The Delphi CLINIC

Edited by Brian Long

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

UNC

Q I need to store file names and paths in a database so that anyone on my network can access them. The trouble is that the `TDirectoryListBox` and `TFileListBox` only deal in paths with drive letters in. How do I turn these into UNC format so that each computer can access the files regardless of their drive mappings?

A Check out the `ExpandUNCFileName` function available in Delphi 2 and 3. This should do it, however it does leave local drives with drive letters in, so you would have to substitute the computer name as appropriate.

Unexpected Disk Space

Q I have recently reconfigured my machine to merge my two hard drives into one volume set (now 3Gb) using the new NT 4.0. Unfortunately when I use Delphi 2's `DiskFree` on this new drive I get negative results. Is there any way to retrieve the amount of free space on a volume set or will I have to reconfigure my drives back to disk partitions?

A The trouble is that `DiskFree` has an `Integer` return type. The biggest number that an `Integer` can represent is 2Gb-1. You can modify the `DiskFree` function from the `SysUtils` unit to use floating point numbers instead as shown in Listing 1 and the problem will go away.

Calling Into DLLs

Q I've noticed an undocumented API call in Windows 95 called `WNetGetCachedPassword`, it's in MPR.DLL. How can I drive this from Delphi?

A You first need to get some documentation for it. If you haven't got any, you might as well forget it. Reverse engineering assembler code to find what the parameters and return values are supposed to be is very difficult.

If you have documentation on the parameter types then you need to write an import declaration for it. All the units in the SOURCE\RTL\WIN directory of all versions of Delphi except Desktop are packed full of import declarations so you can check those out to see how they are written. Or check any book that goes into detail about Delphi DLL programming.

Of course if you just want to get the network password, then check the appropriate Delphi Clinic entry on the subject in Issue 15.

Context Help

Q Many 32-bit Windows applications have the little question mark button on the caption bars of their dialogs to induce convenient context-sensitive help. Pushing this changes the cursor to a question mark (as Delphi does when a `Cursor` property is set to `crHelp`) and the user can then click on a control to get help in a nice little popup window (as opposed to the full Help application). I want this functionality, but I also want minimise and maximise buttons on my form. It seems I cannot have all three. What can I do about this?

A You are right in that the `BorderIcons` set property of a form has certain priorities over which buttons it will put on the form (or rather the underlying Windows implementation does). If you include `biMaximize`, `biMinimize` and also `biHelp`, then the help icon does not show. In fact it will only show if `biMinimize` and `biMaximize` are both removed. However, you could add a button onto your form which has the same functionality as the caption bar button or border icon concerned.

All the things on the caption bar cause `wm_SysCommand` messages to be sent to the form. The `WParam` part of the message contains the actual system command to execute. For example, these two statements mimic the minimise and maximise buttons:

➤ *Listing 1*

```
function DiskFree(Drive: Byte): Single;
var
  RootPath: array[0..4] of Char;
  RootPtr: PChar;
  SectorsPerCluster, BytesPerSector,
  FreeClusters, TotalClusters: Integer;
  SPC: Single;
begin
  RootPtr := nil;
  if Drive > 0 then begin
    StrCopy(RootPath, 'A:\');
    RootPath[0] := Char(Drive + $40);
    RootPtr := RootPath;
  end;
  if GetDiskFreeSpace(RootPtr, SectorsPerCluster, BytesPerSector,
    FreeClusters, TotalClusters) then begin
    SPC := SectorsPerCluster;
    Result := SPC * BytesPerSector * FreeClusters
  end else
    Result := -1;
end;
```

```
Perform(wm_SysCommand,
  sc_Minimize, 0);
  //sc_Minimize replaces sc_Icon
Perform(wm_SysCommand,
  sc_Maximize, 0);
  //sc_Maximize replaces sc_Zoom
```

This assumes these are called in the scope of the form, like in an event handler. Perform is a method of any control and in this case I am calling the form's `Perform` method.

You can send `sc_Close` to a form to close it (although you never would because calling the `Close` method works just fine). Things get a bit more interesting with `sc_Size` and `sc_Move` so it's worth moving off on a slight tangent for a while before coming back to the question proper.

