The Delphi CLINIC

Edited by Brian Long

Problems with your Delphi project?
Just email Brian Long, our Delphi Clinic Editor, on clinic@blong.com or write/fax us at The Delphi Magazine

GUID Generation

Q I am manufacturing a new interface by hand and need to manufacture a `GUID` (**G**lobally **U**nique **ID**entifier). How do I do this?

A General advice says that you call `CoCreateGuid` from the `ActiveX` unit. But that is tedious, as you then need to translate it into a string using `GUIDToString` from the `ComObj` unit:

```
uses ActiveX, ComObj;
...
var
  G: TGUID;
...
OleCheck(CoCreateGuid(G));
ShowMessage(GuidToString(G));
```

It is easier, whilst in the IDE code editor, to use `Ctrl-Shift-G` which does these two steps for you and then inserts the `GUID` into the editor. This is actually documented, contrary to popular opinion. `Choose Help | Keyword Search`, type "shortcuts," press `Enter` and choose `Editor` (default). It appears somewhere in this list.

ReportSmith And The TReport Mystery

Q I just purchased Delphi 3 Professional. I can see that the `TReport` component (the `ReportSmith` wrapper) has been moved out of the VCL, its source file is no longer with the rest of the VCL source and its DCU file is in the Delphi2 subdirectory off Delphi 3's LIB directory.

I added REPORT.DCU to the default user package (ie DCLUSR30.DPK). When it compiled, I was informed that 3 other packages were required and that the `REPORT` unit would be removed. So I thought maybe a component registration routine was required and it probably did not exist in REPORT.DCU.

I proceeded to create a bogus unit with just a `Register` procedure that called `RegisterComponents('Data Access', [TReport])`. In my `uses` clause, I included `Classes` and `Report`. I recompiled the package and it said it was installed successfully, but I still cannot find `TReport` on the palette.

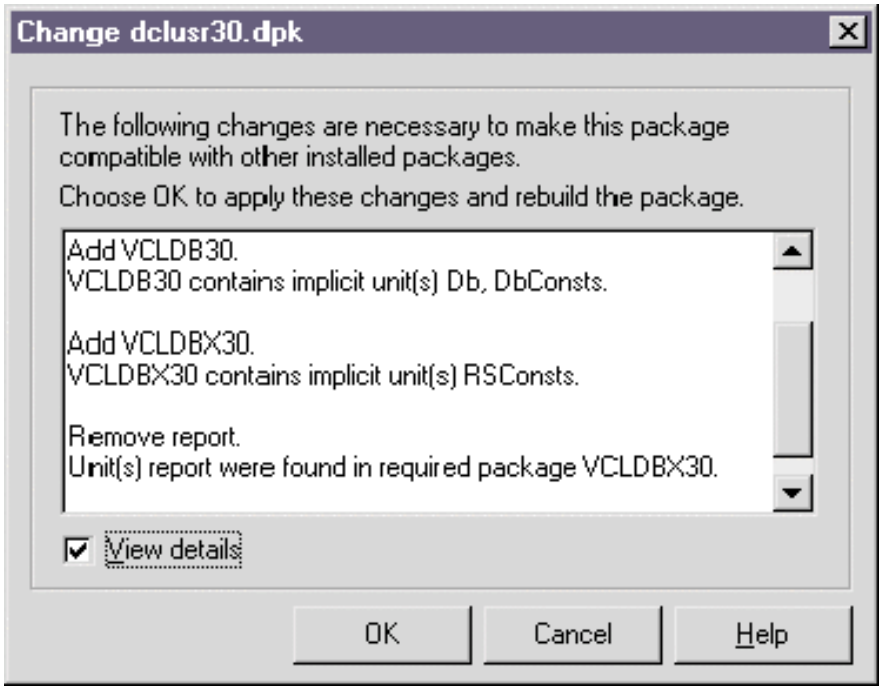
It appears that Borland is directing developers towards QuickReport but I still need to use ReportSmith for now. How do I get `TReport` back on the component palette?

A When you compile a package and it has problems, as your first attempt did, the dialog that tells you about them tries to be as helpful as it can. There is a checkbox to get additional information on the problems (Figure 1). Notice that it tells you two important pieces of information. Firstly that the `REPORT` unit is already present in the VCLDBX30.DPL package. It is also saying that to get your package compatible with the other currently loaded packages, you require VCLDBX30 since that includes the `RSConsts` unit which in turn is used by the `REPORT` unit.

