Migrating Borland® Delphi™ applications to the Microsoft® .NET Framework with Delphi 8

A Borland White Paper

By Bob Swart (aka Dr.Bob),
Bob Swart Training & Consultancy (http://www.drbob42.com)

February 2004

Borland®

Contents

Introduction

With the release of Delphi™ 8 for the Microsoft® .NET Framework (a.k.a. Delphi for .NET), Borland has enabled Delphi developers to target another new platform, supporting the needs of its developer base. Previous versions of Delphi can produce Microsoft® Win32® applications (and with Borland® Kylix,™ we can build Linux® applications using the Delphi language).

Delphi for .NET enables developers to write native .NET applications using Windows® Forms or Web Forms as the framework, or using VCL for .NET components.

This paper discusses the migration of Delphi applications for Win32 to the Microsoft .NET Framework using Delphi 8 for the Microsoft .NET Framework. The difference between Windows Forms and VCL for .NET is covered, as well as several sample migrations from existing Delphi Win32 VCL applications to Delphi for .NET native .NET applications.

Delphi™ 7 to Delphi™ for the Microsoft® .NET Framework

Using Delphi for the Microsoft .NET Framework, we can compile applications that were made in Delphi 7 or previous versions. The Delphi 8 box also includes Delphi 7 to produce Win32 applications. If you want to produce source code that compiles with both Delphi 7 to a Win32 target and with Delphi for .NET to a .NET target, then you might need to use compiler IFDEFs inside your source code.

Delphi 7 contains the following compiler defines:

```
MSWINDOWS
WIN32
```

Delphi for .NET contains the following compiler defines:

```
CLR
CIL
MANAGEDCODE
```

This means that you might want to write code like the following (using the Linux® compiler define to complete the Delphi platform alternatives):

```
project HelloWorld;
{$APPTYPE CONSOLE}
begin
{$IFDEF CLR} // Delphi for .NET
writeln('Hello, .NET world!');
{$ENDIF}
{$IFDEF WIN32} // Delphi 7
writeln('Hello, Win32 world!');
{$ENDIF}
{$IFDEF LINUX} // Kylix
writeln('Hello, Linux world!');
{$ENDIF}
end.
```

Note that we now have three possible platforms, so you should not use {$ELSE} to write code that is not suited for one particular platform. Even if you are certain today that the code is right, future support for other platforms might break your code. Always use specific {$IFDEF} sections to write code for a specific platform.

VCL, VCL for .NET, and Windows® Forms

When Delphi first shipped in 1995, the component library was called the Visual Component Library. It contained more than just visual components, however. A number of these components are platform-independent, and it was mainly the visual components that were specifically bound to the Windows API and controls.

When Kylix was introduced, a new library name was used: CLX (Component Library for X-platform), which divided the components in BaseCLX, DataCLX, VisualCLX, and NetCLX. Using Delphi 6 and 7, we can build visual applications using CLX (VisualCLX is cross-platform for Linux and Win32) or VCL (only on Win32).

Now that Delphi has been ported to the Microsoft .NET Framework, the VCL has been ported to .NET as well. This means that we can not only use native Windows Forms to produce .NET applications with Delphi for .NET, but also VCL for .NET to produce .NET applications. Because VCL for .NET uses the same classes and property/events interfaces that the VCL for Win32 uses, Delphi Win32 projects can be migrated to Delphi for .NET with considerable ease, which will be demonstrated in this paper).

CLX, VCL, and VCL for .NET are similar in terms of class names, property/event names, and their usage. They all use an external stream file to place the property and event assignments: for CLX an .xfm file, for VCL a .dfm file, and for VCL for .NET an .nfm file. In contrast, the Windows Forms projects do not rely on a .nfm file, but assign all property and event handler values in source code (hence the need for code folding in the IDE).

The VCL can be seen as a wrapper around the Win32 API, and the VCL for .NET can be seen as a wrapper around the .NET Framework (or more specifically the Windows Forms classes). The move from VCL to VCL for .NET is fairly painless and involves far less work than the move from the Win32 API to the .NET Framework with Windows Forms. And the future move to Longhorn's XAML (for the new Avalon presentation layer) will also be easier when using VCL than when bound to a native layer such as the Win32 API or Windows Forms. In short, using VCL extends the lifetime of your code.

For more information about VCL, CLX, and Windows Forms, see John Kaster's article at the Borland Developer Network at http://bdn.borland.com/article/0,1410,29460,00.html .

