



No you are not imagining things, and yes I did skip a month on **UNDU**. It seems a certain hurricane called Bertha decided to make an appearance 20 miles to the west of our new house and things are generally a mess around here. Of course my Wife and I (being the brave individuals we are), ran away while all the locals laughed at us. Hey... we are from California, and this is our first hurricane, so I decided better safe than sorry. Anyway, we were fortunate and sustained no damage except for 37 huge lawn and leaf bags worth of fallen leaves and branches. You should see the calluses on my hands! Im a programmer, not a gardener! (*Jim... Im a doctor, not a bricklayer!*)

[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](#)

[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

[Return to Front Page](#)



## Where To Find UNDU

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

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



## Tips & Tricks

If you have some interesting tips and tricks, please share them! This section is one of the favorites in UNDU, so I am always on the lookout for more!

[How to Catch Keys](#)

[Tips for working with String Grids](#)

and something related - [Coloring Columns in a Grid](#)

[Solving a DLL problem](#)

[Reducing an Applications Memory Requirements](#)

[Return to Front Page](#)





## The Component Cookbook

This month we have an interesting component by Paul Harding on how to automatically dial your modem. Check it out!

[Creating an AutoDialer component](#)

Also, back in [Issue #12](#), I included a component by Lars Postuma on how to create an LED Light component. Unfortunately, I forgot to include the RES file necessary to compile the project. In case you would like to look it over again, here it is... along with the missing file.



[LED Light Source](#)

[LED RES File \(UUEncoded\)](#)

[Return to Front Page](#)



## UNDU - A Work In Progress

UNDU continues to be a work in progress. Since I have not done a widely circulated newsletter like this before UNDU, I am constantly trying new things. Sometimes they work out well, and sometimes they don't. All I can do is keep trying. Last issue, I commented that I was going to start doing 2 different versions of UNDU each month and that ultimately it would be just a Windows 95 version help file after a time. A few of you wrote to complain about this policy, and despite the fact that Win95 downloads were about 9 times the Win31 downloads I think it would be best to make sure that all readers can continue to read the issues despite the operating system they are working on. Therefore, I am going back to just having one version that is compatible with both versions. Besides, the things that made it a Win95-only issue (sound and video) weren't all that well received anyway.

The second item that is going to fade away into oblivion was the UNDU awards. I did get a flurry of votes after my desperate pleas for support, yet since then, I have had only 3 more votes. I will hold off on the UNDU votes for a while, and maybe I will bring it back later on. It served its purpose, but it was a lot of work to compile the votes. I think I would rather expend additional effort on other aspects of the issues.

**- Robert**

[Return to Front Page](#)



## UNDU Prizes!

For the entire history of the *Unofficial Newsletter of Delphi Users*, contributors have been very gracious about putting together articles for the benefit of the Delphi community without any thought of reward in return. Well, that's going to change!

Each month I am going to take the names of all the authors whose material I use in the issue and put them in a hat. Then I will draw one name at random to get a prize. To get this idea rolling a bit, there is going to be two drawings this next month for issue #16.

For issue #16, there will be two prizes: The first is a copy of **Delphi-In-Depth** (Osborne/McGraw-Hill), and the second is a copy of the **1995 Delphi Informant Works CD**.

For issues #17 and on, I will award other prizes, and I will mention what that prize is one issue in advance. A contributor can win more than one prize over multiple issues of UNDU.

These prizes are my way of returning a little thanks back to my faithful contributors. I do appreciate very much everyone's hard work and efforts at keeping UNDU a success!

**- Robert**

[Return to Front Page](#)



## The AutoDialer - Making a component that dials a modem

By Paul Harding - CIS: 100046,2604

A very few components are non-visual, and here is how to make one. This component will dial the modem, and its various fields and procedures will be manipulated when your application is running. There are some basic requirements to auto dialing the modem, and so the fields that live within the component need to be specified. Most obviously, we need a field that holds the phone number that the component is going to dial. This would be a string and it has to be possible to read and write the value. Also, there need to be fields that specify the com port that the modem is attached to, and the dial type (either pulse or tone). When a modem dials a telephone number, it stays connected, so if we need to lift the handset and use the connection for voice, we need a way of telling the modem to drop the line - this is a simple ATH (hangup) command. If we want the modem to stay connected, so we can send data, then there must be a way of telling the component not to issue the hangup command, so a boolean field is also required called FKeepConnected.

Let's start making the component. First, we need a unit, which we will save and call AUTODIAL.PAS. The actual type (TDialModem) is a very basic component, and does not need to inherit anything clever from existing classes so we can declare the TDialModem as a class of TComponent, the very basic non-visual component. The fields described above are then declared as being Private, and we need two procedures to be declared as public, namely the procedure that dials the phone, and the procedure that creates the component. Published declarations enable the component user to see in the object inspector the values of the various fields - these values can be set at design time using the object inspector, and then, more usefully, at run time by accessing the component's properties just like any other component using the dot notation. After each published property there is a **read & write** section which enables the component to get and set the values of the private fields.

The register procedure tells the component installation program where to put the AutoDial component on the palette, and if the page 'MyPage' does not exist, it will be created. (You could choose any page you like.). Here is the code for the whole component:

### [AutoDial Source Code](#)

In use, the programmer would read a phone number perhaps from an address book database and on a button click would set the telephone number (string) property of the autodial component, then issue an instruction to dial like this:

```
{set the phone number}
DialModem1.PhoneNumber := Edit1.Text;
{Dial Now!!}
DialModem1.DialModem;
```

It is also important to give the user a small dialog box where the com port and dial type settings can be stored, along with a setting for the delay length (in seconds). I have found it useful to keep these settings in an INI file, so that the dialog box need not be opened very often (if at all after the first time the modem is set up).

The component as it stands needs one more thing done to it - it needs a descent icon on the component

palette. If the Image Editor is run, the icon (which must be a 24x24 bitmap) can easily be created - I used a Wingdings font right parenthesis character: ) which comes out looking like a telephone handset:

)

Call the bitmap (by clicking the rename button) TAUTODIAL and save the bitmap file as AUTODIAL.DCR. Now when you install the component, it will have a better icon.

This basic autodial modem control shows how to create a simple, functional non visual component, and opens up possibilities for component writers to add facilities to the basic functions and fields provided. I have a very simple form that allows me to enter someone's name, and their phone number is looked up using my address book database. The form then dials their number, and I pickup the handset within my 3 seconds of allotted time and hey presto!

