# Tips & Tricks

## TDbGrid Columns

There is an "unexpected feature" you may come across when using the `Columns` property of the Delphi 2 `TDbGrid` component. Set a column button style other than `cbsNone` and try to scroll horizontally, for example to the left, while in edit mode. If the column to the left has a width larger than the form's width, you won't see the correct field contents, but will continue to see the value of the column with the button.

To fix this behaviour, I suggest you derive a new component from `TDbGrid` to hide the editor before scrolling horizontally. This can be achieved by responding to the `WM_HSCROLL` message like this:

```
procedure TScrollDBGrid.WMHScroll(
  var Msg: TWMHScroll);
begin
  HideEditor;
  Inherited;
end;
```

Contributed by Roberto De Marini, email: rdemari@mbox.vol.it

## FreeObject Instead Of Free

The procedure in Listing 1 frees an object and, what's more to the point, set its pointer to nil. The parameter `q` is needed only for type-checking by the compiler. The procedure is called like this, for example:

```
FreeObject(MyObject,MyObject);
```

Delphi is smart enough to call the appropriate `Destroy` (which is called by `Free` implicitly).

Contributed by Reinhard Greeven, Frankfurt, Germany, CompuServe 100544,2773

➤ *Listing 1*

```
procedure FreeObject(var o;q:TObject);
var p : TObject absolute o;
begin
  { check if both parameters point to same instance }
  if p<>q then
    raise exception.Create(
      '"FreeObject":different params');
  { free }
  p.free;
  { set pointer to nil }
  p:=nil;
end;
```

## Property Read/Write Fields

It is a little known "secret" that you can use fields of records and even array elements as property read/write fields:

```
type
  TMyRecord
    Field1: longint;
    Field2: string;
  end;
  TMyClass = class
    private
      FRec : TMyRecord;
    public
      property Prop1: longint
        read FRec.Field1 write FRec.Field1;
      property Prop2: string
        read FRec.Field2 write FRec.Field2;
  end;
```

Contributed by Hallvard Vassbotn, email: hallvard@falcon.no

## Bug In Delphi 2 ComboBox

There is a bug in the write access methods of the `SelStart` and `SelLength` properties of the `TComboBox` component in Delphi 2. These properties are meant to set and return the start and length of the selection within the edit box of the ComboBox. No matter what you set these properties to, the selection will start in position 0. The reason for this erroneous behaviour seems to be a bug in the `TCustomComboBox.SetSelStart` and `TCustomComboBox.SetSelLength` methods in \DELPHI20\SOURCE\VCL\STDCTRLS.PAS, for those that have the VCL source. I will not repeat the code here, but they send a `CB_SETEDITSEL` message to the control with wrong parameters in `wParam` and `lParam`. According to my Win32 help file, the correct parameters are:

```
CB_SETEDITSEL (Win32)
wParam = 0;           /* not used, must be zero */
/* start and end pos */
lParam = MAKELPARAM((ichStart), (ichEnd);
An application sends a CB_SETEDITSEL message to
select characters in the edit control of a combo
box.
```

With this information, we can fix the problematic methods. You could include these corrections directly in STDCTRLS.PAS, but a cleaner approach is to inherit from the buggy `TComboBox` and do a static override of the `SelStart` and `SelLength` properties. This new combobox can then be installed into the component palette.

Listing 2 shows the new component and Listing 3 shows a form unit which tests the bug and the fix. Thanks to Harald Habberstad (haraldha@sn.no) for noticing this bug.

Contributed by Hallvard Vassbotn, email: hallvard@falcon.no

```
unit ComboFx;
interface
uses StdCtrls;
type
  TComboBoxFix = class(TComboBox)
  private
    function GetSelLength: Integer;
    function GetSelStart: Integer;
    procedure SetSelLength(Value: Integer);
    procedure SetSelStart(Value: Integer);
  public
    property SelLength: Integer
      read GetSelLength write SetSelLength;
    property SelStart: Integer
      read GetSelStart write SetSelStart;
  end;
procedure Register;

implementation
uses Messages, Windows, SysUtils, LibConst, Classes;
type
  TWordSelection = packed record
    StartPos: word;
    EndPos : word;
  end;

function TComboBoxFix.GetSelLength: Integer;
begin
  Result := inherited SelLength;
end;

function TComboBoxFix.GetSelStart: Integer;
begin
  Result := inherited SelStart;
end;

procedure TComboBoxFix.SetSelStart(Value: Integer);
var Selection: TWordSelection;
begin
  Selection.StartPos := Value;
  Selection.EndPos := Selection.StartPos + SelLength;
  SendMessage(Handle, CB_SETEDITSEL, 0,
    Longint(Selection));
end;

procedure TComboBoxFix.SetSelLength(Value: Integer);
var Selection: TWordSelection;
begin
  Selection.StartPos := SelStart;
  Selection.EndPos := Selection.StartPos + Value;
  SendMessage(Handle, CB_SETEDITSEL, 0,
    Longint(Selection));
end;

procedure Register;
begin
  RegisterComponents(LoadStr(srStandard),
    [TComboBoxFix]);
end;

end.
```

