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

Paradox Table Corruption

Q I am developing a Paradox table application in Delphi 1 to run on Windows For Workgroups (WFWG) 3.11. During testing we have had various occurrences of records being lost, indexes out of date and indexes corrupted. Why might this be caused, and what can I do to avoid it? The network server PC is also used as a client PC running the .EXE and is known as network drive L to all other PCs. We've added a `SUBST L: C:\` to that PC's AUTOEXEC.BAT, so it's IDAPI.CFG can also refer to `L:\...` for its net directory and database path. We're running WFWG in enhanced mode and so VSHARE.386 is running, so do I really need SHARE? We've tried running the .EXE on Windows 95 and besides the above problem it seems to run OK. Are there any other things we should consider/change before running it live on Windows 95?

A Below are various recommendations you might find in various places and comments about them all, amassed from a number of Paradox and BDE experts. Thanks to Steve Axtell of Borland's European Technical Team, Phil Goulson of the UK Delphi Developer's Group, John O'Connell, Mike Orriss and Eryk Bottomley for their input.

A frequent cause for any of the above data corruption problems is the premature termination (power loss, or PC reset possibly forced upon the user by a program hang) of a program accessing a Paradox table. Lost records will cause the index to get out of sync with the data, which will at some stage be followed by index corruption.

In some cases, bad programming is the cause of the problem. It is important to ensure that records are posted. When you terminate a program, it is the responsibility of the developer to `Post` all un-posted records before the program is terminated, otherwise you will get a 'record loss' problem. This could be achieved by applying something like the following statement for all your table objects in each form's `OnClose` event handler:

```
if TableObject.State in
  dsEditModes then
  TableObject.Post;
```

The program development phase is the time when most tables start inheriting corruption (caused by the developer resetting programs from time to time) which may not become evident until the system is deployed. One possible way of overcoming the problem is to rebuild the indexes periodically. This can be done with a table restructure (using the BDE `DbiDoRestructure` function, or with the Database Desktop) and will often resolve index corruption. It can also be done using `TUtility`, which can resolve data corruption (`TUtility` comes with the full version of Paradox and was included on the disk with Issue 5 of *The Delphi Magazine*, or look on the CompuServe BDEVTOOLS forum).

An alternative, and perhaps more foolproof, way of fixing broken indexes would be to write a routine which physically erases the indexes (with `DeleteFile`) and recreates the indexes from scratch with the BDE `DbiRegenIndexes` call (which relies on the table being opened exclusively). If your tables use referential integrity, then deleting the indexes may cause a problem due to a special checksum in the table header. In these cases you will need to delete the indexes and the .VAL file and use `DbiDoRestructure` to regenerate all tables that are involved in the referential integrity relationship.

In the BDE Configuration application on all PCs that will run the program, set `Local Share` to `True` on the `System` page. This ensures that lock files are written to the local hard disk, thereby ensuring that applications on other machines will be able to find the lock files. This should only be necessary for the machine where the data resides, however the general opinion is that it should always be turned on, provided you have file sharing functionality loaded with either SHARE or VSHARE. On peer to peer networks, the default setting of `Local Share` is a common cause of data loss.

When `False`, `Local Share` instructs the BDE to assume that all access to tables on 'local drives' (a peer to peer LAN counts as 'local') will occur via the same instance of the BDE in memory, it therefore fails in a number of situations including:

- Peer to peer LANs;
- Two applications running in different memory spaces under Windows NT (and maybe OS/2);
- Running a 16-bit BDE app and a 32-bit BDE app on the same machine in Windows 95 or NT.

`SUBST` will disable 32-bit file access and may therefore slow the machine down. If 32-bit file access is disabled, VSHARE won't be loaded. Some would say that `SUBST` isn't very safe especially when used with Windows 95 and is probably provided by Microsoft for compatibility with old DOS applications. Apparently, there are little corners

where Paradox does not function correctly and some people don't trust the command to work for all flavours of Wintel operating systems.

As an aside, it has been observed that the latest 16 bit BDE (2.52, which ships with Paradox 7 for Windows 3.1x) has problems with Auto Refresh not occurring in Paradox 7 for Windows 3.1x on the server when the server is a Windows 95 machine. This may have ramifications for Delphi users.