Delphi™ 7 language and RTL not available in Delphi™ for Microsoft® .NET

Although the move from VCL to VCL for .NET is fairly painless, several migration issues are related to the differences in the Win32 and .NET platforms. These issues are related to the fact that .NET code is executed by the CLR in a safe, managed way, so all potentially unsafe

features and code constructs in Delphi 7 must be replaced by safe counterparts in Delphi for .NET.

Many Delphi 7 language features are no longer available in the Delphi for .NET environment because they are unsafe or could result in unsafe code. The following table contains the most important (most often used) of these language elements, along with suggested Delphi for .NET alternatives.

Delphi™ 7 language feature	Recommended Delphi 8 feature
Real48	Double
absolute, Addr, @	n/a
GetMem, FreeMem, ReAllocMem	New and Dispose, or array structures
the Borland Pascal "object" type	class type
Files of any type (including records)	Streams, Serialization, databases
inline assembly or the asm keyword	n/a
ExitProc	n/a
FillChar, Move	rewrite using for-loops
PChar	String or StringBuilder [1]

Table 1. *Language features*

[1] A string in .NET is not very efficient when you modify it several times (like concatenating substrings), in which case you are better off using the StringBuilder class.

Unsafe code

Delphi 7 can help you prepare Win32 applications for .NET, with a set of three new warnings. Because they are disabled by default, they must be explicitly turned on using the **Project | Options - Compiler Warnings** tab. The new set consists of warnings for unsafe types, unsafe code, and unsafe typecasts. You can either enable these warnings in project options, or—the preferred approach—specify them at the top of source files as follows:

```
{$WARN UNSAFE_TYPE ON}
{$WARN UNSAFE_CODE ON}
{$WARN UNSAFE_CAST ON}
```

You might have to add it to the top of every unit to produce the warnings.

For unsafe types, you'll be notified when you declare or use variables of type PChar, untyped pointers, File of type, Real48, variant records, or when you use untyped var or out parameters. Regarding unsafe code, you'll get warnings in Delphi 7 when you use absolute, Addr, Ptr, Hi, Lo, Swap, BlockRead, BlockWrite, GetMem, FreeMem, and ReallocMem. Finally, any typecast from a pointer or object to something that it may not be is considered worthy of a warning as well.

When you compile unsafe types, code, or casts using Delphi 7 (with the three warnings enabled), you'll get compiler warnings in the message view. Note that unsafe variables are mentioned not only when you declare them, but also at every line where you use them.

If you cannot replace the unsafe code, type, or casts with safe Delphi for .NET code, then you can mark your code as being unsafe for the time being, so that it compiles. This involves two steps: first , mark the section of code inside `{$UNSAFECODE ON} ... {$UNSAFECODE OFF}` compiler directives, and then mark the routine or method that holds the unsafe code, cast with the unsafe keyword.

As a consequence of using the unsafe keyword, the resulting application or package no longer passes PEVerify[2]. However, the unsafe keyword helps you with a first migration (of unsafe sections), which you can later rewrite using native safe .NET code.

New language features

Delphi for .NET has also introduced several new or extended language features to enhance the way it conforms to the .NET standard, such as sealed classes, final methods, and strict private and protected access specifiers. In order to avoid existing code breaking, the private and protected keywords still allow "friends" from the same source file to access the internals of their classes. To conform to the .NET standard, which specifies that private is closed for anyone except the class instance itself, and protected is open only for the class (instance) itself or its descendants, the keyword “strict” should be used before private and protected. This keyword is not supported by Delphi 7 (and neither are sealed classes or final methods), so if you use them, your source code is usable only with Delphi for .NET (until an update or new version of the Win32 Delphi environment is released with support for the new language features).

Delphi™ 7 VCL components not in Delphi™ for the Microsoft® .NET Framework

A number of Delphi 7 VCL components are not present in the VCL for .NET shipping with Delphi for .NET. The next section discusses the VCL for .NET details on a component-by-component basis. The following categories are no longer available in VCL for .NET: dbGo for ADO, WebBroker, InternetExpress, WebSnap, and XML support in the form of TXMLDocument, XML Data Binding, and the XML Mapper with the associated TXMLTransform components.

[2] PEVerify is a .NET Framework SDK utility that can verify whether or not the code in a .NET assembly or executable manipulates data in inappropriate ways that could corrupt data or compromise system security. Only 100% verifiable safe binaries pass the PEVerify test.

