The Delphi CLINIC

Edited by Brian Long

Problems with your Delphi project?

Just email Brian Long, our Delphi Clinic Editor, on clinic@blong.com

Incrementing A Date

Q I need to take a date and add a calendar month to it. The 30th March should give 30th April, but I also want 31st March to give 30th April, since April only has 30 days. This seems to be proving tricky with the varying lengths of months and the requirement to cater for leap years. Is there a routine that will do this for me?

A Fortunately the `SysUtils` unit has the `IncMonth` routine. So the following statement does the trick and increments the passed date by one month:

```
NewDate := IncMonth(OldDate, 1)
```

Access 97 Support

Q I'm using Delphi and wish to write applications that use Access databases. However, the information supplied with Delphi claims that the native Access support (ie BDE driver and not the ODBC driver) relies on MS Access 95 being installed on the machines that my application is to be deployed to. Some of the machines my application will be running on will have had MS Access 97 installed but not MS Access 95. This seems to mean that my application will not run successfully. When will Delphi support MS Access 97?

A The BDE MS Access driver relies on version 3.0 of the Microsoft Data Access Objects (DAO 3) as supplied in Access 95. The Data Access Objects as shipped with Access 97 (DAO 3.5) were different enough to mean that the BDE Access driver would not work with them initially. Delphi 3 shipped with BDE 4.00. The maintenance release, Delphi 3.01, shipped with an updated version of the BDE, version 4.01. Sometime shortly after that, a new version of Visual dBASE was released which used BDE 4.51. This version of the BDE supports DAO 3.0 and DAO 3.5 using different driver libraries; in other words it supports both Access 95 and Access 97. As soon as this new version of Visual dBASE was released, BDE 4.51 was made available on Inprise's website.

So, in summary, if you haven't already got your hands on Delphi 4, which ships with BDE 5, then download BDE 4.51 from Inprise's website (www.inprise.com).

Access Security

Q When I use MS Access it looks for user names and passwords as appropriate in the system security database (SYSTEM.MDW) that I previously attached to with WRKGADM.EXE. However, when I use Delphi 3 to talk to my Access databases, Delphi seems to completely ignore the information in this .MDW file. Can I get the BDE to make use of this file?

A From version 4.01 of the BDE, as first shipped in the Delphi 3.01 maintenance release, the MS Access driver has a `SYSTEM DATABASE` parameter that can be set in the BDE Administrator. This can be set to point to a workgroup information file (.MDW file) of your choice. You can set a default value for the driver, and can also modify it on a per-alias basis.

Memory Management Woes

Q I have been trying to implement a dynamic array in Delphi using a `TList`. It seems to be all wonderful, but I'm left with a question: for proper cleanup on terminating an application, how do you know whether a `TList` has been created but not yet freed? One answer is to create it, use it and free it all in one gulp, within the appropriate `try..finally` block. But I might need to create a `TList`, then use it in various parts of the application and free it along the way. When the program shuts, if the `TList` hasn't yet been freed, then I want to be sure to free it. I've tried in an `OnClose` handler:

```
if Assigned(MyList) then
  MyList.Free;
```

The trouble is, if `MyList` was freed earlier, `Assigned(MyList)` will still be `True`. Then I tried:

```
try
  MyList.Free;
except
  on E: Exception do
    ShowMessage('OnClose, error
      freeing MyList: ' + E.Message);
end;
```

My exception handler doesn't catch this, but after the application appears to terminate, I get an exception dialog saying *Access violation*. Any light you can shed?

A Your point about `Assigned(MyList)` being `True` regardless is incorrect if you remember the rule of 'whenever you free an object, set its object reference back to `nil`', eg:

```
MyList.Free;
MyList := nil;
```

If you do this, calling `Free` on the object reference is perfectly safe as it will observe that it is being called

through a nil reference and perform no action. Check the *Destructor Query* entry in *The Delphi Clinic* in Issue 29 for more on this issue.

