the tip label and the form opens and a tip is shown. At this point, the user has two courses of action. They may either hit the Next Tip or the OK button. Both of these events have one thing in common; they both advance to the next tip. The way this is done is by moving the first tip to the end of the tip list. For instance, if you had the list 1-2-3-4-5, and you moved 1 to the end, you would have 2-3-4-5-1. Since you wanted the next

tip, this moves it to the front position which is where the tip is read from. In the next button clicked event, the tip at the front of the list is then displayed. In the OK button clicked event, a few things are still left. The new show at startup status is written to the first line in the file. Then, the tip list is destroyed and the form is closed. If after all of this talk about the form, you want to see it, here is it's source and form source.

Source for TipBox

Source for the TipBox form

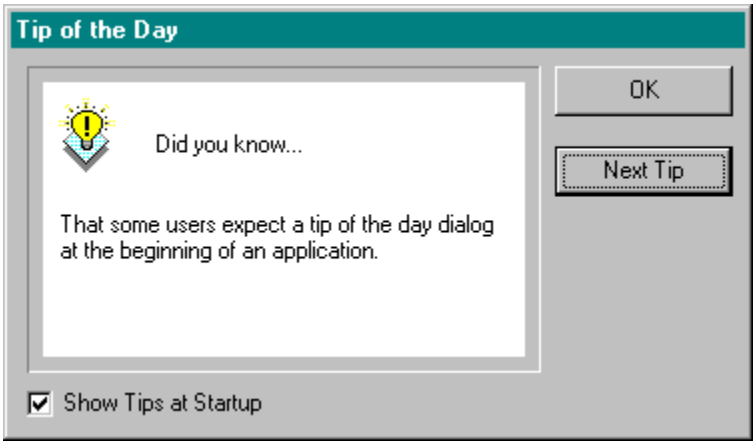
So, now we have a fully functional form. Next we need a component to access the form with. It is a very simple component. It is descended from TComponent because it is a non-visible component. It has only two properties: the name of the file which holds the tips and a read-only property which hold the value of the show on startup checkbox. It has one method, the procedure Execute. Execute takes no arguments but rather simply creates the tip form, passes the file name to the form, shows the form modally, and then destroys the form. The only other method in the unit is the register method which registers the component onto the UNDU page.

Source for TipDlg

To use this component, simply place it on the form, and somewhere call the Execute method like this:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  TipOfTheDayDlg1.Execute;
end;
```

and abracadabra! A tip of the day form pops up onto your screen…



Return to The Component Cookbook
Return to Front Page

InputQX.PAS Source

```
{ The InputQX unit, containing the InputQueryEx function for Borland's
  Delphi is Copyright (c) 1996 by Gene Fowler but may be used freely
  by the good folk shaping Delphi interfaces.

  InputQueryEx produces an Input dialog as InputQuery does, but the
  Edit is replaced by a Dropdown Combobox so that you have a history
  of previous inputs that you can reuse.

  This function is based mostly on Borland's InputQuery function in
  the DIALOGS.PAS unit of Delphi 1.0. This function was copied into
  this unit and used as the basis of the new InputQueryEx function.
  Changes that have been made from the original are noted.

  Want to see this "in action"? Assuming you're on-line (and have BDE
  installed), ftp://ftp.coriolis.com/Controls/dbdxpand.zip. This un-
  zips two "outrigger" editors to use with Database Desktop, MemoEdit
  and PictEdit. PictEdit has an InputQueryEx under the Full Image
  Editors button.

                                                    -- Gene Fowler
}

unit InputQX;

{$S-,W-,R-}
{$C PRELOAD}

interface

uses Classes, Graphics, Controls, Buttons, StdCtrls, Forms, Dialogs;

function InputQueryEx(const ACaption, APrompt: string;
                      var Value: string; var Values: TStringList): Boolean;

implementation

function InputQueryEx(const ACaption, APrompt: string;
                      var Value: string; var Values: TStringList): Boolean;
var
  W     : TForm;
 {Edit: TEdit;}                                          {OLD}
  Combo : TComboBox;                                     {NEW}
  i     : Integer;                                       {NEW}
  S     : String;                                        {NEW}
  L     : TLabel;
  OKButton: TBitBtn;
  CancelButton: TBitBtn;
begin
  Result := False;
  W := TForm.Create(Application);
  try
    with W do
    begin
      BorderStyle := bsDialog;
      Ctl3D := True;
      Width := 280;
      Height := 160;
      Caption := ACaption;
      Font.Name := 'MS Sans Serif';
      Font.Size := 8;
```

