Tips & Tricks

Unique Log Files

I enjoyed Brian Long's series on file handling and stream manipulation *[Issues 6 to 11, all available on the Collection '96 CD-ROM. Editor]*. Often applications need to create a log file and streams can be used for this purpose. The simple class `TLogFile` in Listing 1 can be used to write to a log file: it shows how to get a unique log file name and how to write new lines to this file. An example is included of how this class might be used.

Contributed by Tom Corcoran, tomc@unitime.com

Validating Bitmaps

Do you ever allow a user to load bitmaps at run-time? If so, the routine in Listing 2 should be useful for validation purposes.

Contributed by Tom Corcoran, tomc@unitime.com

Closed DOS Box

When you run a DOS program in Windows 95, its window remains open until closed by the user. To run a DOS program that closes its window after running, you must specify `command.com /c program` in the command line. Using the `WinExec` API function to run a program named progdos.exe, the call should be:

```
WinExec("command.com /c progdos.exe",
  sw_ShowNormal);
```

If you wish that the program is run hidden from the user, the second parameter must be `sw_Hide`. You must

➤ *Listing 2*

```
function ValidateBMP(bmpToChk: string; showMsg: boolean):
  boolean;
{ returns False if invalid BMP or if file doesn't exist }
var bmp: TBitMap;
  function LoadBmp(bmp: TBitmap; nameOfFile: string):
    boolean; { returns False if problem loading bmp }
  begin
    Result := True;
    try
      bmp.LoadFromFile(nameOfFile);
    except
      on EInvalidGraphic do begin
        if showMsg then
          MessageDlg(Format(
            'Bitmap: %s is an invalid image',
            [nameOfFile]), mtError, [mbOk], 0);
          Result := False;
      end;
    end;
  end;
begin
  if bmpToChk = '' then
    Result := False
  else begin
    Result := FileExists(bmpToChk);
    if Result then begin
      bmp := TBitMap.Create;
      try
        Result := LoadBmp(bmp, bmpToChk);
      finally
        bmp.Free;
      end;
    end;
  end;
end;
```

➤ *Listing 1*

```
unit LogStrm;
interface
uses SysUtils, Classes, Forms;
type
  TLogFile = class
  private
    logFileName: string;
    stream: TFileStream;
    function GetLogFileName: string;
  public
    constructor Create;
    destructor Destroy;
    procedure WriteToStream(strToWrite: string;
      blankLine: boolean);
  end;
implementation
constructor TLogFile.Create;
begin
  logFileName := GetLogFileName;
  stream := TFileStream.Create(logFileName, fmCreate);
end;
destructor TLogFile.Destroy;
begin
  stream.Free;
end;
procedure TLogFile.WriteToStream(strToWrite: string;
  blankLine: boolean);
var
  strNew: string;
  lengthStrNew: byte absolute strNew;
begin
  { #13 = carriage return #10 = form feed }
  strNew := Format('%s%s', [strToWrite, #13#10]);
  if blankLine then
    strNew := Format('%s%s', [strNew, #13#10]);
  stream.Write(strNew[1], lengthStrNew);
end;
function TLogFile.GetLogFileName: string;
const logFileName = 'log';
var
  i: integer;
  logFileNoExt: string;
begin
  logFileNoExt := Concat(ExtractFilePath(
    Application.ExeName), logFileName);
  i := 1;
  repeat
    Result := Format('%s%d.txt', [logFileNoExt, i]);
    Inc(i);
  until not FileExists(Result);
end;
end.

{ example of using TLogFile }
procedure WriteToLogFile;
const testDir = 'test';
var dirToMake: string;
begin
  with TLogFile.Create do
    try
      WriteToStream(Format('Log begun at %s',
        [DateTimeToStr(Now)]), True);
      dirToMake := Concat(ExtractFilePath(
        Application.ExeName), testDir);
      if not DirectoryExists(dirToMake) then begin
        MkDir(dirToMake);
        if IOResult = 0 then
          WriteToStream(Format('Made dir %s',
            [dirToMake]), False)
        else
          WriteToStream(Format('Unable to make dir %s',
            [dirToMake]), False);
      end else
        WriteToStream(Format('Directory %s already exists',
          [dirToMake]), False);
    finally
      Destroy;
    end;
end;
```

use the .com extension in command.com or the program will not run.

Contributed by Bruno Sonnino, Sao Paolo, Brazil, bsonnino@geocities.com (for more Tips from Bruno visit http://www.geocities.com/SiliconValley/8055)

Design Time Context Sensitive Help

For component developers, C++Builder introduced a new way of incorporating design-time context-sensitive help. With Delphi 3, Borland have done it again: yet another novel approach.