VCL to VCL for .NET

Most Delphi 7 VCL components appear in the VCL for .NET component set that is included with Delphi for .NET. The Component Palette is replaced by the Tool Palette, but similar categories exist: Standard, Additional, Win32, System, Win 3.1, Dialogs, Data Access, Data Controls, dbExpress, DataSnap, BDE, InterBase, InterBase Admin, Indy Clients, Indy I/O Handlers, Indy Intercepts, and Indy Misc (see **Figure 1**).

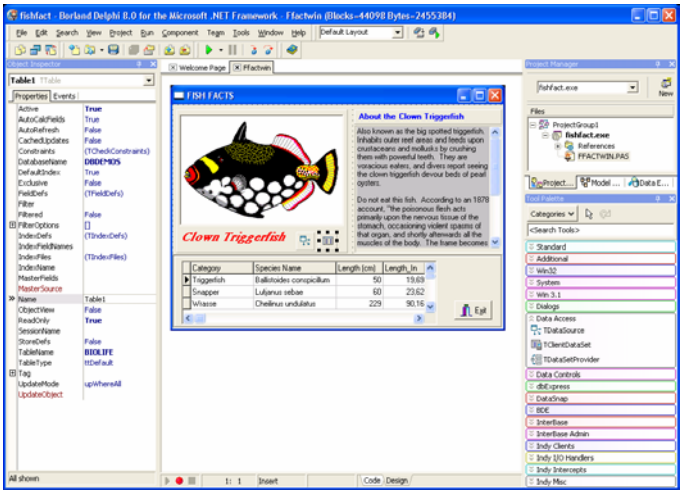

Figure 1. *Delphi for .NET IDE with VCL for .NET component categories*

Rather than list components available in VCL for .NET, the following is a list of components that are part of the Delphi 7 VCL but are not available in the VCL for .NET of Delphi for .NET.

VCL applications

All components from the **Standard** tab of the VCL appear in VCL for .NET. Missing from the **Additional** tabare TChart, TActionManager, TActionMainMenuBar, TActionToolBar, TXPColorMap, TStandardColorMap, TtwilightColorMap, and TCustomizeDlg components. Further investigation shows that TActionManager is listed in the help and in the Borland.Vcl.ActnMan namespace, but not in the Tool Palette. Also note that a free VCL for

.NET version of TeeChart "Standard" will be available at the Steema Web site at http://www.steema.com/.

From the **Win32** tab, all components appear in VCL for .NET. Missing from the **System** tab are OleContainer, DdeClientConv, DdeClientItem, DdeServerConv, and DdeServerItem components. Even the **Win 3.1** tab from VCL is present in VCL for .NET, with the exception of the TDBLookupList and TDBLookupCombo components. Finally, the **Dialogs** tab is completely present in VCL for .NET.

Based on these components, we can pick a number of the standard sample applications that ship with Delphi 7 and open them with Delphi for .NET.

Ownerlist

We can start with the sample application in the Delphi7\Demos\Ownerlist directory, consisting of four files: FontDraw.dpr, FontDraw.res, FontList.pas, and FontList.dfm. Delphi for .NET can open .bdsproj files (the Delphi for .NET project files) as well as Win32-style .dpr project files. If you open FontDraw.dpr in the Delphi for .NET IDE, you can immediately compile the project to a native .NET executable. You might notice warnings, which are mainly platform-specific (caused because the VCL is based on the Windows platform). But these warnings are nothing to worry about; the resulting application is still a native .NET executable, as can be seen in **Figure 2**:

Figure 2: *OwnerDraw sample application for .NET*

Without a single change in the source code, we can migrate this sample project from Delphi 7 to Delphi for .NET. The new executable is a safe executable, which can be proved by running it through PEVerify without errors. When you close the project, a FontDraw.bdsproj is generated, and some configuration settings are written to the FontDraw.cfg file. Fortunately, these new settings do not prevent Delphi 7 from being able to compile the same project.

One change made by Delphi for .NET to the main unit is the addition of the System.ComponentModel unit to the uses clause of the interface section. Slightly modify this uses clause if you want to keep a single-source cross-platform project. Place the System.ComponentModel unit in a `{$IFDEF CLR}` section, like this:

```
uses

Windows, Classes, Graphics, Forms, Controls,
{$IFDEF CLR} System.ComponentModel, {$ENDIF}
StdCtrls;
```

This single change is something you must perform for all units migrating from Delphi 7 to Delphi for .NET, for which you want to enable compatibility with Delphi 7 (to produce a Win32 executable as well as a .NET executable from the same project source code).