```
Font.Style := [fsBold];
Position := poScreenCenter;

L := TLabel.Create(W);
with L do
begin
  Parent := W;
  AutoSize := True;
  Left := 10;
  Top := 10;
  Caption := APrompt;
end;

{Edit := TEdit.Create(W);       }                    {OLD}
{with Edit do                   }                    {OLD}
{begin                          }                    {OLD}
{  Parent := W;                 }                    {OLD}
{  Left := 10;                  }                    {OLD}
{  Top := L.Top + L.Height + 5; }                    {OLD}
{  Width := W.ClientWidth - 20; }                    {OLD}
{  MaxLength := 255;            }                    {OLD}
{  Text := Value;               }                    {OLD}
{  SelectAll;                   }                    {OLD}
{end;                           }                    {OLD}
{L.FocusControl := Edit;        }                    {OLD}

Combo := TComboBox.Create(W);                        {NEW}
with Combo do                                        {NEW}
begin                                                {NEW}
  Parent := W;                                       {NEW}
  Left := 10;                                        {NEW}
  Top := L.Top + L.Height + 5;                       {NEW}
  Width := W.ClientWidth - 20;                       {NEW}
  MaxLength := 127;                                  {NEW}
  SelectAll;                                         {NEW}
  Text := Value;                                     {NEW}
  Combo.Items.Clear;                                 {NEW}
  if Values.Count > 0 then                           {NEW}
    begin                                            {NEW}
      For i := 0 to Values.Count - 1 do              {NEW}
        begin                                        {NEW}
          S := Values[i];                            {NEW}
          Combo.Items.Add(S);                        {NEW}
        end;                                         {NEW}
      if Combo.Items[0] <> Combo.text then           {NEW}
        begin                                        {NEW}
          Combo.Items.Insert(0, Combo.Text);         {NEW}
          for i := 0 to Combo.Items.Count - 1 do     {NEW}
            if Combo.Items[i] = Combo.Items[0] then  {NEW}
              Combo.Items.Delete(i);                 {NEW}
        end                                          {NEW}
    end                                              {NEW}
  else                                               {NEW}
    Combo.Items.Add(Combo.Text)                      {NEW}
end;                                                 {NEW}
L.FocusControl := Combo;                             {NEW}

OKButton := TBitBtn.Create(W);
with OKButton do
begin
  Parent := W;
  Kind := bkOK;
  Style := MsgDlgButtonStyle;
```

```
        if not MsgDlgGlyphs then
        begin
          Glyph := nil;
          Margin := -1;
        end
        else Margin := 2;
       {Top := Edit.Top + Edit.Height + 10;}                {OLD}
        Top := Combo.Top + Combo.Height + 10;               {NEW}
        Width := 77;
        Height := 27;
        Left := (W.ClientWidth div 2) - (((OKButton.Width * 2) + 10) div 2)
      end;

      CancelButton := TBitBtn.Create(W);
      with CancelButton do
      begin
        Parent := W;
        Kind := bkCancel;
        Style := MsgDlgButtonStyle;
        if not MsgDlgGlyphs then
        begin
          Glyph := nil;
          Margin := -1;
        end
        else Margin := 2;
        Top := OKButton.Top;
        Width := 77;
        Height := 27;
        Left := OKButton.Left + OKButton.Width + 10;
      end;
      ClientHeight := OKButton.Top + OKButton.Height + 10;
    end;
    {if W.ShowModal = mrOK then   }                         {OLD}
    {begin                        }                         {OLD}
    {  Result := True;            }                         {OLD}
    {  Value := Edit.Text;        }                         {OLD}
    {end;                         }                         {OLD}
    if W.ShowModal = mrOK then                              {NEW}
      begin                                                 {NEW}
        Result := True;                                     {NEW}
        Value  := Combo.Text;                               {NEW}
        if Combo.Items[0] <> Combo.Text then                {NEW}
          Combo.Items.Insert(0, Combo.Text);                {NEW}
        for i := 1 to Combo.Items.Count - 1 do              {NEW}
          if Combo.Items[i] = Combo.Items[0] then           {NEW}
            Combo.Items.Delete(i);                          {NEW}
        Values.Clear;                                       {NEW}
        For i := 0 to Combo.Items.Count - 1 do              {NEW}
          begin                                             {NEW}
            S := Combo.Items[i];                            {NEW}
            Values.Add(S);                                  {NEW}
          end;                                              {NEW}
      end;                                                  {NEW}
  finally
    W.Free;
  end;
end;

