File Handling 6: Text File Device Drivers

by Brian Long

This is the final part of our series on aspects of file handling in Delphi 1 and 2. This month we are looking at these not very well known things called text file device drivers. If you are wondering what they are, then ask yourself these questions. Have you ever used the WinCrt unit in Delphi 1? Or the console mode facility in Delphi 2? Or the AssignPrn procedure from the Printers unit?

If the answer was yes to any of those then you have used a text file device driver (TFDD), although you might not have known it at the time. You have accessed these input and output facilities by using Read/ReadLn and/or Write/WriteLn and the information has come from the keyboard, or gone to the screen in some fashion, or to the printer.

These three TFDDs are accessed in two slightly different ways. The printer support is used just like a normal text file variable, but with AssignPrn being called instead of AssignFile:

```
var T: TextFile;
...
AssignPrn(T);
WriteLn(T, 'Hello world');
```

However the WinCrt and console mode support is accessed without specifying a file variable, eg:

```
var S: String;
...
WriteLn('Hello world');
ReadLn(S);
```

Read/Ln and Write/Ln do always operate on text files, despite this seeming evidence to the contrary. As it turns out, when no file variable is specified, two System unit text file variables, Input and Output, are implicitly used. The above code is exactly the same as writing this:

```
var S: String;
...
WriteLn(Output, 'Hello world');
ReadLn(Input, S);
```

Using this information, you could access the printer without specifying a file variable as follows:

```
AssignPrn(Output);
WriteLn('Hello world');
```

The idea of a TFDD is to allow information to be gathered from, or sent to, some "device" though the Read/Ln and Write/Ln standard procedures via a text file variable, be it an implicitly or explicitly declared one. The nature of the device does not matter, so long as the relevant supporting code to talk to it is present in the TFDD's implementation. In the cases described above, the devices included the keyboard, a Windows GUI window, a Windows console window and a printer.

Bearing in mind that a text file can only be accessed in either read-only mode (using Reset or Append) or write-only mode (using Rewrite), the TFDD can be written to allow any given any file variable read-only or write-only access to the device.

Writing A Text File Device Driver

When implementing a TFDD you need to write functions for five purposes. One of your functions will be called when the device is opened (using Reset, Rewrite or Append), when the device is read from (using Read/Ln), written to (Write/Ln), flushed (implicitly performed after input and output) and closed (CloseFile). The way you set these functions up is by writing a customised AssignFile substitute which assigns the various functions to four function pointers in the text file variable. Even though there are five possible functions, two are mutually exclusive: the reading and writing routines are never used at the same time, so they both get assigned to the same function pointer, called InOutFunc.

Note that the real AssignFile sets up functions in the RTL to deal with opening, reading, writing etc. files - normal text file access is achieved also using the TFDD mechanism where the "device" is a text file on a disk.

TTextRec Record Type

In order to get access to the function pointers in the text file variable, the customised AssignFile procedure must typecast it into a TTextRec record. This type, defined in the SysUtils unit, defines the internal layout of a TextFile variable. The definition differs slightly between Delphi 1 and 2, though not with any important consequences, as shown in Listing 1 (next page).

The Handle data field normally stores the file handle of the represented text file and the Name field holds its file name. Mode tells you the current state of the file: for a text file this can be fmClosed, fmInput or fmOutput – the other possible value (fmInOut) doesn't apply to text files. The Buffer field is the initial text buffer, 128 bytes in size. This buffer is pointed to by BufPtr, although a different buffer can be set up with a call to SetTextBuf. BufSize tells how big the buffer is, BufPos says how far through the data in the buffer we are and BufEnd signifies the end of the valid data in the buffer. UserData is some space unused by the Delphi file system, intended for use by the TFDD writer for storing private information. In Delphi 1, Private is not used and so can be ignored, which leaves the four pointer fields.

Here is a list of the responsibilities of the customised AssignFile

and also for each of the functions referenced by these pointer fields.

AssignFile Substitute Procedure

This should associate all four of the device interface functions to the function pointer fields in a text file variable. However, this guideline is often not followed to the letter. Since the function referred to by the `OpenFunc` function will always be called before any of the others, many `AssignFile` routines only set up `OpenFunc`. They leave all the other pointers for `OpenFunc` to set up. Depending on the value of `Mode`, `OpenFunc` can set up the remaining pointers to either input- or output-based routines. This can save the functions `InOutFunc`, `FlushFunc` and `CloseFunc` from having to check the `Mode` value.

