# *Under Construction:* Delphi 3 ActiveForms

*by Bob Swart*

This month I'll show what Delphi 3 ActiveForms are: how to create them, how to work with them and how to use our Wizard template from last month's column to create ActiveForm Wizards for the internet.

One of the complaints I often heard about Delphi 2 was the lack of OCX support. Sure, we could register and use just about any OCX, but we couldn't create them ourselves using Delphi 2, without using the third-party *OCX Expert* tool. Fortunately, Delphi 3 fixes this omission. There are at least two major new ActiveX support features added to Delphi 3 (apart from native support for COM and DCOM). The first is one step ActiveX, which allows us to create an ActiveX control in (you guessed it) one step, either from an existing VCL component or from scratch.

The second new feature, which I think is even more interesting, is called ActiveForms. ActiveForms are true ActiveX controls that use the Delphi form as a container for other Delphi components. ActiveForms publish ActiveX property pages and type libraries for adding high-speed functionality to other environments, such as Internet Explorer, Visual Basic, etc. In fact, we can use ActiveForms to deliver our applications over the internet or the corporate intranet, as we'll see shortly. Because of the potential dangers of ActiveX controls, intranets are probably the best place for them, as we shall see.

Since an ActiveForm is in fact just an ActiveX, I often use the two terms for the same thing. And why not? Everything I say about ActiveX controls is also valid for ActiveForms.

## New ActiveForm

To create a new ActiveForm, we only have to select `File | New` in the Delphi 3 IDE and pick the `ActiveForm` item from the `ActiveX` page. In the `ActiveForm Wizard` dialog (Figure 1) we derive our new ActiveForm class from the `TActiveForm` base class. For the implementation unit and project filenames I usually pick short filenames (ie 8.3 length) to avoid problems when moving ActiveX library projects over networks or to different machines. Note that ActiveForms can only be added to an ActiveX library project (resulting in a file with the OCX extension).

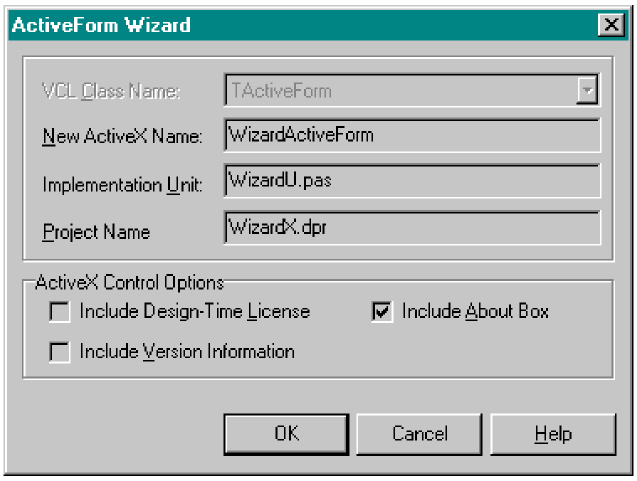
The ActiveForm Wizard also allows us to specify the use of version information, which is needed if we ever want to register the ActiveForm in Visual Basic 4 (VB4). However, I had some problems running ActiveForms in VB4, while in VB5 they worked fine. The `Design-Time License` option can be selected to make sure people can't use the ActiveForm in a development environment at design-time unless they have a key for the control stored in a .LIC file with the same name as the ActiveForm.

After we click OK a new ActiveX library project is created, with one ActiveForm and an AboutBox form. Taking a look inside the code generated for the ActiveForm, we see the ActiveX project itself is just like a regular library (Listing 1).

The `{$E ocx}` compiler directive tells Delphi 3 to compile this `WizardX` library and give it an OCX Extension. The four ActiveX APIs are already exported from this library. Everything is already done for us, we can concentrate on the ActiveForm itself.

