Tips & Tricks

TLists Revisited

At the end of Jonathon Morgan's article *Turbocharging Database Applications With TLists* in the October 1996 issue, he recomends using polymorphism to free a set of `TObjects` in a `TList`.

I was using a similar method to free a set of `TBitmaps` assigned to `TStrings` (the `Items` of a listbox), under Delphi 1. I found that casting them to `TObjects` allowed me to free them, but that not all the memory was being de-allocated. Casting to the correct type (`TBitmap`) solved the problem. It seemed that `TObject` was freeing the objects as `TObjects`, rather than calling their descendant methods to free all the required memory.

Contributed by Mark Williams
Mark@polyhdrn.demon.co.uk

Showing Modal Forms

Here's another addition to Brian Long's addition (October issue) to Richard Smith's excellent Tip in the August issue. Using the code shown in Listing 1, if the form takes a while to start up (due to a slow machine) at least users are pacified! Make sure `Controls` is included added to the uses clause.

Contributed by Glenn Shukster, Thornhill, Ontario, Canada (CompuServe 72734,123)

Rich Text On The Clipboard

Cut and Paste is my favorite technique in the Windows world (under DOS I hate the fact that I can't share information between different applications). With Windows 95 and Delphi 2.0 we have a new format for exchanging text: Rich Text Format (RTF).

RTF has the advantage of being able to include formatting styles in your text. You can make a total sum bold, or you can underline specific text for example. Now you can output RTF text to a file and load it into another application. You may think we can use the Clipboard for that, but the `TClipboard` class which comes with Delphi loses the formatting and we just get plain text.

Each Clipboard format supported by Windows has a unique ID. You can browse the currently active format in the clipboard with the code shown in Listing 2. I used this to spy out the name under which the Rich Text Format is registered. Some of the Clipboard format IDs are defined in the `Windows` unit, for example `CF_TEXT` and `CF_BITMAP`, but *not* `CF_RTF`. However, we can define it ourselves as follows:

```
var CF_RTF : Word;
...
CF_RTF :=
  RegisterClipboardFormat('Rich Text Format');
```

The text `Rich Text Format` is very important.

So, now we can develop a descendant of Delphi's `TClipboard` class to support RTF. I called it `TxRTFClipboard`, because it resides in our xTools library. The `TxRTFClipboard` unit source code is included on this month's disk for your free use. This new class can be accessed by calling `RTFClipboard`. It has a new property of type `string` called `AsRTF`. The RTF format is no more than a very large string with a bunch of formating codes enclosed in {} brackets, so it's very easy to handle with Delphi 2.0 now we have 2Gb strings!

If the RTF format is supported by the Clipboard the following call shows the contents of the clipboard:

```
ShowMessage(RTFClipboard.AsRTF);
```

In the same manner we can put something onto the clipboard. For example the following code puts a formatted address onto the clipboard:

```
RTFClipBoard.AsRTF :=
  '{\rtf1\ansi\deff0\deftab720' +'{\fonttbl'
  + '{\f0\fmodern Times;}}' +'{\colortbl'
  + ColorToRtf(clBlack) +ColorToRtf(clWhite)
  + '}' + '\deflang1031\pard\plain\f0\fs20'
  +'To\par ' +'Fabula Software\par '
  +'Stefan Bother\par ' +'Methfesselstr. 38\par '
  +'{\b 20257 Hamburg\par '
  + 'Germany\par }' + '}';
```

➤ *Listing 1*

```
uses {...} Controls;
...
Procedure CreateModal(FormClass: TFormClass);
begin
  Screen.Cursor :=  crHourglass      ;
  with FormClass.Create(Application) Do
    Try
      Screen.Cursor :=  crDefault ;
      ShowModal
    Finally
      Free
    end
end;
```

➤ *Listing 2*

```
procedure TMainForm.Button3Click(Sender: TObject);
var
  Format : word;
  Tmp: array[0..100] of char;
begin
  Clipboard.Open;
  Format := EnumClipboardFormats(0);
  while Format <> 0 do begin
    GetClipboardFormatName(Format,Tmp,Sizeof(Tmp)-1);
    ShowMessage(IntToHex(Format,8)+':'+String(Tmp));
    Format := EnumClipboardFormats(Format);
  end;
  Clipboard.Close;
end;
```

This address is now ready for pasting into your word processor with a bold zipcode, city and country. Don't forget to also support the same text in plain CF_TEXT format in case the receiving application cannot take RTF. In the above example you would assign it as follows:

```
RTFClipboard.AsText :=
  'To'+#13#10+'Fabula Software'+#13#10+
  'Stefan Bother'+#13#10+
  'Methfesselstr. 38'+#13#10+
  '20257 Hamburg'+#13#10+'Germany';
```