end.
```

Source for Colored String Grids

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure StringGrid1DrawCell(Sender: TObject; Col, Row: Longint;
      Rect: TRect; State: TGridDrawState);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure StringGrid1DblClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
  C, R: Integer;
  Value: Integer;
begin
  {fill the string grid with random integer values}
  Randomize;
  for C := 1 to StringGrid1.ColCount-1 do
    for R := 1 to StringGrid1.RowCount-1 do
      begin
        Value := Random(10) - 5;
        StringGrid1.Cells[C,R] := IntToStr(Value);
        StringGrid1.Objects[C,R] := Pointer(clBlack);
      end;
end;

procedure TForm1.StringGrid1DrawCell(Sender: TObject; Col, Row: Longint;
  Rect: TRect; State: TGridDrawState);
const
  LM = 3; {each indiviual cell's left margin}
  TM = 2; {each indiviual cell's top margin}
var
  ptr: Pointer;
begin
  {use whatever color is stored in the object's pointer}
  ptr := StringGrid1.Objects[Col, Row];
  StringGrid1.Canvas.Font.Color := LongInt(ptr);
  {let'd draw the fixed rows and the fixed columns in silver}
  if gdFixed in State then
     StringGrid1.Canvas.Brush.Color := clSilver;
```

```
  {let's draw the highlight in the following way when the cell is selected}
  if gdSelected in State then
    begin
      StringGrid1.Canvas.Brush.Color := clHighlight;
      StringGrid1.Canvas.Font.Color := clHighlightText;
    end;
  {finally, do the actual cell drawing}
  StringGrid1.Canvas.TextRect(Rect, Rect.Left + LM, Rect.Top + TM,
StringGrid1.Cells[col,row]);
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var
  C, R: Integer;
begin
  for C := 1 to StringGrid1.ColCount-1 do
    for R := 1 to StringGrid1.RowCount-1 do
      begin
        {make all the grid's objects point to nothing}
        StringGrid1.Objects[C, R] := nil;
        {if we had stored objects in the grid, we should free them like this:
        StringGrid1.Objects[C, R].Free;}
      end;
end;

procedure TForm1.StringGrid1DblClick(Sender: TObject);
begin
  With StringGrid1 do
    Case Random(4) of
      0 : Objects[Col,Row] := Pointer(clRed);
      1 : Objects[Col,Row] := Pointer(clLime);
      2 : Objects[Col,Row] := Pointer(clBlue);
      3 : Objects[Col,Row] := Pointer(clFuchsia);
    end;
end;

end.
```

Questions (And Answers) From UNDU Readers

I often get a wide variety of emailed questions from readers of UNDU. Some of them have been quite interesting and the solutions are equally interesting. Anyway, I figured "Why not let everyone help on the solution?"
Each month I will present a few questions here that readers have submitted to me and open them up to all the readers of UNDU. If you know the answer to a question, feel free to send it in to **RobertV@compuserve.com**. I will chose the best solution to the question and post it in the following issue. This way, everyone gets to see the answer!
The solutions can be anything including even shareware components that might solve a particular problem.

Last Months questions were:

Steven Lucey asked *"How do you make forms so that they will display correctly no matter the resolution or font size (large or small) at runtime?"*

I received quite a few responses on this one, but I found that the Borland discussed the issue the best. Take a look at Borland's response to this in their tech sheet #2861.

Steven Gill asked *"I am trying to work out how to add bitmaps to StringGrids. I want to use the first column as a status column with a graphic indicating the status. What's a simple way to do this?"*

Interestingly, this one was answered by an unrelated tip sent in by Paul Harding on Displaying Multi-colored Text in a String Grid. Check out the last few paragraphs and you'll see where this is going!

I didn't get any really good questions for this month, so hopefully, I will get a few for next month!

