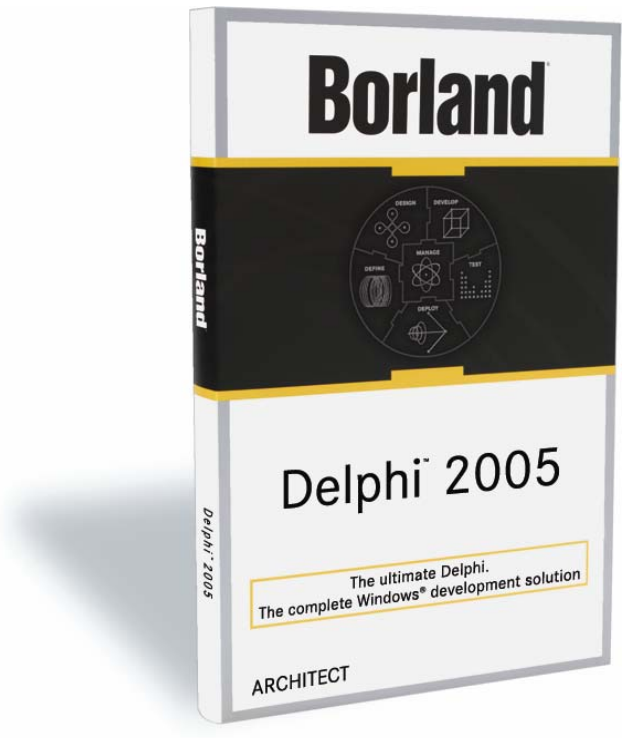
Borland® Delphi™ 2005 Reviewer's Guide

The Complete Windows® Development Solution

Produced for Borland by Cary Jensen, Jensen Data Systems, Inc.
October 2004

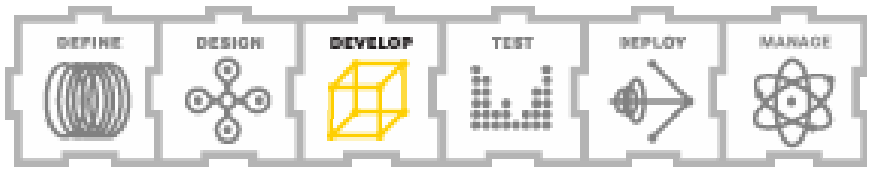


Borland®

Contents

Overview

Borland®
Delphi™ 2005



Overview

Welcome to the Delphi 2005 Reviewer's Guide. This document will familiarize you with Delphi 2005, the newest version of Borland's flagship development environment, culminating more than twenty years of technological innovation.

The Delphi 2005 Reviewer's Guide is organized into two parts. In this first part, the Overview, you will find a general introduction to Borland Delphi 2005.

The second part of this guide takes you on a tour of Delphi 2005. It is organized by the major areas of software development and support in Delphi 2005, providing you with an overview of each area, and is followed by a description of the many updates, enhancements, and additions introduced in this release. If you are already a Borland enthusiast, you may want to quickly scan the overview section, concentrating instead on the updates that make this the most important upgrade to Delphi since it debuted in 1995.

Delphi: Advancing the Art of Software Development

Delphi's legacy began in 1983, when Turbo Pascal set a new standard for software engineering. The evolution of Turbo Pascal reads like a history lesson in the advancement of software development, including the introduction of such groundbreaking innovations as an integrated development environment (IDE), integrated debugging, syntax-highlighting, a powerful object-oriented programming (OOP) model, and OWL, the Object Windows Library.

With the release of Delphi 1.0 in February of 1995, Borland proved that component-based development could be applied in an object-oriented environment, permitting developers to rapidly build applications while maximizing code reuse. In more ways than one, Delphi blazed a trail that would eventually be followed by the framework class library (FCL) of the Microsoft .NET Framework.

Delphi 2005 represents another impressive advance in software development by Borland, making it the ultimate and complete development solution for Windows. Delphi 2005 converges Delphi, C#, Microsoft® .NET Framework and Win32 support for graphical user interface (GUI), Web, database, and model-driven application development, and is wrapped with the essential application lifecycle management (ALM) tools into a unified, highly-productive rapid application development (RAD) environment. With Delphi 2005, you have everything you need to increase Windows developer productivity, personal developer productivity, and team productivity.

- Windows developer productivity: The Delphi 2005 IDE makes Windows development tasks faster, easier, and better by supporting the Win32 standard of yesterday and today, with the Windows-based Microsoft .NET Framework development standard of today and tomorrow. With world-class compilers and debuggers, a rich legacy of standards-based tools, and a seamless migration path between current and emerging platforms, there is no better Windows development tool on the market today.

- Personal developer productivity: Delphi 2005 takes the power of Delphi to a new level, with speed and productivity enhancements throughout. With a code editor that simplifies every aspect of your programming experience, the largest collection of reusable components, powerful code-generating wizards, and much more, Delphi 2005 is the most prolific development environment available.

- Team productivity: Delphi 2005 allows teams to take full control of the application lifecycle. In addition to state-of-the-art tools for software development, certain editions of Delphi 2005 also include StarTeam® for team source code control, Borland Enterprise Core Objects II (ECO™ II) for model-powered development in the .NET framework, integrated Unit Testing Framework, and Borland Optimizeit™ Profiler for the Microsoft .NET Framework for performance-testing. In short, Delphi 2005 provides you with a complete, integrated solution for all your development and project management needs.

Borland Products = Technical Excellence

Throughout the years, Borland products have been recognized for excellence and innovation. Here are a few of the honors received recently by the products that represent the heritage of Delphi 2005:

- Borland Delphi 8 for the Microsoft .NET Framework won Best of Show in the developer tool category at TechEd Europe, 02-July-04
- Borland C#Builder won the Visual Studio Magazine Reader Choice Award for best developer tool 24-May-04
- Borland Delphi 7 Studio won the Web Services Journal Readers' Choice Award for in the Best GUI for Web Services Product category, 25-February-04

The remaining sections of this guide are organized into related topics associated with software development. Each section begins with a general overview, and then continues with a description of the new and enhanced features introduced in Delphi 2005.

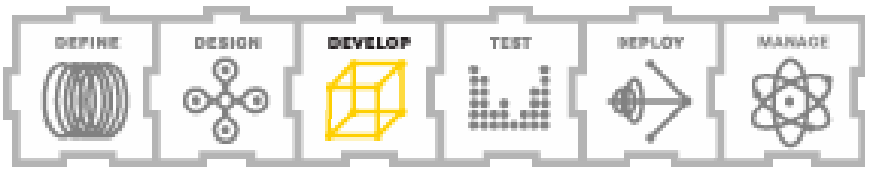
Disclaimer

This reviewer's guide is based on a pre-release version of Delphi 2005. Features in the shipping product may vary slightly from the descriptions found here.

Integrated Development Environment

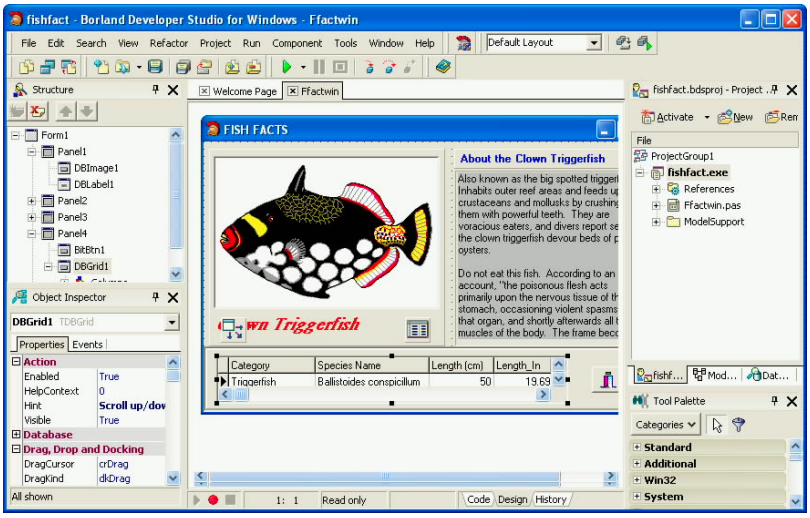
Borland®

Delphi™ 2005



The Integrated Development Environment

The Delphi 2005 IDE (integrated development environment) represents state-of-the-art in software development tools. Growing out of Borland's Galileo IDE technology first release with Borland C#Builder™ and Delphi 8 for .NET, Delphi 2005's IDE continues Borland's rich heritage of enabling you to develop applications faster and better.



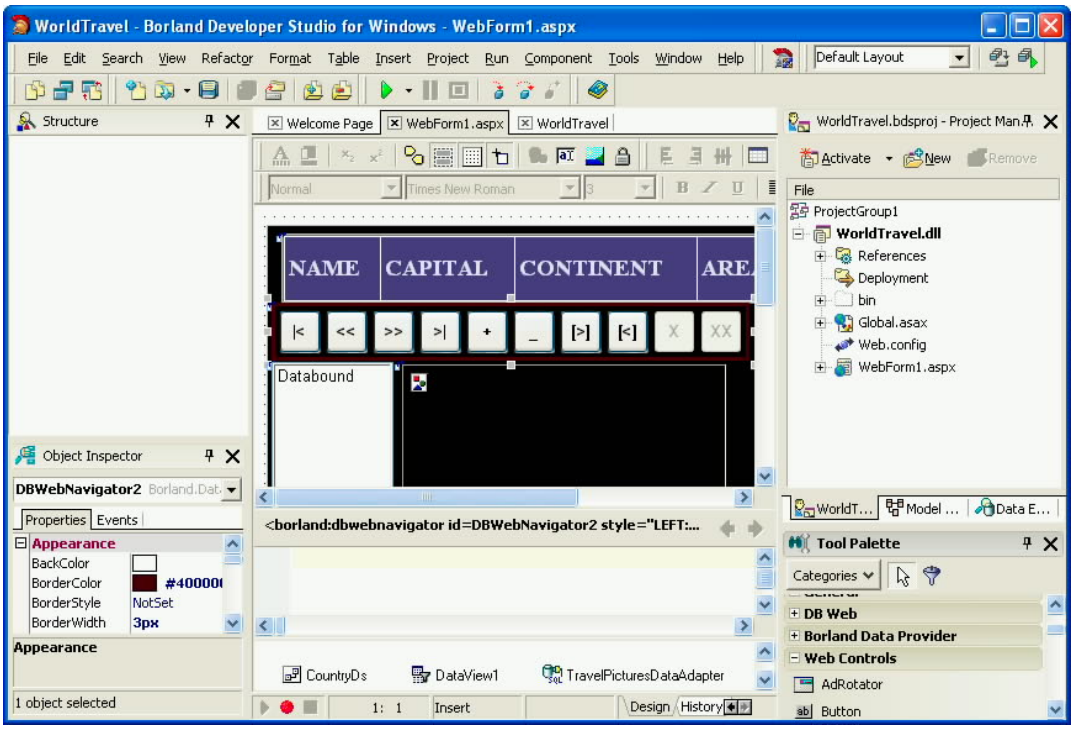
This section focuses on the features found in the various panels, designers, dialog boxes, and views of the IDE. Features that are specific to the code editor are detailed separately in a later section of this guide.