As well as the function pointers, this must also assign `fmClosed` to `Mode`, store the size of whatever buffer is being used in `BufSize`, place the address of the buffer in `BufPtr`, and clear the `Name` null-terminated string field. It can also do any other initialisation required, such as storing some data in `UserData`. One other thing that is sometimes done is to set the `Handle` field (which won't be used in a TFDD) to `$FFFF`.

OpenFunc Function

This is called when the file is opened by `Reset`, `Rewrite` or `Append`. Upon entry, the `Mode` field contains `fmInput`, `fmOutput` or `fmInOut` respectively, as an indicator as to which procedure was called. Any preparation required for input or output, as indicated by `Mode`, can be done. If `Append` was called, and `Mode` therefore has a value of `fmInOut`, it must be changed to `fmOutput`.

InOutFunc Function

This function gets invoked by `Read`, `ReadLn`, `Write`, `WriteLn`, `Eof`, `Eoln`, `SeekEof`, `SeekEoln`, `Close` and Flush (if `Mode` is `fmOutput`) when device input or output is needed.

If `Mode` is `fmInput`, this routine must read up to `BufSize` characters into `BufPtr^`, assign to `BufEnd` the number of characters read and set `BufPos` to zero. If `BufEnd` is zero, `Eof` will be `True` for the file.

If `Mode` is `fmOutput`, this routine should write `BufPos` characters from `BufPtr^` and set `BufPos` to zero.

Delphi 1 TTextRec

```
PTextBuf = ^TTextBuf;
TTextBuf = array[0..127] of Char;
TTextRec = record
  Handle: Word;
  Mode: Word;
  BufSize: Word;
  Private: Word;
  BufPos: Word;
  BufEnd: Word;
  BufPtr: PTextBuf;
  OpenFunc: Pointer;
  InOutFunc: Pointer;
  FlushFunc: Pointer;
  CloseFunc: Pointer;
  UserData: array[1..16] of Byte;
  Name: array[0..79] of Char;
  Buffer: TTextBuf;
end;
```

Delphi 2 TTextRec

```
PTextBuf = ^TTextBuf;
TTextBuf = array[0..127] of Char;
TTextRec = record
  Handle: Integer;
  Mode: Integer;
  BufSize: Cardinal;
  { Note no Private field }
  BufPos: Cardinal;
  BufEnd: Cardinal;
  BufPtr: PChar;
  OpenFunc: Pointer;
  InOutFunc: Pointer;
  FlushFunc: Pointer;
  CloseFunc: Pointer;
  UserData: array[1..32] of Byte;
  Name: array[0..259] of Char;
  Buffer: TTextBuf;
end;
```

➤ *Listing 1*

FlushFunc Function

This routine is called by default at the end of each `Read`, `ReadLn`, `Write` and `WriteLn`. This function can optionally flush the text-file buffer. Note that this is curiously not called by the `Flush` procedure, which instead calls `InOutFunc` if `Mode` is `fmOutput`.

If `Mode` is `fmInput`, this can set `BufPos` and `BufEnd` to zero to flush the remaining unread characters in the buffer. This feature is hardly ever used.

If `Mode` is `fmOutput`, this can behave much like the `InOutFunc` function, thus ensuring that text written to the file variable gets sent to the device immediately. If this routine does nothing, the text won't appear in the device until the buffer becomes full, the file is closed, or the `Flush` procedure is called. It is because this routine can optionally not do anything that `Flush` does not use it.

CloseFunc Function

`CloseFile` invokes this function when closing a text file associated with a device. Additionally, `Reset`, `Rewrite` and `Append` will call this if the file is already open.

If `Mode` is `fmOutput`, then `InOutFunc` will be called before `CloseFunc` to ensure all data is written to the device.

Errors

Each of these four or five functions (not all of which are necessarily required) have the following interface:

```
function DeviceFunc(
  var F: TTextRec): Integer;
```

In Delphi 1, they must be compiled in the `far` call model. This can be done by, amongst other things, declaring them in the `interface` section of a unit, or by placing the `far` keyword after the declaration line.

If the functions return a non-zero value, this signifies an I/O error has occurred. The return value of each function becomes the value that `IOResult` will return, or that may turn up as the `ErrorCode` property of an `EInOutError` exception, depending on the state of the I/O checking option.