Return to Front Page

Source for TipBox.pas

```
unit tipbox;

interface

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

type
  TTipOfTheDayForm = class(TForm)
    Panel1: TPanel;
    OKBtn: TButton;
    ShowOnStartup: TCheckBox;
    Bevel1: TBevel;
    NextBtn: TButton;
    BulbImage: TImage;
    diduknow: TLabel;
    Tip: TLabel;
    procedure NextBtnClick(Sender: TObject);
    procedure OKBtnClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
  protected
    Tips: TStringList;
  public
    TipFileName: String;
  end;

var
  TipOfTheDayForm: TTipOfTheDayForm;

implementation

{$R *.DFM}

procedure TTipOfTheDayForm.NextBtnClick(Sender: TObject);
begin
   Tips.Move(1, Tips.Count - 1);
   Tip.Caption := Tips[1];
end;

procedure TTipOfTheDayForm.OKBtnClick(Sender: TObject);
begin
   if ShowOnStartup.Checked
     then Tips[0] := 'True'
     else Tips[0] := 'False';
   Tips.Move(1, Tips.Count - 1);
   Tips.SaveToFile(TipFileName);
   Tips.Free;
   Close;
end;

procedure TTipOfTheDayForm.FormShow(Sender: TObject);
begin
   Tips := TStringList.Create;
   Tips.LoadFromFile(TipFileName);
   Tip.Caption := Tips[1];
   if Tips[0]='True'
     then ShowOnStartup.Checked := True
     else ShowOnStartup.Checked := False;
end;
```

```
end.
```

Source for TipDlg.pas

```
unit tipdlg;

interface

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

type
  TTipOfTheDayDlg = class(TComponent)
  private
    FTipFileName: String;
    function GetShowStatus: Boolean;
  published
    property TipFile: String read FTipFileName write FTipFileName;
    property ShowOnStart: Boolean read GetShowStatus;
  public
    procedure Execute;
  end;

procedure Register;

implementation

function TTipOfTheDayDlg.GetShowStatus: Boolean;
   var FileHandle: Integer;
       Buffer: String;
begin
   FileHandle := FileOpen(FTipFileName, fmOpenWrite);
   if FileHandle = -1
   then begin
      Result := False;
      exit;
   end
   else begin
      FileRead(FileHandle, Buffer, 4);
      FileClose(FileHandle);
      if Buffer = 'True'
      then Result := True
      else Result := False;
   end;
end;

procedure TTipOfTheDayDlg.Execute;
begin
   TipOfTheDayForm := TTipOfTheDayForm.Create(Application);
   try
     TipOfTheDayForm.TipFileName := FTipFileName;
     TipOfTheDayForm.ShowModal;
   finally
     TipOfTheDayForm.Free;
   end;
end;

procedure Register;
begin
  RegisterComponents('UNDU', [TTipOfTheDayDlg]);
end;

end.
```

