Dynamic Loading And Unloading Of Fonts

by Stewart McSporran

The problem with delivering applications for the Windows environment is that you can never guarantee that any non-standard fonts you use in your application will be available on your users' systems. The traditional solutions are either to give the user the font and tell them to install it (which is messy and unprofessional), or to use one of the installation programs which knows how to install fonts (this is often overkill). A simpler solution is to provide the font file with your application and to automatically install it when your program starts and then remove it when the program ends.

This article describes a unit, DYNFONT.PAS, which allows dynamic loading and unloading of fonts for Delphi 1 and Delphi 2 applications.

Installing A Font

Installing a True Type font in Windows is essentially a three step process:

➢ Create a resource file, with the extension .FOT, based upon the supplied .TTF font file;

➢ Add this resource file to the Windows font table;

➢ Send a `WM_FONTCHANGE` message to the system to inform it that the fonts have changed.

So, how do we create a font resource file based upon the .TTF file? In swoops the API `CreateScalableFontResource` function to our rescue. It needs to be told where the .TTF file is and where the new .FOT file is to be created. If your program is only ever going to be run on a standalone machine then the location of the .FOT file can simply be the directory where your program is installed. However, if you intend to keep your program on a network drive and have it accessed by multiple users then things become more complex.

For `CreateScalableFontResource` to work the network user must have create permission on the directory where the .FOT file will be stored. As it is common practice to only give users read access to application directories we can't assume anything about the location of the .FOT file. Even assuming that the users do have create permission on the application directory still gives problems: if we simply give the .FOT file the same name as the .TTF file then each new user will overwrite the .FOT file created by the previous user (and will delete it when they finish running the program). Experiments with a Netware 3.12 system have shown that this does not appear to affect the running of the program, but such a hit and miss approach to the creation and deletion of files sends shivers down my spine.

The solution to this is to use the Windows API `GetTempFileName` and `GetTempDrive` (for Windows 3.x) or `GetTempPath` (Win32) functions. These will provide us with a path to each user's `TEMP` directory and a unique filename for the .FOT file.

When the font resource has been created the Windows font table is updated by calling `AddFontResource` and passing it the path to the .FOT file. Finally `WM_FONTCHANGE` is sent to notify the system of the changes.

Removing A Font

Subsequent removal of a font is another three step process:

➢ Remove the font resource file from the Windows font table;

➢ Delete the font resource file;

➢ Send a `WM_FONTCHANGE` message to the system to inform it that the fonts have changed.

Removing a previously installed font is done by calling `RemoveFontResource` and passing it the path to the .FOT file, which was returned to us from the earlier call to `CreateScalableFontResource`. This removes the entry from the Windows font table. As the .FOT file is no longer required it is then deleted from the `TEMP` directory. Finally the `WM_FONTCHANGE` message is sent to notify the system of the changes.

DynFont Unit

The `DynFont` unit exports two procedures and an exception type. The exception `EDynFontError` is raised if any problems occur inside its procedures. The two procedures `LoadFonts` and `DeleteFonts` are described below. `DynFont` should be added to the uses clause of the project's .DPR file.

Since, in order to remove a font at a later time, it is necessary to know the names and locations of the .FOT files created by `CreateScalableFontResource`, `DynFont` also contains a private global variable `slFonts` which is a stringlist that stores the names of all the .FOT files. This variable is created in the `initialization` section of the unit and freed by either the `DeleteFonts` procedure for Delphi 1 or the `finalization` section for Delphi 2.

Listing 1 shows the complete source for this unit.

LoadFonts Procedure

The `LoadFonts` procedure is responsible for installing the fonts and should be called from the project file before any forms are created. Its parameter is an array of font file names, without their .TTF extension. For example, the line of code below will load the fonts STRANGE.TTF, WEIRD.TTF and CRAZY.TTF into the system (the fonts should reside in the same directory as the application's executable):

