Under Construction: Components for Workgroups 1

by Bob Swart

Delphi offers the best support around for a single programmer. The power, speed and flexibility you can get from Delphi are second to none. When it comes to team-development using Delphi, there are fewer success stories known. What about Delphi for workgroups?

In this first part of a two-part series on components for workgroups, we'll explore Delphi component management, using a shared component library. The next part will expand on that when we move on to Delphi project management using version control systems.

Delphi For Workgroups

It's not hard to imagine at least a few problems that can surface when several programmers are working together on one project. Even if the programmers in the team make good solid rules on who edits which file and when (to avoid more than one programmer using the same file at any given time), there are synchronisation problems that can occur. Does everyone have the same version of the components for the project? And what if one of them makes a change in a component? Do all of them need the same new source file? Not if only the *implementation* has changed, but it may be a problem if the *interface* has also changed. Without a good version control system and a shared repository (with all project source files stored safely on the network, where they can be shared among each other with the version control system in charge) it seems hard to even think about Delphi workgroup development. Yet, it is possible...

Component Management

Component management offers support for the problem where multiple programmers are using the same component and all of them need to use the same version of the component. It can also be used to enforce uniformity in the components that are used amongst the programmers in a team (for example, if you want to prohibit the use of a certain type of component, like VBXs) and it will add to the portability of your programmers among the different working machines (a programmer should be able to work on any machine, not only on his/her personal workstation with only his/her settings and component library).

To support component management, we need someone to be the component manager, plus a fileserver with a shared directory everyone in the team can read and a shared COMPLIB.DCL component library.

Shared Component Library

To start with the last two issues: a shared COMPLIB.DCL should be placed on the fileserver so everyone that needs to use it can read it (write permission is only needed for the component manager – more about this later). In order for someone to use a component library on the network, a setting in the DELPHI.INI file in the Windows directory needs to be changed. Usually, the COMPLIB.DCL file is taken from the C:\DELPHI\BIN directory and we now want to take if from the N:\DELPHI directory, for example.

```
[Library]
;ComponentLibrary=C:\DELPHI\BIN\COMPLIB.DCL
ComponentLibrary=N:\DELPHI\COMPLIB.DCL

;SearchPath=C:\DELPHI\LIB
SearchPath=C:\DELPHI\LIB;N:\DELPHI\LIB

SaveLibrarySource=1
```

➤ *Listing 1*

We can specify this when Delphi is running with the `Options | Open Library` dialog. It is easier, however, to just quit Delphi and modify the `ComponentLibrary` entry in the DELPHI.INI file, located in the Windows directory. Modifying this entry is not enough, however. We need to update the `SearchPath` as well to enable everyone to find the correct versions of the .DCU files (which then need to be stored on the network as well). A DELPHI.INI file that is prepared for shared usage of the Component Library and .DCU search path can be seen in Listing 1 (note that the old settings are just commented out, so you can re-install them at any time).

The advantages of this approach are as follows:

- One component builder can produce something and then all the other developers are free to use it (you don't have to re-invent the wheel). Furthermore, people may have beneficial comments and additions on components they can see (compared to components that only exist on a single machine).
- Errors that are fixed in a component are automatically shared by all users of the component, you only need to re-compile the projects that are involved.
- If the components are properly documented with on-line help this will increase the speed of use for these components, so in the end everyone will know how

to use all jointly developed components.

These advantages can lead to a real team-effort component library that will be more efficient, tested, debugged and often better documented, compared to components used by a single person only.

Of course, there are also some potential problems associated with a shared component library:

- If the COMPLIB.DCL file is in use by even one (local) version of Delphi, then it cannot be updated with new components. This means that the component manager often only has a few opportunities to install new, updated versions of components (like late at night). Of course, for an important bug-fix a broadcast message on the network that asks everyone to quit working with Delphi for one or two minutes might be acceptable.
- Even if wheels don't have to be invented more than once, they need to be documented in order to make them roll for everyone. And this documentation and online help is best written by the original component builder (which might be a good reason for some to keep components for themselves after all).
- There has to be a back-up plan for those occasions when the fileserver is unavailable.
- Starting Delphi with a large COMPLIB.DCL file on the network takes a little longer than working with a local component library (about 10 seconds).