ConvertIt

The Ownerlist sample application worked easily and took only one manual step. Let's take another example, the first one used to demonstrate the capabilities of the Delphi for .NET preview command-line compiler: ConvertIt. The Demos\ConvertIt directory contains five files: ConvertIt.dpr and ConvertIt.res, ConvertItUnit.pas and ConverItUnit.dfm, and a EuroConv.pas unit.

This project also loads immediately in the Delphi for .NET IDE, and results in another native .NET executable (**Figure 3**).

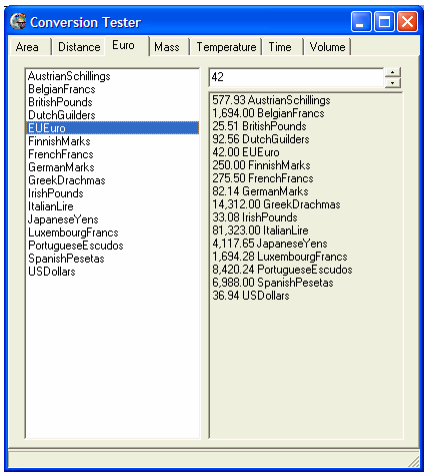

Figure 3: *ConvertIt sample application for .NET*

AppEvents

Let's end the VCL sample applications with a more complex application in the Delphi7\Demos\AppEvents directory. Again, this sample application works as expected when loaded in the Delphi for .NET IDE and run as a native .NET application (**Figure 4**).

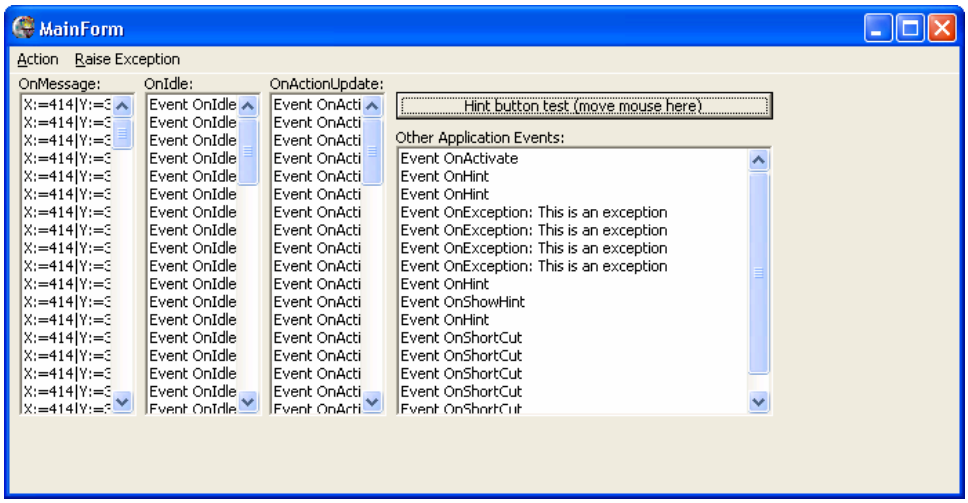

Figure 4: *AppEvents sample application for .NET*

VCL for .NET deployment

Before we move on to sample applications with database support, let's look at deployment of a Delphi for .NET application—specifically, a VCL for .NET application. If we take the last sample application, and look inside the Project Manager, the reference node of the project lists only the System.Drawing.dll assembly. This is the only assembly the AppEvents sample application for .NET requires. All VCL for .NET units are compiled into the executable, which as a consequence is about 1.5 MB.

On the bright side, you need to deploy only the AppEvents executable (the System.Drawing.dll assembly exists on any Microsoft .NET Framework installation), and no additional VCL for .NET assemblies. In some situations, however, it might be more desirable to deploy a smaller AppEvents executable and rely on VCL for .NET functionality in VCL for .NET assemblies that are already (or at the same time) deployed on the target machine. In that respect, .NET assemblies can be seen as runtime packages. Use this functionality when the project is modified frequently and the distribution of a small updated executable is more efficient than the distribution of a larger monolithic execution.