[Return to Component Cookbook](#)

[Return to Front Page](#)

## Source code for the AutoDial component

```
unit Autodial;

interface

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

type
  TDialModem = class(TComponent)
  private
    { Private declarations }
    FComPort: Integer;
    FToneDial: Boolean;
    FKeepConnected: Boolean;
    FPhoneNumber: String;
  protected
    { Protected declarations }
  public
    { Public declarations }
    procedure DialModem;
    constructor Create(AOwner: TComponent); override;
  published
    { Published declarations }
    property CommPort: Integer read FComPort write FComPort;
    property ToneDial: Boolean read FToneDial write FToneDial;
    property KeepConnected: Boolean read FKeepConnected write FKeepConnected;
    property PhoneNumber: String read FPhoneNumber write FPhonenumber;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Paul', [TDialModem]);
end;

constructor TDialModem.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  CommPort := 2;
  ToneDial := True;
end;
```

```

end;

procedure TDialModem.DialModem;
var
  ArrayToDial: Array [0..128] of Char;
  ArrayToHangup: Array[0..10] of Char;
  TelNumber: Array[0..100] of Char;
  CommsPort: Array[0..5] of Char;
  sFullCommsPort: String;
  iComID: Integer;
begin
  if Length(FPhoneNumber) = 0 then
    begin
      ShowMessage('Telephone number required. ');
      exit;
    end;
    {Make up the dial string like 'ATDT0932456789ATH'+<enter>}
    {decide if we start with ATDT or ATDP...}
    if FToneDial = True then
      StrCopy(ArrayToDial, 'ATDT')
    else
      StrCopy(ArrayToDial, 'ATDP');
      {add the phone number...}
      StrPCopy(TelNumber, FPhonenumber);
      StrCat(ArrayToDial, TelNumber);
      {add the hangup code if we do not want to keep a connection...}
      if FKeepconnected = False then StrCat(ArrayToDial, 'ATH');
      {add a CR or an <enter>...}
      StrCat(ArrayToDial, Chr(13));
      {generate COMx as a string...}
      sFullCommsPort := 'COM'+IntToStr(FComPort);
      {make a PChar out of that string...}
      strPCopy(CommsPort, sFullCommsPort);
      {try opening that comm port...}
      iComID := OpenComm(CommsPort, 32, 32);
      if iComID <= 0 then
        ShowMessage('Unable to open the port '+sFullCommsPort)
      else
        begin
          {now dial the number...}
          WriteComm(iComID, ArrayToDial, length(ArrayToDial));
          closeComm(iComID);
        end;
      end;
    end;
end.

```

[Go back to Article](#)

[Return to Component Cookbook](#)

[Return to Front Page](#)





## Core Concepts With Delphi - Parameter Passing

by Alan G. Labouseur

Greetings and good day! When we last gathered the topic was sorting. In the introduction of that discussion I promised that we'd talk about parameter passing this time. Being a man of my word, that's exactly what we shall do. And in doing so we'll be pulling in many of the ideas from all previous explorations into core concepts.

Recall that in the final demonstration program from last time we declared a global (remember variable scope?) array of integers (remember data types?) to sort with our QuickSort procedure (remember subprograms? Of course you do!). It was well and good, but there is room for improvement. After all, a computer program is never finished, it just comes due.

As it existed, our sort could operate only on the globally declared array. What if we had four arrays to sort? We'd make four global array declarations and then we'd have to write four separate sort routines. Ugh! There's a lot to be said for copy and paste, but it's just not going to do it for us here. Besides, what happens when we modify the sort routine? We'll have to remember to make exactly the same changes in all four subprograms. It would be a maintenance nightmare. And what if we had 4000 arrays to sort? What we really need is a general sort module that will operate on any array of integers. Guess what . . .

We can make the QuickSort more general by providing it with a parameter. Subprograms are "passed" parameters which provide them with data on which to operate. We have been passing parameters for a while now and never even noticed it. The WRITELN command, for example, takes a string parameter containing the data to print.

```
writeln(Hello, ball.);
```

In the above statement, the string Hello, ball. is the parameter being passed in to the (built-in) procedure called WRITELN. So too could our sort routine accept a parameter representing the array to sort.

Okay, that's it. Buh bye.

Just kidding. It's not quite that simple. Parameters are very important constructs in Delphi and provide a great deal of programming fire power. As such, there are many aspects about them which we need to explore.

Let's begin this exploration with a simple parameter-accepting subprogram, ShowName.

```
Program Param1;  
  
uses  
    WinCrt;  
  
Procedure ShowName (name: string);  
Begin  
    writeln(name);  
End; { Procedure ShowName }  
  
Begin
```

```

    ShowName('Alan');
End. { Program Param1 }

```

We pass the ShowName routine a parameter from the body of the program by including a string literal in parentheses. This string literal is said to be "passed in" to the procedure. This is called the "actual" parameter: the data being passed in to the routine. Up between the program header and the program BEGIN we have declared the ShowName routine. It is here, within the procedure declaration, that we specify the parameter we are expecting to receive, as well as its data type. This is called the "formal" parameter list, and it is extremely important. The compiler will be checking to see that all calls to ShowName send one and exactly one parameter to it and that the singular parameter is always of the string data type.

We can use variables as parameters too. Actually, we'll be doing this most of the time. Here's a slightly different version of the program:

```

Program Param2;

uses
  WinCrt;
var
  me: string;

Procedure ShowName(name: string);
Begin
  writeln(name);
End; { Procedure ShowName }

Begin
  me := 'Alan';
  ShowName(me);
End.

```

In this version we have a global string called "me" which is assigned a value in the main program. "Me" is then passed into the ShowName procedure and the value it was assigned in the main program is output. Why, you may ask, do I bother to pass "me" as a parameter when, according to Delphi's scope rules, it is visible to ShowName anyway? Two reasons. First, it's good programming technique. Second, it illustrates my point that parameters can be literal values or variables.

What happens if we alter the value of a parameter in a procedure? Check out this program:

```

Program Param3;

uses WinCrt;

var
  me: string;

Procedure ShowName(name: string);
Begin
  writeln('In procedure ShowName: ' + name);
  name := 'Cate';
  writeln('In procedure ShowName: ' + name);

```

```

End; { Procedure ShowName }

Begin
    me := 'Alan';
    ShowName(me);
    writeln('In the main program: ' + me);
End.

```

The output reads:

```

In procedure ShowName: Alan
In procedure ShowName: Cate
In the main program: Alan

```

The identifier "me" is assigned the value 'Alan' by the main program and is passed into the procedure ShowName as a string parameter called "name". "Name" now contains a copy of "me": the value 'Alan'. When "name" is printed, 'Alan' is the output. Now "name" is assigned the value "Cate" and printed again. "Cate" is the output. Since "name" was a copy of "me", the value of "me" remained the same, even though "name" was changed. This explains why, back in the main program, the output of "me" yields 'Alan'. Got it?

Good. This method of parameter passing is called pass by value. Received parameters are copies of the calling values. So when parameter variables are modified, their calling variables are not. But this will not help us sort 4000 integer arrays with a single procedure, since we'll want the arrays returned to us modified to be in sorted order. Luckily there is another way: pass by reference. Here is the previous program, modified only slightly:

```

Program Param3;

uses
    WinCrt;

var
    me: string;

Procedure ShowName(var name: string);
Begin
    writeln('In procedure ShowName: ' + name);
    name := 'Cate';
    writeln('In procedure ShowName: ' + name);
End; { Procedure ShowName }

Begin
    me := 'Alan';
    ShowName(me);
    writeln('In the main program: ' + me);
End.

```

I added only 4 characters to the program. On line 4, I added "var" and a space. This tells Delphi that I want the "name" parameter to be passed by reference, not by value. This means that when I modify "name" in the ShowName procedure, I am also modifying "me" in the main program. Actually, they

share the same space in memory. Both the identifiers "name" and "me" address, or point to, the same location in memory. Here is the output from this new version of program Param3:

```
In procedure ShowName: Alan
In procedure ShowName: Cate
In the main program: Cate
```

Pass by value and pass by reference are the two fundamental parameter passing schemes that Delphi offers. The Ada programming language defines parameters as in, out, and in/out. An "in" parameter is one whose value is used as input only, and never modified. An "out" parameter is one whose value is computed in the subprogram and returned. An "in/out" parameter is one where data is passed in, modified, and passed out. In Delphi terms, we can conceptualize a pass by reference parameter as an "in/out" parameter and a pass by value parameter is an "in" parameter. But an "in" parameter, as Ada sees it, cannot be modified. Delphi provides the constant parameter for this.

