Note: This article previously appeared in the February, 1998 issue of *Delphi Informant*

What's in the Package?

Design Time & Run Time Packages in Delphi 3

by Adam Chace

Packages offer Delphi developers a new way to deploy applications with run time libraries. Forget the hassles of trying to incorporate VCL objects into standard DLLs, with Delphi 3's new packages feature, deploying shared libraries of Delphi units and components is a cinch, and you won't have to change a single line of code. This article will provide an overview of what packages are and how you use them.

Like standard DLLs, packages are essentially just bundles of code which are referenced by a given application, be it an EXE, an ActiveX control, a DLL, or even another package. Borland has given packages a DPL extension to differentiate them from common DLLs, but their architecture is almost identical to any implicitly linked DLL. These libraries are used during design by the IDE and can then be incorporated at run time by your application. The DPL file isn't the only file associated with a package though. There's also the DCP which is the just a single file collection of all the DCUs your package contains, and the DPK which is the editor file which specifies which PAS files a particular package contains and which DPL files a package requires. The DCP file is only of interest if a package is distributed without source code or the associated DCUs. In these cases, the DCP file is used when you compile your package. The DPK file is used any time you edit a package, as we'll soon see, and is modified every time you add or remove contents in a packages.

The New Component Palette Model

Although you may not know it, if you've developed in Delphi 3, you're already using this new feature. That's because the VCL model has been replaced with a new one based entirely on design time packages. Design time packages are the new way of installing and removing components in Delphi. The Delphi 3 component palette is made up entirely of design time packages, which in turn are collections of Delphi components and units. To configure the palette, select Project|Options and click on the Packages tab to display the dialog box shown in Figure 1.

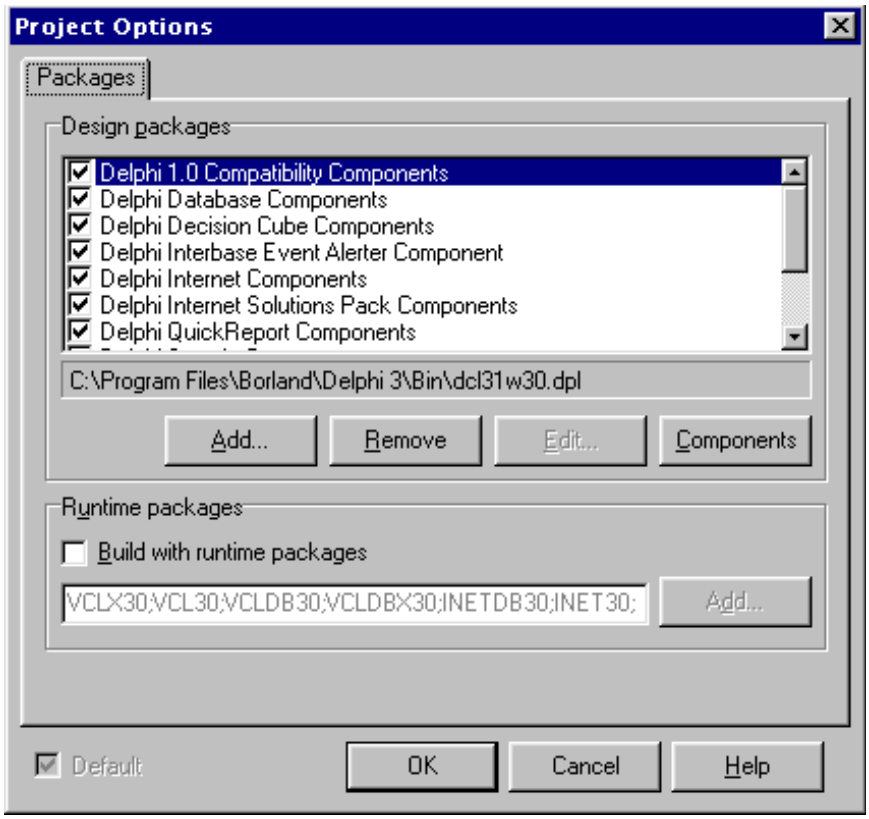

Figure 1: Adding and Removing Design time packages

From this dialog, you can load and unload packages of components without having to recompile your entire component palette, unlike in previous Delphi versions. This is a major time saver, especially if you are doing a lot of switching between projects. You can also get a quick snapshot of what components a particular package contains by clicking on the Components button. Adding or removing packages is just a matter of selecting the DPL file you want by checking or unchecking the box and clicking Add or Remove. Once you add a new package, Delphi pops up the message box shown in Figure 2 telling you whether the package was successfully installed and if so, what components were added as a result.

