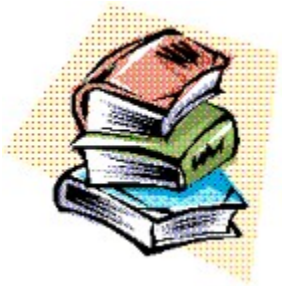
Okay folks… you now have some new competition. Elliot Glen Vivrette was born on November 24th, 1997 and weighed in at a healthy 10 lbs 6 oz. He will shortly be starting some private tutoring in Delphi programming by his dad. Just as soon as we teach him to walk, talk, and speak of course…

This is going to be a bit of a small issue, due to some of the additional responsibilities I am facing towards the end of this year. I am working to simplify things quite a bit and am looking forward to streamlining the production of future issues of UNDU. I will get my act together again, so please be patient!

As a look ahead to next month, I will be having a full review of the latest version of **WinHelp Office** by *Blue Sky Software*. I put together this latest issue with it and I must say it is **very** impressive. They have very smoothly integrated their RoboHelp tool into Word 97 and it is great for making stand-alone help files, HTML docs, Help files for applications, or whatever your heart desires. It is going to save me a **load** of time. Look forward to a full look at this latest version next issue. I am like a kid in a candy store with this one!

The randomly selected winner of the UNDU prize this month is Alan Lloyd for his beginners article on Pointers . His prize is a copy of **Kick-Ass Java Programming** by Tonny Espeset (Coriolis Group Books).

- Robert



Index of Past Issues

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

Issue #4 - May 24, 1995

Issue #5 - June 26, 1995

Issue #6 - July 25, 1995

Issue #7 - August 31, 1995

Issue #8 - October 10, 1995

Issue #9 - November 9, 1995

Issue #10 - December 12, 1995

Issue #11 - January 18th, 1996

Issue #12 - Feburary 23rd, 1996

Issue #13 - May 1st, 1996

Issue #14 - June 1st, 1996

Issue #15 - August 1st, 1996

Issue #16 - September 1st, 1996

Issue #17 - October 1st, 1996

Issue #18 - November 1st, 1996

Issue #19 - January 1st, 1997

Issue #20 - March 1st, 1997

Issue #21 - May 1st, 1997

Issue #22 - June, 1997

Delphi 2 & 3 on the Same Computer

Issue #23 - September, 1997

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/undu/index.htm`. This site houses all the issues in both HTML and Windows HLP format. Click on the large icons for the HTML versions and the small red book icons for downloadable Windows HLP files.
2. *Borland's* Delphi forum on CompuServe (**GO DELPHI**) in the "Delphi IDE" file section. This forum will only hold the issues in Windows HLP format.

Tips & Tricks

Jens Fudge brings us a quick tip that I think is quite handy. Many times when accessing Master-Detail tables that involve queries to get the detail records, you may have some un-needed processing overhead as you scroll through master records. Jens discusses this and offers a solution in his tip on Speeding up Master-Detail Tables.

Alan Lloyd also brings us two interesting discussions this month. First, he has a beginners discussion on pointers in his article on Beginners Block – Intro to Pointers . Then, he carries on the discussion about pointers by explaining how Delphi free's objects. Check this second part out in the article on Nil Pointers – How Delphi Free's Objects .

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 reader's email addresses and when a new issue is released, I will fire off a batch mailing to notify everyone that it is available.

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

That's all there is to it!

Return to Front Page

Delphi Users Groups

If you run a Delphi Users Group that is not on this list, please feel free to send me an email with the essentials. Please include basic information about where and when the group meets, and particularly any web links you may have to support the group. I am going to keep the info to a minimum so I can include in each issue (or maybe every other issue). I have provided a sample form to help you provide the right information.

Arizona

Tucson Delphi Users Group - TDug

California

Orange County Delphi Users Group

North Bay Delphi SIG

Canada

Metro Halifax Delphi Developers Group

The Toronto Area Delphi Developers Association

Colorado

Western Slope Visual Developers Group

Iowa

Central Iowa Delphi Users Group

Massachusetts

The Delphi Developers Group of Greater Boston

Michigan

Royal Oak Delphi Users Group

North Carolina

Research Triangle Park Delphi Users Group

Ohio

Delphi SIG of the Greater Cleveland PC Users Group

Pennsylvania

Pittsburgh Area Delphi Users Group

Return to Front Page

Group Name:	Orange County Delphi Users Group
Person(s) In Charge:	Maurie Seymour
Meeting Date/Time:	Second Saturday of each month at 1:00pm
Meeting Location:	1520 Brookhollow Drive, Suite 38, Santa Ana, CA
Dues:	???
Email Address:	delphibegin.sig@ocipug.org or mauries@pacbell.net
Phone Number:	(714) 633-2914
Internet URL:	http://www.ocipug.org
Description:	

Return to Delphi Users Groups

Group Name:	Pittsburgh Area Delphi Users Group
Person(s) In Charge:	Kirk A. Farra
Meeting Date/Time:	Second Thursday of the month at 7:00pm
Meeting Location:	Mastech Inc.
Dues:	None
Email Address:	
Phone Number:	(412) 452-6121
Internet URL:	http://www.nauticom.net/www/kfarra
Description:	We cover Delphi and any related products/tools. Members are exposed to many 3rd party components and have a local support network for Q/A. Members get to review many products before they are released and get information on local job opportunities and contracting positions.

Return to Delphi Users Groups

Group Name:	Central Iowa Delphi Users Group
Person(s) In Charge:	David W. Body
Meeting Date/Time:	Third Thursday of every month at 5:30pm
Meeting Location:	Offices of Iowa Bankers Association, Liberty Building, 418 6th Ave. Suite 430, Des Moines, Iowa.
Dues:	None
Email Address:	
Phone Number:	(515) 984-6243
Internet URL:	http://www.bigcreek.com/delphi
Description:	

Return to Delphi Users Groups

Group Name:	Royal Oak Delphi Users Group
Person(s) In Charge:	???
Meeting Date/Time:	First Thursday of each month at 7:00pm
Meeting Location:	William Beaumont Hospital (Lower Level), Royal Oak, MI. Classrooms in AB-West building.
Dues:	None
Email Address:	???
Phone Number:	(810) 551-5000
Internet URL:	http://ourworld.compuserve.com/homepages/STNICOL/
Description:	Constructing DLL's, SQL Server 6.5, CGI, Email enabled databases, data collection alternatives, VCL parties, book review night, etc.

Return to Delphi Users Groups

Group Name:	Metro Halifax Delphi Developers Group
Person(s) In Charge:	Dave Hackett
Meeting Date/Time:	First Tuesday of each month from 7:00pm to 9:00pm
Meeting Location:	Maritime Life Business Park
Dues:	None
Email Address:	71650.2646@compuserve.com
Phone Number:	
Internet URL:	
Description:	News, Q&A, Presentations

Return to Delphi Users Groups

Group Name: North Bay Delphi Special Interest Group
Person(s) In Charge: Bjarne Winkler
Meeting Date/Time: Second and Fourth Wednesday of each month from 7:00pm to 9:00pm
Meeting Location:
Dues:
Email Address: nts4618086@aol.com
Phone Number:
Internet URL:
Description: Announcements, Q&A, Special topics each meeting.

Return to Delphi Users Groups

Group Name:	Research Triangle Park Delphi Users Group
Person(s) In Charge:	Mark Hutchinson
Meeting Date/Time:	Third Wednesday of each month, 5pm
Meeting Location:	New Horizons Computer Learning Center - Willow Oak Bldg, Suite 116, 4625 Creekstone Drive, Durham, NC
Dues:	
Email Address:	aikimark@aol.com
Phone Number:	
Internet URL:	http://www.geocities.com/CapeCanaveral/Lab/1185/rtpdig.html
Description:	

Return to Delphi Users Groups

Group Name:	The Toronto Area Delphi Developers Association (TaDDA!)
Person(s) In Charge:	Henri Fournier (President), Robert Kozak (Membership Director), Ted Nye (Executive Director)
Meeting Date/Time:	First Tuesday of each month
Meeting Location:	North Youk Public Library, Toronto, Ontario, Canada
Dues:	$60.00/year
Email Address:	membership@TaDDA.COM
Phone Number:	(416) 366-7150
Internet URL:	http://www.tadda.com
Description:	3 Hours. Sessions by well known Delphi experts such as Mark Miller, Ray Konopka, Charlie Calvert, etc. and in depth sessions on all aspects of Delphi for New to Advanced users.