An Input/Output Device

Our first look at implementing a TFDD will involve an edit control. The project TFDD1.DPR does this, and the TFDD is in RWEDITU.PAS (see Listing 2). In this case, we are keeping track of which edit control is associated with which file variable by storing the edit's object reference in the `UserData` field. To do this requires a record (`TUserData`) to be defined to use in typecasting `UserData`, which is not defined as an appropriate type. Listing 2 contains one such definition of `TUserData`, which is a variant record. The conditional compilation is required because `UserData`

```
unit Rweditu;
interface
uses
  StdCtrls;
procedure AssignRWEdit(var F: TextFile; E: TEdit);
implementation
uses
  Forms, SysUtils;
const
{$ifdef Win32}
  FillerMax = 32;
{$else}
  FillerMax = 16;
{$endif}
type
  TUserData = packed record
    Edit: TEdit;
    Filler: array[SizeOf(TEdit)+1..FillerMax] of Byte;
  end;
  { TUserData = packed record
    case Byte of
      1: (Edit: TEdit);
      2: (Filler: array[1..FillerMax] of Byte);
  end; }
function RWEditOpen(var F: TTextRec): Integer;
  far; forward;
function RWEditInput(var F: TTextRec): Integer;
  far; forward;
function RWEditOutput(var F: TTextRec): Integer;
  far; forward;
function RWEditClose(var F: TTextRec): Integer;
  far; forward;
procedure AssignRWEdit(var F: TextFile; E: TEdit);
begin
  { Set up text file variable }
  with TTextRec(F) do begin
    Handle := $FFFF;
    OpenFunc := @RWEditOpen;
    Mode := fmClosed;
    BufSize := SizeOf(Buffer);
    BufPtr := @Buffer;
    Name[0] := #0;
    { Set up edit control, store it in text file variable }
    TUserData(UserData).Edit := E;
  end;
end;
function RWEditOpen(var F: TTextRec): Integer;
begin
  Result := 0;
  with F do begin
    if Mode = fmInput then begin
      InOutFunc := @RWEditInput;
      FlushFunc := nil;
    end else begin
      Mode := fmOutput;
      InOutFunc := @RWEditOutput;
      FlushFunc := @RWEditOutput;
    end;
    CloseFunc := @RWEditClose;
  end;
end;
function RWEditInput(var F: TTextRec): Integer;
begin
  Result := 0;
  with F, TUserData(UserData).Edit do begin
    BufPos := 0;
    BufEnd := GetTextBuf(PChar(BufPtr), BufSize);
    { Pop a carriage return line feed combo in }
    StrCat(PChar(BufPtr), #13#10);
    Inc(BufEnd, 2);
    Text := '';  { Clear the edit }
  end;
end;
function RWEditOutput(var F: TTextRec): Integer;
var
  { Temporary PChar holder }
  Buf: packed array[0..255] of Char;
begin
  Result := 0;
  { Gets called when a Delphi 2 app shuts, in closing
    Output. Since it refers to the edit which won't
    exist, don't run it }
  if not Application.Terminated then
    with F, TUserData(UserData).Edit do
      if BufPos <> 0 then begin
        { Get PChar with BufPos characters in }
        StrLCopy(Buf, PChar(BufPtr), BufPos);
        { Put that in the edit }
        SetTextBuf(Buf);
        { Reset BufPos }
        BufPos := 0;
      end;
end;
function RWEditClose(var F: TTextRec): Integer;
begin
  Result := 0;
end;
end.
```

➤ *Listing 2*

doubles in size in Delphi 2. Note that a non-variant record would have done just as well. A commented one appears in Listing 2 below the variant one.

The device interface routines are declared `forward` at the top of the unit's `implementation` section. The customised `AssignFile` routine does its housekeeping as per the outlined requirements above. The `OpenFunc` routine does all the function pointer assignments based on the `Mode` value, as also described above. You'll notice that the `FlushFunc` pointer is set to `nil` for input, but to the `InOutFunc` value for output. The `CloseFunc` routine, on the other hand, has a very easy ride of it: it simply returns zero.

The input and output routines follow the previously stated guidelines to read from or write to the edit control. Note that `RWEditOutput` checks that the program isn't terminating.

Remember that `CloseFile` will call the `InOutFunc` routine first. This is important since the shutdown code contained in a Delphi 2 application calls `CloseFile` for `Output`. If we continue trying to access the edit control, we are risking an access violation.