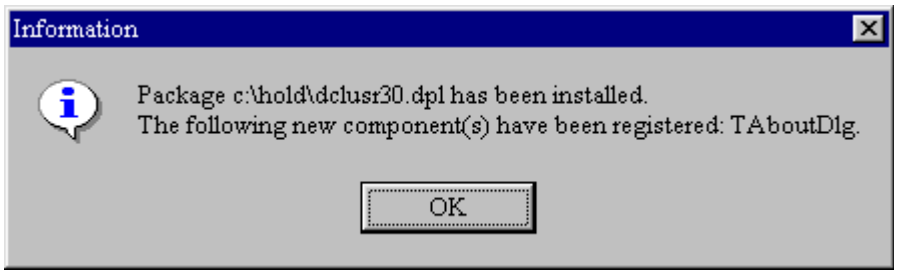

Figure 2: Delphi tells what components the package has installed

We can also edit any packages we have the DPK file for from this dialog. If we highlight a package and click Edit, we'll now see a dialog which allows us to specify which components and units this package contains, as well as any other packages that are required by it.

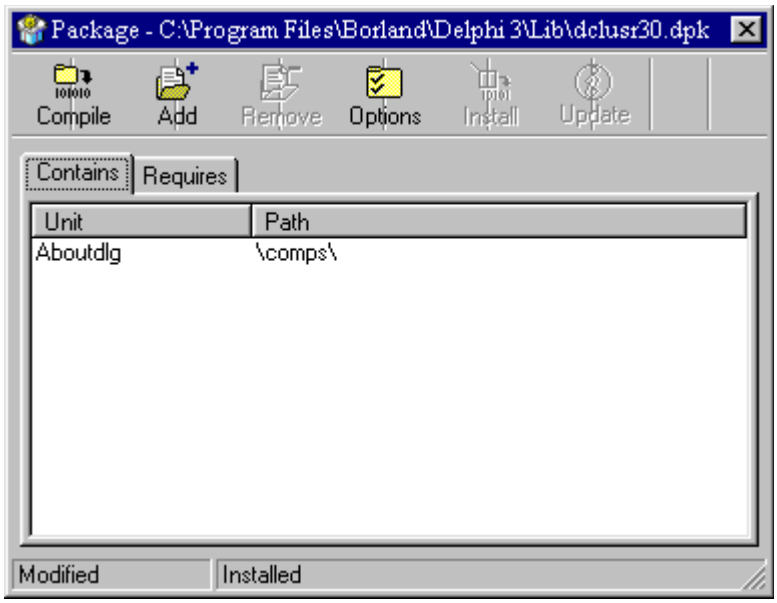

Figure 4: Editing a packages contents

This form also gives us access to the package options which allow us to specify whether this package is a runtime, design time package, or both, the package description, compiler information about the package, directory information and other pertinent details about this particular package.

Using Non Package Based Components in Delphi 3

Soon most of your 3rd party components will be distributed in the form of design time packages. But what about your old components, or newer components that aren't in packages? Can you still use these in Delphi 3? The answer is, of course, yes. Borland has provided us with an empty design time package called DCLUSR30.DPL. Any non package based component can be placed in it. To place a component into this package, just select Components|Install and you will see the dialog shown in Figure 3.

