



Well, this month brings another new feature into my work with Delphi: The release of my first "for-sale" product. I thought long and hard about whether or not I was going to mention it in UNDU. After all, UNDU is a non-commercial endeavor and I didn't want to turn it into one. In the end however, I felt I could walk that fine line. If you are interested in learning more about this new product, click on the "IniOut" link below. Otherwise, read-on... There's lots of other great stuff in this issue!

The randomly chosen winner of the UNDU prize for material submitted this issue is **William A. Portillo**. His prize is the updated edition of the Coriolis book, *Delphi 2 Programming Explorer*. 806 pages packed with very useful info on Delphi 2.0 complete with excellent source examples on a CD-Rom. To learn more about the UNDU prizes, refer to issue #17. Thanks again for everyone's excellent contributions to UNDU!

[Object Express by OOPSoft Inc.](#)

[Tips & Tricks](#)

[The Component Cookbook](#)

[IniOut Component Property Manager](#)

[New Book - Delphi Component Design by Danny Thorpe](#)

[UNDU Subscriber List](#)

[Index of Past Issues](#)

[Where To Find UNDU](#)



Index of Past Issues

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

Issue #1 - March 15, 1995

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

Issue #2 - April 1, 1995

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

Issue #3 - May 1, 1995

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

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

[Issue #4 - May 24, 1995](#)

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

[Issue #5 - June 26, 1995](#)

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

[Issue #6 - July 25, 1995](#)

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

[The Loaded Method](#)
[Tips & Tricks](#)
[What's In Print](#)

[Issue #7 - August 31, 1995](#)

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

[Issue #8 - October 10, 1995](#)

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

[Issue #9 - November 9, 1995](#)

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

The Delphi Magazine
Tips & Tricks
Using Sample Applications

Issue #10 - December 12, 1995

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

Issue #11 - January 18th, 1996

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

Issue #12 - February 23rd, 1996

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

The Component Cookbook

Issue #13 - May 1st, 1996

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

Issue #14 - June 1st, 1996

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

Issue #15 - August 1st, 1996

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

Issue #16 - September 1st, 1996

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

[Splash Screen with a Time Delay](#)

[Issue #17 - October 1st, 1996](#)

[Does Windows 95 give you a Square Deal?](#)

[The Great StringList](#)

[Manipulating Regions with Delphi](#)

[Tips & Tricks](#)

[When Delphi's smart-linker doesn't seem so smart](#)

[Cut, Copy, & Paste](#)

[A Quick Way of Setting the Tab Order](#)

[Background Bitmaps on Forms](#)

[Non-Rectangular Windows](#)

[Return to Front Page](#)



Where To Find UNDU

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

1. UNDU's official web site at <http://www.informant.com>. This site houses all the issues in both HTML and Windows HLP format.
2. *Borlands* Delphi forum on CompuServe (**GO DELPHI**) in the "Delphi IDE" file section. This forum will only hold the issues in Windows HLP format.
3. *Informant Communications* forum also on CompuServe (**GO ICGFORUM**) in the "Delphi 3rd Party" file section. Again, this forum will only hold issues in the Windows HLP format.



Tips & Tricks

You can save just about anything in an INI file. But when it comes to some of the more complex types, the INI File support stops short. Learn how to store complete font definitions in the tip on [Storing Fonts in INI Files](#).

Ever want to easily allow sorting on any column in a DBGrid just by clicking on the column header? Well, check out [Sorting Columns in a DBGrid](#).

Don't know what sub-version of Delphi you are running? Then check out [What's Your Version Number](#) to make sure you have the latest!

Last month I showed how to put bitmaps on the background of your forms. Unfortunately I failed to mention that the code presented was specific to Delphi 2.0. The only change necessary to make it work with Delphi 1.0 is to change the line in the Form Create that looks like this:

```
BackgroundBitmap.LoadFromResourceName(hInstance, 'MYBITMAP');
```

into one that looks like this:

```
BackgroundBitmap.Handle := LoadBitmap(hInstance, 'MYBITMAP');
```

Delphi 2.0 added the method LoadFromResourceName, but I forgot it wasn't in Delphi 1.0

Remember the article on [Drawing MetaFiles](#) in Delphi in issue #12 of UNDU? Well, Grahame Marsh has an update on Delphi 2.0's support for [Drawing MetaFiles](#).

Last month, Brad Evans brought out an article on implementing [Cut, Copy, & Paste](#). Now he has gone one step further by adding Undo! Learn more in [Adding Undo to your Edit Menu](#).

In the past, we have shown you how to put bitmaps and wave files into your Delphi EXE. Well, now... how about [How To Put Anything In Your Delphi EXE!!!](#)

Where can you turn for more information on Delphi? Try the [Delphi Newsgroups](#) on the Internet!