One IDE, Multiple Personalities

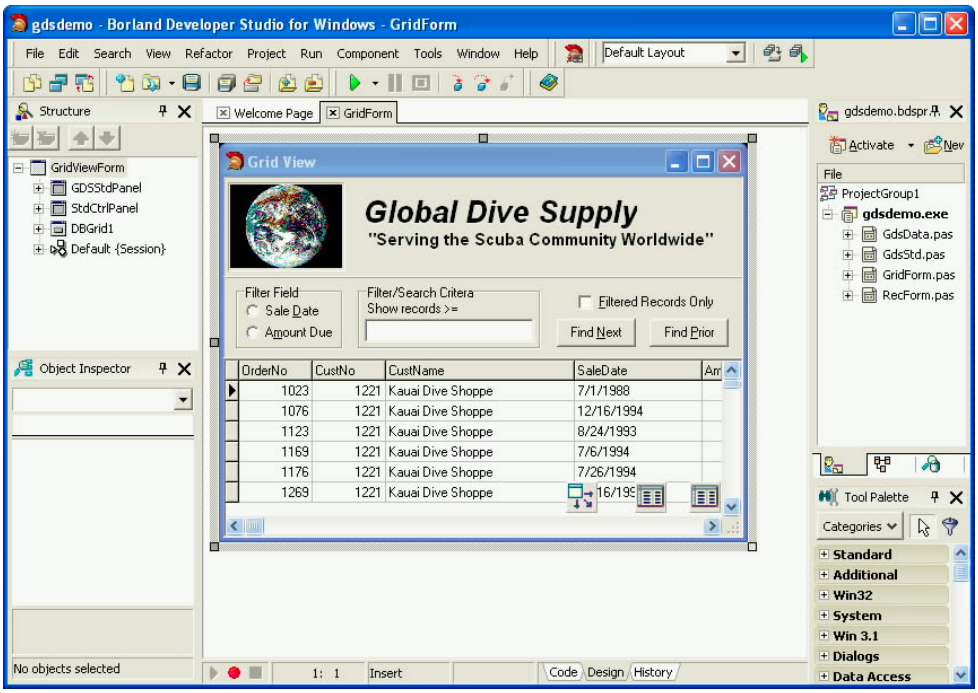
Whether you are coding in Delphi or C#, writing Win32 applications or .NET managed code, building ASP.NET Web pages or traditional client applications, Delphi 2005's IDE provides you with a consistent and powerful set of development tools designed to increase your productivity.

With Delphi 2005, the IDE keeps track of what kind of application you are working with, providing you with the designers, views, and features consistent with the task at hand. For

example, if you are building an ASP.NET Web application, the HTML designer allows you to design your Web pages visually, permitting you to drag-and-drop the components that you want to see on your Web page and configure them with little or no code. The following figure shows Delphi 2005 with an open ASP.NET Web application and its visual HTML designer.



If you create a new Win32 client application, or open an existing one, the VCL (visual component library) designer kicks in, again providing you with unmatched support for designing your user interfaces.



You can even create project groups that include two or more different kinds of projects. When you do this, the type of application that is currently active in the project group determines which designers are available, and which options you see in the supporting views. For instance, if your project group includes both an ASP.NET Web Service application and a Win32 VCL Form application, Delphi 2005 notes which of these projects is currently active, providing you with the designer and editor appropriate for each as you switch between your projects.

One IDE, Multiple Languages

Delphi 2005 is more than just context-sensitive designers — it is a full multiple-language development environment. The native languages and debuggers that are included in Delphi 2005 are Delphi for Win32 development, Delphi for the Microsoft .NET Framework, and C# for the Microsoft .NET Framework.

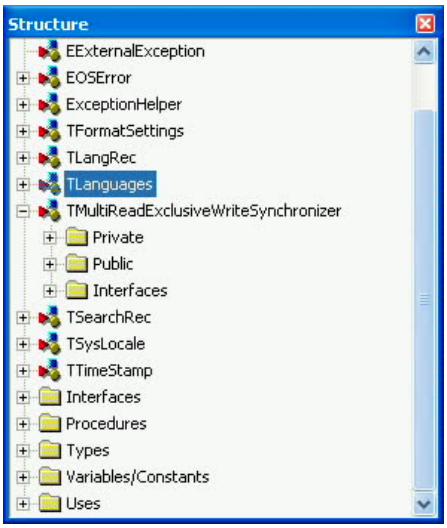
While other IDEs support multiple languages, Delphi 2005 is unique in that it supports both multiple platforms and multiple languages transparently. For example, you can create a

project group that includes a C# ASP.NET Web application, a Delphi for .NET Web Control class library, and a traditional Windows DLL (dynamic link library) written in Delphi Win32. Not only will the appropriate compiler and debugger be used for each project, based on its underlying language, but also the code editor features and Tool Palette snippets will expose the appropriate features as you navigate between the various projects.

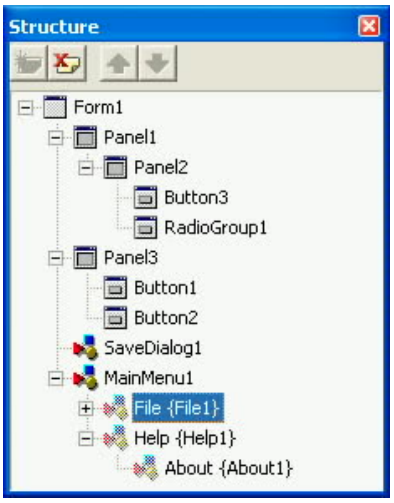
Delphi 2005 can also support additional compilers, if you wish. For example, so long as you have the VB for .NET compiler installed on your workstation, you can create, open, edit, compile, and debug VB for .NET applications without ever leaving the Delphi 2005 IDE.

The Structure Pane

The Structure pane is a context-sensitive view that provides you with detailed information about what ever is displayed in your main view. When you are using the code editor, the Structure pane displays the classes, types, interfaces, and other symbols in the current file, as shown in the following figure. (In Delphi 7, this view was called the Code Explorer.)



By comparison, when you are designing a VCL Form, the Structure pane displays the components that appear on your form, with the various nodes representing the containership of your controls. (In Delphi 7, this view was referred to as the Object Tree View.)



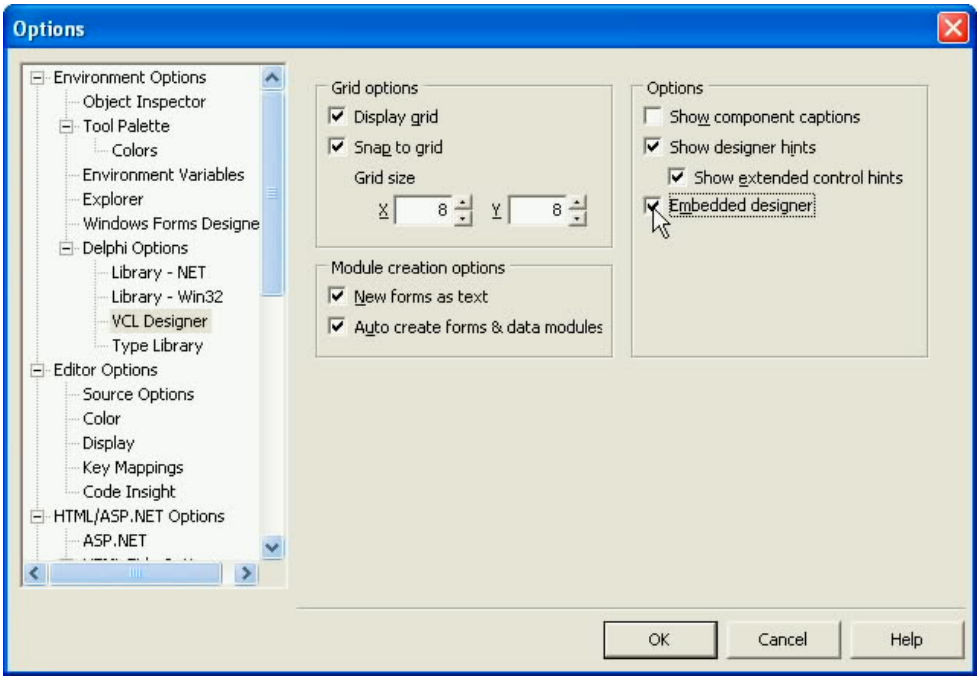
Not only does the Structure pane provide you with valuable insight into your projects, it also serves as a convenient tool for navigating the symbols and objects that you are using. When you are editing your code, double-clicking a symbol in the Structure pane takes you to the associated line of code in the editor. When you are designing a VCL Form, clicking an object selects it in the designer, permitting you to quickly change its properties or assign event handlers.

The Structure pane is also invaluable when there are errors in your code. When Delphi 2005's new Error Insight feature identifies problems in your source files, these appear automatically, as you type, in the Structure pane, permitting you to quickly navigate to the position in the code editor where problems exist. Error Insight is described in more detail in the "The Next Generation Code Editor" section of this guide.

The VCL and VCL for .NET Floating Designer

Some developers who used Delphi 8 for the Microsoft .NET Framework wished for a "floating" VCL designer, like the one available in Delphi 7. Borland listened. For Delphi development of VCL and VCL for .NET applications, Delphi 2005 provides you with a choice between using the .NET-style embedded designer or the classic floating designer.

To enable the floating designer in Delphi 2005, select Tools | Options. Navigate to the VCL Designer node under Delphi Options, and uncheck the Embedded designer check box.