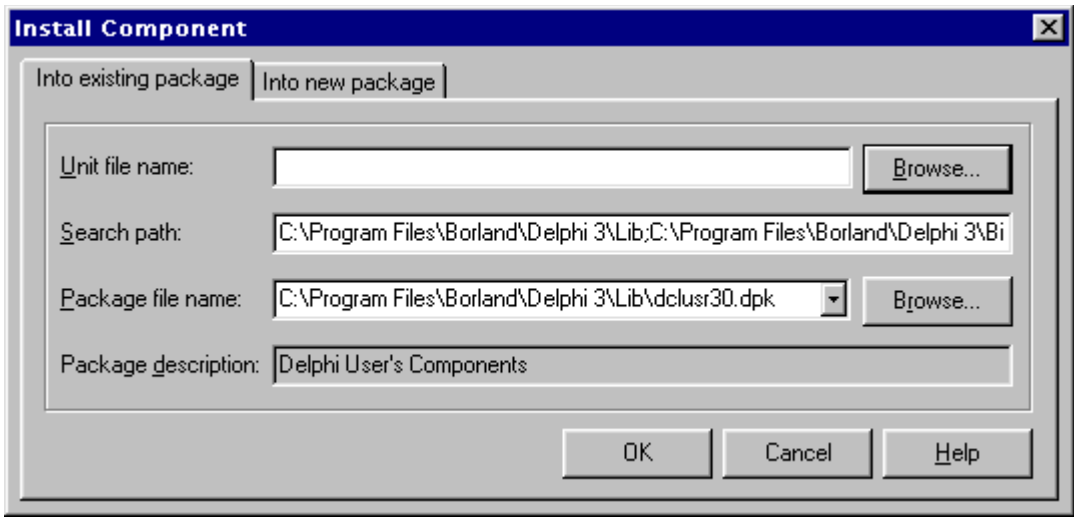

Figure 4: Installing 'loose' components

From here, all you need to do is select your .PAS or .DCU file and you're done. As long as there aren't any errors in the code, the component will install automatically as the package is loaded onto the palette. If you are adding DCUs this way you must be sure they were compiled in Delphi3 since the DCU architecture has changed again. If you don't have the source code for a component that wasn't compiled in Delphi 3 then you will be unable to install the component into a package.

Running With Packages

That about covers design time packages which, for the most part, were created to make configuring the palette quicker and easier. Now we get into the more interesting topic of run time packages. Run time packages are typically the same DPL file as their design time counterpart but are used in deployment rather than development. With a typical Delphi application, all the necessary code for components and units used in the app is compiled directly into the executable. But this can produce a large executable which can be tedious to update, especially if it is being done via modem or the Internet. Building your project with run time packages allows you to reduce your update size at the cost of a larger initial deployment. The initial delivery must include your using application (EXE, DLL etc.) and all the required run time packages. Once these packages are delivered, only the using application needs to be updated, as long as no code in the library files is changed.

Lets see an example of the impact of using run time packages. The application, PACK.EXE was built as a single form with the a TLabel, TEdit, TDataSource, TTable, TClientSocket, and a TDBListBox dropped on it. Each of these components is contained in a design time package and has a corresponding run time package (which again is usually just the same file). If we were to compile this application conventionally, we would see it has a file size of 392,192 bytes as shown in Figure 4.