The Windows API help says that if you trap for `wm_SysCommand` you should perform a binary and operation between the `WParam` and `$FFF0` and check whether the result matches the code you want to trap. The reason for this is that each of the codes themselves have their lowest four bits clear so that additional information can be passed along if necessary in these spare bits. `sc_Move` and `sc_Size` take advantage of this potential extra information.

If you send a normal `sc_Move` code to a form (or indeed to any control for that matter), the cursor changes to the shape you get if you choose Move from the system menu. This allows keyboard actions to move the form (or control) around. However if you send `sc_Move+1` you get the same effect as when you click on the caption bar of a form and drag it with the mouse. In other words, when you send this new command any further mouse movements move the target window until you do something with the mouse button (like click it).

Similarly when you send `sc_Size` it allows the cursor keys to resize the item. However eight of the values that follow `sc_Size` allow the mouse to resize the window. They mimic the various sections of the form border that you drag to resize the form in the various ways. Listing 2 defines a set of new constants that represent these new valid system codes. It's the ones that start `sc_Size...` that can be used for sizing. So `Perform(wm_SysCommand, sc_SizeBottomRight, 0)` matches what happens when you drag the bottom-right hand portion of a form's border.

Getting back to the plot with respect to the context help button, you can call

```
Perform(wm_SysCommand,
  sc_ContextHelp, 0)
```

in a button's `OnClick` handler to do the same as the help icon would. Or almost the same. The difference is that the help comes up in the full WinHelp application. To get the proper effect make sure you include `biHelp` in the form's `BorderIcons` property. Windows will then invoke the help in the pleasant little popup when you send the message.

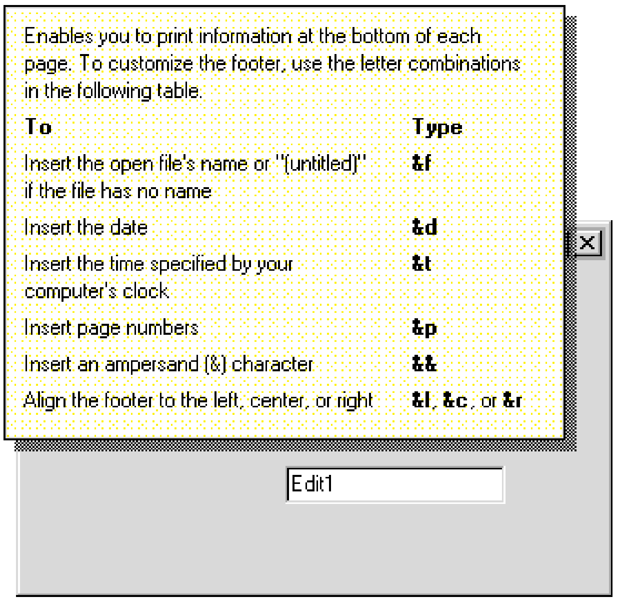
Figure 1 shows an application (SysCmd.Dpr) which has borrowed Notepad's help file. Some of Notepad's context numbers have been given to the controls on the

```
const
  sc_DragMove = sc_Move + 1;
  sc_Left = 1;
  sc_Right = 2;
  sc_Top = 3;
  sc_Bottom = 6;
  sc_SizeKeys = sc_Size + 0;
  sc_SizeLeft = sc_Size + sc_Left;
  sc_SizeRight = sc_Size + sc_Right;
  sc_SizeTop = sc_Size + sc_Top;
  sc_SizeBottom = sc_Size + sc_Bottom;
  sc_SizeTopLeft = sc_SizeTop + sc_Left;
  sc_SizeTopRight = sc_SizeTop + sc_Right;
  sc_SizeBottomLeft = sc_SizeBottom + sc_Left;
  sc_SizeBottomRight = sc_SizeBottom + sc_Right;
```