The form unit in the TFDD1.DPR project, TFDDU.PAS, sets up the TFDD in the form's `OnCreate` handler. We can't do this in the TFDD unit initialisation, as it relies on an object in the form that only gets created after the initialisation sections have finished executing.

```
AssignRWEdit(Input, Edit1);
Reset(Input);
AssignRWEdit(Output, Edit1);
Rewrite(Output);
```

This allows other event handlers to use `Read/Ln` and `Write/Ln` without specifying a file variable: `Input` and `Output` will be used implicitly.

There are two buttons and an edit on the form. The first button reads two floating point values from the edit and then writes some text back in.

The second button reads a string from the edit control. The code in Listing 3 (on the next page) from the two buttons' `OnClick` handlers show how these I/O procedures can liaise with the edit. The program is shown in Figure 1.

An Output Only Debugging Device

This time, for a second example, we will look at a write-only TFDD. This one, demonstrated by TFDD2.DPR, will act as a debugging tracing type tool: the strings you give to `WriteLn` are displayed in a separate window. When you come to deploy an application using this TFDD, you can modify it using a compiler directive so that no output occurs.

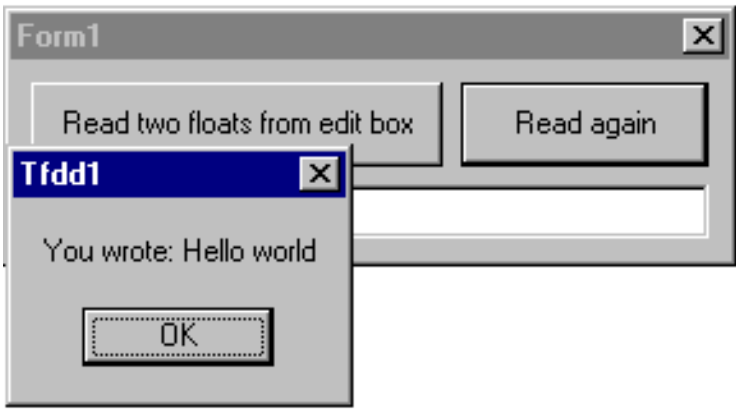
Since this TFDD relies on no external edit controls or other objects it can set itself up (ie call its `AssignFile` substitute on the `Output` file variable) in its unit initialisation

section. You can see this at the end of Listing 4 (file DEBUGU.PAS).

The `AssignDebug` procedure does much the same as the earlier `AssignRWEdit` did, leaving most of the TFDD function pointers to be set up by the `OpenFunc` routine `DebugOpen`). This in turn ensures the device is only opened for output by causing an I/O error 5 (access denied) if opened with `Reset`. Otherwise it sets up the pointers and creates the debug output form (under control of conditional compilation). `DebugClose` will similarly free the form.

The output routine ensures the form is visible. This means that the first `WriteLn` will cause the debugging form to appear on the desktop, as it starts off hidden. It then sets the selected text in the form's memo to be the supplied text. Normally, there will be no selected text, and the caret will be at the end of the memo's text, meaning the new text will be appended to the memo. Notice again that conditional compilation is used to prevent references to the form. In

➤ *Figure 1*




```
procedure TForm1.Button1Click(Sender: TObject);
var D1, D2: Double;
begin
{$ifdef OnePossibility}
  ReadLn(D1, D2);
{$else}
  Read(D1);
  Read(D2);
  ReadLn;
{$endif}
  ShowMessage(Format('First value: %f', [D1]));
  ShowMessage(Format('Second value: %f', [D2]));
  Write('Type something in me and push the 2nd button');
end;

procedure TForm1.Button2Click(Sender: TObject);
var S: String;
begin
  ReadLn(S);
  ShowMessage(Format('You wrote: %s', [S]));
  Write('Game over');
end;
```