A constant parameter is a read-only variable that gets its value from the actual parameter passed in. Assignments to a constant parameter are not allowed, which also means that a constant parameter cannot be passed by reference to another subprogram. In other words, a constant parameter is passed to a procedure and acts like a constant within the body of the procedure. To implement one, use the key word "const" instead of "var" in the formal parameter list.

There are distinct advantages to using constant parameters. Runtime overhead is a bit less and some memory is saved. But by far the most important advantage is that the compiler will not let you accidentally modify it. Presumably, if you have declared a constant parameter, your intent is that it never be modified. So it's really nice when the compiler flags those logical errors that would otherwise go unnoticed until runtime. And even then, you may never become aware of the side effects until it's too late. That's the beauty of strongly typed languages: more errors discovered at compile time mean less errors discovered when the program is in use. (Another note on this galactically important principle: always set the project-compiler options to include all runtime error checking options.)

Finally, back to our sorting problem. If we add a pass by reference parameter to the QuickSort procedure we'll be able to have a single sorting routine capable of handling any number of integer arrays.

Parameter passing can be tricky, and as such hard to master. But the payoff is really worth the practice. And practice, as they say, makes perfect. It's one of the keys to good programming. I hope this has been helpful in providing some core information upon which we can build in the near future. I'll have a "putting it all together" article in an upcoming issue. Stay tuned. In the mean time, feel free, encouraged even, to e-mail me with any comments, questions, or suggestions. Thank you and goodnight.

## **About the Author**

Alan G. Labouseur is Vice President and co-owner of AlphaPoint Systems, Inc., a custom programming consultancy located in Brewster, NY. Alan has a Masters degree in Computer Science and has been developing custom software for ten years. He is currently working on Clipper and Delphi applications for clients in the NY-NJ-CT area. You can reach Alan through e-mail at [AGL007@IX.NETCOM.COM](mailto:AGL007@IX.NETCOM.COM) or 70312,2726 here on CompuServe.

[Return to Front Page](#)



## **Delphi Programmer's Book Shelf**

*by Harvey Kaye*

Borland's latest visual programming tool, Delphi 2.0, has been making a spectacular impression on the computer programming community. When the first version came out last year, it was quickly billed as the "Visual Basic killer." It has all the features of Visual Basic, but is not hampered by VB's annoying inability to compile true machine code. (VB produces p-code which executes 2 to 20 times slower than machine code, depending on the particular operation.)

With version 2, Borland enhanced Delphi with dozens of powerful new features that will make it the preferred choice for many Windows developers. For 32-bit database work (be it desktop or client-server) Delphi has all the tools you need to produce world-class applications. If you're into visual programming at all, I suggest you look at Borland's Delphi page on the Web at <http://www.borland.com> for more details about its capabilities.

I've spent a few weeks developing apps with Delphi 2.0, and I already appreciate its power and flexibility. However, everything has its price. In the case of Delphi, the learning curve is somewhat steeper than Visual Basic. Delphi 2.0 is a significantly larger product with many more bells and whistles than VB. (A typical installation takes 70 MB of disk space.) Although the visual placement and naming of objects is very similar to that of VB, Delphi insists that you spend considerably more attention to its variable types and syntax. Also, since Delphi is fully object-oriented like C++, more time is required to become familiar with its implementation of inheritance and encapsulation.

The on-line help in Delphi 2.0 takes up about 10 MB of space and is nicely cross-referenced and hyper-linked. Yet, the help files presume more than a passing knowledge of the product itself. Even though I used Object Pascal (Delphi's native dialect) with Turbo Pascal a few years back, I still felt lost. So, I looked for some third-party books to bring me up to speed. The good news is that there are over two dozen Delphi books on the market. The bad news is that no single volume stands out as a universal best choice. If you're going to work with Delphi, I recommend getting at least two or three of the following strong contenders. As I shall explain, the particular ones you should get will depend on your programming level and areas of interest.



### **Foundations of Delphi Development for Windows 95 by Tom Swan**

**IDG Books, December 1995 \$39.99.**

The title of Swan's book is misleading, because it is based on the 16-bit version of Delphi (Delphi 1.0). Nevertheless, this book earns my highest recommendation for those new to Delphi, even if you are using version 2.0. Why? The exposition is extremely clear and easy to follow. The numerous examples will get you up and running in no time. But most importantly, the companion CD contains a handy, hypertext version of the book. I found it very helpful to read the book on-line, switching back and forth to Delphi to try the step-by-step examples. Also, when I was stuck on my own projects, it was easier to search through Swan's book for an answer than to venture into Delphi's help system. I liked this book so much that I installed the hypertext version (all 5 MB) to my hard disk for quick access.



## **Teach Yourself Delphi 2 in 21 Days by Dan Osier et al**

***Borland Press/Sams Publishing***

**ISBN 0-672-30863-0 \$35.00**

This book provides a balanced introduction to Delphi 2, starting with the basics. The first 7 days cover the IDE (Integrated Development Environment), Object Pascal, GUI design, and the rudiments of object-oriented programming. The second 7 days cover the Visual Component Library, graphics, multimedia, file I/O, and databases in Delphi. The final 7 days cover Delphi's Interbase and Reportsmith, printing, OLE2, and advanced features such as creating your own dll's and components.

What I like about this book is that it is not all "Step 1, Step 2, ..." The authors give enough explanation of Delphi's operation to let you understand what you are doing. There is more "meat" in this 700-page book than in similar introductions. The book does not come with a code disk, but you can download the code examples from the Sams web site.



## **Database Developer's Guide with Delphi 2 by Ken Henderson**

***Borland Press/Sams Publishing***

**ISBN 0-672-30862-2 \$49.99**

Henderson's book focuses on using Delphi to build client/server database applications. It presumes a general acquaintance with Delphi, but starts at the beginning in its comprehensive discussion of database construction. In 850 pages, Henderson explains the necessary steps to creating your own commercial quality apps. It starts with SQL, works through Delphi's database component hierarchy, visual form inheritance, form linking, Interbase, and Reportsmith. There is extensive discussion of client-server connectivity issues, database drivers, and optimizing for fast performance. The CD-ROM includes all source code, sample applications, and demo software. This book is a great resource if you want to develop professional level database apps with Delphi.



## **Revolutionary Guide to Delphi 2 Programming by Brian Long, Bob Swart et al**

***Wrox Press Ltd.***

**ISBN 1-874416-67-2 \$49.95**

The Revolutionary Guide to Delphi 2 Programming is targeted at developers who have previous exposure to the basics of Delphi. It starts where the introductory books leave off and delves into object-oriented programming, classes, methods, and properties. It then moves on to code re-use and application design. There is an excellent chapter on debugging applications, followed by a thorough discussion of database development. The book then turns to creating Delphi components, dll's, experts, and property editors. Next are excellent chapters on using the Windows API with Delphi and interfacing with other applications. The book winds up with a detailed discussion of optimization techniques. Accompanying the book is a CD with the entire text in hypertext format, all source code, and a full-featured example database application.

