Audio-Enabled Components

by Paul Warren

Have you ever wanted to include sound in your components? Of course we all have at one time or another. Sound is becoming an important part of a polished application.

Delphi provides a `TMediaPlayer` component, on the `System` page, which will play .WAV files. There is also the `sndPlaySound` procedure from the `MMSystem` unit. Both these methods work but have some drawbacks as well.

First, the sound file has to be read from disk. This always causes a perceptible delay to the user. Then you have to distribute the .WAV sound files with your application and check they exist at run time (you never know when a user might delete or move files *[Life would be so much easier without users! Editor]*). Finally you have to write code for file loading and playing the sound files.

It would be nice to store the WaveAudio data right in a component. This would eliminate load time, and the distribution of sound files, by compiling binary sound data into the executable.

In fact, couldn't sound be treated the same way as graphics? I was convinced it could. Before my convictions became reality though, I came to realize that in Delphi nothing is impossible.

Catch 22

Using the component expert I created a `TWavePlayer` skeleton and installed it on my `Misc` component palette page. I added an `FWaveFile` property and immediately realized I had a problem. There is no editor for WaveAudio data. Undaunted, I set out to create one...

In Issue 6 Bob Swart wrote an excellent article on creating property editors and I knew this was exactly what I needed. I quickly created a form similar to Dr.Bob's `ImageForm` and called it `WaveForm`. An editor interface came next in a unit called `WaveEdit`. All I had to do was substitute a `WavePlayer` component for the `DrBobImage` component and... Oops, there *is* no `WavePlayer` component! This is why I was writing an editor in the first place.

Well, this is Delphi after all! Maybe creating a component to create an editor to create the component isn't too outrageous.

```
MemoryStream := TMemoryStream.Create;
MemoryStream.LoadFromFile('S_16_44.WAV');
sndPlaySound(MemoryStream.Memory, SND_ASYNC OR SND_MEMORY);
...
sndPlaySound(nil, 0);
MemoryStream.Free;
```

➤ *Listing 1*

```
object SpeedButton1: TSpeedButton
  Left = 24
  Top = 16
  Width = 25
  Height = 25
  Glyph.Data = {
    78010000424D780100000000000076000000280000002000000010000000100
    0400000000000000000120B0000120B0000000000000000000000000000000
    8000008000000080800080000000800080008080000007F7F7F00BFBFBF000000
    FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFFFF00333333333333
    33333333333FFF3333F33333330003333933333337773F33733333330008033
    933333333737F7F37333333307078733339333337337373F3337333077088803
    33933337F37F337F33733330770888033933333F7337FF37F3733300777008803
    9333377F33773F7F733307088808087333337F7F337F7F7F3FFF070777080873
    99997F7F337F7F7F777708088808087333337F337F737F3F33300888008803
    93333773F377337F73F333308807880339333337F37F337F373F333088077803
    339333373F73F3733373333308777733339333373F7F7F33F37333330807033
    933333333737F73373F33333330003333933333337777333373300000}
  NumGlyphs = 2
end
```

➤ *Listing 2*

Binary Data

WaveAudio files are simply BLOB (binary large object) files. They can be read from disk by the method `TMemoryStream.LoadFromFile`. Listing 1 shows a code fragment I have used before to read a WaveAudio file into memory and call `sndPlaySound`.

Bitmap images are essentially binary data as well, so I started investigating the storage scheme Delphi uses for `TBitMap` types. If you put a `TSpeedButton` on a form and set the `Glyph` property to display a 16x16 bitmap you can copy it and paste a second `TSpeedButton` to the form. The pasted copy also has a copy of the `Glyph` set in the first `TSpeedButton`. How is this done?

Obviously the binary image data is copied along with the `TSpeedButton`. If you use the Windows clipboard viewer you'll see Listing 2. This is the same as the data you see (Figure 1, next page) if you open the form file as a *.dfm.