The Tool Palette

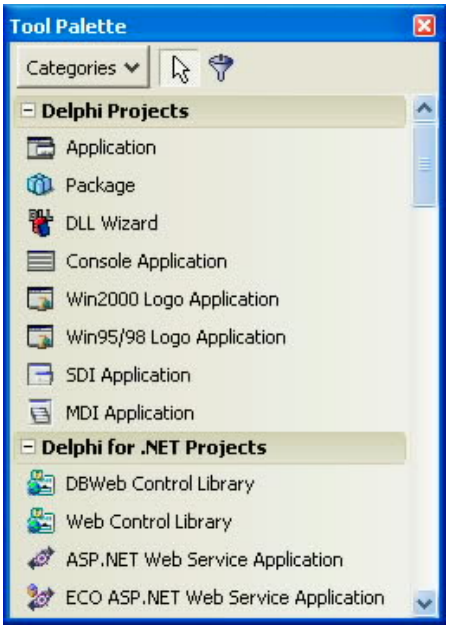
When you work in a component-based environment like Delphi 2005, you typically make extensive use of design-time components, which are placed into the designer and configured using the Object Inspector. These components are available from the Delphi 2005 Tool Palette.

The Tool Palette is organized by component category. Which categories are displayed, and which components appear within them, is context sensitive, based on the type of project on which you are working. Furthermore, the Tool Palette permits you to controls its organization. You can change the position of a component within a Tool Palette category, as well as move a component to a different category, simply by dragging the component within the Tool Palette. You can even define your own custom categories into which you can drag your components.

Delphi 2005 includes a number of enhancements to the Tool Palette. These are discussed in the following sections.

Enhanced Tool Palette Behavior

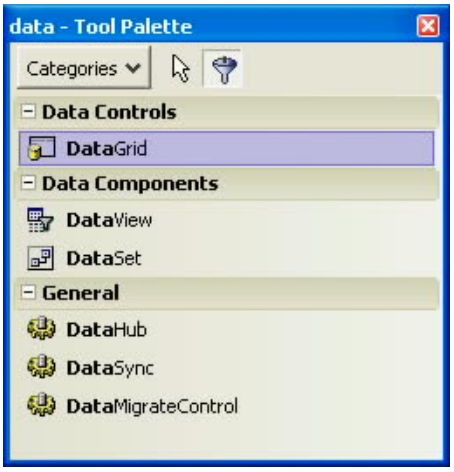
Delphi 2005's Tool Palette is better than ever. In addition to providing access to design-time components and code snippets, depending on whether you are working with a designer or code editor, the updated Tool Palette can also be used to create new projects, files, and objects. When you do not currently have a project open, the Tool Palette provides access to all of the wizards and templates of the Object Repository. Some of these are shown in the following figure.



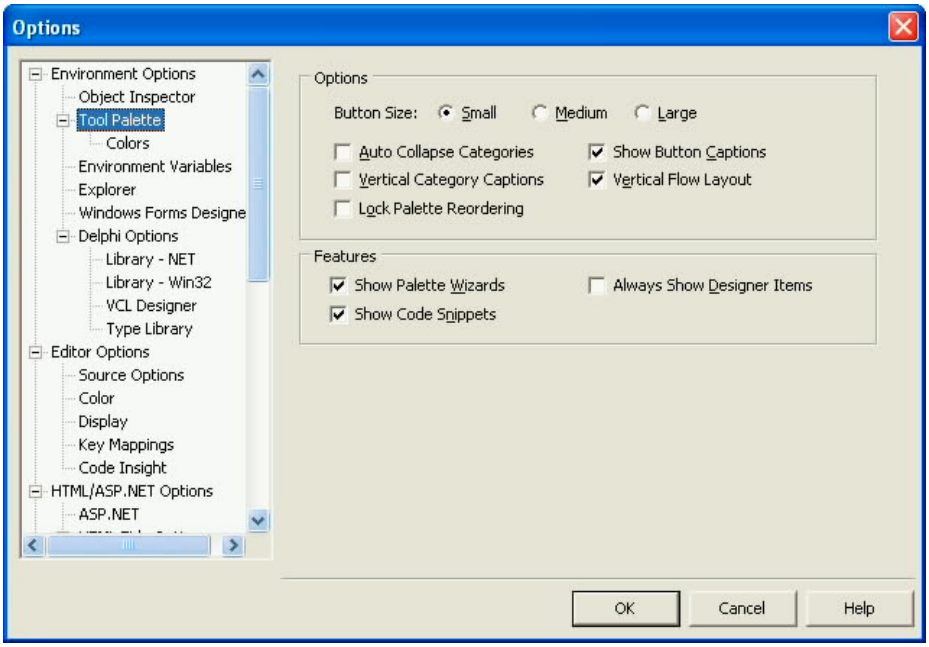
When you are using the code editor, the Tool Palette now includes these same options in addition to code snippets, and reusable pieces of code that you can drag into the code editor.

Selecting objects from the Tool Palette has also been enhanced, greatly improving the speed with which you can build forms and applications. Simply click the Filter Current Items button in the Tool Palette toolbar, or press Ctrl-Alt-P, and start typing the name of the object you want to select. As you type, the characters you've entered so far appear in the Tool Palette title

bar, and a filtered list of matching objects appears below, as shown in the following figure. Press Enter when the item you want is selected.



You also have additional options for controlling the Tool Palette display. To see these options, select Tools | Options from the main menu. Tool Palette configuration options are available under the Tool Palette node of the Options dialog box.



Finally, the Tool Palette in Delphi 2005 now supports true drag-and-drop placement of components into the designer you are working with. Previously, component placement with VCL Forms could be better described as click-and-click, though that technique also works in Delphi 2005.

New VCL for .NET Components

For Delphi VCL-based development, the Tool Palette now includes a number of new controls for creating better using interfaces. These include a TButtonGroup, TCategoryButtons, and TDockTabSet. These components, which you can use in your Win32 and VCL for .NET applications, permit you to easily create interfaces similar to those used in Delphi 2005's Tool Palette and the Structure pane. As you have probably already guessed, these new components are the same ones that Borland engineers developed to build the Delphi 2005 IDE.

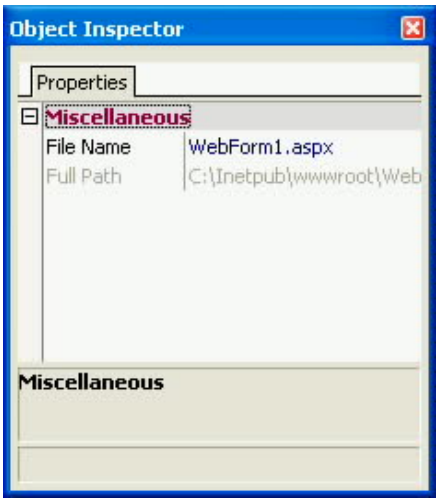
In addition, VCL for .NET has been expanded to include even more Delphi VCL-compatible classes. These additional classes make it even easier than before to migrate your existing Win32 projects to the .NET framework.

For a complete list of the new components in Delphi 2005, see "What's New in Delphi 2005" in the Delphi 2005 help.

The Object Inspector

The Delphi 2005 Object Inspector, which you use to configure objects placed on your form at design time, has also been updated. Not only does the Object Inspector permit you to configure properties and events for the objects that you have placed into the designer, but you can also use it to control file names and get information about objects that you select in the Project Manager.

For example, select a file in the Project Manager, such as an .aspx file in an ASP.NET Web application, and the file path and file name will appear in the Object Inspector, as shown in the following figure.



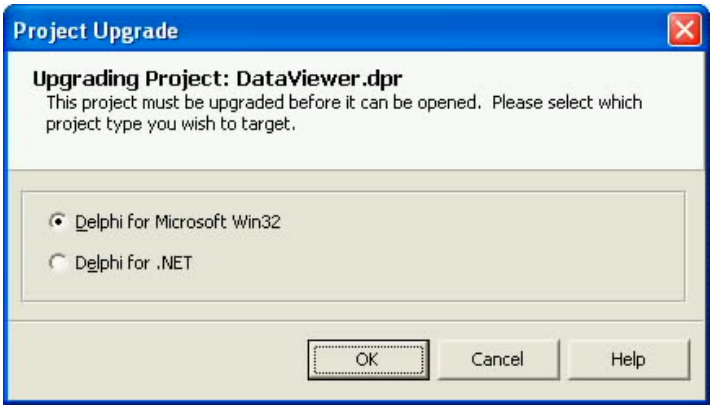
The File Name property in the preceding figure is shown in an enabled font, indicating that you can edit the name of this file using the Object Inspector. Changing the file name here not only changes the name of the file displayed within the Project Manager, but since this file is a Delphi unit, the unit name changes as well. Of course, you can still rename a file the old fashioned way, by selecting File | Save As from the main menu.

Other objects selectable within the Project Manager can also be viewed in the Object Inspector. For example, if you select one of the assemblies listed under the References node of a .NET project in the Project Manager, the Object Inspector displays details about that assembly, as shown in this next figure.

The Upgrade Project Wizard

Because Delphi 2005 includes both Win32 and .NET compilers for the Delphi language, it can be used to create new Win32 applications as well as further the development of your existing Win32 projects that you created in Delphi 7 and earlier. You can also use Delphi 2005 to migrate your existing Win32 applications to VCL for .NET, the 100% .NET managed-code solution that maintains component and source code compatibility between Win32 and .NET development.

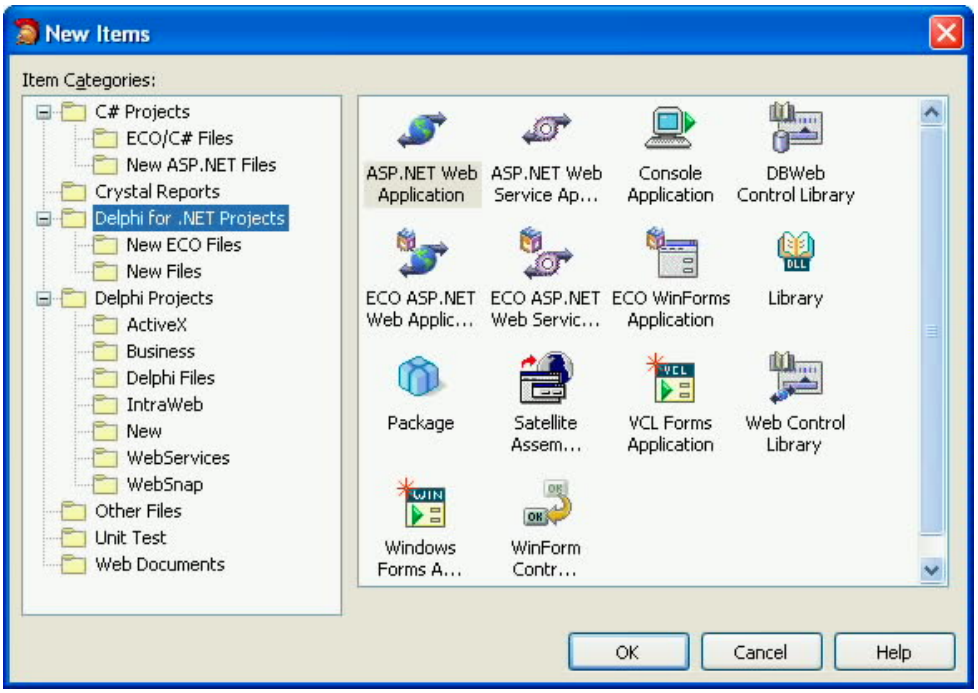
The Upgrade Project Wizard is a special utility that runs the first time you open a Win32 application in Delphi 2005. Using this utility, you can choose to continue the current project as a Win32 application, or you can convert it to a .NET application.