[Return to Front Page](#)



The Component Cookbook

This month, Grahame Marsh returns with another useful component that allows you to view and edit the contents of the clipboard. Learn more about it in the section [A Simple Clipboard Viewer Component](#).

[Return to Front Page](#)



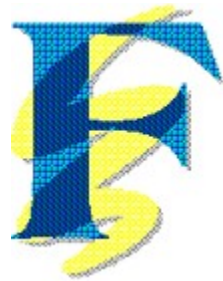
UNDU Subscriber List

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

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

Thats all there is to it!

[Return to Front Page](#)



Storing Fonts in INI Files

by Jose Alfredo Garcia Guirado - jgarcia@pinos.com

Frequently, I need to store Fonts in INIFILES and I had to do a lot of boring lines in the INI file to store the pitch, color and so on. In order to avoid that, I created two methods that takes the Font and stores it in a single line of the INIFILE.

These methods are:

```
Function ReadFont(Const Section,Ident:String; default:TFont):TFont;  
Procedure WriteFont(Const Section,Ident:String;Value:TFont);
```

The first two parameters are the same as those found in TINIFILE's methods unit (**Section** and **Ident**), and the third is the font that you want to store. The procedure WriteFont processes and compresses it in a single line of the INI File. *The Pitch, Color, Name, Size* and *Style* are stored in a simple code that you can restore by the Function ReadFont.

The parameter **Default** in ReadFont, gives you a 'Default' Font if the line specified by **Ident** is empty in the INI Files. This avoids a **nil** font return value.

If you are adventurous and don't mind modifying your VCL source, you can add these procedures to your INIFILES unit and have them always available.

Here the [source code](#) for these methods.

[Return to Tips & Tricks](#)

[Return to Front Page](#)

 **The Unofficial Newsletter of Delphi Users - Issue #18 - November 1996**
Source code for Fonts in INI Files

```
unit IniFile2;

{Extension of Inifiles to support ReadFont WriteFont }

interface

uses IniFiles, Graphics;

Type
TNewIniFile = Class(TIniFile)
    procedure WriteFont(Const Section, Ident: String; Value: TFont);
    function ReadFont(Const Section, Ident: String; default: TFont): TFont;
end;

implementation

uses
    SysUtils, WinProcs, Consts;

{Write the Font in [Section], Line Ident= font code strings}
procedure TNewIniFile.WriteFont(Const Section, Ident: String; Value: TFont);
var
    Str :String;
    v :char;
Begin
    Str := '';
    Str := Value.Name + ',';
    Str := Str + ColorToString(Value.Color) + ',';
    Str := Str + IntToStr(Value.Size) + ',';
    Case Value.Pitch of
        fpdefault   : v :='d';
        fpvariable  : v :='v';
        fpfixed     : v :='f';
    end;
    Str:= Str + v + ',';
    if fsBold in Value.Style then Str:=Str + 'b';
    if fsItalic in Value.Style then Str:=Str + 'i';
    if fsUnderline in Value.Style then Str:=Str + 'u';
    if fsStrikeOut in Value.Style then Str:=Str + 's';
    WriteString(Section, Ident, Str);
end;

{Return the Font stored in [Section], line Ident= Font code String}
{If the font is not found the default font will be returned}
function TNewIniFile.ReadFont(Const Section, Ident: String;
                               default: TFont): TFont;

var
    FFont      : TFont;
    Str,sc : String;
    I          : byte;
Begin
    Str := ReadString(Section, Ident, '');
    ReadFont := default;
    {read font name}
    I := Pos(',', Str);
    if i = 0 then exit;
    FFont := TFont.Create;
```

```

FFont.Name := Copy(Str,1,i-1);
Delete(Str,1,i);
{read font color}
i:=Pos(',',Str);
if i = 0 then exit;
try
  FFont.Color:= StringtoColor(Copy(Str,1,(i-1)));
except
  FFont.Free;
  exit;
end;
Delete(Str,1,i);
{read font size}
i:=Pos(',',Str);
if i = 0 then exit;
try
  FFont.Size:= StrToInt(Copy(Str,1,(i-1)));
except
  FFont.Free;
  exit;
end;
Delete(Str,1,i);
{read font pitch}
i:=Pos(',',Str);
if i = 0 then
  begin
    FFont.Free;
    exit;
  end;
sc:=Copy(Str,1,(i-1));
if sc = 'd' then FFont.pitch := fpdefault;
if sc = 'v' then FFont.pitch := fpvariable;
if sc = 'f' then FFont.pitch := fpfixed;
Delete(Str,1,i);
{read font style}
FFont.Style:=[];
While byte(Str[0]) > 0 do
  Begin
    Case Str[1] of
      'b': FFont.Style := FFont.Style + [fsbold];
      'i': FFont.Style := FFont.Style + [fsitalic];
      'u': FFont.Style := FFont.Style + [fsUnderline];
      's': FFont.Style := FFont.Style + [fsStrikeout];
    end;
    Delete(Str,1,1);
  end;
ReadFont.Assign(FFont);
FFont.Free;
end;

end.