Source for TipBox.dfm

```
object TipOfTheDayForm: TTipOfTheDayForm
  Left = 187
  Top = 115
  Width = 375
  Height = 218
  BorderIcons = []
  Caption = 'Tip of the Day'
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OnShow = FormShow
  PixelsPerInch = 96
  TextHeight = 13
  object Bevel1: TBevel
    Left = 8
    Top = 8
    Width = 257
    Height = 153
  end
  object Panel1: TPanel
    Left = 16
    Top = 16
    Width = 241
    Height = 137
    BevelOuter = bvNone
    BorderWidth = 1
    Color = clWhite
    TabOrder = 0
    object BulbImage: TImage
      Left = 8
      Top = 8
      Width = 28
      Height = 37
      AutoSize = True
      Picture.Data = {
        07544269746D6170C6020000424DC602000000000000760000002800000001C00
        0000250000000100040000000000500200000000000000000000000100000001000
        0000000000000000800000800000008080008000000080008000808000000C0C0
        C000808080000000FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFF
        FF00FFFFFFFFFFFFF8FFFFFFFFFFFFFFF0000FFFFFFFFFFFF888FFFFFFFFFFFFF
        0000FFFFFFFFFFF88888FFFFFFFFFFFFF0000FFFFFFFFFF8888888FFFFFFFFFF
        0000FFFFFFFFF888808888FFFFFFFFFF0000FFFFFFFF88880F08888FFFFFFFFF
        0100FFFFFFF88880FFF08888FFFFFFFF0000FFFFFF88880FEFEF08888FFFFFFF
        0100FFFFF88880FFFFFFF08888FFFFFF0000FFFF88880FEFEFEFEF08888FFFFF
        0000FFF88880FFFFFFFFFFF08888FFFF0000FF88880FEFEF0000FFEF08888FFF
        0000FFF880FFFFF007700FFFF088FFFF0000FFFF0FEFEFE088880FEFEF0FFFFF
        0000FFF0FFFFFFF077770FFFFFF0FFFF0000FF0FEFEFEFE088880FEFEFEF0FFF
        0000FFF8FFFFFFF0BBBB0FFFFFF8FFFF0100FFFF8FEFEF0BBBBBB0EFEF8FFFFF
        0000FFFFF8FFF0BBB00BBB0FF8FFFFFF0000FFFFFF8F0BBBB00BBBB08FFFFFFF
        0000FFFFFFF0BBBBBBBBBBB0FFFFFFF5E00FFFF0FF0BBBBB00BBBBB0FF0FFFF
        0000FFFFFF0BBBBBB00BBBBBB0FFFFFFFFFFBFBFBF0BBBBBB00BBBBBB0FBFBFB
        00A0FFF80F0BBBBBB00BBBBBB0F08FFF0100BFBFBF0BBBBBB00BBBBBB0FBFBFB
        0100FFFFFF0BBBBBB00BBBBBB0FFFFFF0100FFFF0FF0BBBBB00BBBBB0FF0FFFF
        0100FFFFFFFF0BBBBBBBBBB0FFFFFFFF5E00FFFFFFFFF00BBBBBB00FFFFFFFFF
        0000FFFFFB0FFFF000000FFFF0BFFFFF0000FFFFF8BFFFFFFFFFFFFFFFB8FFFFF
        0000FFFBFFFFF0FFB0BFF0FFFFFFBFFF0000FFFFBFFFFFFFF8FFFFFFFFFBFFFF
        0000FFFFFFFFFFFFB8BFFFFFFFFFFFFFF0000FFFFFFFFFFFFFFFFFFFFFFFFFFFF
        0000FFFFFFFFFFFFBFBFFFFFFFFFFFFF00F0}
    end
```

```
    object diduknow: TLabel
      Left = 56
      Top = 24
      Width = 74
      Height = 13
      Caption = 'Did you know...'
    end
    object Tip: TLabel
      Left = 8
      Top = 64
      Width = 225
      Height = 65
      AutoSize = False
      WordWrap = True
    end
  end
  object OKBtn: TButton
    Left = 272
    Top = 8
    Width = 89
    Height = 25
    Caption = 'OK'
    Default = True
    TabOrder = 1
    OnClick = OKBtnClick
  end
  object ShowOnStartup: TCheckBox
    Left = 8
    Top = 168
    Width = 129
    Height = 17
    Caption = 'Show Tips at Startup'
    State = cbChecked
    TabOrder = 2
  end
  object NextBtn: TButton
    Left = 272
    Top = 48
    Width = 89
    Height = 25
    Caption = 'Next Tip'
    TabOrder = 3
    OnClick = NextBtnClick
  end
end
```

Source for WSC.pas

```
unit WSC;

interface

uses
  SysUtils, Messages, Classes, Forms, Windows;

type
  TMyShell = class(TComponent)
  private
    FParent : THandle;                    // keep forms handle
    FOldDefWndProc,                      // keep the forms WndProc
    FNewDefWndProc : pointer;      // our new WndProc
    procedure NewDefWndProc (var Msg : TMessage);
  public
    constructor Create (AOwner : TComponent); override;
    destructor  Destroy; override;
    procedure   Loaded; override;
  end;

implementation

// Get the forms Parent and create a pointer to our new WndProc
constructor TMyShell.Create (AOwner : TComponent);
begin
  inherited Create (AOwner);
  FParent := (AOwner as TForm).Handle;
  FNewDefWndProc := MakeObjectInstance (NewDefWndProc)
end;

// Chuck it all
destructor TMyShell.Destroy;
begin
  SetWindowLong (FParent, GWL_WndProc, longint(FOldDefWndProc));
  FreeObjectInstance (FNewDefWndProc);
  inherited Destroy
end;

// Put our WndProc into the Form
procedure TMyShell.Loaded;
begin
  inherited Loaded;
  FOldDefWndProc := pointer(SetWindowLong (FParent, GWL_WndProc,
longint(FNewDefWndProc)))
end;

// Our new WndProc - this does nothing
procedure TMyShell.NewDefWndProc (var Msg : TMessage);
begin

// put your code in here ---

// Call the forms WndProc
  with Msg do
    Result := CallWindowProc (FOldDefWndProc, FParent, Msg, wParam, lParam)
end;

end.
```

