The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

Well, Delphi 3.0 is here. It has taken a bit longer than anyone wanted, but better a late great product than an early flawed one! As Delphi 3.0 starts establishing itself in the marketplace, we will start seeing more articles in magazines and programming journals that specifically discuss some of the new features of Delphi 3.0. Although this latest version is still just a 32-bit programming environment like its 2.0 predecessor, the new features added (like creation of ActiveX controls for example) will give a bit of incentive for many to upgrade.

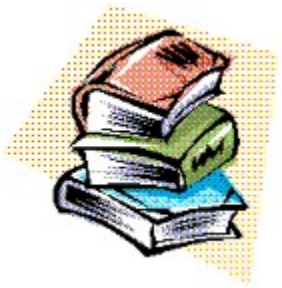
If you have discovered some really helpful new feature of Delphi 3.0 that you would like to share with others, please jot it down in an email and send it to me. It is always nice to get real-world views of a product from day-to-day programmers.

UNDU is turning out to be a great way of sharing ideas with other Delphi programmers. If you have a tip or trick or product review you would like to share, please send it to RobertV@compuserve.com. I have quite a few articles that I am preparing for future issues of UNDU so if you have sent something in, but havent seen it in print yet, please be patient!

I continue to be impressed by the international acceptance of Delphi. Many of us here in the United States might be naive enough to think that programming is the sole domain of the English speaking world. Not a chance! It is fascinating hearing from programmers from every conceivable corner of the globe. I have received countless emails from Delphi programmers in Belgium, South Africa, Brazil, China, the Philippines, Australia, many of the former Soviet Republics… the list goes on and on. Pardon me for using a cliché, but each day the world gets a little smaller and we all draw a little closer together…

This months randomly chosen winner for the UNDU prize goes to Jorge Romero Gomez for his article on Low-Level Windows Stuff. His prize is a copy of the book **Kick-Ass Java** published by Coriolis Group Books. In addition, as mentioned last month, OOPSoft, Inc has offered a special set of prizes. For last issue and this issue (only) they have given out 5 copies of their products. Last month, it was 5 copies of the ObjectExpress package (reviewed is issue #18) and this month it is their new SQLExpress package. The five randomly chosen winners this month are Mark DeBelder, Paul Furbacher, Mike Yui, David Sugden, and Benjamin Morin.

- 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

Return to Front Page

Where To Find UNDU

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

1. UNDUs official web site at `http://www.informant.com/undu/index.htm`. This site houses all the issues in both HTML and Windows HLP format. Click on the large icons for the HTML versions and the small red book icons for downloadable Windows HLP files.
2. *Borlands* Delphi forum on CompuServe (**GO DELPHI**) in the "Delphi IDE" file section. This forum will only hold the issues in Windows HLP format.

Tips & Tricks

The most popular section of UNDU is its Tips & Tricks section, so I always try to bring you the best ones I can. Keep those tips coming!

Delphi does a very good job at providing a simple wrappers around complex Windows API functions. This month, Jorge Romero Gomez (I hope I got that right) presents his own set of wrappers around Low Level Windows Stuff.

Sometimes, finding that one procedure or function in piles of source code can be a real chore. David Sugdens project this month is a program to extract the names of procedure and functions within your Delphi code. Check it out in the tip on Listing Procedures.

A while back, I got an email from Frank Harlow (nstn1846@fox.nstn.ca) on how to hide an applications icon in the Windows 95 Task Bar. After thinking about it for a bit, I put together a few lines of code which seemed to do the trick. Check it out in Hiding Apps from the Task Bar.

Joelito Real brings an interesting discussion to this months issue on Excel OLE Tips. If you have been wanting to tinker with OLE, this would give you a good first shot at how it is done.

If you are a sound aficionado, you might be interested in the follow-up discussion by Alan Lloyd on Playing Sounds Asynchronously.

Also, in answer to a Question from Readers in issue #19, Mark DeBelder (pekari@glo.be) brings an answer on how to put Bitmaps on StringGrids.

And last but not least, we have a tip by Ken Dowling on How To Find Up-to-date Delphi 2 Books, one by Paul Furbacher on a neat use for the Margin Marker in Delphi 1 & 2, a short re-visit on the topic of Lock Violations by Paul Harding, and a quick Object Creation Tip by Mike Yui.

Return to Front Page

The Component Cookbook

This month in the Component Cookbook, Herbert Beemster brings us an interesting discussion of How to Compress a Bitmap, followed by a great solution for providing shutdown behavior for your Delphi application using Benjamin Morins TEndSession component.

Return to Front Page

UNDU Subscriber List

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

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

Thats all there is to it!

Return to Front Page

Questions (And Answers) From UNDU Readers

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

In answer to a Question from Readers in issue #19, Mark DeBelder (pekari@glo.be) brings an answer on how to put Bitmaps on StringGrids.

This Months Questions Are:

Keith Brown at BEDCO@aol.com asks:

I am writing a program that generates a one-page invoice based on the contents of four tables: Details (has customer name & info, plus a description of the work in a Memo field), Materials, Miscellanous and Labor.

My problem is that ReportSmith and QuickReports appear to only allow horizontal bands that can contain one table at a time. I need to be able to print from two tables side-by-side horizontally (two tables and the memo field are stacked vertically). Can this be done in ReportSmith or QuickReports?

Brian Boyce at BriBoyce@aol.com asks:

I am trying to use the validation information entered in database desktop to validate user entries to data aware components. I have tried using ValidateEdit but this doesn't seem to take the picture field into account. e.g. {Y,N} to only accept Y or N as input. It works when adding data from within DB desktop but not when I access the database from within Delphi.

Any help would be greatly appreciated, and thanks for the magazine - I'm new to Delphi and it's helping me to pick-up a lot of information in a very short space of time.

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

Danny Thorpes "Delphi Component Design"

A review by Robert Vivrette - RobertV@compuserve.com

Ever since I started programming in Delphi, I have had the opportunity to go through quite a number of books on the subject. Most have been good basic reading, but only a few stand out above the rest. *Delphi Component Design* by Danny Thorpe is one of this latter group. You won't find too many example programs or cool new component designs within its pages, but you will learn quite a bit about what goes on inside Delphi.

The author, Danny Thorpe, is an R&D engineer on Borlands Delphi development team, which makes him imminently qualified to write about detailed Delphi internal concepts. While most books go into showing you how to design new components or explain things the Delphi on-line help left out, *Delphi Component Design* takes a different approach. Rather than telling you this is how you do this, Danny explains why it is done that way and the details of how it is achieved.

Part 1 of the book covers Analysis and Design. There is a discussion of the Delphi Programming model as well as how we got to that point through the various languages that came before it. There is also a high-level discussion of component design. Sometimes we develop a component only thinking about the immediate need, but Danny explains where the real power of components comes from, and how you can retrain your thinking to develop extensible and reusable ones.

Part 2 of the book covers some of the implementation details of component design. Chapters include Virtual Methods, Polymorphism, Exceptions, Run-Time Type Information, Streaming, Messaging, VCL Subsystems, OLE and COM interfaces, and Optimization Techniques (my favorite). I learned far more than I really wanted to about Virtual Methods here! Occasionally, Danny will illustrate his point with a few lines of assembler. It really helps knowing what is going on behind a few Object Pascal statements!

Part 3 wraps up the book with a discussion of Design-Time Support Tools, including Delphis Open Tools Architecture, Property Editors, Component Editors, Experts and Add-in Tools.

As mentioned earlier, my favorite chapter is the one on Optimization Techniques. I have always been a fan of learning better, faster, and more efficient ways to accomplish something, and this chapter did not let me down. In fact, my copy of this chapter is riddled with little post-it note scraps stuck to most every page… Not many Delphi books make me want to mark a section to come back to later! In this chapter, he discusses things such as Window handles, memory management techniques, string management, as well as some Plain Old Dirty Tricks. The String management section was particularly an eye opener and after reading it, I jumped back into a few of my applications to change the way I had worked with Strings.

Currently, there are only 3 books that I possess that I consider to be indispensable works. The first is the *Win32 Programming API Bible*, then Ray Lischners *Secrets of Delphi 2*, and now Danny Thorpes *Delphi Component Design*. While the first two are primarily reference works for me, Dannys book has been more of an educational work… One that helps me think beyond the scope that I programmed in before.

If you are a newbie to component design or to Delphi, many of the topics discussed will likely go over your head. However, if you think youre hot stuff and know all there is to know about Delphi… buy this book. It will give you a renewed sense of humility…

If you want to learn about the inner workings of Delphi, you need look no further than *Delphi Component Design*.

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

How to Compress a Bitmap

by Herbert J. Beemster - HerbertJB@compuserve.com

Currently I'm working on a project involving lots of bitmaps… we're talking about some fifty to ninety bitmaps used by a program. As you might know the structure of a bitmap has a remarkable resemblance with a soap-bubble, in that is there is plenty of air but only a fraction of water! So we can use some compression here.

After a long quest involving resources as UNDU and CompuServe I ended up with a couple of compressors and some samples using them but no direct bitmap compressor. So it was time to produce something of my own. Fortunately with the sources of the things I found it was relatively simple to knit together a descendant of TBitmap which has the extra features of storing and loading a compressed bitmap.

The result of this all is implemented in a unit called UBitmap. I also added a simple test program to show how it works. The test program loads a normal bitmap and writes out a compressed form as TEST.LZR. By implementing the components load and saving methods, you can then deal strictly with these compressed files.

Enjoy!

Source for Ubitmap.pas

Source for LZRW1KH.pas

Source for FBitmap.pas

Source for FBitmap.dfm

Return to Component Cookbook

Return to Front Page

Source for Ubitmap.pas

```
unit UBitmap;
{
Extented TBitmap object for Delphi 1.0
Added methods to load and save a compressed bitmap.
Copyright © 1997 by Herbert J.Beemster.

Questions, positive remarks, improvements etc. : herbertjb@compuserve.com

Credits goes to:
- Kurt Haenen for the LZRW1KH unit.
- Dan English for the trick with the streams.
- Danny Heijl for sample of using LZRW1KH.

This piece of source is hereby donated to the public domain. Enjoy!
}

interface

uses WinTypes, WinProcs, SysUtils, Classes, Graphics;



type
 TLZRBitmap = class(TBitmap)
 public
   procedure LZRLoadFromFile(const Filename : string); virtual;
   procedure LZRSaveToFile(const Filename : string); virtual;
 end; {TLZRBitmap}


implementation

uses
  LZRW1KH; {Credits : Kurt Haenen}

const
  ChunkSize = 32768;
  IOBufSize = (ChunkSize + 16);
  LZRWIdentifier : LONGINT =
  ((((((ORD('L') SHL 8)+ORD('Z')) SHL 8)+ORD('R')) SHL 8)+ORD('W'));

var
  InStream   : TMemoryStream;
  OutStream  : TMemoryStream;

procedure TLZRBitmap.LZRLoadFromFile(const Filename : string);
var
  Tmp,
  Identifier,
  OrigSize,
  SrcSize,
  DstSize    : LongInt;
  SrcBuf,
  DstBuf     : BufferPtr;
begin
  try
  {Create InStream & OutStream}
    InStream  := TMemoryStream.Create;
    OutStream := TMemoryStream.Create;
  {Create buffers for LZWR1KH}
```

```
    Getmem(SrcBuf, IOBufSize);
    Getmem(DstBuf, IOBufSize);

  {Load the compressed bitmap}
    InStream.LoadFromFile( Filename);
    InStream.Seek(0,0);

  {Decompress the lot...}
  {Read compression ID }
    InStream.Read( Identifier, SizeOf( LongInt));

  {Read in uncompressed filesize }
    InStream.Read( OrigSize, SizeOf( LongInt));

    DstSize := ChunkSize;
    SrcSize := 0;
    while (DstSize = ChunkSize) do
    begin
    {Read size of compressed block }
      Tmp := InStream.Read( SrcSize, SizeOf( Word));
    {Read compressed block }
      InStream.Read( SrcBuf^, SrcSize);
    {Decompress block }
      DstSize := Decompression( SrcBuf, DstBuf, SrcSize);
    {Write decompressed block out to OutStream }
      OutStream.Write( DstBuf^, DstSize);
    end;

  {TBitmap thinks its loading from a file!}
    OutStream.Seek(0,0);
    LoadfromStream( OutStream);

  finally
  {Clean Up Memory}
    InStream.Free;
    OutStream.Free;
    Freemem( SrcBuf, IOBufSize);
    Freemem( DstBuf, IOBufSize);

  end; {try}

end; {LZRLoadFromFile}


procedure TLZRBitmap.LZRSaveToFile(const Filename : string);
var
  Size,
  CompIdentifier,
  SrcSize,
  DstSize    : LongInt;
  SrcBuf,
  DstBuf     : BufferPtr;

begin
  try
  {Create InStream & OutStream}
    InStream  := TMemoryStream.Create;
    OutStream := TMemoryStream.Create;
  {Create buffers for LZWR1KH}
    Getmem(SrcBuf, IOBufSize);
    Getmem(DstBuf, IOBufSize);

  {Save the bitmap to InStream}
```

```
    SaveToStream( InStream);
    InStream.Seek(0,0);

  {Compress the lot...}
  {Write out compression ID }
    CompIdentifier := LZRWIdentifier;
    OutStream.Write( CompIdentifier, SizeOf( LongInt));

  {Write out uncompressed filesize }
    Size := InStream.Size;
    OutStream.Write( Size, SizeOf( LongInt));

    SrcSize := ChunkSize;
    while (SRCSize = ChunkSize)
    do
    begin
    {Read a block of data }
      SrcSize := InStream.Read( SrcBuf^, ChunkSize);
    {Compress it }
      DstSize := Compression( SrcBuf, DstBuf, SrcSize);
    {Write out compressed size }
      OutStream.Write( DstSize, SizeOf( Word));
    {Write out compressed data }
      OutStream.Write( DstBuf^, DstSize);
    end; {while}

  {Save compressed OutStream to file}
    OutStream.SaveToFile( Filename);

  finally
  {Clean Up Memory}
    InStream.Free;
    OutStream.Free;
    Freemem( SrcBuf, IOBufSize);
    Freemem( DstBuf, IOBufSize);

  end; {try}

end; {LZRSaveToFile}

end.
```