Once you made your choice using this wizard, Delphi 2005 will remember your selection. If you tell the Project Upgrade Wizard that you want to continue working with a Delphi project as a Win32 project, and at some later time decide to migrate it to VCL for .NET, simply delete your project's *.bdsproj file. After that, open the .dpr file in Delphi 2005. Once again, the Project Update Wizard will ask you to choose whether to continue working with the project as a Win32 project or to migrate it to VCL for .NET.

Delphi 2005 Wizards

Wizards are small applets that help you to quickly create the projects, objects, and files that you use in Delphi 2005. For example, the ASP.NET Web Application Wizard creates for you the necessary web.config, global.asax, and initial .aspx file, and configures an IIS virtual directory into which these are placed, among other tasks. In short, wizards increase your productivity, getting you off to a fast start in the right direction. The following figure shows the Delphi 2005 object repository, displaying just a few of the many available wizards.



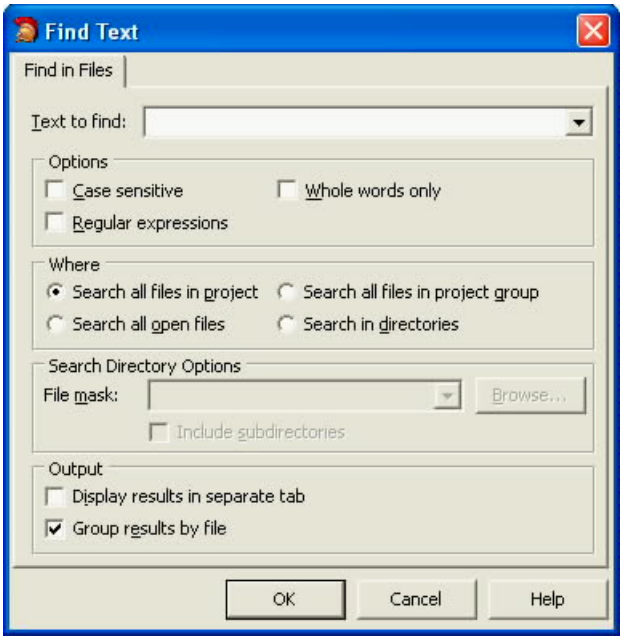
Delphi has always provided you with a rich collection of wizards that support almost every aspect of Windows development. For Win32 development, these include the Windows 2000 Logo Wizard, the DLL Wizard, the Automation Object Wizard, the Web Service Wizard, the IntraWeb Application Wizard, the Database Form Wizard, and the Thread Wizard. These are just some of the dozens of powerful wizards that are available.

For Delphi for .NET and C#, you will find the ASP.NET Web Application Wizard, the Windows Form Application Wizard, ASP.NET Web Service Application Wizard, the Web Control Library Wizard, and many, many more.

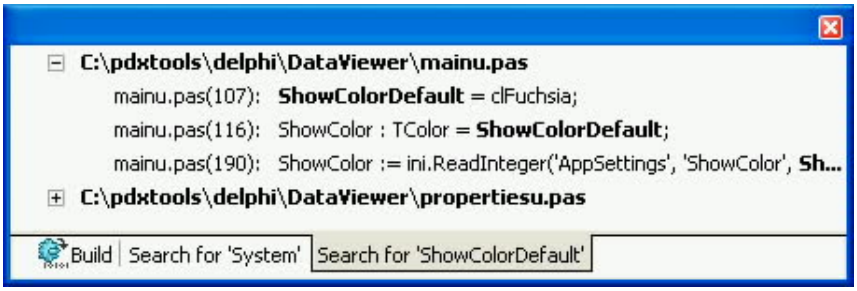
Delphi 2005 includes Wizards that were previously available in Delphi 7, Delphi 8 for the Microsoft .NET Framework, and C# for the Microsoft .NET Framework. In addition, Delphi 2005 includes a number of new and improved wizards — accelerating your development efforts even more. These include the updated New Component Wizard, the new DB Web Control Library Wizard, the ECO ASP.NET Application Wizard, the ECO Web Service Application Wizard, and the Satellite Assembly Wizard, just to name a few.

Find in Files Enhancements

Delphi 2005 makes it even easier for you to search your project files by allowing you to group search results by file. Simply check the Group results by file check box in the Find Text dialog box.



The following figure shows what a grouped search result looks like. As you can see, each file in which the search string appears forms a base node in a tree view. Expanding the node for a given file lists the lines on which the located search string was found. You can then double-click a particular entry to go to that line of code in the code editor.



Updated Support for International Characters

The Delphi 2005 IDE has been upgraded across the board to support UTF-8 characters in all of its wizards, windows, dialog boxes, and panes.

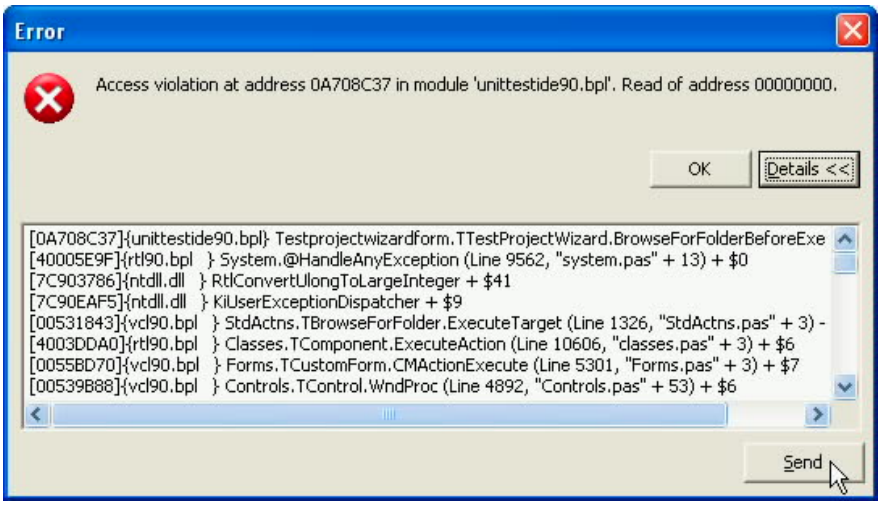
Message List Enhancements

Delphi 2005 uses the Message List pane to list compiler errors, warnings, and hints. You can now save the contents of the Message List pane by right clicking in the Message List pane and selecting either Copy, to copy selected messages to the Windows clipboard, or Save, to save the Message List contents to a file.

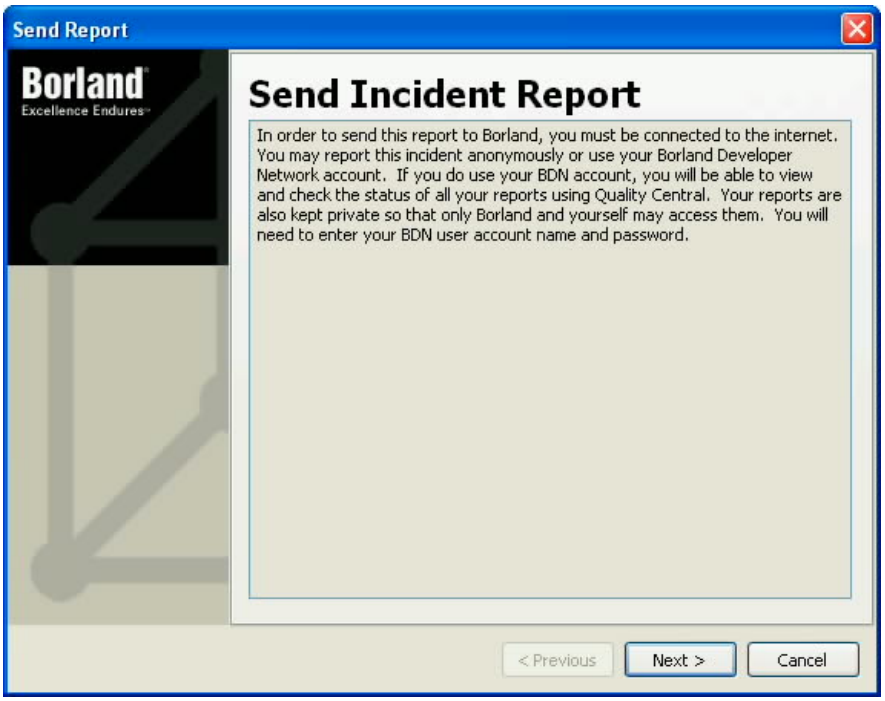
IDE Error Reporting

Borland's commitment to creating better software has lead to the development of a number of programs for reporting and fixing problems. One of the most recent of these is Quality Central, a Web-based application for submitting bug reports located at http://qc.borland.com.

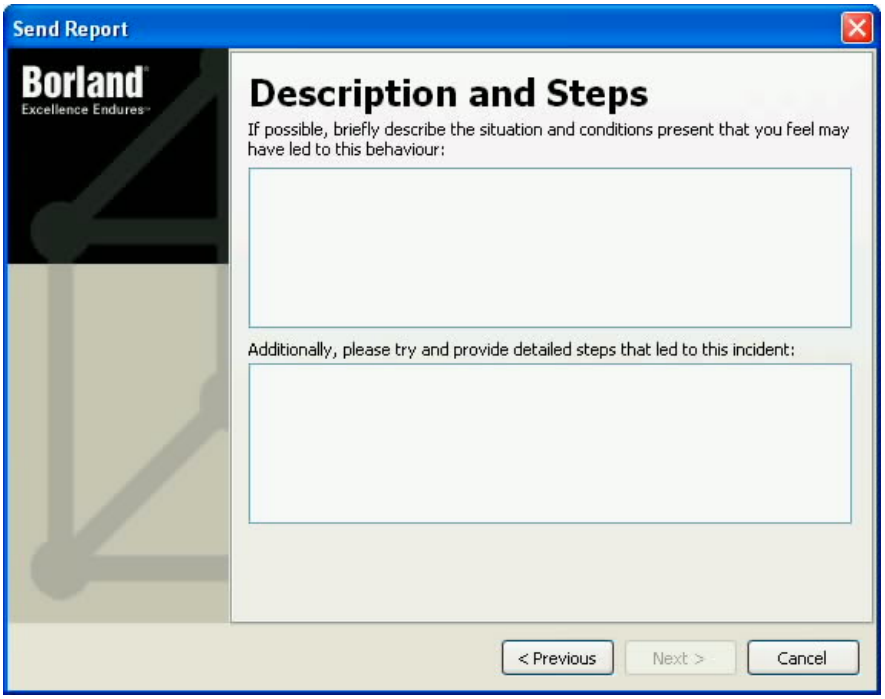
With Delphi 2005, Borland has embedded an error reporting system directly into the IDE. This feature is called IDE Error Reporting. If an exception is raised within the IDE, Delphi 2005 displays the Error dialog box. If you click the Details button, you see a detailed trace of the error.



