An "*Expert*" Expert

by Paul Warren

Ask any Delphi programmer what makes Delphi special and you'll hear about the visual development tools, the components, the database connectivity and the superb compiler. You may also hear about the open tools API. But what is the open tools API? Is it just a fancy way of integrating third party tools or is it more?

The Project and Form templates are useful for quickly creating frameworks, but they are fairly simple and not as flexible as I would like. Experts, on the other hand, are extremely flexible, somewhat undocumented and even a little mysterious. This is the perfect combination to catch my interest.

Armed with my back issues of (you guessed it) *The Delphi Magazine* I set out to explore experts. The first Expert I ever installed was Bob Swart's *DLLSkeleton Expert*. As you recall, Dr.Bob explained how he created this in Issue 3. I used it to create the framework for my password protection system (included on a recent free disk). The *DLLSkeleton Expert* was so useful I started thinking about creating my own experts.

One bad habit I have (and it's not the only one) is forgetting to add my copyright panel and a description to my source files before releasing them. I had often wished the Delphi component expert would do this for me. I seriously thought about creating a component expert but if experts were as useful as I thought they'd be I would be writing more of them, maybe many more. Then I wondered whether I could write an expert to create experts. Well, let's see, I'll start by calling it an "Expert" expert...

The Expert Form

Creating an expert is quite straight forward (at least it is after reading the various articles that have appeared in these pages). My intention with this article is to briefly cover the creation of my "Expert" expert and concentrate on some tips that I found useful.

The first step is to design the interface form. There are three types of expert: Project, Form and Standard (four in Delphi 2.0). I wanted to offer users the option to create any of these. An expert also needs a name and may or may not need a form. I also decided to add memo fields for a description and for copyright information. Figure 1 shows the form for my "Expert" expert.

Every expert has to have a unique ID string, the comments in the `ExpIntf` unit say that the format of the ID string is, by convention, `CompanyName.ExpertFunction`, hence the company name edit.

The Expert

After I had my form designed I added the expert's type declaration. All experts derive from the `TIExpert` abstract base class and they must declare and override at least some of the `TIExpert` methods.

Next, I added the override methods. I had already decided that my "Expert" expert should be a standard expert. It really made no sense for it to be a form or project expert. `GetStyle` therefore returns `esStandard`. The rest of the code is shown in Listing 1.

The `Execute` method is the heart of a standard expert. It creates the form and launches the expert. The

```
function ThgExpertExpert.GetStyle: TExpertStyle;
begin
  Result := esStandard;
end;
function ThgExpertExpert.GetIDString: String;
begin
  Result := 'hgsoft.ExpertExpert';
end;
function ThgExpertExpert.GetComment: String;
begin
  Result := ''; { not needed for esStandard }
end;
function ThgExpertExpert.GetGlyph: HBITMAP;
begin
  Result := 0; { not needed for esStandard }
end;
function ThgExpertExpert.GetName: String;
begin
  Result := 'Expert Generator';
end;
function ThgExpertExpert.GetState: TExpertState;
begin
  Result := [esEnabled];
end;
function ThgExpertExpert.GetMenuText: String;
begin
  Result := 'Home&Grown''s Expert Expert...';
end;
procedure ThgExpertExpert.Execute;
begin
  if not Assigned(hgExpExpert) then
    hgExpExpert := ThgExpExpert.Create(Application);
  hgExpExpert.ShowModal;
end;
```

➤ *Left: Listing 1, Below: Figure 1*

form can be modal or modeless as desired. For my "Expert" expert I chose a modal dialog style.

Believe it or not this is already an expert. It can be compiled into COMPLIB.DCL and the form will appear when you select the corresponding menu item. Unfortunately, if you want an expert that actually *does* anything you have to do a little more work.

Making It Work

Since the are several options available to the user I had to have some flags to hold the choices. I chose a set type called `TExpAttrs` for the four choices. The `StyleClick` method sets or clears the set elements.

For the copyright information I decided to load a text file from disk using the `Memo2.LoadFromFile` method. This way any user can include copyright information by simply putting a file in their \WINDOWS directory called CPYRIGHT.TXT.