Return to Article

Return to Component Cookbook

Return to Front Page

Source for LZRW1KH.PAS

```
{$R-} { NO range checking !! }

{
-------------------------------------------------------------
    This posting includes the sources for the Turbo Pascal
version of the LZRW1/KH compression algoritm.
-------------------------------------------------------------
File #1 : The LZRW1KH unit
--------------------------
}
{    ##################################################################    }
{    ##                                                              ##    }
{    ##      ##    ##### #####  ##   ##  ##      ## ##  ## ##  ##     ##    }
{    ##      ##      ### ##  ## ## # ## ###     ##  ## ##  ##  ##     ##    }
{    ##      ##     ###  #####  #######  ##    ##   ####   ######     ##    }
{    ##      ##    ###   ##  ## ### ###  ##   ##    ## ##  ##  ##     ##    }
{    ##      ##### ##### ##  ## ##   ## #### ##     ##  ## ##  ##     ##    }
{    ##                                                              ##    }
{    ##   EXTREMELY FAST AND EASY TO UNDERSTAND COMPRESSION ALGORITM  ##    }
{    ##                                                              ##    }
{    ##################################################################    }
{    ##                                                              ##    }
{    ##   This unit implements the updated LZRW1/KH algoritm which   ##    }
{    ##   also implements  some RLE coding  which is usefull  when   ##    }
{    ##   compress files  containing  a lot  of consecutive  bytes   ##    }
{    ##   having the same value.   The algoritm is not as good  as   ##    }
{    ##   LZH, but can compete with Lempel-Ziff.   It's the fasted   ##    }
{    ##   one I've encountered upto now.                             ##    }
{    ##                                                              ##    }
{    ##                                                              ##    }
{    ##                                                              ##    }
{    ##                                              Kurt HAENEN     ##    }
{    ##                                                              ##    }
{    ##################################################################    }

UNIT LZRW1KH;

INTERFACE

uses SysUtils;

{$IFDEF WIN32}
type Int16 = SmallInt;
{$ELSE}
type Int16 = Integer;
{$ENDIF}

CONST
    BufferMaxSize  = 32768;
    BufferMax      = BufferMaxSize-1;
    FLAG_Copied    = $80;
    FLAG_Compress  = $40;

TYPE
    BufferIndex    = 0..BufferMax + 15;
    BufferSize     = 0..BufferMaxSize;
       { extra bytes needed here if compression fails      *dh *}
    BufferArray    = ARRAY [BufferIndex] OF BYTE;
    BufferPtr      = ^BufferArray;
```

```
    ELzrw1KHCompressor = Class(Exception);


FUNCTION  Compression    (    Source,Dest    : BufferPtr;
                              SourceSize     : BufferSize   )    : BufferSize;

FUNCTION  Decompression  (    Source,Dest    : BufferPtr;
                              SourceSize     : BufferSize   )    : BufferSize;

IMPLEMENTATION

type
  HashTable      = ARRAY [0..4095] OF Int16;
  HashTabPtr     = ^Hashtable;

VAR
  Hash                     : HashTabPtr;

                             { check if this string has already been seen }
                             { in the current 4 KB window }
FUNCTION  GetMatch       (    Source         : BufferPtr;
                              X              : BufferIndex;
                              SourceSize     : BufferSize;
                              Hash           : HashTabPtr;
                           VAR Size           : WORD;
                           VAR Pos            : BufferIndex  )    : BOOLEAN;
VAR
  HashValue      : WORD;
  TmpHash        : Int16;
BEGIN
  HashValue := (40543*((((Source^[X] SHL 4) XOR Source^[X+1]) SHL 4) XOR
                                      Source^[X+2]) SHR 4) AND $0FFF;
  Result := FALSE;
  TmpHash := Hash^[HashValue];
  IF (TmpHash <> -1) and (X - TmpHash < 4096) THEN BEGIN
    Pos := TmpHash;
    Size := 0;
    WHILE ((Size < 18) AND (Source^[X+Size] = Source^[Pos+Size])
                       AND (X+Size < SourceSize)) DO begin
      INC(Size);
    end;
    Result := (Size >= 3)
  END;
  Hash^[HashValue] := X
END;
                                    { compress a buffer of max. 32 KB }
FUNCTION  Compression(Source, Dest : BufferPtr;
                      SourceSize   : BufferSize) :BufferSize;
VAR
  Bit,Command,Size         : WORD;
  Key                      : Word;
  X,Y,Z,Pos                : BufferIndex;
BEGIN
  FillChar(Hash^,SizeOf(Hashtable), $FF);
  Dest^[0] := FLAG_Compress;
  X := 0;
  Y := 3;
  Z := 1;
  Bit := 0;
  Command := 0;
  WHILE (X < SourceSize) AND (Y <= SourceSize) DO BEGIN
    IF (Bit > 15) THEN BEGIN
```

```
      Dest^[Z] := HI(Command);
      Dest^[Z+1] := LO(Command);
      Z := Y;
      Bit := 0;
      INC(Y,2)
    END;
    Size := 1;
    WHILE ((Source^[X] = Source^[X+Size]) AND (Size < $FFF)
                        AND (X+Size < SourceSize)) DO begin
              INC(Size);
    end;
    IF (Size >= 16) THEN BEGIN
      Dest^[Y] := 0;
      Dest^[Y+1] := HI(Size-16);
      Dest^[Y+2] := LO(Size-16);
      Dest^[Y+3] := Source^[X];
      INC(Y,4);
      INC(X,Size);
      Command := (Command SHL 1) + 1;
    END
    ELSE begin { not size >= 16 }
      IF (GetMatch(Source,X,SourceSize,Hash,Size,Pos)) THEN BEGIN
        Key := ((X-Pos) SHL 4) + (Size-3);
        Dest^[Y] := HI(Key);
        Dest^[Y+1] := LO(Key);
        INC(Y,2);
        INC(X,Size);
        Command := (Command SHL 1) + 1
      END
      ELSE BEGIN
        Dest^[Y] := Source^[X];
        INC(Y);
        INC(X);
        Command := Command SHL 1
      END;
    end; { size <= 16 }
    INC(Bit);
  END; { while x < sourcesize ... }
  Command := Command SHL (16-Bit);
  Dest^[Z] := HI(Command);
  Dest^[Z+1] := LO(Command);
  IF (Y > SourceSize) THEN BEGIN
    MOVE(Source^[0],Dest^[1],SourceSize);
    Dest^[0] := FLAG_Copied;
    Y := SUCC(SourceSize)
  END;
  Result := Y
END;

                                    { decompress a buffer of max 32 KB }
FUNCTION  Decompression(Source,Dest    : BufferPtr;
                        SourceSize     : BufferSize) : BufferSize;
VAR
  X,Y,Pos                  : BufferIndex;
  Command,Size,K           : WORD;
  Bit                      : BYTE;
  SaveY                    : BufferIndex; { * dh * unsafe for-loop variable Y }

BEGIN
  IF (Source^[0] = FLAG_Copied) THEN  begin
    FOR Y := 1 TO PRED(SourceSize) DO begin
      Dest^[PRED(Y)] := Source^[Y];
      SaveY := Y;
```

```
    end;
    Y := SaveY;
  end
  ELSE BEGIN
    Y := 0;
    X := 3;
    Command := (Source^[1] SHL 8) + Source^[2];
    Bit := 16;
    WHILE (X < SourceSize) DO BEGIN
      IF (Bit = 0) THEN BEGIN
        Command := (Source^[X] SHL 8) + Source^[X+1];
        Bit := 16;
        INC(X,2)
      END;
      IF ((Command AND $8000) = 0) THEN BEGIN
           Dest^[Y] := Source^[X];
           INC(X);
           INC(Y)
      END
      ELSE BEGIN  { command and $8000 }
        Pos := ((Source^[X] SHL 4)
               +(Source^[X+1] SHR 4));
        IF (Pos = 0) THEN BEGIN
          Size := (Source^[X+1] SHL 8) + Source^[X+2] + 15;
          FOR K := 0 TO Size DO begin
               Dest^[Y+K] := Source^[X+3];
          end;
          INC(X,4);
          INC(Y,Size+1)
        END
        ELSE BEGIN  { pos = 0 }
          Size := (Source^[X+1] AND $0F)+2;
          FOR K := 0 TO Size DO
               Dest^[Y+K] := Dest^[Y-Pos+K];
          INC(X,2);
          INC(Y,Size+1)
        END; { pos = 0 }
      END;  { command and $8000 }
      Command := Command SHL 1;
      DEC(Bit)
    END { while x < sourcesize }
  END;
  Result := Y
END;  { decompression }

{
  Unit "Finalization" as Delphi 2.0 would have it
}

var
  ExitSave : Pointer;

Procedure Cleanup; far;
begin
  ExitProc := ExitSave;
  if (Hash <> Nil) then
    Freemem(Hash, Sizeof(HashTable));
end;


Initialization

  Hash := Nil;
```

```
  try
    Getmem(Hash,Sizeof(Hashtable));
  except
    Raise ELzrw1KHCompressor.Create('LZRW1KH : no memory for HASH table');
  end;
  ExitSave := ExitProc;
  ExitProc := @Cleanup;
END.
```

Return to Article

Return to Component Cookbook

Return to Front Page

Source for FBitmap.pas

```
unit FBitmap;

{Test unit for UBitmap's TLZRBitmap}

interface

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


type
  TFrmMain = class(TForm)
    PnlButtons: TPanel;
    DlgOpen: TOpenDialog;
    BtnPack: TButton;
    BtnUnpack: TButton;
    BtnOpen: TButton;
    ScrollBox: TScrollBox;
    Image: TImage;
    procedure BtnPackClick(Sender: TObject);
    procedure BtnUnpackClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure BtnOpenClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    LZRBitmap : TLZRBitmap;
  end;

var
  FrmMain: TFrmMain;

implementation

{$R *.DFM}

procedure TFrmMain.BtnPackClick(Sender: TObject);
var
  TmpPicture : TPicture;
begin
  TmpPicture := TPicture.Create;
  TmpPicture.Assign( Image.Picture);
  Image.Picture.Bitmap.Monochrome := TRUE;
  Update;

  LZRBitmap.LZRSaveToFile( 'TEST.LZR');

  BtnUnpack.Enabled := TRUE;
  Image.Picture := TmpPicture;
  Update;
  TmpPicture.Free;
end;

procedure TFrmMain.BtnUnpackClick(Sender: TObject);
begin
  Image.Picture.Bitmap.Monochrome := TRUE;
  Update;
```

```
  LZRBitmap.LZRLoadFromFile( 'TEST.LZR');
(*  LZRBitmap.SaveToFile( 'TEST.BMP'); *)  {uncomment this if you want it
                                             written to disk}
  Image.Picture.Bitmap := LZRBitmap;
end;

procedure TFrmMain.FormCreate(Sender: TObject);
begin
  LZRBitmap := TLZRBitmap.Create;
end;

procedure TFrmMain.BtnOpenClick(Sender: TObject);
begin
  DlgOpen.Filename := '*.BMP';
  if (DlgOpen.Execute)
  then
  begin
    Image.Picture.Bitmap.Monochrome := TRUE;
    Update;

    LZRBitmap.LoadFromFile( DlgOpen.Filename);
    DlgOpen.InitialDir   := ExtractFilePath( DlgOpen.Filename);
    Image.Picture.Bitmap := LZRBitmap;

    BtnPack.Enabled   := TRUE;
    BtnUnpack.Enabled := FALSE;

    Image.Picture.Bitmap.Monochrome := FALSE;
    Update;

  end; {if}

end;

procedure TFrmMain.FormDestroy(Sender: TObject);
begin
  LZRBitmap.Free;
end;

end.
```

Return to Article

Return to Component Cookbook

Return to Front Page

Source for FBitmap.dfm

```
object FrmMain: TFrmMain
  Left = 200
  Top = 95
  Width = 344
  Height = 308
  Caption = 'FrmMain'
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'Arial'
  Font.Style = []
  PixelsPerInch = 96
  OnCreate = FormCreate
  OnDestroy = FormDestroy
  TextHeight = 16
  object PnlButtons: TPanel
    Left = 0
    Top = 225
    Width = 336
    Height = 56
    Align = alBottom
    TabOrder = 0
    object BtnPack: TButton
      Left = 20
      Top = 12
      Width = 89
      Height = 33
      Caption = 'Pack'
      Enabled = False
      TabOrder = 0
      OnClick = BtnPackClick
    end
    object BtnUnpack: TButton
      Left = 124
      Top = 12
      Width = 89
      Height = 33
      Caption = 'Unpack'
      Enabled = False
      TabOrder = 1
      OnClick = BtnUnpackClick
    end
    object BtnOpen: TButton
      Left = 227
      Top = 12
      Width = 89
      Height = 33
      Caption = 'Open'
      TabOrder = 2
      OnClick = BtnOpenClick
    end
  end
  object ScrollBox: TScrollBox
    Left = 0
    Top = 0
    Width = 336
    Height = 225
    Align = alClient
    TabOrder = 1
    object Image: TImage
      Left = 0
```

```
      Top = 0
      Width = 334
      Height = 223
      AutoSize = True
    end
  end
  object DlgOpen: TOpenDialog
    DefaultExt = 'BMP'
    Filter = 'Bitmaps|*.BMP'
    Left = 306
    Top = 201
  end
end
```

Return to Article

Return to Component Cookbook

Return to Front Page

Low Level Windows Stuff

by Jorge Romero Gomez, Merchise - merchise@minrex.cu

This unit has (I think) some very nice functions for low-level manipulation of windows. The ones that are of most use to me are the WndProc related ones (for subclassing windows), the LockPainting ones (locks the updating of windowed controls), and the WindowSizeable ones (I don't understand why Delphi always forces us to allow resizing of any normal-looking form, what if I don't want a dialog?)