I have mixed feelings about The Revolutionary Guide. The chapters were written by ten different authors, and the book lacks a sense of continuity. Furthermore, the chapters on component development left me confused whether the author was discussing Delphi 1.0 or Delphi 2.0. Overall, however, there is a great deal of useful material in this book and I'm sure that I'll be referring to it for my development projects.



## **Delphi How-To by Gary Frerking, Wayne Niddery, and Nathan Wallace**

**Waite Group Press**

**ISBN 1-57169-019-0 \$39.95**

The Waite Group Press has made quite a hit with its series of "How-To" books. This Delphi volume highlights a wide variety of problems facing the beginning programmer and presents clear and comprehensive solutions. The format is familiar by now: For example, "How do I create a hot spot in a picture?" First, the book steps you through a complete example solution. Then, the authors explain how it works in detail. Finally, comments about the solution provide insight into possible limitations of the method or how the solution can be extended to greater functionality. All of the source code is included on the CD, so you can cut and paste parts into your own projects as required. This 850 page book covers all aspects of Delphi and is a great help when you are stuck on a particular technique. Delphi How-To was written for Delphi 1.0, but 90% of the solutions work in Delphi 2.0 (A new version for Delphi 2.0 should be out in a few months.)



## **Delphi 2 Unleashed, Second Edition by Charlie Calvert**

**Borland Press/Sams Publishing**

**ISBN 0-672-30858-4 \$59.99**

Unleashed? Has Borland got a dog or cat inside the Delphi box in addition to the disks? Unusual title notwithstanding, this book sets something of a record with respect to its size, price, and comprehensive scope. It tips in at 1400 pages, costs almost \$60, and covers nearly everything advanced Delphi programmers would want to know. Charlie Calvert, an acknowledged Delphi expert who works on Borland's Developer Relations Team, has written a very authoritative reference on Delphi 2.0.

Despite the book's higher price, you get excellent value for your money. The companion CD contains not only a wealth of demos, source code, and Delphi components, but also the introductory material written for version 1 of the book. (The version 1 chapters are presented in Word for Windows format. If Calvert had included them in the printed version, the page count would have gone beyond 2,000!) I really like the goodies that come on the CD and think the book is an excellent reference. On the internet's Delphi chat areas, Calvert's book seems to be the favorite of advanced programmers, and I can see why.

### **About the Contributor**

Harvey Kaye is the author of "Decision Power" (Prentice Hall, 1992). A second edition of his book "Inside the Technical Consulting Business" was published by John Wiley & Sons in January 1994. He can be reached at 716-223-4502. 42 Bent Oak Trail, Fairport, NY 14450 Tel: (716)-223-4502 Email: harvkaye@cris.com

[Return to Front Page](#)



## Coloring Columns in Grid

by Dale L. Johnston - INTERNET: [dlj@micropro.dungeon.com](mailto:dlj@micropro.dungeon.com)

After using Delphi and the DBGrid I was unhappy with the color of the highlight bar for the selection. After a couple of hours I came up with this solution:

```
procedure TBrowForm.DrawDataCell(Sender: TObject; const Rect: TRect; Field: TField;
State: TGridDrawState);
```

```
begin
```

```
  {Are we displaying the selector bar?}
```

```
  if (gdSelected in State) then
```

```
    begin
```

```
      with (Sender as TDBGrid) do
```

```
        begin
```

```
          {change the bar color to white}
```

```
          Canvas.Brush.Color := clWhite;
```

```
          {make sure the font color is set to black}
```

```
          Canvas.Font.Color := clBlack;
```

```
        end;
```

```
      end;
```

```
      { If we want to set up enhancement, do it here. I want the font color }
```

```
      { to change to red if the value displayed is within 14 days of the }
```

```
      { current date. }
```

```
      if Field.FieldName = 'CloseOut' then
```

```
        if (Field as TDateField).Value <= (Date - 14) then
```

```
          with (Sender as TDBGrid) do Canvas.Font.Color := clRed;
```

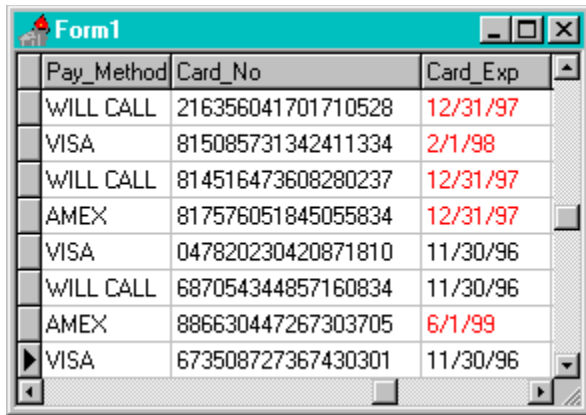
```
          { After the above conditions have been set, redraw the data cells }
```

```
          with (Sender as TDBGrid) do DefaultDrawDataCell(Rect, Field, State);
```

```
        end;
```

This illustration shows how the technique can applied. In this case, all cards that expire after the end of 1996 are highlighted.





Pay_Method	Card_No	Card_Exp
WILL CALL	216356041701710528	12/31/97
VISA	815085731342411334	2/1/98
WILL CALL	814516473608280237	12/31/97
AMEX	817576051845055834	12/31/97
VISA	047820230420871810	11/30/96
WILL CALL	687054344857160834	11/30/96
AMEX	886630447267303705	6/1/99
VISA	673508727367430301	11/30/96

[Return to Tips & Tricks](#)  
[Return to Front Page](#)

## Using TDatabase/TQuerys in a DLL

by *Brad Huggins* - *INTERNET:bhuggins@itctel.com*

with help from *Chris Kastin kasten@brookings.net*

We have solved a problem that many people seem to have (with some help from Borland's Tech Support and fellow employees).

Here's the problem:

We're trying to do a seemingly simple task.

Our 'main' application contains a TDatabase component.

We have Many DLL's with TQuerys and we want them to use that TDatabase component in the app.

TDatabase Properties:

KeepConnection = TRUE

Connected = TRUE

LoginPrompt = FALSE

The application connects to the DB automatically at start up with args or ini entries.

DLL's are dynamically linked (static or dynamic = same results).

Oracle db, SQL Links (32)

This almost works. One of the function arguments (in the DLL) is of type TDatabase (a pointer, right?). We then programmatically set the TQuery.DatabaseName to the TDatabase.DatabaseName in the DLL.

It then prompts us for name and password - WHY??!!

That TDatabase component is already active. We even put `in ShowMessage` to show us the Connected and other useful properties at the beginning of the DLL function. Everything is groovy until we open any datasets in the DLL (eg TQuery.Open). We are then prompted for the name and password!

Here's the [Solution](#) with a Code Sample

It turns out to be really simple, but it took us a lot of time to solve it.