The vital details are buried deep in the Delphi 3 help system. The easiest way to find them is to do a help index search for *help files*, select the *Providing Help for your component* topic, then click the >> button on the help screen *twice*. Here you will learn about the all-new "A" footnote type and how to go about incorporating your enhanced help file into Delphi 3. But the following useful details were not included:

- The list of keywords for each topic don't have to be in the order shown. This is good, because help creation tools like Forehelp will usually sort them alphabetically.
- The CNT file is not mandatory: you don't need to supply one unless you want to.
- Above all, after installing your help file and editing Delphi3.CFG, a user installing your component must delete the (hidden) DELPHI3.GID file so that it will be recreated, including your help file's information, next time Delphi 3's help is used.

Any bets for how JBuilder will do it?

Contributed by Peter Hyde, peter@spis.co.nz

Working With Notebook Pages

Back in May 1996 (Issue 9) I wrote about how to convert strings to an enumerated type so that they could be used as (for instance) case statement selectors. One application of this that I've used repeatedly in my work is to simplify the code for working with TabSet and TabbedNotebook components. A similar technique should also work with the Win32 TabControl and PageControl components.

The trick is to create an enumerated type that matches the tabs/pages of the control in question (I just pulled some likely tab titles out of the air here, adding the nt prefix just makes them look associated and is not strictly necessary):

```
type
  TNotebookTabs = ( ntInit, ntSettings,
    ntGeneral, ntMiscellaneous);
```

Then, I use a little utility function that converts a notebook page index into the enumerated type (Listing 3). Now, in the OnChange or other event handlers you can do something like Listing 4.

The advantages of this technique became very apparent to me when I was working on a project wherein the main interface was a large tabbed notebook. On several occasions the client asked for the pages to be rearranged or new pages to be inserted. Until I started using this technique, each such change required some tedious and error prone re-structuring of the case statements to keep the correct processing associated with each page index. Using this technique, the processing is logically associated with the page title rather than a "magic number" index, so rearrangements or insertions have no effect on what's already there. The only requirement is that if a page is added or removed or a page name changes, you have to modify the enumerated type and the case references accordingly. I also think this approach makes the code eminently more readable with regard to what page is being operated upon.

Contributed by Stephen Posey, slposey@concentric.net

QuickReports Band Layout

The version of QuickReports that comes with Delphi 2 is minimally documented, but more than powerful enough for most reporting requirements. I learned this and the following tips during development of a report intensive application recently.

Inserting new bands into a report usually results in stacked up bands that are almost impossible to untangle. The trick is to set the Align property to None for each band below where you want to insert, and before inserting move each band down out of the way so there is enough space for the new band to drop in. You don't really need to go back and reset the Align of each QRBand to Top, since the component will process the bands in order anyway. In other words, if your form shows grey space between each QRBand, that space will not show up in your final report.

Contributed by Brandon Smith, synature@aol.com

➤ *Listing 3*

```
function Tform1.GetNotebookPage(Index : integer):
  TNotebookTabs;
var S: string;
begin
  S := 'nt' + TabbedNotebook1.Pages[Index];
  Result := TNotebookTabs(GetEnumValue(
    TypeInfo(TNotebookTabs), S));
end;
```

➤ *Listing 4*

```
procedure TForm1.TabbedNotebook1Change(Sender: TObject;
  NewTab: Integer; var AllowChange: Boolean);
var CurrentTab, NextTab: TNotebookTabs;
begin
  CurrentTab := GetNotebookPage(TabbedNotebook1.PageIndex);
  NextTab := GetNotebookPage(NewTab);
  case CurrentTab of
    ntInit          : { Init page specific processing }
    ntSettings      : { Settings page specific processing }
    ntGeneral       : { General page specific processing }
    ntMiscellaneous : { Miscellaneous... }
  end { case CurrentTab };
  case NextTab of
    ntInit          : { Init page specific processing }
    { I think you get the idea... }
  end { case NextTab };
end;
```

QuickReports Multiple Detail Fields

In one of my reports, I needed to have items from two detail tables listed side by side in the same band. In order to accomplish this I needed a TQRGroup and a TQRDetailLink. I put an rbSubDetail band on the report where the data would go and an rbGroupFooter band under it. Without going through the long trial and error process, the next steps were to set the PrintBand variable False in the rbSubDetail band's BeforePrint handler so it never showed. In one of the detail (ie linked to the master table) band's QRDBText component's BeforePrint handler, I gathered the data I wanted and put it into TQRMemo components I'd placed in the rbGroupFooter band. Here's a fragment from that gathering operation:

```
with dm.wttMeds, QRM_Dose.lines do begin
  clear;
  first;
  while not eof do begin
    add(fieldbyname('DOSE').asString);
    next;
  end;
end;
```

Finally, in the rbFooter band's BeforePrint handler, I sized and placed the QRMemos.