This is a *key* point about packages: if a unit is included in one package that gets used by an application, it *may not* appear in any other package used by the same application. VCLDBX30 is already loaded by the Delphi IDE.

So VCLDBX30 has the code present and, as it turns out, the `TReport` component is already registered by the DCLDB30.DPL design-time package. You can see this if you select `Delphi Database Components`

➤ *Figure 1*



(package DCLDB30.DPL) in Component | Install Packages... and press Components. The reason you cannot find the component on the palette is that it has been configured to be hidden.

So, go to Component | Configure Palette..., select [All] in the Pages: list, locate TReport (probably at the end of the list) and press Show. Hey presto.

Static Class Data

Q In C++, in order to distinguish between object methods and fields and class methods and fields, the keyword static is used, see Listing 1. Is there an equivalent in Delphi?

A There is an equivalent for methods: class methods, but not for data fields. However, you can get much the same effect for data fields by using two class methods and a global typed constant.

Before worrying about the details too much let's take a look at class methods. The idea with normal methods is that they can only be called via some object reference. When the method is executing, a hidden parameter called Self points back to the object reference. Class methods can also be called via an object reference, but they can also be called via a class reference An example of a class method that all classes possess is ClassName. You can call ClassName as a method of any object, but you can also say, for example, TButton.ClassName or HintWindowClass.ClassName. In all cases, the class name is returned in the form of a string.

When a class method is executing, Self is still available, but this time it points back to the class reference. To make a class method simply involves the use of the word class at the start of the declaration and the implementation as we will see (check Listing 2).

Now let's get back to the class data field problem. In the implementation part of the unit that implements the class, declare a typed constant of the appropriate type, initialised to a sensible starting value. Then implement a class procedure method that can be used to write a value to the typed constant, and a class function method to retrieve the value. Since there is only one typed constant, it acts rather like a static field in that instances of the class can all access it. Moreover, since the accessor methods are class methods, you do not need an object through which to access the value; it's about the closest you will get as far as I can see.

Listing 2 shows a class that has a (fake) class data field and a class method. The project STATIC.DPR has code that accesses all these class parts, as shown in Listing 3.

```
class TStatic: public TObject
{
  __public:
  int AnObjectField;        // an object attribute int
  static int AClassField;   // a class attribute int
   static void AClassMethod(void);
  void AnObjectMethod(void);
}
```

➤ *Listing 1*

```
type
  TStatic = class(TObject)
  public
    AnObjectField: Integer;
    class function GetAClassField: Integer;
    class procedure SetAClassField(Value: Integer);
    procedure AnObjectMethod;
    class procedure AClassMethod;
  end;
implementation
const StaticField: Integer = 0;
class function TStatic.GetAClassField: Integer;
begin
  Result := StaticField
end;
class procedure TStatic.SetAClassField(Value: Integer);
begin
  StaticField := Value
end;
class procedure TStatic.AClassMethod;
begin
  ShowMessage('This is a class method')
end;
procedure TStatic.AnObjectMethod;
begin
  ShowMessage('This is an object method')
end;
```

➤ *Listing 2*

```
Static.AnObjectMethod
...
Static.AClassMethod
...
TStatic.AClassMethod
...
ShowMessage('Object field was ' + IntToStr(Static.AnObjectField));
Inc(Static.AnObjectField, 20);
ShowMessage('Object field now is ' + IntToStr(Static.AnObjectField));
...
ShowMessage('Class field was ' + IntToStr(Static.GetAClassField));
Static.SetAClassField(Static.GetAClassField + 20);
ShowMessage('Class field now is ' + IntToStr(Static.GetAClassField));
...
ShowMessage('Class field was ' + IntToStr(TStatic.GetAClassField));
TStatic.SetAClassField(TStatic.GetAClassField + 20);
ShowMessage('Class field now is ' + IntToStr(TStatic.GetAClassField));
```

➤ *Listing 3*

WideStrings

Q I have read that Delphi 3 supports WideStrings in an article in your magazine. The question follows for full Unicode support within controls like TEdits and TLabels. Is it possible to assign a String or WideString to Caption or Text properties?