We quickly notice that there's one big difference between a regular form and a `TActiveForm`:

➤ *Figure 1*


ActiveForm Wizard

VCL Class Name: TActiveForm
New ActiveX Name: WizardActiveForm
Implementation Unit: WizardU.pas
Project Name: WizardX.dpr

ActiveX Control Options
☐ Include Design-Time License
☑ Include About Box
☐ Include Version Information

➤ *Listing 1*

```
library WizardX;
uses
  ComServ,
  WizardX_TLB in 'WizardX_TLB.pas',
  WizardU in 'WizardU.pas' {WizardActiveForm: TActiveForm}
                           {WizardActiveForm: CoClass},
  About1 in 'About1.pas' {WizardActiveFormAbout};
exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;
{$R *.TLB}
{$R *.RES}
{$E ocx}
begin
end.
```

ActiveForms also have (OLE) interfaces. This is not something to worry about, but just a fact of life that happens behind the scenes. Probably the easiest way to spot this is the definition of the TWizard-ActiveForm itself (full source is on the disk):

```
type
  TWizardActiveForm =
    class(TActiveForm,
          IWizardActiveForm)
  ...
  end;
```

Multiple inheritance? Sure looks like it! Well, rest assured, it's not. Or maybe it is, but only of *interfaces*, and not of *classes*. A Delphi class can be derived from one other Delphi class only, but from one or more OLE interfaces, such as IDispatch or IWizardActiveForm in our case (which was generated by the ActiveForm Wizard for us). See Brian Long's Delphi 3 preview article in the May issue for details.

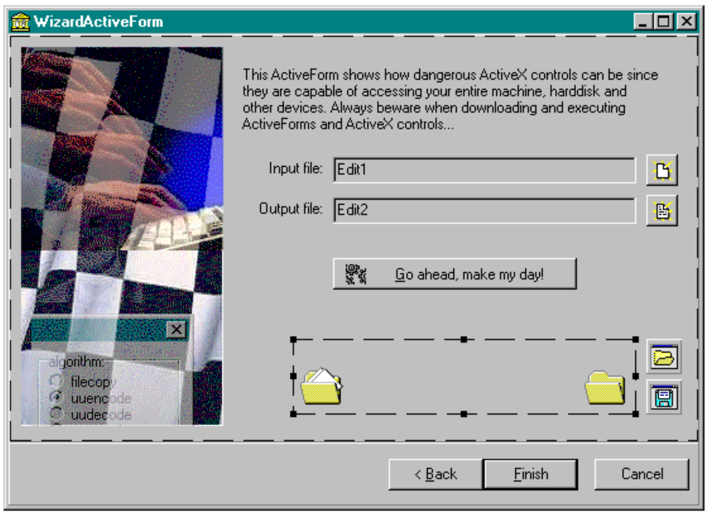
Another thing to keep in mind is the fact that when compiled as an ActiveForm, only the ActiveForm itself will be "available" to the outside world. When used in other development environments, such as C++Builder or VB, there's no way to access the controls on the ActiveForm individually. As far as these environments are concerned, the entire ActiveForm is just one big control. So, if we need to somehow interface properties, methods or events from the controls on the ActiveForm to the outside world, we need to re-wrap them to the ActiveForm itself. See pages 37-5 and 38-2 of the *Delphi 3 Developer's Guide* for more information.

But we don't have to know all this right now. All we have to do here is paint the form just as we're used to with Delphi.

## WizardActiveForm

Now, lets drop the TWizardTemplate component from last issue onto the ActiveForm, making sure to resize the form and components so you again have something that looks close to the Wizard from last month (thanks to Hubert Klein Ikkink, hubert@bolesian.nl, for the new bitmap). Now drop a TLabel with WordWrap set to True and insert a message specific to this Wizard. Then, drop two more TLabels for input and output filenames, two TEdits, two TSpeedButtons and an Action button, then a TOpenDialog and TSaveDialog and one of the new Delphi 3 TAnimate components. Voila! Figure 2: a special Wizard that can copy files from one place to another on a local machine. The only code I needed to add are the events for the input, output and action buttons. See Listing 2.

➤ *Figure 2*



This ActiveForm shows how dangerous ActiveX controls can be since they are capable of accessing your entire machine, harddisk and other devices. Always beware when downloading and executing ActiveForms and ActiveX controls...

