The Delphi CLINIC

Edited by Brian Long

Problems with your Delphi project? Just email Brian Long, our Delphi Clinic Editor, on clinic@blong.com

Delphi Component Templates

Q If I create some Delphi component templates, which files do I need to copy on to another PC to to use them there?

A All you need to copy from machine to machine is Delphi32.DCT from the BIN directory of Delphi 3 or 4. Incidentally, there are other files in the BIN directory which also have potential value in being copied. Delphi32.DMT (or Delphi.DMT in Delphi 1) is the Menu Designer's store of menu templates (accessed through its right-click popup menu). Also Delphi32.DCI is a text file containing all the Code Insight Code Templates from Delphi 3 or 4 (accessed via Ctrl-J in the editor, and set up from `Tools | Environment Options, Code Insight, Code Templates`).

Current Printer?

Q How do I find out which is the currently selected printer?

A Check out the `Printers` unit. As well as the `AssignPrn` routine, that allows simple text printing by treating the printer like a text file, this unit offers a `TPrinter` object through a function called `Printer`. In Delphi 1 `Printer` was a variable declared in the `interface` section of the `Printer` unit and set up in the `initialisation` section, but to aid smart linking it turned into a so-called 'factory' function in version 2. The first time it is called, it creates a `TPrinter` object and stores it in a variable in the `implementation` section. Subsequent calls simply return the value of that variable.

To answer your question, the `Printer` object has a property called `Printers`, which is a `TStrings` object containing all the currently installed printers. Another property, `PrinterIndex`, indicates which one is the current printer. The following statement would display the current printer on the screen:

```
uses
  Printers;
...
with Printer do
  ShowMessage(
    Printers[PrinterIndex])
```

Export To Excel

Q One of our clients has requested the functionality to export to Excel files (not using DDE or OLE), similar to the way Paradox for Windows exports to Excel. We have searched the net for assistance in this regard, to no avail. Is it possible?

A You should be able to get hold of an ODBC driver to write information out to Excel files. A `TBatchMove` component could help copy from, say, a Paradox table to an Excel file. Alternatively, if you were feeling very adventurous, you could look up the details of the Excel file format and write code that generates a file in the appropriate format. Excel stores its worksheets in BIFF files (Binary Interchange File Format) the layout of which can be found documented on the Microsoft Developer Network CDs.

Another option would be to use Automation. You could write a routine that invoked Excel through Automation, and then looped through your fields, writing information to the cells in an Excel worksheet, and finally saving the worksheet to disk. Maybe we'll have a look at this in a future *Clinic*.

Flashing Text

Q Is there an easy way of getting text to flash? Do you have to do it manually or is there an easier way? By manually I mean by changing the text colour at certain intervals.

A In text mode, the video card supports flashing colours, but not in graphics mode. So you are forced to do this manually (maybe using a `TTimer` component). Alternatively you could use some component that supports flashing text. If you do, then you are taking advantage of some pre-written code that does it manually also.

Microsoft Internet Mail Style Application

Q I have a requirement to write a Delphi application as follows. The first form that comes up (maybe the main form?) will be some form of logon screen. Having successfully navigated past this screen, all the subsequent forms will be fairly independent. As such, I want each of these subsequent forms to have its own entry on the task bar, and when minimised I want them to disappear into the task bar, instead of sitting just above the task bar as they do now. When one of them is clicked on, I don't want all the others to be brought to the foreground as is usually the case in a Delphi app.

Also, since the main logon form is only required at the beginning of the program, and then hidden from that point on, I want the main form to have *no* task bar entry. Finally, when the last one of my secondary, non-main forms is closed, I want my application to shut.

All this will mimic what I see of Microsoft Internet Mail's appearance (apart from my logon form). Mail can have lots of windows up (mail messages being written, plus the main mail window), and each of these is visible on the task bar. Bringing any of these windows to the foreground has no effect on any other Mail window. You can close the Mail windows in any order you like, and the last one shut presumably unloads Microsoft Mail. Can all this be done in Delphi?

A Ask and ye shall receive ☺. Sometimes.

Of course Delphi can handle this, so long as you know how to coax it into co-operating. In fact your main logon form requirement helps make this job reasonably easy. You see, a potential issue is that when the main form is up on the screen, the icon used on the task bar (or on the desktop in Delphi 1 or in Windows NT 3.5) is not actually the form's icon. It is in fact the `Application` object's window icon. There is a certain amount of under-the-hood trickery going on that relies upon this fact, and the icon of the main form itself is not visible. So if we can make the `Application` object's icon disappear, then the main form will appear to have no icon.

