Tips & Tricks

Using Strings As Case Selectors

The February issue's *Tips & Tricks* column included Michael Ax's description of using the GetEnumName function from Delphi's run time type information (RTTI) services to retrieve string equivalents of enumeration type constants. There is also an inverse function called GetEnumValue which turns a string back into an enumeration constant. This function, and a bit of ingenuity, can effectively give Delphi an ability that it's not generally thought to have, namely using strings as case selectors.

I recently had cause to create a simple macro language to automate certain text processing tasks. I originally wrote the engine in the ScriptMaker language that comes as part of Norton Desktop v3.0, mostly because it provides SendKeystrokesToWindow and FindWindowByPartialTitle routines that make manipulating other windows relatively simple. The main routine of my macro processor is a big "switch" statement that does different things depending on which macro command is being processed and is selected by the name of the command.

When I recently came up with Delphi equivalents to send keystrokes and find windows, I started planning to convert the macro processor to Delphi, but quickly baulked at the prospect of all the nested if statements to process the commands, because Delphi wouldn't allow strings as case selectors. But, why not define an enumerated type that paralleled the commands in the macro language, and use GetEnumValue to turn the strings retrieved from the macro file into enumerated constants of that type, which could then be used as part of a case statement! Listing 1 shows the idea.

Note that adding the tmc prefix to the enumerated constants isn't strictly necessary, but for this example it does prevent naming conflicts with Pascal reserved words. It also ties them together as constants of a particular type, and serves as a reminder of their purpose.

I'm sure that there are any number of other ways that this facility could be put to use, such as case selecting on the contents of a listbox or a set of radio buttons or even menu items. Any situation where you have string values that should remain more or less constant they can easily be used as case selectors via this method, saving you the hassle of writing and maintaining your own conversion function.

Contributed by Stephen Posey, University of New Orleans, USA, email: SLP@uno.edu

```
type
  TMacroCommand = (tmcIF,  tmcGOTO, tmcCALL, tmcEND);
var
  S : string;
begin
  GetNextCommand(S); { retrieve next command into S }
  case TMacroComand(GetEnumValue(
    TypeInfo(TMacroCommand), 'tmc'+S)) of
    tmcIF    : { do If stuff }
    tmcGOTO  : { do Goto Stuff }
    tmcCALL  : {etc...}
  end { case } ;
  ...
end;
```

➤ *Listing 1*

```
Uses Printers;
function SetNumCopies(NumCopies : integer) : boolean;
var
  ADevice, ADriver, APort : array[0..255] of char;
  DevMode : THandle;
  PtrDevMode : PDevMode;
begin
  Printer.PrinterIndex := Printer.PrinterIndex;
  Printer.GetPrinter(
    ADevice, ADriver, APort, DevMode);
  if DevMode <> 0 then begin
    PtrDevMode := GlobalLock(DevMode);
    with PtrDevMode^ do begin
      dmFields := dmFields or DM_COPIES;
      dmCopies := NumCopies;
    end;
    GlobalUnlock(DevMode);
    Printer.PrinterIndex := Printer.PrinterIndex;
    SetNumCopies := True;
  end else
    SetNumCopies := False;
end;
```

➤ *Listing 2*

Changing Printer Parameters

Recently I needed to print out large numbers of the same label onto sheet laser labels. In order to reduce download time to print manager, conserve disk space and program professionally I thought that I would simply print one page and let the printer driver's number of copies feature do the rest.

I sought help from a couple of friends on the CompuServe Delphi forum, Listing 2 shows what we came up with. This basic function can be changed to alter many of the printer driver's parameters including paper orientation, custom paper size, input bin and print quality.