Return to Delphi Users Groups

Group Name: The Delphi Developers Group of Greater Boston
Person(s) In Charge: Al Reinhart, DisCom Systems
Meeting Date/Time: Third Monday of each month, 7pm
Meeting Location: 10 New England Executive Park, Burlington MA
Dues:
Email Address: reinhart@discom.com
Phone Number: 508-869-6417
Internet URL:
Description:

Return to Delphi Users Groups

Group Name:	Delphi SIG of the Greater Cleveland PC Users Group
Person(s) In Charge:	Bill Querry
Meeting Date/Time:	Fourth Wednesday of each month, 6:30pm - 8:30 pm
Meeting Location:	2747 SOM Center Rd, Willoughby OH 44094
Dues:	Free for Meetings. $25/year for GCPCUG
Email Address:	bq@bigfoot.com
Phone Number:	440-944-9980
Internet URL:	www.gcpcug.org
Description:	Demos of some feature of Delphi or 3rd Party Add-On. General discussion of user's current projects & problems. Talk about latest magazines/book releases.

Return to Delphi Users Groups

Group Name:	Western Slope Visual Developers Group
Person(s) In Charge:	Shane Holmes, Software Engineer, ViA Software Services
Meeting Date/Time:	Last Wednesday of each month, 6:30pm
Meeting Location:	ViA Software Services, 743 Horizon Court, Suite 368, Grand Junction, CO 81506
Dues:	Free
Email Address:	sholmes@viagj.com
Phone Number:	970-257-7010, 970-248-9733
Internet URL:	http://www.gj.net/~hap
Description:	

Return to Delphi Users Groups

Group Name:	Tucson Delphi Users Group - TDug
Person(s) In Charge:	Matthew J. Brock, Oscar Speed
Meeting Date/Time:	Third Saturday of each month, 9:00am-10:00am
Meeting Location:	DRA Software Training Center, 3434 E. 22nd St. Suite 100, Tucson, AZ
Dues:	Free
Email Address:	mjb@azstarnet.com
Phone Number:	520-571-7203
Internet URL:	
Description:	One or more presentations, Q&A, Product Reviews

Return to Delphi Users Groups

Form For Users Groups

If your Delphi Users Group is not mentioned in the list, please take a moment to email me the particulars about your group. To make the information as uniform as possible, please *cut and paste this form* into an email and send it to me at RobertV@compuserve.com. Please only send one if you are in charge of the group… I don't want to have to wade through 30-40 copies for each group… Thanks!

Group Name:

Person(s) In Charge:

Meeting Date/Time:

Meeting Location (include City & State):

Dues:

Email Address:

Phone Number:

Internet URL:

Brief description of meeting format:

Return to Delphi Users Groups

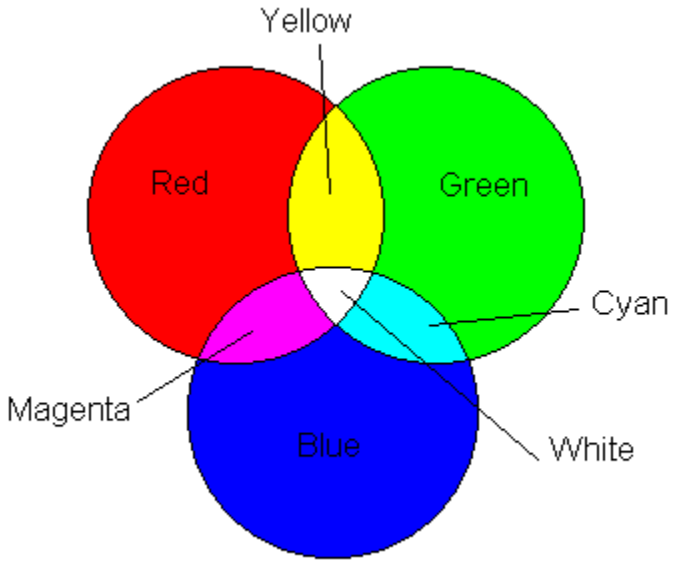
Return to Front Page

RGB and HSL Colour Models

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

RGB Model

The colour model most famliar to Delphi programmers will be the RGB colour model where a colour is represented by intensity values of red, green and blue light within a colour. This is not suprising as the RGB model is normally used for devices that emit light. It is also very easy to understand, In Delphi (and Windows in general) valid red, green and blue light intensities are represented using 0 to indicate a minimum value and 255 to indicate a maximum intensity. This illustration shows how the three primary colours can be combined to produce four additional colours:



Note that for display devices, the colour black results from the absence of colour. The following are the eight colours and their associated RGB values:

Colour	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Cyan	0	255	255
Magenta	255	0	255
Yellow	255	255	0
Black	0	0	0
White	255	255	255

Delphi defines and uses *TColor* as a means of storing the the complete colour information in a single 32

bit value. The internal storage, in hexadecimal, is $00BBGGRR - the low-order byte is used to store the red intensity, the second byte contains the green and the third the blue. The high order byte is usually zero but has its uses. Delphi can also represent some TColor colours with negative values but this specialist use I shall ignore here.

You can use the *RGB* function to combine three colour intensities to make a *TColor* and use the *GetRValue*, *GetGValue* and *GetBValue* functions to extract the intensity values from a TColor.

The HSL Colour Model

The HSL colour model is very different. It represents colours based on a scale of 0 to 1 using three parameters:

- Hue - a measure of the colour tint (red, green etc) present
- Saturation - a measure of the amout of colour present (a saturation of zero is a total absence of colour (black, grey, white), a saturation of 1 is a totally pure colour tint)
- Luminosity (or lightness) - the brightness of a colour (a luminosity of 0 is black, and 1 is white, between 0 and 1 are shades of grey or colour. A luminosity of 0.5 is used for generating a pure colour.)

You can see HSL colours in use in the common colour pick dialog box where the scale 0 to 1 is replaced by 0 to 240. But I define the scales as 0 to 1 above as the universal representation as using the 0 to 240 scale is handy as only three bytes are needed to represent the values.

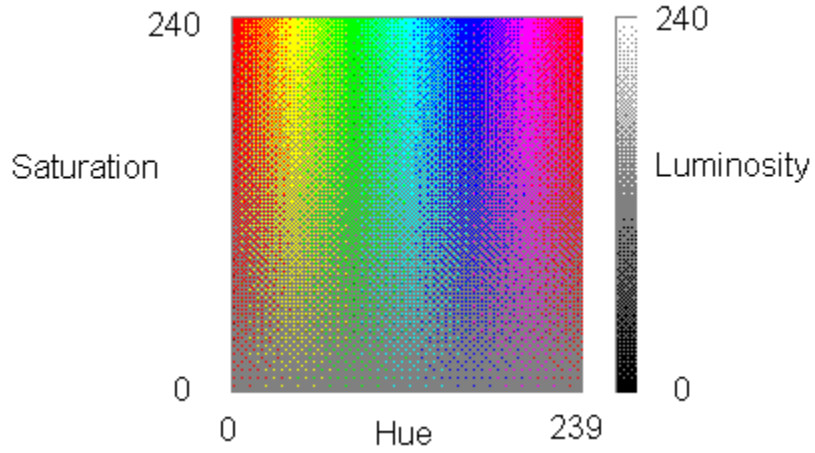
The eight common colour values are now (on the 0 to 240 scale):

Colour	**Hue**	**Sat.**	**Lum.**
Red	0	240	120
Green	40	240	120
Blue	80	240	120
Cyan	120	240	120
Magenta	160	240	120
Yellow	200	240	120
Black	*	0	0
White	*	0	240

* - undefined - not needed

Note that saturation and luminosity are unchanged as the "colour" changes, and that hue is not used for black or white, and finally that zero saturation is the absence of colour.

The following illustration shows the colour spectrum control and the luminosity slide control that appears in the common colour dialog box (with the screen in 256 colour mode). The spectrum control shows the colours available at a luminosity of 120.



The RGB model scores on it's simplicity, so what are the advantages of the HSL colour model? I think there are several:

- You can generate grey scales using only one parameter - the luminosity when saturation is set to 0.
- You can vary the colour by varying the hue alone such that the brightness remains unchanged
- You can fade or darken several colours, or whole bitmaps, such that the lightness (or darkness) stay in step

I suppose it comes down to that the HSL model is easier to use visually because it suits the eye, whereas the RGB model is easier to use in programming. What lead me to need to use the HSL model was the last point above. I wanted to draw an overlay image over a coloured bitmap. But the bitmap was too bright to see the overlayed lines properly without using very thick lines. But by fading the colours towards white (by uniformally increasing the luminosity of all of the colours that made up the bitmap's palette), the black lines of the overlay became much clearer.