In fact, at this stage, a 32-bit application will have no icon at all, because all subsidiary forms have no icon. They don't actually need an icon, because when they are minimised they sit themselves at the bottom of the screen above the task bar, in just the same way as minimised MDI child windows sit themselves at the bottom of the MDI parent. Issue 19 of *The Delphi Magazine* discussed this aspect of 32-bit Delphi application in the -*Window startup mode* entry of *The Delphi Clinic*.

To get rid of the task bar entry for a 32-bit application, you can add `ws_Ex_ToolWindow` into the extended window style of the underlying window. This requires a bit of API work to get the old extended window style, add the new value in, and then set the new, updated extended window style. Listing 1 shows some code that can be placed in a main form's `OnCreate` event handler to do the job in Delphi 3. I specify a version here because in fact this does not have the desired effect in Delphi 2. Delphi 1 and 2 prefer using the `ShowWindow` API call to hide the `Application` object window (which doesn't quite cut it in Delphi 3). Really, some conditional compilation is required here.

A test project on the disk, called UITest.Dpr, has this code in and a button is used to create ten instances of a secondary form of type `TOtherForm`, using a looped call to:

```
TOtherForm.Create(
  Application).Show,
```

after which the main form hides itself.

The next thing to get sorted is a sure-fire way of terminating the application. Currently, if you close all the secondary forms, then the application will still be sitting in memory, but with no icon on the task bar to close the hidden main form. So the `OnClose` handler shown in Listing 2 can be used in each of the non-main forms. Then, as the last non-main form is being closed, the whole program is terminated.

With the application now able to be successfully operated, we can try the last piece of the puzzle. We need each non-main form to have its own task bar icon, and also to act independently of any other form in the application. As it turns out, we can kill both of these metaphorical birds with one similarly metaphorical stone. By default, all Delphi forms are manufactured with Windows calls to be children of the `Application` object's window. If we override the virtual `CreateParams` method, we can choose a new window parent. Specifying the Windows desktop as the new parent happens to fulfil both requirements. See Listing 3.

The sample project implements all this code to be an example of an application that acts as required in Delphi 3 and 4, as well as taking account of the previously mentioned conditional compilation requirement for Delphi 2.

```
SetWindowLong(
  Application.Handle, gwl_ExStyle,
  GetWindowLong(Application.Handle, gwl_ExStyle) or ws_Ex_ToolWindow)
```

➤ *Listing 1*

```
procedure TOtherForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  { If the only forms left are this one being closed and the main form... }
  if Screen.FormCount = 2 then
    { ...then close the main form }
    Application.MainForm.Close;
  { Ensure this form is destroyed ASAP }
  Action := caFree;
end;
```

➤ *Listing 2*

```
TOtherForm = class(TForm)
...
private
  procedure CreateParams(var Params: TCreateParams); override;
end;
...
procedure TOtherForm.CreateParams(var Params: TCreateParams);
begin
  inherited CreateParams(Params);
  Params.WndParent := HWnd_Desktop
end;
```

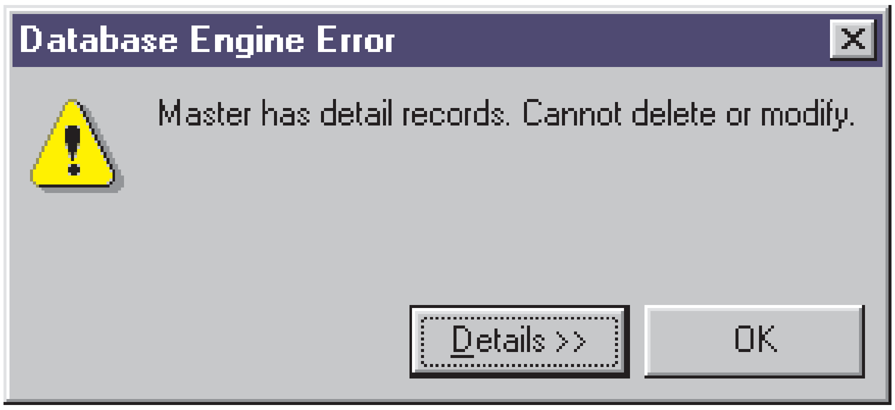
➤ *Listing 3*

BDE Error Dialog

Q I read the *BDE Errors* item in Issue 34's *Delphi Clinic*. I know that I can make a replacement default exception handler, but I have noticed that the 32-bit Delphi IDEs have a nice little dialog to report BDE errors. Can I make use of this dialog in my applications?

A Yes you can. The dialog is a form implemented in the VCL's `DBExcept` unit. The DCU and DFM files were supplied in Delphi 2, but with no source code, which made it rather tricky to work out the details. When Delphi 3 came along, the source file was supplied as well and so the job becomes much easier. The only problem is that the class seems to have changed somewhat between versions 2 and 3, and so the source file doesn't prove too helpful to Delphi 2 users. The information in this entry therefore pertains to Delphi 3 and upward only.

