How to store files inside an executable program

Why do it?

Have you ever needed to distribute one or more critical data files with a program? Often only your program needs to access the data file(s) and they don't need to be changed by it. How do we stop users from deleting the files? One answer is to store the data file inside our executable (or in a DLL) as a custom (*RCDATA*) resource and to link the resource into our application using the {$R} directive.

This article shows how to create a resource file containing a copy of any file.

How it's done

The resource file we're going to create has the following format:

- a header that introduces the file
- a header for our *RCDATA* resource
- the data itself - an *RCDATA* resource is simply a sequence of bytes
- any padding required so that any following resource begins on a *DWORD* boundary

It's much simpler to create a resource file that is identified by an ordinal (e.g. 200) than it is to create one identified by a string (e.g. 'MY_RESOURCE'), since the resource header records are a fixed size in the first case and are variable in the second case. We will only consider the first case here. We will also just copy one file into the resource – it's simple to extend this to more than one.

Because we're sticking with ordinal IDs the resource header can be defined as:

```
TResHeader = record
  DataSize: DWORD;        // size of our data
  HeaderSize: DWORD;      // size of this record
  ResType: DWORD;         // lo word = $FFFF => ordinal
  ResId: DWORD;           // lo word = $FFFF => ordinal
  DataVersion: DWORD;     // *
  MemoryFlags: WORD;
  LanguageId: WORD;       // *
  Version: DWORD;         // *
  Characteristics: DWORD; // *
end;
```

Listing 1

We will not be using the fields marked *.

Here's the code that creates the resource file and copies in a given file:

```
procedure CreateResourceFile(
  DataFile, ResFile: string;  // file names
  ResID: Integer              // id of resource
);
var
  FS, RS: TFileStream;
  FileHeader, ResHeader: TResHeader;
  Padding: array[0..SizeOf(DWORD)-1] of Byte;
begin

  { Open input file and create resource file }
  FS := TFileStream.Create(  // to read data file
    DataFile, fmOpenRead);
  RS := TFileStream.Create(  // to write res file
    ResFile, fmCreate);

  { Create res file header - all zeros except
    for HeaderSize, ResType and ResID }
  FillChar(FileHeader, SizeOf(FileHeader), #0);
  FileHeader.HeaderSize := SizeOf(FileHeader);
  FileHeader.ResId := $0000FFFF;
```

```
  FileHeader.ResType := $0000FFFF;

  { Create data header for RC_DATA file
    NOTE: to create more than one resource just
    repeat the following process, using a different
    resource ID each time }
  FillChar(ResHeader, SizeOf(ResHeader), #0);
  ResHeader.HeaderSize := SizeOf(ResHeader);
  // resource id - FFFF says "not a string!"
  ResHeader.ResId := $0000FFFF or (ResId shl 16);
  // resource type - RT_RCDATA (from Windows unit)
  ResHeader.ResType := $0000FFFF
    or (WORD(RT_RCDATA) shl 16);
  // data file size is size of file
  ResHeader.DataSize := FS.Size;
  // set required memory flags
  ResHeader.MemoryFlags := $0030;

  { Write the headers to the resource file }
  RS.WriteBuffer(FileHeader, SizeOf(FileHeader));
  RS.WriteBuffer(ResHeader, SizeOf(ResHeader));

  { Copy the file into the resource }
  RS.CopyFrom(FS, FS.Size);

  { Pad data out to DWORD boundary - any old
    rubbish will do!}
  if FS.Size mod SizeOf(DWORD) <> 0 then
    RS.WriteBuffer(Padding, SizeOf(DWORD) -
      FS.Size mod SizeOf(DWORD));

  { Close the files }
  FS.Free;
  RS.Free;
end;
```

Listing 2

The above code should be sufficient to illustrate the problem, but it is not very elegant – and the streams should be protected by **try .. finally** blocks. A better solution is to create a class that encapsulates the code. A further improvement would be to permit either strings or ordinals to be used to identify the resource.

On occasion you may want to write formatted data to the resource file rather than just copy a file – this is easy to do. You need to do five things:

1. Write a place-holder header record for your resource and record its position in the stream.
2. Write the formatted data to the file (replace the code that copies the file with code that writes the data).
3. Keep a record of the size of the data you are writing.
4. Pad the data out to a *DWORD* boundary.
5. Store the length of data (excluding padding) in your header record, return to the position of the place-holder.

Of course there's now the problem of getting the file information back out of the executable! This is quite a trivial process and is dealt with in *another article*.

Worked Example

You can download a *worked example* that demonstrates what has been described here – it uses the above code. The zip file contains a pair of projects. The first a program that embeds a supplied rich text file in a resource file. The second program includes the resource file and displays the rich text in a rich edit component.

This article is copyright © Peter Johnson 2000-2005

SOME RIGHTS RESERVED

Licensed under a *Creative Commons License*.

Copyright © Peter Johnson (*DelphiDabbler*) 2002-2020