The Delphi CLINIC

Edited by Brian Long

Problems with your Delphi project?
Just email Brian Long, our Delphi Clinic Editor, on 76004.3437@compuserve.com or write/fax us at The Delphi Magazine

File Handle Leak

Q I'm using the code below to make a new file when I need it. The problem is that after the program has been running a while, I get an `EInOutError` exception saying there are too many open files. I seem to be losing file handles somewhere.

```
var F :TextFile;
try
  if not FileExists(FileName)
    then FileCreate(FileName);
  AssignFile(F, FileName);
  Rewrite(F);
  CloseFile(F);
except
  { Handle problem }
end;
```

A The core problem is your call to `FileCreate`. This does create a new file, but also opens it and returns the file handle. Since you not storing this handle, the file remains open and each call to this code uses another handle. However, the call to `FileCreate` is not required anyway. Your call to `Rewrite` will accommodate `FileCreate`'s purpose.

You possibly have a misunderstanding of how file handles work. If a file handle is returned for a file, it is not the only file handle for that file. Another file routine can open the same file and that will yield a second file handle. Each one will take up space in your program's file handle table, and each one will individually need to be closed.

Validation En Masse

Q I have approximately seventy `TDBEdit` controls all of which are only allowed values of 0, 1 or 2. I wish to check each value when it has been entered and so I tried using the edit's `OnExit` event. I have been unsuccessful since from within the `OnExit` handler I can't identify the control that I have just left.

A There are two possibilities that come to mind. Your `OnExit` handler can identify the control of interest by looking at its `Sender` parameter. This `TObject` parameter is in fact the control that just lost the focus. The `OnExit` routine, shared amongst all the `TDBEdit` controls, could be like Listing 1.

An alternative (and maybe better) idea would be to get the field objects to validate themselves rather than the data-aware user interface elements. You can make an `OnValidate` event handler for each of the field objects (use the Fields Editor for the table or query to make these); again, make a shared one if it is appropriate. In the `OnValidate` event, check the contents of the field (it is passed as a parameter to the event) and if it is bad, raise an exception. See Listing 2. *[See Bob Swart's article on page 29 for more on validation. Editor]*

Enum As String

Q Given a value as an enumerated type, is there any way of turning it into a string?

A Normally in a compiled language the answer is no. But we do have run-time type information (RTTI) in Delphi and there are a number of subroutines in the `TypInfo` unit which know how to access it, the important one for this question being `GetEnumName`. This unit is not documented, but can be very useful: its interface can be found in TYPINFO.INT in the \DELPHI\DOC directory. We can cause RTTI to be generated for any given type by passing the type to the special `TypeInfo` function. Given the enumerated type `TFormBorderStyle`, used for a form's `BorderStyle` property and defined:

➤ *Listing 1*

```
procedure TForm1.DBEdit1Exit(Sender: TObject);
var Value: String;
begin
  if Sender is TDBEdit then
    if TDBEdit(Sender).Field <> nil then begin
      Value := TDBEdit(Sender).Field.AsString;
      if (Length(Value) <> 1) or not (Value[1] in ['0'.. '2']) then begin
        ActiveControl := TWinControl(Sender);
        raise EDatabaseError.Create('Value must be 0, 1 or 2');
      end;
    end;
end;
```

➤ *Listing 2*

```
procedure TForm1.Table1SpeciesNoValidate(Sender: TField);
var Value: String;
begin
  Value := Sender.AsString;
  if (Length(Value) <> 1) or not (Value[1] in ['0'..'2']) then
    raise EDatabaseError.Create('Value must be 0, 1 or 2');
end;
```

```
TFormBorderStyle =
  (bsNone, bsSingle,
   bsSizeable, bsDialog);
```

we can say:

```
uses TypInfo;
Caption := GetEnumName(
  TypeInfo(TFormBorderStyle),
  Ord(BorderStyle))^;
BorderStyle :=
  TFormBorderStyle(
  GetEnumValue(TypeInfo(
  TFormBorderStyle),
  'bsSingle'));
```

In Delphi 2 you do not use the ^ symbol in the first line. This idea was first presented by Michael Ax in the *Tips & Tricks* section of Issue 6, and again in Issue 9 by Stephen Posey.