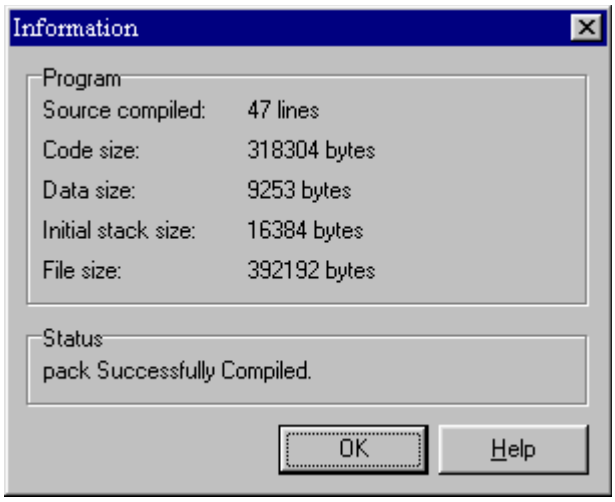

Figure 5: Our application compiled **without** using packages

Now we want to indicate that we will be distributing this application not as a single executable, but rather as an executable with run time packages. To do so, we select Project|Options and click the 'Build with runtime packages' checkbox. This enables the Project Options dialog box shown in Figure 5 and we can now, if desired, add or remove any run time packages from this list that we choose. Why would we want to remove a package name from this list at this point? Say for instance that we are using a single function in a single unit that is included in a very large package. We may want to just compile the code into the executable rather than deploying a sizable file for just one function. We may also know that we will be updating a particular component a great deal for this application. Since we ideally want components and code in packages to be static, we would avoid having to constantly update this DPL file and the executable by removing the runtime package it is contained in from this list.

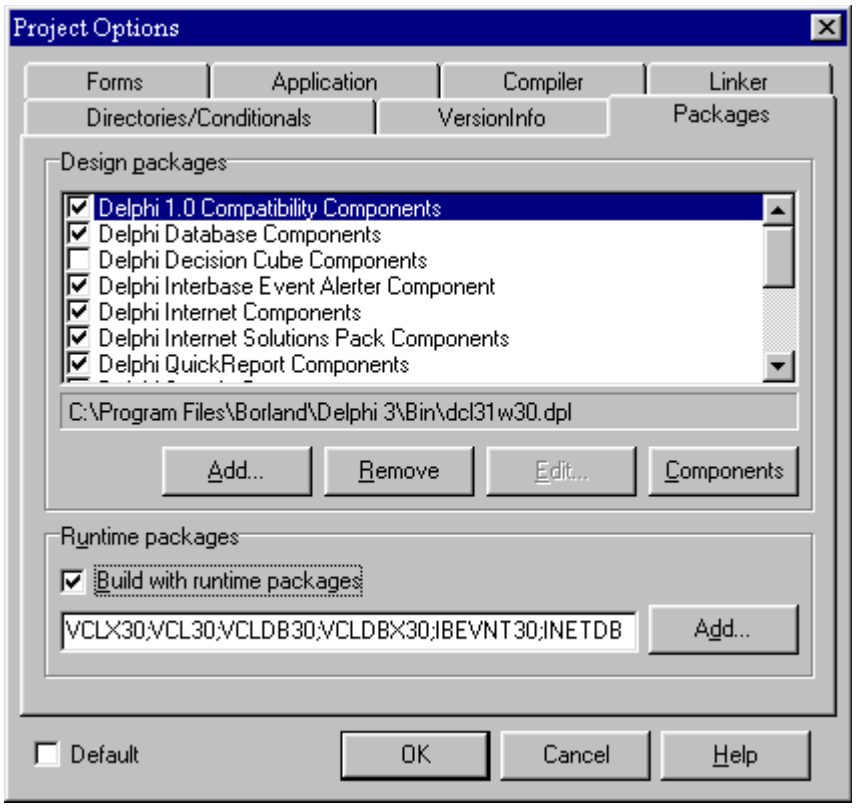

Figure 6: Choosing to build with packages

Once you've decided if you want to remove any files from this list, you click OK and build the application. It's important to note, even though a runtime package appears in the list, it will only be required by the executable if it uses code contained in that package. It isn't necessary to go through this list and remove those that you don't think you need. The compiler takes care of referencing the packages used for you. So once we click build we see in Figure 6 that the application is now only 13,312 bytes!

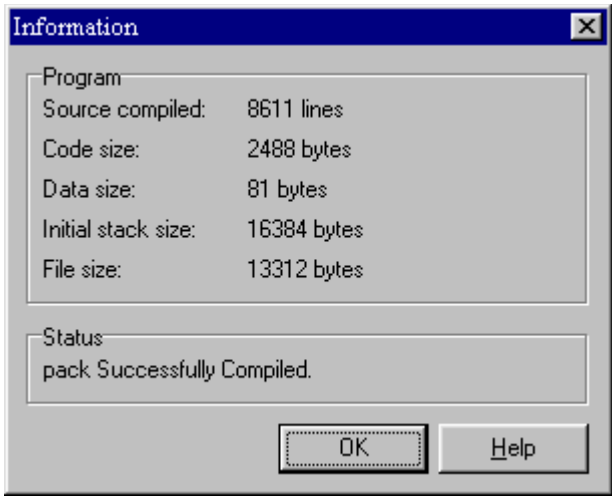

Figure 7: The same application compiled **with** runtime packages

The next step is to determine which run time packages need to be distributed with this application. We can do this one of several ways.