Component Builders

For component builders in the team, there are a few potential problems. For one, the component builder cannot easily add a new prototype component to the shared COMPLIB.DCL to see how it works, because the Component Library file will be in (shared) use most of the time. In practice, there is a solution to this problem, which involves some discipline from the component builder. The component builder could make a copy of the shared COMPLIB.DCL file to the local C:\DELPHI\BIN directory. Now, with a local component library, he is able to install the prototype component again. Of course, once the component works (typically the same day), the component builder should go to the component manager and 'check in' his new component. Also, the difficult part for the component builder would be to remove the local version of the COMPLIB.DCL, give up that freedom again, and go back to the shared version of the component library. Who knows, someone might have fixed something during that day which will be installed that night as well, so if he didn't go back to the shared COMPLIB.DCL, the component builder might actually still use an outdated local version tomorrow.

```
library Complib;
{$S 32768}
uses
  StdReg,
  VBXReg,
  DBReg,
  SysReg,
  OLEReg,
  DDEReg,
  SampReg,
  DrBobReg,
  LibExpt,
  LibMain;
{$R C:\DELPHI\LIB\STDREG.DCR}
{$R C:\DELPHI\LIB\DBREG.DCR}
{$R C:\DELPHI\LIB\SYSREG.DCR}
{$R C:\DELPHI\LIB\OLEREG.DCR}
{$R C:\DELPHI\LIB\DDEREG.DCR}
{$R C:\DELPHI\LIB\SAMPREG.DCR}
{.$R N:\DELPHI\LIB\DRBOBREG.DCR}
exports
  InitLibrary name LibrarySignature resident,
  FaultHandler name FaultHandlerSignature resident;
begin
  RegisterModule('StdReg', StdReg.Register);
  RegisterModule('VBXReg', VBXReg.Register);
  RegisterModule('DBReg', DBReg.Register);
  RegisterModule('SysReg', SysReg.Register);
  RegisterModule('OLEReg', OLEReg.Register);
  RegisterModule('DDEReg', DDEReg.Register);
  RegisterModule('SampReg', SampReg.Register);
  RegisterModule('DrBobReg', DrBobReg.Register);
  RegisterModule('LibExpt', LibExpt.Register)
end.
```

➤ *Listing 2*

Component Manager

While this may seem a little hard for component builders, it is actually the task of the component manager to keep the component builders happy. In my company, we have more than a dozen Delphi programmers and we've just started to use a shared COMPLIB.DCL on our fileserver. Most Delphi programmers from our company are component users and application builders. Some of them are component builders, about three or four of them actually. Only these guys will ever need to make local copies of COMPLIB.DCL and install their own components for test and debug purposes. I trust them to come to me (the component manager) to report new components, bug fixes etc. But it is my job to make them happy with this new way of component management compared to their old way of working, when they didn't need to care about anything or anyone and could just write any component they needed themselves.

Other than that, it's up to the component manager to install new or updated components and make sure the shared COMPLIB.DCL is always up-to-date and in tip-top shape. For this, I actually wrote a little script to automatically recompile COMPLIB.DPR for me and generate a new Component Library when I need it.