The Message queue inspectors were used for a FlicPlayer that could drop frames when needed (I was using a high resolution multimedia timer, it could become a real pain with a BIG flic).

If you think this unit is useful, write to me and maybe I can send you some higher level stuff...

Source for WinUtils

Return to Tips & Tricks

Return to Front Page

Source for WinUtils

```
unit WinUtils;

// Copyright (c) 1996 Jorge Romero Gomez, Merchise.

interface

  uses
    Windows, Messages, SysUtils;

  // Message Queue Inspection

  function WindowHasMessageWaiting( WinHandle : HWND ) : boolean;

  const
    // See QS_XXX flags in GetQueueStatus
    QS_USERINPUT = QS_KEY or QS_MOUSEBUTTON or QS_HOTKEY;
    QS_IMPORTANT = QS_USERINPUT or QS_POSTMESSAGE;

  function ThreadHasInputWaiting : boolean;
  function ThreadHasMessageWaiting : boolean;
  function ThreadHasImportantMessageWaiting : boolean;
  function ThreadHasTheseMessagesWaiting( Mask : integer ) : boolean;

  // Mask = QS_XXX flags

  // Window searching

  function GetWindowOfClass( const ClassName : string ) : HWND;
  function GetWindowOfCaption( const Caption : string ) : HWND;

  // Window info

  function  GetWindowCaption( WinHandle : HWND ) : string;
  function  GetWindowClass( WinHandle : HWND ) : string;

  function  GetWindowInstance( WinHandle : HWND ) : HMODULE;
  function  GetWindowID( WinHandle : HWND ) : integer;

  // Window style

  procedure ExcludeWindowStyle( WinHandle : HWND; NewStyle : integer );
  procedure IncludeWindowStyle( WinHandle : HWND; NewStyle : integer );
  procedure SetWindowStyle( WinHandle : HWND; NewStyle : integer );
  function  GetWindowStyle( WinHandle : HWND ) : integer;

  procedure ExcludeWindowExStyle( WinHandle : HWND; NewExStyle : integer );
  procedure IncludeWindowExStyle( WinHandle : HWND; NewExStyle : integer );
  procedure SetWindowExStyle( WinHandle : HWND; NewExStyle : integer );
  function  GetWindowExStyle( WinHandle : HWND ) : integer;

  // Window list

  function  GetWindowOwner( WinHandle : HWND ) : HWND;
  function  GetFirstChild( WinHandle : HWND ) : HWND;
  function  GetFirstSibling( WinHandle : HWND ) : HWND;
  function  GetLastSibling( WinHandle : HWND ) : HWND;
  function  GetNextSibling( WinHandle : HWND ) : HWND;
  function  GetPrevSibling( WinHandle : HWND ) : HWND;

  // Window Visuals
```

```
  procedure SetWindowRedraw( WinHandle : HWND; Redraw : boolean );
  // For compatibility with Windowsx.h
  procedure LockWindowPainting( WinHandle : HWND; Locked : boolean );
  procedure UnlockPainting( WinHandle : HWND );

  function  IsMinimized( WinHandle : HWND ) : boolean;
  function  IsMaximized( WinHandle : HWND ) : boolean;
  function  IsRestored( WinHandle : HWND ) : boolean;

  procedure MaximizeWindow( WinHandle : HWND );
  procedure MinimizeWindow( WinHandle : HWND );
  procedure RestoreWindow( WinHandle : HWND );
  procedure ActivateWindow( WinHandle : HWND );

  procedure UpdateWindowFrame( WinHandle : HWND );
  procedure SetWindowSizeable( WinHandle : HWND; WinSizeable : boolean );
  function  WindowIsSizeable( WinHandle : HWND) : boolean;

  function  GetRealClientRect( WinHandle : HWND ) : TRect;

  function  WindowIsDropTarget( WinHandle : HWND) : boolean;

  type
    EWndProcCannotBeRestored = class( Exception );

  function SubclassWindow( WinHandle : integer; NewWndProc : pointer ) : pointer;
  // For compatibility with Windowsx.h
  function ChangeWndProc( WinHandle : integer; NewWndProc : pointer ) : pointer;
  function RestoreWndProc( WinHandle : integer; OldWndProc, SafeCheck : pointer ) :
pointer;

implementation

  uses
    Debug;

  // Window Messages

  function WindowHasMessageWaiting( WinHandle : HWND ) : boolean;
    var
      Msg : TMsg;
    begin
      Result := PeekMessage( Msg, WinHandle, 0, 0, PM_NOREMOVE );
    end;

  function ThreadHasInputWaiting : boolean;
    begin
      Result := GetInputState;
    end;

  function ThreadHasMessageWaiting : boolean;
    begin
      Result := GetQueueStatus( QS_ALLINPUT ) <> 0;
    end;

  function ThreadHasImportantMessageWaiting : boolean;
    begin
      Result := GetQueueStatus( QS_IMPORTANT ) <> 0;
    end;

  function ThreadHasTheseMessagesWaiting( Mask : integer ) : boolean;
    begin
```

```
    Result := GetQueueStatus( Mask ) <> 0;
  end;

// Window info

function GetWindowCaption( WinHandle : HWND ) : string;
  begin
    SetLength( Result, MAX_PATH );
    SetLength( Result, GetWindowText( WinHandle, pchar( Result ), length(Result)));
  end;

function GetWindowClass( WinHandle : HWND ) : string;
  begin
    SetLength( Result, MAX_PATH );
    SetLength( Result, GetClassName( WinHandle, pchar( Result ), length(Result)));
  end;

function GetWindowStyle( WinHandle : HWND ) : integer;
  begin
    Result := GetWindowLong( WinHandle, GWL_STYLE );
  end;

procedure SetWindowStyle( WinHandle : HWND; NewStyle : integer );
  begin
    SetWindowLong( WinHandle, GWL_STYLE, NewStyle );
  end;

procedure IncludeWindowStyle( WinHandle : HWND; NewStyle : integer );
  begin
    NewStyle := GetWindowStyle( WinHandle ) or NewStyle;
    SetWindowLong( WinHandle, GWL_STYLE, NewStyle );
  end;

procedure ExcludeWindowStyle( WinHandle : HWND; NewStyle : integer );
  begin
    NewStyle := GetWindowStyle( WinHandle ) and (not NewStyle);
    SetWindowLong( WinHandle, GWL_STYLE, NewStyle );
  end;

function GetWindowExStyle( WinHandle : HWND ) : integer;
  begin
    Result := GetWindowLong( WinHandle, GWL_EXSTYLE );
  end;

procedure SetWindowExStyle( WinHandle : HWND; NewExStyle : integer );
  begin
    SetWindowLong( WinHandle, GWL_EXSTYLE, NewExStyle );
  end;

procedure IncludeWindowExStyle( WinHandle : HWND; NewExStyle : integer );
  begin
    NewExStyle := GetWindowExStyle( WinHandle ) or NewExStyle;
    SetWindowLong( WinHandle, GWL_EXSTYLE, NewExStyle );
  end;

procedure ExcludeWindowExStyle( WinHandle : HWND; NewExStyle : integer );
  begin
    NewExStyle := GetWindowExStyle( WinHandle ) and (not NewExStyle);
    SetWindowLong( WinHandle, GWL_EXSTYLE, NewExStyle );
  end;

function GetWindowOwner( WinHandle : HWND ) : HWND;
  begin
```

```
    Result := GetWindow( GW_OWNER, WinHandle );
  end;

function GetWindowInstance( WinHandle : HWND ) : HMODULE;
  begin
    Result := GetWindowLong( WinHandle, GWL_HINSTANCE);
  end;

function GetFirstChild( WinHandle : HWND ) : HWND;
  begin
    Result := GetTopWindow( WinHandle );
  end;

function GetFirstSibling( WinHandle : HWND ) : HWND;
  begin
    Result := GetWindow( WinHandle, GW_HWNDFIRST );
  end;

function GetLastSibling( WinHandle : HWND ) : HWND;
  begin
    Result := GetWindow( WinHandle, GW_HWNDLAST );
  end;

function GetNextSibling( WinHandle : HWND ) : HWND;
  begin
    Result := GetWindow( WinHandle, GW_HWNDNEXT );
  end;

function GetPrevSibling( WinHandle : HWND ) : HWND;
  begin
    Result := GetWindow( WinHandle, GW_HWNDPREV );
  end;

function GetWindowID( WinHandle : HWND ) : integer;
  begin
    Result := GetDlgCtrlID( WinHandle );
  end;

// Window searching

function GetWindowOfCaption( const Caption : string ) : HWND;
  begin
    Result := FindWindow( nil, pchar(Caption) );
  end;

function GetWindowOfClass( const ClassName : string ) : HWND;
  begin
    Result := FindWindow( pchar(ClassName), nil );
  end;

// Window Visuals

procedure SetWindowRedraw( WinHandle : HWND; Redraw : boolean );
  begin
    SendMessage( WinHandle, WM_SETREDRAW, WPARAM( Redraw ), 0 );
  end;

procedure LockWindowPainting( WinHandle : HWND; Locked : boolean );
  begin
    SendMessage( WinHandle, WM_SETREDRAW, WPARAM( not Locked ), 0 );
  end;

procedure UnlockPainting( WinHandle : HWND );
```

```
  begin
    LockWindowPainting( WinHandle, false );
    InvalidateRect( WinHandle, nil, false );
  end;

function IsMinimized( WinHandle : HWND ) : boolean;
  begin
    Result := IsIconic( WinHandle );
  end;

function IsMaximized( WinHandle : HWND ) : boolean;
  begin
    Result := IsZoomed( WinHandle );
  end;

function IsRestored( WinHandle : HWND ) : boolean;
  begin
    Result := GetWindowStyle( WinHandle ) and ( WS_MINIMIZE or WS_MAXIMIZE ) = 0;
  end;

procedure MaximizeWindow( WinHandle : HWND );
  begin
    ShowWindow( WinHandle, SW_MAXIMIZE );
  end;

procedure MinimizeWindow( WinHandle : HWND );
  begin
    ShowWindow( WinHandle, SW_MINIMIZE );
  end;

procedure RestoreWindow( WinHandle : HWND );
  begin
    ShowWindow( WinHandle, SW_RESTORE );
  end;

procedure ActivateWindow( WinHandle : HWND );
  begin
    ShowWindow( WinHandle, SW_SHOWNORMAL );
  end;

procedure UpdateWindowFrame( WinHandle : HWND );
  begin
    SetWindowPos( WinHandle, 0, 0, 0, 0, 0, SWP_FRAMECHANGED or
    SWP_NOSIZE or SWP_NOMOVE or SWP_NOACTIVATE or SWP_NOZORDER );
  end;

procedure SetWindowSizeable( WinHandle : HWND; WinSizeable : boolean );
  begin
    if WinSizeable
      then IncludeWindowStyle( WinHandle, WS_SIZEBOX )
      else ExcludeWindowStyle( WinHandle, WS_SIZEBOX );
    UpdateWindowFrame( WinHandle );
  end;

function WindowIsSizeable( WinHandle : HWND ) : boolean;
  begin
    Result := ( GetWindowStyle( WinHandle ) and WS_SIZEBOX ) <> 0;
  end;

function GetRealClientRect( WinHandle : HWND ) : TRect;
  var
    WinStyle : dword;
  begin
```

```
      WinStyle := GetWindowStyle( WinHandle );
      GetClientRect( WinHandle, Result );
      with Result do
        begin
          if WinStyle and WS_HSCROLL <> 0
            then inc( Bottom, GetSystemMetrics( SM_CYHSCROLL ) );
          if WinStyle and WS_VSCROLL <> 0
            then inc( Right, GetSystemMetrics( SM_CXVSCROLL ) );
        end;
    end;

  function WindowIsDropTarget( WinHandle : HWND) : boolean;
    begin
      Result := ( GetWindowExStyle( WinHandle ) and WS_EX_ACCEPTFILES ) <> 0;
    end;

  function SubclassWindow( WinHandle : integer; NewWndProc : pointer ): pointer;
    begin
      Result := pointer( SetWindowLong( WinHandle, GWL_WNDPROC,longint(NewWndProc)));
    end;

  function ChangeWndProc( WinHandle : integer; NewWndProc : pointer ) : pointer;
    begin
      Result := pointer( SetWindowLong( WinHandle, GWL_WNDPROC, longint(NewWndProc)));
    end;

  function RestoreWndProc( WinHandle : integer; OldWndProc, SafeCheck: pointer ) :
pointer;
    begin
      Result := ChangeWndProc( WinHandle, OldWndProc );
      if Debugging and
         (Result <> nil) and (Result <> SafeCheck)
        then raise EWndProcCannotBeRestored.Create( '' );
    end;
end.
```