Clicking the Send button displays the Send Report dialog box.



Click the Next button to see the stack trace that will be submitted to Borland along with your error report. Click Next again to enter a description of what you were doing when the error occurred.



Click Next once more to optionally provide your Borland Developer Network (BDN) logon email address and password. Submitting your report using your BDN account allows you to easily follow up on your report using Borland's Quality Central. If you want to submit the report anonymously, check the Anonymous Report check box.

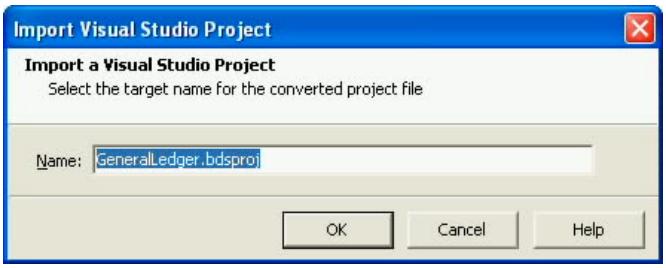
Click Next one final time to submit the error report.

Import/Export Project from/to Visual Studio .NET

Do you currently have C# projects in Visual Studio .NET 2003, but need the advanced features offered by Delphi 2005? Don't worry. Importing these projects into Delphi 2005 is easy.

Simply select File | Open, and open the Visual Studio C# project file (*.csproj). The Delphi 2005 Import Visual Studio Project Wizard will ask you for the name you want to give to the imported project. From that point forward, you can use the Delphi 2005 features to design, develop, compile, test, and deploy the application.

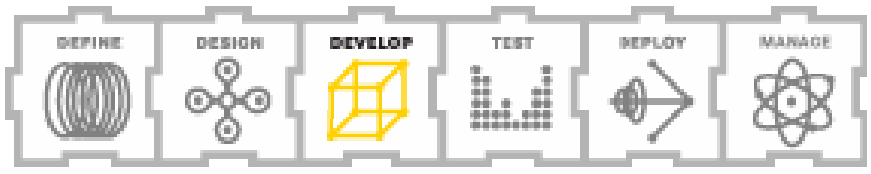
The following figure shows a C# project created in Visual Studio .NET 2003 being imported into Delphi 2005.



While the features of Delphi 2005 make it the preferred environment for .NET development, C# projects built in Delphi 2005 can be exported to Visual Studio if you need to share the results of your work with a VS-based developer. To do this, select Tools | Export to Visual Studio from Delphi 2005's main menu. Note that this menu item is only available when the current project in the Project Manager is a C# project.

The Next Generation Code Editor

Borland®
Delphi™ 2005



The Next Generation Code Editor

Delphi 2005 continues Borland's heritage of providing developers with a world-class programming environment. To most developers, that also means a world-class code editor. And that's exactly what you get in Delphi 2005.

In fact, for most developers, the updates that Borland has introduced to the code editor in Delphi 2005 will provide ample justification to upgrade from a previous version of Delphi or C#Builder. These features include refactoring support, SyncEdit, Error Insight, Help Insight, the History Manager, and much, much more. These new features are described in the following sections.

Refactoring

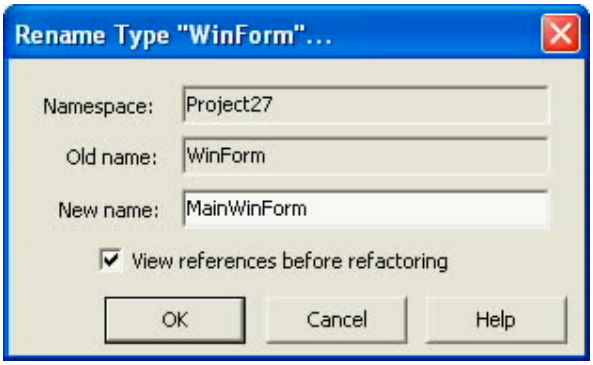
Refactoring is the process of updating existing code to improve its readability, maintainability, and efficiency, without changing the essential behavior of the software. Common refactorings include providing more expressive names for variables, replacing duplicate code segments with a call to a common function that performs the same task, and replacing literal values with constants or resource references.

Delphi 2005 includes a number of impressive refactorings. These include symbol renaming, method extraction, variable and field declarations, and resource refactorings.

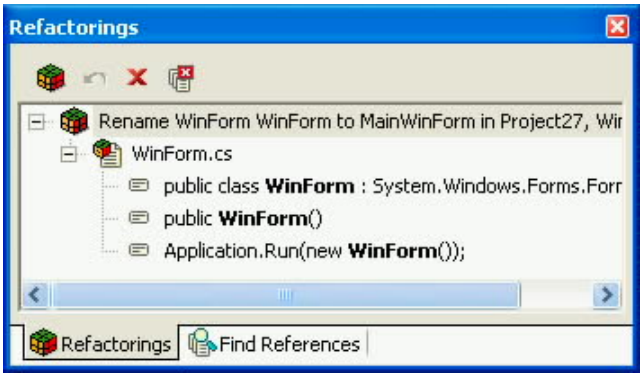
Symbol Renaming

Symbol renaming allows you to change all instances of a symbol's name throughout your project. Unlike a search-and-replace feature, symbol renaming respects the context in which the symbol name appears. Symbols that can be renamed using this refactoring include class and interface names, properties, methods, functions and procedures, as well as variables and constants.

To perform a symbol renaming refactoring, select the symbol whose name you want to change in the code editor, and select Refactoring | Rename. Use the Rename dialog box to define a new name for your symbol.



If you leave the View references before refactoring option checked, Delphi 2005 displays the Refactorings pane, which lists all of the instances within your code where the change will be applied.



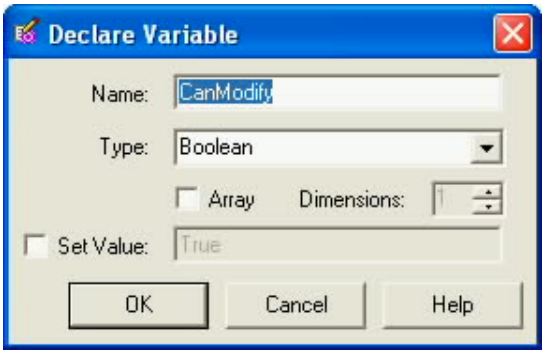
Click the Refactor button on the Refactoring pane toolbar to apply the changes. Alternatively, you can choose to remove one or more of the refactorings before applying them, or even cancel the refactoring altogether.

Variable and Field Declarations

The Declare Variable and Declare Field options on the Refactor menu permit you to quickly create a local variable or member field declaration. This option is available with Delphi code,

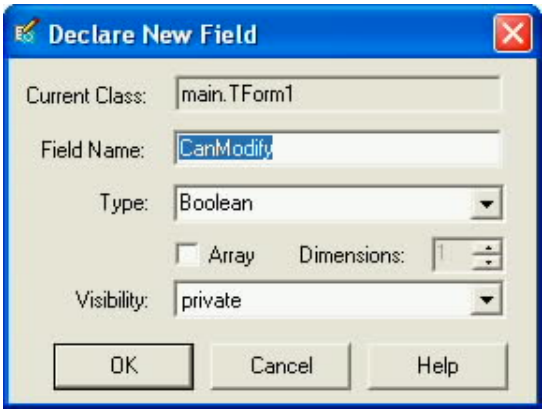
but not with C# projects. (This feature is not needed in C# since fields can appear almost anywhere within a C# class. By comparison, in Delphi, variables must appear in a **var** block, and member fields must appear in a **type** block.)

To insert a local variable or member field, select the symbol name that you created in the code editor, and select Refactor | Declare Variable or Refactor | Declare Field (or press Ctrl-Shift-V or Ctrl-Shift-D, respectively). If you select Declare Variable, the Declare Variable dialog box is shown.



You use the Declare Variable dialog box to change the variable name, set its data type, make the variable an array type with a specific dimension, or to initialize the newly created variable to a specific value. Click OK to create the local variable, and initialize its value (if you chose that option).

If you select Declare Field, the Declare New Field dialog box is displayed. You use this dialog box to set the name and data type of the new field, to declare it as an array of a given dimension, and to define its visibility within the associated class. When you click OK, the newly named field is created in the selected section of the class within whose method the symbol is located.



Resource Refactoring

Resource refactorings are used in Delphi code to convert string literals into **resourcestring** block entries, replacing the original literal with the resource string symbol. (There is no **resourcestring** block in the C# language.) Using resource strings instead of string literals is particularly valuable when a specific string literal is used repeatedly, as well as when you need to create localized (language and/or culture specific) versions of your application.

After placing your cursor within a string literal in the Delphi code editor, select Refactor | Extract Resource String. Use the Extract Resource String dialog box to modify the string and to change the default name for the resource symbol. When you click OK, the string literal is replaced with the resource symbol, and the named symbol is inserted into a **resourcestring** block in the associated unit's interface section.