[Return to Tips & Tricks](#)

[Return to Front Page](#)



**uses**

Windows, Messges, SysUtils, Classes, Graphics, Contrils,  
Forms, Dialogs, StdCtrls, DB, BDE;

**type**

```
TForm1 = class(TForm)
  dbOracle : TDatabase;
  Button1 : TButton;
  procedure Button1Click(Sender : TObject);
end;
```

**var**

```
Form1 : TForm;
```

```
procedure myDllCall(dbHandle : hdbidb); stdcall; external 'TESTDLL.DLL'
```

**implementation**

```
{$R *.DFM}
```

```
procedure TForm1.Button1Click(Sender : TObject);
begin
  dbOracle.Open;
  myDllCall(dbOracle.Handle);
end;
```

[Return to Article](#)

[Return to Tips & Tricks](#)

[Return to Front Page](#)



## Reducing Your Applications Memory Requirements

by Gordon J. Johnson - INTERNET: [SethToby@ix.netcom.com](mailto:SethToby@ix.netcom.com)

I've been trying to write a formless program, which I've done by putting my code in the main unit (the project source unit). The only problem is that even like this, a Delphi program takes up more than 1 megabyte of memory while it's running (according to Norton Utilities). A good way to cut down on the amount of memory used, *assuming your program does not use OLE*, is to use the command:

```
FreeLibrary(GetModuleHandle('OleAut32'));
```

This unloads OleAut32.dll and OLE32.dll, and it made my program use about one megabyte less memory than it used to.

[Return to Tips & Tricks](#)

[Return to Front Page](#)



## Catching Keys

by Yigal Ederly - CIS: 100264,644

A while ago, I was looking for a way to 'catch' some keys in a component I was deriving from TMemo. I wanted to use the Gray-Plus key for some internal function of this component. I found that if I just catch it in the KeyPress method, and set KEY := 0, it still feeds a '+' into the text.

When I looked at the VCL source code, I found that Delphi sends the WM\_KeyPress message to the TranslateMessage windows procedure, which puts the WM\_CHAR with '+' in the queue, before it sends the message to my component, and that is the reason I still get the WM\_CHAR.

I also found out that Delphi sends notification messages for any keyboard/mouse message, and if I catch these messages, and signal that I've handled them (by setting Msg.Result:=1), it enables me to cancel this default behavior.

These messages are CN\_xxx, and in my case, CN\_KEYDOWN (which is compatible with WM\_KEYDOWN).

I declared a message procedure for this CN\_KEYDOWN, check if it's VK\_ADD, do whatever I wanted to do with it, and set Message.Result:=1, thus effectively killing any further processing of this key. For example:

**type**

```
TOnKeyEvent = procedure (Sender:TObject; var Key:Word; Shift:TShiftState) of object;
```

```
TMyMemo = class (TMemo)
```

```
  private
```

```
    FOnKey : TOnKeyEvent;
```

```
  protected
```

```
    procedure CNKeyDown (var Msg:TWMKeyDown); message CN_KEYDOWN;
```

```
    procedure DoKey (var Key:Word; Shift:TShiftState); virtual;
```

```
  published
```

```
    property OnKey: TOnKeyEvent read FOnKey write FOnKey;
```

```
  end;
```

```
procedure TMyMemo.CNKeyDown (var Msg:TWMKeyDown);
```

```
var
```

```
  ShiftState: TShiftState;
```

```
  Key:Word;
```

```
begin
```

```
  with Msg do
```

```
  begin
```

```
    { Check the ShiftState, like delphi does while processing WMKeyDown }
```

```
    ShiftState := KeyDataToShiftState (KeyData);
```

```
    DoKey (CharCode, ShiftState);
```

```
    if CharCode = 0 then Result:=1;
```

```
  end;
```

```
end;
```

```
procedure TMyMemo.DoKey(var Key:Word; Shift:TShiftState);  
begin  
    if Assigned(FOnKey) then FOnKey(Self,Key,Shift);  
end;
```

If you use this memo, you have a new event called 'OnKey' that you can use to really capture keys and prevent any further processing of them. Note that in this way, you can also catch keys that are used as shortcut keys in your application's menu.

[Return to Tips & Tricks](#)

[Return to Front Page](#)

## Tips for Working With TStringGrid

by *Duncan Campbell* - INTERNET: [duncan@tvl.com](mailto:duncan@tvl.com)

Recently I have been doing a lot of work using the Delphi TStringGrid component. My App required that the user clicked on several different cells, and a component (TCombo, TEdit etc..) would position itself over the appropriate cell and allow the user to enter/choose some text. Then when the user moved to another cell, the text that they had just entered/chosen would be copied into that cell.

I wanted to "mask off" most of the grid so that if the user clicked, nothing would happen (i.e. the "clicked" cell would not appear to have focus). This proved initially difficult.

Which Event trapped the click/cell-focus-draw event? Was it "OnClick"?, "OnDrawCell"?, "OnSelectCell"? I just couldn't seem to find the answer in the on-line help.

My first approach was to check to see if the cell was valid, and then if not, copy a blank, borderless "hide"-TPanel over that cell. This seemed to work quite well, except for a small "quirk" of the StringGrid - when it lost focus, and I made the hide-panel invisible, the cell that I last clicked on was colored as if it had focus!

The solution seemed to be in re-drawing over the focused cell with a blank rectangle. On delving a bit deeper into the On-Line Help, I managed to come up with the following code to put into the "OnDrawCell" event:

```
{override the default cell-select color}  
If (gdSelected in State) then  
  with grdMyStringGrid do  
    begin  
      Canvas.Brush.Color := Color;  
      Canvas.FillRect(Rect);  
    end;
```

All this does is check if the current cell has focus, and then re-draws the cell in the grid's background color. Simple, and easy!

[Return to Tips & Tricks](#)

[Return to Front Page](#)





## UNDU Subscriber List

As the readership of UNDU continues to expand, I am discovering that older methods of distributing the newsletter are starting to have their drawbacks. As many of you may be aware, UNDU has an official web site now (<http://www.informant.com>) courtesy of Informant Communications, and this has greatly helped to improve UNDU's visibility.

From some of the feedback I receive regarding UNDU, one of the larger complaints is that readers will constantly hit the web site looking for the new issue, or go to the Delphi or Informant forums on CompuServe to see if the latest issue has been released. This seems like a lot of wasted effort to me, so I have decided that the time has come to make a *subscriber list*.

The subscriber list will simply be 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@cis.com`

That's all there is to it!

[Return to Front Page](#)