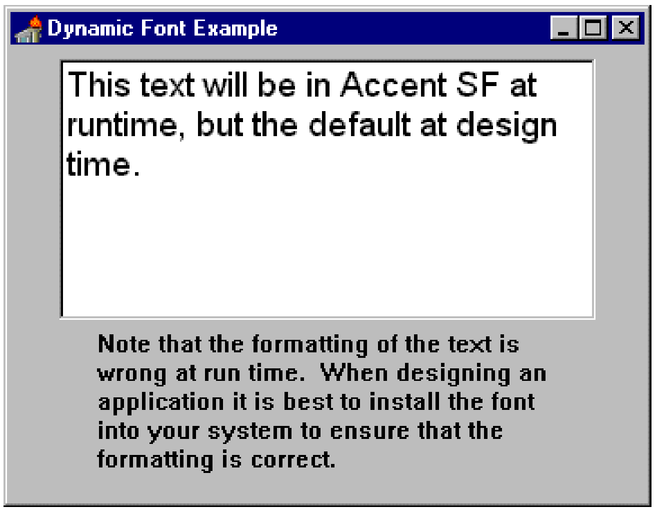
```
LoadFonts(['strange','weird',
  'crazy']);
```

LoadFonts in turn calls LoadFont for each font file you wish to install before finally sending the WM_FONTCHANGE message to notify the system that there have been changes to the font table.

LoadFont is responsible for installing one font into the system. Its single parameter is the name of a font file minus the extension. For example to load MYFONT.TTF we call LoadFont('MyFont'). It expects to find the .TTF file in the same directory as the application's executable file.

There is one amendment you may wish to make to this procedure. The first parameter of CreateScalableFontResource indicates whether the font is only for the use of this application (has the value 1) or may be made available to other applications (has the value 0). The current implementation sets this to 1, since the font may be removed from the system, by exiting this Delphi application, while another application is still using it.

Figure 1: Sample application at design time


Dynamic Font Example

This text will be in Accent SF at runtime, but the default at design time.

Note that the formatting of the text is wrong at run time. When designing an application it is best to install the font into your system to ensure that the formatting is correct.

DeleteFonts Procedure

DeleteFonts removes all the fonts installed by any previous call(s) to LoadFonts. It should be called from the project file, but should come after the line Application.Run in the .DPR project file.

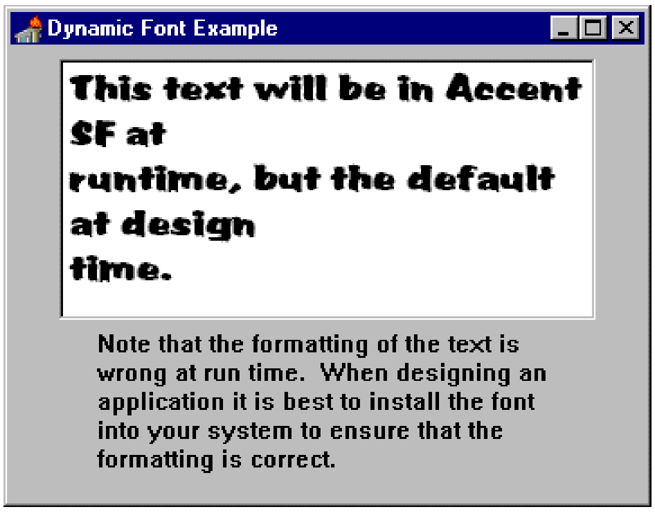
DeleteFonts iterates through all the .FOT files stored in the slFonts stringlist, calling RemoveFontResource for each to remove it from the font table and then deleting the .FOT file. Once it is finished it sends a WM_FONTCHANGE message to inform the system of the changes to the font table.

Where's My TEMP?

The strTempPath function is used by the Delphi 2 implementation to get the location of the TEMP directory. It