So what's the catch? The big catch is that Windows does not include any routines to convert between the RGB and HSL colour models. So I include with this article four functions to convert between the models. The interface looks like this:

```
var
  HSLRange : integer = 240;

// convert a HSL value into a RGB in a TColor
// HSL values are 0.0 to 1.0
function HSLtoRGB (H, S, L: double): TColor;

// convert a HSL value into a RGB in a TColor
// SL values are 0 to the HSLRange variable
// H value is to HSLRange-1
function HSLRangeToRGB (H, S, L : integer): TColor;

// convert a RGB value (as TColor) into HSL
// HSL values are 0.0 to 1.0
procedure RGBtoHSL (RGB: TColor; var H, S, L : double);

// convert a RGB value (as TColor) into HSL
// SL values are 0 to the HSLRange variable
// H value is to HSLRange-1
procedure RGBtoHSLRange (RGB: TColor; var H, S, L : integer);
```

Also included is a simple form which illustrates a rudimentary colour picker, to show how easy it is to

include a HSL colour picker with your applications. Hope you find this useful!

Cheers!

Grahame

Source Code for HSL1

Source Code for HSLUtils

Form Definition for HSL1

References

- WIN32.HLP - search on "HSL"
- http://www.r2m.com/win-developer-faq/graphics/8.html
- *Fundamentals of Interactive Computer Graphics* by Foley and van Dam

Return to Front Page

Source Code for HSL1.PAS

```
unit HSL1;

// primative HSL colour picker

interface

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

type
  THSLDemoForm = class(TForm)
    HSLImage: TImage;
    HueTrackBar: TTrackBar;
    SatTrackBar: TTrackBar;
    Bevel1: TBevel;
    ColourSwatch: TShape;
    Bevel2: TBevel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    LumTrackBar: TTrackBar;
    Bevel3: TBevel;
    ReportListView: TListView;
    Bevel4: TBevel;
    UseLumBtn: TBitBtn;
    Lum120Btn: TBitBtn;
    procedure HSLMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
    procedure FormCreate(Sender: TObject);
    procedure RedoSwatch(Sender: TObject);
    procedure Lum120BtnClick(Sender: TObject);
    procedure HSLImageClick(Sender: TObject);
    procedure HSLMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure HSLMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure UseLumBtnClick(Sender: TObject);
  private
    FSaturation,
    FHue : integer;
    FMode256,
    FTracking : boolean;
  public
  end;

var
  HSLDemoForm: THSLDemoForm;

implementation

{$R *.DFM}

uses
  HSLUtils;
```

```
procedure THSLDemoForm.FormCreate(Sender: TObject);
var
  DC : hDC;
begin
  DC := GetDC(0);
  try
    FMode256 := GetDeviceCaps(DC, BITSPIXEL) = 8
  finally
    ReleaseDC (0,DC)
  end;

  Lum120Btn.Click;
  UseLumBtn.Click
end;

procedure THSLDemoForm.RedoSwatch(Sender: TObject);
var
  Hue,
  Saturation,
  Luminosity : byte;
  Colour : TColor;
begin
  Hue := HueTrackBar.Position;
  Saturation := 240 - SatTrackBar.Position;
  Luminosity := 240 - LumTrackBar.Position;
  Colour := HSLRangetoRGB (Hue, Saturation, Luminosity);
  ColourSwatch.Brush.Color := Colour;

  with ReportListView do
  begin
    Items[0].SubItems[0] := IntToStr (GetRValue (Colour));
    Items[1].SubItems[0] := IntToStr (GetGValue (Colour));
    Items[2].SubItems[0] := IntToStr (GetBValue (Colour));
    Items[4].SubItems[0] := IntToStr (Hue);
    Items[5].SubItems[0] := IntToStr (Saturation);
    Items[6].SubItems[0] := IntToStr (Luminosity)
  end
end;

procedure THSLDemoForm.Lum120BtnClick(Sender: TObject);
begin
  LumTrackBar.Position := 120;
  RedoSwatch (Self)
end;

procedure THSLDemoForm.HSLMouseMove(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
begin
  FSaturation := Y;
  FHue := X;
  if FTracking then
    HSLImageClick (Self)
end;

procedure THSLDemoForm.HSLImageClick(Sender: TObject);
begin
  HueTrackBar.Position := FHue;
  SatTrackBar.Position := FSaturation;
  RedoSwatch (Self)
end;

procedure THSLDemoForm.HSLMouseDown(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
```

```
begin
  FTracking := true
end;

procedure THSLDemoForm.HSLMouseUp(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
begin
  FTracking := false
end;

procedure THSLDemoForm.UseLumBtnClick(Sender: TObject);
const
  Box256 = 4;
var
  Hue,
  Sat : integer;
  HSLBitmap : TBitmap;
begin
  Screen.Cursor := crHourglass;
  try
    HSLBitmap := TBitmap.Create;
    with HSLBitmap do
    begin
      Width := 239;
      Height := 240;

      if FMode256 then
        for Hue := 0 to Width div Box256 do
          for Sat := 0 to Height div Box256 do
          begin
            Canvas.Brush.Color := HSLRangeToRGB (Hue*Box256, 240-Sat*Box256, 240 -
LumTrackBar.Position);
            Canvas.FillRect (Rect(Hue*Box256, Sat*Box256, Hue*Box256+Box256,
Sat*Box256+Box256))
          end
      else
        for Hue := 0 to 239 do
          for Sat := 0 to 240 do
            Canvas.Pixels [Hue, 240-Sat] := HSLRangeToRGB (Hue, Sat, 240 -
LumTrackBar.Position)
    end;
    HSLImage.Picture.Bitmap := HSLBitmap
  finally
    Screen.Cursor := crDefault
  end
end;

end.
```

Return to Article

Return to Front Page

Source Code for HSLUtils.PAS

```
//---------------------------------------------------------------------------
//
// HSL - RGB colour model conversions
//
// These four functions can be used to convert between the RGB and HSL colour
// models.  RGB values are represented using the 0-255 Windows convention and
// always encapsulated in a TColor 32 bit value.  HSL values are available as
// either 0 to 1 floating point (double) values or as a 0 to a defined integer
// value.  The colour common dialog box uses 0 to 240 by example.
//
// The code is based on that found (in C) on:
//
//   http:/www.r2m.com/win-developer-faq/graphics/8.html
//
// Grahame Marsh 12 October 1997
//
// Freeware - you get it for free, I take nothing, I make no promises!
//
// Please feel free to contact me: grahame.s.marsh@corp.courtaulds.co.uk
//
// Revison History:
//    Version 1.00 - initial release  12-10-1997
//
//---------------------------------------------------------------------------

unit HSLUtils;

interface

uses
  Windows, Graphics;

var
  HSLRange : integer = 240;

// convert a HSL value into a RGB in a TColor
// HSL values are 0.0 to 1.0 double
function HSLtoRGB (H, S, L: double): TColor;

// convert a HSL value into a RGB in a TColor
// SL values are 0 to the HSLRange variable
// H value is to HSLRange-1
function HSLRangeToRGB (H, S, L : integer): TColor;

// convert a RGB value (as TColor) into HSL
// HSL values are 0.0 to 1.0 double
procedure RGBtoHSL (RGB: TColor; var H, S, L : double);

// convert a RGB value (as TColor) into HSL
// SL values are 0 to the HSLRange variable
// H value is to HSLRange-1
procedure RGBtoHSLRange (RGB: TColor; var H, S, L : integer);

implementation

function HSLtoRGB (H, S, L: double): TColor;
var
  M1,
  M2: double;
```

```
  function HueToColourValue (Hue: double) : byte;
  var
    V : double;
  begin
    if Hue < 0 then
      Hue := Hue + 1
    else
      if Hue > 1 then
        Hue := Hue - 1;

    if 6 * Hue < 1 then
      V := M1 + (M2 - M1) * Hue * 6
    else
      if 2 * Hue < 1 then
        V := M2
      else
        if 3 * Hue < 2 then
          V := M1 + (M2 - M1) * (2/3 - Hue) * 6
        else
          V := M1;
    Result := round (255 * V)
  end;

var
  R,
  G,
  B: byte;
begin
  if S = 0 then
  begin
    R := round (255 * L);
    G := R;
    B := R
  end else begin
    if L <= 0.5 then
      M2 := L * (1 + S)
    else
      M2 := L + S - L * S;
    M1 := 2 * L - M2;
    R := HueToColourValue (H + 1/3);
    G := HueToColourValue (H);
    B := HueToColourValue (H - 1/3)
  end;

  Result := RGB (R, G, B)
end;

function HSLRangeToRGB (H, S, L : integer): TColor;
begin
  Result := HSLToRGB (H / (HSLRange-1), S / HSLRange, L / HSLRange)
end;

// Convert RGB value (0-255 range) into HSL value (0-1 values)

procedure RGBtoHSL (RGB: TColor; var H, S, L : double);

  function Max (a, b : double): double;
  begin
    if a > b then
      Result := a
    else
      Result := b
```

```
  end;

  function Min (a, b : double): double;
  begin
    if a < b then
      Result := a
    else
      Result := b
  end;

var
  R,
  G,
  B,
  D,
  Cmax,
  Cmin: double;

begin
  R := GetRValue (RGB) / 255;
  G := GetGValue (RGB) / 255;
  B := GetBValue (RGB) / 255;
  Cmax := Max (R, Max (G, B));
  Cmin := Min (R, Min (G, B));

// calculate luminosity
  L := (Cmax + Cmin) / 2;

  if Cmax = Cmin then  // it's grey
  begin
    H := 0; // it's actually undefined
    S := 0
  end else begin
    D := Cmax - Cmin;

// calculate Saturation
    if L < 0.5 then
      S := D / (Cmax + Cmin)
    else
      S := D / (2 - Cmax - Cmin);

// calculate Hue
    if R = Cmax then
      H := (G - B) / D
    else
      if G = Cmax then
        H  := 2 + (B - R) /D
      else
        H := 4 + (R - G) / D;

    H := H / 6;
    if H < 0 then
      H := H + 1
  end
end;

procedure RGBtoHSLRange (RGB: TColor; var H, S, L : integer);
var
  Hd,
  Sd,
  Ld: double;
begin
  RGBtoHSL (RGB, Hd, Sd, Ld);
```

```
    H := round (Hd * (HSLRange-1));
    S := round (Sd * HSLRange);
    L := round (Ld * HSLRange);
  end;

  end.
```