The developer must choose, but the default "setting" for new VCL for .NET applications is to compile executables without linking in the VCL for .NET assemblies (in other words: small executables that need the VCL for .NET assemblies to be deployed as well). When migrating VCL projects to Delphi for .NET, however, the IDE will not add the VCL for .NET assemblies to the list of references, and as a result the VCL for .NET units will be compiled into a monolithic executable.

In order to change a migrated VCL for .NET project, manually add the VCL for .NET assemblies as references to the project (specifically the Borland.Delphi.dll and Borland.Vcl.dll), and recompile the project. This results in an AppEvents sample application for .NET of only 12 KB, albeit one that requires the Borland.Delphi.dll and Borland.Vcl.dll assemblies to be deployed alongside.

Next, we focus on more-difficult applications with database support.

Database applications

Many new powerful technologies in the Microsoft .NET Framework are available to developers. Some of these, such as ADO.NET, make current Win32 technologies either unnecessary or obsolete. This section describes the data access technologies offered by Delphi 7 and explains whether and how they are available to use in VCL for .NET applications with Delphi for .NET.

The following table gives an overview of the available data access technologies in Delphi 7, and lists the VCL for .NET counterparts in Delphi for .NET:

Delphi™ 7	Delphi 8
Borland® Database Engine (BDE) (dBASE, Paradox®)	BDE (dBASE, Paradox)
SQL Links	Deprecated
Borland® dbExpress (InterBase,® Microsoft® SQL Server,® Oracle,® IBM® DB2,® Informix®)	dbExpress (InterBase, SQL Server, Oracle, IBM DB2, Informix, SQL Anywhere®)
Borland® IBExpress™ (IBX)	IBExpress (IBX)
Borland® dbGo™ for ADO	not available at this time

Table 2: *Data-access technologies*

Apart from dbGo™ for ADO, which was not ported to .NET, we have the choice of BDE (SQL Links is deprecated (see http://bdn.borland.com/article/0,1410,28688,00.html) but local BDE for dBASE and Paradox tables is still present), dbExpress, and InterBase Express (IBX).

The sample applications from Delphi 7 that illustrate this, and are migrated to Delphi for .NET with little to no modifications, are Demos\Db\FishFact (BDE) and Demos\Frames\Db (BDE). Apart from these two sample applications, we'll build a small dbExpress application in Delphi 7 and move it over to Delphi for .NET.

Also, for reporting purposes, Rave Reports® is available with Delphi for .NET.

One category of VCL components that wasn't migrated to VCL for .NET is the Decision Cube. Because the source code is included with Delphi 7 (at least in the Enterprise edition), you can attempt to migrate these components yourself if you desperately need them

Data Access components

In the Data Access category, the TXMLTransform, TXMLTransformProvider, and TXMLTransformClient components are not included with Delphi for .NET. Support for XML in .NET can be found in the System.Xml namespace.

The Data Controls category is complete, with the exception of the TDBChart component.

Although the BDE is still supported in .NET, this only covers the local table components TTable, TQuery, TDatabase, TSession, and TBatchMove, and not the SQL Links specific components TStoredProc, TUpdateSQL, TNestedTable (which are therefore not available in the Tool Palette, although they can be found in the VCL for .NET unit Borland.Vcl.DBTables.pas).

In the dbExpress category, components from VCL are present in VCL for .NET with the exception of TSimpleDataSet.

The InterBase (for InterBase Express) and InterBase Admin categories are complete, with the exception of the TIBEvents component from the **InterBase** tab, and the TIBInstall and TIBUninstall components from the **InterBase Admin** tab.

The DataSnap category in VCL for .NET contains only the TDCOMConnection component, and not TSocketConnection, TSimpleObjectBroker, TWebConnection, TConnectionBroker, TSharedConnection, and TLocalConnection. Note that TConnectionBroker can be found in Borland.Vcl.DBClient.pas, TSharedConnection in Borland.Vcl.MConnect.pas, and TLocalConnection in Borland.Vcl.TConnect.pas.

Using the TDCOMConnection component in Delphi for .NET, developers can build DataSnap clients connecting to Win32 Delphi DataSnap servers, which is another way the Win32 and .NET worlds are bridged.

FishFact (BDE)

The FishFact sample application, using a local BDE Paradox table, available even in Delphi 1, can be opened in Delphi for .NET and compiled without problems. The resulting.NET sample application can be seen in **Figure 5**. This 16-bit project can be compiled with Delphi for .NET to a native .NET executable without modifications!