➤ *Listing 2*

## Delphi 2 For Loop Bug

Delphi 2.0 doesn't seem to be able to handle some `for` loops correctly. Look at the following code, for example:

```
var D, N: Integer;
begin
  D := -5;
  for N := 1 downto D do
    Writeln (N);
  Readln;
end;
```

What should happen is very simple: it does a `for` loop and displays all values of `N`. But it instead of counting from 1 down to -5, it counts from 1 to -3! If you single-step the loop, you can see it jump out of the loop for no apparent reason! If you run the same code on Delphi 1 it works fine. The bug can surface with the optimizer on and off.

I looked at the assembly generated from some simple variations. To explain what the compiler does wrong, its best to look at some correct code first. If we modify the original example to:

```
for N := 2 downto D do
```

the compiler generates (optimized version) the code shown in Listing 4.

The loop works by keeping an internal counter in `EBX` that starts out as a negative number and is then increased at the end of the loop. The loop starts over again as long as `EBX` does not reach zero. Note that `EBX` is adjusted to the starting index boundary (the `sub ebx, 00000002` in the listing) and is decreased once more (to include zero in the loop). This code works as expected and loops from 2 to -5.

Now to the buggy variant where the loop looks like:

```
for N := 2 downto D do
```

The assembly now looks like Listing 5.

```
unit TestCBF;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    { Simply drop these components on the form and
      keep all default properties }
    Button1: TButton;
    ComboBox1: TComboBox;
    Label1: TLabel;
    Button2: TButton;
    { Add click-event handlers for the buttons and a
      create handler for the form }
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
  public
  end;
var Form1: TForm1;

implementation
uses ComboFx;
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
{ Try the buggy TComboBox properties. This should
  select and copy '345' to the label. Instead the
  result is '012' }
begin
  ComboBox1.SelStart := 3;
  ComboBox1.SelLength := 3;
  Label1.Caption := ComboBox1.SelText;
end;

procedure TForm1.Button2Click(Sender: TObject);
{ Test the fixed combobox by doing a simple
  type-cast. This correctly results in '345' being
  selected and copied. }
begin
  TComboBoxFix(ComboBox1).SelStart := 3;
  TComboBoxFix(ComboBox1).SelLength := 3;
  Label1.Caption := ComboBox1.SelText;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Set the combobox up with some known text }
  ComboBox1.Text := '0123456789';
end;

end.
```

➤ *Listing 3*

```
bug_for.11:  D := -5;
:00405D62 B8FBFFFFFF     mov    eax,FFFFFFFB
bug_for.12:  for N := 2 downto D do
:00405D67 8BD8           mov    ebx,eax
; *** Notice this next line: ***
:00405D69 83EB02         sub    ebx,00000002
:00405D6C 7F20           jg     bug_for.14 (00405D8E)
; *** and this next one: ***
:00405D6E 4B             dec    ebx
:00405D6F BE02000000     mov    esi,00000002
bug_for.13:  Writeln (N);
:00405D74 8BD6           mov    edx,esi
:00405D76 B804724000     mov    eax,00407204
:00405D7B E81CCAFFFF     call   @WriteOLong
:00405D80 E892DDFFFF     call   @WriteLn
:00405D85 E872C8FFFF     call   @_IOTest
:00405D8A 4E             dec    esi
bug_for.12:  for N := 2 downto D do
:00405D8B 43             inc    ebx     ;<< Loop more?
; Then jump to start of loop
:00405D8C 75E6           jne    bug_for.13 (00405D74)
```

➤ *Listing 4*

Basically this is the same code, but it misses the `sub ebx, 00000001` and the `dec ebx` before the loop. This means that the loop will run two times fewer than intended. This code will run through 1 to -3 instead of from 1 to -5. If you change `D := -5` to some other negative number, it will still run two times too few.

Note that the same bug appears when optimization is turned off. The code is different but the underlying technique (and bug) is the same.