The idea is to create an instance of the form class, `TDbEngineErrorDlg`, and store it in its associated variable, `DbEngineErrorDlg`. Then you have a choice of how you deal with things. If you want to use the dialog in normal exception handlers, then you simply pass the relevant `EDBEngineError` exception as a parameter to `DbEngineErrorDlg.ShowException`. Alternatively, you can ask the form to automatically deal with any unhandled `EDBEngineError`, by installing its own default exception handler (`Application.OnException` event handler) through a call to its `HookExceptions` method. If you set up your own `Application.OnException` event handler, make sure you do it before calling `HookExceptions`.

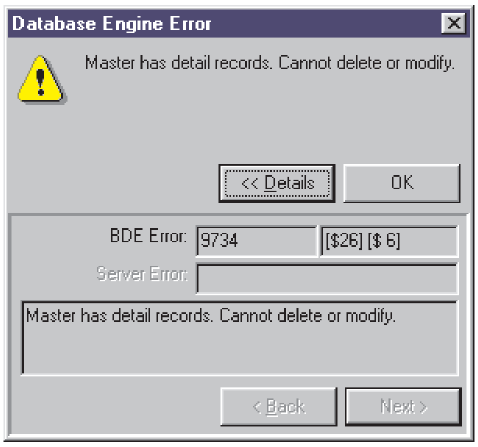

Database Engine Error

Master has detail records. Cannot delete or modify.

➤ *Figure 1*

When an exception is picked up by this form in one way or another, you get the form shown in Figure 1. Pressing the `Details` button gives the more detailed form in Figure 2. As you can see, this gives the information on the BDE error code that caused the exception, as well as the associated error message.

The sample project DBError.Dpr is a 32-bit app that uses this unit to report `EDBEngineError` exceptions. The project sets up its own system-wide exception handler through `Application.OnException` in the main form's `OnCreate` handler. The customised exception handler reacts differently to Access Violation exceptions as you can see in Listing 4. Immediately after installing this new global exception handler, the code instantiates the error form and tells it to hook into the default exception handling chain. Now, any `EDBEngineError` exceptions are handled by the error form, Access Violations are handled with a simple message from my `DoException` routine, and all other exceptions are dealt with in their natural way, thanks to the call to `Application.ShowException`.


Database Engine Error

Master has detail records. Cannot delete or modify.

BDE Error: 9734 [$26] [$ 6]

Server Error:

Master has detail records. Cannot delete or modify.

➤ *Figure 2*

To test out these differing exception handling options, the supplied project has a `DBGrid` on it displaying the customer table. Changing one of the existing `CustNo` field values to another, already existing, `CustNo` field value will give an `EDBEngineError`. There is also a button on the form dedicated to causing an Access Violation.

A second button on the form programmatically causes an `EDBEngineError`, by forcing a unique key violation. A `try..except..end` statement traps this error and explicitly invokes the error form through its `ShowException` method. The code for these two buttons is in Listing 5.

➤ *Listing 4*

```
procedure TForm1.DoException(Sender: TObject; E: Exception);
begin
  if E is EAccessViolation then
    //Simple reaction to an Access Violation
    ShowMessage('Ooch, Ouch, Ow!! That hurt!')
  else
    //Deal with all other exceptions in the normal way
    Application.ShowException(E);
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  //Set up personal default exception handler first
  Application.OnException := DoException;
  //Now create error dialog
  DbEngineErrorDlg := TDbEngineErrorDlg.Create(Application);
  //And install automatic support for it
  DbEngineErrorDlg.HookExceptions
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //Generate an Access Violation
  IntToStr(PInteger(nil)^)
end;
procedure TForm1.Button2Click(Sender: TObject);
var
  CustNo: Double;
begin
  try
    //Make sure we are not editing
    Table1.Cancel;
    //Record an existing unique key value
    CustNo := Table1['CustNo'];
    //Add new record
    Table1.Insert;
    //Use same unique value
    Table1['CustNo'] := CustNo;
    //Try and save, giving a key violation error
    Table1.Post
  except
    on E: EDBEngineError do begin
      //Ask error form to deal with exception
      DbEngineErrorDlg.ShowException(E);
      Table1.Cancel;
    end
  end
end;
```

➤ *Listing 5*

Capturing DOS Output

Q I am trying to capture output from a 32-bit non-GUI application (called `DIFF`, it compares two directories) which works at the command prompt. So if I type:

```
C:\Diff C:\DiffADir C:\
  DiffBDir > C:\Out.Txt
```

then the file Out.Txt is created, containing the output from the command line. However, when I use `CreateProcess` and pass the above line, the application does not process the output redirection. Using `CreateProcess`, I cannot even capture the output from `DIR`, which I can do at the command prompt using:

```
Dir C:\ > C:\DirList.Txt
```

In fact it gives me an error. What am I missing?