The rest of the work is done in the `BitBtn1Click` event (the `OK` button). First we have to include the `ToolIntf` unit in the `uses` clause. Then check if `ToolServices` is not `nil`. This guarantees the IDE is running. Next we ask `ToolServices` for a new module name. The comments in `ToolIntf` explain that `GetNewModuleName` automatically generates a valid file name and unit identifier. With valid unit and file names we can generate the expert.

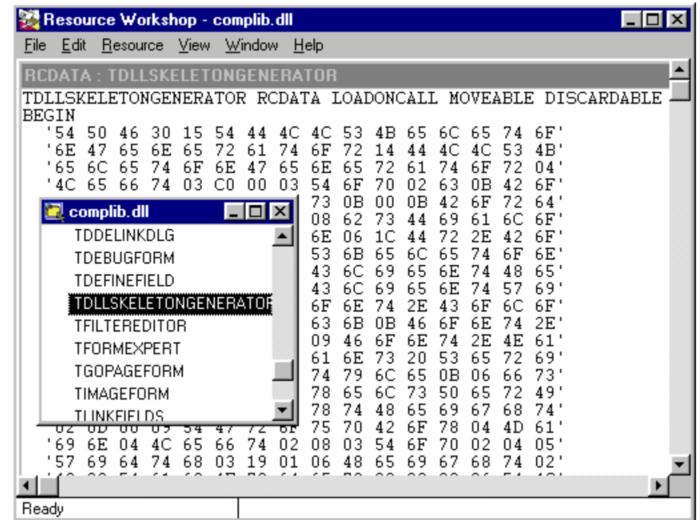
Generating the expert involves creating a form (if requested), creating synchronous source code, writing the form to disk and passing the source to the editor. There is more than one way to do this. Marco Cantu and Bob Swart wrote their *Database Expert* form and source to disk and then copied the source to a virtual stream via a file stream.

If you look at the source in the \DELPHI\DEMOS\EXPERT directory you will find the source of a number of experts from EXPDEMO.DLL. The file DLG.PAS uses a different way to create an expert: both form and source are written directly to virtual streams. The process is completed by calling `ToolServices.CreateModule`, passing the form and source streams as parameters. This was the approach I took.

```
function ThgExpExpert.DoFormCreation(const FormIdent: string): TForm;
{ Create the dialog defined by the user }
begin
  Result := TForm.Create(nil);
  Proxies.CreateSubClass(Result, 'T' + FormIdent, TForm);
  with Result do begin
    BorderStyle := bsSizeable;
    Width := 400;
    Height := 250;
    Position := poScreenCenter;
    Name := FormIdent;
    Caption := FormIdent;
  end;
end;
function ThgExpExpert.CreateForm(const FormIdent: string): TMemoryStream;
var NewForm: TForm;
begin
  Result := nil;
  NewForm := DoFormCreation(FormIdent);
  try
    Result := TMemoryStream.Create;
    Result.WriteComponentRes(FormIdent, NewForm);
    Result.Position := 0;
  finally
    NewForm.Free;
  end;
end;
procedure ThgExpExpert.BitBtn1Click(Sender: TObject);
var
  FileName: TFileName;
  ISourceStream, IFormStream: TIMemoryStream;
  UnitIdent, FormIdent: string;
begin
  { code for expert goes here }
  if ToolServices <> nil then begin
    { I'm an expert!! }
    if ToolServices.GetNewModuleName(UnitIdent, FileName) then
    try
      UnitIdent := LowerCase(UnitIdent);
      UnitIdent[1] := Upcase(UnitIdent[1]);
      FormIdent := 'Form' + Copy(UnitIdent, 5, 255);
      if eaCreateForm in Definition then begin
        IFormStream := TIMemoryStream.Create(CreateForm(FormIdent));
        IFormStream.AddRef;
        ISourceStream :=
          TIMemoryStream.Create(CreateSource(UnitIdent, FormIdent));
      end else
        ISourceStream := TIMemoryStream.Create(CreateSource(UnitIdent, ''));
      try
        ISourceStream.AddRef;
        if eaCreateForm in Definition then
          ToolServices.CreateModule(FileName, ISourceStream, IFormStream,
            [cmShowSource, cmShowForm, cmUnNamed, cmMarkModified])
        else
          ToolServices.CreateModule(FileName, ISourceStream, nil,
            [cmShowSource, cmUnNamed, cmMarkModified]);
      finally
        ISourceStream.OwnStream := True;
        ISourceStream.Free;
      end;
      Close;
    finally
      if eaCreateForm in Definition then begin
        IFormStream.OwnStream := True;
        IFormStream.Free;
      end;
    end;
  end;
end;
```