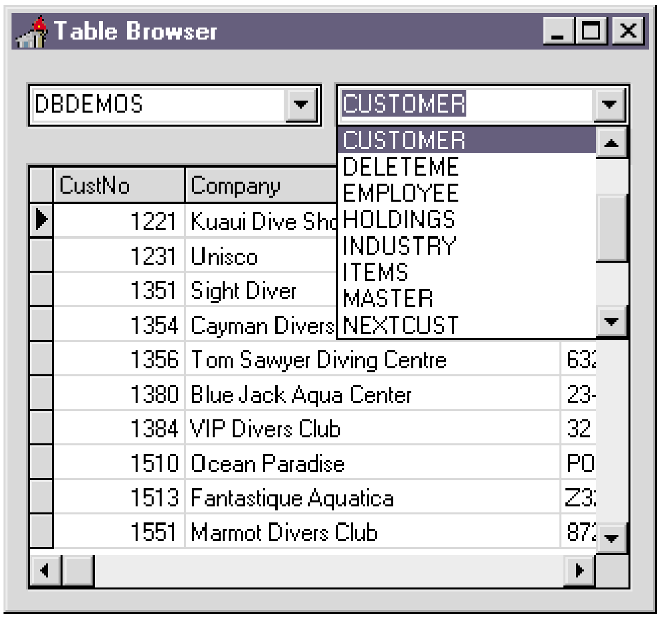
➤ *Listing 2*

➤ *Listing 3*

```
const Delta = 5;
function GetSizeCommand(X, Y: Integer; Control: TWinControl): Cardinal;
begin
  Result := sc_Size;
  if X > Control.ClientWidth - Delta then
    Inc(Result, sc_Right)
  else if X < Delta then
    Inc(Result, sc_Left);
  if Y > Control.ClientHeight - Delta then
    Inc(Result, sc_Bottom)
  else if Y < Delta then
    Inc(Result, sc_Top)
end;
procedure TForm1.Edit1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var SystemCode: Cardinal;
begin
  with Sender as TWinControl do
    if ChkDesign.Checked and (ssCtrl in Shift) then begin
      SystemCode := GetSizeCommand(X, Y, TWinControl(Sender));
      if SystemCode <> sc_Size then begin
        { Do fake mouse button release, since the control won't notice the
          real one due to its resizing }
        Perform(wm_LButtonUp, X, Y);
        Perform(wm_SysCommand, SystemCode, 0);
      end
    end else begin
      ReleaseCapture;
      Perform(wm_SysCommand, sc_DragMove, 0);
    end;
end;
procedure TForm1.Edit1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
var NewCursor: TCursor;
begin
  NewCursor := crDefault;
  with Sender as TWinControl do begin
    if ChkDesign.Checked and (ssCtrl in Shift) then
      case GetSizeCommand(X, Y, Sender as TWinControl) of
        sc_SizeTop, sc_SizeBottom: NewCursor := crSizeNS;
        sc_SizeLeft, sc_SizeRight: NewCursor := crSizeWE;
        sc_SizeTopLeft, sc_SizeBottomRight: NewCursor := crSizeNWSE;
        sc_SizeTopRight, sc_SizeBottomLeft: NewCursor := crSizeNESW;
      end;
    Cursor := NewCursor
  end
end;
```


Enables you to print information at the bottom of each page. To customize the footer, use the letter combinations in the following table.

To	Type
Insert the open file's name or "(untitled)" if the file has no name	&f
Insert the date	&d
Insert the time specified by your computer's clock	&t
Insert page numbers	&p
Insert an ampersand (&) character	&&
Align the footer to the left, center, or right	&l, &c, or &r



CustNo	Company	
1221	Kuaui Dive Sho	
1231	Unisco	
1351	Sight Diver	
1354	Cayman Divers	
1356	Tom Sawyer Diving Centre	632
1380	Blue Jack Aqua Center	23-
1384	VIP Divers Club	32
1510	Ocean Paradise	PO
1513	Fantastique Aquatica	Z32
1551	Marmot Divers Club	872

➤ *Above left: Figure 1*
➤ *Above right: Figure 2*

form via their `HelpContext` properties. Additionally, SysCmd.Dpr shows how `sc_Move` and `sc_Size` and their variations can be used. You can drag the edit control on the form around by standard click-and-drag operations. Ctrl-click-and-drag allows the edit control to be resized. To suggest that resizing is available inside the perimeter of the edit control, the cursor is changed in much the same way as it is when over a normal form's border. The code for this is a bit clunky, see Listing 3.