Source for Aspect.pas

```
// Aspect Component - used to control form or client area shape (aspect ratio)

unit Aspect;

interface

uses SysUtils, Messages, Classes, Forms, Windows;

type
  TAspect = class(TComponent)
  private
    FParent : THandle;
    FOldDefWndProc,
    FNewDefWndProc : pointer;
    FAspectX,
    FAspectY : integer;
    FActive,
    FClient : boolean;
    procedure NewDefWndProc (var Msg : TMessage);
  protected
  public
    constructor Create (AOwner : TComponent); override; destructor
    Destroy; override;
    procedure  Loaded; override;
  published
   // activate the aspect ratio control
   property Active : boolean read FActive write FActive default true;
   // set the X part of the aspect ratio
   property AspectX : integer read FAspectX write FAspectX default 1;
   // set the Y part of the aspect ratio
   property AspectY : integer read FAspectY write FAspectY default 1;
   // does the ratio apply to the client area or to the whole form area?
   property Client : boolean read FClient write FClient default true;
 end;

implementation

constructor TAspect.Create (AOwner : TComponent);
begin
  inherited Create (AOwner);
  FParent := (AOwner as TForm).Handle;
  FNewDefWndProc := MakeObjectInstance (NewDefWndProc);
  FActive := true;
  FClient := true;
  FAspectX := 1;
  FAspectY := 1;
end;

destructor TAspect.Destroy;
begin
  SetWindowLong (FParent, GWL_WndProc, longint(FOldDefWndProc));
  FreeObjectInstance (FNewDefWndProc);
  inherited Destroy;
end;

procedure TAspect.Loaded;
begin
  inherited Loaded;
  FOldDefWndProc := pointer(SetWindowLong(FParent, GWL_WndProc,
                            longint(FNewDefWndProc)));
```

```
end;

procedure TAspect.NewDefWndProc (var Msg : TMessage);
var
  CaptionHt,Xi,Yi : integer;
begin
  if FActive then with Msg do
    begin
      if Msg = WM_Sizing then
        begin
          if FClient then
            CaptionHt := GetSystemMetrics (sm_CYCaption)
          else
            CaptionHt := 0;
          with PRect (lParam)^ do
            case wParam of
              WMSZ_BottomRight,
              WMSZ_Bottom : Right := Left + (Bottom - Top) *
                            AspectX div AspectY - CaptionHt;
              WMSZ_BottomLeft,
              WMSZ_Right : Bottom := Top + (Right - Left)*
                            AspectY div AspectX + CaptionHt;
              WMSZ_TopRight,
              WMSZ_Left  : Top := Bottom - (Right - Left) *
                            AspectY div AspectX - CaptionHt;
              WMSZ_TopLeft,
              WMSZ_Top   : Left := Right - (Bottom - Top) *
                            AspectX div AspectY + CaptionHt;
            end;
          Result := 0;
          exit;
        end;
      if Msg = WM_GetMinMaxInfo then
        begin
          if FClient then
            CaptionHt := GetSystemMetrics (sm_CYCaption)
          else
            CaptionHt := 0;
          with PMinMaxInfo (lParam)^.ptMaxSize do
            begin
              Xi := X;
              Yi := X * FAspectY div FAspectX;
              if Yi > GetSystemMetrics (sm_CYScreen) then
                begin
                  Yi := Y;
                  Xi := Y * FAspectX div FAspectY
                end;
              X := Xi - CaptionHt;
              Y := Yi;
            end;
          Result := 0;
          exit;
        end;
    end;
  with Msg do
    Result := CallWindowProc(FOldDefWndProc,FParent, Msg,wParam,lParam);
end;

end.
```

