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

Multimedia Help

Q I came across the routine `sndPlaysound` in a Delphi book and as I'm interested in including sound effects in my programs I tried to look it up in the Delphi API help. For some reason it isn't shown. Can you throw any light on this?

A `sndPlaySound` is a multimedia routine. The multimedia API (as declared in the `MMSystem` unit) is not considered part of the core Windows API and so lives in its own help file. This is file `\DELPHI\BIN\MMSYSTEM.HLP` in Delphi 1 or in Delphi 2 `\Delphi2.0\Help\MM.HLP`. Delphi 1 does not search through the multimedia help file (Delphi 2 does), though you can add it as another help icon in your Delphi Program Manager (or Windows Explorer) group.

Waiting For DOS

Q I need to call a DOS program in order to get some data which I can then process for output. I need to wait for the DOS program to terminate, because I will read its output and if I try to read the file before it gets closed, I get a sharing violation. How can I be sure that the DOS program has finished? I do not want to ask the user to push a button when it has finished – that is very inelegant.

A The most common approach to doing this is, instead of running the DOS .EXE, you run a batch file in the same directory. The batch file's job is to write a file to the disk, run the DOS .EXE, and then delete the file.

Your program uses `WinExec` to run the batch file, then does a Windows-friendly loop until the file exists (or some timeout occurs, or the user gets bored and terminates your app). When the file appears, you presume the program has started. You then do another loop waiting for the file to disappear. When that loop terminates, the program and the batch file have finished. An example batch file would look like:

```
@echo off
echo Semaphore > semafore.dat
command.com
del semafore.dat
```

A routine to use the batch file could look like Listing 1.

Notice that the batch file isn't executed directly, but via a DOS command shell. This means that when it terminates the DOS window will close automatically (presuming the Windows session has that option enabled).

Incidentally, there are plenty more ways of detecting when a program has finished given in *The Revolutionary Guide To Delphi 2* (which also covers version 1) by various authors, published by Wrox Press, ISBN 1-874416-74-5. *[OK, Brian, plug permitted, but you owe me a pint from the royalties... Editor]*.

OnExit And The Lost Caret

Q I do validation of `TEdit` and `TDBEdit` controls in their `OnExit` handlers. If there is a problem, I show a message box. However, the edit control that should then receive focus has no caret, although text can still be typed in. The caret won't come back until another normal focus change occurs.

A This problem occurs with most 16-bit Windows programming tools. If you invoke a dialog when focus is being switched to edit controls, the caret gets lost. Win32 fixes this. Principally, the workaround in a 16-bit program is to set focus back to the

➤ *Listing 1*

```
procedure TForm1.Button1Click(Sender: TObject);
var
  OldTime: TDateTime;
const
  Startup = 5;
  Semaphore = 'semafore.dat';
begin
  WinExec('command.com /C \temp\delme.bat', sw_ShowNormal);
  Button1.Enabled := False;
  OldTime := Time;
  repeat
    Application.ProcessMessages
  until Application.Terminated
    or (Time > OldTime + EncodeTime(0, 0, Startup, 0))
    or FileExists(Semaphore);
  if FileExists(Semaphore) then begin
    repeat
      Application.ProcessMessages
    until Application.Terminated or not FileExists(Semaphore);
    if not FileExists(Semaphore) then
      ShowMessage('It''s finished!');
  end;
  Button1.Enabled := True;
end;
```

control losing focus. So instead of:

```
procedure TForm1.Edit1Exit(
  Sender: TObject);
begin
  ShowMessage('No!');
end;
```

you use:

```
procedure TForm1.Edit1Exit(
  Sender: TObject);
begin
  ShowMessage('No!');
  if Sender is TEdit then
    TEdit(Sender).SetFocus;
end;
```

Error-Free Custom Open File Dialogs

Q I am trying to implement my own file open dialog using a `TDirectoryList`, `TFileListBox` and `TDriveComboBox`. I experience three problems. Firstly, whenever I try and go to drive A: when there is no floppy in the drive, I get the Windows system error message. This one I can fix using the `SetErrorMode` tip in Issue 7's Clinic. When I get rid of this, I get a Delphi `EInOutError` exception (invalid filename) instead. I can't see any place to put a `try..except` block, as the exception occurs inside one of the components, and I am not using any code to manipulate them. The last problem occurs if I do successfully log onto the A: drive, take the floppy out, and try and go to another drive. This causes the same exception to happen from that point on.

A There are a number of problems here. The Windows modal error message can indeed be removed by using `SetErrorMode` as shown in the *Windows File Errors* Delphi Clinic entry you refer to. The other two are less easy.