Recent revisions of the BDE allow different Sessions/Users to reference the PDOXUSRS.NET file using different drive letters so long as the remainder of the path is identical. Since the server has shared its root directory there is therefore no need to use SUBST and put up with the associated drawbacks: simply set the server machine's IDAPI.CFG Net Dir setting (on the Drivers page in the PARADOX driver settings) to C:\MYDIR and the workstations' Net Dir to L:\MYDIR. You can do this in code by assigning a value to Session.NetFileDir if you want to avoid editing IDAPI.CFG.

Note that using the root directory for the NET file can confuse certain BDE revisions. It is advisable to avoid this as a matter of policy even though the current revision seems happy with it. Also, if the user of the server does not want to share the entire C drive, it might be better to create a small partition for the Net Dir location.

If Local Share is True then the BDE will detect an incorrect NetFileDir and refuse to access the tables. If an incorrect assignment here is causing corruption then Local Share is still the real culprit.

On the Aliases page of the BDE Configuration application on all PCs in that will run the program, ensure the alias's Path points to the same network data directory.

Ensure that all users have their own private directory, preferably local. This is set with the Session object's PrivateDir property. Note that the online help specifies that if there will be multiple instances of the program running simultaneously on any one machine you should make sure each instance is given a unique path to avoid interference between temporary files from the different instances.

Call DbiSaveChanges after each table post (done simplest by putting the call in the table's AfterPost event handler). This should be unnecessary if the local share option has been set properly. When the BDE knows that the Paradox table is on a network, each record is saved to disk automatically. Therefore, DbiSaveChanges may only be necessary for saving local tables. There are two cases where a call to DbiSaveChanges can be a definite life saver: when you empty a table and when you restructure/pack a table (using DbiDoRestructure); this is because the actual table file is deleted and recreated but isn't necessarily committed to disk.

Check your other software/hardware caching as delayed writes are not good news on a network.

Instead of repeated calls to DbiSaveChanges, call DbiUseIdleTime in the Application's OnIdle event handler (also set the event handler's Done parameter to True). A call to DbiUseIdletime writes one dirty buffer to disk. Putting it in the OnIdle event means buffers will be written whenever your program is waiting for user input. Avoid using both DbiSaveChanges and DbiUseIdletime as they both do the same thing and so you'll be causing excessive function calls. This routine is becoming very popular as a general alternative to DbiSaveChanges, as it requires much less coding to use.

Have SHARE loaded with parameters of /F:4096 /L:40 as recommended by Borland. This advice is generally for Windows 3.10 users only. VSHARE from Windows for Workgroups supersedes SHARE: it's much better, although there is a caveat. Apparently VSHARE is a 32-bit driver which won't work with 16-bit disk controllers/drives which are present on non-local bus IDE PCs. In those circumstances, excluding SHARE from AUTOEXEC.BAT, and enabling VSHARE from Control Panel causes an error from IDAPI indicating SHARE isn't loaded.

Write a message handler for wm_EndSession in your main form class. Delphi 1 doesn't automatically handle this message which is sent when Windows is shut down by the user (although Delphi 2 does). Consequently, if a Delphi app is running when Windows is terminated, it won't be closed properly, and so BDE buffers may remain unwritten. It would be good practice to call Halt on receipt of a wm_EndSession message handler. Halt is not normally an advisable way to close a program, usually we use Application.Terminate. However that operates by posting (as opposed to sending) a wm_Quit message and so won't get around to doing what it needs to before Windows is gone. Halt causes exit procedures to be called, including the one in the DB unit which frees the Session object, thereby closing down the BDE in a proper fashion.

Listing 1 shows part of a form unit which takes up some of these suggestions. This code is from the project LOSS.DPR on the disk.

DLL Debugging Issues

Q How do I debug a DLL that I write? How do I debug a Delphi-written DLL being used by another program (eg a Visual Basic app)? How do I debug the design-time behaviour of the components (or experts) I write? *[These questions all came in separately, but can all be answered together. Editor]*

A Delphi's integrated debugger cannot debug DLLs (it looks like Delphi 3's will be able to) so you have to resort to other means. You need to get hold of an appropriate version of Turbo Debugger for Windows (TDW) that understands the debugging information generated by Delphi. When compiling the DLL don't forget to go to the Linker page of the Project Options dialog and turn on the Include TDW debug info switch. This will allow TDW to operate sensibly.