When using RTF Clipboard support with Delphi 2.0's RTF Edit control (TRichEdit) I found the documentation lacking. It's not very clear how to assign and get RTF text from and to the control. So I wrote two simple methods to do it. The trick is to use the TRichEdit.Lines.LoadFromStream and ...SaveToStream calls with a memory stream and then use a typecast to the memory buffer. The following example assumes that you have a RichEdit1 control on your form. From String to RichEdit control use:

```
strToRtf(MyRTFString, RichEdit1.Lines);
```

and from RichEdit control to String use:

```
ShowMessage(RtfToStr(RichEdit1.Lines));
```

So I now hope you will write a lot of RTF enabled applications. At Fabula we used our experience in RTF to write tools for creating Help and HTML files with the RichEdit control. The formatting is also a great plus for database applications.

Contributed by Stefan Boether, who works for Fabula Software in Hamburg, Germany and can be reached by email at stefc@fabula.com

Which Event?

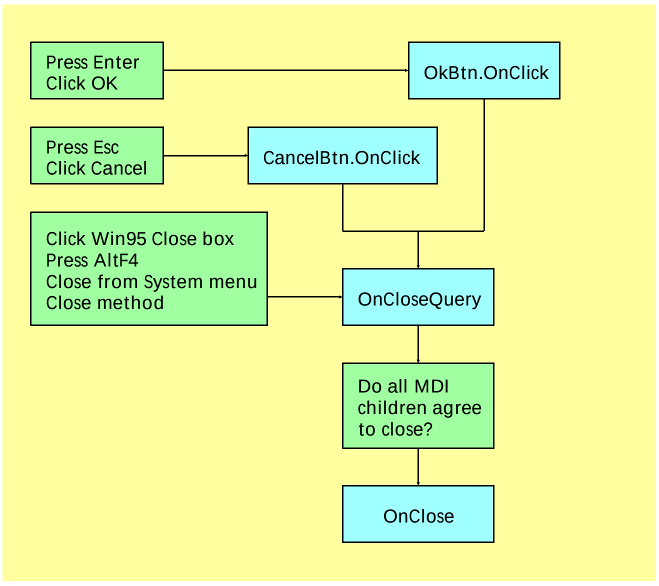
When Delphi closes a form, it generates many related events. Which should we use? The flow of events is described in the diagram shown in Figure 1.

One difference between clicking OK and pressing Enter is that a mouse click moves the focus to the button but pressing the Enter key doesn't. For some controls which only update their value when the focus changes, pressing Enter may cause some problem. I solve this by putting:

```
OkBtn.SetFocus;
```

in the first line of the OK button handling code to make sure the focus is always moved.

I suggest you avoid putting form close handling code in OK or Cancel's OnClick event, because they are not executed if the form is closed from the System menu. Instead, I put both OK and Cancel processing code in OnCloseQuery. Then you can use the ModalResult property to distinguish between them. Note that for non-MDI forms, OnCloseQuery and OnClose can be used interchangeably.



➤ *Figure 1*

To stop a form from closing, we may use the following methods:

- From OnClick, set ModalResult to mrNone (default is mrOk/mrCancel);
- From OnCloseQuery, set CanClose to False (default is True) or raise an exception;
- From OnClose, set Action to caNone (default is caHide) or raise an exception.

Surprisingly, raising and exception inside of OK or Cancel's OnClick event doesn't stop the form from closing. I consider this a Delphi bug.

Contributed by Steve Tung, email: stung@po.pacific.net.sg

Text File Device Drivers In 32-Bit Land

Programming in the 32-bit world may not be Wonderland, but it is certainly different to 16-bit programming. My text file device driver (TFDD) for string parsing (see the article in Issue 12, August 1996), written before Delphi 2 existed, has served me well in Delphi 1, but anyone who tried to use it in Delphi 2 may have noticed two things: it won't compile and even if you do find a way to compile it, it will not work.

Several things must be changed in order to use the sdd unit in Delphi 2. To compile it as a 32-bit unit, you must eliminate the 16 bit assembly code. In this version I just eliminated the delim procedure, since I decided to handle converting the delimiter character another way. To avoid a typecasting problem in compilation, we must recognize that Borland changed the size of the UserData field to 32 bytes. Listing 3 shows the new sdd unit, including the new declaration of the usr type. To keep the same kind of string as before I used the {$H-} directive. I have never needed even the full 255 characters available in the shortString for strings that I parse in this way, so I saw no need to change to the longer ansiString.