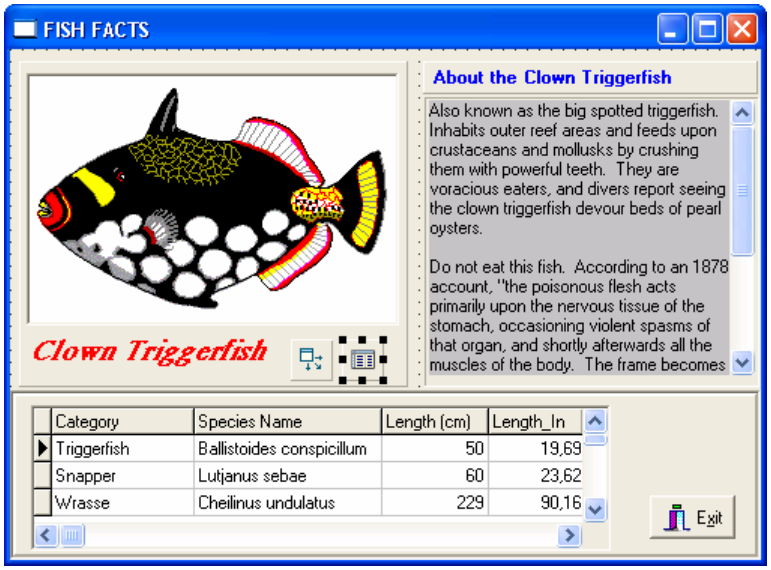

Figure 5: *FishFact BDE sample application for .NET*

When migrating BDE applications with Delphi for .NET to the .NET Framework, you must be aware that the underlying data access architecture is still the Win32 version of the BDE itself. So when it comes to deploying the VCL for .NET application, you must deploy the BDE with it.

Frames\Db (Frames and BDE)

The Frames\Db sample application illustrates the use of BDE tables in combination with the support for Frames in VCL for .NET. Frames, an important feature of VCL for .NET, enable developers to design frames as reusable "jigsaw" GUI elements, each consisting of controls to produce a consistent, reusable, and easily maintainable collection of screen elements.

Delphi 7 projects that rely on frames migrate to Delphi for .NET without problems, as you can see in **Figure 6** with the BDE frames sample application.