TWave Class

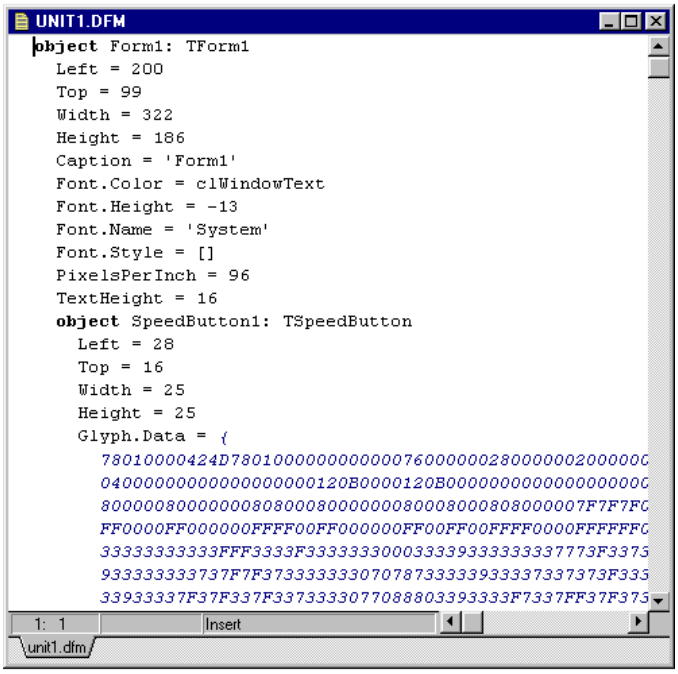
I was pretty sure I could do the same with WaveAudio data. As usual, after some head scratching and forehead wrinkling, out came the VCL source code.

Type `TBitMap` is a descendant of `TGraphic` in the `Graphics` unit. `TGraphic` is a base class implementing a number of methods used by `TBitMap`, `TMetaFile` and `TIcon` to read and write image data to disk files and streams. By assuming these same methods are likely to be used by the component library and form editor I started creating a `TWave` class to duplicate the functionality of `TGraphic`.

TWave would have to descend from TPersistant as TGraphic does. It would need to read and write WaveAudio data from disk and from streams and it would need to know when it had been modified. A field to hold the WaveAudio data is also necessary.

Unlike TGraphic, TWave doesn't need to be a base class (unless you want to store other binary data types similar to WaveAudio data as well). Therefore, there are none of the virtual; abstract; methods seen in TGraphic. The TWave LoadFromStream and SaveToStream methods are much simpler as well because there is no palette, file header, etc.

➤ *Figure 1: Binary glyph data included in a Delphi form file*



➤ *Listing 3*

```
unit Waveplay;
interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
  Graphics, Controls, Forms, Dialogs, MMSystem;
type
  TWave = class(TPersistent)
  private
    FModified: Boolean;
    FWaveData: TMemoryStream;
    FOnChange: TNotifyEvent;
    function GetEmpty: Boolean;
    procedure SetModified(Value: Boolean);
  protected
    procedure ReadData(Stream: TStream); virtual;
    procedure WriteData(Stream: TStream); virtual;
    procedure Changed(Sender: TObject);
    procedure DefineProperties(Filer: TFiler); override;
  public
    constructor Create; virtual;
    destructor Destroy;
    procedure Assign(Source: TPersistent); override;
    procedure LoadFromFile(const Filename: string); virtual;
    procedure SaveToFile(const Filename: string); virtual;
    procedure LoadFromStream(Stream: TStream);
    procedure SaveToStream(Stream: TStream);
    property Empty: Boolean read GetEmpty;
    property Modified: Boolean
      read FModified write SetModified;
    property OnChange: TNotifyEvent
      read FOnChange write FOnChange;
  end;
implementation
constructor TWave.Create;
begin
  inherited Create;
  { create WaveAudio data buffer }
  FWaveData := TMemoryStream.Create;
end;
destructor TWave.Destroy;
begin
  FWaveData.Free;
  inherited Destroy;
end;
function TWave.GetEmpty;
{ returns false if buffer is nil }
begin
  Result := FWaveData = nil;
end;
procedure TWave.Assign(Source: TPersistent);
{ method to copy WaveAudio data when required }
begin
  inherited Assign(Source);
end;
procedure TWave.LoadFromStream(Stream: TStream);
{ method to load WaveAudio data from a stream -
  used by the library and by the LoadFromFile method }
begin
  FWaveData.SetSize(Stream.Size);
  Stream.ReadBuffer(FWaveData.Memory^, Stream.Size);
  Changed(Self);
end;
procedure TWave.SaveToStream(Stream: TStream);
{ method to save WaveAudio data to a stream -
  used by the library and by the SaveToFile method }
begin
  Stream.WriteBuffer(FWaveData.Memory^, FWaveData.Size);
end;
procedure TWave.Changed(Sender: TObject);
{ method to indicate data has changed -
  triggers the OnChange method }
begin
  FModified := True;
  if Assigned(FOnChange) then FOnChange(Self);
end;
procedure TWave.DefineProperties(Filer: TFiler);
{ method to allow "fake" data to be read and
  written by the library }
begin
  Filer.DefineBinaryProperty(
    'Data', ReadData, WriteData, not Empty);
end;
procedure TWave.SetModified(Value: Boolean);
{ method to set modified flag }
begin
  if Value then
    Changed(Self)
  else
    FModified := False;
end;
procedure TWave.LoadFromFile(const Filename: string);
{ method to read data from *.wav file, calls LoadFromStream }
var Stream: TStream;
begin
  Stream := TFileStream.Create(Filename, fmOpenRead);
  try
    LoadFromStream(Stream);
  finally
    Stream.Free;
  end;
end;
procedure TWave.SaveToFile(const Filename: string);
{ method to write data to *.wav file - calls SaveToStream }
var Stream: TStream;
begin
  Stream := TFileStream.Create(Filename, fmCreate);
  try
    SaveToStream(Stream);
  finally
    Stream.Free;
  end;
end;
procedure TWave.ReadData(Stream: TStream);
{ method for library to read data from stream -
  calls LoadFromStream }
begin
  LoadFromStream(Stream);
end;
procedure TWave.WriteData(Stream: TStream);
{ method for library to write data to stream -
  calls SaveToStream }
begin
  SaveToStream(Stream);
end;
```