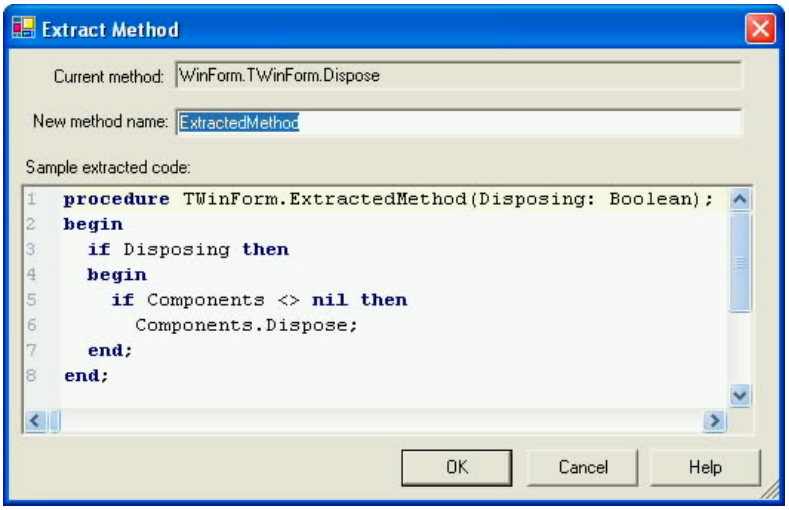
Extract Method Refactoring

Most developers think of method extraction when they think of refactoring. Method extraction involves converting one or more lines of code into an independent method call, replacing those lines with an invocation of the extracted method.

In Delphi 2005, method extraction refactoring is only available for the Delphi language.

Method extraction is particularly useful when the same or similar lines of code appear repeatedly in your project. By extracting those lines to a separate method, replacing each of the repeated instances with an invocation of the method, you greatly enhance your code's maintainability by creating a single location where changes to those lines of code, if desired, need to be implemented.

To perform a method extraction refactoring, select the lines of code that you want to extract to a method, and then select Refactor | Extract Method. Use the Extract Method dialog box to define a name for the new method, as well as to examine the code that will be placed inside of this new method.



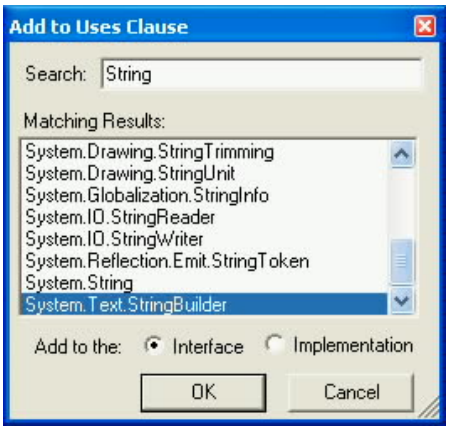
Delphi 2005's extract method refactoring is intelligent, with respect to variables, properties, and objects referenced within the code being extracted. For example, since the code in the preceding figure includes a reference to the Disposing property of the method's class, the

value of this property is passed by value to the refactored code. By comparison, if the code actually made a change to the value of a variable that needs to be passed into the refactored method, the associated parameter would be passed by reference (using the **var** keyword).

Import Namespace (C#) and Find Unit (Delphi)

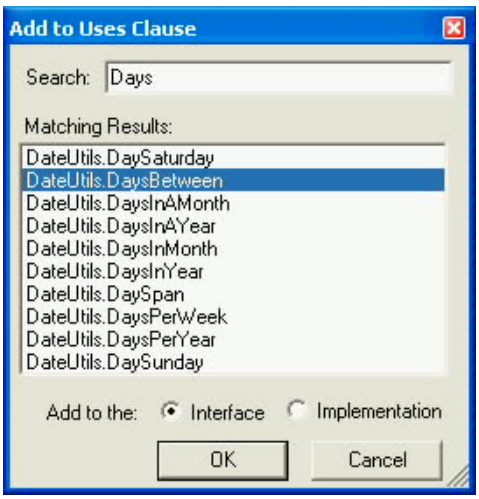
Although not exactly a refactoring, the Import Namespace and Find Unit options under the Refactor menu permit you to quickly locate and import the namespace associated with a particular symbol. If you are coding in C#, you select Refactor | Import Namespace. Delphi developers select Refactor | Find Unit.

After selecting this option from the Refactor menu, the displayed dialog box lists all of the classes in all of the namespaces available to the environment you are working in. For example, if you are creating a Delphi .NET Windows Forms application, the namespaces of the FCL and RTL for .NET (the .NET version of the Delphi runtime library) are available. Delphi VCL for .NET developers will find the VCL for .NET namespaces as well.



By comparison, if you are creating a Delphi Win32 application, the various units of the VCL and RTL are listed. Type the name of the class that you want to be able to access in the Search field. As you type, the Matching Results list is filtered to include only those classes, and their associated namespaces, whose names match what you've typed so far.

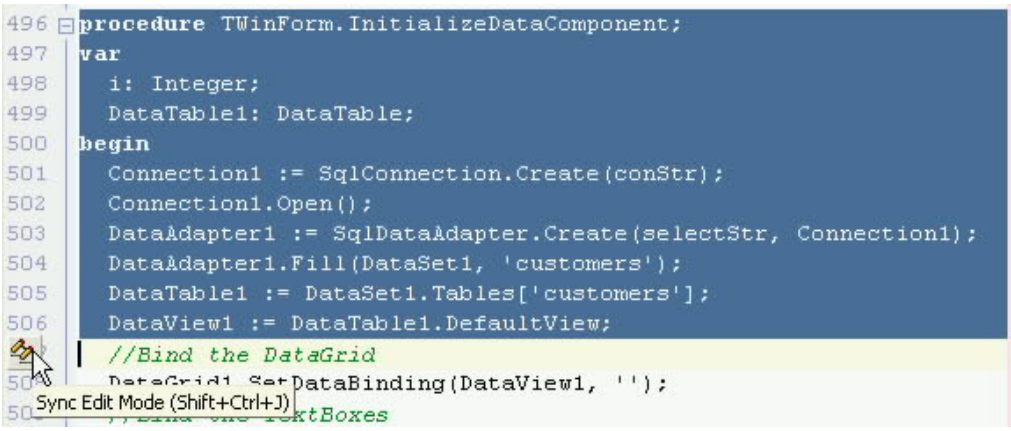
Select the name of the class whose namespace you want and click OK. If you are working in Delphi, you can also specify whether the namespace will be added to your interface or implementation section uses clause.



SyncEdit

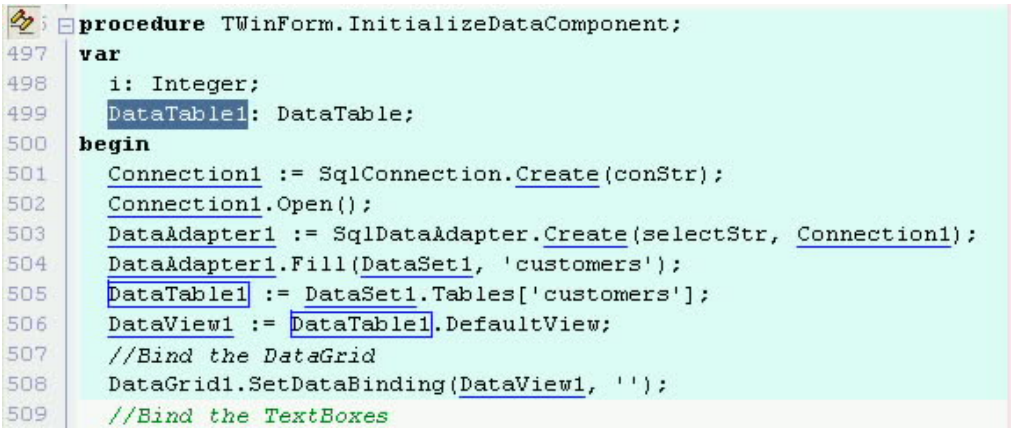
SyncEdit is a new feature in Delphi 2005 that provides support similar to symbol renaming refactoring. Unlike symbol renaming, however, SyncEdit performs localized renaming of symbols for a selected code block only. This is a powerful capability and one of the most popular new features with developers.

SyncEdit becomes available anytime you select a code block that includes at least two instances of the same symbol name. For example, consider the following figure, which depicts a selected code block that includes more than one reference to a local variable named DataTable1 (as well as DataSet1, DataAdapter1, Connection1, and the Create method).



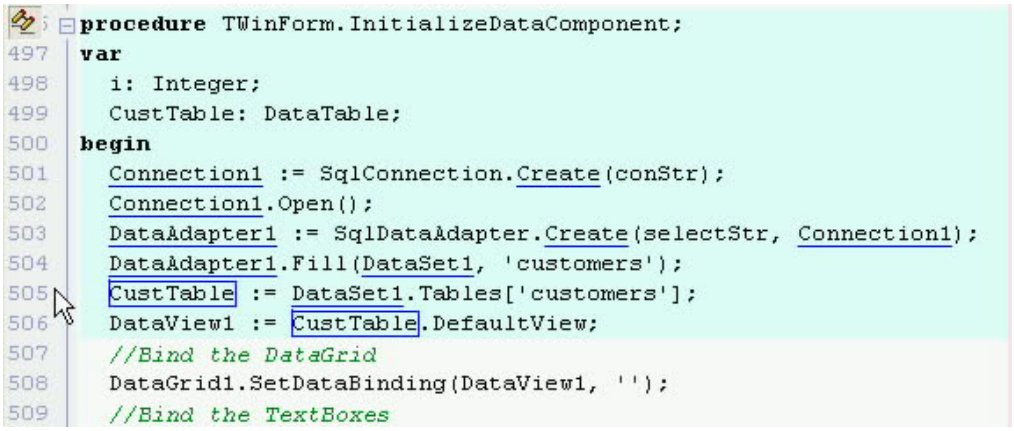
The SyncEdit icon appears in the left gutter of the editor window, indicating that synchronized changes to the selected code block are available. To enter the SyncEdit mode, you either click this icon or press Shift-Ctrl-J.

Once you enter the SyncEdit mode, the duplicate symbols are identified, and the symbol selected for synchronized editing appears highlighted (with the duplicates being displayed enclosed in boxes). If you want to edit a symbol other than the one selected by default, press the Tab key until the symbol you want to SyncEdit is selected.



After selecting the symbol to edit, begin typing. The name of the selected symbol, and its duplicates, are updated as you type. The following figure shows the name of the DataTable

being changed to CustTable. (The edit is being performed on the first instance of DataTable1, which appeared in the **var** declaration of this method.)



```
procedure TWinForm.InitializeDataComponent;
var
  i: Integer;
  CustTable: DataTable;
begin
  Connection1 := SqlConnection.Create(conStr);
  Connection1.Open();
  DataAdapter1 := SqlDataAdapter.Create(selectStr, Connection1);
  DataAdapter1.Fill(DataSet1, 'customers');
  CustTable := DataSet1.Tables['customers'];
  DataView1 := CustTable.DefaultView;
  //Bind the DataGrid
  DataGrid1.SetDataBinding(DataView1, '');
  //Bind the TextBoxes
```

SyncEdit is a great productivity tool when you are writing functions, procedures, and methods, in that this feature is so easy to use. There are, however, important differences between SyncEdit and symbol renaming refactorings. SyncEdit is lexical, so it works with comment lines as well as compilable code, unlike symbol renaming refactorings, which work only on actual symbol references. Likewise, symbol renaming refactoring extends its reach into descendant classes, as well as to resource files (such as VCL and VCL for .NET form files). SyncEdit only applies to the currently selected code block.