Input file: Edit1

Output file: Edit2

```
procedure TWizardActiveForm.BtnInputFileClick(Sender: TObject);
begin
  OpenDialog1.FileName := EditInputFile.Text;
  if OpenDialog1.Execute then
    EditInputFile.Text := OpenDialog1.FileName;
end;
procedure TWizardActiveForm.BtnOutputFileClick(Sender: TObject);
begin
  SaveDialog1.FileName := EditOutputFile.Text;
  if SaveDialog1.Execute then
    EditOutputFile.Text := SaveDialog1.FileName;
end;
procedure TWizardActiveForm.EditInputFileChange(Sender: TObject);
begin
  BtnAction.Enabled := (EditInputFile.Text <> '') and (EditOutputFile.Text <> '')
end;
procedure TWizardActiveForm.BtnActionClick(Sender: TObject);
begin
  { copy EditInputFile.Text to EditOutputFile.Text }
  Animate1.Active := not Animate1.Active;
end;
```

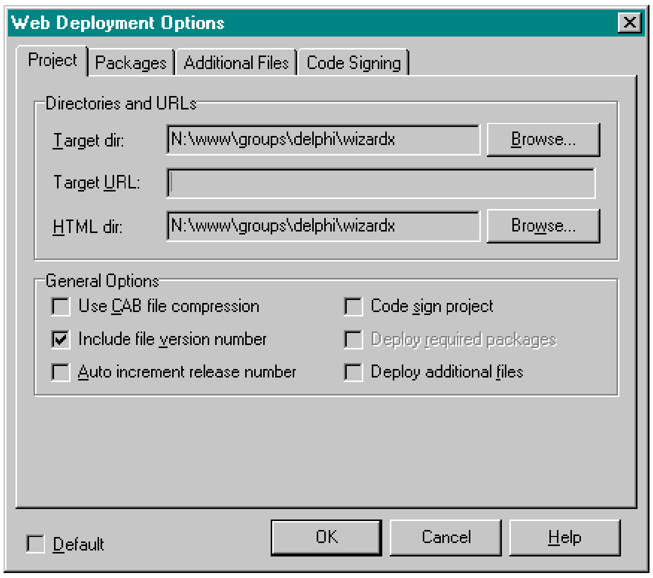
➤ *Listing 2*

The cool new TAnimate component contains a set of pre-defined animations for CopyFile, CopyFiles, DeleteFile, EmptyRecycle, FindComputer, FindFile, FindFolder and RecycleFile. You can also add your own AVI animation files (most have about 30 frames, keep that in mind when designing yours).

This illustrates the power and danger of ActiveForms and ActiveX controls in general, especially when used in a Web Browser: once downloaded, registered and running, the ActiveX control has access to your entire PC and can do as it pleases. For this reason, code signing has been invented. But still, it's important to realise that an ActiveX control or ActiveForm running in your Web Browser can in fact access your *entire* machine and wreak havoc. My recommendation is to never download, register and run ActiveX controls from unknown third parties.

## Web Deployment

Compiling the ActiveX library project results in an ActiveX file, with the .OCX extension: WizardX.OCX in our case. We can use this file in a Win32-only Web Browser such as Microsoft Internet Explorer 3.0 or higher and Netscape Navigator 4.0 or higher, but first we need to perform one final step: specify the Web Deployment options in the Project menu (Figure 3).



Web Deployment Options

Project | Packages | Additional Files | Code Signing

Directories and URLs
Target dir: N:\www\groups\delphi\wizardx
Target URL:
HTML dir: N:\www\groups\delphi\wizardx

General Options
[ ] Use CAB file compression
[✓] Include file version number
[ ] Auto increment release number
[ ] Code sign project
[ ] Deploy required packages
[ ] Deploy additional files

[ ] Default

➤ *Figure 3*

```
<HTML>
<H1> Delphi ActiveX Test Page </H1><p>
You should see your Delphi forms or controls embedded in the form below.
<HR><center><P>
<OBJECT
  classid="clsid:2A16BE8C-C309-11D0-A1E6-00805F6C3277"
  codebase=
    "http://www.bolesian.nl/groups/delphi/wizardx/WizardX.ocx#version=1,0,0,0"
  width=350
  height=250
  align=center
  hspace=0
  vspace=0
>
</OBJECT>
</HTML>
```

➤ *Listing 3*

The first page of this dialog has to be filled in completely before we can deploy our ActiveForm. As the on-line help and manuals are not too clear about what to specify as the `Target dir`, `Target URL` and `HTML dir`, let's look at these in detail.