Looking up GetPrinter in the help will reveal absolutely nothing about the DevMode parameter except that it is of type THandle. It turns out that it is a handle to a piece of global memory storing a TDevMode record structure recently obtained from the printer driver. You must set PrinterIndex before calling GetPrinter otherwise this handle will be invalid. Once you have a valid handle you can retrieve a pointer to it by locking it in memory using the GlobalLock API call. You make your changes to the record structure and unlock it. Resetting PrinterIndex to itself again causes the TDevMode record structure to be sent back to the printer driver for evaluation. The dmFields item in the record structure indicates which items you want changing. The constants can be found by looking up TDevMode in the

```
uses
  DBIProcs, DBITypes, DBIErrs;
function PackTable(
  tbl:TTable; db:TDatabase):DBIResult;
var crtd: CRTblDesc;
begin
  Result := DBIERR_NA;
  with tbl do
    if Active then
      Active := False;
  with db do
    if not Connected then
      Connected := True;
  FillChar(crtd,SizeOf(CRTblDesc),0);
  StrPCopy(crtd.szTblName,tbl.TableName);
  crtd.bPack := True;
  Result := DbiDoRestructure(
    db.Handle,1,@crtd,nil,nil,nil,FALSE);
end;
{Example of use:}
procedure TForm1.Button1Click(Sender: TObject);
begin
  if PackTable(Table1,DataBase1) = DBIERR_NONE then
    ...
  else
    MessageBeep(0);
end;
```

➤ *Listing 3*

API help. *[If this has whetted your appetite for information on printing with Delphi, we have good news: Xavier Pacheco, of "Delphi Developer's Guide" fame, is at this moment preparing a series of articles on printing for your favourite magazine (aren't you, Xavier...). Editor]*

Contributed by William Thorp, CompuServe 100025,3500

Moving Components To Delphi 2

When recompiling your components developed with Delphi 1 using Delphi 2, you may receive an error message *'Unsupported 16 bit resource in file ...'*. Don't panic, the most likely cause is that your component has a .DCR file for the palette bitmap and you forgot to convert it to 32-bit format. The simplest way to do this is using Bob Swart's RESCONV.EXE program included on this month's disk in directory CONSTRUC.

Contributed by Paul Warren, hg_soft@uniserve.com

Repacking Paradox Tables

If your application uses Paradox tables it can be very useful to include an option to allow users to repack the tables. The function shown in Listing 3 will do just this. It requires a TDatabase component pointing to the same directory.

Contributed by Mike Orriss, CompuServe 100570,121

Replacing Controls

If you've ever had to replace one type of control on a form with another, it can be a real pain re-setting all the properties of the new control to match the one it replaces. Here's the easiest way to not lose the values of properties that are common to both controls:

1) Cut the control to the clipboard,
2) Paste back to the PAS form after the final end. (it will show as its text form),
3) Modify the text as required,
4) Copy text to the clipboard,
5) Paste to the form as usual,
6) Iterate through 3-5 until it pastes without error!
7) Remove code after final end. to prevent a compiler warning in Delphi 2.

Contributed by Mike Orriss, CompuServe 100570,121

Dead MaxLength Of DBedit

The MaxLength property of the DBEdit control doesn't work as advertised: it's supposed to control the maximum number of characters but doesn't. You can work around the problem via an OnKeyPress routine:

```
procedure TForm1.DBEdit1KeyPress(
  Sender: TObject; var Key: Char);
const maxl = 4;
begin
  if Key <> #8 then  {allow backspace}
    with Sender as TDBEdit do
      if Length(Text) = maxl then
        Key := #0;
end;
```

Contributed by Mike Orriss, CompuServe 100570,121

Lost Cursor

If you lose the cursor after displaying dialogs such as ShowMessage in an OnExit event, the problem is that you are issuing commands that change focus inside an event handler that sets focus, which can cause recursion. You can use a work-around to handle this by calling a user-defined message to do all OnExit processing, controlled by a boolean flag that prevents recursion. The code required is shown in Listing 4.

Contributed by Mike Orriss, CompuServe 100570,121

➤ *Listing 4*

```
interface
const
  WM_MyExitRtn = WM_USER + 1001;
type
  TForm1 = class(TForm)
  private
   {prevent message recursion}
   bExitInProgress: Boolean;
  public
   Procedure WMMyExitRtn(Var msg:TMessage);
     message WM_MyExitRtn;
  end;
implementation
procedure TForm1.DBEdit1Exit(Sender: TObject);
begin
  If not bExitInProgress then
    PostMessage(Handle,WM_MyExitRtn,
      0,LongInt(Sender));
end;
procedure TForm1.WMMyExitRtn(var msg:TMessage);
begin
  bExitInProgress := True;  { prevent recursive call }
  {user exit code }
  bExitInProgress := False; { clear the flag }
end;
```