Source for Min/Max

```
// MinMax component - used to form resizing


unit
  Minmax;

interface

uses
  SysUtils, Messages, Classes, Forms, Windows;

type
  TBeforeResizeEvent = procedure (Sender : TObject; var MaxMinInfo : TMinMaxInfo) of
object;

  TMinMaxChange = (mmMaximizedWidth, mmMaximizedHeight, mmMaximizedLeft,
mmMaximizedTop,
                   mmMinTrackWidth,  mmMinTrackHeight,  mmMaxTrackWidth,
mmMaxTrackHeight);
  TMinMaxChanges = set of TMinMaxChange;

  TMinMax = class(TComponent)
  private
    FParent : THandle;
    FOldDefWndProc,
    FNewDefWndProc : pointer;
    FActive : boolean;
    FMaximizedWidth,
    FMaximizedHeight,
    FMaximizedLeft,
    FMaximizedTop,
    FMinTrackWidth,
    FMinTrackHeight,
    FMaxTrackWidth,
    FMaxTrackHeight : integer;
    FChanges : TMinMaxChanges;
    FBeforeResize : TBeforeResizeEvent;
    procedure NewDefWndProc (var Msg : TMessage);
  protected
    constructor Create (AOwner : TComponent); override;
    destructor Destroy; override;
    procedure Loaded; override;
  public
  published
    property Active : boolean read FActive write FActive default true;
    property Changes : TMinMaxChanges read FChanges write FChanges default [];
    property MaximizedWidth : integer read FMaximizedWidth write FMaximizedWidth;
    property MaximizedHeight : integer read FMaximizedHeight write FMaximizedHeight;
    property MaximizedLeft : integer read FMaximizedLeft write FMaximizedLeft;
    property MaximizedTop : integer read FMaximizedTop write FMaximizedTop;
    property MinTrackWidth : integer read FMinTrackWidth write FMinTrackWidth;
    property MinTrackHeight : integer read FMinTrackHeight write FMinTrackHeight;
    property MaxTrackWidth : integer read FMaxTrackWidth write FMaxTrackWidth;
    property MaxTrackHeight : integer read FMaxTrackHeight write FMaxTrackHeight;
    property OnBeforeResize : TBeforeResizeEvent read FBeforeResize write
FBeforeResize;
  end;

implementation
```

```
constructor TMinMax.Create (AOwner : TComponent);
begin
  inherited Create (AOwner);

  FParent := (AOwner as TForm).Handle;
  FNewDefWndProc := MakeObjectInstance (NewDefWndProc);
  FActive := true;
  FChanges := [];
  FMaximizedLeft   := - GetSystemMetrics (sm_CXFrame);
  FMaximizedTop    := - GetSystemMetrics (sm_CXFrame);
  FMaximizedWidth  := GetSystemMetrics (sm_CXScreen) - 2 * FMaximizedLeft;
  FMaximizedHeight := GetSystemMetrics (sm_CYScreen) - 2 * FMaximizedTop;
  FMinTrackWidth   := GetSystemMetrics (sm_CXMin);
  FMinTrackHeight  := GetSystemMetrics (sm_CYMin);
  FMaxTrackWidth   := FMaximizedWidth;
  FMaxTrackHeight  := FMaximizedHeight
end;

destructor TMinMax.Destroy;
begin
  FBeforeResize := nil;
  SetWindowLong (FParent, GWL_WndProc, longint(FOldDefWndProc));
  FreeObjectInstance (FNewDefWndProc);
  inherited Destroy
end;

procedure TMinMax.Loaded;
begin
  inherited Loaded;
  FOldDefWndProc := pointer(SetWindowLong (FParent, GWL_WndProc,
longint(FNewDefWndProc)))
end;

procedure TMinMax.NewDefWndProc (var Msg : TMessage);
begin
  with Msg do
    if (Msg = WM_GetMinMaxInfo) and FActive then
    begin
      with PMinMaxInfo (lParam)^ do
      begin
        if mmMaximizedWidth in FChanges then
          ptMaxSize.X := FMaximizedWidth;
        if mmMaximizedHeight in FChanges then
          ptMaxSize.Y := FMaximizedHeight;
        if mmMaximizedLeft in FChanges then
          ptMaxPosition.X := FMaximizedLeft;
        if mmMaximizedTop in FChanges then
          ptMaxPosition.Y := FMaximizedTop;
        if mmMinTrackWidth in FChanges then
          ptMinTrackSize.X := FMinTrackWidth;
        if mmMinTrackHeight in FChanges then
          ptMinTrackSize.Y := FMinTrackHeight;
        if mmMaxTrackWidth in FChanges then
          ptMaxTrackSize.X := FMaxTrackWidth;
        if mmMaxTrackHeight in FChanges then
          ptMaxTrackSize.Y := FMaxTrackHeight
      end;

      if Assigned (FBeforeResize) then
        FBeforeResize (Self, PMinMaxInfo (lParam)^);

      Result := 0
    end else
```

```
      Result := CallWindowProc (FOldDefWndProc, FParent, Msg, wParam, lParam)
end;

end.
```