The `Target dir` should contain the full pathname of the directory on the web server (or intranet fileserver) where the ActiveX is to be copied to. If you are already working on the web server, this can be a local path. Otherwise, it will be a mapped drive (such as drive N: in my case) or a UNC pathname, such as:

```
\\BOLESIAN_1\VOL3\www\groups\
  delphi\wizardx
```

The `Target URL` should contain the URL (without the filename) for the outside world to get to the above pathname, in my case:

```
http://www.bolesian.nl/groups/
  delphi/wizardx
```

on our local intranet. This entry is used to generate the codebase for the `OBJECT` tag in the generated HTML file.

Finally, the `HTML dir` should contain the full pathname of the directory, either local or on the web server, where you want to create your HTML test page. I usually place this file on the web server as well, in the same directory as the ActiveForm itself.

The general options include the use of CAB file compression, which is especially useful in an internet environment (where every byte to download counts), since ActiveForms get big really fast. We can minimise the size of an ActiveForm by using both CAB and packages (more about packages in the next issue). For local intranet deployment, however, I usually don't bother with CAB or packages.

Code signing is another option that's most useful in an internet environment. In a local intranet, I'd trust most ActiveX or ActiveForms without code signing (this will vary from company to company), on the internet I'd *never* download an ActiveX or ActiveForm without code signing.

Actually, this last thing is not hard to enforce if you're using Netscape Navigator version 3, which doesn't support ActiveX controls without the ScriptActive plug-in from NCompass. A 30-day trial version is at

```
http://www.ncompasslabs.com/
  products.htm
```

Once we've finished filling in the `Web Deployment Options` dialog, it's time to select the menu item `Project | Web Deploy` which then copies the ActiveX to the specified directory on the web server and generates the HTML test page (either local or also on the web server). The generated HTML test page for my WizardX ActiveForm is shown in Listing 3.

The first thing I noticed is the fact that the codebase contains the full path of the WizardX.ocx, while this file is in the very same directory on the web server where the HTML test page (and final deployment page) is. So, I edited this file and changed this line to:

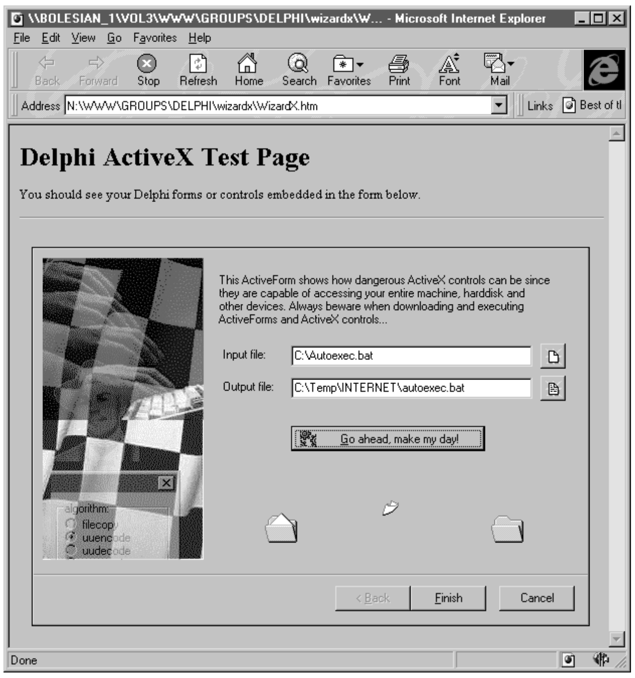
```
codebase=
  "WizardX.ocx#version=1,0,0,0"
```

and sure enough, things still work fine (it turns out you can specify a relative path in the codebase, knowledge which can be quite handy if you want to keep your HTML pages and ActiveX controls together at all times).

Now, how do we use the ActiveForm from anther machine? We don't need to register the ActiveForm before downloading it via the web. Actually, that's the whole point of web deployment, to take care of this nasty business. All we need to do from another machine is point an ActiveX enabled web browser to the HTML test page we've just generated.