COMPLIB.DPR

Did you know that COMPLIB.DCL is in fact nothing more or less than a Dynamic Link Library with a .DCL extension instead of a regular .DLL

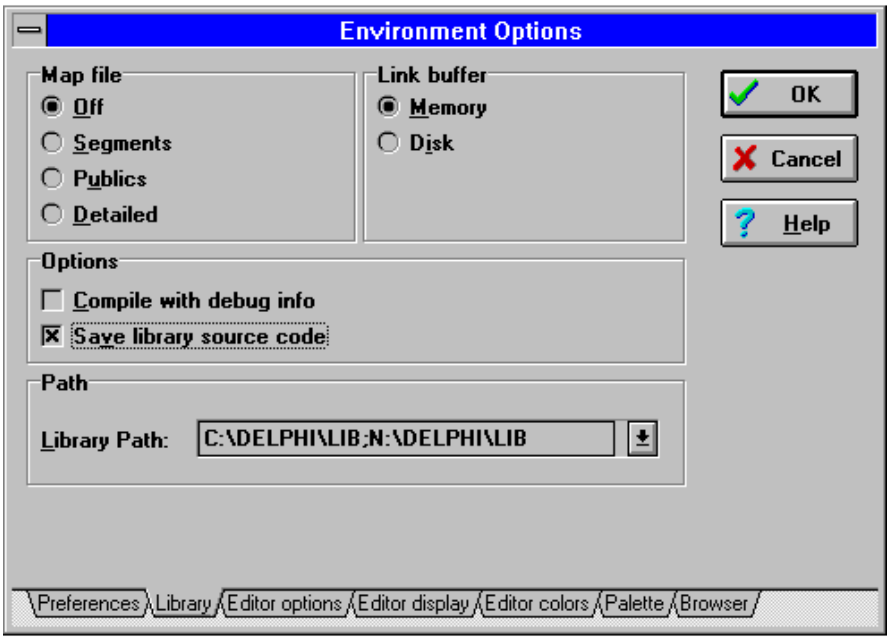
extension? You can see this for yourself if you set the Save Library Source option in the Options | Environment dialog (the same place where you can check the library path) – see Figure 1.

Now, when you re-compile your COMPLIB.DCL with this option set on, you'll find a COMPLIB.DPR file in your C:\DELPHI\BIN directory. This file is the one that is generated by Delphi and used to build a DLL with internal name COMPLIB (so Delphi can recognise it). You can even rename COMPLIB.DCL to something like DRBOB.DCL and enter that file as your new component library. As long as the original source file to reproduce the DLL is as in Listing 2, there should be no problem.

The listing contains the definition of the default COMPLIB.DCL as shipped with Delphi. However, I've already added a unit of my own to it, namely DrBobReg, and I called the DrBobReg.Register procedure. I could also have added a .DCR file with component bitmaps, to group them together instead of adding them to the individual component source files (more about this later).

Note that you can actually use this source file to reconstruct COMPLIB.DCL when it has become corrupted or even disappeared from your machine.

I once managed to crash it, tried to rebuild it and ended up with an empty component library and a COMPLIB.DPR that only exported the InitLibrary and FaultHandler (no components were registered). I could either have re-installed COMPLIB.DCL from the Delphi CD, or copy a backup of COMPLIB.DPR and re-compile the component library. The latter seemed to be the easier solution and it worked like a dream. So, instead of having a backup copy of COMPLIB.DCL, have a back-up copy of the COMPLIB.DPR file, since you can actually see from the source which components were installed and you can make your own changes if you want as well.



➤ *Figure 1*

```
unit DrBobReg;
interface
{ the .DCR files for some of the components that are installed: }
{$R TTT.DCR}
{$R TBUUCODE.DCR}
procedure Register;
implementation
uses DrBob,    { TDrBob }
     Convert,  { TConvert }
     TDosEnv,  { TDosEnvironment }
     TTT,      { TTicTacToe }
     TBUUCode, { TBUuEncode, TBUuDecode }
     FileName,
     PictEdit,
     SysUtils, DsgnIntf, Classes;
procedure Register;
begin
  { components }
  RegisterComponents('Dr.Bob', [TDrBob,
                                TConvert,
                                TDosEnvironment,
                                TTicTacToe,
                                TBUuEncode,
                                TBUuDecode]);
  { property editors }
  RegisterPropertyEditor(TypeInfo(TFilename), nil,
                         'InputFile', TFilenameProperty);
  PictEdit.Register
end {Register};
end.
```

➤ *Listing 3*

DRBOBREG.PAS

But what about this extra DrBobReg unit? What magic is hidden in this single source file? Actually, this is the single unit that I use to register all my shared components. And before we go into the benefits of registering components in one big unit, let's first see what DrBobReg actually looks like – Listing 3.