```

[Return to Article](#)

[Return to Front Page](#)



INIOUT© is a *Component Property Manager* for Delphi. With it you can easily provide INI file support to your Delphi application with **virtually no code** and a runtime footprint of only **6k!**

Sure... I've Seen This Before!

Yes, there are other products available that allow a component to save its value off to an INI file and restore the value later. Most of these perform this feat by taking a standard Delphi component and deriving a new component from it with the code necessary to link it to an INI file. The result? You now have a new component that is *INI-file aware*. **But what are the disadvantages of such an approach?**

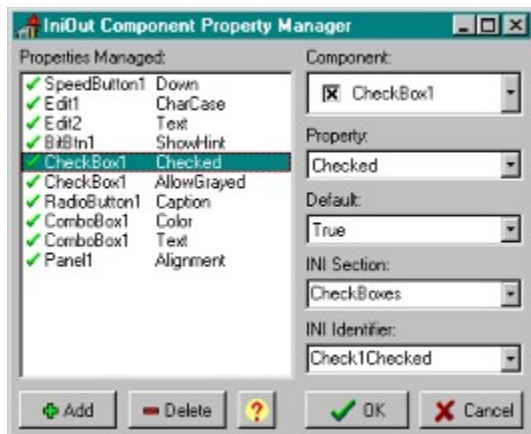
1. Only the components provided are INI-File aware - You may have 15 or 20 INI-aware components to use. But what if you want to save the value of a component that you don't have an INI-aware version of? Plus, this is 15 or 20 more components you need to manage on your Component Palette!
2. Only one property will be saved by the component - Typically, the author of the INI-aware components might well feel that there is really only one property of a component that needs to be written to and read from an INI file. For example, a TEdit would most likely want to write out its *Text* property. But what if you want to save the TEdit's *location* on the form, or its *Enabled* state?
3. Won't work on any 3rd party components - You won't be able to make any components INI File aware that you get from other companies. Plus, what if you made your own version of TEdit? Unless you want to rewrite your component to descend from a TIniEdit, then your own creation also won't be INI-file aware.

The Solution: The INIOUT© Component Property Manager

With INIOUT, you don't need to deal with these issues anymore. Now there is a complete solution to making components Ini-File aware.

1. Works with any component - Since INIOUT is a manager of component properties and not just new INI-aware components derived from existing ones, it doesn't matter what component you want to make Ini-File aware. INIOUT will work with any component or control you might have. Even 3rd party controls or controls you have designed yourself!
2. Works with virtually any property - Want to save the color of that panel? The location of some TEdit fields? How about the Caption of a RadioButton or the Font of a Memo field? With INIOUT, this isn't a problem, because you can read or write virtually any property of any component to an INI File.

Here's a quick peek at how IniOut looks:



INIOUT can be found in a number of locations including the **Informant Communications** Web site (<http://www.informant.com/unddu/iniout.zip>) as well as at **The Delphi Information Connection** (<http://www.delphi32.com/apps>). On CompuServe, you can find INIOUT on the **Informant Communications** forum (GO ICGFORUM) in the **Delphi 2 3rd Party Forum**. The file name is INIOUT.ZIP. If you don't see it on these forums, please keep trying. This issue of UNDU is coming out about the same time I will be posting INIOUT, so there might be a day or two where one is there and the other isn't.

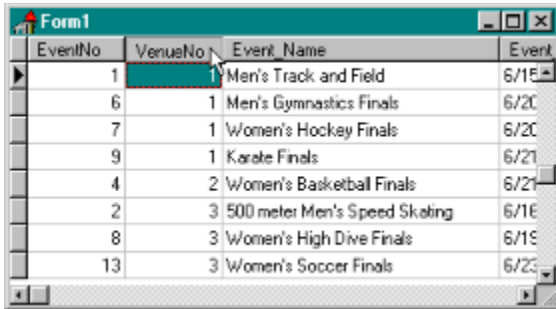
Currently, INIOUT is a Delphi 2.0 component but a 1.0 compatible version is in the works and should be ready shortly. If you would like to be kept up to date on any developments with INIOUT, simply send me an email and I will advise you when there are new releases or updates. Send EMail to Robert Vivrette at 76416.1373@compuserve.com.