The codebase will tell the web browser where to download the ActiveX control from (ie download it from the web server and place it

➤ *Figure 4*


\\BOLESIAN_1\VOL3\WWW\GROUPS\DELPHI\wizardx\W... - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font Mail
Links Best of t

Address N:\WWW\GROUPS\DELPHI\wizardx\WizardX.htm

**Delphi ActiveX Test Page**

You should see your Delphi forms or controls embedded in the form below.

This ActiveForm shows how dangerous ActiveX controls can be since they are capable of accessing your entire machine, harddisk and other devices. Always beware when downloading and executing ActiveForms and ActiveX controls...

Input file: C:\Autoexec.bat

Output file: C:\Temp\INTERNET\autoexec.bat

on the local machine. Let's take a look at deploying with Microsoft Internet Explorer 3.

## Internet Explorer 3

With Microsoft Internet Explorer 3.0 (included on the Delphi 3 CD-ROM) you must go to the `View | Options | Security` dialog and set the security level to medium. A level of high will cause Internet Explorer to skip all non-certified ActiveX controls. Setting the level to medium you get a warning message. Setting the security level to low will stop all messages, but that's a little risky when you connect on the internet, though on a corporate intranet it would be fine.

After you've allowed Internet Explorer to go ahead and try installing the ActiveForm, first the GUID is checked to see if the control has already been registered. If not, Internet Explorer will use the codebase to download the ActiveX control to the \windows\occache subdirectory, call `LoadLibrary` on the control and call the control's `DllRegisterServer` function. If the library cannot be downloaded (incorrect URL), or cannot be loaded (because it contains static references to DLLs which Windows cannot find) or if Windows can't find the `DllRegisterServer` API, or the call to `DllRegisterServer` fails, then all we see is an empty box in Internet Explorer with a small `X` in the upper left corner.

If all is OK the ActiveForm will be installed and registered and Internet Explorer will then attempt to create an instance of your control. Your `DllGetClassObject` function will be called, from which Internet Explorer will obtain an `IClassFactory` for our ActiveForm control. The browser will then call `IClassFactory.CreateInstance` to create an instance of the ActiveForm. If for some reason our control fails to create itself (for example when an exception is raised in the constructor) we'll also see the small red `X`.

Finally, just make sure you're running Internet Explorer version 3.01 or 3.02, since the plain version 3.0 not only has some security problems (you can activate any executable on your disk!), it also has major download problems with ActiveX controls.

Anyway, assuming everything above goes as planned, Internet Explorer activates our ActiveX control, which for the `WizardActiveForm` looks like Figure 4.

Wow! Our first Delphi WizardActiveForm alive and kicking in Internet Explorer! Let's move with the mouse to one of the edit boxes and edit some text. Hmmm, as soon as you try to use the arrow keys, something seems to be wrong. The problem is that the controls don't see the arrow or tab keys, so the only way to navigate around within them is with the mouse: not nice for a keyboard intensive activity. This is caused by a bug in Internet Explorer which causes it to eat the arrow keys until you tab out of the control and then back in again. Note that it's *not* a bug in Delphi and the same ActiveForm will function just fine in other environments.

## ActiveForm Employment

Apart from using ActiveForms in a web browser on the internet or intranet, we can also deploy them in a development environment such as Visual Basic (5, 4 gives some problems), or, of course, Delphi (2 or higher). Since this is *The Delphi Magazine*, let's try it in Delphi 3 just for the fun of it.

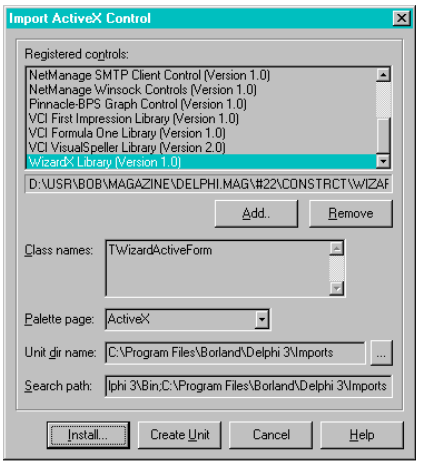
Select the `Component | Import ActiveX Control` menu item, click `Add` and select WizardX.OCX (or whatever).