The DefineProperties, ReadData and WriteData methods are the heart of the TWave class. Delphi's Help says *'The DefineProperties method designates methods for storing an object's unpublished data on a stream such as a form file'*. If you check out the example you will see that TComponent.Left and TComponent.Top are not properties at all. They are stored using DefineProperties.

DefineProperties takes a TFiler as its only parameter. It then calls the Filer's DefineBinaryProperties method. Consulting the Help again we find *'The DefineBinaryProperty method defines binary data the filer object will store as if the data were a property'* and *'the binary property is written directly to a stream object, rather than going through a filer object'*.

So it seems TWave is passed a Filer object by the editor, calls WriteData which in turn calls SaveToStream, which takes care of writing the WaveAudio data to the calling stream using Stream.WriteBuffer. The process is reversed for stream reads. The code for TWave is shown in Listing 3.

TWavePlayer Component

I had to leave TWave untested while I created TWavePlayer. Using the component expert as before I created a TWavePlayer skeleton and gave it an FWave: TWave field, a Wave property and a SetWave procedure stub. Listing 4 shows the initial component skeleton.

TWave Property Editor

I already had my property editor form and interface unit created, minus the TWavePlayer component. To complete the project I needed to install the TWavePlayer component and TWave property editor. Somewhat to my amazement, everything compiled first try. Next, I opened my WaveForm and added a TWavePlayer component. I got a real surprise when I tried to save the form, though. Faster than I could blink I found myself staring at a DOS command prompt. This nasty GPF was caused by a syntax error in TWave.SaveToStream. After correcting this error the WaveForm saved fine. Copying the TWavePlayer to the clipboard confirmed that my WaveAudio data was streaming correctly (see Listing 5).

Putting It All Together

I created a new project and put a TWavePlayer component on the form. I then tried to set the TWave property using my new editor. The first try was a total failure. Absolutely nothing happened.

The override Edit method for my property editor shown in Listing 6 invokes the SetWave procedure of my TWavePlayer. As I would do for a TBitMap, I had used the Assign method to copy the WaveAudio data held in the editor's TWave instance to my component's TWave instance. The best I could figure is the inherited Assign method was inadequate.

```
procedure TPersistent.Assign(
  Source: TPersistent);
begin
  if Source <> nil then
    Source.AssignTo(Self)
  else
    AssignError(nil);
end;
```

The code for TPersistent.Assign was singularly uninformative so I tried some code similar to TBitMap.Assign. A little playing around with the code finally succeeded in copying the WaveAudio data from the editor's TWavePlayer.Wave source to my new WavePlayer1.Wave destination. The final TWave.Assign override and the TWavePlayer.SetWave methods are in Listing 7. Of course all the source code is on this month's disk.