Please note that this downloadable version of INIOUT is a slightly restricted demo version. For a mere US\$20 you can get the full version from me by mail or email. The added features in the full version are detailed in the document that accompanies the package. Source code is also available for US\$129. Send your check or money order to:

Robert Vivrette
PO Box 1947
Swansboro, NC 28584-1947

Make checks payable to "Robert Vivrette". Be sure to include your email address (if you have one) so I can send you the appropriate version electronically.

[Return to Front Page](#)



EventNo	VenueNo	Event Name	Event
1		Men's Track and Field	6/1E
6	1	Men's Gymnastics Finals	6/2C
7	1	Women's Hockey Finals	6/2C
9	1	Karate Finals	6/2I
4	2	Women's Basketball Finals	6/2I
2	3	500 meter Men's Speed Skating	6/1E
8	3	Women's High Dive Finals	6/1E
13	3	Women's Soccer Finals	6/2C

Sorting Columns in a DBGrid

by Robert Vivrette - 76416.1373@compuserve.com

Many professional applications will display data in grid fields and allow you to sort on any one of the columns simply by clicking on the column header. Although what is proposed here is not the best way to accomplish this, it is a fairly simple way to mimic the same behavior.

The key hurdle in this problem is the DBGrid itself. It has no OnClick or OnMouseDown events, so it really was not designed to capture this kind of input. It does provide an OnDoubleClick, but this really doesn't work too well. What we need is a way to make the column headers clickable. Enter the THeaderControl component.

THeaderControl is a component that comes in Delphi 2.0 and provides the basic functions that we want. It can detect clicks on its individual panels, and the panels even go up and down when pressed (like a button). The key is to connect the THeaderControl to the DBGrid. Here is how it is done:

First, start a new application. Drop a THeaderControl on the form. It will automatically align to the top edge of the form. Now drop a DBGrid on the form and set its Align property to alClient. Next, add a TTable, and TDataSource component. Set the Tables DatabaseName property to DBDEMOS and its TableName to EVENTS.DB. Set the DataSource's DataSet property to point at Table1 and the DBGrid's DataSource property to point to DataSource1. Set Table's Active property to False in case it has been turned on. Now the fun begins!

Now we need to setup the THeaderControl component to look like the DBGrid's column headers. This will be done in code in the Form's FormCreate method. DoubleClick on Form1's OnCreate event and enter the following code:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  TheCap : String;
  TheWidth,a : Integer;
begin
  DBGrid1.Options := DBGrid1.Options - [dgTitles];
  HeaderControl1.Sections.Add;
  HeaderControl1.Sections.Items[0].Width := 12;
  Table1.Exclusive := True;
  Table1.Active := True;
  For a := 1 to DBGrid1.Columns.Count do
    begin
      with DBGrid1.Columns.Items[a-1] do
        begin
          TheCap := Title.Caption;
          TheWidth := Width;
        end;
      with HeaderControl1.Sections do
        begin
          Add;
          Items[a].Text := TheCap;
          Items[a].Width := TheWidth+1;
          Items[a].MinWidth := TheWidth+1;
          Items[a].MaxWidth := TheWidth+1;
        end;
    end;
end;
```

```

    end;
  try
    Table1.AddIndex(TheCap,TheCap, []);
  except
    HeaderControl1.Sections.Items[a].AllowClick := False;
  end;
end;
Table1.Active := False;
Table1.Exclusive := False;
Table1.Active := True;
end;

```

Since the THeaderControl will be taking the place of the Grid's column headers, we first remove (set to False) the dgTitles option in the DBGrid's Options property. Then, we add a column to the HeaderControl and set its width to 12. This will be a blank column that is the same width as the Grid's status area on the left.

Next we need to make sure the Table is opened for Exclusive use (no other users can be using it). I will explain why in just a bit.

Now we add the HeaderControl sections. For each one we add, we will be giving it the same text as the caption of that column in the DBGrid. We loop through the DBGrid columns, and for each one we copy over the column's caption and width. We also set the HeaderControl's MinWidth and MaxWidth properties to the same as the column width. This will prevent the column from being resized. If you need resizable columns, you will need a bit more code, and I wanted to keep this short and sweet.

Now comes the interesting part. We are going to create an index for each column in the DBGrid. The name of the index will be the same as the columns title. This step is in a try..finally structure because there are some fields that cannot be indexed (Blobs & Memos for example). When it tries to index on these fields, it will generate an exception. We catch this exception and turn off the ability to click that column. This means that non-indexed columns will not respond to mouse clicks. The creation of these indexes is why we had to open the table in Exclusive mode. After we are all done, we close the table, set Exclusive off and reopen then table.

One last step. When the HeaderControl is clicked, we need to turn on the correct index for the Table. The HeaderControl's OnSectionClick method should be as follows:

```

procedure TForm1.HeaderControl1SectionClick(
    HeaderControl: THeaderControl;
    Section: THeaderSection);
begin
  Table1.IndexName := Section.Text;
end;

```

That's it! When the column is clicked, the Table's IndexName property is set to the same as the HeaderControl's caption.

Pretty simple, huh? There is a lot of room for improvement however. It would be nice if clicking on a column a second time would reverse the sort order. Also, column resizing would be a nice added touch. I am going to leave these to you folks!

[Return to Tips & Tricks](#)

[Return to Front Page](#)



What's your Version Number?

In Delphi, there is a quick way to get your version number. Simply call up the Delphi About Box and type in the word VERSION while holding down the ALT key.

The original version of Delphi 2.0 reports **2.17.53.0** and the upgrade that I have to Delphi 2.0 reports **2.17.76.0**

If you need an upgrade to Delphi, contact Borland, not me!

[Return to Tips & Tricks](#)

[Return to Front Page](#)

Drawing Metafiles with Delphi 2

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

Back in Issue #12, I presented code for [Drawing Metafiles](#). A number of people have contacted me pointing out that the TDrawMetafile code does not work under Delphi 2 - it produces an Invalid Metafile Format exception. Users of Delphi 2 should use the built-in metafile draw object. Look up TMetafileCanvas in the Delphi documentation or help file. It is fairly easy to write conditional compiles for code which is intended to be compiled under both Delphi 1 and 2.

[Return to Tips & Tricks](#)

[Return to Front Page](#)



A Simple Clipboard Viewer Component

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

I recently wrote a text utility which involved a lot of cutting and pasting to the clipboard. It became apparent that to see the clipboard contents would be good idea as I was frequently interrupted and often had important text cut to the clipboard which I then lost with another cut. I initially used the Microsoft clipboard viewer, but my MDI application was best used zoomed to full screen and task switching was yet something else to do. So to make life really easy I looked at having a clipboard display as an MDI Child within my app. In the end it developed into a component descended from a TMemo.

Putting your own clipboard view into windows is remarkably simple:

1. Call the SetClipboardViewer() API function - this takes your window handle as a parameter and returns the handle of the next window in the clipboard chain (or zero if there is no next window):

```
FNextClip : THandle;  
.....  
FNextClip := SetClipboardViewer (Handle)
```

2. Respond to the WM_DrawClipboard message which is called when the clipboard contents are changed by doing what drawing you want and then passing the message onto the next viewer. A minimal, do nothing method would be:

Interface

```
procedure WMDrawClipboard(var Msg: TWMDrawClipboard);  
message WM_DrawClipboard;
```

Implementation

```
procedure TClipboardViewer.WMDrawClipboard(var Msg: TWMDrawClipboard);  
begin  
  { do nothing }  
  if FNextClip <> 0 then  
    SendMessage(FNextClip, WM_DrawClipboard, 0, 0)  
end;
```

3. Respond to the WM_ChangeCBChain message. This is called when the clipboard viewer chain is changed. The parameters sent are the viewer handle being removed and the next viewer handle in the chain. If the locally held next viewer (FNextClip) is the Next parameter then we can service the call by putting the Next parameter into our local variable. Otherwise we must pass the call onto the next viewer in the chain.

Interface

```
procedure WMChangeCBChain(var Msg :TWMChangeCBChain);  
message WM_ChangeCBChain;
```

Implementation

```
procedure TClipboardViewer.WMChangeCBChain(var Msg: TWMMChangeCBChain);
begin
  with Msg do
    if FNextClip = Remove then
      FNextClip := Next
    else
      if FNextClip <> 0 then
        SendMessage(FNextClip, WM_ChangeCBChain, Remove, Next)
  end;
end;
```

4. Finally, we need to respond the WMDestroy message by telling the clipboard chain that our viewer is to be removed:

Interface

```
procedure WMDestroy(var Msg: TWMDestroy); message WM_Destroy;
```

Implementation

```
procedure TClipboardViewer.WMDestroy(var Msg : TWMDestroy);
begin
  ChangeClipboardChain(Handle, FNextClip);
  inherited;
end;
```

To write a clipboard viewer it is now a simple matter of writing the display code for the case where the clipboard holds the format that you are interested in. In this example I wrote the component to be descended from a TMemo for the ease of using the TMemo to display the text. So the text draw method becomes, in it's simplest form:

```
procedure TClipboardViewer.WMDrawClipboard(var Msg: TWMDrawClipboard);
begin
  Lines.Clear;
  if Clipboard.HasFormat(cf_text) then
    PasteFromClipboard;
  if FNextClip <> 0 then
    SendMessage(FNextClip, WM_DrawClipboard, 0, 0)
end;
```

In the full method I save and restore the value of the ReadOnly property and include a default text message if the clipboard doesn't contain text.

The final twist, is to allow the user to edit the TMemo contents and then post the changes back to the clipboard. This is provided as a second component. It has Post, Clear and Revert methods which I call using button clicks.

I have been working on a full feature clipboard viewer which will display text and graphics, and play sound and video clips. Please contact me if you are interested in this version as it exists so far. I am stuck on a couple of points but it is basically working.



[Complete source for Clipboard Viewer](#)

[Return to Component Cookbook](#)

[Return to Front Page](#)

 **The Unofficial Newsletter of Delphi Users - Issue #18 - November 1996**
Source for Clipboard Viewer

```
{ Clipboard Text Viewer and Editor Components }

unit
  Clipview;

interface

uses
  Classes, Clipbrd, Forms, StdCtrls, WinTypes, Messages, WinProcs;

type
  TClipboardViewer = class (TMemo)
  private
    FNextClip : THandle;
    FNoText : string;
    procedure LoadFromClipboard;
    procedure Loaded; override;
    procedure WMDrawClipboard (var Msg: TWMDrawClipboard); message
  WM_DrawClipboard;
    procedure WMChangeCBChain (var Msg : TWMChangeCBChain); message
  WM_ChangeCBChain;
    procedure WMDestroy (var Msg : TWMDestroy); message WM_Destroy;
    function StoreNoText : boolean;
  public
    constructor Create (AOwner : TComponent); override;
  published
    property NoTextWarning : string read FNoText write FNoText stored
  StoreNoText;
  end;

type
  TClipboardEditor = class (TClipboardViewer)
  public
    constructor Create (AOwner : TComponent); override;
    procedure Post;
    procedure Revert;
    procedure Clear;
  end;

implementation

constructor TClipboardViewer.Create (AOwner : TComponent);
begin
  inherited Create (AOwner);
  BorderStyle := bsNone;
  ReadOnly := true;
  ScrollBars := ssBoth;
  Wordwrap := false;
  FNoText := '<no text on clipboard>'
end;

procedure TClipboardViewer.Loaded;
begin
  inherited Loaded;
  FNextClip := SetClipboardViewer (Handle)
end;

function TClipboardViewer.StoreNoText : boolean;
begin
```

```

    Result := FNoText <> ''
end;

procedure TClipboardViewer.LoadFromClipboard;
var
    R : boolean;
begin
    R := ReadOnly;
    ReadOnly := false;
    try
        Lines.Clear;
        if Clipboard.HasFormat (cf_text) then
            PasteFromClipboard
        else
            if FNoText <> '' then
                Lines.Add (FNoText);
            Modified := false
        finally
            ReadOnly := R
        end
    end;
end;

procedure TClipboardViewer.WMDrawClipboard (var Msg : TWMDrawClipboard);
begin
    LoadFromClipboard;
    if FNextClip <> 0 then
        SendMessage (FNextClip, WM_DrawClipboard, 0, 0)
    end;
end;

procedure TClipboardViewer.WMChangeCBChain (var Msg : TWMChangeCBChain);
begin
    with Msg do
        if FNextClip = Remove then
            FNextClip := Next
        else
            if FNextClip <> 0 then
                SendMessage (FNextClip, WM_ChangeCBChain, Remove, Next)
            end;
    end;
end;

procedure TClipboardViewer.WMDestroy (var Msg : TWMDestroy);
begin
    ChangeClipboardChain (Handle, FNextClip);
    inherited
end;

{--- Clipboard Editor ---}

constructor TClipboardEditor.Create (AOwner : TComponent);
begin
    inherited Create (AOwner);
    ReadOnly := false
end;

procedure TClipboardEditor.Post;
begin
    if Lines.Count = 0 then
        Clipboard.Clear
    else begin
        SelectAll;
        CopyToClipboard
    end;
    Modified := false
end;
end;

```



```
procedure TClipboardEditor.Clear;
begin
  inherited Clear;
  Modified := true
end;

procedure TClipboardEditor.Revert;
begin
  if Modified then
    LoadFromClipboard
  end;
end.
```

[Return to Article](#)

[Return to Component Cookbook](#)

[Return to Front Page](#)

 **The Unofficial Newsletter of Delphi Users - Issue #18 - November 1996**
New Book!

Delphi Component Design by Danny Thorpe - Available November 1996
The Inside Story of Delphi's 32-bit Component Architecture and Development

FEATURES:

A Borland insider reveals the secrets behind component analysis and design, implementation details, and design-time tools. The book highlights the Delphi Visual Component Library (VCL) and discusses how to take advantage of little-known VCL classes and services. It also shows how to dramatically improve a component's ease-of-use, code reuse, flexibility, and performance.

The CD-ROM contains utilities including a royalty-free fractal image decompression library.

The book was written for Delphi application and component writers using the Delphi VCL.

Danny Thorpe is an R&D engineer on Borland's Delphi development team. He has served as technical editor and advisor for dozens of Delphi programming books and written articles for PC Magazine and Dr. Dobb's Journal.

Delphi Component Design is available November 1996 with a suggested retail price of \$36.95. It is 368 pages and includes a CD-ROM. ISBN 0-201-43136-6

[Return to Front Page](#)



Adding Undo to your Edit Menu

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

As soon I finished last months tip on [cut copy & paste](#) my users wanted an undo. As you would expect undo is just as easy as copy & paste was. But (why is there always a but) it isn't always possible for the users to choose undo, ie there is nothing to undo. The undo menu item must be dimmed at times to show the users they can't undo the previous operation.

Firstly lets do the code for undo. Just throw the following code at the Undo menu items' on click event.