Delphi 1 programs require a 16-bit version of TDW and Delphi 2 apps need the 32-bit version (called TD32). If you have Borland C++ 5 then you'll have these

```
  TForm1 = class(TForm)
  ...
  public
    procedure DoIdle(Sender: TObject; var Done: Boolean);
   {$ifdef VER80}
    procedure WMEndSession(var Msg: TWMEndSession);
      message wm_EndSession;
   {$endif}
  end;
...
uses DbiProcs;
...
procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnIdle := DoIdle;
end;
procedure TForm1.DoIdle(Sender: TObject; var Done: Boolean);
begin
  { Each idle period, write a dirty buffer to disk }
  DbiUseIdleTime;
  Done := True;
end;
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var Loop: Integer;
begin
  {Generic way of ensuring all table changes are saved when form is closed}
  for Loop := 0 to ComponentCount - 1 do
   if Components[Loop] is TDataSet then
     with TDataSet(Components[Loop]) do
       if State in dsEditModes then
         Post;
end;
{$ifdef VER80}
procedure TForm1.WMEndSession(var Msg: TWMEndSession);
begin
  { If session is ending, call Halt to get exit routines executed. The DB
    unit's exit routine frees the Session object, which will unload the
    BDE, flushing any unsaved changes to disk }
  if Msg.EndSession then
    Halt;
end;
{$endif}
```

➤ *Listing 1*

versions. Similarly, if you purchase a copy of Turbo Assembler 5 (rather cheaper than Borland C++) then you also get the appropriate versions. So how does it work? It will take you a while to get used to Turbo Debugger, but you should persevere: some of the keystrokes are different, and it works in text mode, rather than using the standard Windows API.

When using Delphi's integrated debugger to debug a program that calls a routine in a DLL, if you try to trace into the DLL routine (F7) you get the same effect as if you'd pressed F8 (step over): you skip straight past it. However when you do the same in Turbo Debugger, provided the source for the DLL is available, you step into the source code for the DLL routine so seamlessly that you wouldn't realise it was in a separate module.

So now onto the question about debugging a DLL used by a non-Delphi program (eg one written in C++, VB, Paradox, Visual dBASE). The same question applies to debugging a DLL that is used by a Delphi program that was compiled with no debugging information. The question about debugging design-time component behaviour boils down to the same thing as well, since your component source is compiled into the 32-bit CMPLIB32.DCL or 16-bit COMPLIB.DCL (which are DLLs with different extensions). To ensure TDW debug information is generated for the component library, go to the `Library` page of the `Options|Environment` dialog (Delphi 1) or `Tools|Options` dialog (Delphi 2) and choose `Compile with debug info`.

The ten steps go like this:

1. Ensure debugging information is included in the DLL using either the appropriate linker option, or the option on the `Library` page for the component library.

2. Load Turbo Debugger (TDW.EXE or TD32.EXE), from Windows Explorer or Program Manager.

3. Remove the opening dialog box by pressing `OK`.

4. Choose `File|Open...`, press the `Browse` button, locate the program that uses the DLL and press `OK`. For debugging components this will be DELPHI.EXE or DELPHI32.EXE as appropriate.

5. Turbo Debugger will tell you the program has no symbol table: that says there is no debugging information in the EXE. Press `OK`.

6. Choose `View|Module...` (F3) and enter the name of your DLL (eg CMPLIB32.DCL), but no path, into the DLL name edit box and press the `Add DLL` button. It will be added into the DLLs and programs list with a double exclamation mark symbol if the DLL is not yet in memory.

7. Ensure `Load Symbols` and `Debug startup` are both set to `Yes` and press `Cancel` to shut the dialog.

8. Continue running the program with `Run|Run` (F9) and when the DLL gets loaded, Turbo Debugger pops back up.

9. Now you can choose `View Module...` (F3) and see all the source files from the DLL that have debug information available.

10. Choose the appropriate file and place breakpoints as required (with `Breakpoints|Toggle` or F2) or add watches (`Data|Add watch`, or F7) and continue running (F9).

Annoying Workgroup Menu

Q When I installed Delphi 2 I got the `Workgroup` menu added. I don't currently use the version control software but the menu is very annoying. When the mouse brushes over it, there is a large delay before the menu first drops down, presumably caused by the version control link DLL and version control software being loaded. What do I need to do to remove the menu?