Forcing Windows Shutdown

Q I'm using Delphi 3 on an NT 4 Workstation and I want to do a complete system shutdown from a Delphi app. Here's a relevant section from the Win32 API help:

'The ExitWindowsEx function can be used to shut down the system. Shutting down flushes file buffers to disk and brings the system to a condition in which it is safe to turn off the computer. The calling process must have the SE_SHUTDOWN_NAME privilege to shut down the system.

The following example enables the SE_SHUTDOWN_NAME privilege and then shuts down the system [shown in Listing 1].'

I have translated this into Delphi, and tried running it, but it does not work. ExitWindowsEx returns ERROR_NO_SUCH_PRIVILEGE despite me logging in as the Administrator. Can you see anything wrong in my translation?

A I checked through your translation and it was fine. However, the original example from the Windows API help was wrong. The reference to TEXT("SE_SHUTDOWN_NAME") should have just been SE_SHUTDOWN_NAME. In the Win32 SDK, WinNT.H defines SE_SHUTDOWN_NAME to be TEXT("SeShutdownPrivilege"). In Delphi, you just use the string in the quotes and forget about what the TEXT() macro does. Some code that works fine in Delphi 2 and 3 is shown in Listing 5, and it caters for both Windows NT and Windows 95.

You might notice I commented out the EWX_FORCE flag, as normally you would want the user to terminate their applications individually, saving data where they wish to. Forcing a shutdown over the users' heads is to be discouraged. The code is in ShutDown.Dpr.

Since Win32Check was introduced in Delphi 3, the code in Listing 5 works because a substitute routine is made available if being compiled in Delphi 2. Win32Check is a procedure that takes a Boolean parameter, and if it is False, raises an exception with a message generated by SysErrorMessage having GetLastError passed to it. My code looks like Listing 2.

In fact there was another problem in making the code work with Delphi 2. The declaration of the OpenProcessToken API changed from Delphi 2 to 3. Delphi 2's declaration looks like Listing 3.

```
HANDLE hToken;
TOKEN_PRIVILEGES tkp;
/* Get a token for this process. */
if (!OpenProcessToken(GetCurrentProcess(),
  TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, &hToken)) error("OpenProcessToken");
/* Get the LUID for the shutdown privilege. */
LookupPrivilegeValue(NULL, TEXT("SE_SHUTDOWN_NAME"), &tkp.Privileges[0].Luid);
tkp.PrivilegeCount = 1;  /* one privilege to set    */
tkp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
/* Get the shutdown privilege for this process. */
AdjustTokenPrivileges(hToken, FALSE, &tkp, 0, (PTOKEN_PRIVILEGES)NULL, 0);
/* Cannot test the return value of AdjustTokenPrivileges. */
if (GetLastError() != ERROR_SUCCESS) error("AdjustTokenPrivileges");
/* Shut down the system and force all applications to close. */
if (!ExitWindowsEx(EWX_SHUTDOWN | EWX_FORCE, 0)) error("ExitWindowsEx");
```

➤ *Above: Listing 1*

➤ *Below: Listing 2*

```
procedure Win32Check(Value: Boolean);
begin
  if not Value then
    raise Exception.Create(SysErrorMessage(GetLastError))
end;
```

```
function OpenProcessToken(ProcessHandle: THandle; DesiredAccess: DWORD;
TokenHandle: PHandle): BOOL; stdcall;
```

➤ *Above: Listing 3*

➤ *Below: Listing 4*

```
Function OpenProcessToken(ProcessHandle: THandle; DesiredAccess: DWORD; var
TokenHandle: THandle): BOOL; stdcall;
```

The declaration in Delphi 3 changes to that shown in Listing 4. Notice the last parameter, TokenHandle, has changed from being a pointer to a THandle to being a var parameter of type THandle. So, in Delphi 2 the last parameter should be @HToken instead of HToken. This is taken care of by conditional compilation in the files on disk.

Delphi Colour Changes

Q I know I can change the colour that the editor displays my Pascal code in, but are there any more colour preferences I can customise in Delphi?