Contributed by Brandon Smith, synature@aol.com

QuickReports Sizing And Placement

In many reports, especially form type reports, it is difficult to determine ahead of time how much space to lay out for each data-aware field. For example, suppose we have first and last names as separate fields of 27 characters each. Do we want this line on our report:

```
Name:  John                        Smith
```

Or would we rather see:

```
Name:  John Smith
```

To obtain the latter, I use lines of code like the following to eliminate the extra spaces. Place this kind of code in the appropriate QRBand.BeforePrint event.

```
i := canvas.textwidth('xx');
QRT_Fname.left := QRL_Fname.left +
  QRL_Fname.width + i;
QRT_Lname.left := QRT_Fname.left +
  QRT_Fname.width + i;
```

Sizing a TQRmemo is also reasonably easy. Suppose I want to fill a TQMemo from a TStringList I've gotten from somewhere else. This code fragment adjusts the height of the QRM_reason so that all the lines in the StringList slreason are showing:

```
if slreason.count > 0 then begin
  QRM_reason.lines.assign(slreason);
  i := canvas.textheight('Z');
  QRM_reason.height := QRM_reason.lines.count * i;
end;
slReason.free;
```

But suppose we've allowed the user to do something radical, such as select his or her font of choice for printing the report.

Control over the location of items on a QuickReport depends on the QPrinter's Canvas. QPrinter is an object QuickReport instantiates as a result of a call to the Prepare, Preview or Print method. So, in order to find out how many pixels wide the Canvas is on which one is placing memos whose size is not known until run time, one has to either prepare or preview the report first. I use the following code to find out how much width I've got to play with:

```
with QuickReport1 do begin
  DM.SetToASingleRecord;
  showProgress := false;
  prepare;
  ActualImageWidth := pageWidth;
  QRPrinter.cleanup;
end;
```

The size of the font is also problematic and I'm not going to offer this as the best way to discover it, only that it appears to work. The handler and its nested procedure shown in Listing 5 also illustrate runtime placement and spacing of (TQMemo) elements in the report. PszFromPstrNew is a function that creates a pchar from a traditional Pascal string and MaxOf simply returns the larger of two numbers.

Contributed by Brandon Smith, synature@aol.com

➤ *Listing 5*

```
procedure TQR_DocVis.QRB_prnBeforePrint(
  var PrintBand: Boolean);
var lineheight, spacer : integer;
  procedure SetMemoSize(WhichMemo : TQRMemo;
    heading : string);
  var
    i, tmpwidth : integer;
    textSize : tSize;
    dc, thisfont: THandle;     {a dc handle}
    tmpPchar : pchar;
  begin
    with WhichMemo do begin
      dc := GetDC(handle);     {Get the Dc for the memo}
      thisFont := SelectObject(dc, Font.handle);
      height := (lineheight * lines.count) + 5;
      {make sure the column heading sets the minimum width
       we're going to use for this memo}
      tmpPchar := pszFromPstrNew(heading+medSpaceStr);
      GetTextExtentPoint32(dc, tmpPchar,
        length(heading+medSpaceStr), textSize);
      tmpWidth := textsize.cx;
      StrDispose(tmpPchar);
      for i := 0 to lines.count - 1 do begin
        tmpPchar := pszFromPstrNew(lines[i]+medSpaceStr);
        GetTextExtentPoint32(dc, tmpPchar,
          length(lines[i])+length(medSpaceStr), textSize);
        tmpwidth := maxof(tmpwidth, textsize.cx);
        strDispose(tmpPchar);
        end;
      width := tmpwidth;
      ReleaseDC(handle, dc); {Release the Dc}
    end;
  end;
begin
  lineheight := QRPrinter.canvas.textheight('X');
  {medSpaceStr is set by user and saved in ini file}
  spacer := QRPrinter.canvas.textwidth(medSpaceStr);
  { set size of the memo boxes, in this case there were 5 }
  SetMemoSize(QRM_med,QRL_med.caption);
  { make sure left to right orientation is OK, no overlap}
  QRM_Dose.left := QRM_med.left + QRM_med.width;
  ... add other adjustments as needed ...
end;
```