Incidentally, a note for the unwary regarding help files. When you use the `Help file:` option on the `Application` page of the `Project Options` dialog, the net effect is that an assignment to `Application.Helpfile` is inserted into the project source file. The trouble is that when you browse for a help file, a fully qualified path is used in the assignment. This is not usually a good idea. And it is not necessarily required in a Win32 application. Win32 keeps a record of all the help files that are used and stores the path of where they can be found in the registry. This is in fact why you get help file problems with Delphi 2 and 3 on the same machine. They have the same help file names, but in different directories. The registry stores only one entry per help file name.

Anyway, during your application's installation you could add the path of the help file into the registry under

```
HKEY_LOCAL_MACHINE\Software\
  Microsoft\Windows\Help
```

and you can then omit the path information from the `Application.HelpFile` assignment.

Table Names

Q I am using an MS Access database in my project (ODBC). The user can create and view tables and so I need to provide a drop down list of the tables in the MS Access database. However I cannot see any way of listing the names of the tables, how do I find the table names. I guess it can be done because the property editor of `TTable` can do it.

A Indeed. You need to use methods of your database application's Session object to achieve this. Listing 4 shows some event handlers from Tables.Dpr that do the business (see Figure 2).

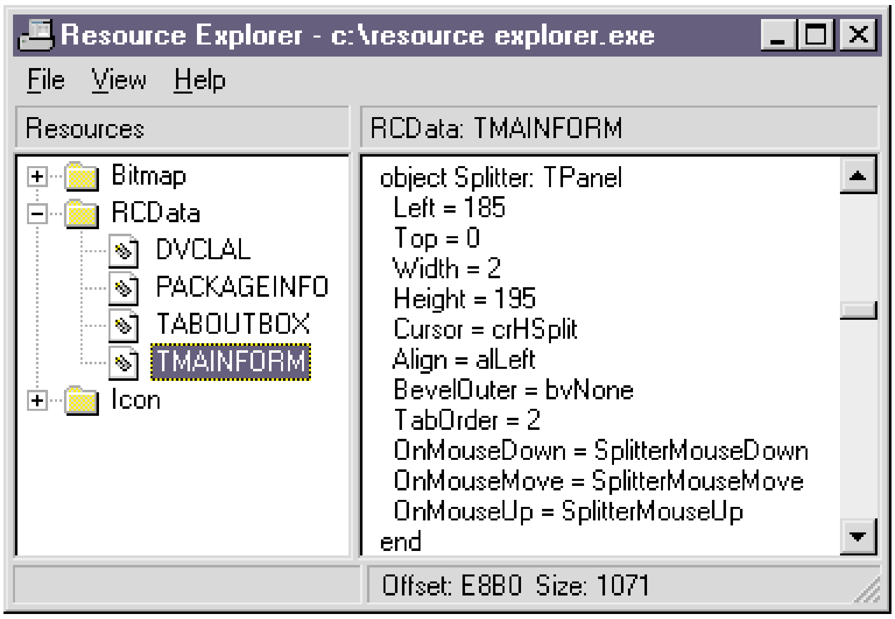
Forms Compiled Into Exes

Q I know I can use CONVERT.EXE or the `ObjectResourceToText` procedure to translate DFM files into text. But can I examine the forms that have been compiled into executable files?

A Well this must verge on reverse engineering, but yes you can. DFM files are linked in as custom resource files and so reside at the end of the executable file with all the other resources. They are stored as `RCData`, along with other custom resource data which may well not be forms.

➤ *Listing 4*

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Session.GetAliasNames(cmbAlias.Items);
end;
procedure TForm1.cmbAliasChange(Sender: TObject);
begin
  Session.GetTableNames(cmbAlias.Text, '*.*', False, False, cmbTable.Items)
end;
procedure TForm1.cmbTableChange(Sender: TObject);
begin
  Table1.DisableControls;
  try
    Table1.Close;
    Table1.DatabaseName := cmbAlias.Text;
    Table1.TableName := cmbTable.Text;
    Table1.Open
  finally
    Table1.EnableControls
  end
end;
```


Resource Explorer - c:\resource explorer.exe

File View Help

Resources | RCData: TMAINFORM

- Bitmap
- RCData
  - DVCLAL
  - PACKAGEINFO
  - TABOUTBOX
  - TMAINFORM
- Icon