The exception when going to drive A: can be trapped in a new component derived from `TDriveComboBox`. The exceptions when going from drive A: unfortunately need two VCL changes (or one if you are using Delphi 2).

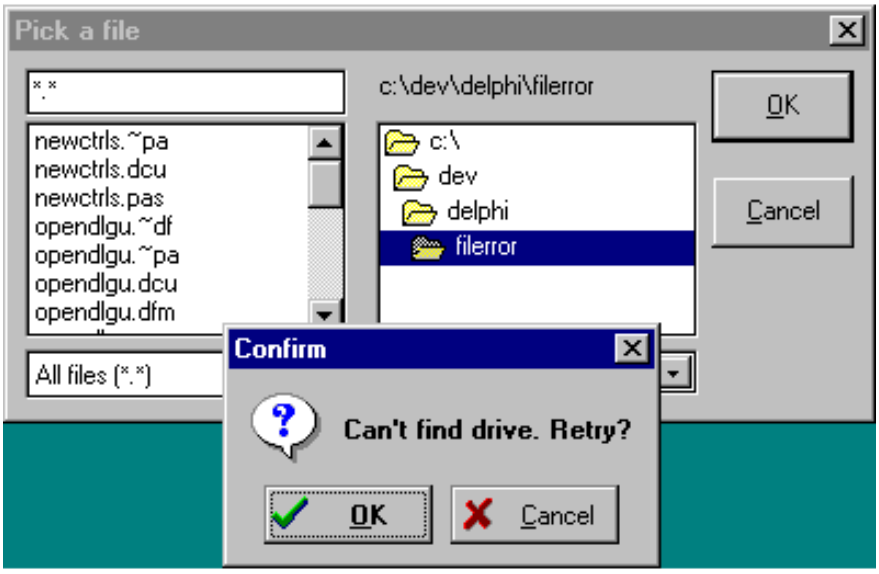
Let's take the new drive combo component for now. The exception that needs trapping occurs when the drive is changed. If this is being done visually, then that must mean when the drive entry in the drive combo is changed. Whether this is done by mouse or keyboard, the virtual `Click` method executes, so we can override this and trap the exception. In my component I have tried to make it flexible by adding a new event which is triggered when a drive error occurs. This allows the drive combo user to add code to invoke a retry/cancel dialog (see Figure 1).

The fly in the ointment comes when trying to cater for both Delphi 1 and 2. When the problem occurs in Delphi 1, an `EInOutError` exception, code 3, is raised. In Delphi 2 it is still an `EInOutError` exception, but code 21. To try and ensure I am only trapping the particular problem I want, I have used conditional compilation to check for the right code number. The `TNewDriveCombo` is shown in Listing 2. Install this into your component library in the usual way.

The last problem needs a fix or two to the VCL. When the file and directory list boxes are trying to go somewhere else, for some reason they first try and change to the current directory. If there is no floppy in the drive, this will cause an exception to occur each time you try and change drive. In Delphi 2 a fix has been applied to the directory list box, but not to the file list box. Fixing the file list is more important for the visual use of these components, but to cater for programmatic use, we need to fix both.

➤ *Listing 2*

```
unit Newctrls;
interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, StdCtrls, FileCtrl;
type
  TDriveErrorEvent = procedure(Sender: TObject;
    var Retry: Boolean) of object;
  TNewDriveCombo = class(TDriveComboBox)
  private
    FOnDriveError: TDriveErrorEvent;
  protected
    procedure Click; override;
  published
    property OnDriveError: TDriveErrorEvent
      read FOnDriveError write FOnDriveError;
  end;
procedure Register;
implementation
procedure TNewDriveCombo.Click;
var
  OldDrive, NewDrive: Char;
  Ouch, Retry: Boolean;
begin
  OldDrive := Drive;
  if Items.Count > 0 then
    NewDrive := Items[ItemIndex][1];
  Ouch := False;
  { If there's a problem (empty floppy for example)
    stop any exception message being printed }
  try
    inherited Click;
  except
    on E: EInOutError do
{$ifdef Windows}
      { DOS gives error 3 (Path not found)
        if drive not ready on a directory change }
      if E.ErrorCode = 3 then
{$else}
      { Win32 gives error 21 (Device not ready) if drive
        not ready; this is more precise - there is an
        equivalent error 3 defined. For a list of Win32
        errors, (albeit with missing characters) look up
        "error codes", "Error Codes (Win32 Programmer's
         Reference)" }
      if E.ErrorCode = ERROR_NOT_READY then
{$endif}
        { Signal to later code that a problem occurred }
        Ouch := True;
  end;
  if Ouch then
    repeat
      try
        { Set back decent drive first }
        Drive := OldDrive;
        { Then check for a retry }
        Retry := False;
        if Assigned(FonDriveError) then
          FOnDriveError(Self, Retry);
        if Retry then
          { Try setting target drive again }
          Drive := NewDrive;
        { If no exception occurs, we're done
          so set loop terminator }
        Ouch := False;
      except
        { Mask the potential problem }
        on E: EInOutError do
          if E.ErrorCode = 3 then;
      end;
    until not (Ouch and Retry);
end;
procedure Register;
begin
  RegisterComponents('Samples', [TNewDriveCombo]);
end;
end.
```