Listing 1

```
unit dynfont;
interface
uses SysUtils;
type {Create a customised exception for this process}
  EDynFontError = class(Exception);
procedure LoadFonts(aFontNames : array of string);
procedure DeleteFonts;
implementation
uses
 {$ifdef WIN32} windows,
 {$else} wintypes, winprocs, {$endif}
 messages, classes, forms;
var slFonts : TStringList;
{$ifdef WIN32}
{ Returns the path to the system's TEMP dir (Win32)}
function strTempPath : string;
var
  strTempPath : string;
  nChars : integer;
begin
  SetLength(strTempPath,255); {allocate 255 chars in string}
  { get the temp location }
  nChars := GetTempPath(254,PChar(strTempPath));
  { Check that GetTempPath worked ok }
  if (nChars = 0) or (nChars > 254)  then
    raise EDynFontError.Create(
      'Can not get location of TEMP directory');
  result := strTempPath;
end;
{$endif}
procedure LoadFont(name : string);
var
  pstrFotFile, pstrTmp, pzttf : array [0..250] of char;
  ttf : string;
begin
  {Create a path to this directory & the font file}
  ttf := ExtractFilePath(Application.ExeName) +
    name + '.ttf';
  StrPCopy(pzttf,ttf);
  {We want to create the .fot files in the temp dir}
 {$ifdef WIN32}
  GetTempFileName(PChar(strTempPath), PChar(name),
    0, pstrFotFile);
 {$else}
  GetTempFileName(GetTempDrive('x'),
    StrPCopy(pstrTmp,name), 0, pstrFotFile);
 {$endif}
  {store temp filename in string list to delete it later}
  slFonts.Add(StrPas(pstrFotFile));
  {if the fot file exists then delete it}
  SysUtils.DeleteFile(StrPas(pstrFotFile));
  {Create the fot file}
  if not CreateScalableFontResource(1,
    pstrFotFile, pzttf, nil) then
    raise EDynFontError.Create(
      'Error in CreateScaleableFontResource.'+#10+ttf);
  {Add it to the font table}
  if AddFontResource(pstrFotFile) = 0 then
    raise EDynFontError.Create(
      'Error in AddFontResources ' + name);
end;
procedure LoadFonts(aFontNames : array of string);
var
  i : integer;
begin
  {call loadfont for each item in the array}
  for i:=low(aFontNames) to high(aFontNames) do
    LoadFont(aFontNames[i]);
  {Inform the system that the fonts have changed}
  SendMessage($FFFF,WM_FONTCHANGE,0,0);
end;
procedure DeleteFonts;
var
  pstrTmp : array [0..150] of char;
  i : integer;
begin
  {iterate through FOT files}
  for i:=0 to slFonts.Count - 1 do begin
    {Remove the font from the window's font table
     then delete the tmp file}
    if not RemoveFontResource(
      StrPCopy(pstrTmp,slFonts[i])) then
      raise EDynFontError.Create(
        'Error in RemoveFontResource');
    SysUtils.DeleteFile(slFonts[i]);
  end;
  {Inform the system that the fonts have changed}
  SendMessage($FFFF,WM_FONTCHANGE,0,0);
 {$ifndef WIN32}
  slFonts.Free;  {Finished with the string list}
 {$endif}
end;
initialization
  slFonts := TStringList.Create;
{$ifdef WIN32}
finalization
  slFonts.Free;
{$endif}
end.
```



This text will be in Accent SF at
runtime, but the default at design
time.

Note that the formatting of the text is wrong at run time. When designing an application it is best to install the font into your system to ensure that the formatting is correct.

➤ *Figure 2: Sample application at run time*

shows how the new long string type available in Delphi 2 may be used as both a Pascal and a null terminated string.

Sample Application

The sample application contains the font file ACCE.TTF (font Accent SF). Assuming that you do not already have this font installed in your system then at design time the demo form will look similar to Figure 1. The memo's `Font.Name` property is set to `Accent SF` but as this font is not available the Windows font manager substitutes a default font.

When you run the application the font is installed and the form should look like Figure 2.

Note that the formatting of the text is different at run time to that at design time. When designing an application it is best to install the font into your system to ensure that the formatting is correct.

Running With Delphi 1 And 2

One final point, found out the hard way. If you build a Delphi project in Delphi 1 and then re-compile it with Delphi 2, everything's fine. If you then try to re-compile it with Delphi 1 it won't work. The project's .RES file is converted from 16-bit to 32-bit format by Delphi 2. I found it was best to keep a copy of the 16-bit .RES file when I wanted to test any changes with Delphi 1.

Stewart McSporran works for training and consultancy company Buchanan International in sunny, warm, Glasgow in the UK, and can be contacted as 100753,1703 on CompuServe.