```
object Splitter: TPanel
  Left = 185
  Top = 0
  Width = 2
  Height = 195
  Cursor = crHSplit
  Align = alLeft
  BevelOuter = bvNone
  TabOrder = 2
  OnMouseDown = SplitterMouseDown
  OnMouseMove = SplitterMouseMove
  OnMouseUp = SplitterMouseUp
end
```

Offset: E8B0 Size: 1071

➤ *Figure 3*

What you need is some code set up to access resources, and then continue to use `ObjectResourceToText` on the located data. Fortunately (if you are using Delphi 2 or 3) you do have such code in the form of a demo application. The RESXPLOR demo is a good example of a program that pulls a Win32 EXE or DLL to bits.

To modify it to show textual versions of the forms therein, load the project from the Delphi's DEMOS\RESXPLOR directory and find the implementation of the `UpdateViewPanel` method of the main form. Declare two new local variables as follows:

```
 BinaryStream, TextStream:
   TMemoryStream;
```

Then in the case statement, add a new choice in, just before the else part. Listing 5 shows how the code gets inserted. The job is now done. Run the program, load up a Delphi EXE and look at the `RCData` section. Any items which represent forms can now be viewed at leisure. Figure 3 shows the Resource Explorer examining its own EXE, looking at its main form. This modified Resource Explorer was used to aid in the development of Archaeopteryx (see my article in this issue).

➤ *Listing 5*

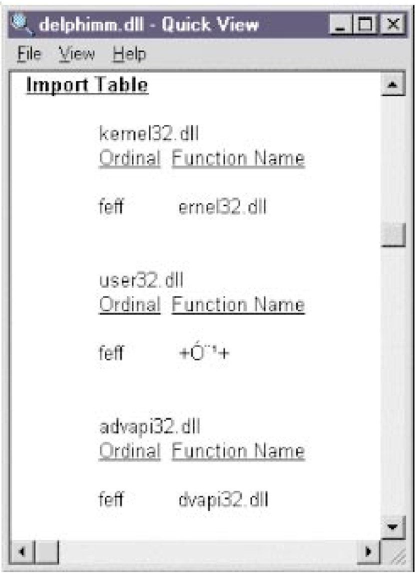
```
...
  rtString, rtMenu:
    begin
      StringViewer.Lines.Assign(R);
      StringViewer.SelStart := 0;
      Notebook.PageIndex := 2;
    end;
  {beginning of new code}
  rtRCData:
    begin
      TextStream := TMemoryStream.Create;
      BinaryStream := TMemoryStream.Create;
      try
        R.SaveToStream(BinaryStream);
        BinaryStream.Position := 0;
        try
          ObjectBinaryToText(BinaryStream, TextStream);
          FormText.Lines.Clear;
          TextStream.Position := 0;
          FormText.Lines.LoadFromStream(TextStream);
          Notebook.PageIndex := 4;
        except
          R := TResourceItem(Selected.Data);
          HexDump.Address := R.RawData;
          HexDump.DataSize := R.Size;
          Notebook.PageIndex := 3;
        end
      finally
        TextStream.Free;
        BinaryStream.Free
      end
    end;
  {end of new code }
  else begin
    HexDump.Address := R.RawData;
    HexDump.DataSize := R.Size;
    Notebook.PageIndex := 3;
  end;
end;
...
```

DLL Imports

Q I've been examining my Delphi applications with QuickView (supplied with Windows 95). Why do Delphi apps have a garbled import table in QuickView, whereas those produced by other tools seem fine?

A This seems to be a problem with QuickView. All Delphi binaries seem to cause QuickView a problem (Figure 4), but they clearly work ok at run-time (which implies Windows can read the import table correctly). A different tool such as `TDump` accurately reflects that delphimm.dll (shown in Figure 4 in QuickView) imports things from kernel32.dll, user32.dll, advapi32.dll, oleaut32.dll, and moreover lists which functions are imported from the DLLs (QuickView can usually manage this but not with Delphi binaries).

➤ *Figure 4*


delphimm.dll - Quick View

File View Help

Import Table

kernel32.dll

Ordinal	Function Name
feff	ernel32.dll

user32.dll

Ordinal	Function Name
feff	+Ó¨'+

advapi32.dll

Ordinal	Function Name
feff	dvapi32.dll