➤ *Listing 3*

➤ *Listing 4*

```
unit Debugu;
{$define Debugging}
interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
{$ifdef Debugging}
type
  TDebugFrm = class(TForm)
    DebugMemo: TMemo;
    procedure FormClose(Sender: TObject;
      var Action: TCloseAction);
  end;
var DebugFrm: TDebugFrm;
{$endif}
procedure AssignDebug(var F: TextFile);

implementation
{$ifdef Debugging}
{$R *.DFM}
{$endif}
function DebugOpen(var F: TTextRec): Integer; far; forward;
function DebugOutput(var F: TTextRec): Integer; far; forward;
function DebugClose(var F: TTextRec): Integer; far; forward;

procedure AssignDebug(var F: TextFile);
begin
  { Set up text file variable }
  with TTextRec(F) do begin
    Handle := $FFFF;
    OpenFunc := @DebugOpen;
    Mode := fmClosed;
    BufSize := SizeOf(Buffer);
    BufPtr := @Buffer;
    Name[0] := #0;
  end;
end;

function DebugOpen(var F: TTextRec): Integer;
begin
  Result := 0;
  with F do begin
    if Mode = fmInput then
      Result := 5   { Access denied }
    else begin
      Mode := fmOutput;
      InOutFunc := @DebugOutput;
      FlushFunc := @DebugOutput;
    end;
    CloseFunc := @DebugClose;
  end;
{$ifdef Debugging}
  DebugFrm := TDebugFrm.Create(Application);
{$endif}
end;

function DebugOutput(var F: TTextRec): Integer;
var Buf: array[0..255] of Char;
begin
  Result := 0;
{$ifdef Debugging}
  { This gets called when a Delphi 2 app shuts, in closing
    Output. Since it refers to the form which won't exist,
    don't run it }
  if not Application.Terminated then begin
    { If output form ain't showing, show it }
    if not DebugFrm.Visible then
      DebugFrm.Show;
    with F do begin
      StrLCopy(Buf, PChar(BufPtr), BufPos);
      DebugFrm.DebugMemo.SelText := StrPas(Buf);
      BufPos := 0;
    end;
  end;
{$endif}
end;

function DebugClose(var F: TTextRec): Integer;
begin
  Result := 0;
{$ifdef Debugging}
  DebugFrm.Free;
{$endif}
end;

{$ifdef Debugging}
procedure TDebugFrm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  Action := caNone;
end;
{$endif}

initialization
  AssignDebug(Output);
  Rewrite(Output);
end.
```

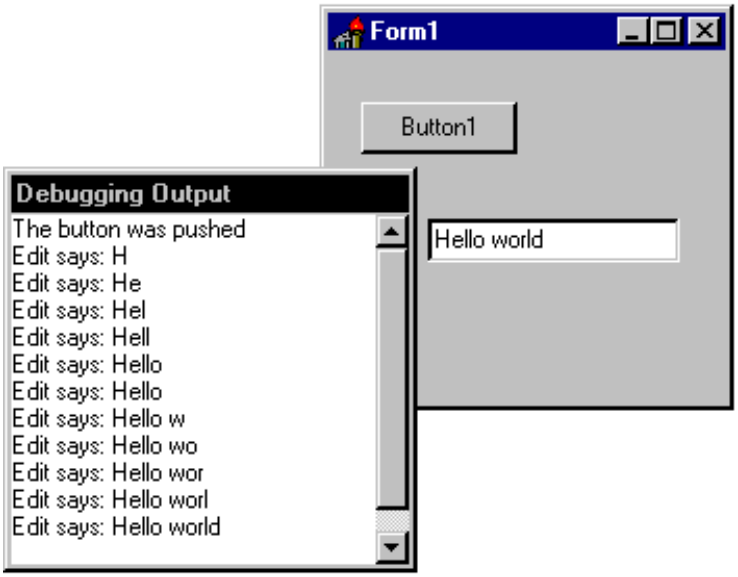
fact the entire debugging form class is conditionally compiled, as is the compiler directive that links the form into the executable. There is only one event handler in the form, for `OnClose`, which prevents it from being closed apart from by normal application termination.

The form that uses this TFDD has a button, whose `OnClick` handler writes a message to the debug window, and an edit control whose `OnChange` handler keeps updating the debug window with the current text in the edit. One of these is shown below and the program, showing the debug window is in Figure 2.

```
procedure TForm1.Edit1Change(
  Sender: TObject);
begin
  WriteLn('Edit says: ' +
    Edit1.Text);
end;
```

In order to get a deployable application, with no TFDD form resource included, simply remove the compiler directive from the top of the unit. It currently defines the symbol `Debugging`. Deleting the line will stop it being defined.

➤ *Figure 2*



TFDD References

For more information, take a look at the *Borland Pascal With Objects 7.0 Language Guide*, Chapter 14, on *Input and Output*. Also, from the Delphi 1 source files:

`\SOURCE\RTL\WIN\WINCRT.PAS`

and from the Delphi 1 or 2 source:

`\SOURCE\VCL\PRINTERS.PAS`

Brian Long is a freelance Delphi consultant and trainer based in the UK. He is available for bookings and can be contacted by email on 76004.3437@compuserve.com