## LED Component

```
unit LedLight;
```

```
(*
```

```
  TLedLight by Lars Posthuma; December 26, 1995.
```

```
  Copyright 1995, Lars Posthuma.
```

```
  All rights reserved.
```

```
  This source code may be freely distributed and used. The author  
  accepts no responsibility for its use or misuse.
```

```
  No warranties whatsoever are offered for this unit.
```

```
  If you make any changes to this source code please inform me at:
```

```
  LPosthuma@COL.IB.COM.
```

```
*)
```

```
{ $R LedLight.res }
```

```
interface
```

```
uses
```

```
  WinTypes, WinProcs, Classes, Graphics, Controls, ExtCtrls, Forms, SysUtils;
```

```
Const
```

```
  MinLedSize = 8;
```

```
  MaxLedSize = 30;
```

```
type
```

```
  String5 = String[5];
```

```
  TLedLightOrientation = (LedHorizontal, LedVertical);
```

```
  TLedLightState       = (LedOff, LedOn);
```

```
  TLedLight = class(TCustomPanel)
```

```
    private
```

```
      { Private declarations }
```

```
      fBorderWidth : Integer;
```

```
      fOrientation : TLedLightOrientation;
```

```
      fState       : TLedLightState;
```

```
      fLedSize     : Integer;
```

```
      fLedSpacing  : Integer;
```

```
      fHeight      : Integer;
```

```
      fWidth       : Integer;
```

```
      fValueChange : TNotifyEvent;
```

```
      fCtl3D       : Boolean;
```

```
      procedure PaintLeds(Led1, Led2: String5);
```

```

procedure SetBorderWidth (value: Integer); virtual;
procedure SetBounds (Left,Top,fWidth,fHeight: integer); override;
procedure SetCtl3D(value : boolean); virtual;
procedure SetHeight(value: Integer); virtual;
procedure SetOrientation(value: TLedLightOrientation);
procedure SetLedSize(value: Integer);
procedure SetLedSpacing(value : Integer);
procedure SetState(value: TLedLightState); virtual;
procedure SetWidth(value: Integer); virtual;
protected
  { Protected declarations }
procedure Paint; override;
public
  { Public declarations }
constructor Create(AOwner: TComponent); override;
destructor Destroy; override;
procedure ChangeState; virtual;
published
  { Published declarations }
property Align;
property BevelInner;
property BevelOuter;
property BevelWidth;
property Cursor;
property Enabled;
property Color default clBtnFace;
property Height: Integer {Height of Outer Limits}
  read fHeight write SetHeight
  default 17;
property Width : Integer {Width of Outer Limits}
  read fWidth write SetWidth
  default 36;
property Orientation: TLedLightOrientation {}
  read fOrientation write SetOrientation
  default LedHorizontal;
property LedSize: Integer {True Size of Leds}
  read fLedSize write SetLedSize
  default 13;
property LedSpacing: Integer {Space between the Leds}
  read fLedSpacing write SetLedSpacing
  default 6;
property State: TLedLightState {Indicates Leds On or Off}
  read fState write SetState
  default LedOff;
property BorderStyle;
property BorderWidth: Integer
  read fBorderWidth write SetBorderWidth

```

```

        default 1;
    property Ctl3D: Boolean
        read fCtl3D write SetCtl3D
        default False;
    property OnValueChange: TNotifyEvent      {Userdefined Method}
        read fValueChange write fValueChange;
    property Visible;
    property Hint;
    property ParentShowHint;
    property ShowHint;
    property Tag;
    property OnClick;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDrag;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
end;

procedure Register;

implementation

constructor TLedLight.Create(AOwner: TComponent);
begin
    inherited Create(AOwner);
    Color      := clBtnFace;           {Set initial (default) values}
    fLedSize   := 13;
    fLedSpacing := 6;
    fHeight    := 17;
    fWidth     := 36;
    fBorderWidth:= 1;
    fOrientation:= LedHorizontal;
    fState     := LedOff;
    fCtl3D     := False;
    BevelInner := bvNone;
    BevelOuter := bvNone;
    Caption    := ' ';
end;

destructor TLedLight.Destroy;
begin
    inherited Destroy;
end;

procedure TLedLight.SetHeight(value: integer);

```

```

begin
  if value <> fHeight then begin
    fHeight:= value;
    if fHeight < MinLedSize + 4 then fHeight:= MinLedSize + 4;
    if fHeight > MaxLedSize + 4 then fHeight:= MaxLedSize + 4;
    fLedSize:= fHeight - 4;
    SetBounds(Left,Top,fWidth,fHeight);
    Invalidate;
  end
end;

procedure TLedLight.SetWidth(value: integer);
Var
  MyWidth: Integer;
begin
  if value <> fWidth then begin
    fWidth:= value;
    MyWidth:= (MinLedSize + 2) * 2 + LedSpacing;
    if fWidth < MyWidth then fWidth:= MyWidth;
    MyWidth:= (MaxLedSize + 2) * 2 + LedSpacing;
    if fWidth > MyWidth then fWidth:= MyWidth;
    fLedSize:= (fWidth - 4 - LedSpacing) div 2;
    SetBounds(Left,Top,fWidth,fHeight);
    Invalidate;
  end
end;

procedure TLedLight.SetBounds(Left,Top,fWidth,fHeight : integer);
begin
  case fOrientation of
    LedHorizontal: begin
      fWidth := (fLedSize + 2) * 2 + fLedSpacing;
      fHeight:= fLedSize + 4;
    end;
    LedVertical : begin
      fWidth := fLedSize + 4;
      fHeight:= (fLedSize + 2) * 2 + fLedSpacing;
    end;
  end;
end;
inherited SetBounds(Left,Top,fWidth,fHeight);
end;

procedure TLedLight.SetLedSize(value : Integer);
begin
  if value <> fLedSize then begin
    fLedSize:= value;
    if fLedSize < MinLedSize then fLedSize:= MinLedSize;
  end;
end;

```

```

    if fLedSize > MaxLedSize then fLedSize:= MaxLedSize;
    if LedSpacing > fLedSize then LedSpacing:= fLedSize;
    SetBounds(Left,Top,fWidth,fHeight);
    Invalidate;
end
end;

procedure TLedLight.SetLedSpacing(value : Integer);
begin
    if value <> fLedSpacing then begin
        fLedSpacing:= value;
        if fLedSpacing < 2 then fLedSpacing:= 2;
        if fLedSpacing > LedSize then fLedSpacing:= LedSize;
        if (fLedSpacing mod 2) <> 0 then Inc(fLedSpacing);
        SetBounds(Left,Top,fWidth,fHeight);
        Invalidate;
    end
end;

procedure TLedLight.SetOrientation(value : TLedLightOrientation);
begin
    if value <> fOrientation then begin
        fOrientation:= value;
        SetBounds(Left,Top,Height,Width);           {Swap Width/Height}
        Invalidate;
    end
end;

procedure TLedLight.SetBorderWidth(value : Integer);
begin
    if value <> fBorderWidth then begin
        fBorderWidth:= value;
        if fBorderWidth < 0 then fBorderWidth:= 1;
        if fBorderWidth > 2 then fBorderWidth:= 2;
        Invalidate;
    end
end;

procedure TLedLight.SetState(value : TLedLightState);
begin
    if value <> fState then begin
        fState:= value;
        if Assigned(fValueChange) then OnValueChange(Self);
        Invalidate;
    end
end;

```

```

procedure TLedLight.SetCtl3D (value : boolean);
begin
  if value <> fCtl3D then begin
    fCtl3D:= value;
    Invalidate;
  end
end;

procedure TLedLight.ChangeState;
begin
  if fState = LedOff
    then fState:= LedOn
    else fState:= LedOff;
  if Assigned(fValueChanged)                               {Call Userdefined Method}
    then OnValueChanged(Self);
  Invalidate;
end;

procedure TLedLight.Paint;
begin
  inherited Paint;
  with Canvas do begin
    Brush.Color:= Color;
    with ClientRect do begin
      if Ctl3D = True then begin                               {Give 3d look}
        Pen.Color:= clBtnShadow;
        MoveTo(Left + 1, Bottom - 3);
        LineTo(Left + 1, Top + 1);
        LineTo(Right - 2, Top + 1);
        MoveTo(Left, Bottom - 1);
        LineTo(Right-1, Bottom - 1);
        LineTo(Right-1, Top - 1)
      end;
      Case fState of
        LedOff: PaintLeds('Green','Gray');
        LedOn : PaintLeds('Gray','Red');
      end;
    end;
  end;
end;

procedure TLedLight.PaintLeds(Led1,Led2: String5);
var
  BMP1, BMP2 : TBitmap;
  TheLed      : Array[0..5] of Char;
  MyOffs      : Integer;
  DRect, SRect: TRect;