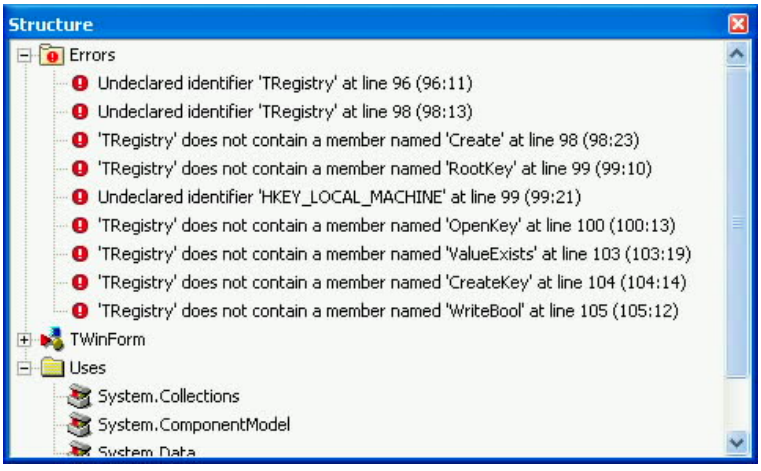
Error Insight

Error Insight, which makes its debut in Delphi 2005, provides you with a service that can be roughly described as spell checking and grammar checking for programmers. As you write your Delphi or C# code, the IDE actively evaluates your work, identifying the symbols, keywords, and directives that you use, looking for syntax and semantic errors that the compiler cannot resolve. When Error Insight locates an error, it identifies the problem by underscoring the offending text with red squiggly lines, similar to how Microsoft Word identifies words not in its dictionary.

```
procedure TWinForm.Button1_Click(sender: System.Object;
  e: System.EventArgs);
var
  RegVar: TRegistry;
begin
  RegVar := TRegistry.Create;
  RegVar.RootKey := HKEY_LOCAL_MACHINE;
  if RegVar.OpenKey('HKEY_CURRENT_USER\Software\' +
    'Borland\BDS\3.0\Form Design', False) then
  begin
    if not RegVar.ValueExists('Embedded Designer') then
      RegVar.CreateKey('Embedded Designer');
    RegVar.WriteBool('Embedded Designer', False);
  end;
end;
```

When you pause your mouse pointer briefly over a symbol that Error Insight does not recognize, Error Insight displays a hint window with information about the identified error.

In addition to the Error Insight features available in the code editor, the problems located by Error Insight also dynamically appear in the Structure pane, under the Errors node, and disappear as they are corrected. The following figure shows the Structure pane with a number of identified errors.



In the example shown in the preceding figures, adding the Borland.Vcl.Registry unit (for the TRegistry class) and Borland.Vcl.Windows (for the HKEY_LOCAL_MACHINE constant) to

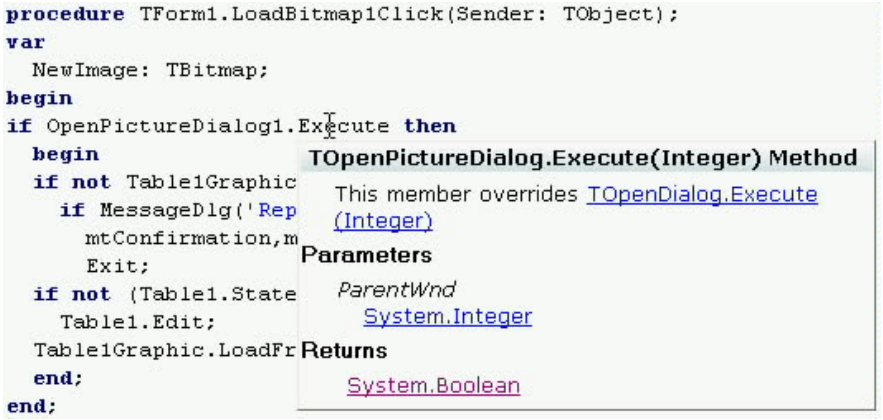
this unit's uses clause allows Error Insight to see the various symbols that it identified as problems. Once these two units are added to the uses clause, both the Structure pane and the code editor are updated, indicating that no problems are detected in this code.

You can configure Error Insight from the Code Insight node of the Options dialog box. Display this dialog box by selecting Tools | Options from the main menu.

Help Insight

Another new Code Insight feature appearing in Delphi 2005 is Help Insight. Help Insight provides you with information about the classes, interfaces, methods, properties, and fields that appear in your code, without you ever having to leave the code editor.

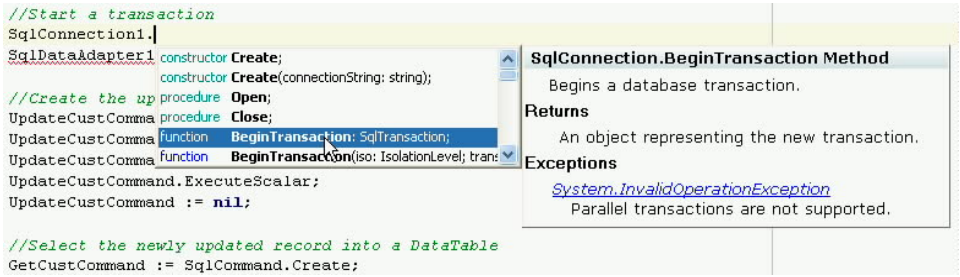
To access Help Insight, briefly pause your mouse pointer over a symbol in the code editor. After a moment, a hint window appears, displaying information about the symbol.



In many instances, Help Insight includes one or more links within the hint window. Clicking one of these links may drill down into the help, displaying an additional hint window with information about the link you clicked. Alternatively, clicking a link may take you to the line of code where the clicked symbol is defined.

Help Insight is also available from the windows displayed by Code Insight, including the Class Completion and Argument Value List windows. When a Code Insight window is active,

select an item in the Code Insight window to show the Help Insight for that item. For example, in the following figure Help Insight is displaying information about the BeginTransaction method of a SqlConnection object. This help became available after BeginTransaction was selected in the Code Completion window.



You can configure Help Insight from the Code Insight node of the Options dialog box.

The History Manager

One of the more exciting additions to the Delphi 2005 code editor is the History Manager. The History Manager, which you display by clicking the History tab when a source file is active in the code editor, allows you to view changes to your source files over time, view comments about specific versions of your source code, view the differences between the various saved versions of your files, and easily revert to any backup state or checkin.

By default, the History Manager transparently maintains local copies of your source files in a folder named __history under your project directory each time you save your changes. This feature is called local file backup, and you use the Options dialog box to configure how many versions of your local backup to keep. Delphi 2005 maintains the last 10 saved versions of each source file, by default. Depending on your available hard disk space, you may want to increase the number of backups.

If you are using Borland's StarTeam version control server, the History Manager maintains StarTeam checkins as well. Using this feature, you can not only view changes that you have made to the source files, but also compare your changes with those implemented by other developers working on the StarTeam-managed project. StarTeam also permits you to track

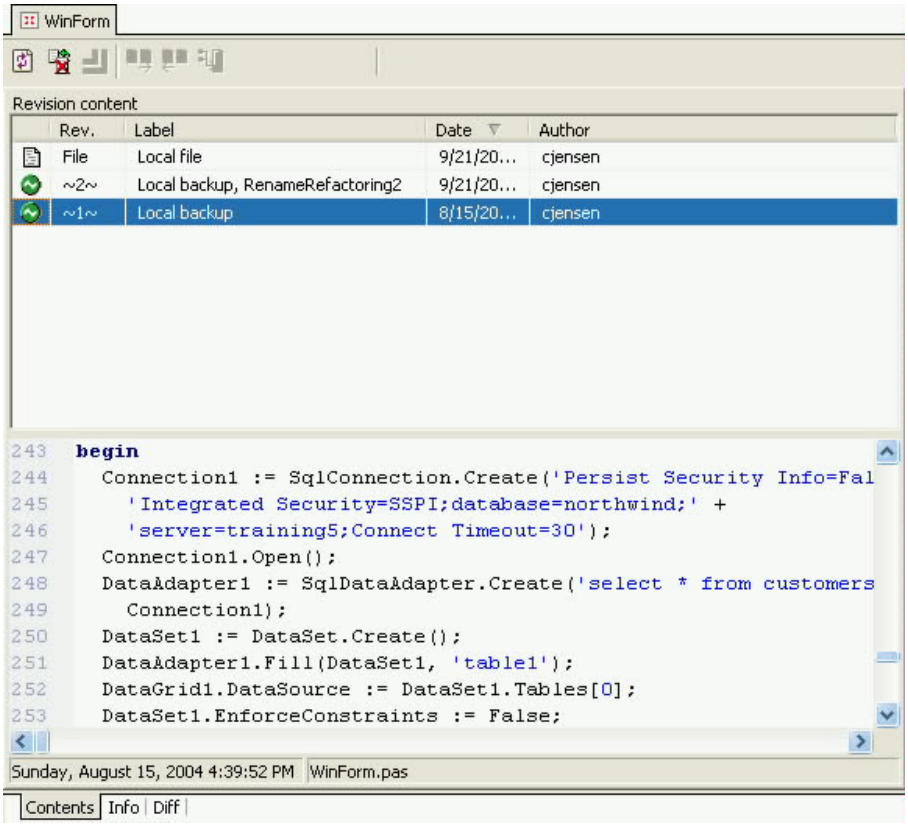
changes even after you have changed a file's name. In short, the History Manager provides you with a convenient and powerful interface to the robust StarTeam project asset management system.

It's worth noting that the History Manager also works with the DFM files of VCL and VCL for .NET applications. DFM files are used in those applications to persist information about the properties of the objects that appear on your forms, data modules, and frames. As a result, the History Manager permits you to view, manage, and restore changes made to your form designs using the same tools as those used on Delphi code files.

There are three panes available within the History Manager. These panes are named Content, Info, and Diff. Each of these panes is described in the following sections.

The Content Pane

You use the Content pane to review the contents of your saved source files, and optionally revert to a previously saved version. When you select a specific backup or the current saved version of the file, the contents of that file are displayed in the code area. In addition, the file name and the date last saved appear in the History Manager's status bar.