Return to Article

Return to Front Page

Form Definition for HSL1.DFM

```
object HSLDemoForm: THSLDemoForm
  Left = 340
  Top = 225
  BorderStyle = bsDialog
  Caption = 'HSL colour picker demo'
  ClientHeight = 384
  ClientWidth = 593
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object Bevel3: TBevel
    Left = 8
    Top = 8
    Width = 417
    Height = 361
  end
  object Bevel4: TBevel
    Left = 432
    Top = 8
    Width = 145
    Height = 233
  end
  object Bevel1: TBevel
    Left = 16
    Top = 32
    Width = 257
    Height = 257
  end
  object Bevel2: TBevel
    Left = 464
    Top = 16
    Width = 81
    Height = 81
  end
  object HSLImage: TImage
    Left = 24
    Top = 40
    Width = 240
    Height = 241
    Cursor = crCross
    OnClick = HSLImageClick
    OnMouseDown = HSLMouseDown
    OnMouseMove = HSLMouseMove
    OnMouseUp = HSLMouseUp
  end
  object ColourSwatch: TShape
    Left = 472
    Top = 24
    Width = 65
    Height = 65
  end
  object Label1: TLabel
    Left = 264
    Top = 16
```

```
  Width = 60
  Height = 16
  Caption = 'Saturation'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentFont = False
end
object Label3: TLabel
  Left = 328
  Top = 272
  Width = 6
  Height = 13
  Caption = '0'
end
object Label4: TLabel
  Left = 26
  Top = 344
  Width = 6
  Height = 13
  Caption = '0'
end
object Label5: TLabel
  Left = 256
  Top = 344
  Width = 18
  Height = 13
  Caption = '239'
end
object Label6: TLabel
  Left = 128
  Top = 344
  Width = 25
  Height = 16
  Caption = 'Hue'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentFont = False
end
object Label2: TLabel
  Left = 328
  Top = 40
  Width = 18
  Height = 13
  Caption = '240'
end
object Label7: TLabel
  Left = 352
  Top = 16
  Width = 63
  Height = 16
  Caption = 'Luminosity'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentFont = False
```

```
end
object SatTrackBar: TTrackBar
  Left = 280
  Top = 32
  Width = 45
  Height = 257
  Max = 240
  Orientation = trVertical
  Frequency = 10
  Position = 0
  SelEnd = 0
  SelStart = 0
  TabOrder = 1
  TickMarks = tmBottomRight
  TickStyle = tsAuto
  OnChange = RedoSwatch
end
object HueTrackBar: TTrackBar
  Left = 16
  Top = 296
  Width = 257
  Height = 45
  Max = 239
  Orientation = trHorizontal
  PageSize = 1
  Frequency = 10
  Position = 0
  SelEnd = 0
  SelStart = 0
  TabOrder = 0
  TickMarks = tmBottomRight
  TickStyle = tsAuto
  OnChange = RedoSwatch
end
object LumTrackBar: TTrackBar
  Left = 360
  Top = 32
  Width = 45
  Height = 257
  Max = 240
  Orientation = trVertical
  Frequency = 10
  Position = 120
  SelEnd = 0
  SelStart = 0
  TabOrder = 2
  TickMarks = tmTopLeft
  TickStyle = tsAuto
  OnChange = RedoSwatch
end
object ReportListView: TListView
  Left = 440
  Top = 112
  Width = 129
  Height = 121
  Color = clBtnFace
  Columns = <
    item
      Caption = 'Parameter'
      Width = 75
    end
    item
      Caption = 'Value'
```

```
      end>
    Items.Data = {
      CA0000000700000000000000FFFFFFFFFFFFFFFF0100000000000000003526564
      0130000000000FFFFFFFFFFFFFFFF0100000000000000005477265656E01300000
      0000FFFFFFFFFFFFFFFF0100000000000000004426C7565013000000000FFFFFF
      FFFFFFFFFF000000000000000000000000000FFFFFFFFFFFFFFFF010000000000
      000003487565013000000000FFFFFFFFFFFFFFFF0100000000000000A536174
      75726174696F6E0130000000000FFFFFFFFFFFFFFFF0100000000000000A4C75
      6D696E6F736974790130}
    TabOrder = 3
    ViewStyle = vsReport
  end
  object UseLumBtn: TBitBtn
    Left = 336
    Top = 304
    Width = 75
    Height = 25
    Caption = 'Apply Lum.'
    TabOrder = 4
    OnClick = UseLumBtnClick
  end
  object Lum120Btn: TBitBtn
    Left = 336
    Top = 336
    Width = 75
    Height = 25
    Caption = 'Lum. = 120'
    TabOrder = 5
    OnClick = Lum120BtnClick
  end
end
```

Return to Article

Return to Front Page

More on Moving from VB to Delphi

by Michael Cone - mikecone@advancedtubular.com

This is an article for VB programmers who have just "signed on with a tour" for Delphi. If you and Delphi are already real cozy, then this article isn't really for you. If you, however, find that the Delphi learning curve is looking steeper all the time, then I invite you to read on.

Like many Delphi users, I am a total convert from Visual Basic. Some of my older applications are still written in VB, so I must use it, at times, to maintain code. However, I avoid VB like the plague with all new projects. Experiencing Delphi's robustness, speed, and power for the first time created an emotion of being freed from prison. I had wasted too much time attempting to coax VB to run large projects. Delphi needs no such coaxing. Its performance was so superior to VB that I wrote to the Delphi team praising them for their excellent effort in turning out such a great product. That was with Delphi 1. Now, with Delphi 3 on my computer, my praise is still the same, in spite of some of its relatively minor shortcomings.

As a former VB programmer, I assure you this: You are indeed on the right track with Delphi. It is incredibly powerful, and yet, if you want it to, it takes care of many of the minutely detailed programming tasks automatically. (You just read my nutshell explanation of "RAD" – short for Rapid Application Development.) I don't say this as a warning, but as something to look forward to: It takes a long time for average programmers (like me) to plummet the depth of its riches. For example, a comforting thought is that most of the Delphi development environment is actually written using Delphi! That is a true testimony of its power. (VB is absolutely not written in VB. Microsoft knows better than attempting something like that.) The fact is, that it is really, really hard for a VB programmer to find limits in Delphi. (C programmers can find a few limitations in Delphi, but they are WAY beyond the average VB programmer's ability. And even so, enterprising Delphi-supporting companies always find ways around the obscure way-out problems anyway.)