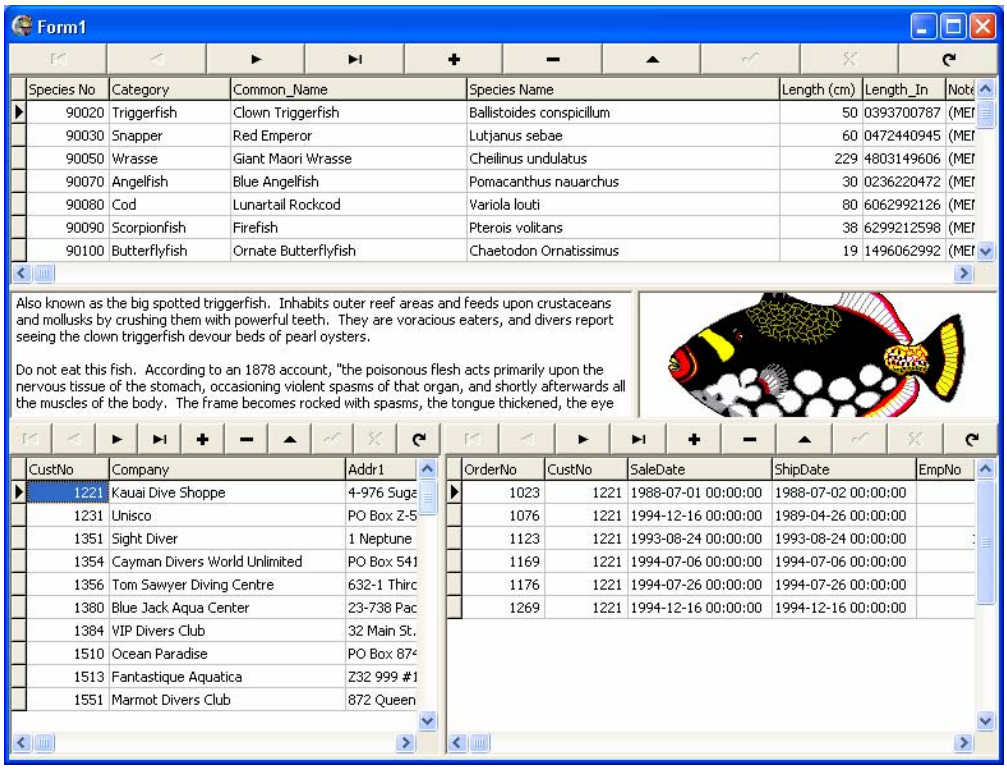

Figure 6: BDE *FishFact and Frames sample application for .NET*

The BDE can be used in VCL for .NET applications to work with local dBASE and Paradox® files. In order to work with database management systems (DBMSs) such as InterBase,® Oracle,® Microsoft® SQL Server, IBM® DB2,® or Informix,® you need to use a different data-access technology. For InterBase, the choice can be IBExpress (IBX) or dbExpress, but for others, the only VCL for .NET data access technology available is dbExpress.

dbExpress

Many dbExpress sample applications shipping with Delphi 7 are based on CLX for cross-platform compatibility between Delphi 7 and Kylix 3. This means some uses clauses must be

changed (a bit more work compared to VCL applications), and that we end with an application using IFDEFs that can be compiled with Delphi 7, Delphi for .NET, and Kylix.

In order to build a new dbExpress application for Win32, start Delphi 7 and create a new VCL application. Add a TSQLConnection component from the **dbExpress** tab of the Component Palette onto the form. Right-click on this component to start the Connections Editor, and edit the settings to connect to the IBLocal InterBase database (username sysdba, password masterkey). Place a TSQLDataSet component and point the SQLConnection property to the TSQLConnection component. Now use the Query Editor to specify the following SQL statement for the CommandText property:

```
SELECT * FROM EMPLOYEE
```

Next, add a TDataSetProvider component on the form, and point its DataSet property to the TSQLDataSet component. Place a TClientDataSet component on the form and point its ProviderName property to the TDataSetProvider component. Finally, add a TDataSource component and point its DataSet property to the TClientDataSet component.

We can now use data-aware controls, such as the TDBGrid component. Make sure the DataSource property is pointing to the TDataSource component. You can get live data at design-time if you open the ClientDataSet (by setting Active to True).

In order to ensure that the changes in the DBGrid are posted back to the InterBase table, write one line of code in the OnAfterPost and OnAfterDelete event handlers, namely:

```
procedure TForm1.ClientDataSet1AfterPostOrDelete(DataSet:
TDataSet);
begin
  (DataSet as TClientDataSet).ApplyUpdates(0)
end;
```

Now, we can compile and run the application. After you've saved the project in Delphi 7, you can open the project in Delphi for .NET and compile it to a native .NET executable without errors or warnings.

For a more complex sample application, you can use a VCL data module, which also migrates to VCL for .NET without problems. Like Frames, Data Modules offer VCL for .NET developers a powerful means to group and manage components that belong together within a single container (either the frame or the data module).

Web applications

Many powerful new technologies are available within the Microsoft .NET Framework. Some of these new technologies, such as ASP.NET, make current Win32 technologies either unnecessary or obsolete. This section discusses the technologies available to build Web server applications offered by Delphi 7, and explains if and how they are available to use in VCL for .NET applications with Delphi for .NET.

The ASP.NET technology of the .NET Framework enables developers to build Web server applications that can be visually designed and have the deployment ease of a CGI executable, while retaining the speed and efficiency of an ISAPI DLL. This means that the need to migrate WebBroker, InternetExpress, or WebSnap applications to the .NET world is expected to be nonexistent. Maintain those web server projects in Delphi 7, and start new development using Delphi for .NET and ASP.NET.

For more information about ASP.NET development with Delphi for .NET, see the BDNtv Flash movie at http://bdn.borland.com/article/0,1410,31890,00.html , which demonstrates the development of an ASP.NET Web application using the Borland Data Provider for InterBase and the Borland® DB Web Controls. This BDNtv movie also shows how easy it can be to migrate a Delphi Win32 application to the Microsoft .NET Framework with Delphi for .NET

Delphi 7 included IntraWeb 5, a third-party tool, which is migrated to .NET and available as IntraWeb for .NET. Existing IntraWeb applications can be expected to migrate to IntraWeb for .NET with few to no problems. Note, however, that IntraWeb for .NET is not included with Delphi for .NET.

The Indy tabs of VCL are all accounted for in VCL for .NET, so VCL applications that use the Indy components should migrate over to VCL for .NET without problems.