As we can see from Figure 5, the ActiveX control is read and the class name `TWizardActiveForm` is identified. Note that we can have multiple class names, since one ActiveX library project can contain more than one ActiveForm. Now, what should we do next: click `Create Unit` or `Install`? Actually, `Install` also creates a unit, opens the Package Editor and installs the unit in the package you specify (Figure 6). `Create Unit` just creates a unit and displays the unit code in the code editor without including the unit in the current project. So, usually you will click `Install`.

After installation we can find the `TWizardActiveForm` on the `ActiveX` palette page and if we drop it on a new form we see the old `TWizardTemplate` back again, but this time as an ActiveX control. If we right-click on the ActiveX (ActiveForm), we see a pop-up menu with various options (such as property pages: a topic for another time), including the ActiveX AboutBox we implemented.

## Dynamic ActiveForm

We can even dynamically create an instance of a `TWizardActiveForm` on the fly. For this, the information in the WizardX_TLB.pas unit with the


Import ActiveX Control

Registered controls:

NetManage SMTP Client Control (Version 1.0)
NetManage Winsock Controls (Version 1.0)
Pinnacle-BPS Graph Control (Version 1.0)
VCI First Impression Library (Version 1.0)
VCI Formula One Library (Version 1.0)
VCI VisualSpeller Library (Version 2.0)
WizardX Library (Version 1.0)

D:\USR\BOB\MAGAZINE\DELPHI.MAG\#22\CONSTRCT\WIZAF

Class names: TWizardActiveForm

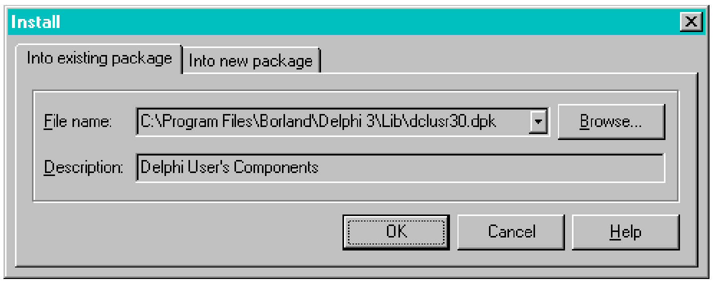
Palette page: ActiveX

Unit dir name: C:\Program Files\Borland\Delphi 3\Imports

Search path: lphi 3\Bin;C:\Program Files\Borland\Delphi 3\Imports

➤ Left: Figure 5

➤ Right: Figure 6


Install

Into existing package | Into new package

File name: C:\Program Files\Borland\Delphi 3\Lib\dclusr30.dpk

Description: Delphi User's Components

type library is used. Unfortunately, most of this information is needed at compile time, so it looks like we cannot write a general ActiveForm viewer that takes a "foreign" .OCX file (now that would be neat!). Anyway, the unit to create a dynamic instance of `TWizardActiveForm` is implemented as shown in Listing 4.

Note that like any other control, we need to set the `Parent` property to `Self`, otherwise the ActiveForm wouldn't be shown inside the parent form.