```
procedure TfrmQuotes.mniUndoClick(Sender: TObject);
begin
  SendMessage(ActiveControl.Handle, EM_Undo, 0, 0);
end;
```

Simple as that. The harder bit was working out how to enable and disable the undo menu item. Well back to the Win32 API I went, I knew there must be a message in there somewhere for it, there was. However this message, unlike the others, returns something. As if to prove it was meant to be easy the message returns a value that can be used as a boolean. Check out the code I have attached to my edit menu:

```
procedure TfrmQuotes.mnuEditClick(Sender: TObject);
begin
  {Before the menu appears enable or disable the undo
  menu option as required.}
  mniUndo.Enabled := SendMessage(ActiveControl.Handle,
    EM_CanUndo, 0, 0);
end;
```

This method has a few drawbacks (ie users can't undo a database save or delete) but it will keep the users happy for a while.

About the Author

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

[Return to Tips & Tricks](#)

[Return to Front Page](#)



Put Anything In Your Delphi EXE!

by *William A. Portillo* - hallcom@wantree.com.au

I don't know if this is useful stuff but after a couple of weeks of playing with the resources side of Delphi, I ended up writing a routine that will "extract" other files out of a Delphi EXE. I found this useful for distributing little picture or sound files with the application or as a setup program with internal files. No doubt many of you will find other uses for this technique.

First, I create an RC project with NOTEPAD.EXE and ARJ.EXE in it as follows (we'll call it RESJUNK.RC):

```
NOTEPAD EXEFILE C:\WINDOWS\notepad.exe
ARJ EXEFILE C:\UTILS\ARJ.EXE
```

then I compile it with BRCC32 into a RES file. After this I include it in my Delphi project by using the \$R compiler directive like this:

```
{ $R RESJUNK.RES }
```

and extract this files by using the TResourceStream class.