➤ *Listing 4*

```
TWavePlayer = class(TComponent)
private
  FWave: TWave;
  procedure SetWave(AWave: TWave);
protected
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
published
  property Wave: TWave read FWave write SetWave;
end;
implementation
constructor TWavePlayer.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FWave := TWave.Create;   { create class instance }
end;
destructor TWavePlayer.Destroy;
begin
  sndPlaySound(nil, 0);    { must make this call or crash and burn }
  FWave.Free;              { free class instance }
  inherited Destroy;
end;
procedure TWavePlayer.SetWave(AWave: TWave);
{ method to copy WaveAudio data from property editor, or programmatically }
begin
  { copy data from source }
end;
```

➤ *Listing 5*

```
object WavePlayer1: TWavePlayer
  Enabled = True
  Wave.Data = {
    52494646F4050000574156456665D74201000000001000100F82A0000F82A0000
    0100080064617461D005000088486090847094B080404894B888586C88706C9C
    AC78587078686890C094383C9CB4807C90785C608CAC94707C804444A4DC9840
    487C8874789C94604C8090707CA88C444890BC9C545090906080BC985C607884
    ...
    A458609C944C4CA0C0743C78B880487CB0804870A8945868AC903C50B4B45C48
    90A46C5890AC744478AC784C88C07C3864B0A0585CA09C5858A8AC5C54949C60
    5098BC74386CB08C5074AC844468AC985864A098504CA4BC684488B0785484B0
    804870B090547CB4883854ACB0645494A4644C90B06C4888A86C4C88}
  Left = 24
  Top = 20
end
```

What's Next

It shouldn't be too hard to add clipboard support to `TWave` which would make applications using this class capable of exchanging audio with other applications. With clipboard support and the `SaveToFile` method your applications could accept clipboard sounds and write them to disk.

Conclusion

Talk about pulling yourself up by your bootstraps, creating a component to create an editor to create a component. Only in Delphi!

`TWavePlayer` is only the start of what you can do with the `TWave` class. Having a streamable WaveAudio class means your audio is compiled into your executables, no more having to ship .WAV files with your apps. Gone is the noticeable delay while audio files load. And finally, your components can have audio built right in.

On the disk with this issue you will find a `TSndBitBtn` that will play a `TWave` sound in the `Click` method, and a `TImageSnd` that will play sounds in the inherited `OnClick` event. There is also a demo of these two components and a demo of `TWavePlayer`. Everything works fine with Delphi 2 as well as Delphi 1, but you will need to convert the .DCR component bitmap file to 32 bit before installing the component into Delphi 2 *[Use Dr.Bob's RESCONV.EXE program in the CONSTRUC directory of this Issue's disk. Editor]*.

Paul Warren runs HomeGrown Software Development in Langley, British Columbia, Canada and can be contacted by email at hg_soft@uniserve.com

➤ *Listing 6*

```
{ override of the edit method -
  this is the nuts and bolts of the property editor }
procedure TWaveEditor.Edit;
begin
  with TWaveForm.Create(nil) do
  try
    WavePlayer1.Wave := TWave(GetOrdValue);
    if ShowModal = mrOk then
      if (GetPropType^.Name = 'TWave') then
        SetOrdValue(LongInt(WavePlayer1.Wave));
  finally
    Free;
  end
end;
```

➤ *Listing 7*

```
procedure TWave.Assign(Source: TPersistent);
{ method to copy WaveAudio data when required }
begin
  if (Source = nil) or (Source is TWave) then begin
    if Source <> nil then
      FWaveData := TWave(Source).FWaveData;
    Changed(Self);
    Exit;
  end;
  inherited Assign(Source);
end;
procedure TWavePlayer.SetWave(AWave: TWave);
{ method to copy WaveAudio data from property editor,
  or programmatically }
begin
  FWave.Assign(AWave);   { copy data from source }
end;
```