There are a few special things about this unit. For one, it only registers the components, it does not define them. The components themselves are defined in their own units, but they are registered in this multi-component register unit. Also, note the fact that I include two .DCR files in this unit. This means that I don't need to include them in the component units themselves. Which means that the component units will be stripped of the register procedure as well as the .DCR component bitmap file. And since the DRBOBREG.PAS file is only used to construct COMPLIB.DCL, not the final executable, this additional

information is also stripped from the final executable.

Another advantage is the fact that when I move up to Delphi 2.0, I only need to convert the .DCR files and move this unit with the converted .DCR files to the Delphi 2.0 LIB directory. I don't have to modify the single component source files at all (except when porting issues arise, which will be dealt with another time).

As a consequence of this approach, my component source units do not need a register procedure anymore. Which makes them even smaller when linked into the final executable. For example, take a look at the `TDrBob` base class component in Listing 4. Real short, eh?

The `FAbout` property of the component base class is useful to identify when a new version of a certain component has been installed. Just override the `Create` constructor of the new version and put another string in the `FAbout` property.

```
unit DrBob;
interface
uses Classes;
Type
  TDrBob = class(TComponent)
  private
  { Private 'dummy' field for read-only design property About... }
    Dummy: String;
  protected
  { Protected 'FAbout' declarations }
    FAbout: String;
  public
  { Public class declarations (override) }
    constructor Create(AOwner: TComponent); override;
  published
  { Published About property }
    property About: String read FAbout write Dummy;
  end {TDrBob};
implementation
constructor TDrBob.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FAbout := 'TDrBob 1.0 (c) 1996 by Bob Swart (aka Dr.Bob - 100434,2072)'
end {Create};
end.
```

➤ *Listing 4*

```
[Experts]
ExptDemo=C:\DELPHI\BIN\EXPTDEMO.DLL
DrBob=N:\DELPHI\EXPERTS\DLL\DRBOB.DLL

[DrBob]
Delay=0
```

➤ *Listing 5*

DRBOBC.ZIP

The DRBOBREG.PAS register unit and DRBOB.PAS component are part of a collection of components and property editors that I've put together for this issue's disk as file DRBOBC.ZIP (you can also find it at http://www.pi.net/~drbob/). The components include:

```
TDrBob
TConvert
TDosEnvironment
TTicTacToe
TBUuEncode & TBUuDecode
```

There's also a `TFileName` and a new `TPicture/Image` property editor, plus ResConv, a DOS/Windows command-line utility to convert 16-bit .RES or .DCR files (containing bitmaps) to 32-bit .RES or .DCR files – very handy for Delphi 2.

The installation of the component collection is simple: just copy all these files in this archive to a single directory (for example C:\DELPHI\DRBOB) and add the file DRBOBREG.PAS to the list of installed components (in the `Options | Install Components` dialog). Note that one install file is enough, you don't need to install each component separately!

`TDrBob` is the 'kernel' component from this collection: almost all the others are derived from this one.

`TConvert` converts numerical values to hexadecimal and roman digits (and back) using an internal field and several conversion routines (from Issue 1).

`TDosEnvironment` returns the DOS environment strings in a property of type `TStringList`.