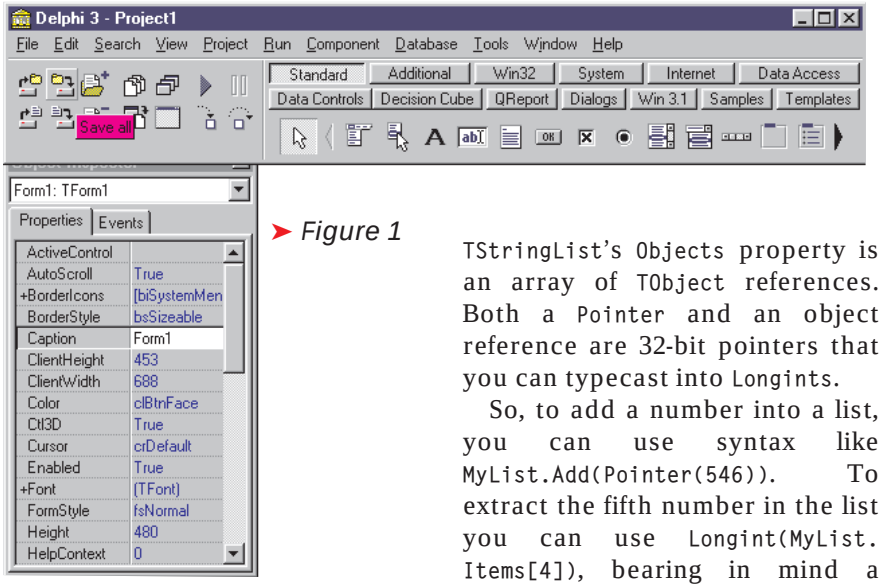
A Of course many of the colours used by the product are relative colours, which will change if you alter your Windows colours in Control Panel. However, Delphi does have a couple more options tucked up its sleeve. You can customise the colour the Object Inspector uses to display its property values. Additionally, the Delphi main window allows you to customise the colour used to draw the background of tooltips.

These two settings rely on a section of Delphi's registry area that appears to be undocumented since Delphi 2, though the equivalent area of the Delphi.Ini file was documented for Delphi 1. In

```
HKEY_CURRENT_USER\Software\
  Borland\Delphi\3.0
```

(or 2.0) you can define a Globals key. This key supports at least three string values. HintColor is a string representation of a Windows colour number, decimal or hexadecimal, that specifies a tooltip background colour and PropValueColor dictates the Object Inspector's property value colour. Finally, PrivateDir defines a potentially shared directory containing certain configuration files such as the Delphi menu template collection. Delphi 4 is a bit more choosy about what it reads, and seems to ignore the HintColor entry.

IDESetup.Dpr shows how to set up the two IDE colour options in the appropriate registry area, or in



➤ *Figure 1*

the `[Globals]` section of Delphi 1's Delphi.Ini (see Listing 6). Figure 1 shows a possible effect on the Delphi 3.0 IDE.

List Of Numbers

Q I need to store a list of numbers in memory for my program to work with. I can use an array, but that means I need to declare it as large as the largest list I will ever need. Are there other alternatives?

A Yes, you could use a dynamically sized array (watch out for my *Dynamic Arrays* article in the next issue). But, as an alternative, you can store 32-bit numbers directly in a `TList`, or in the `Objects` array of a `TStringList`. A `TList`'s `Items` property (its default array property) is an array of pointers. A `TStringList`'s `Objects` property is an array of `TObject` references. Both a `Pointer` and an object reference are 32-bit pointers that you can typecast into `Longints`.

So, to add a number into a list, you can use syntax like `MyList.Add(Pointer(546))`. To extract the fifth number in the list you can use `Longint(MyList.Items[4])`, bearing in mind a `TList`'s elements are numbered from zero. Since the `Items` property is the default array property, you can abbreviate this slightly to: `Longint(MyList[4])`. Of course any other type that fits into four bytes can also be stored directly in the pointers of a `TList`.