Web Services

Delphi 6 introduced support for building and consuming Web Services using Borland's implementation of the SOAP protocol, which is also used in Kylix 2 and higher (and C++Builder 6). The .NET Framework has support for SOAP and Web Services built in using the ASP.NET technology. As a result, Delphi for .NET makes use of the native .NET functionality to build ASP.NET Web Services and consume Web Services.

Delphi for .NET on-line help contains two sections entitled "Porting Web Service Clients to Delphi 8 for .NET" and "Porting a Web Service Client Application from Delphi 7 to Delphi 8 for .NET" that can be read for more information about migrating Web Service client applications from Delphi 7 to Delphi for .NET.

Miscellaneous

Three VCL component categories from Delphi 7 that are not immediately available in Delphi for .NET are the **ActiveX**, **COM+**, and **Servers** tabs. Although these components are not installed by default, you can import COM and ActiveX components by adding them as reference to your project, as you can see in the dialog box in **Figure 7**.

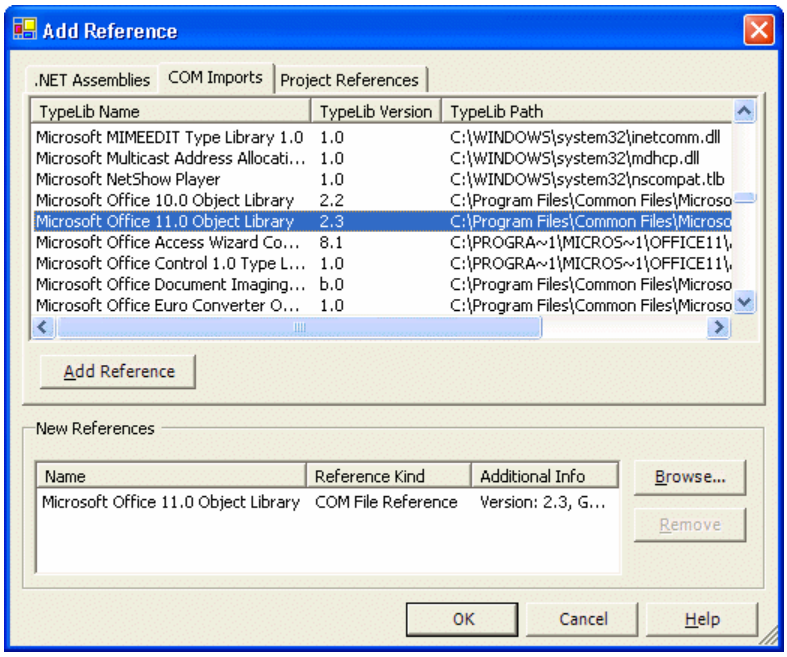

Figure 7: *COM imports*

Note that this means that your managed, safe .NET executable will use COM Interop to use an unmanaged (and potentially unsafe) COM object. This solution can be used for migration purposes, but in the long run aim for a fully managed, safe .NET executable.

Finally, although the Samples category isn't listed in the Tool Palette of Delphi for .NET, the components can be found in the Demos\Vcl\Samples directory, including the Borland.Vcl.Samples package.

If that's not enough, you can also use standard .NET controls in VCL for .NET applications. Read an article about doing so at http://bdn.borland.com/article/0,1410,31886,00.html .

Summary

From VCL you can migrate to VCL for .NET with considerable ease. For new .NET applications, you can choose between VCL for .NET and Windows Forms. Using VCL for

.NET, you can use the BDE, dbExpress, IBX, or DataSnap components. Using Windows Forms, you can choose ADO.NET or Borland Data Provider for .NET components.

For Web development, WebBroker, InternetExpress, and WebSnap technologies are replaced by the ASP.NET framework, which supports ASP.NET Web forms and Web Services. IntraWeb also has migrated to .NET as a third-party solution.

References

"Delphi 8 ASP.NET Development, and Win32 migration," John Kaster (presented by Troy Kitch)
http://bdn.borland.com/article/0,1410,31890,00.html

"Using standard .NET controls in VCL .NET applications with Delphi 8," Tim Jarvis and John Kaster
http://bdn.borland.com/article/0,1410,31886,00.html

"Overview of the VCL for .NET," John Kaster
http://bdn.borland.com/article/0,1410,29460,00.html

"The Future of the Borland Database Engine (BDE) and SQL Links," John Kaster
http://bdn.borland.com/article/0,1410,28688,00.html