A Load up the registry editor (REGEDIT.EXE) and navigate down the following path:

```
HKEY_CURRENT_USER\Software\
 Borland\Delphi\2.0
```

Select the Version Control key and note down the current value of VCSManager (you'll need this to restore things if you need the menu back later). By default VCSManager points to STDVCS32.DLL in Delphi 2's BIN directory. Having selected the Version Control key, delete it with the Delete key. Next time Delphi 2 loads, the menu will be gone.

Making A Unique Key Value

Q There seems to be no well defined way to make a unique key field value for a record: there is no obvious way to define a unique Longint (say) serial number for a new record (RecordCount is useless as it only looks at the current size of the table and with deletions over time this is not unique). Once a new record is being built (say, because the user has hit the + on a DBNavigator) there is no obvious way to test a new key for uniqueness by using eg LookUp() in a BeforePost event, since the act of calling LookUp() on the table attempts a post on the new record and one recurses indefinitely until stack overflow!

A There would be some who would say that ensuring a unique key value is unique when you make a new record doesn't really work as well as you might want. When you come to post the record, someone may have already posted a record with the same value (which they would have found to be unique as your record had not been posted at that time). This would give a key violation!
But anyway, you've asked the question about calculating a unique value, so let's use your scenario to look at the question. The key to this is to use another TTable object to do the searching. Since you mentioned the dataset Lookup method, you must be using Delphi 2 (it's new in that version). This solution uses that method and so is restricted to Delphi 2.

Listing 2 shows a function called Unique that takes a TTable object, a field name and a potential value that needs checking for uniqueness. If that number is not unique it finds the nearest higher value that actually is unique and returns it. Also in the listing is in an OnNewRecord event handler for a table that calls Unique. It passes the current record count as a possible unique value and then Unique returns a definitely unique value that is greater than or equal to it. This value is placed in an appropriate field object. The sample project UNIQUE.DPR implements these routines.

The Millennium Bug

Q How can I get Delphi to display database dates with a four digit year (instead of two), to avoid the millennium bug?

A There are several ways to achieve this. If you don't have many date fields, then you can set the DisplayFormat properties of the field objects to an appropriate string like dd/mm/yyy or mm/dd/yyy. Alternatively you can change things on a more global basis. The default date representation comes from the short date setting in the Control Panel international or regional settings.

You can either change the setting in Control Panel, or to be less destructive to all Windows applications you can make a change in your Delphi code. When your application starts, the initialisation code of the SysUtils unit reads the Control Panel setting into the ShortDateFormat variable in a procedure called GetFormatSettings. In your main form's OnCreate event handler, or in one of your form units' initialisation sections you can turn the two digit year code (yy) into a four digit code (yyy or yyyy).

Bearing in mind the variety of possible short date formats, we need some algorithm to do the replacement of the year section, which could be in any part of the string. Listing 3 shows a potential form OnCreate handler that seems to do the job.

➤ *Listing 2*

```
function Unique(Tbl: TTable; Fld: String; Value: Integer): Integer;
begin
  with TTable.Create(Application) do
    try
      DatabaseName := Tbl.DatabaseName;
      TableName := Tbl.TableName;
      Open;
      repeat
        Inc(Value)
      until Lookup(Fld, Value, Fld) = Null;
      Result := Value
    finally
      Free
    end
end;

procedure TForm1.Table1NewRecord(DataSet: TDataSet);
begin
  { Ensuring uniqueness on a new record doesn't say much for its state when
    posted. Someone else may have already posted that supposedly unique
    value }
  Table1['CustNo'] := Unique(Table1, 'CustNo', Table1.RecordCount);
end;
```

➤ *Listing 3*

```
procedure TForm1.FormCreate(Sender: TObject);
var Index: Integer;
const
  TwoDigitYear = 'yy';
  FourDigitYear = TwoDigitYear + 'y';
begin
  if Pos(FourDigitYear, ShortDateFormat) = 0 then begin
    Index := Pos(TwoDigitYear, ShortDateFormat);
    if Index <> 0 then
      { Insert an extra letter 'y' }
      Insert('y', ShortDateFormat, Index)
  end
end;
```