A Well, yes you can assign a WideString to them, but these properties are still defined

as `String (AnsiString)` and an implicit conversion will take place, potentially losing information. The VCL is still `String`-based, mainly due to Windows 95's lack of Unicode support.

However, there is quite a useful section in the Delphi 3 help file that covers writing international applications, and it goes to great lengths to describe Unicode (two byte characters) and how it differs from DBCSs (double byte character sets) which can be dealt with rather well using the new RTL routines that were referred to in passing in part 1 of the Delphi 3 preview article. So international applications can be done, but not very well with Unicode.

This part of the help file can be found by choosing `Help | Keyword Search` and looking for "international applications." From this topic, the browse sequence (that is, the pages that can be reached using the >> button) will go through many useful topics on the subject. Included five pages into this list is the one titled "Enabling application code" that has sections called "Character sets," "OEM and ANSI character sets," "Double byte characters sets" and "Wide characters."

I am not entirely comfortable with all the issues involved in writing a localised application and would certainly be happy to see an article on the subject in a future issue of *The Delphi Magazine [Any takers? Editor]*.

Win32s?

Q How can one create applications using Delphi 3 to run under the Win32s subsystem of Windows 3.x?

A Sorry. This is a non-starter. Delphi 2 and 3 do stuff that isn't Win32s compatible. In short they only support Win32 (as, in fact, do Microsoft these days).

Command-Line Compiling

Q Can you tell me where I can find some information about the command line interface to the Delphi compiler and if it will use the project options file.

We are going to be building several delphi applications, as a set, and would like to automate the process.

A See Appendix A of the *Object Pascal Language Guide*. Note that the original release of Delphi 1 didn't supply this manual, you had to buy it separately, or download an online version from CompuServe or the Internet.

The command-line compiler does not use the project options file; you will need to edit the command-line compiler's DCC.CFG to specify the options as dictated by the syntax of that file (which has a .OPT extension with Delphi 1 and a .DOF file with versions 2 and 3).

Also see the *Run-Time Location Information* article by Vitaly Miryanov in Issue 22 for a tool that translates a project options file into a command-line compiler configuration file.

Accessing Components By Name

Q I am dynamically creating a random number of `TEdits` on a form. As I create them I am also naming them along the lines of `edtMyEdit1, edtMyEdit2...edtMyEditn`. How can I reference the properties of these `TEdits` when all I know is their name?

A When a component is created, it is common to give it an owner through the constructor parameter. This causes a reference to your component to be added into the owner's `Components` array and its `ComponentCount` property to be incremented. The owner has a method called `FindComponent` which, given a component name, will iterate through its `Components` array, looking for the component with a `Name` property that matches. It returns this as a `TComponent` so you will need to use typecasting to be able to access all the `TEdit` properties.

Check the use of `FindComponent` here in Listing 4, it should give you the general idea.

➤ *Listing 4*

```
var
  Loop: Integer;
const
  NamePrefix = 'edtMyEdit';
begin
  for Loop := 1 to 20 do begin
    { Not saving the created object's reference - uh-oh! }
    TEdit.Create(Self).Name := NamePrefix + IntToStr(Loop);
    { Not to worry - the owner can find it for us }
    with TEdit(FindComponent(NamePrefix + IntToStr(Loop))) do
    begin
      Left := 10;
      Top := Loop * 20;
      Parent := Self;
    end;
  end;
```

Pixel Colours Changing

Q If I make a bitmap file in the image editor, I sometimes find a problem. Some pixels that I give a certain colour lose their colour when the bitmap is used in a Delphi app. Are you familiar with this problem?

A This "problem" occurs with any bitmap used by a `TSpeedButton` or `TBitBtn` and in other places besides. In order to get speed buttons and bitmap buttons looking nice and consistent, they needed some capability for transparency, but Windows bitmap resources don't support that. So, they decided they would do it themselves. When the bitmap is drawn on a speedbutton or bitmap button, any pixel that has the colour of `MyBitmapObject.TransparentColor` is made transparent. How do they choose what `TransparentColor` returns? It's the bottom left pixel in the bitmap. Make that pixel a unique colour and everything's fine again.