Return to Article

Return to Tips & Tricks

Return to Front Page

Hiding Apps from the Task Bar

by Robert Vivrette - RobertV@compuserve.com

A while back, I received an email from a gentleman who was wondering if it was possible to hide the icon of an application in the Windows 95 Task Bar. What I came up with was really just a matter of looking for a specific window and hiding it. With Delphi applications, all you have to do is this:

```
procedure TForm1.HideAppWindow;
var
  H : HWnd;
begin
  H := FindWindow(Nil,'Project1');
  if H <> 0 then ShowWindow(H,SW_HIDE);
end;
```

This example calls the FindWindow API function to go out and look for the Applications window by name. Every Delphi application has this hidden application window which is separate from any regular forms the app might have. The first parameter is the Class Name of the window you are looking for, and the second is its Window caption. In this example, we are telling FindWindow that we are not interested in the class name but rather just the Window caption. After we find the window, I use the ShowWindow API function with the SW_HIDE flag to make it invisible (I guess they shouldnt have called it ShowWindow the huh?)

And then I call my new HideAppWindow procedure from the FormCreate. The icon will be hidden, but the form will remain on the screen normally. The only odd behavior is that when the form is minimized, the app window appears in the taskbar and stays there when you restore it. The first part of the behavior (having it appear on the taskbar when it is minimized) is acceptable I think… How else would you get it restored? But the second behavior (staying visible after restoring it) isnt acceptable. This is easily overcome by adding a simple event handler to the Application.OnRestore method, and then from that method, call our HideAppWindow procedure again.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  HideAppWindow;
  Application.OnRestore := AppRestore;
end;

procedure TForm1.AppRestore(Sender: TObject);
begin
  HideAppWindow;
end;
```

Return to Tips & Tricks

Return to Front Page

Lock Violations Revisited

by Paul Harding - paulharding@compuserve.com

In UNDU #19 (January 97) I wrote about Lock Violations that could not be debugged due to running the 16 bit BDE on a 32 bit operating system and sharing the data tables. At a recent Borland Developer's conference, Bill Todd told me another way of getting around this problem, and so I thought I'd share it with everyone (I have also received email on this obscure but important topic).
Another way of avoiding getting strange 'Lock Violation' errors that occur if you mix using the 32 bit BDE with the 16 bit BDE on a network of, say Windows 3.1 and Windows 95 machines, is to do this:

Go into every Win 95 machine's control panel. Select as follows:

```
SYSTEM
PERFORMANCE
FILE SYSTEM
TROUBLESHOOTING
```

Now check the box Disable New File Sharing & Locking Semantics, and you will have to reboot. Make sure you do this for EVERY Win 95 machine, and hey presto - now you can run Delphi 1.0 shared database programs on your Win 95 and NT machines!

I hope this presents another approach to people who do have Win 3.x/95/NT mixed networks and many thanks to Bill Todd for this tip.

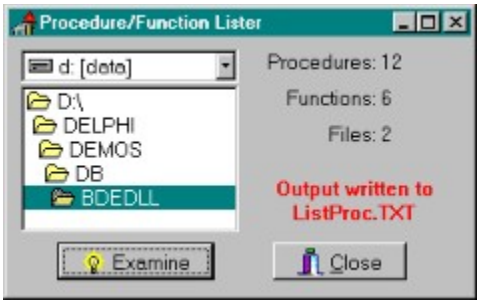
Return to Tips & Tricks

Return to Front Page

Listing Procedures and Functions

by David Sugden - DBSugden@compuserve.com
http://ourworld.compuserve.com/homepages/DBSugden

A very simple Delphi program - but one I've found useful if like me you write monolithic lumps of source and have to try to find your way around them later - I started off as a FORTRAN programmer... It's Delphi 1.0 for a Win 3.1 environment and should work elsewhere, no problems.
It looks through all the *.PAS files in a directory/folder and adds a file LISTPROC.TXT. This is a sorted tab separated text file giving the line numbers and file for each of the procedure and function declarations.
All you have to do is use your "worm-processor" to get the tab columns in sensible positions and print it out.




```
BitBtn1Click                appunit1        18
CheckDLLUse                 appunit1        68
ExitDLL                     appunit1        40
ExitDLL                     dllunit1        46
ExitDLL                     dllunit1       105
FormClose                   dllunit1        22
SelectBtn1Click             appunit1        17
TAppForm1.BitBtn1Click      appunit1        49
TAppForm1.SelectBtn1Click   appunit1        43
TAppForm1.ViewBtn1Click     appunit1        54
TDllForm1.FormClose         dllunit1        92
UseDLL                      appunit1        39
UseDLL                      dllunit1        45
UseDLL                      dllunit1       125
ViewBtn1Click               appunit1        19
ViewTable                   appunit1        41
ViewTable                   dllunit1        47
ViewTable                   dllunit1        69
```

Source for Listmain.pas

Source for Listmain.dfm

Return to Tips & Tricks

Return to Front Page

Source for Listmain.pas

```
unit Listmain;  { create a tab separated list of procs/fns
                  & where they appear in the source a project.
                  Comments to DBSugden@compuserve.com
                  http://ourworld.compuserve.com/homepages/DBSugden
                  Copyright D B Sugden Feb 97 - Public Domain software }
interface

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

type
  TLForm = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    BitBtn2: TBitBtn;
    ProcCount: TLabel;
    FnCount: TLabel;
    OutHint: TLabel;
    FilesCount: TLabel;
    Examine: TBitBtn;
    DriveComboBox1: TDriveComboBox;
    DirList1: TDirectoryListBox;
    procedure ExamineClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure DirList1Change(Sender: TObject);
  private
    SList : TStringList;
  public
    { Public declarations }
  end;

var
  LForm: TLForm;

implementation

{$R *.DFM}

procedure TLForm.ExamineClick(Sender: TObject);
var
   procs, fns, files         : integer;
   line, x, y, p, code       : integer;
   T                         : textfile;
   s, buff, buffl   : string;
   ans                       : TSearchRec;

   procedure xy( z : char );  { see if "z" is the terminating char }
   begin
      y := pos( z, buff );
      if y = 0 then y := 255;
      if y < x then x := y;
   end;

   procedure AddIt;   { up to rest of line of name of proc/fn }
   begin
      if length( buff ) = 0 then exit;
      while buff[1] = ' ' do buff := copy( buff, 2, 255 );
```

```
      x := pos( '(', buff );
      if x = 1 then exit;  { don't get confused with type or proc declarations }
      if x = 0 then x := 255;
      xy( ';' );  xy( ':' );  xy( '(' );  xy( '{' );
      str( line:4, s );
      SList.Add( copy( buff, 1, x-1 ) + chr(9)
        + lowercase( copy( ans.Name, 1, pos( '.', ans.Name ) - 1 ) )
        + chr(9) + s );
   end;
begin
   procs := 0;
   fns := 0;
   files := 0;
   OutHint.Visible := false;
   code := FindFirst(DirList1.Directory+'\*.pas', faArchive, ans );
   while code = 0 do
      begin    { analysis of this file }
        try
         AssignFile(T,DirList1.Directory+'\'+ans.Name);
         Reset(T);
         inc(files);
         FilesCount.Caption := IntToStr(files);
         line := 1;
         while not eof(T) do
            begin
               readln(t, buff);
               buffl := Lowercase( buff );
               p := pos('procedure', buffl);
               if p <> 0 then
                  begin
                     inc(procs);
                     ProcCount.Caption := IntToStr(procs);
                     buff := copy(buff, p+10, 255);
                     AddIt;
                  end;
               p := pos('function', buffl);
               if p <> 0 then
                  begin
                     inc(fns);
                     FnCount.Caption := IntToStr(fns);
                     buff := copy(buff, p+9, 255);
                     AddIt;
                  end;
               inc(line);
            end;
         Application.ProcessMessages;
        finally
        end;
        CloseFile(T);
        code := FindNext( ans );
      end;
   SList.Sort;
   AssignFile( T, DirList1.Directory + '\ListProc.txt' );
   rewrite( T );
   for x := 0 to SList.Count - 1 do writeln( T, SList[x] );
   Closefile( T );
   OutHint.Visible := true;
end;

procedure TLForm.FormCreate(Sender: TObject);
begin
  SList := TStringList.Create;
  SList.Duplicates := dupAccept;
```

```
end;

procedure TLForm.FormDestroy(Sender: TObject);
begin
  SList.Free;
end;

procedure TLForm.DirList1Change(Sender: TObject);
begin
  OutHint.Visible := False;
end;

end.
```

Return to Article

Return to Tips & Tricks

Return to Front Page

Source for Listmain.dfm

```
object LForm: TLForm
  Left = 208
  Top = 111
  Width = 316
  Height = 197
  Caption = 'Procedure/Function Lister'
  Font.Color = clBlack
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  Position = poScreenCenter
  OnCreate = FormCreate
  OnDestroy = FormDestroy
  PixelsPerInch = 96
  TextHeight = 16
  object Label2: TLabel
    Left = 160
    Top = 8
    Width = 85
    Height = 16
    Alignment = taRightJustify
    AutoSize = False
    Caption = 'Procedures:'
  end
  object Label3: TLabel
    Left = 160
    Top = 32
    Width = 85
    Height = 16
    Alignment = taRightJustify
    AutoSize = False
    Caption = 'Functions:'
  end
  object ProcCount: TLabel
    Left = 248
    Top = 8
    Width = 30
    Height = 16
  end
  object FnCount: TLabel
    Left = 248
    Top = 32
    Width = 30
    Height = 16
  end
  object OutHint: TLabel
    Left = 176
    Top = 92
    Width = 109
    Height = 32
    Alignment = taCenter
    Caption = 'Output written to ListProc.TXT'
    Font.Color = clRed
    Font.Height = -13
    Font.Name = 'System'
    Font.Style = []
    ParentFont = False
    Visible = False
    WordWrap = True
```

```
end
object Label1: TLabel
  Left = 160
  Top = 56
  Width = 85
  Height = 16
  Alignment = taRightJustify
  AutoSize = False
  Caption = 'Files:'
end
object FilesCount: TLabel
  Left = 248
  Top = 56
  Width = 30
  Height = 16
end
object BitBtn2: TBitBtn
  Left = 176
  Top = 136
  Width = 89
  Height = 25
  TabOrder = 0
  Kind = bkClose
end
object Examine: TBitBtn
  Left = 32
  Top = 136
  Width = 105
  Height = 25
  Caption = 'Examine'
  TabOrder = 1
  OnClick = ExamineClick
  Glyph.Data = {
    78010000424D78010000000000007600000028000000200000000100000000100
    0400000000000000000000120B0000120B0000000000000000000000000000000
    8000008000000080800080000000800080008080000007F7F7F00BFBFBF000000
    FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFFFF00333333303333
    3333333333FF333333333333000333333333333777F333333333333080333
    3333333F33777FF33F3333B33B000B33B3333373F777773F7333333BBB0B0BBB
    333333737F7F77F333333BBB0F0BBB33333337337373F73F3333BBB0F7F0BB
    B333337F3737F73F7F3333BB0FB7BF0BB3333F737F37F37F73FFBBBB0BF7FB0B
    BBB3773F7F37337F377333BB0FBFBF0BB333337F73F333737F3333BBB0FBF0BB
    B3333373F73FF7337333333BBB000BBB33333337FF777337F333333BBBBBBBBB
    3333333773FF3F773F3333B33BBBBB33B3333373377377373373333333333B3333
    333333333337F33333333333333B333333333333337337333333330000}
  NumGlyphs = 2
end
object DriveComboBox1: TDriveComboBox
  Left = 8
  Top = 8
  Width = 145
  Height = 22
  DirList = DirList1
  TabOrder = 2
end
object DirList1: TDirectoryListBox
  Left = 8
  Top = 32
  Width = 145
  Height = 97
  ItemHeight = 16
  TabOrder = 3
  OnChange = DirList1Change
```

```
  end
end
```

Return to Article

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

Managing Shutdown Procedures with TEndSession

by Benjamin Morin - bem@prolaw.com

In UNDU Issue #17 Grahame Marsh showed us how to a intercept the Win 95 WM_Sizing message. In UNDU Issue #20 he gave us a shell to which we could create our own WndProc function and intercept any windows message (that was passed to the form).
Using the code set forth in his shell code, I created a component that intercepts the WM_QUERYENDSESSION message and allows the application to process an event prior to windows closing the program, and optionally halting the shut down all together.
Notice that I'm replacing WndProc for the Application Handle, not the Form Handle as Mr. Marsh does for his sizing forms.

Code added to the components OnEndSession event will execute when the application receives notice that the user is logging out or shutting down the system. By modifying the value of the ShutDownOK parameter, you can abort the shutdown process.

I'd been working on this for 2 days when I finally came across Mr. Marshs article. Thanks to Mr. Marsh for his help on WndProc.