With this in mind, these are some foundational tips for former VB programmers who have fallen in love with Delphi.

TIP ONE:

Do not hope to rely on a VB to Delphi Translator

That sounds pretty dreary, but it's actually good advice. I have read comments about VB to Delphi translators, and how they could be useful to VB programmers changing to Delphi. My company has one of them that we paid a lot for, and it is gathering dust on our shelves. It creates code that is strewn with extra support code in order to make our old VB code run in Delphi. The converted code is something akin to a Frankenstein. It is an ugly patch-work, and will never be as good as clean Pascal. I'm not going to name the translator software, because the resulting code is not the fault of the translator. In fact, I am amazed at the amount of effort the VB to Delphi translators must go through to perform this feat. The problem is that the traditional Visual Basic way of thinking, especially through VB version 3, is much different than ANY version of Delphi.

So, my recommendation is that it is far better to re-write your code in Delphi Pascal than to attempt to use a translator to do it for you. Not only is it cleaner, but I am convinced that Delphi programmers save thousands of future hours by completely re-writing code that is robust and finely tuned Pascal, not Pascal that hangs on to Visual Basic paradigms.

TIP TWO:

Make good use of free technical documents and source code snippets on the Internet

There are probably thousands of Delphi programmers who share really good tips and examples for free on the Internet. (Yes, The Unofficial Newsletter of Delphi Users is very good example of this kind of incredible value.) Search for "Delphi" (or "Borland Delphi) using a search engine like Yahoo, and you will begin a trail of finding plenty of source code showing how to do practically whatever is needed to be done. Learning from example snippets of source code is CRITICAL to understanding Delphi - one step at time. With all of this shared knowledge, new Delphi programmers can find solutions to almost any programming problem we want to solve.

TIP THREE:

Take the time to get used to thinking about programming from the perspective of Delphi objects.

Delphi allows you to program similarly to the traditional VB style of code, but its whole structure is built around OBJECTS. I know that many VB programmers, if they are honest, may ask "What in the world is an "object" anyway?!". Objects are confusing, frustrating, and even a little scary at first for some VB programmers, but the rewards of understanding and using them are incredible. Be comforted if you are struggling with the concept of objects. Many professional programmers also struggle with the concept at first. In fact, it was using Delphi that the definition and use of objects finally became clear to me a few years ago.

Objects are a bit like really fancy VB TYPE structures. If you've ever used a TYPE structure, then you know that they combine a whole bunch of variables under a single new variable name. (If you don't know what a TYPE structure is you're probably already confused. Send me an e-mail, and I'll teach you what a BASIC "TYPE" is.) Well, objects are WAY more powerful than VB TYPE structures, because they not only contain multiple variables, but they also contain functions and procedures (procedures are like VB SUBs). So objects don't only store variables, they can even do things with the variables, like perform calculations, and store the results – all inside the object.

Because of this, some really useful thing happens with Delphi objects. For example, all of this internal capability that you program into an object follows it wherever it goes. If you take the source code for an independent object, and copy it from one program to another, it is relatively easy to get it to work properly in the new program. A well written object frees you from thinking about the ramification of all those GLOBAL and SHARED variables that were intertwined in your old VB code. With objects, all you have to remember is that "all that stuff is already inside it, so I'm not going to worry about it." If you ever do copy an object as I described above, then you've will have practiced the famous idea of "re-use" that gets bantered about so often in programming circles. Remember that the example that I describe here is only a single example of the benefits of objects. There are other benefits to objects that some programmers may consider even more important than easy re-use. But what I've given you here should serve as an incentive to get going with objects.

"How," you may ask, "is the best way to learn how to use objects?" First, I suggest that you learn the basic structure of an object by examining someone else's simple working object (per TIP TWO above). The structure not only defines what the object does through internal functions and procedures, but also controls the variable scope rules of an object ("scope" means "what part of the program sees each particular variable"). It is very important, for example, to understand what the "Private" versus the "Public" sections of a Delphi object do. Variables, functions, and procedures defined under the "Private" section are only visible to the source code that is part of the object itself, never to the outside. Functions and procedures under the "Public" section, however, can be accessed by source code outside the object. Second, I suggest that you take the example object and experiment with it by making changes to it following the structure of what the programmer has already done. For example, add a variable under the Private section, then try to access it inside one of the object's procedures or functions.

FINALLY

Of course, there are many other basic tips that could be covered for VB programmers. For example, A basic discussion of variable scope rules with the VB programmer in mind would be helpful. Or how about a discussion of "what in the world is that variable in that parameter line?" It is normal for VB programmers to have questions about these and other Delphi topics at first. So be persistent - Delphi is worth the effort!

Return to Front Page

Delphi and the Turn of the New Century

by Kevin S. Gallagher

Pharmaceutical perceptional example:

"A drug was issued with an expiration date of "00". An elderly woman who needed the drug interpreted the "00" as "invalid" and decided not to take the drug. Her condition deteriorated until the situation was identified and rectified. The case is still pending." (Source: GartnerGroup)

This is not an isolated incident nor is this a worst case scenario, it doesn't take a lot to show what will happen if an application is not geared to handle dealing with multiple centuries. The question is, will the applications I work with be ready, and how about other programs which handle my credit cards and other bills!!! Well the top candidates for failing to work with more then one century are applications that reside on large monolithic mainframes, AS400s, or have to depend on them. How did we all get into such a pickle? There are many reasons, just search the web and you will see things like: "My application will not be around at the turn of the century", "It take to much of a computers resources to use four digits to store the century and year" or one that haunts even Microsoft, "Downwards compatibility". So what is a person to do to resolve the situation, there is no one answer because there are so many reasons to what caused the problem on any one given application.

For those of us who are writing new application and have selected Delphi for the language of choice, well you are in luck! Unlike Microsoft Visual Basic which handles dates fairly well Delphi excels with working with dates. The Delphi help says that it can handle dates in the range of 1899 to 2099 and Visual Basic's range is 1899 to 2030. I am sure Microsoft will argue the point as always, but that does not help programmers who need to project into the future and end up with a language barrier.

The hardest part of working with dates in Delphi is knowing were to get help, what routines are available and what are things to watch out for. Other visual languages are no different in the above statement, they just have less things available. So once you know the basics the rest is easy. Do not fear January 3, 2000 coming, instead sit back and watch the poor souls who are not as fortunate as us Delphi programmers!

The first thing to understand about dates in Delphi is that date/time information is stored in a float format (X.Y) were the X is the date information and Y is the time information. You would think it would be a pain in the butt to retrieve date/time information from such a format, but as you might guess Delphi provides us with methods to extract the needed information. Delphi provides TDateTime type for working with dates, for example if you want to know the current date we would use the *Date* function which returns TDateTime type and would be converted to a string using DateToStr as shown below:

```
MessageDlg('Today is ' + DateToStr(Date),mtInformation,[mbOk],0);
```

To extract any part of the date (Month,Day,Year) then there is a procedure called DeCodeDate

```
var
  TheMonth,TheDay,TheYear:word;
begin
  DecodeDate(Date,TheYear,TheMonth,TheDay);

  Label1.Caption := IntToStr(TheMonth) + '/' + IntToStr(TheDay) +
IntToStr(TheYear);
```

If you look at the declaration for DecodeDate as shown below:

```
procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);
```

Notice that the second, third and fourth parameters are passed by reference as stated with the key word "var", this allows the procedure to alter there values and deliver to us broken segments of the date past to it. Once the procedure returns these values you can change them and "encode" them back into a suitable date format using a function called EncodeDate which returns as no surprise a TDateTime variable.

```
function EncodeDate(Year, Month, Day: Word): TDateTime;
```

An example might be, your boss says that the company is extending the amount of time the a customer can take to pay bills at this time of year, so you might code something that gets the current month from the a field in the customer table and increment the month. The example below contains less code then needed for the requirement above, but shows how to use DecodeDate to gather a date from a table and EncodeDate to place the date back into the same field with the month incremented by one month. Also note that in real life we need to take into consideration that if the month was incremented and the day becomes invalid because the next month has less days in it we need to code some logical statements to prevent this, the required logic does not exist in this example:

```
procedure Tform1.cmdTraverseClick(Sender: TObject);
var
  Month,Day,Year:word;
begin

  while not Table1.Eof do begin
    Table1.Edit;
    DecodeDate(Table1.FieldByName('TheDate').AsDateTime,Year,Month,Day);
    if Month <> 12 then
      Inc(Month)
    else
      Month := 1;

      try
        Table1.FieldByName('TheDate').AsDateTime :=
          EncodeDate(Year,Month,Day);
        Table1.Post;
      except on E:EConvertError do ShowMessage('Error adding month');
      end;
    Table1.Next;
  end;
end;
```

Well I lied about incrementing months, Delphi has a function called *IncMonth* which handles the problem were the next month has less days in it then the month before we add a month to it. Just goes to show you the power of Delphi!

```
// Example of IncMonth:
Table1.FieldByName('TheDate').AsDateTime :=
        IncMonth(Table1.FieldByName('TheDate').AsDateTime,1);
```

if the date in 'TheDate' field was 03/31/2078, after using IncMonth the field would contain 04/30/2078. If we had simply used Inc(Month,1) an exception would have been generated.