```

```

begin
  (* Create the Leds from Resource file.
     Dynamic creation by means of Old-fashioned API call *)
  StrPCopy(TheLed,Led1);
  BMP1      := TBitmap.Create;
  BMP1.Handle:= LoadBitmap(hInstance, TheLed);
  StrPCopy(TheLed,Led2);
  BMP2      := TBitmap.Create;
  BMP2.Handle:= LoadBitmap(hInstance, TheLed);
Try
  (* If BorderStyle is bsSingle the leds are somewhat shifted; to fix this
     "problem" paint the leds at a somewhat smaller offset *)
  MyOffs:= 2;
  if BorderStyle = bsSingle then Dec(MyOffs);
  (* Create Source Rectangle; used for both Leds *)
  SRect:= Rect(0,0,13,13);
  MyOffs:= 2;
  if BorderStyle = bsSingle then Dec(MyOffs);
  (* Create Destination Rectangle; used for drawing first Led *)
  DRect:= Rect(MyOffs,MyOffs,(MyOffs + LedSize),(MyOffs + LedSize));
  (* Draw first Led *)
  Canvas.BrushCopy(DRect,BMP1,SRect,clBtnFace);
  (* Create Destination Rectangle; used for drawing Second Led *)
  case fOrientation of
    LedHorizontal: DRect:= Rect((Width - 2 - LedSize),MyOffs,(Width - 2),(MyOffs +
LedSize));
    LedVertical  : DRect:= Rect(MyOffs,(MyOffs + LedSize + LedSpacing),(2 + LedSize),
(2 + (2 * LedSize) + LedSpacing));
  end;
  (* Draw second Led *)
  Canvas.BrushCopy(DRect,BMP2,SRect,clBtnFace);
Finally
  BMP1.Free;
  BMP2.Free;
end;
end;

procedure Register;
begin
  RegisterComponents('Lars', [TLedLight]);
end;

end.

```

[Return to The Component Cookbook](#)

[Return to Front Page](#)



 The Unofficial Newsletter of Delphi Users - Issue #15 - August 1996  
**LED Light Resource File**

```
begin 644 LEDLIGHT.RES
M_P(`1U)!60`P`/@$`H`#0`T`!`@`-`
M`0`@`@`("`(``"``(`@(``,`#`P`#`W`,`
M\,BD`/#*I@#N[NX`W=W=`,`S,S`"[N[L`JJJJ`)F9F0"(B(@`=W=W`&9F9@!5
M554`1$1$`# ,S,P`B(B(`$1$1`,`S_P"9__\`90__#/_P#_S/\`S,S_)G,
M_P!FS/\`,`\S_`# ,_P#_F?`\S)G_)F9_P!FF?`\ ,YG_`"9_P#_9O\`S&;_
M`)EF_P!F9O\`,`V;_`!F_P#_ ,_`S#/_`)DS_P!F,`\ ,S/_`S_P# ,`/\`
MF0#_`&8`_P`S`/\`_ ,`,`S_S`"9_\P`9O_ ,`#/_S``_\P`_S,`)G,S`!F
MS,P`,`\S,`# ,S`#_F<P`S)G,`)F9S`!FF<P`,`YG,``"9S`#_9LP`S&; ,`)EF
MS`!F9LP`,`V; ,`!FS`#_ ,\P`S#/,`)DSS`!F,\P`,`S/,``SS`#_ ,P`S`# ,
M`)D`S`!F`,`P`,`P# ,`/_F0# ,_YD`F?^9`&;_F0`S_YD``/^9`/_ ,F0# ,S)D`
MF<R9`&; ,F0`SS)D`,`R9`/^9F0# ,F9D`9IF9`# .9F0``F9D`_V:9`,`QFF0"9
M9ID`9F:9`#-FF0``9ID`_S.9`,`PSF0"9,YD`9C.9`# ,SF0`` ,YD`_P"9`,`P`
MF0"9`)D`9@"9`# ,`F0#_V8`S/]F`)G_9@!F_V8`,`_]F`#_9@#_S&8`S,QF
M`)G,9@!FS&8`,`\QF`# ,9@#_F68`S)EF`)F99@!FF68`,`YEF``"99@#_9F8`
MS&9F`)EF9@`S9F8`&9F`/\S9@# , ,V8`F3-F`&8S9@`S,V8`#-F`/\`9@# ,
M`&8`F0!F`&8`9@`S`&8`_`S`,`S_P"9_S ,`9O\`S#/_ ,P`_S ,`_`PS`,`S ,
M,P"9S# ,`9LPS`#/, ,P`S# ,`_YDS`,`R9,P"9F3 ,`9IDS`# .9,P`F3 ,`_V8S
M`,`QF,P"99C ,`9F8S`#-F,P`9C ,`_S,S`,`PS,P"9,S ,`9C,S``S,P#`_# ,`
MS`S`)D`,`P!F`# ,`,`P`S`,`S_`"9_P`9O``#/_`#_S``S,P`)G,``!F
MS```,`\P`/^9`# ,F0`F9D`&:9````.X`#=```S``+L``"J``
MF0``(@``!W``9@``%4``!$`` ,P``"(```1``#N``W0`` ,P`
M``"[``J@``)D``"(```=P``&8``!5``1``# ,``B``$0``
M[@``-T``# ,``NP``*H``"9``B``'<``!F``50``$0``S
M```\O`*2@H``@(```#``_``#``_``_\`_P``/\`_P#_P`_`_`<`
M!P?X^/CX^/@`!P<````!_CX^?X!_CX^<`````!_@`^?X!_@`^?X!P`
M`^?X^?X!_@`^?X^<``#X^?X!_@`^?X!_CX```^?X!_@`^?X!_@`
M^````/CX!_`^?X!_@`^/@``#X!_C_P?X!_@`^?X```^/@`_`!_@`
M^?X````?X^?`_`^?X^<````!^?X!_@`^?X!_@```!P?X^/@`
M^?X^/@`!P`<`<`!P?X^/CX^<`!P<``#_@`!'4D5%3@`P`/@$`H`
M#0`T`!`@`-`0`@`@`("`(``"``(`@(``,`#`P`#`W`,`
M@`(``"``(``"``(`@(``,`#`P`#`W`,`\,BD`/#*I@#N[NX`W=W=`,`S,
MS`"[N[L`JJJJ`)F9F0"(B(@`=W=W`&9F9@!5554`1$1$`# ,S,P`B(B(`$1$1
M`,`S_P"9__\`90__#/_P#_S/\`S,S_)G,_P!FS/\`,`\S_`# ,_P#_F?`\
MS)G_)F9_P!FF?`\ ,YG_`"9_P#_9O\`S&;_`)EF_P!F9O\`,`V;_`!F_P#_
M,`\`S#/_`)DS_P!F,`\ ,S/_`S_P# ,`/\`F0#_`&8`_P`S`/\`_ ,`,`S_
MS`"9_\P`9O_ ,`#/_S``_\P`_S,`)G,S`!FS,P`,`\S,`# ,S`#_F<P`S)G,
M`)F9S`!FF<P`,`YG,``"9S`#_9LP`S&; ,`)EFS`!F9LP`,`V; ,`!FS`#_ ,\P`
MS#/,`)DSS`!F,\P`,`S/,``SS`#_ ,P`S`# ,`)D`S`!F`,`P`,`P# ,`/_F0# ,
M_YD`F?^9`&;_F0`S_YD``/^9`/_ ,F0# ,S)D`F<R9`&; ,F0`SS)D`,`R9`/^9
MF0# ,F9D`9IF9`# .9F0``F9D`_V:9`,`QFF0"99ID`9F:9`#-FF0``9ID`_S.9
M`,`PSF0"9,YD`9C.9`# ,SF0`` ,YD`_P"9`,`P`F0"9`)D`9@"9`# ,`F0#_V8`
MS/]F`)G_9@!F_V8`,`_]F`#_9@#_S&8`S,QF`)G,9@!FS&8`,`\QF`# ,9@#_
MF68`S)EF`)F99@!FF68`,`YEF``"99@#_9F8`S&9F`)EF9@`S9F8`&9F`/\S
```