As an example, the project Lottery.Dpr will choose six different random numbers between 1 and 49, as required when playing the UK National Lottery. To ensure the six numbers are all different, when each number is chosen, it will be removed from the list. The numbers will be stored in a `TList`, created in the form's `OnCreate` event handler and destroyed in the `OnDestroy` handler (see Listing 7).

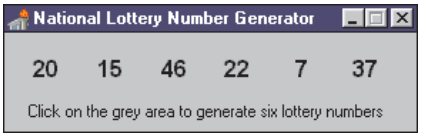
Each time a new set of six numbers is required, the list will be cleared, using the `Clear` method, and a loop will then add the 49 numbers in. A second loop iterates six times to pick a number. A random element in the list will be chosen (employing the list's `Count` property) and that number will be written into an appropriate label's caption. Finally, the number is removed from the list via the `Delete` method, before moving on to the next number. The code for this bit, an `OnClick` handler shared by everything on the form is in

➤ *Listing 5*

```
procedure TForm1.btnRestartClick(Sender: TObject);
var
  HToken: THandle;
  TP, OldTP: TTokenPrivileges;
  ReturnLen: Integer;
begin
  if Win32Platform = VER_PLATFORM_WIN32_NT then begin
    //Get a token for this process
    Win32Check(OpenProcessToken(GetCurrentProcess(),
      TOKEN_ADJUST_PRIVILEGES or TOKEN_QUERY, HToken));
    //Get the LUID for the shutdown privilege
    Win32Check(LookupPrivilegeValue(nil,
      'SeShutdownPrivilege', TP.Privileges[0].Luid));
    TP.PrivilegeCount := 1; //One privilege to set
    TP.Privileges[0].Attributes := SE_PRIVILEGE_ENABLED;
    //Acquire shutdown privilege for this process
    Win32Check(AdjustTokenPrivileges(HToken,
      False, TP, SizeOf(OldTP), OldTP, ReturnLen));
  end;
  //Shut down the system and force all applications to close
  if not ExitWindowsEx(EWX_SHUTDOWN {or EWX_FORCE}, 0) then
    raise Exception.Create('Cannot shut Windows');
  Close;
end;
```

➤ *Listing 6*

```
uses
{$ifdef Win32}
  Registry,
{$else}
  Inifiles,
{$endif}
  Windows, Messages, ...;
type
  TForm1 = class(TForm)
    ...
  private
  {$ifdef Win32}
    Ini: TRegIniFile;
  {$else}
    Ini: TIniFile;
  {$endif}
  end;
...
procedure TForm1.FormCreate(Sender: TObject);
begin
{$ifdef Win32}
  Ini := TRegIniFile.Create('Software\Borland\Delphi\3.0');
{$else}
  Ini := TIniFile.Create('Delphi.Ini');
{$endif}
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
  Ini.Free
end;
procedure TForm1.btnTooltipClick(Sender: TObject);
begin
  CD.Color := StrToInt(Ini.ReadString('Globals',
   'HintColor', IntToStr(Application.HintColor)));
  if CD.Execute then
    Ini.WriteString('Globals', 'HintColor',
      Format('$%x', [CD.Color]))
end;
procedure TForm1.btnPropClick(Sender: TObject);
begin
  CD.Color := StrToInt(Ini.ReadString('Globals',
    'PropValueColor', IntToStr(clWindowText)));
  if CD.Execute then
    Ini.WriteString('Globals', 'PropValueColor',
      Format('$%x', [CD.Color]))
end;
```



➤ *Figure 2*

Listing 8. The program is shown running in Figure 2.

The labels on the form are called `lblNo1`, `lblNo2`... up to `lblNo6`. The code in Listing 8 takes advantage of this to simplify the setting of their captions. The loop manufactures the component name as a string and passes it to the form's `FindComponent` method. If `FindComponent` locates a component owned by the form with the given name, it returns a reference to it. However, since `FindComponent` always returns a `TComponent` reference, the `TLabel` typecast is required in order to access the `Caption` property.