When working with dates retrieved from databases there is always a possibility that the field you want to get a date from has no value stored in it. In the old days for languages such as dBase, Fox Pro and Clipper and empty date would be " / / " and you could check for it by using a function called Empty(), in Delphi an empty date shows up as "12/31/1899". To validate we have a proper date you can code a routine (shown below) which can do the validation for us:

```
function IsValidDate(ADate: TDateTime): Boolean;
var
```

```
  Year, Month, Day: Word;
 begin
  try
    DecodeDate(ADate, Year, Month, Day);
    Result := Trunc(ADate) > 0;
  except
    Result := False;
  end;
end;

if not IsValidDate(Table1.FieldByName('Expires').AsDateTime) then
```

Since we are on the topic of dissecting dates into separate pieces it is good to know that even though Delphi always returns the century and year in from a date you may not always be able to see the century part. This is due to a setting under MS-Windows95 called "Regional Settings" which we have direct access to under Control Panel, on a tabbed page called "Dates". Here you will find a setting called "Short Date Format" which tells Windows how to display dates i.e. M/d/yy, MM/dd/yyyy. If the end users computer is setup with the setting that only has two Y's in it then only the year is shown and not the century! This is a minor nascence which will mostly cause your forms to display with less information and under some circumstances not allow the user to distinguish between two different centuries when they need to make visual comparisons on date fields.

One method to insure that the century is shown in your applications is to change the settings via your setup program through the system registry. The only problem is that the user can change the setting through Control Panel because they like a particular view of dates. A much better approach is to have your application change the setting at program startup and then restoring the old setting when the application terminates. For example, place code in the main forms OnActivate event to query/change the Short Date Format, save the old setting. In the OnClose event of the main form retrieve the old setting and restore it.

To accomplish this task there are several variables in SysUtils

```
DateSeparator: Char;
ShortDateFormat: string;
LongDateFormat: string;
ShortMonthNames: array[1..12] of string;
LongMonthNames: array[1..12] of string;
ShortDayNames: array[1..7] of string;
LongDayNames: array[1..7] of string;
```

which give us access to information taken from the system registry and used in your applications. Also the "application" object has a property called UpdateFormatSettings which is used to either allow the current application to see changes that may occur in Control Panel during program execution or to ignore changes. Below shows the basic logic to control the Short Date Format as described above:

```
procedure Tform1.FormActivate(Sender: TObject);
begin
  SetCentury(True);
end;
```

In another global unit:

```
procedure SetCentury(L:boolean);
var
  S,Temp:String;
begin
  S := DateSeparator;
  Temp := 'yy';
  if L then Temp := 'yyyy';
  SetShortDateFormat('MM' + S + 'dd' + S + Temp);
end;
```

In the OnActivate event a routine called *SetCentury* is called with a boolean value of True which toggles how the century setting is to be set i.e. show century by passing true or pass false to not show the century. Note that in SetCentury I ask Delphi to give us the current date separator so that when the Short Date Format is set we use the prefer separator of the person using the computer. Next there is the actual routine to set the Short Date:

```
function SetShortDateFormat(S:String): boolean;
var
  DefLCID: LCID;
  Buffer: array[0..255] of char;
begin
  Application.UpdateFormatSettings := True;

  StrPCopy(Buffer,S);
  DefLCID := GetThreadLocale;

  Result := False;

  if SetLocaleInfo(DefLCID,LOCALE_SSHORTDATE,Buffer) then begin
    { If the setting has been updated then Buffer (same as S)
      is placed into the variable ShortDateFormat. The application
      will now recognize the new setting.  }
    ShortDateFormat := Buffer;
    Result := True;
  end;
  Application.UpdateFormatSettings := False;
end;
```

If you are not well versed in Delphi then this would not be an easy routine to write. There are several API calls which are used, and in case you have not noticed most API calls are not documented with Delphi programmers in mind.

As mentioned above we now know that Delphi stores various registry settings for us, but what if you need to get this information on your own? Below shows how to get the ShortDateFormat directly from Windows:

```
function GetShortDateFormat: String;
var
  L: Integer;
  DefLCID: LCID;
  Buffer: array[0..255] of char;
begin
  Result := '';

  DefLCID := GetThreadLocale;
  L := GetLocaleInfo(DefLCID,LOCALE_SSHORTDATE,Buffer,SizeOf(Buffer));

  if L > 0 then
    SetString(Result,Buffer, L -1);
end;
```

The most important piece of information is the second parameter which specifies the setting to get. To learn about other constants, go into Delphi help and search in the index for LOCALE, this will show all the other available constants for regional settings. If you want to know how easy Delphi made getting information then look at GetDateFormat

```
int GetDateFormat(

    LCID  Locale,       // locale for which date is to be formatted
    DWORD  dwFlags,     // flags specifying function options
    CONST SYSTEMTIME *  lpDate,      // date to be formatted
    LPCTSTR  lpFormat,  // date format string
    LPTSTR  lpDateStr,  // buffer for storing formatted string
    int  cchDate        // size of buffer
```

```
    );
```

not a pleasant sight! Lastly on Short Date Format, if you need to quickly determine if the century is to be shown then use

```
if Pos('YYYY', AnsiUpperCase(ShortDateFormat)) > 0 then
```

if Pos returns 0 then we immediately know that the century will not display.

Closing up on the settings in SysUtils, some are stored in arrays and can be accessed by referencing them by their index, i.e. ShortMonthNames[1] returns the short name for the first month of the year. The example below fills four list boxes with Short/Long month names and also Short and Long day names according to the locale settings.

```
for i := 1 to 12 do begin
    ListBox1.Items.Add(ShortMonthNames[i]);
    ListBox2.Items.Add(LongMonthNames[i]);
    if i <= 7 then begin
      ListBox3.Items.Add(ShortDayNames[i]);
      ListBox4.Items.Add(LongDayNames[i]);
    end;
  end;
```

List of other useful date routines

```
Date                    Returns the current date.
DateTimeToStr           Converts a value from time format to a string.
DateTimeToString        Converts a value from time format to a string.
DateToStr               Converts a value from date format to a string.
DayOfWeek               Returns the current day of the week
DecodeDate        Decodes the specified date
DecodeTime        Decodes the specifies time.
EncodeDate        Returns values specified in date format.
EncodeTime        Returns values specified in time format.
FormatDateTime          Formats a date and time using the specified format.
IsLeapYear        Determines if a year is a leap year
Now                     Returns the current date and time.
StrToDate               Coverts a string to a date format.
StrToDateTime           Converts a string to a date/time format.
StrToTime               Converts a string to a time format.
Time                    Returns the current time.
TimeToStr               Converts a time format to a string.
```

There are also other misc. functions like DateTimeToTimeStamp and DateTimeToSystemTime which are not needed as much as the ones listed above, but it is good to know they exists.

Well hopefully I have armed you with enough information to show that working with dates and dealing with centuries in Delphi is not difficult. Take some time and experiment with working with dates before leaping head first into a project that is date intense and all will be fine.

You can get the source code for the projects associated with this file at http://www.informant.com/undu/dn971201/year2k.zip.