➤ *Figure 1*

```
procedure TForm1.FormCreate(Sender: TObject);
var SearchRec: TSearchRec;
begin
  if FindFirst('HELP????.HLP', faAnyFile, SearchRec) then
    Application.HelpFile := SearchRec.Name;
  FindClose(SearchRec);
end;
```

➤ *Listing 3*

In FILECTRL.PAS, locate the routine `TFileListBox.SetDirectory` and also `TDirectoryListBox.SetDir` and change `ChDir(FDirectory);` to:

```
{$ifdef BROKEN}
  ChDir(FDirectory);
{$else}
  {$I-}  { ignore errors }
  ChDir(FDirectory);
  {$I+}
  if IOResult = 0 then
    ; {clear error }
{$endif}
```

Do this for both routines. Once both these changes are made, copy the file into your \DELPHI\LIB directory and re-compile your project.

A project to show a rather more safe error box (providing the fixes described have been applied, and the new component has been installed) than you would normally get is given in TESTDLG.DPR on the disk. OPENDLGU.PAS implements the open dialog form and the drive combo's `OnDriveError` handler allows the user to retry if a drive is not available.

Multiple Choice Help Files

Q I have an application that uses a help file. I know that when the `HelpFile` option is set up in the project options dialog on the `Application` page, Delphi adds a statement to the project source:

```
Application.HelpFile :=
  'HELP.HLP';
```

The problem is that I will be replacing the help file each month with a different one (HELP0196.HLP, HELP0296.HLP etc) and wish to be able to do this without changing the Delphi program. Can I give wild cards to `Application.HelpFile`?

A No you can't. But we can solve your problem. You can search for an appropriate help file first, and if you find one, assign its name to `Application.HelpFile`. For example, you could add the code in Listing 3 into your main form's `OnCreate` handler.

Note the use of `FindClose`. In Delphi 1 this does nothing, but using it sets us up for Delphi 2 which does require it. Delphi 1 uses DOS interrupts to do file searching, which require no tidying up, whereas Delphi 2 uses the Win32 API which has very specific needs.

Oracle SQL*Net 2 And SQL Links

Q I can connect to Oracle using SQL*NET 1 but am having no joy with SQL*NET 2. How do I set up the Oracle SQL Link to talk to version 2?

A Unfortunately, the documentation assumes version 1 and so doesn't help much. But you can connect with version 2 by using these guidelines.

Either specify your TNS (Transparent Network Substrate) alias as the server name and specify the network protocol to be TNS, *or* use @TNS:<TNS Alias> for the server name (do not include the angle brackets) and leave the network protocol blank.

Broken Minimising Behaviour

Q In Delphi 2, a normal program will minimise correctly into the taskbar. However, if you set your main form's `WindowState` property to `wsMinimized`, when the program starts the form is minimised like an MDI child, sitting at the bottom of the desktop. Is there a fix available?

A Roy Nelson at Borland suggests the following. Declare a method `FormRestore` in your form class's `public` section, then add an `OnCreate` handler for the form. Set `FormRestore` and `FormCreate` up as shown in Listing 4.

Synchronised Listboxes

Q I'm trying to get two listboxes to synchronise their scrolling. In other words when one is scrolled, I want the other to scroll to the same place. There doesn't seem to be an appropriate event to help me out here.

A You're right – we'll have to make one. Or two as it turns

```
TForm1 = class(TForm)
...
public
  procedure FormRestore(
    Sender: TObject);
end;
...
procedure TForm1.FormRestore(
  Sender: TObject);
begin
  Perform(
    wm_SysCommand, sc_Restore, 0);
  Application.OnRestore := nil;
end;
procedure TForm1.FormCreate(
  Sender: TObject);
begin
  if WindowState =
    wsMinimized then begin
    Application.ShowMainForm :=
      False;
    Application.OnRestore :=
      FormRestore;
    Application.Minimize;
  end;
end;
```

➤ *Listing 4*

out. Principally, we need to trap whatever Windows message is sent when the listbox scrolls. Unfortunately it's not just one message. When the scrollbar is dragged around with the mouse, a `wm_VScroll` message is sent to the listbox and the listbox's view of its contents changes (although the focused item does not change). The online help for this message discusses the information that comes along with this message.