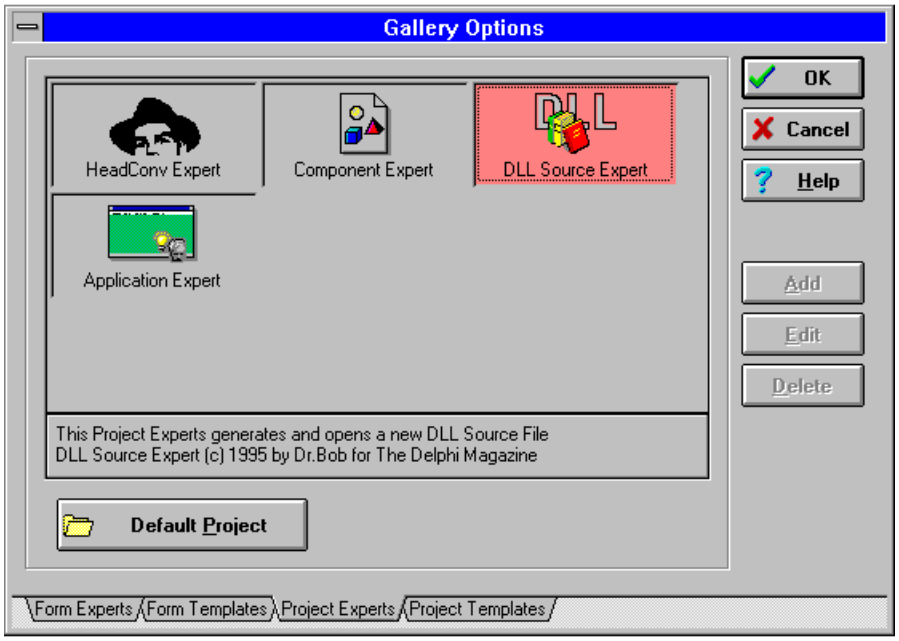
`TTicTacToe` is the component for the game also known as noughts-and-crosses, based on a strategy MAGIC.DLL (this is a new version!). Make sure MAGIC.DLL is available in your \WINDOWS\SYSTEM directory or the directory with the final application. A first version was included with Issue 2.

`TBUuEncode` and `TBUuDecode` are wrapper components around my UUCODE.DLL, which implements the uuencode/uudecode algorithm to en/decode a file which may contain any character into a file with a standard character set, so this file can be sent over diverse networks that do not support binary files (such as the internet). For more information about these components, read the TBUUCODE.HLP help file. Note that this component has its own property editors installed to give you as much support during design time as possible (using the `Activate` property it is even possible to actually encode and decode files at design time)!

Dr.Bob's Enhanced Picture Editor is a replacement for the Picture Editor which Borland provided with Delphi, which unfortunately cannot give you a preview when you are browsing through a directory of .BMP files). When you have installed DRBOBREG.PAS you can find the new Picture Editor in, for example, the following places: `TImage` (picture), `TBitBtn` (glyph) and `TSpeedButton` (glyph).

Shared Experts

A final advantage when sharing the component library that I offered to the Delphi programmers at my company was the fact that they could also share my collection of



➤ *Figure 2*

Delphi 1.0 experts from the same fileserver. The DELPHI.INI file was again modified as shown in Listing 5 (the `Delay` entry is used to limit the time the splash screen is shown – it can be removed entirely for registered users, by the way).

The disk with the last issue of the Delphi Magazine already contained the DRBOB1.ZIP file, with the installation program and a free (fully functional) trial version included (you can also find it in Library 22 of the DELPHI forum on CompuServe and from my Web site). You'll get the Project Experts in the Gallery which are shown in Figure 2, for example.

Version Control System?

Sharing the same DRBOB.DLL expert DLL is nice but does not have any additional benefits (compared to using the DRBOB.DLL on a standalone workstation). There is no shared repository with files when using these experts. For this we need a true version control system, one that is connected to a shared database on the same fileserver. I've built one myself, called ViCiouS, which does exactly that and can be used to enhance the component management model into a true project management model. But this part of the Delphi for Workgroups story is best left for next time...

Bob Swart (aka Dr.Bob at http://www.pi.net/~drbob/) is a professional software developer using Borland Delphi, C++ and Pascal for Bolesian in The Netherlands, and a freelance technical author for The Delphi Magazine. In his spare time Bob likes to watch video tapes of Star Trek Voyager and Deep Space 9 with his 2-year old son Erik Mark Pascal.