Return to Front Page

Speeding Up Master-Detail Tables

By Jens Fudge, LPTdata Denmark - archer@post4.tele.dk

If you've ever built an application that has two tabel grids connected (by TQuery's) to especially a remote database, the easy way is to set the detail-querys datasource property to the master-querys datasource...This will result in the detail grid only showing records with foreign key to the master. However, if there are 5000 records in the master, and to each master there is 50000 detail's, your user will have a problem using the arrow-keys to go through the master table. In fact each time you change record in the master table, you will be sending a query across the network to select all records in the detail table that match the current record in the master table. Sometimes a user is on a current record (in the master) and he can see the next interesting record is 7 records down. He might press the down-arrow-key numerous times until he gets there. This will be very slow, and he'll probably end up one or two records below the desired, and he has to press the up-arrow-key a couple of times. A way to work around this is to user the master-datasource's property OnDataChange, and NOT point the detail query's datasource property to anything!!!
On the OnDataChange property of the master datasource, simply reset a timer...

```
Begin
  {This actually resets the timer if its running....}
  Timer1.enabled := false;
  Timer1.enabled := true;
end;
```

The timer's interval is set to 300

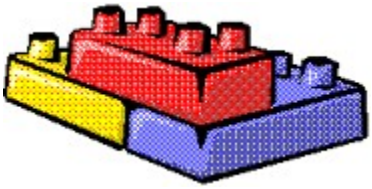
The OnTimer on the timer procedure you send the query yourself.....

```
begin
  Timer1.enabled := false;
  with Query_detail.SQL do
    begin
      clear;
      add(SELECT FIELD FROM TABLE WHERE);
      add(FOREIGN_KEY = :primarykey);
    end;
  Query_detail.prepare;
  Query_detail.parambyname('primarykey').asString :=
Query_master.FieldByName('PK').value;
  try
    Query_detail.open;
  Except on EdatabaseError do
     begin
       timer1.enabled := false;
       YourErrorHandling;
     end;
end;
```

By using this method, the user can scroll through the master table very quickly, using the up and down-arrow-key, without the detail having to keep up. When the user stops the detail will follw up. The only draw-back, as far as I can see, is that if the user changes the contents of a master-record, the detail query will be sent again. There should actually be a OnRowChange or OnRecordChange event on a TQuery, perhaps someone will make it some day.

Hope all you db-programmers find this tip usefull

Return to Tips & Tricks
Return to Front Page



:Beginners Block

by Alan Lloyd - alanglloyd@aol.com

begin

```
{ more pointers than Custer saw - part one}
```

When micro-computers first appeared, programming was a matter of hand entering values into memory locations, and keeping a list of what went where. Data locations held values, and machine code locations held processor instructions. To add A to B one had to remember (on paper if necessary) that variable A was held in location 2A3, and B in 1E7. So move the contents of 2A3 into one register, the contents of 1E7 into another, add the two registers - now where do I put the result, ah yes back in 23C, but that's where I put a character . . .. This sordid activity came to a halt when the program was held in a file, and a clever program turned A + B = C into the correct form.

What we had was a compiler (of sorts) which knew that A was held in 2A3, and whenever A was needed, it used the contents of 2A3. 2A3 was a pointer to the value of variable A, and that, as they say, was History^ :-)) (why History^, well the Delphi operator ^ (known as a caret) means 'contents' when appended to a pointer).

So a pointer is the memory address of an item of interest which is held in the PC's memory. Pointers can be the address of a variable (of any type), of a procedure, of a function, or even the address of a pointer to another item of interest. In fact as far as data is concerned, the compiler only deals with pointers in the program it is compiling. Sure, it holds a list of names (which humans readily understand) corresponding to those pointers, but no variable, procedure, or other name gets into the executable code, only pointers.

Pointers can be generic or typed, and the Delphi compiler's strict typing comes into play when pointers are used. To get the address of an item to assign it to a pointer variable we use the @ operator as a prefix, so @MyInt returns the address of MyInt. This pointer (the result of an @ operation) is a generic pointer, it can be assigned to a pointer of any type. But when used directly to obtain the contents of an address (by using the ^ suffix operator without assigning it to a typed pointer) it must usually be type-cast.

Let's declare some, types and values of pointers and variables

```
type
  Pinteger = ^Integer;  {pointer to an integer}
var
  MyInt1, MyInt2 : integer;
  PtrToInt1, PtrToInt2 : Pinteger;
```

Notice first of all the convention for naming pointers and pointer types with a 'P' or 'Ptr' prefix. Secondly that pointer types are declared using the ^ (caret) prefix operator.

We can now say

```
begin
  MyInt := 23;           { set a value for MyInt }
  PtrToInt1 := @MyInt;   { assign the address of MyInt to PtrToAnInt }
  PtrToInt1^ := 18;      { set the contents of the address in PtrToAnInt to}
```

```
                          { 18. MyInt now a value of 18 }
  YourInt := 36;
  PtrToInt2 := @YourInt;
  PtrToInt2^ := PtrToInt1^; { YourInt is set to the same value as MyInt}
end;
```

Similarly with strings we can say

```
type
  Pstring = ^string;    {pointer to a string}
var
  MyString : string;
  PtrToStringA : Pstring;
begin
  MyString := 'Hello World';
  PtrToStringA := @MyString;
  PtrToStringA^ := 'Bonjour ami';  {MyString is now 'Bonjour ami'
end;
```

The value of a pointer is a number, and the contents of the address with that number may be a string, an integer, or whatever. The @ operator returns a generic pointer (it can point to any variable) but the variable we allocate it to is usually a typed pointer. De-referencing is obtaining the value of the address pointed to by the pointer. The value returned by a typed pointer which is de-referenced, is of the type corresponding to the pointer. So de-referencing a string pointer returns a string type, de-referencing an integer pointer returns an integer type. The compiler ensures that you assign the de-referenced types correctly. You can of course type-cast the pointers, or their de-referenced values, but as with all type-casts, the compiler trusts you to know what you are doing. When you assign a generic pointer to a typed pointer, the compiler cannot check that what you are doing is correct, because the generic pointer can point to anything. So we can code :-

```
type
  PLongInt : ^LongInt;
var
  MyString : string;
  PtrToLongInt : PLongInt;
  MyLongInt : LongInt;
Begin
  MyString := 'Hello World';
  PtrToStringA := @MyString[1];
  PtrToLongInt := @MyString[1];
  MyLongInt := PtrLongInt^;
end;
```

Remember that a Delphi string's first byte (MyString[0]) is the number of characters, and the second byte is the first character, so what we've done is to turn the byte values of the first four characters of Hello into the value of one longint. MyLongInt is 1802200392 because the byte values of the string 'Hell' (longints are four bytes long) are 48h 65h 6Bh 6Bh which if the bytes are interpreted as a longint instead of a string (and the byte order reverses) become 6B6B6548h which equals 1802200392d. The data are just values in the computer memory, what they mean is up to the compiler and (particularly when you are dealing with pointers) you.

In order to display the values of the pointers (not what they point to) in labels on a form, we would not be able to simply do an IntToStr on them, because they are of type 'pointer to string', 'pointer to integer', or 'pointer to longint', even though they are all numbers. We have to type-cast them to an integer and then perform an IntToStr on them .:-

```
Label1.Caption := IntToStr(integer(PtrToLongInt));
Label2.Caption := IntToStr(PtrToLongInt^);
Label3.Caption := IntToStr(integer(PtrToStringA));
Label4.Caption := PtrToStringA^;
```

Having said all the above, Delphi deals with most aspects of referencing and de-referencing in the background, and you don't have to get involved with this low-level stuff. But there are some areas where

you have to know what is happening in order to properly handle objects and structures for instance, or in order to do a legitimate but crafty hack. Next issue of UNDU I'll deal with strings and PChars, functions and procedures, pointers to structures, auto-de-referencing

end;

Return to Tips & Tricks

Return to Front Page

Freeing Objects and Nil Pointers

by Alan Lloyd - alanglloyd@aol.com

An interesting discussion recently on a newsgroup centred on occasional GPFs when freeing an object which was conditionally created. In other words :-

```
var
  MyStringList : TStringList
Begin
if (some_condition) then
  MyStringList := TStringList.Create(Self);
  etc
end;
```

Later on…

```
MyStringList.Free;
```

Now the Free method will only Destroy an object when the pointer to that object is not free, but occasional GPFs by the enquirer suggested that the pointer was not always nil (maybe because sometimes local variables do not get initialised). This raised an explanation of what happens in such a situation. When you declare an object . . .

```
MyStringList : TStringlist
```

. . . the compiler allocates an address for MyStringList to use to point to the memory containing all the bits of MyStringList. This address is a pointer to MyStringList, and should contain the value Nil because MyStringList has not been created. (* we'll refer back here later). When MyStringList has been created this address will contain the valid address of MyStringList.