```
unit tEndSession;

(*                                                  *)
(*  TEndSession - Benjamin Morin (bem@prolaw.com)   *)
(*                                                  *)

interface

uses
  Messages, SysUtils, Classes, Forms, Windows;

type
  TShutDownEvent = procedure (Sender: TObject; var ShutDownOK: boolean) of object;

type
  TtEndSession = class(TComponent)
  private
    FApp: THandle;
    FParent: THandle;
    FOldWndProc: pointer;
    FNewWndProc: pointer;
    sOnEndSession: TShutDownEvent;
    ShutDownOK: boolean;
    procedure NewWndProc(var M: TMessage);
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure Loaded; override;
  published
    property OnEndSession: TShutDownEvent read sOnEndSession write sOnEndSession;
end;
```

```
procedure Register;

implementation

constructor TEndSession.Create (AOwner : TComponent);
begin
  inherited Create(AOwner);
  ShutDownOK := TRUE;
  FApp := Application.Handle;
  FParent := (AOwner as TForm).Handle;
  FNewWndProc := MakeObjectInstance(NewWndProc);
end;

destructor TEndSession.Destroy;
begin
  SetWindowLong(FApp, GWL_WndProc, longint(FOldWndProc));
  FreeObjectInstance(FNewWndProc);
  inherited Destroy;
end;

procedure TEndSession.Loaded;
begin
  inherited Loaded;
  FOldWndProc := pointer(SetWindowLong(FApp, GWL_WndProc,longint(FNewWndProc)));
end;

procedure TEndSession.NewWndProc(var M: TMessage);
begin
  with M do
  begin
    if (Msg=WM_QUERYENDSESSION) then
    begin
      if Assigned(sOnEndSession) then sOnEndSession(Self,ShutDownOK);
      if ShutDownOK then
        Result := CallWindowProc(FOldWndProc, FApp, Msg, wParam,lParam)
      else
        Result := 0;
    end
    else
      Result := CallWindowProc(FOldWndProc, FApp, Msg, wParam,lParam);
  end;
end;

procedure Register;
begin
  RegisterComponents('Samples', [TEndSession]);
end;

end.
```

Return to Component Cookbook

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

Excel OLE Tips for Everyone:

by Joselito Real - reajos@kinwticsys.com

I mainly use Excel to dump the output of my database reports because of the very good printer output that I can get, and besides, my clients want all the output in Excel so that they can tweak the reports a little bit and still print it or reformat according to their whims.

To make my tips work, you must have at least Excel version 7, I haven't tried the earlier versions of Excel. I have been using OLE to drive Excel at a very acceptable speed for my clients, of course, if I know ALL about the Excel Type of Clipboard format to dump to Excel, that is the fastest way to go and dump my output there. However, OLE is here, and after digging into it, here are some things that I have uncovered by myself:

An example of program flow might be:

```
Procedure ExcelOut;
  var  XL:variant;
begin
 //start Excel by creating an instance of Excel Application
 //add a workbook where you dump your data
 //do your data dump routines
 //control Excel's cell format
 //control Excel's page setup
 //make Excel visible
 //tell excel to preview/print your report
 //tell Excel to save your output
 //quit and free the instance of Excel
end;
```

Here are some details:

```
// start Excel by creating an instance of Excel Application
XL:=CreateOLEObject('Excel.Application');

// add a workbook where you dump your data
XL.WorkBooks.add;
```

You must include declaration of variant variables to facilitate your data dump, for example, included in the **var** statement following procedure ExcelOut;

```
// do your data dump routines
Var XL, XArr: Variant;
```

then after the begin statement you may wish to create XArr, ie.,

```
XArr:=VarArrayCreate([1,10],varVariant);
```

the array's purpose in our example is to dump 10 cells at a time per OLE call to Excel. Then load your data to XArr, for example:

```
XArr[1]:=12.85;
XArr[2]:='Yes';
XArr[3]:='California';
```

```
  XArr[4]:=Table1.FieldByName('FIRSTNAME').AsString;
  XArr[5]:='';
  XArr[6]:=i;   //i must have been predeclared elsewhere
  XArr[7]:=TempStr;  //TempStr is a string predeclared elsewhere
  XArr[8]:=AVG;  //AVG is a predeclared real number
  XArr[9]:=STD;  //STD is a predeclared real number
  XArr[10]:=j;  //j is a predeclared integer
```

of course, it is always better to load the array using a for-next loop instead of the brute-force illustration above. A sample of such application is to transfer a record of a database table to an Excel row, ie.:

```
  XArr:=VarArrayCreate([1,Table1.FieldDefs.Count],varVariant);
  for i:=1 to Table1.FieldDefs.Count do
    XArr[i]:=Table1.Fields[i-1]; //first field of a table is 0
                                 //FieldDefs.Count is no. of fields
```

whatever method you choose, you then dump XArr unto Excel for example, we want to dump XArr to first row and from columns A to J:

```
  XL.Range('A1:J1').Value:=XArr;
```

another programmatic approach may be to define RowRange as string and then assign string values to RowRange controlled by our program, ie.:

```
  RowRange:='A'+IntToStr(j)+':'+CHR(64+Table1.FieldDefs.Count)+IntToStr(j);
  XL.Range(RowRange).Value:=XArr;
```

of course, as long as the number of fields do not go over 26 columns for the above example, where j in the above column is the desired row number in the excel spreadsheet. You may also dump a 2-dimensional array at a time but please avoid dumping one cell at a time because OLE process could become sooo slooowwww!

Assuming you have dumped your data, you may wish to format them, like add bold lines, thick lines, adjust to correct cell width, and here are the list of Excel OLE commands from within Delphi:

To format a cell or group of cells, you must select it first:

```
 //control Excel's cell format
 XL.Range('A1:J25').Select;
```

To select the entire spreadsheet cells use:

```
 XL.cells.select;
```

Then apply the format commands:

```
 XL.Selection.Font.Name:='Arial Cyr';
 XL.Selection.Font.Size:=9;
 XL.selection.Columns.AutoFit;
```

And here some other detailed cell formatting commands: These commands will generate a border for the selected cells using thin lines. For other types of lines, you may email me on how to get them

```
      XL.Selection.Borders(xlLeft).Weight := xlThin;
      XL.Selection.Borders(xlRight).Weight := xlThin;
      XL.Selection.Borders(xlTop).Weight := xlThin;
      XL.Selection.Borders(xlBottom).Weight := xlThin;
```

For these detailed commands to work, you must predeclare:

```
const
  xlLeft=-4131;
  xlRight=-4152;
  xlTop=-4160;
  xlBottom=-4107;
```

```
xlThin=2;
xlHairline=1;
xlNone=-4142;
xlAutomatic=-4105;
```

where did I get the values for these constants? email me!

```
//control Excel's page setup
XL.ActiveSheet.PageSetup.PrintTitleRows := 'A1:J1'; //Repeat this row/page
XL.ActiveSheet.PageSetup.LeftMargin:=18;   //0.25" Left Margin
XL.ActiveSheet.PageSetup.RightMargin:=18;  //0.25"  will vary between printers
XL.ActiveSheet.PageSetup.TopMargin:=36;    //0.5"
XL.ActiveSheet.PageSetup.BottomMargin:=36; //0.5"
XL.ActiveSheet.PageSetup.HeaderMargin:=18; //0.25"
XL.ActiveSheet.PageSetup.FooterMargin:=18; //0.25" Footer Margin
XL.ActiveSheet.PageSetup.CenterHorizontally:=1; //zero, means not centered
XL.ActiveSheet.PageSetup.Orientation:=2; //landscape=2, portrait=1

//make Excel visible
XL.visible:=true;

//tell excel to preview/print your report
XL.ActiveSheet.PrintPreview;  //for previewing
XL.ActiveWindow.SelectedSheets.PrintOut (Copies := 1); //print directly

//tell Excel to save your output
XL.ActiveWorkBook.SaveAs ('MyOutput');
```

you can also save your output in other formats

```
//quit and free the instance of Excel
XL.visible:=False;
XL.quit;
XL:=unassigned;
```

How Did I Dig Out These Commands?

Well, it is easy, just run your Excel application, record a macro, and then do what you want, like page setup, open file, save file, sort, etc,... then stop recording the macro, then print your recorded macro, LO and behold!

All the commands are exposed for your perusal in Delphi, just tweak them to conform to Delphi Pascal's syntax.

My Best Tip for the Fastest Transfer of Database Data to Excel Using OLE:

Batchmove your query results, paradox tables, or sections of your large dBASE Tables into a temporary Table having a format of dBASEIV or earlier format and then using OLE, control Excel to convert the table instantly into Excel format or print the Table in Excel by opening your temporary table as a dBASE file.

For example, Temp.DBF is a dBASEIV formatted file produced as a result of a TBatchMove procedure from a query. To start the process of conversion to Excel file do the following:

```
Procedure ExcelOut;
  var  XL:variant;
begin
  XL:=CreateOLEObject('Excel.Application');
  XL.workbooks.open ('\Temp.DBF');  //supply the directory path if needed
  XL.ActiveWorkBook.SaveAs (Filename := '\MyTable', FileFormat :=  -4143);
  //the above line saves whatever loaded file as MyTable.XLS
  //at this point, conversion to Excel File is done, no need for further code
  //except for cleaning up the instance of Excel Application
  //but say, you want a nice printed output, add the following codes:
  XL.Cells.select; //Select all Cells and prepare for format
```

```
    XL.Selection.Font.Name:='Arial';
    XL.Selection.Font.Size:=9;
    XL.selection.Columns.AutoFit;
    XL.ActiveSheet.PageSetup.PrintTitleRows := '$1:$1'; //repeat column headings/page
    XL.ActiveSheet.PageSetup.PrintGridlines := 1;       //print with grid lines
    XL.ActiveWindow.SelectedSheets.PrintOut (Copies := 1); //print directly to
    //House-cleaning is required
    XL.quit;
    XL:=unassigned;
  end;
```

The above example, is the fastest so far (less than 10 seconds on my Pentium 133 MHz machine to transfer 15 Fields by 9,656 records!) to convert your medium to large dBASE tables into Excel using only the current OLE technology but without using those expensive conversion DLL's or OCX's, and without using those QuickReports and other fancy printer formatting tools!

Well I hope you have some fun in using my tips! You can even build a faster OLEExcel component using my tips! Should you find my tip useful, kindly post my dream component in the appropriate places:

One of my wishes is really that if someone out there could write an XLDBGrid or XLStringGrid component:

This component should have the ease of use of cut and paste type of data entry for repetitive data as you would with an Excel spreadsheet, the use of ctrl-C, ctrl-V, Ctrl-X, direct cut and paste to/from an open Excel spreadsheet onto these Grid components are allowed, it should also facilitate the ease of inserting/deleting cells or entire rows or columns, it should have the same way of selecting the group of cells using a mouse or shift keys. No need to include the formula computations. You see, almost everybody who have some exposure to computers knows how to enter data in an Excel spreadsheet without the need for further training. Let them enter Tabular data in a DBGrid... AAARRRRGH!

The Paradox style DBGrid in Delphi is such a pain to enter your data, for example all the other columns are correct, except that there is one item you need to insert into that one-column and everything will be aligned correctly--you don't need to reenter or type over those other entries--just move them automatically after inserting an item! The same is true for the DBEdits, they are not suited for tabular types of data. Most receipts and scientific data are tabular, and you enter them in a tabular format, with the ease and convenience of an Excel or Lotus spreadsheet! If such a component is available, without me doing the code for those mouse and key controls, and without digging those Microsoft Clipboard Excel format, it would be a very nice front end for tabular types of data entry all controlled within Delphi. Yes there is a Formula One OCX, that is available but it is an overkill! and besides, you have to intercept or redefine the way it is handling the clipboard. Lots of work to do to emulate it to behave like an Excel-style of data entry and the OCX that ships together with your program is a big file. So please, if somebody out there who has this type of StringGrid or DBGrid, I'd gladly buy that component. For me, it would be an indispensable data entry front-end.

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

Playing Sounds Asynchronously From Resources - Revisited

by Alan G. Lloyd - AlanGLLoyd@aol.com

In UNDU number 12, Adrian Bottoms described how to incorporate WAV files in resources, and how to play them from the resource. He pointed out that if the sound was played synchronously then everything stopped until it had finished. However if the sound was played asynchronously then the program continued after the Play Sound call and promptly unloaded the resource while it was playing (causing big red button time interrupt). Also only one sound could be played asynchronously, requesting another sound to play asynchronously while the previous one was so playing, caused the previous sound to abort.
One way round the first part of the problem is to load all the resources in memory at form creation, lock them, and leave them locked until the program finishes. This has problems. Firstly the operating system gets slow as it tries to re-arrange the memory it needs around the locked regions. Secondly, with a large number of sound resources, a large amount of memory is used holding resources which are rarely used..

The approach I started with was to load resources and lock them until either another resource is needed, or until a defined time after the use of the resource.

The original scheme then looks something like this:

1. Find resource
2. Load resource
3. Lock resource
4. Play resource
5. Unlock resource
6. Free resource

And the new scheme would look something like this:

1. If resource is locked then
2. Stop playing old resource (in case it is playing)
3. Unlock old resource
4. Free old resource
5. Find new resource
6. Load new resource
7. Lock new resource
8. Set ResourceIsLocked flag
9. Play new resource asynchronously

However this could leave a resource locked for a long time, so a timer (which must be longer than the time to play the longest sound) is set to free the resource when the timer fires. So the final scheme looks

like:

1. Clear old resource
2. Find new resource
3. Load new resource
4. Lock new resource
5. Set ResourceIsLocked flag
6. Start timer
7. Play new resource asynchronously

And the ClearOldResource procedure would then look like:

1. If ResourceIsLocked flag is set then
2. Disable timer
3. Stop playing previous resource
4. Unlock old resource
5. Free old resource
6. Clear ResourceIsLocked flag

The brain then started working, and I realized that if I knew the time the sound took to play, I could start a timer and leave the resource locked until it had finished . The next step was to put the desired sounds into a StringList and play each sound (asynchronously) when the previous sound had finished. I thus had the ability to have a list of sounds played consecutively but asynchronously. Finally, bytes, 6., 5, and 4 of the .WAV file give the (triple byte) size of the WAVE file (resource) data. Byte 24 is $11, $22, or $44 to indicate the type of WAVE data (11kb/s mono, 22Kb/s mono or 44kb/s stereo). (I may not be fully correct on this latter info.)

Playing the sound stringlist is:

```
while stringlist.count > 0
       if resource_in_use flag is clear then
              PlaySound(top sound in stringlist)
              delete top sound in stringlist
        end
       ProcessMessages {lets other events continue while in this while loop}
end
```

PlaySound is:

```
find  new resource
load new resource {handle <> 0 = resource_in_use flag}
lock new resource
get resource data length
resource time = length divided by appropriate data rate
set timer interval
start timer
play new resource asynchronously
```

TimerTimer is:

```
If resource_in_use flag is set then
disable timer
play a sound asynchronously with a nil resource pointer
{stops any sound playing in case play timer is too short}
unlock old resource
       free old resource
clear resource_in_use flag
```

Adding additional sound to the list is:

```
stringlist.add(sound name), which adds the name to the bottom of the list.
```

The hRes handle becomes a convenient flag to be > 0 if a resource is in use. Note that I have not put any exception protection or handling in the program (well what do you expect for a quickie !!).

Note that the Win API call FindResource(ResInfoHandle, ResName, ResType) needs a PChar for the ResType. This has the same string as you use as the resource type in your .RC file. A number of type constants are pre-defined by Win API using RT_ plus the resource type in the file. I have continued that format nominating RT_WAVE as the constant containing WAVE.

I have USEd MMSYSTEM in Delphi 1, but the program could be changed to make Win API calls in Delphi 2.

The test program enables you to play a number of sounds asynchronously, while doing some other task (making a shopping list of course…) I got into playing sounds while building a program to help my grand-daughter tell the time (hence the sounds in the test program).

Source for Async Sounds
Return to Tips & Tricks
Return to Front Page

Object Creation Tip

by Mike Yui - mikeyui@netcom.ca

This may be obvious to those innately savvy programmers out there, but it took a while for this one to dawn on me (a "while" equals just over 2 years of Delphi programming)
Instead of creating objects by explicitly declaring a variable in the var section, you can create much simpler code simply by using the **with** statement. For example, instead of:

```
var
  MyForm: TForm;
begin
  MyForm := TForm.Create(Self);
  with MyForm do
    try
      Show;
    finally
      Free;
    end;
end;
```

you could simply write:

```
begin
  with TForm.Create(Self) do
    try
      Show;
    finally
      Free;
    end;
end;
```

Personally I think the latter example, is simpler and easier to read, not to mention more elegant; thus affirming my long-standing belief that there is elegance in simplicity.

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

Video Capture in Delphi

by G.L. Alston

About a year ago I was strapped into service doing a specialized video capture application. The tool I chose to use was my (then new) Delphi 1.0 compiler, mostly because I know that it could be done in C/C++ and I wanted to find out for myself if Delphi was as fast as Borland claimed it was. If Delphi could handle real-time stuff like video capture, the sky was the limit!

Video capture isn't as exotic as you might think. In fact, it's really pretty simple to do, thanks to AVICAP.DLL. This DLL ships with Win95 and is a video capture engine. All you need to do to capture video is create a capture window and then do some message sending, and AVICAP handles the hard stuff. That's not to say that you have no control, either; AVICAP provides all sorts of control options via settings and callback functions. Essentially, your app talks to AVICAP, which in turn talks to the driver, which in turn does all of the talking to the hardware. The mechanism for the app talking to AVICAP is not the prettiest in the world, however: it uses user-defined message numbers and requires the SendMessage() API call. I changed the constants in my code to WM_USER + N so that you could get a better idea of what it looks like.

My primary source of information was Microsoft's Win32 SDK which assumes C code, so there was a bit of translating to do. The example code is Delphi 1.0 and is culled from my working app. It should translate to 32 bits with no problems.

OK, to start, you need to create a structure that will hold the set of capture control parameters that AVICAP uses. The "capturerec" record is the Delphi equivalent of what I found in the SDK docs. I left the names the same as the C names.

AVICAP basically grabs video frames and streams them to a disk file. You can specify the name of this file. If you don't, it will create CAPTURE.AVI on the root directory of the drive it's running on.

The first thing that needs to be done is to create a capture window, which is done by CreateCapWin. First, you need to create a window and give it a name.

```
hWndC := capCreateCaptureWindow(captit,WS_CHILD or WS_VISIBLE,
                                { The next 4 items are coordinates)
                                0, 0,       { upper left }
                                320, 240,   { height / width}
                                main.handle, 0);

{ Tell AVICAP to connect to the Capture driver. }
smreturn := SendMessage(hWndC, WM_USER + 10, 0, 0);
```

The position coordinates simply outline the location and size of the preview window, although for the best looking results the window sizes ought to be the same as the video being captured. More on that later.

The last parameter specifies which capture device. In my case I knew that the capture board was an Intel ISVR Pro, and the driver for it doesn't do screen captures. On the other hand, 0 on a Video Blaster is used for screen capture and 1 is used for video capture. The Win32 SDK docs have some examples that will help you figure this out dynamically if you need to. This same number must also be sent to the driver via the SendMessage() call that does the driver connect using the wParam (third) parameter. A Video Blaster implementation would look like this:

```
smreturn := SendMessage(hWndC, WM_USER + 10, 1, 0);
```

Note that the code also specifies "main.handle", which is the hWnd that your preview window will show up in. You will need to rename this to your main form's handle (or any other window handle) accordingly.

After that, the rest is mostly a matter of telling AVICAP exactly what to do. To start, you need to retrieve the current capture parameters AVICAP is using so that you can initialize the capturerec record. You could just create a record from scratch, but since you probably are not going to fill in every field, it's safest to let AVICAP fill in the defaults for you.

It turns out that AVICAP is quite flexible. For instance, note that in the code we're specifying a preview window that is showing video at 30 frames per second. The frames per second rate is variable, with 30 being the maximum. Typical frame rates are 15, 24 and 30.

In the app I wrote, audio wasn't important, so this was disabled. The app needed to capture 30 fps at (MAX) 320x200, which is really hard on the hardware, so I elected to not capture audio since this steals some clock cycles. If you decide to capture audio, you will probably have to reduce the capture frame rate, the size of the capture window, or both -- unless you have top notch hardware.

There are a group of parameters that in my case were left to their default states, mostly because they didn't have any effect on what I was doing. You can look the descriptions up in the SDK and decide if these will effect you. They probably won't. After writing the modified capturerec back to AVICAP, all you have to do is specify any callbacks and it's pretty much ready to go.

Callbacks:

AVICAP has the ability to let your program know when a video frame (or an audio frame if you're also capturing audio) has been captured, and if your system is fast enough, do something to the frame before it is streamed to the disk. It uses a callback to do this. A callback function is one in YOUR program that is called by Windows. Essentially you can think of a callback as a way to extend something that resides in another program. For instance, if you wanted to have a visible time and date stamp on the captured video, if your hardware is fast enough (or the capture speed is slow enough, etc.) you could get into the video data (it's basically a bitmap), add the stamp, and restore the video data in the proper format before finishing the function. As you could guess, it's important to make sure that the callback can execute cleanly between the times that it is called.

The method for specifying callbacks with AVICAP is a little ugly, which essentially is the use of the 32 Bit lParam parameter of SendMessage. What we have to do is get the function address using the addr() function, typecast this to a longint, and then include this in the message:

```
{ tell AVICAP where to find the framecount callback }
SendMessage(hWndC, WM_USER + 6, 0, longint(addr(StreamCallback)));
```

Now for every video frame AVICAP will call the StreamCallback function by referencing its address. In the example code, StreamCallback does little more than count up the number of frames that have been processed, and if you want to play with the code (and test callbacks in general) to cause an autoabort of the capture process, all you have to do is set the framestopcount variable to a number > 1. In the app this code is pulled from, I was looking at some hardware for a specific condition, which if met, would cause the process to stop.

The procedures ShowVideoParamsWindow and ShowCompressionWindow aren't required for capturing but serve to illustrate the relationship of your code, AVICAP, and the driver for the hardware. These procedures cause dialogs to appear that modify the capture, such as which Codec is used, how many colors are captured, etc. Essentially these functions are little more than wrappers for SendMessage() calls; they tell AVICAP to tell the driver that we need the dialogs. The important thing here is that these dialogs are supplied by the driver and will vary with the hardware.

The procedures ShowCapWin, HideCapWin and KillCapWin are wrappers for the window management API calls. Lastly, CreateVideo serves as a small function that acts as a main() for the purposes of showing what the final app would tend to look like. All you need to do capture a video is (of course) send a message! In sum, video capture is a simple process:

a) set up a capture window
b) initialize AVICAP
c) tell AVICAP to capture video.

Other things to consider when capturing video mainly depend on your hardware. For instance, the largest video you can currently capture with consumer grade hardware is 320x200 at 30 frames/sec. And believe it or not, only the Intel ISVR PRO board can do this reliably. I tested a lot of different boards, and the frame dropout rate was not good at all, and this was on a moderate speed Pentium. The reason the Intel board is so much more reliable is because it is fine-tuned to run the INDEO Codec and use your main processor. Even then, it's not perfect. At 30 frames per second, you can expect to drop frames depending on what is happening in the picture. Too much movement, and you'll lose frames. We're a long way away from being perfect, but if you're needing to do multimedia work or games, the ISVR board is about $250 or so and you can hook it up to a standard camcorder.

By the way, Delphi 1.0 did everything either at the same speed as a C app or faster. The Win32 SDK provides benchmarks which I compiled using Borland's BC++ 4.5, and the Delphi app offered equivalent performance. Apparently Borland wasn't kidding.

OK, that about sums it all up. I hope I haven't forgotten anything really important. If I did, or you'd like further information, you can contact me via the ASL website at http://www.siteit.com/asl. (Currently I'm in the process of getting a different ISP for my email otherwise I'd give you my email address. The web site will always point to the current one!)

Note: ISVR PRO is a trademark of Intel Corp. The author is not affiliated in any way with Intel Corp. or any subsidiaries.

Source for Video Capture

Return to Front Page

Up To Date Delphi 2 Books

by Ken Dowling - Ken.Dowling@earthlight.co.nz

Having trouble deciding which Delphi 2 book to buy?
You can quickly weed out books that were hastily written prior to the commercial release of Delphi 2 by checking their index. If there is no information on the database verbs **LOCATE** and **LOOKUP**, then keep looking.

Return to Tips & Tricks
Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

Gutters in Delphi 1 & 2

by Dr. Paul Furbacher - pf@bilbo.bio.purdue.edu

I am sure this has happened to you… All too often, I would accidentally set a breakpoint if I used the mouse to place the insertion point at the beginning of a line in the source-code editor. Of course, I could use the "Home" key to move the insertion point to the beginning of a line, but there are times when I get lazy and use the mouse.

I had wished for some kind of indicator to visually tell me where it's okay to click without setting that breakpoint -- a line or a shaded region, perhaps. After some thought, I realized that there is a quick and easy solution -- just show and set the right margin to zero in the "Environment Options | Display" dialog. (I had not found a good use for the right margin line up to this point: I suppose you could use it as a hint for breaking long lines. But that's only for printing purposes, and I don't do a lot of printing, if any, of my source code from Delphi or C++ Builder.)

When the right margin is set to zero, the faint gray line is drawn to very left of the first character in a line. A breakpoint isn't set unless you click about halfway into the region to the left of the margin line, so there's some room for error while clicking. This has virtually eliminated the accidental setting of breakpoints.

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

Bitmaps on StringGrids?

by Mark DeBelder - pekari@glo.be

In issue #19 Steven Gill had a questions that I wanted to present a solution for. The question was: *I am trying to work out how to add bitmaps to StringGrids. I want to use the first column as a status column with graphics indicating the status. Whats a simple way to do this?*

Im not really an expert but I think that StringsGrids are there for Strings and DrawGrids for Bitmaps. Ive seen samples on the shareware market that combine these two components, but it seems expensive to buy these when you already have the above 2 components available. I admit that the routines below are not a solution to all Steves problems, but may be it will start him off in the right direction.

To illustrate the solution Ive created an array with some items Id like to display. The first column in this array would be made visible with a red, yellow or green light (a bitmap); the other three columns are plain text and fit perfectly in a StringGrid:

```
type
  EatRecord = record
                Status : Byte;
                Who    : String;
                What   : String;
                Result : String;
              end;

const
  EatArray : array[1..7] of EatRecord =
    ((Status:0; Who:'Fred Flintstone'; What:'A Steak';         Result:'SeemsOK'),
     (Status:1; Who:'Veggy Saurus';    What:'Eucalyptus Tree'; Result:'Suffocated'),
     (Status:2; Who:'G.E. Hoover';     What:'Bookworm';        Result:'Died'),
     (Status:1; Who:'T. Rex';          What:'Veggy Saurus';    Result:'Escaped'),
     (Status:2; Who:'Tom';             What:'Jerry';           Result:'Got wacked'),
     (Status:0; Who:'B.C. Linton';     What:'Beans in sauce';  Result:'A bit windy'),
     (Status:2; Who:'Tarantula';       What:'Chameleon';       Result:'Tarantula
Eaten'));
```

To display the Status as a bitmap I dropped a DrawGrid on the Form. Ive then adjusted some properties so this DrawGrid does nothing but display a single column. I then dropped a StringGrid on the Form and positioned it to the right side of the DrawGrid and adjusted its properties so that both grids seem to be one grid. It is important to disable the vertical scrollbar of the DrawGrid.

The DrawGrid will be setup something like this:

```
object DrawGrid1: TDrawGrid
    Left = 16
    Top = 16
```

```
    Width = 49
    Height = 152
    ColCount = 1
    DefaultColWidth = 24
    FixedCols = 0
    RowCount = 11
    ScrollBars = ssHorizontal
    TabOrder = 1
    OnDrawCell = DrawGrid1DrawCell
end
```

and the StringGrid like this:

```
object StringGrid1: TStringGrid
    Left = 41
    Top = 16
    Width = 407
    Height = 152
    ColCount = 3
    DefaultColWidth = 128
    FixedCols = 0
    RowCount = 11
    TabOrder = 0
    OnTopLeftChanged = StringGrid1TopLeftChanged
end
```

In the OnCreate method I do some initializing. Put headers in the StringGrid and create a Tbitmap variable.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  { create a bitmap variabel }
  Bmp := TBitMap.Create;
  { put some headers in the Fixed Row of the StringGrid }
  StringGrid1.Cells[0,0] := 'Who ate it?';
  StringGrid1.Cells[1,0] := 'What was eaten?';
  StringGrid1.Cells[2,0] := 'Result?';
end;
```

The OnDestroy method is even simplier with only a Free statement for the bitmap.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  { free the bitmap }
  Bmp.Free;
end;
```

Ive used the OnShow method to copy the contents of the EatArray to the StringGrid.

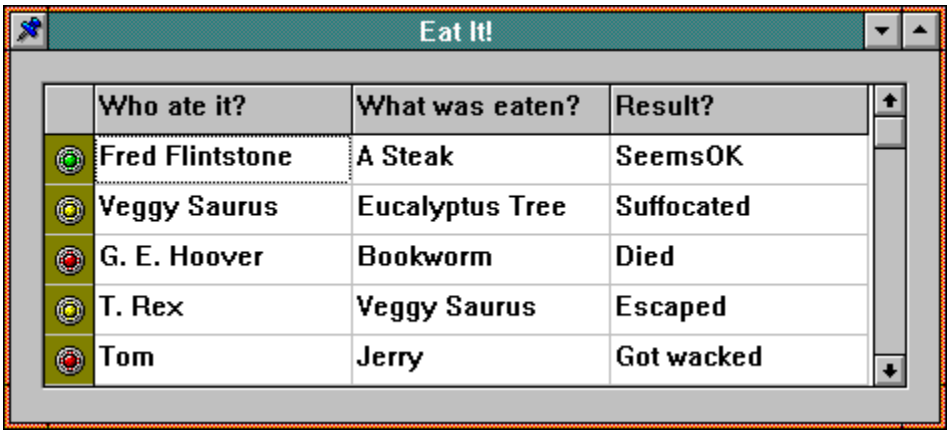
```
procedure TForm1.FormShow(Sender: TObject);
var I: Byte;
begin
  for I := 1 to 7 do
    begin
      StringGrid1.Cells[0, I] := EatArray[I].Who;
      StringGrid1.Cells[1, I] := EatArray[I].What;
      StringGrid1.Cells[2, I] := EatArray[I].Result;
    end;
end;
```

The OnDrawCell of the DrawGrid is not difficult to understand. It will just display bitmaps according to the Status byte in the EatArray. Note that I only do this starting with the TopRow.

```
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; Col, Row: Longint;
```

```
  Rect: TRect; State: TGridDrawState);
begin
  { if the cell is visible load the bitmap according to Status byte }
  if Row >= DrawGrid1.TopRow          (if excluded a GPF is generated}
    then begin
           case EatArray[Row].Status of
             0: Bmp.Handle := LoadBitMap(HInstance, 'LED_GREEN');
             1: Bmp.Handle := LoadBitMap(HInstance, 'LED_YELLOW');
             2: Bmp.Handle := LoadBitMap(HInstance, 'LED_RED');
           end;
           { draw the bitmap on the DrawGrid }
           DrawGrid1.Canvas.Draw(Rect.Left, Rect.Top, Bmp);
         end;
end;
```

If I leave like this I have a nice look-a-like grid with the first column showing a LED-bitmap and columns 2 to 4 plain text. Neat huh?



The last problem that remains to be solved is the scrolling. When you scroll the StringGrid the DrawGrid has to scroll too, equally, of course. This can be achieved with one simple statement in the OnTopLeftChanged method:

```
procedure TForm1.StringGrid1TopLeftChanged(Sender: TObject);
begin
  DrawGrid1.TopRow := StringGrid1.TopRow;
end;
```

This will set the TopRow of DrawGrid equal to the TopRow of StringGrid, so that when you scroll the StringGrid, you will also scroll the DrawGrid.

This solution requires a bit of manipulation at design time to get the two components together. But the code around it is nothing special really.

Return to Front Page

Source for Video Capture

```
unit vidcap;

interface

type
   capturerec = record
     dwRequestMicroSecPerFrame: longint;
     fMakeUserHitOKToCapture: wordbool;
     wPercentDropForError: word;
     fYield: wordbool;
     dwIndexSize: longint;
     wChunkGranularity: word;
     fUsingDOSMemory: wordbool;
     wNumVideoRequested: word;
     fCaptureAudio: wordbool;
     wNumAudioRequested: word;
     vKeyAbort: word;
     fAbortLeftMouse: wordbool;
     fAbortRightMouse: wordbool;
     fLimitEnabled: wordbool;
     wTimeLimit: word;
     fMCIControl: wordbool;
     fStepMCIDevice: wordbool;
     dwMCIStartTime: longint;
     dwMCIStopTime: longint;
     fStepCaptureAt2x: wordbool;
     wStepCaptureAverageFrames: word;
     dwAudioBufferSize: longint;
     fDisableWriteCache: wordbool;
     AVStreamMaster: word;
   end;


procedure InitVariables;
function  CreateVideo: integer;
procedure CreateCapWin;
procedure ShowCapWin;
procedure HideCapWin;
procedure KillCapWin;
procedure ShowVideoParamsWindow;
procedure ShowCompressionWindow;
procedure UserMessage(msg: string);
function  StreamCallback(wnd: THandle; vh: longint): wordbool; export;

var
  framespersec,
  framecount,
  framestopcount:   integer;
  videofilename:    string;
  cparams,
  captureparams:    capturerec;

implementation

{
  this prototype needs to be listed to access the AVICAP function
}
function capCreateCaptureWindow(var lpszWindowName;
    dwStyle: longint; x: integer; y: integer;
```

```
    nWidth: integer; nHeight: integer;
    anhwnd: THandle; nID: integer): THandle; far; external 'avicap';

procedure InitVariables;
begin
  { This is used by the callback to auto-stop after the given
    number of frames have been captured. }

  framestopcount := 0;

  { AVICAP will default to creating the file CAPTURE.AVI in the
    root directory of the drive the program is called from. This
    variable can store a path\name so that you have some better
    control over where it gets captured to. }

  videofilename := 'c:\mydir\testcap.avi';

  { The frame capture rate is dependant on many conditions. These
    are addressed in the article. For now, we'll set it to MAX. }

  framespersec := 30;

end;

procedure CreateVideo;
begin
    InitVariables;
    CreateCapWin;
    { Delete any capture file if it exists }
    if(FileExists(videofilename)) then
       DeleteFile(videofilename);
    { There are no frames captured yet }
    framecount := 0;
    { Tell AVICAP to begin capturing }
    smreturn := SendMessage(hWndC, WM_USER + 62, 0, 0);
    KillCapWin;
end;

{
  Callback from AVICAP.DLL... every frame that gets captured
  generates a function call from AVICAP. In this case we are
  using it strictly to count the number of captured frames.
  This callback gets initialized by CreateCapWin().
}
function StreamCallback(wnd: THandle; vh: longint): wordbool;
begin
    if(framestopcount > 0) then begin
       inc(framecount);     { note the frame number }
       if(framecount > framestopcount ) then
          { Tell AVICAP to abort the operation. }
          SendMessage(hWndC, WM_USER + 69, 0, 0);
    end;
    { Reassure AVICAP that all is OK. }
    result := wordbool(1);
end;

{
  This procedure creates the capture window and initializes
  all of the capture parameters.
}

procedure CreateCapWin;
var
```

```
    capavi: array[0..40] of char;
    captit: array[0..40] of char;
    smreturn:               longint;
    apntr:                  pointer;
    asize:                  integer;
begin
    {
      STEP 1: INIT THE CAPTURE WINDOW AND GET CONNECTED TO
              THE DRIVER
    }
    strpcopy(capavi, videofilename);  { captured video file name }
    strpcopy(captit, 'capture win');  { capture window }
    (*
      SEE THE ARTICLE TEXT ABOUT THE FOLLOWING WINDOW CREATION ROUTINE
    *)
    hWndC := capCreateCaptureWindow ( captit, WS_CHILD or WS_VISIBLE, 0, 0,
                                      320, 240, main.handle, 0);
    ShowWindow(hWndC, SW_SHOW);
    { Tell AVICAP to connect to the Capture driver. }
    smreturn := SendMessage(hWndC, WM_USER + 10, 0, 0);

    if(smreturn <> 0) then begin
       usermessage( 'Connected' );  { feedback }
       { tell AVICAP what the name of the file to capture to is }
       apntr := addr(capavi);
       SendMessage(hWndC, WM_USER + 20, 0, longint(apntr));

       { STEP 2: SET IMAGE PREVIEW UP }
       { Set preview rate at 30 frames per second, in mSec.
         1000 mSec/30 frames = 33 mSec. }
       SendMessage(hWndC, WM_USER + 52, 33, 0);
       { Now go ahead and preview }
       SendMessage(hWndC, WM_USER + 50, 1, 0);

       { STEP 3: INITIALIZE CAPTURE PARAMETERS }
       { First, the capture parameters structure gets initialized
         by asking AVICAP to fill it in for us. }
       apntr := addr(captureparams);
       asize := sizeof(captureparams);
       SendMessage(hWndC, WM_USER + 65, asize, longint(apntr));

       { Then start setting up the preferences: }
       { 1 = capture audio, 0 = disable }
       captureparams.fCaptureAudio := wordbool(0);

       (*
         The time limit params are used to force a stop of the
         video capture at a specified time, just in case
         anything goes wrong. The params here are filled out
         to stop capture automatically after 15 seconds just
         for the sake of illustration.
       *)

       { 1 = enable time limiting, 0 = disable }
       captureparams.fLimitEnabled := wordbool(1);
       { In this case, 15 seconds of video, translated to hex }
       captureparams.wTimeLimit := word($0E);

       { max error rate = 1%. This is somewhat hardware dependant. }
       captureparams.wPercentDropForError := 1;

       { these are the most common frame rates }
       if(framespersec = 30) then
```

```
          captureparams.dwRequestMicroSecPerFrame := 33334  { 30 fps }
       else
          captureparams.dwRequestMicroSecPerFrame := 41667; { 24 fps }