The second version of this project performs a couple of extra user-friendly actions (Listing 9 shows the main controlling code). Before each number is written onto the label, some small, embarrassingly minor, attempt at animation is made, a sort of spinning line thing. This is done by looping through the characters /, -, \ and | a few times. The routine guilty of playing this little game is shown in Listing 10. As you can see, since it makes the whole number calculation thing take longer, it makes sure `Application.ProcessMessages` and `Application.Terminated` are used appropriately to let the user give up if necessary. Also, because of the nature of `ProcessMessages`, Listing 9 uses a small semaphore flag to ensure that the routine isn't re-entered by the user clicking the form midway through its number generation. The final action of the `OnClick` handler in Listing 9 is to sort the numbers on the form into numerical order. A routine, `SortNumbers`, does this using some simplistic binary sort algorithm.

Acknowledgements

Thanks to Steve Axtell of Inprise's Professional Services Department for most of the database-related information used in this month's column.

```
type
  TfrmLottery = class(TForm)
    ...
    List: TList;
  end;
...
procedure TfrmLottery.FormCreate(Sender: TObject);
begin
  Randomize;  { Initialise random number generator }
  List := TList.Create  { Create number list }
end;
procedure TfrmLottery.FormDestroy(Sender: TObject);
begin
  List.Free;  { Dispose of number list }
  List := nil
end;
```

➤ *Above: Listing 7*

➤ *Below: Listing 8*

```
procedure TfrmLottery.GenericClick(Sender: TObject);
var   Loop, Index: Integer;
const MaxNum = 49;
begin
  List.Clear;  { Empty the list }
  { Fill the list with 49 numbers }
  for Loop := 1 to MaxNum do
    List.Add(Pointer(Loop));
  { Loop for each number sought }
  for Loop := 1 to 6 do begin
    { Choose one of the remaining numbers in the list }
    Index := Random(List.Count);
    { Write it in the appropriate label }
    (FindComponent('lblNo' + IntToStr(Loop)) as TLabel).Caption :=
      IntToStr(Longint(List[Index]));
    { Remove the number from the list, so it won't be picked again }
    List.Delete(Index);
  end
end;
```

```
procedure TfrmLottery.GenericClick(Sender: TObject);
var
  Loop, CharLoop, Index: Integer;
const
  MaxNum = 49;
  InHandler: Boolean = False;  { Re-entry protection flag }
begin
  if not InHandler then begin
    InHandler := True;
    List.Clear;    { Empty the list }
    { Fill the list with 49 numbers }
    for Loop := 1 to MaxNum do
      List.Add(Pointer(Loop));
    { Loop for each number sought }
    for Loop := 1 to 6 do begin
      { Choose one of the remaining numbers in the list }
      Index := Random(List.Count);
      { Do "pretty" animation }
      AnimateLabel(FindComponent('lblNo' + IntToStr(Loop)) as TLabel, 15, 50);
      { Write it in the appropriate label }
      TLabel((FindComponent('lblNo' + IntToStr(Loop)))).Caption :=
        IntToStr(Longint(List[Index]));
      { Remove the number from the list, so it won't be picked again }
      List.Delete(Index);
    end;
    ShowMessage('Press OK to sort the numbers into numerical order');
    SortNumbers([lblNo1, lblNo2, lblNo3, lblNo4, lblNo5, lblNo6]);
    InHandler := False
  end;
end;
```

➤ *Above: Listing 9*

➤ *Below: Listing 10*

```
{$ifndef Win32}
procedure Sleep(MSec: Integer);
var OldTime: TDateTime;
begin
  OldTime := Now;
  repeat until Now >= OldTime + MSec / MSecsPerDay
end;
{$endif}
procedure AnimateLabel(Lbl: TLabel; Loops, Delay: Integer);
var
  Loop: Integer;
const
  Chars: String = '/-\|';
begin
  for Loop := 1 to Loops do begin
    Lbl.Caption := Chars[Loop mod Length(Chars) + 1];
    Lbl.Parent.Invalidate;        { Post label's parent a repaint message }
    Application.ProcessMessages;  { Allow all posted messages to be processed }
    if Application.Terminated then Break;    { Let user bail out if bored }
    Sleep(Delay)
  end
end;
```