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  OleCtrls, WizardX_TLB, StdCtrls;
type
 TForm1 = class(TForm)
   Button1: TButton;
   procedure Button1Click(Sender: TObject);
   procedure FormCreate(Sender: TObject);
   procedure FormDestroy(Sender: TObject);
 private
   Wizard: TWizardActiveForm;
 end;
var Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Wizard := nil;
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
  if Assigned(Wizard) then Wizard.Free
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  Wizard := TWizardActiveForm.Create(Self);
  Wizard.Parent := Self;
  Wizard.Height := 377;
end;
end.
```

➤ Listing 4

## Activate Form

Now, just suppose we already have a rather big Delphi 2 project with many forms. Can we simply recompile these as ActiveForms? Unfortunately, it's not as simple as that. An ActiveForm is a specialised OleControl with special OLE interfaces attached to it (remember the "multiple inheritance" part?). We cannot just make a form an ActiveForm, because they're at two entirely different locations in the VCL hierarchy, and besides, the project types are different (plain executable for plain forms and an ActiveX library for ActiveForms). What can we do?

One solution for converting plain forms into ActiveForms is to create a component template from an existing form and then drop it onto an ActiveForm. That way, all the events are taken over without manually having to type them in again. Hmmm, remember the `TWizardTemplate`? Sure sounds like we've been doing exactly this since last issue... (I have to admit, this was a setup from the beginning!).

So, from any big project, just open up a form, select the menu item `Edit | Select all`, then choose `Component | Create Component Template`. It's easiest to give each component template a name which reflects the name of the form being converted. After you've converted all your forms into component templates, just close your old project and start a new ActiveX library (by creating the first new ActiveForm). For each form to convert, create a new ActiveForm and drop one of the component templates on that ActiveForm. It may be a bit of a mess at first, but it sure is one way to convert all your plain Forms into ActiveForms.

Another solution, which I've been using a lot on our local intranet, is just to use one "real" ActiveForm as the introduction to an ActiveForm application and use the plain forms from the old Delphi 2 project just as they are, as plain forms. This way, we can even use two project wrappers (one plain EXE and one ActiveX library) that both use the same "plain" forms, and just have one set of code to maintain! The introductory ActiveForm is displayed in the web browser as usual and can pop-up (with `ShowModal`) the plain forms as forms. It's really strange at first to see your whole application come alive from a web browser, but once you realise that the entire application is in fact inside this ActiveX library, with the ActiveForm just acting as a big Splash Screen, it's no big deal any more. At least in an intranet environment, people may not be scared too much by forms popping up from the browser (in fact, most of the people who saw this were quite amazed and wanted to make an ActiveForm themselves). On the (outside) internet, I'm not so sure. On the one hand, a real form will illustrate the danger of ActiveX controls so much better: a real application can do anything, whilst a web browser "feels" reasonably safe. Yet, I'm more than a bit willing to believe that the average internet user would not like applications popping up from their web browser...

## Lessons

So, what have we learned today? A lot, I'd say. First of all, it's easy to write an ActiveForm. Second, an ActiveForm is just an ActiveX, but one that can contain multiple sub-controls. Third, we can deploy an ActiveForm just like an ActiveX on a Win32 web browser, like Microsoft Internet Explorer 3 or Netscape Navigator 4, or in a development environment such as Visual Basic. Fourth, it's not so easy to convert a project consisting of many forms to an ActiveForm project, although Component Templates help a lot.

Let's not forget another important reason why ActiveForms are best suited to an intranet: database support. Since an ActiveForm is running on the web browser on the client machine, it can't do any database stuff on the server (without either making an ODBC connection or using the `TClientDataSet`, which isn't easy to do). To do local database stuff, the Borland Database Engine (BDE) must be installed on the client machine. Now, how many client machines on the internet would have the BDE installed? Not that many, I think. But, how many on your company intranet? Almost everyone at my company has the BDE installed. Of course, another option is to use a database engine which can be built into an executable, such as TurboPower's new Flash Filer.

## Next Time

Now that we've seen what ActiveForms are, let's return to one of the issues mentioned often in this column. A hot topic, first demonstrated by none other than Anders Hejlsberg last year at the first public demo of Delphi 97, which we now know as Delphi 3. I'm talking about packages, of course. The Holy Grail? Or Pandora's box? We'll find out next month. Stay tuned, and until Bill Gates becomes CEO of Borland, make mine Delphi! *["until"?!! What do you know that we don't, Bob...? Editor]*.

Bob Swart (home.pi.net/~drbob/), aka Dr.Bob, is a professional knowledge engineer technical consultant using Delphi and C++Builder for Bolesian, freelance technical author and co-author of *The Revolutionary Guide to Delphi 2*. Bob is now co-working on *Delphi Internet Solutions*, a new book about Delphi and the internet/intranet. In his spare time, He likes to watch video tapes of *Star Trek Voyager* and *Deep Space Nine* with his 3 year old son Erik Mark Pascal and his 7 month old daughter Natasha Louise Delphine.