       { 0 = automatic mode, 1 = put up an OK button to initiate  }
       captureparams.fMakeUserHitOKToCapture := wordbool(0);
       { 1 = abort capture on Left button click, 1 = disable  }
       captureparams.fAbortLeftMouse := wordbool(0);
       { 1 = abort capture on Right button click, 1 = disable  }
       captureparams.fAbortRightMouse := wordbool(0);
       { escape key aborts capturing }
       captureparams.vKeyAbort := VK_ESCAPE;

       (*
       These parameters are listed but aren't required for general
       purpose video capture. Refer to the Win32 SDK discussion of
       AVICAP to see if your intended application will be affected.

       captureparams.wChunkGranularity := 0;
       captureparams.dwIndexSize := 0;
       captureparams.fUsingDOSMemory := byte(0);  { don't use DOS memory }
       captureparams.fStepMCIDevice := 0;
       captureparams.fMCIControl := 0;
       captureparams.fStepCaptureAt2x := 0;
       captureparams.fDisableWriteCache := byte(0);
       captureparams.wStepCaptureAverageFrames := 3;
       *)

       { Now write the capture parameters }
       apntr := addr(captureparams);
       asize := sizeof(captureparams);
       SendMessage(hWndC, WM_USER + 64, asize, longint(apntr));

       { tell AVICAP where to find the framecount callback }
       SendMessage(hWndC, WM_USER + 6, 0, longint(addr(StreamCallback)));

    end else begin
       usermessage( 'Invalid video driver.');
       killcapwin; { just to be safe. It *is* windows, after all... }
    end;
end;

{ show the capture window (including the live video) }
procedure ShowCapWin;
begin
   ShowWindow(hWndC, SW_SHOW);
end;

{ hide the capture window }
procedure HideCapWin;
begin
   ShowWindow(hWndC, SW_HIDE);
end;

{ destroy the capture window used by AVICAP }
procedure KillCapWin;
begin
    ShowWindow(hWndC, SW_HIDE);
    DestroyWindow(hWndC);
end;

{ Show the video format dialog. This is supplied by
  the capture driver. All we need to do is tell
```

```
  AVICAP to make the proper call to the driver. }
procedure ShowVideoParamsWindow;
begin
   ShowCapWin;
   SendMessage(hWndC, WM_USER + 41, 0, 0);
   { allow this to happen }
   application.processmessages;
   HideCapWin;
end;

{ Show the video compression options dialog supplied by the
  capture driver. This works like the ShowVideoParamsWindow
  procedure. }
procedure ShowCompressionWindow;
begin
   ShowCapWin;
   SendMessage(hWndC, WM_USER + 46, 0, 0);
   { allow this to happen }
   application.processmessages;
   HideCapWin;
end;

{ display a message to the user }
procedure UserMessage(msg: string);
begin
    {
    use for troubleshooting or your own messages...
    main.messagelabel.caption := msg;
    }
end;

end.
```

Return to Article

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #21 - May 1997

Product Announcement - Addict for Delphi

by Michael Novak

Addictive Software

http://www.flinthills.com/~addict

addict@flinthills.com

Editors Note - Next issue we will have a review of Addict so stay tuned!

Addictive Software is pleased to announce the release of the Addict 2.0 suite for Delphi 1.0 and 2.0. Addict consists of, what we believe to be the most powerful spell check component available for Delphi, as well as the only known thesaurus component for Delphi 1.0 and 2.0.

AddictSpell provides your users with a full featured spell check component to give your applications a more professional look and feel. AddictSpell mirrors, and in some ways exceeds, the functionality of several spell check engines residing in commercial word processors. Following is a brief list of some of the major features of AddictSpell:

- Full Component Source Code: When you order Addict, you will receive the full component source code used to implement AddictSpell.
- Dictionary Compiler: AddictSpell comes complete with an executable utility you can use to create new, royalty free dictionary files to be used with AddictSpell.
- Multiple Main and User Dictionaries Open Simultaneously: Any number of dictionary files and user dictionaries may be open simultaneously, giving new freedom to multi-lingual users.
- Can Use Microsoft Word User Dictionaries. Microsoft Word user dictionaries can be directly imported by your users for use with AddictSpell.
- Auto Correct Feature: Auto-corrections can be added to user dictionaries to enable AddictSpell's ability to automatically correct common spelling errors (i.e. teh=the).
- Checks a Wide Range of Controls: AddictSpell checks a number of controls / data structures, including PChar Buffers, Strings, and any descendant of TCustomEdit, TCustomMemo, and TCustomRichEdit. AddictSpell will also check UDC's MemoWriter components and Turbo Power's Orpheus editing components.
- Extensive Configuration System: AddictSpell's multi-user configuration system gives each user of a multi-user application independent configuration.
- Configuration Dialog: AddictSpell's configuration dialog allows any user of your application to independently specify a multitude of configuration options.
- User Constructed Dialogs: You are free to use the included spell check dialog, or design your own spell check dialog. The default dialog is, in fact, simply a user constructed dialog.
- Non-Modal Dialog: The spell check dialog is not a modal dialog freely allowing users to edit their document in the middle of a spell check. AddictSpell automatically detects this and restarts the check at the cursor position when the user hits the start button.
- Multiple Undo: AddictSpell automatically remembers the positioning and replacement information necessary to allow users to undo multiple spell check actions.
- Selection Avoidance: Initial position and selection avoidance properties prevent the dialog from covering the selected word and provide extensive control of dialog positioning.
- Background Suggestions: Though AddictSpell can generate suggestions in the usual manner, background word suggestions can be continuously generated in the background, eliminating the need for the user to wait on the appearance of suggestions before choosing a course of action.
- Ignore HTML Tags: For those writing HTML compatible applications, this feature allows the AddictSpell to natively ignore HTML tags.
- Fast: AddictSpell's spell check engine is fast and flexible native Delphi code (11,000 words checked per second on a P100 w/24 MB RAM).

Following is a brief list of some of the major features of the thesaurus component:

- Large Thesaurus: The component comes with a compiled version of Roget's 1911 Public Domain Thesaurus, containing over 1000 context topics and 30,000 words.
- Thesaurus Compiler: This version of this component comes with a utility that allows you to compile context-sensitive thesaurus topics of your own into the format used by the component.
- Selection Avoidance: When used in conjunction with an editing control, the thesaurus dialog box will automatically avoid the selected word in the editing control and position the dialog directly under it.
- Fast: Despite the size of the thesaurus file (it is compressed), the component loads in less than a second on a P100 (24 MB RAM). Context topic lookup is near instantaneous.

Stop by our web site to see screen shots, or try out our trial-run version!

Return to Front Page

Source for ASync Sounds

```
unit As_playu; {6 May 1997}
{consecutive aysynchronous playing of WAV resources}

interface

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

type
  TASyncPlayTestF = class(TForm)
    OneBtn: TButton;
    TwoBtn: TButton;
    ThreeBtn: TButton;
    HickDickBtn: TButton;
    ExitBtn: TButton;
    Timer1: TTimer;
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    Label1: TLabel;  {to show Play Now}
    ListBox1: TListBox; {to display Play List}
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    {to do user activities while playing}
    ComboBox1: TComboBox;
    ListBox2: TListBox;
    ClearListBtn: TButton;
    procedure AddSoundClick(Sender: TObject);
    procedure CloseRes;
    procedure ComboBox1Change(Sender: TObject);
    procedure ExitBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure PlaySound(WaveRes : string);
    procedure PlayStack;
    procedure Timer1Timer(Sender: TObject);
    procedure ClearListBtnClick(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

function StrAsPChar(var S : OpenString) : PChar;

var
  ASyncPlayTestF: TASyncPlayTestF;
  hRes : THandle;   {handle to the loaded resource
                     if 0 indicates nothing playing}
  PlayList : TStringList; {holds list of sounds to play}

implementation

{$R *.DFM}
{$R MY_SOUND.RES}
{comments == prefix statements which are for the purpose
```

```
 of Now Playing & Play List display purposes and are not
 necessary for async playing}

procedure TASyncPlayTestF.FormCreate(Sender: TObject);
{sets up display & resource handle}
begin
  hRes := 0; {resource handle and flag for CloseRes}
  Timer1.Interval := 12000;
  {timer interval must be longer than longest sound}
  PlayList := TStringList.Create;
  {==}Label1.Caption := ''
end;

{ ===== This is the nub of the action ====== }
procedure TASyncPlayTestF.PlayStack;
{plays all sounds in the PlayList stringList}
begin
  with PlayList do
    while Count > 0 do begin
      if hRes = 0 then {no resouce is loaded so play next sound}
        begin
        PlaySound(Strings[0]); {play top sound of list ...}
        {==}Label1.Caption := Strings[0];
        Delete(0);             { ... and delete it}
        {==}ListBox1.Items := PlayList;
      end; {if not playing}
      Application.ProcessMessages; {let other processes live}
    end; {while count > 0}
end;

procedure TASyncPlayTestF.PlaySound(WaveRes : string);
{plays a WAV resource - called only if no resource is playing}
var
  PtrRes : PChar;
  hResInfo : THandle;
  ResLength : longint;
  temp, ResTime {mSecs} : integer;
const
  RT_WAVE : PChar = 'WAVE'; {my resource type}
begin
  {find resource}
  hResInfo := FindResource(HInstance, StrAsPChar(WaveRes), RT_WAVE);
  if hResInfo <> 0 then {found the rssource}
    begin
    {load the resource}
    hRes := LoadResource(HInstance, hResInfo);
    if hRes > 32 then {its a good load}
      begin
      {lock the resource}
      PtrRes := LockResource(hRes);
      {calculate sound run time from data in resource}
      ResLength := longint(Byte(Ptr(hRes, 6)^)) * 65536
                 + longint(Byte(Ptr(hRes, 5)^)) * 256
                 + Byte(Ptr(hRes, 4)^);
      case Byte(Ptr(hRes,24)^) of
        $11 : {telephone quality - 8 bit mono 11.025Kb/s}
              temp := ResLength div 11;
        $22 : {radio quality - 8 bit mono 22.050Kb/s}
              temp := ResLength div 22;
        $44 : {CD quality - 16 bit stereo 44.1/sec = 176.4Kb/s}
              temp := ResLength div 176;
      end;
      ResTime := temp - (temp div 440); {=/11.025, 22.050 etc}
```

```
      {set timer}
      Timer1.Interval := ResTime;
      Timer1.Enabled := true;
      sndPlaySound(PtrRes, snd_ASync or snd_Memory); {play sound}
      end
    else
      hRes := 0; {resource could not be found}
    {hRes > 32}
  end; {hResInfo <> 0}
end;

procedure TASyncPlayTestF.Timer1Timer(Sender: TObject);
begin
  CloseRes;
end;

procedure TASyncPlayTestF.CloseRes;
begin
  if hRes <> 0 then {we have a locked resource}
    begin
    Timer1.Enabled := false;  {stop timer}
    sndPlaySound(Nil, snd_ASync or snd_Memory); {stop sound}
    UnlockResource(hRes); {unlock . . .}
    FreeResource(hRes);   { . . . and free reource}
    {==}Label1.Caption := '';
    hRes := 0;  {flag for any later call to CloseRes}
    end;
end;
{ ===== End of the nub of the action ===== }

procedure TASyncPlayTestF.AddSoundClick(Sender: TObject);
{button action to add sound to list}
begin
  if TButton(Sender) = OneBtn then
    PlayList.Add('One');
  if TButton(Sender) = TwoBtn then
    PlayList.Add('Two');
  if TButton(Sender) = ThreeBtn then
    PlayList.Add('Three');
  if TButton(Sender) = HickDickBtn then
    PlayList.Add('Hickory');
  {==}ListBox1.Items := PlayList;
  PlayStack;
end;

procedure TASyncPlayTestF.ClearListBtnClick(Sender: TObject);
{clears list of sounds to play}
begin
  PlayList.Clear;
  {==}ListBox1.Items := PlayList;
end;

procedure TASyncPlayTestF.ExitBtnClick(Sender: TObject);
begin
  CloseRes;
  PlayList.Clear;
  Close;
  {PlayList.Free is in S_Play.DPR}
end;

procedure TASyncPlayTestF.ComboBox1Change(Sender: TObject);
{this is just to do something while it's playing sounds}
begin
```

```
  with ComboBox1 do
    ListBox2.Items.Add(Items[ItemIndex]);
end;

function StrAsPChar(var S : OpenString) : PChar;
{returns a PChar from a string}
begin
  if Length(S) = High(S) then dec(S[0]);
  S[Ord(Length(s)) + 1] := #0;
  Result := @S[1];
end;

end.

(*
Contents of MY_SOUND.RC

  ONE     WAVE    ONE.WAV
  TWO     WAVE    TWO.WAV
  THREE   WAVE    THREE.WAV
  HICKORY WAVE    HICKORY.WAV
*)
```

Source for the DPR File

```
program As_play;

uses
  Forms,
  As_playu in 'AS_PLAYU.PAS' {ASyncPlayTestF};

{$R *.RES}

begin
  Application.CreateForm(TASyncPlayTestF, ASyncPlayTestF);
  Application.Run;
  PlayList.Free; {put here because GPF is caused if put in
                  TASync_PlayTestF.ExitBtnClick()}
end.
```

Return to Article

Return to Tips & Tricks

Return to Front Page