Making the unit compilable is a worthy goal, but we still need to make it *work!* In Delphi 2, the `InStr` function was never called. It turned out that some values which I could take for granted before now have to be initialized. To `AssignSt` I added:

```
BufPos := 0;
BufEnd := 0;
```

The final change I made was not required, but I consider it an enhancement. At first I replaced `delim` with ordinary Delphi code, but I decided to move the conversion of delimiters to #13 into `InStr`. Rather than make changes to the source string that later must be reversed, I make the changes in the temporary buffer. For this purpose I embarked on my first adventure with string manipulation assembly code in 32-bit mode.

For output with the string device driver, I do not use `WriteLn` at all, since that puts a carriage return *and* a line feed into the string. If I use `Write`, I explicitly write the delimiter:

```
write(t,data,delimiter);
```

Thanks to my co-worker Brent Sherwood for assuring me that `scasb` will work in 32-bit mode and to reader Rene Holst who encouraged me to make this work for Delphi 2.

Contributed by Jon Jacobs, Mastercomp,
email: mstrcomp@gte.net

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 on 70630.717@compuserve.com

➤ *Listing 3*

```
unit sdd;
{string device driver to convert numbers and parse strings}
{$H-}
interface
uses SysUtils;
procedure AssignSt(var t:textFile;var s:string;dlm:char);
implementation
type
  PString=^string;
  usr=record
    ps : PString;
    dlim : char;
    ud : array[6..32] of byte;
  end;
function InStr(var t:textFile):integer; far;
var
  p : pointer;
  e : cardinal;
  dlm : char;
begin
  InStr := 0; {for ioResult}
  with TTextRec(t),usr(UserData) do begin
    if (BufPos<BufEnd) and (Handle<>0) then exit;
    BufPos := 0;
    BufEnd := length(ps^)-Handle;
    if BufEndBufSize then BufEnd := BufSize;
    move(ps^[succ(Handle)],BufPtr^,BufEnd);
    inc(Handle,BufEnd);
    p := BufPtr;
    e := BufEnd;
    dlm := dlim;
    asm
      push edi       {save Delphi's string pointer}
      mov edi,p      {point to buffer}
      mov al,dlm
      mov ah,13
      mov ecx,e   {keep track of how much buffer left to search}
      cld            {go forward}
      @@1:
      repnz scasb    {search for the delimeter in al}
      jnz @@2        {if not found this time must be at end}
      mov [edi-1],ah {found, so make the substitution }
      jcxz @@2       {if counter 0 then done}
      jmp @@1        {if not at end, look for another}
      @@2:
      pop edi        {restore Delphi's string pointer}
    end;
  end;
end; {InStr}
function OutStr(var t:textFile):integer; far;
var i : integer;
begin
  with TTextRec(t),usr(UserData) do begin
    for i := BufEnd to BufPos-1 do
      ps^ := ps^+PTextBuf(BufPtr)^[i];
    Handle := length(ps^);
    BufEnd := BufPos;
  end; {with}
  OutStr := 0; {for ioResult}
end; {OutStr}
function FlushStr(var t:textFile):integer; far;
begin
  FlushStr := 0; {for ioResult}
end; {FlushStr}
function CloseStr(var t:textFile):integer; far;
begin
  with TTextRec(t) do begin
    Mode := fmClosed;
    Handle := 0;
    BufEnd := 0;
  end;
  CloseStr := 0; {for ioResult}
end; {CloseStr}
function OpenStr(var t:textFile):integer; far;
begin
  with TTextRec(t),usr(UserData) do begin
    CloseFunc := @CloseStr;
    case Mode of
      fmInOut : begin
        Mode := fmOutput;
        InOutFunc := @OutStr;
        FlushFunc := @OutStr;
        Handle := length(ps^);
      end;
      fmInput : begin
        InOutFunc := @InStr;
        FlushFunc := @FlushStr;
      end;
      fmOutput : begin
        InOutFunc := @OutStr;
        FlushFunc := @OutStr;
        ps^ := '';
      end;
    end; {case}
  end; {with}
  OpenStr := 0; {for ioResult}
end; {OpenStr}
procedure AssignSt(var t:textFile;var s:string;dlm:char);
begin
  with TTextRec(t),usr(UserData) do begin
    Mode := fmClosed;
    BufSize := SizeOf(buffer);
    BufPtr := @buffer;
    OpenFunc := @OpenStr;
    BufPos := 0;
    BufEnd := 0;
    Name[0]:=#0;
    dlim := dlm;
    ps := @s;
    Handle := 0;
  end; {with}
end; {AssignStr}
end.
```