- Determine which design time packages are used by your application and create a list of their corresponding run time packages.
- Examine the application with an editor like Windows QuickView and note the DPL files that are referenced in the file.
- If it is an EXE or a DLL, use a freeware tool like PFinder to obtain the list.

Since we have built a simple Windows executable, we can use Pfinder to create the list, as shown in Figure 8. This free utility is available from Apogee Information Systems, Inc. and can be downloaded from www.apogeeis.com/delphi.

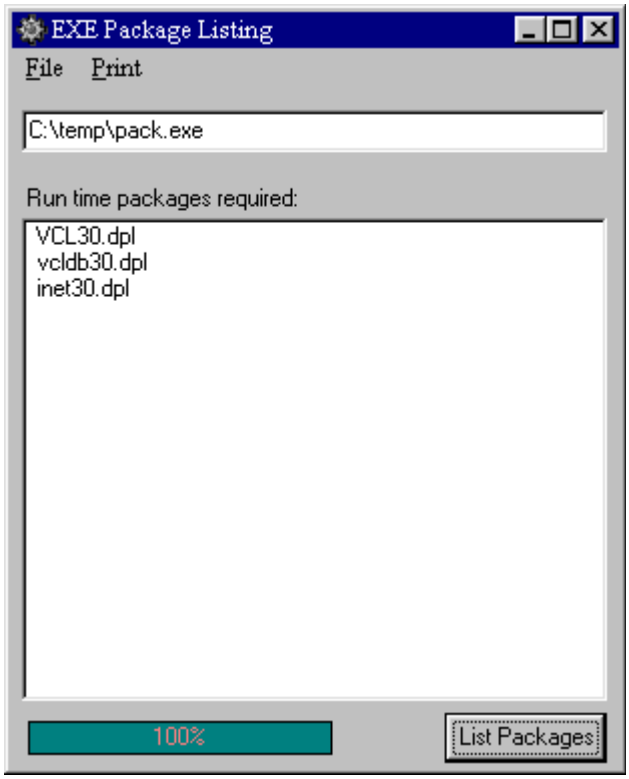

Figure 8: Pfinder package listing utility

Now we just need to deploy VCL30.DPL, VCLDB30.DBPL and INET30.DPL with our executable once. After that, as the project changes, we can just update our 13K executable. This is especially helpful during the testing phase of a project where you may be distributing new builds of an application daily. Another benefit of using runtime packages is, like standard DLL's, a user machine only needs one copy of any one package. So if a project will be distributed as a suite of using applications you can save a

lot of space by only deploying the components they share once in a runtime package, rather than redundantly compiling the component code into each using program.

That's all there is to it. Runtime packages are an easy way to reduce your deployment costs and you don't need to make any coding changes whatsover. In fact, you can decide to deploy with runtime packages as late as your very last build without having to be concerned with the issues of conventional DLLs.

Conclusion

As we have seen, packages are an exciting new feature for Delphi programmers to exploit. While design time packages streamline the use of the component library, run time packages provide the real power. Delphi programmers can now easily segment and distribute portions of functionality independent of the .EXE file. This significantly reduces the amount of effort involved with distributing application updates.

Biography: Adam Chace is an Application Developer with Apogee Information Systems, an Inprise Premier Partner specializing in multi-tier Delphi solutions. He presented "Packages in Delphi 3" at the 1997 Borland Conference in Nashville, TN and is Delphi 3 Client/Server certified. This is his first published article. You can reach him at achace@apogeeis.com.