CPU Level Debugging

Q Is there any way of seeing the assembler instructions generated by the Delphi 2 compiler without resorting to the cumbersome Turbo Debugger?

A There seems to be a consensus of opinion that Turbo Debugger is a tricky beast to tame: unfounded in my opinion. Turbo Debugger does the same stuff as Delphi's debugger, just with sometimes differing keystrokes to Delphi's default. But in any case, it has come to my attention that there is an undocumented registry entry that controls an assembler view of your Delphi program. It is quite rudimentary but shows the instructions and allows you to step through the program at a machine instruction level. Using the Windows 95 Registry Editor, navigate through `HKEY_CURRENT_USER`, `Software`, `Borland`, `Delphi`, `2.0`, `Debugging`. Ensuring `Debugging` is the selected section, choose `Edit | New | String value`. Type in the name `EnableCPU` and press `Enter` to confirm it. Now choose `Edit | Modify` and type in a value of 1. Next time you launch Delphi 2, choose the new item `View | CPU`. It will appear more useful if you run a program and stop it at a breakpoint.

If memory serves correctly, the NT 3.5 Registry Editor is a little less friendly than that in Windows 95. If you are running on NT, put the snippet from Listing 3 in a program and add Registry to an appropriate `uses` clause.

Stop Splashing About

Q Since Delphi 2 is 32-bit, I am able to continue working in other applications whilst it is loading. However, the splash screen that shows during the load-up process gets in the way. Is there some way to turn it off?

A Indeed. Modify your Delphi 2 shortcut to pass a `/NS` command-line switch and your wish is granted.

Form-wide OnMouseMove

Q When you make an `OnMouseMove` event handler for a form, it only fires when the mouse moves over the form, not when over any controls on the form. Can I make it fire for the entirety of the form, including all controls and still get form-relative X/Y co-ordinates passed in?

A Let's say your form's `OnMouseMove` handler looks like `GenericMouseMove` in Listing 4, which writes the mouse coordinates on the caption bar as well as which quarter the mouse is in. To make it work across the form, you can go round all your controls and share the event handler by selecting the `OnMouseMove` event, dropping down the list of available handlers and choosing `GenericMouseMove`. This event handler sharing could also be achieved using a text version of the form. Delphi 2 users can get this by right-clicking on the form and choosing `View as text`, but Delphi 1 users will need to choose `File | Open file`, change the `List files of type` to show DFM files, and open the appropriate one. Locate the `OnMouseMove` event property for the form, and then copy it into all the other objects on the form. A third alternative which is probably easier is to use the code from Listing 5 in your form's `OnCreate` handle to automate the process.

➤ *Listing 3*

```
with TRegIniFile.Create('Software\Borland\Delphi\2.0') do
  try
    WriteString('Debugging', 'EnableCPU', '1')
  finally
    Free
  end
```

➤ *Listing 4*

```
procedure TForm1.GenericMouseMove(
  Sender: TObject; Shift: TShiftState; X, Y: Integer);
type
  TBoolStrings = array[Boolean] of String;
const
  Vert: TBoolStrings = ('Bottom', 'Top');
  Horz: TBoolStrings = ('right', 'left');
begin
  Caption := Format('%s %s (%d,%d)', [Vert[Y < ClientHeight div 2],
    Horz[X < ClientWidth div 2], X, Y]);
end;
```

➤ *Listing 5*

```
procedure TForm1.FormCreate(Sender: TObject);
var Loop: Integer;
begin
  for Loop := 0 to ComponentCount - 1 do
    if Components[Loop] is TControl then
      TButton(Components[Loop]).OnMouseMove := GenericMouseMove
end;
```

One of these approaches has now shared the event handler between all the controls but the coordinates are control-relative. The `OnMouseMove` event handler needs some modification to give form-relative controls all the time, as shown in Listing 6. The project MOVE.DPR on the disk shows this.