When the time comes for MyStringList to be destroyed, Delphi looks at this pointer to MyStringList, if it has a value of Nil then it assumes MyStringList has not been created, and there is no need or mandate to free anything. However if it has a value other than Nil then it attempts to free TStringList amount of memory starting from the address in MyStringList.

HOWEVER (referring to the previous *) if the pointer has an initial value other than Nil (say because the memory was not properly initialised) then Delphi will attempt to free the TStringList amount of memory starting at whatever (random) address it finds. But because the value is meaningless it will be attempting to free memory it has no right to touch and will receive a BIG GPF.

Therefore - for conditional object creation, do your own initialising to Nil . . .

```
MyStringList := nil;
if (some_condition) then
  MyStringList := TStringList.Create(Self);
```

. . . and all should be well. I have seen programmer recommendations to always set object pointers to nil before creating them and after Free-ing or specifically FreeMem-ing them. Certainly if an object is conditionally created, this would be good practice.

Note that this is NOT creating MyStringList, only filling the pointer to MyStringList with nil, Delphi is doing the referencing for you, giving you the same as

```
var
  MyStringList :TStringList;
```

```
  Ptr : ^TStringList;
Begin
  Ptr := @MyStringList;  {get address of MyStringList pointer}
  Ptr^ := nil;                   {set its contents to nil}
end;
```

Return to Tips & Tricks

Return to Front Page

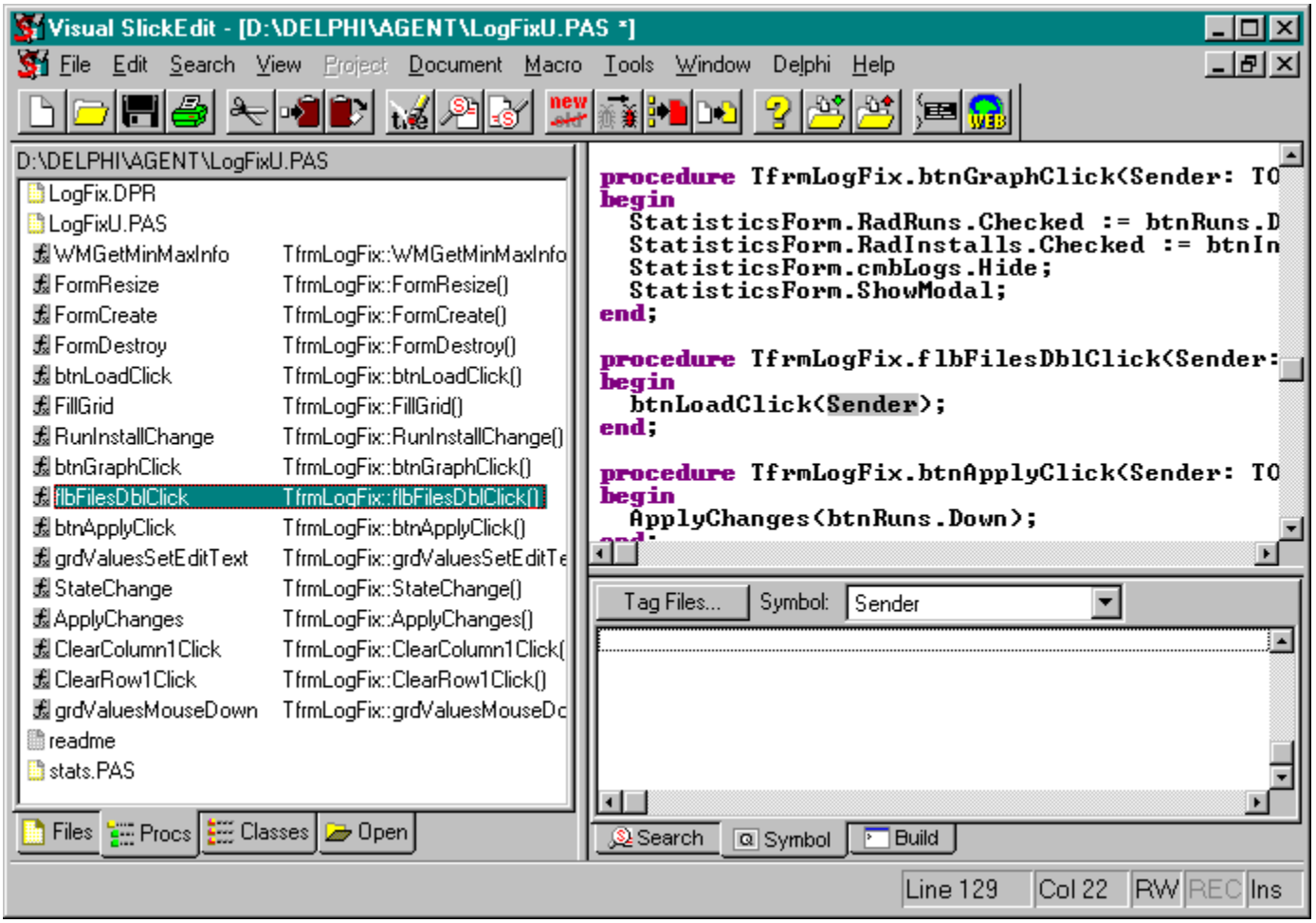
The Unofficial Newsletter of Delphi Users - Issue #24 - December 1997

Visual SlickEdit 3.0 – Delphi Edition

By Robert Vivrette – RobertV@csi.com

Visual SlickEdit is a great programmers editor and it continues to get better with each new release. This latest release, Version 3.0 – Delphi Edition is no exception. This latest edition was released just a couple of weeks ago and I got an opportunity to check out all the new features. For those of you who missed my initial review of VSE, you can check it out in the June 1st issue of UNDU (Issue #22). In this review, I am going to be covering just the changes since this prior version.

One of the first things you will notice in version 3.0 is the revised user interface as shown below:



On the left side, you can see a tree-like arrangement that shows all the files associated with a project. When you select the **Procs** tab, it allows you to view a list of just the procedure headers associated with a particular unit. This list can of course be sorted in a few different ways. By double-clicking on a header, VSE will take you to that procedure and display it in the code editing window on the right.
The code editor has undergone a number of nice enhancements with this new version.

In my prior review, I noted that the Delphi support was a little weak in areas like function keys usage. For example, in the old version, the key combination for compiling or running your project was different for VSE and Delphi. This has been corrected in this latest version. While in VSE, you can hit F9 and your program will be compile and executed. A small addition, but one that is appreciated for those who repeatedly code and test an application.

In my pre-release version of VSE, I did encounter one problem with file sharing. Occasionally, when I switched back and forth, I would get a error box from Delphi saying that another user was already using a

file. By clicking "Retry" it went ahead normally, so this was only a minor problem in my mind. Also, as I said this was a pre-release version of the product, so they likely have corrected this.

Some of the additional features in this latest version include:

- C++/Java/Delphi Class Browser. Displays class members, optionally shows inherited accessible members, views inheritance, searches for members, etc.
- Dynamic tagging. Automatically updates project tag files and global tag files. Tracks current function, functions/methods in current file, and tag symbols for word at cursor.
- Customizable dockable tool bars. Project toolbar with tabs for Class Browser, tags in file, file open, and project files. Output toolbar with tabs for tag symbol tracking, multi-file search output, and compilation output.
- Improved SmartPaste for character selections
- Multi-file spell checking
- Improved Multi-file searching
- Special character display
- More options in the code beautifier
- Support for Microsoft IntelliMouse (32-bit only)
- Color coding can now be configured for 4 multi-line comments.

My one big wish-list item I would like is still a deeper integration with the Delphi IDE. As it stands, VSE is still a separate code editor that runs along with Delphi and synchronizes information between them. But I would still like to see the editor as a replacement to the Delphi IDE code editor. That way, you could launch Delphi, and VSE and all its tremendous power would be right there in the code editing window. But in fairness, this would probably be quite an effort and would make a very Delphi-specific version that would be hard to keep up to date with the other versions of VSE. One of the features that MicroEdge prides themselves on is that whatever operating system you have and whatever programming language you are working on, there is a version of VSE that will work for you.

VSE continues to grow stronger and stronger in its Delphi and this latest version is a great addition to any Delphi Programmer's repertoire. There are literally hundereds of advanced features that you may not ever use in VSE, but you will be glad they are there when you need them.

You can learn more about Visual SlickEdit v3.0 Delphi Edition at http://www.slickedit.com

Return to Front Page