M9@#,,V8`F3-F`&8S9@`S,V8``#-F`/\`9@#`,`&8`F0!F`&8`9@`S`&8`\_\_\S  
M`,`S`,`P`"9\_S`,`9O\S`#/\_`,`P`\_S`,`\_`PS`,`S`,`P`"9S#`,`9LPS`#//`,`P`S#`,`  
M\_YDS`,`R9,P`"9F3`,`9IDS`#.#9,P`F3`,`\_V8S`,`QF,P`"99C`,`9F8S`#-F,P`  
M9C`,`\_S,S`,`PS,P`"9,S`,`9C,S``S,P#`\_`#`,`S``S`)D`,`P!F`#`,`P`S`,`S\_  
M`"9\_P`"9O`\`#/\_`#`\_S``S,P``)G`,`!FS`,`\P`"/^9`#`,`F0`F9D`  
M`&:9`````.X```#=```S````+L````"J````F0````(@````!W````9@````%4`  
M`!\$`````,`P````"(```1``#N````W0```,`P````"〔````J@````)D````"(```  
M=P````&8````!5````1````#`,```B````\$0````[@````-T````#`,```NP````\*H`  
M`"9````B````!<``!F````50````\$0````S````\/\_`\*2@H`"``@(```#`\_`  
M`#`\_````\_\_\`\_P````/\`\_P#`\_P````<`!P?X^/CX^/@`!P<````!`\_CX`  
M`OH``^@+X^<`!````!`\_CZ`OH``^@+Z`OKX!P````?X`OH``^@+Z`OH``^<`#X`  
M`OH``^@+Z`OH``^@+X````^/H``^@+Z`OH``^@+Z`OH``^@/@"^O`Z`OH``^@+Z`O@`  
M`#X^@+`\_H``^@+Z`OKX````^+Z`\_``^@+Z`OH``^````?X`OK`\_Z`OH`  
M^<````!^/H``^@+Z`OH``^O@`!P?X^+Z`OH``^/@`!P````<`!P?X^/CX`  
M^<`!P<``#`\_@!2140`,`#X!``\*````T````-````0`(```#0````  
M```````````````\$``````````(````(````"``@``"````@``"``"``("``#`  
MP`,`P-S``/`#(I`#PRJ8`[N[N`-W=W0#`,`S,P`N[N[`\*JJJ@`"9F9D`B(B(`'=W  
M=P!F9F8`5555`\$1\$1``S,S`,`(B(B`!\$1\$0#`,`\_`\F?`\_`&`,`\_P`S`\_`\`\_S`  
M`,`S`,`P`"9S/\`9LS`\_`#/\_`,`P`S/\`\_YG`\_`,`R9\_P`"9F?`\`9IG`\_`#.#9\_P`F?`\`  
M\_V;`,`\_`,`QF\_P`"99O`\`9F;`,`#-F\_P`"9O`\`\_S/\_`,`PS\_P`"9`,`\_`\`9C/\_`#`,`S\_P`  
M`,`\_`\`S`#`\_`)D`\_P!F`/\`,`P#`\_/\_`S`#`,`\_P`F?`,`&`,`S``S`\_P`\_/\_`,`/\_`,`  
MS`"9S,P`9LS`,`#/\_`,`S``S,P`\_YG`,`R9S`"9F<P`9IG`,`#.#9S``F<P`\_V;`,`  
M`,`QFS`"99LP`9F;`,`#-FS``9LP`\_S/\_`,`PSS`"9,\P`9C/`,`#`,`SS```,`\P`  
M\_P#`,`P`S`"9`,`P`9@#`,`#`,`S`#`\_YD`S/^9`)G\_F0!F\_YD`,`\_`^9`#`\_F0#`\_`  
MS)D`S,R9`)G,F0!FS)D`,`\R9`#`,`F0#`\_F9D`S)F9`&:9F0`SF9D``)F9`/]F`  
MF0#`,`9ID`F6:9`&9FF0`S9ID``&:9`/\SF0#`,`YD`F3.9`&8SF0`S,YD`#.#9`  
M`/\`F0#`,`)D`F0`"9`&8`F0`S`)D`\_]F`,`S\_9@`"9\_V8`9O]F`#/\_`9@`\_`V8`  
M`\_QF`,`S\_9@`"9S&8`9LQF`#/\_`,`9@`\_`S&8`\_YEF`,`R99@`"9F68`9IEF`#.#99@`  
MF68`\_V9F`,`QF9@`"99F8`,`V9F``!F9@#`,`V8`S#-F`)DS9@!F,V8`,`S-F``S`  
M9@#`\_`&8`S`!F`)D`9@!F`&8`,`P!F`/\_`,`P#`,`S`,`F?`S`&`,`\_P`S`S`,`\_`\S`  
M`/\_`,`P#`,`S#`,`F<PS`&`,`P`SS#`,``,`PS`/^9,P#`,`F3`,`F9DS`&:9,P`SF3`,`  
M`)DS`/]F,P#`,`9C`,`F68S`&9F,P`S9C`,`&8S`/\S,P#`,`S`,`F3,S`&8S,P`  
M,S`,`\_P`S`,`P`,`P`"9`#`,`9@`S`#`,`P#`,`\_P`F?`\`&`,`\_``S\_P`\_`\_P`\_`,`S`,`  
M`"9S``9LP`#/\_`,`#`\_F0`S)D``)F9`!FF0``#N``W0```,`P``"〔  
M````J@````)D````"(```=P````&8````!5````1````#`,```B````\$0````[@``  
M`-T````#`,```NP````\*H````"9````B````!<``!F````50````\$0````S````  
M(@````!\$``.X```#=```S````+L````"J````F0````(@````!W````9@````%4`  
M`!\$`````,`P````/#[`\_P`"DH\*``@(```````\_P`\_P````/\_`/\``#`\_`\`\_`\_`  
M`/\_`\_P`!`!P<`^/CX^/CX!P<````!P?X^`Y`?D!^/@`!P````?X^O`Y`?D!  
M^O`Y`^<````!^`Y`?D!^O`Y`?@`!````!^`Y`?D!^O`Y`?D!^````/CY`?D!  
M^O`Y`?D!^?@``#X`?G`^O`Y`?D!^O`X````^/D!`\_Y`?D!^O`Y`^````/@`!  
M^?`\_D!^O`Y`?@`!````!^`Y`^`\_`^O`Y`?@`!````!`\_CY`?D!^O`Y`?GX!P`  
M`<`^/@`!^O`Y`?CX!P<````!P<`^/CX^/@`!P<````\_P`\_P`\$`,`!PP````  
M#`"``&``\$=205D`#0`"``\*``\$=2145.`L`@`#`@!2140```````````  
#````

end

[Return to The Component Cookbook](#)

[Return to Front Page](#)