Default OLE Automation Property

Q In products other than Delphi, such as VB, you can access an OLE automation server's default property by not specifying a property at all. How do you set up such a property in a Delphi 2 server, and can you then reference it from a Delphi 2 written controller? The same question applies to the `Evaluate` method. A statement that would normally be

```
X.Evaluate("A1:C1").Value = 10
```

can be abbreviated to

```
X.[A1:C1].Value = 10
```

if an `Evaluate` method has been set up. How do we set this up in Delphi, and does Delphi support the same abbreviation?

A The default OLE server property is one with a dispatch id of `DispId_Value (0)`. You can specify your automated property dispatch ids using the little used `dispid` directive as shown in the example automation object in Listing 7.

Delphi controller code could access this property either in the normal way by specifying it's name, or without as shown in Listing 8. Notice that it seems sensible to use a typecast to change the returned `TDateTime` into a string, but since `Server` is a variant it is not strictly necessary. The first two lines use the typecast, but the second pair of lines are quite successful even without it.

Delphi doesn't support the shorthand form of the `Evaluate` routine, but you should be able to set up such a routine using a `dispid` of `DispId_Evaluate (-5)`.

Inheritance Addendum

Last month's discussion of `virtual`, `dynamic` and `override` made use of the `inherited` keyword. Normally, in any method, you can use `inherited` to specify you are referring to a method or property from the ancestor class (extensively used in conjunction with polymorphism) as below:

```
procedure TDerived.AMethod(
  Value1, Value2: Integer);
begin
  inherited AMethod(
    Value1, Value2)
end;
```

When writing Window message handlers, it is quite well documented that you can get the inherited functionality to be called by using the `inherited` word on its own (to avoid problems due to the ancestor method possibly not existing, or having a different name or parameter type). Delphi 2 introduced form inheritance. In an inherited form's event handlers, the original event handler from the ancestor form is called by also using just the word `inherited` (try it and check). This is possible in Delphi 2 due to a combination of two things. Firstly, Borland have pleasantly, but quietly, added general support for calling an ancestor procedure with `inherited` alone (functions and property references still need full syntax). Secondly, event handlers are all procedures.

The code below is slightly shorter than the previous snippet, but in Delphi 2 it will be equally valid:

```
procedure TDerived.AMethod(
  Value1, Value2: Integer);
begin
  inherited
end;
```


Acknowledgements

Thanks to Roy Nelson for various contributions to this month's answers.

➤ *Listing 6*

```
procedure TForm1.GenericMouseMove(
  Sender: TObject; Shift: TShiftState; X, Y: Integer);
type
  TBoolStrings = array[Boolean] of String;
const
  Vert: TBoolStrings = ('Bottom', 'Top');
  Horz: TBoolStrings = ('right', 'left');
var
  Pt: TPoint;
begin
  Pt := Point(X, Y);
  Pt := Form1.ScreenToClient((Sender as TControl).ClientToScreen(Pt));
  with Pt do
    Caption := Format('%s %s (%d,%d)', [Vert[Y < ClientHeight div 2],
      Horz[X < ClientWidth div 2], X, Y]);
end;
```

➤ *Listing 7*

```
TServer = class(TAutoObject)
private
  function GetCurrentTime: TDateTime;
automated
  property CurrentTime: TDateTime read GetCurrentTime dispid DispId_Value;
end;
```

➤ *Listing 8*

```
var Server: Variant;
...
Server := CreateOLEObject('MyOLE.Server');
...
Caption := TimeToStr(Server.CurrentTime);
Caption := TimeToStr(Server);
Caption := Server.CurrentTime;
Caption := Server;
```