```
procedure ExtractRes(ResType, ResName, ResNewName : String);
var
  Res : TResourceStream;
begin
  Res := TResourceStream.Create(Hinstance, ResName, Pchar(ResType));
  Res.SaveToFile(ResNewName);
  Res.Free;
end;
```

I hope this proves to be relevant to someone at least. If you want to share ideas on this, please email me. If you mail me back, please put "WP:" on the subject line. This lets my boss know that the mail is for me.

[Return to Tips & Tricks](#)

[Return to Front Page](#)

Delphi-related Newsgroups

Need more help with Delphi? There's always plenty going on in the Delphi newsgroups!

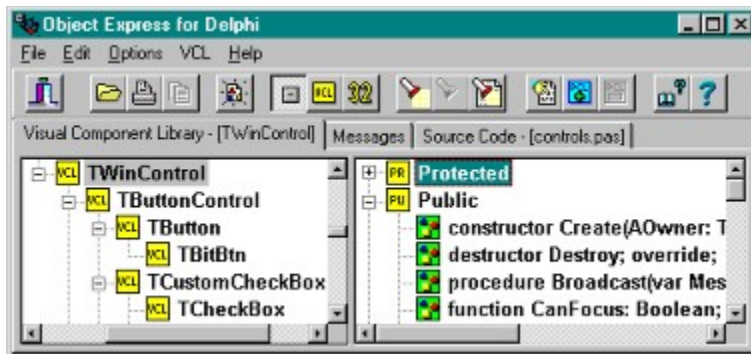
```
alt.comp.lang.borland-delphi
comp.lang.pascal.delphi.components.misc
comp.lang.pascal.delphi.components.usage
comp.lang.pascal.delphi.components.writing
comp.lang.pascal.delphi.databases
comp.lang.pascal.delphi.misc
delphi-talk@bridge.net: Delphi Talk-list
```

[Return to Tips & Tricks](#)

[Return to Front Page](#)

 **The Unofficial Newsletter of Delphi Users - Issue #18 - November 1996**
Object Express - by OOPSoft Inc.

A quick look by Robert Vivrette



When I first got a look at Object Express for Delphi, I must say my first thought was: "So what... its just a fancy version of Delphi's Browser Window." It took me a while to get over that initial impression, but I am glad that I did... there is much, much more to Object Express than meets the eye.

OOPSoft describes the purpose of Object Express as follows:

- 1. Our first goal with Object Express is to provide an environment that allows lightning access to the entire Visual Component Library in an easy to use tree view manner at your desktop. The VCL is an excellent source of information when programming in Delphi but until now there was no easy way to get around the 300+ classes within 70+ files spread across 4 directories. Object Express manages all these files so you don't have to be bothered with looking in directories to find the file where TComponent resides in.*
- 2. Our second goal with Object Express is to give the programmer the ability to manage their own classes (Personal Component Library - PCL) within the same VCL Tree and gain the same flexibility and speed that comes in Object Express to manage the VCL.*
- 3. Our third goal with Object Express is to give the programmer the ability to reuse classes in the VCL/PCL with almost no typing required. With drag and drop inheritance, simply drag a class off the VCL Tree and a new class will be created with all properties and inherited/overrideable methods included and converted to override. Drag additional properties, methods and prebuilt messages to your new class. It's that easy.*

With that in mind lets see what Object Express can do. As you can see in the screen shot above, the left pane shows a complete hierarchy of the Delphi VCL. Further, you can also see all of your own custom classes in the same tree (OOPSoft categorizes these as part of your PCL for "Personal Component Library").

Clicking on any class on the left pane will show a complete interface description in the right pane. So far nothing new. But if you select a class and click on a little button on the toolbar, Object Express immediately creates a descendant class based off of that class. You are asked to name it and then you will see it in a little sub-window. Once you have this sub-window open, you can drag any of the hundreds of windows messages onto the Private, Public, or Protected sections and the program will automatically add them to the class - complete with interface definition and implementation boilerplate code.

This drag and drop capability of Object Express is very addictive. You want your custom control to handle a specific window message? No problem, just go to the messages tab and drag a copy onto your class. Need another property or method from a different class added to your own? No problem again, just drag it over.

Once you are done, you can save the class, print it out, add it to your own Personal Component Library, or anything else you can think of.

Even after working with Object Express for a little while, I still feel that there is so much more that I havent even touched. A very useful tool indeed for Custom Delphi Component Developers!

[OOPSoft Press Release - Object Express](#)

[Return to Front Page](#)

Object Express Press Release

OOPSoft Inc. introduces software for managing the VCL and inherited classes.

DALLAS, TX - OOPSoft, Inc. has introduced innovative programming software that makes it easier and faster to navigate through both the VCL and inherited classes. The product, called Object Express, utilizes an easy-to-use tree-view format, and is currently available for Delphi 2.0 and Delphi 1.0. It is designed for use with Microsoft Windows 95 and Windows NT.

With Object Express, programmers can browse both the 16/32 bit VCL Tree and right click access a classes source code, directly inherit classes from the VCL Tree, add their own classes to the VCL tree and access Delphi's help files.

Quick Search provides direct access to the more than 300 classes that make up the VCL. The user clicks on the Quick Search button, then enters the class name. With lightening speed, Quick Search finds the class and displays where it's located on the tree. It also lists all of the class' properties and methods - from protected to published. A simple right click will then take the user to the source where the class is defined.

Another feature is its drag and drop inheritance capabilities, which greatly simplifies the process of creating new classes. The user now has the power to simply click on a particular class, and drag and drop it off the tree. This improves efficiency and lessens the chance of error, since there is no need to type in code.

Using Object Express, it's also possible to gain direct access to Delphi's help files. Using the Extended Help button, the user can get in-depth information on a class in the VCL tree, an object in the Class Object tree, or a message in the Messages tree.

Possibly the most innovative aspect of Object Express is that it enables users to add their own classes to the VCL tree. This creates the Personal Component Library, or PCL. Now, programmers have more flexibility than ever before. That's because all the conveniences that Object Express brings to managing the VCL - including tree-view, Quick Search, and drag and drop inheritance - are now available for managing inherited classes.

Object Express also makes it possible to create and manage multiple inherited classes

simultaneously. This eliminates the need to close out one file in order to work on another. For the user, it means having the ability to create inherited classes as fast as the resources- of the PC allow.

For more information on Object Express Version 1.0, including how to obtain a free screen-cam demonstration, contact OOPSoft, Inc. directly by phone: 1-888-OOPSoft (1-888-667-7638) or by e-mail: OOPSoft@Airmail.net.

Price: us \$159

Phone: 1-888-667-7638, or (214) 355-7401

E-Mail: OopsoftC&Airmail.net

Web site: Coming soon

[Return to Review](#)

[Return to Front Page](#)