A The issue at hand is that the redirection of screen output to a file you get at the DOS prompt is in fact implemented by the command processor. When you launch your application from `CreateProcess`, you are leaving the command processor out of the equation.

The specific problem with the `DIR` call is that `DIR` is not a standalone command. `DIR` is a command which is internal to the command processor.

In both cases you will actually need to launch `COMMAND.COM` as the application, and ask it to run `DIFF` or `DIR` as a command. You can then add on the appropriate redirection or pipe symbols to the command line and your desires will be met.

Listing 6 has some code that will work in any 32-bit Delphi application. It extracts the fully qualified path to the command processor, and then appends the appropriate command line to make something like this:

```
C:\WINDOWS\COMMAND.COM /C
  DIR C:\ >C:\DirList.Txt
```

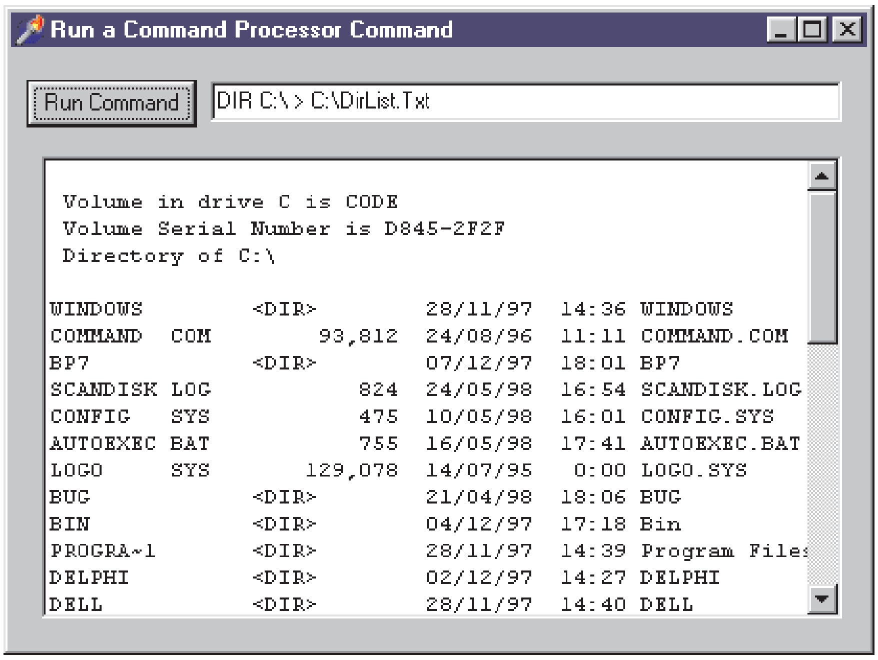
`CreateProcess` invokes the command. The code waits for the command to start and then waits again for it to finish. Finally, the redirected output in the text file is read into a listbox for the user to see. Figure 3 shows the program running.

And that's all till next time...

➤ *Listing 6*

```
procedure TForm1.Button1Click(Sender:TObject);
var
  SI: TStartupInfo;
  PI: TProcessInformation;
  ComSpec: array[0..MAX_PATH] of Char;
  CmdLine: String;
begin
  GetEnvironmentVariable('COMSPEC', ComSpec, SizeOf(ComSpec));
  CmdLine := String(ComSpec) + ' /C ' + Edit1.Text;
  GetStartupInfo(SI);
  Win32Check(CreateProcess(nil, PChar(CmdLine),
    nil, nil, False, 0, nil, nil, SI, PI));
  WaitForInputIdle(PI.hProcess, Infinite);
  WaitForSingleObject(PI.hProcess, Infinite);
  ListBox1.Items.LoadFromFile('C:\DirList.Txt')
end;
```

➤ *Figure 3*



```
 Volume in drive C is CODE
 Volume Serial Number is D845-2F2F
 Directory of C:\

WINDOWS        <DIR>        28/11/97  14:36 WINDOWS
COMMAND  COM        93,812  24/08/96  11:11 COMMAND.COM
BP7            <DIR>        07/12/97  18:01 BP7
SCANDISK LOG           824  24/05/98  16:54 SCANDISK.LOG
CONFIG   SYS           475  10/05/98  16:01 CONFIG.SYS
AUTOEXEC BAT           755  16/05/98  17:41 AUTOEXEC.BAT
LOGO     SYS       129,078  14/07/95   0:00 LOGO.SYS
BUG            <DIR>        21/04/98  18:06 BUG
BIN            <DIR>        04/12/97  17:18 Bin
PROGRA~1       <DIR>        28/11/97  14:39 Program File
DELPHI         <DIR>        02/12/97  14:27 DELPHI
DELL           <DIR>        28/11/97  14:40 DELL
```