If the user clicks on a listbox item and drags the mouse up or down, this can also cause scrolling, but the `wm_VScroll` message does not get sent under these circumstances. Similarly, if the user uses the arrow keys, or `Page Up/Down`, or `Home` or `End`, the listbox can scroll but no `wm_VScroll` message is seen. Instead, in these other circumstances, a component notification message is sent to the listbox – this is a `cn_Command` message with a `lbn_SelChange` parameter, ie the listbox selection has changed.

The component implemented in SLISTBOX.PAS and shown in Listing 5 traps both these messages and triggers events (`OnScroll` and `OnSelChange`) for them, if they have been set up. A user of a `TSListbox` component can make event handlers for these events and set any other listbox up to match its current state. The project LISTBOX.DPR does just this. It has two `SListboxe` controls side by side, and they mimic each other. Both share `OnScroll` and `OnSelChange` event handlers, and in the case of the latter event handler I have offered two choices of code to use.

Listing 6 shows the two event handlers from LISTBOXU.PAS. When an `OnScroll` event occurs, the `TopIndex` property of the mimic listbox is set to that of the scrolled listbox. That caters for the scrollbar. Bearing in mind there are two versions of the `OnSelChange` handler, the first one simply does the same as the `OnScroll` handler, ensuring that both listboxes scroll to the same place.

The second one is a bit more adventurous and makes sure the selections in each listbox mimic each other, catering for both single-selection and multi-selection listboxes. The former one will be compiled by default. To see the second one, remove the `$define` compiler directive at the top of Listing 6 and re-run.

Acknowledgements

Thanks to Roy Nelson for the Delphi 2 minimisation fix and also to Steve Axtell for the Oracle connection information.


```
unit Slistbox;
interface
uses
  Messages, Classes, Controls, StdCtrls;
type
  TScrollEvent = procedure(
    Sender: TObject; ScrollCode, Pos: Word) of object;
  TSListbox = class(TListBox)
  private
    FOnScroll: TScrollEvent;
    FOnSelChange: TNotifyEvent;
  protected
    procedure WMVScroll(var Msg: TWMVScroll);
      message wm_VScroll;
    procedure CNCommand(var Msg: TWMCommand);
      message cn_Command;
  published
    property OnScroll: TScrollEvent
      read FOnScroll write FOnScroll;
    property OnSelChange: TNotifyEvent
      read FOnSelChange write FOnSelChange;
  end;

procedure Register;
implementation
procedure TSListbox.WMVScroll(var Msg: TWMVScroll);
begin
  inherited;
  if Assigned(FOnScroll) then
    FOnScroll(Self, Msg.ScrollCode, Msg.Pos);
end;
procedure TSListBox.CNCommand(var Msg: TWMCommand);
begin
  inherited;
  if (Msg.NotifyCode = lbn_SelChange) and
Assigned(FOnSelChange) then
    FOnSelChange(Self);
end;
procedure Register;
begin
  RegisterComponents('Samples', [TSListbox]);
end;
end.
```

➤ *Above: Listing 5*

➤ *Below: Listing 6*

```
{$define SIMPLE}
procedure TForm1.ListboxScroll(
  Sender: TObject; ScrollCode, Pos: Word);
begin
  if Sender = SListbox1 then
    SListbox2.TopIndex := SListbox1.TopIndex
  else
    SListbox1.TopIndex := SListbox2.TopIndex;
end;
{$ifdef SIMPLE}
procedure TForm1.ListboxSelChange(Sender: TObject);
begin
  ListboxScroll(Sender, 0, 0);
end;
{$else}
procedure TForm1.ListboxSelChange(Sender: TObject);
var
  Loop: Word;
  Src, Dest: TListbox;
begin
  { Set up Src and Dest as original and mimic }
  Src := Sender as TListbox;
  Dest := SListbox1;
  if Src = Dest then
    Dest := SListbox2;
  with Src do begin
    { Stop destination from flickering as we update it }
    Dest.Items.BeginUpdate;
    { Make multi-selections match in both listboxes }
    if Dest.MultiSelect then
      for Loop := 0 to Pred(Items.Count) do
        Dest.Selected[Loop] := Selected[Loop]
    else
      Dest.ItemIndex := ItemIndex;
    Dest.TopIndex := TopIndex;
    Dest.Items.EndUpdate;
  end;
end;
{$endif}
```