Contributed by Hallvard Vassbotn, email: hallvard@falcon.no

## Generating The Date

To achieve the same effect as Bob Swart's `DateGen` utility (in the October 1996 issue) I'm using the modification date of the application's EXE file to display the compilation date in my About box, as shownbelow. It's automatic and easy!

```
procedure TFmMain.mnAboutClick(Sender: TObject);
var AboutBox : TAboutBox;
    MyFileID : LongInt;
begin
  AboutBox := TAboutBox.Create(self);
  MyFileID :=
    FileOpen(Application.ExeName, fmOpenRead);
  FileDateToDateTime(FileGetDate(MyFileID));
  AboutBox.MyVersion.Text := DateTimeToStr(
    FileDateToDateTime(FileGetDate(MyFileID)));
  AboutBox.ShowModal;
  AboutBox.Free;
end;
```

Contributed by Thilo Bretschneider, email: thilo@Promo.DE

## Environment Variables

Those who have come to Delphi from a Borland Pascal background will probably lament the lack of a `GetEnvVar` function in Delphi (at least, I have!). Well, here comes the answer. I have written a function to do the same. The code is in Listing 6. It takes the name of the environment variable to obtain as its only parameter, eg:

```
bug_for.11:  D := -5;
:00405D62 B8FBFFFFFF     mov    eax,FFFFFFFB
bug_for.12:  for N := 1 downto D do
:00405D67 8BD8           mov    ebx,eax
:00405D69 85DB           test   ebx,ebx      * Bug here *
:00405D6B 7D1F           jnl    bug_for.14 (00405D8C)
:00405D6D BE01000000     mov    esi,00000001 * and here *
bug_for.13:  Writeln (N);
:00405D72 8BD6           mov    edx,esi
:00405D74 B804724000     mov    eax,00407204
:00405D79 E81ECAFFFF     call   @WriteOLong
:00405D7E E894DDFFFF     call   @WriteLn
:00405D83 E874C8FFFF     call   @_IOTest
:00405D88 4E             dec    esi
bug_for.12:  for N := 1 downto D do
:00405D89 43             inc    ebx
:00405D8A 75E6           jne    bug_for.13 (00405D72)
```

➤ *Listing 5*

```
Function GetEnvVar(Const Env:String) : String;
{ return contents of environment variable Env, or an empty
  string if variable does not exist, eg:
  Var Command : String;
  begin
     Command := GetEnvVar('COMSPEC')+#0;
     WinExec(@Command[1], sw_Normal);
  end; }
var
  p2 : pchar;
  i  : Word;
  s  : String;
Begin
  Result := '';
  p2 := GetDOSEnvironment;
  while p2[0] <> #0 do begin
    s := StrPas(p2);
    if (Pos(UpperCase(Env), UpperCase(s)) = 1) then begin
      i := Pos('=', s);
      If i=0 then i := pos(#32, s);
      Delete(s,1,i);
      Result := s;
      Exit;
    end;
    while (p2[0] <> #0) do Inc(p2); { goto end of current }
    Inc(p2); { point to next }
  end;
End {GetEnvVar};
```

➤ *Listing 6*

```
Function FileExistsOnPath(Const fName:String;
  Var aReturn:String):Boolean;
{ searches for filename FName in current directory or in
  directories in PATH. FName should NOT contain file path.
  Returns full pathname to file in aReturn, if found, eg:
  Var s1, s2:String;
  begin
     s1 := 'PROGMAN.EXE'
     If FileExistsOnPath(s1, s2) then {do something with s2}
     else MessageBox(0, 'Program Manager not Found',
       'Fatal Error', 0);
  end; }
Begin
  {search current directory}
  If FileExists(fName) then begin
    aReturn := ExpandFileName(fName);
    Result := True;
    Exit;
  end;
  {else search path}
  aReturn := FileSearch(fName, GetEnvVar('PATH'));
  Result := aReturn > '';
End {FileExistsOnPath};
```

➤ *Listing 7*

```
MyCommand := GetEnvVar('COMSPEC');
```

You can now use this for getting at all those juicy environment variables, or even for implementing a new function, say one that searches for a filename in all the directories in the `PATH`, see the example in Listing 7.

Contributed by The African Chief, email laa12@cc.keele.ac.uk

## Selecting The Library

A while back I suggested keeping multiple .DCL library files (or at least one test one) for all those components you're not sure you want to keep.

Listing 8 shows a speedier way of selecting a component library: a simple program to list all the .DCL files in the component library directory in a menu and allow selection of a specific component library before starting Delphi. This is an application programmer's attempt at low-level code and may not be the best example in the world...