➤ *Listing 2*

The `BitBtn1Click` event code is shown in Listing 2 along with the `CreateForm` and `DoFormCreation` methods. One interesting feature of this code is the call to `Proxies.CreateSubClass` method. As far as I can tell the `Proxies` unit is completely undocumented. There is not even a PROXIES.INT interface included with Delphi 1 or 2. `Proxies.CreateSubClass` obviously creates a form in memory



Left: Figure 2, Right: Figure 3

which does not get destroyed when the expert form closes.

Creating the synchronous source is straightforward but tedious and since the full source code for the "Expert" expert is included with this month's free disk I will leave it to you to explore how to generate the source. All that remains to be done is re-compile the library and the "Expert" expert is ready to use.

```
procedure TForm1.FormShow(Sender: TObject);
var i: integer;
begin
  ListBox1.Clear;
  if ToolServices <> nil then { I'm an expert!! }
    for i := 0 to ToolServices.GetUnitCount do
      ListBox1.Items.Add(ToolServices.GetUnitName(i));
end;
procedure TForm1.OkButton1Click(Sender: TObject);
begin
  Close;
end;
```

Listing 3

Using The "Expert" Expert

Let's create a useful expert using the "Expert" expert. While you can find the units belonging to any project directly from the IDE, only the file names are available. I wanted a listing with fully qualified paths. In the `ToolIntf` unit there are methods to get the number of units in the active project and the unit name fully qualified. Why don't we create a `UnitsInProject` expert?

Run the "Expert" expert from the `Help` menu. Set up the main form as shown in Figure 2. When you click the `OK` button a framework for the `UnitsInProject` expert is created and opened in the editor.

Add an `OK` button and a `TListBox`. The only code you need to write is in the `Form1.OnShow` event and the `OkButton1.OnClick` event. Listing 3 shows the code. Save the file as UNITSIN.PAS in a directory on your library path and add it to COMPLIB.DCL. That's it: simple isn't it? Now you can create experts at the drop of a hat.

What Are Experts

It's all well and fine to create experts at will but if you're anything like me you want to know how they work. `ExpIntf` and `ToolIntf` are mysterious and marginally documented. How do these calls to the library and editor work?

While I was creating my "Expert" expert I tried poking around in the COMPLIB.DCL using the venerable Resource Workshop. On the chance that COMPLIB.DCL was a .RES file I renamed it COMPLIB.RES. Some resources were visible, but there was a lot of garbage too. Obviously COMPLIB.DCL is not a .RES file.

I tried renaming it COMPLIB.DLL and this time Resource Workshop opened and de-compiled it perfectly. Inside I found resources corresponding to the component palette, the various experts and many other goodies like cursors and bitmaps. What a treasure! This is the heart of Delphi's bag of tricks. If you look at Figure 3 you'll see Dr.Bob's *DLLSkeleton Expert* in all its binary glory.

So COMPLIB.DCL is really a DLL. Makes you wonder whether you can insert your own resources directly into it with Workshop or if you can access resources from outside the IDE. Are you getting any ideas?

Conclusion

Although I have been brief, we have looked at all the steps needed to create an expert. We have developed a useful tool and started putting it through its paces by creating the `UnitsInProject` expert. Perhaps most importantly, though, we've had a glimpse of the true elegance of Delphi in the open tools API.

Paul Warren runs HomeGrown Software Development in Langley, British Columbia, Canada and can be reached by email at hg_soft@uniserve.com or visit http://haven.uniserve.com/~hg_soft