Contributed by Phil Alexander, CompuServe 100121,1001

**Thanks for all your Tips, keep them coming in!**

**If you have any hints that you think will be of use to fellow Delphi developers, just drop them in an email to the Editor at 70630.717@compuserve.com**

➤ *Listing 8*

```
program Dstart;
{$R *.res}
uses
  Classes, WinTypes, WinProcs, Messages, SysUtils;
var
  SearchRec   : TSearchRec;
  PopupMenu   : HMenu;
  Msg         : TMSG;
  ReturnValue: integer;
  ret         : integer;
  cpos        : TPOINT;
  iptr        : PChar;
  buf         : array [0..255] of char;
  noch        : integer;
  flags       : word;
  i,x,y       : integer;
  hHookedWnd  : HWND;
  tmpstr      : string;
  startdir    : string;
  curDCL      : string;
  files       : TStringList;
  dStartClass: array [0..255] of char;
function NewMsgHandler(Window : HWnd; Message : Word;
  wParam : Word; lParam : LongInt) : LongInt; export;
begin
  case Message of
    wm_Command :
      begin
        case WParam of
          1..20: ReturnValue := WParam;
      end;
    end;
  end;
end;
function QuickWindow:HWND;
var
  wc : TWNDCLASS;
begin
  { Register the window class. }
  StrPCopy(dStartClass,'Delphi Start');
  wc.style := 0;
  wc.lpfnWndProc   := @NewMsgHandler;
  wc.cbClsExtra    := 0;
  wc.cbWndExtra    := 0;
  wc.hInstance     := hInstance;
  wc.hIcon         := 0;
  wc.hCursor       := LoadCursor(0,IDC_ARROW);
  wc.hbrBackground := 0;
  wc.lpszMenuName  := nil;
  wc.lpszClassName := dStartClass;
  if RegisterClass(wc) then
    QuickWindow := CreateWindow(dStartClass,
     'Delphi Start', WS_OVERLAPPED or WS_SYSMENU,
     CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, 0, 0,
     hInstance, nil );
end;
{ main program }
begin
  { record start dir, i.e. working directory of pm icon}
  getdir(0,StartDir);
  ReturnValue := 0;
  GetCursorPos(cpos);
  x := cpos.x;
  y := cpos.y;
  {create invisible window to receive messages}
  hHookedWnd := QuickWindow;
  i:=0;
  flags := 0;
  {load current component library setting,
   and get directory part of it by setting last \ to 0}
  ret := GetPrivateProfileString('Library',
    'ComponentLibrary', '', buf, 225, 'DELPHI.INI');
  iptr := StrRScan(buf,'\');
  iptr[0] := chr(0);
  tmpstr := StrPas(buf);
  iptr := iptr+1;
  curDCL := StrPas(iptr);
  files := TStringList.Create;
  PopupMenu := CreatePopupMenu;
  AppendMenu(PopupMenu,mf_disabled,0,buf);
  {Change directory to component library dir and list all
   files in the directory. For each file, add to menu
   and a string list - the latter for use later if a
   menu item is chosen}
  chdir(tmpstr);
  ret := FindFirst('*.DCL',faAnyFile , SearchRec);
  while ret = 0 do begin
    StrPCopy(buf,SearchRec.Name);
    files.add(SearchRec.Name);
    i:= i + 1;
    if SearchRec.Name = curDCL then
          flags := mf_checked
    else
          flags := 0;
    AppendMenu(PopupMenu,flags,i,buf);
    ret := FindNext(SearchRec);
  end;
  {..and display the menu}
  TrackPopupMenu(PopupMenu,0,x,y,0,hHookedWnd,nil);
  {Check for whether a menu selection was made}
  if PeekMessage(Msg,hHookedWnd,0,32767,PM_REMOVE) then
    DispatchMessage(Msg);
  {get rid of window created by QuickWindow}
  DestroyWindow(hHookedWnd);
  UnregisterClass(dStartClass,hInstance);
  {only process anything if return value > 0 i.e. a menu
   item has been chosen}
  if ReturnValue > 0 then begin
    getdir(0,tmpstr);
    tmpstr := tmpstr + '\' + files[ReturnValue - 1];
    StrPCopy(buf,tmpstr);
    WritePrivateProfileString('Library',
     'ComponentLibrary', buf, 'DELPHI.INI');
    getdir(0,tmpstr);
    tmpstr := tmpstr + '\DELPHI.EXE ';
    StrPCopy(buf,tmpstr);
    StrCat(buf, cmdLine);
    chdir(StartDir);
    WinExec(buf,cmdShow);
  end;
end.
```