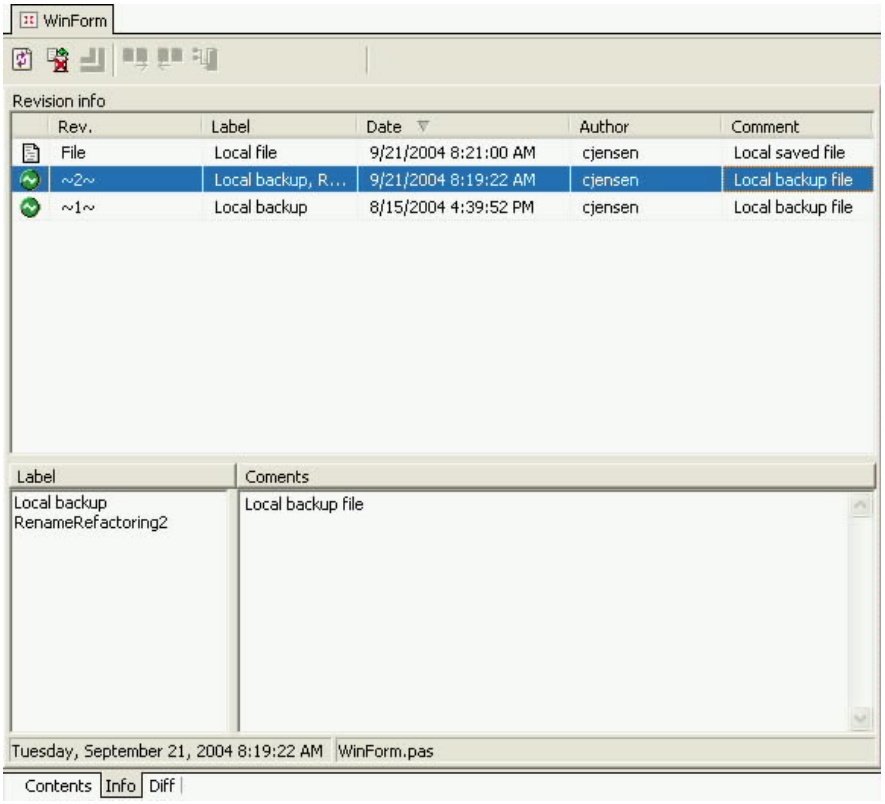
Use the code area to view the contents of the selected file. If you want, you can use the code area to select and copy (Ctrl-C) lines of code that you want to paste elsewhere within your project (or even into other projects).

If you want to revert your code to one of the previous saved versions, select the saved backup that you want to revert to and click the Revert to previous version button in the History Manager toolbar.

The Info Pane

You use the Info pane to view comments and notes associated with a particular version of your source file. If you are using StarTeam to manage your History Manager contents, these comments are linked to your StarTeam backups. If you are using local backups, these

comments are generated by Delphi 2005 and cannot be modified. Some operations, such as refactorings, write information into the Info pane of the History Manager.

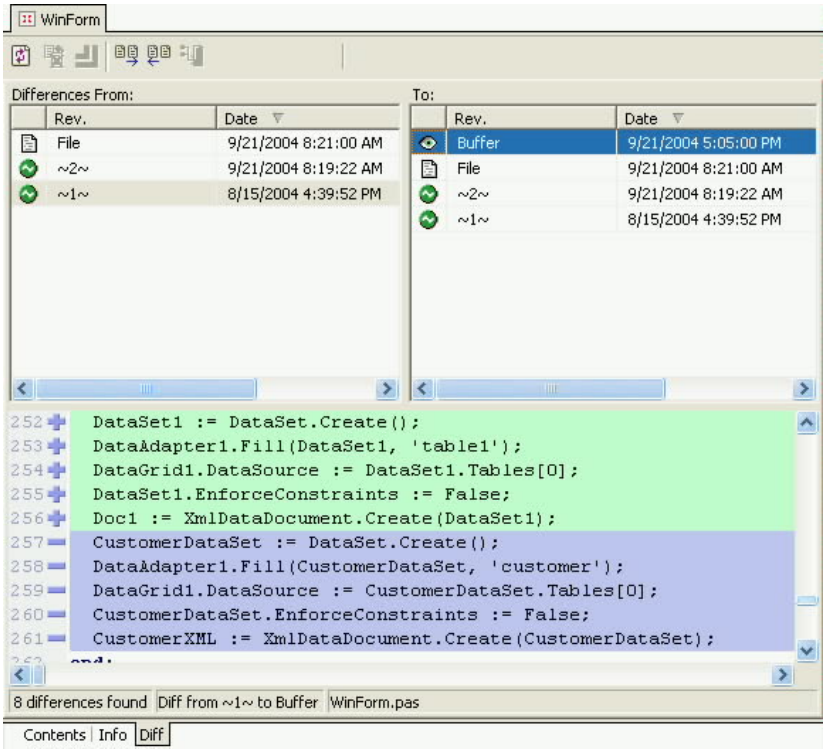


The Diff Pane

For most developers, the Diff pane offers the most valuable feature of the History Manger. The Diff pane provides insight into the differences between the multiple versions of your source code, including comparisons between the current edit buffer and saved source files.

Select one of the saved versions of your source file from the Differences From: list on the left side of the Diff pane, and either the contents of the current edit buffer or one of the other saved versions from the To: list on the right side. The difference view is displayed in the code area, with the newer code versions identified with a plus sign (+) in the left gutter, and the older versions identified with a minus (-) sign.

The following figure depicts changes between the current version of the file in the edit buffer and one of the saved local backups.



Code Navigation Enhancement

Code navigation is a feature of Delphi 2005 that permits you to easily move between sections of your code. For example, by pressing Ctrl-Shift-UpArrow (or Control-Shift-DownArrow), you can move effortlessly between a method name in a Delphi class declaration to the associated implementation of that method.

Delphi 2005 introduces a small but valuable enhancement to code navigation in Delphi code, allowing you to move between your interface and implementation section uses clauses, as well as between your unit's initialization and finalization sections, using Ctrl-Shift-UpArrow.

Code navigation is not necessary in C# projects, as the associated modules in C# do not have a two-part structure, as is the case with Delphi units.

Toggling Code to/from Comments

Delphi 2005 introduces a new feature that permits you to quickly comment and uncomment a selected code block. To comment one or more consecutive lines of code, select the code in the code editor, right-click, and then select Toggle Comment from the displayed context menu (or press Ctrl-/). When you do this, Delphi 2005 places the single line comment characters (//) at the start of each of the lines in the selected block.

To uncomment one or more consecutive lines, select those lines and press Ctrl-/, or right-click and select Toggle Comment. Delphi 2005 will respond by removing the single-line comment characters from each selected line in the block. The single-line comment characters do not have to be in the first column of the code editor for Delphi 2005 to remove them.

Persistent Bookmarks

Bookmarks are special tags that you place within a source file to enhance your navigation within that file. You place a bookmark by pressing Ctrl-Shift, followed by a single digit, from 0 to 9. Once placed, the bookmark appears in the left gutter of the code editor using a glyph that represents the digit.

Once a bookmark has been placed, you can quickly navigate to that bookmark within the code editor by pressing the Ctrl key followed by the digit used to place the bookmark. For example, if you have previously placed a bookmark using Ctrl-Shift-1, and subsequently navigate to a different area of your code file, you can instantly return to the bookmarked line in your source code by pressing Ctrl-1.

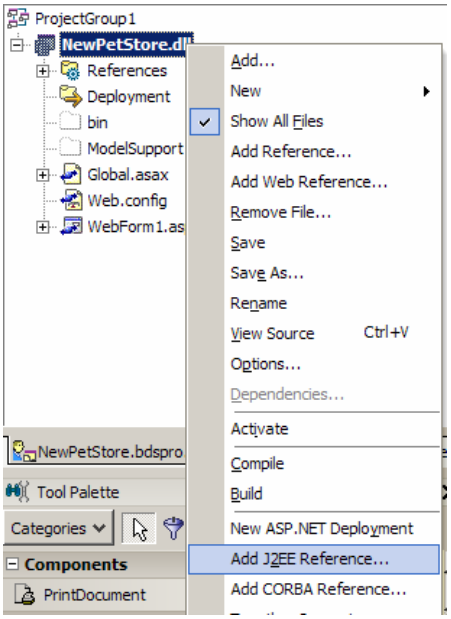
Delphi 2005 now supports persistent bookmarks. If persistent bookmarks are enabled, a placed bookmark will remain in the source code until you specifically remove it. This means that you can place a bookmark in one editing session, and that bookmark will still be there the next time you open that source code file in Delphi 2005.

In order to enable persistent bookmarks, check the Project desktop check box under the Autosave options group on the Environment Options page of the Options dialog box. You display the Options dialog box by selecting Tools | Options from the main menu.

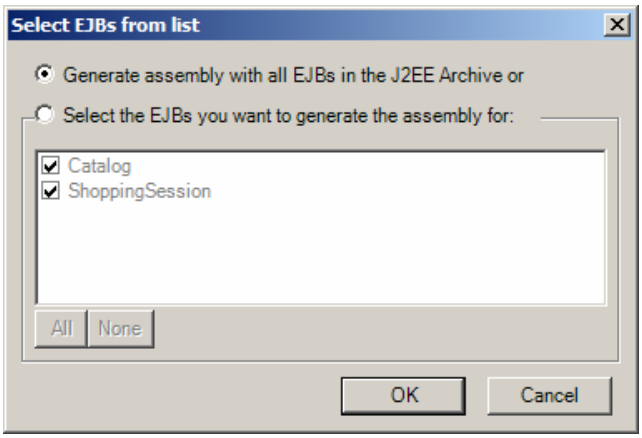
J2EE and CORBA to .NET Integration with Janeva

Janeva is Borland's middleware solution for using J2EE (Java 2 Enterprise Edition) Enterprise JavaBeans and CORBA (common object request broker architecture) objects from your Delphi 2005 applications. With Janeva, you can leverage your existing enterprise-level objects, calling them from your Web-enabled or workstation client .NET applications.

To enable access to a J2EE or CORBA object from your Delphi 2005 application, select the Project menu on Delphi 2005's main menu, or right-click the current project in the Project Manager, and select either Add J2EE Reference or Add CORBA Reference.



Use the displayed dialog box to select the Java .jar or .ear file, or the CORBA IDL (interface definition language) file. If you select a Java archive, for example, Delphi 2005 then permits you to choose which of the contained Enterprise JavaBeans you want to use, as shown in the following figure.



Selecting OK generates a proxy class that you use to make calls to the Java server at runtime.

User Selectable File Encoding

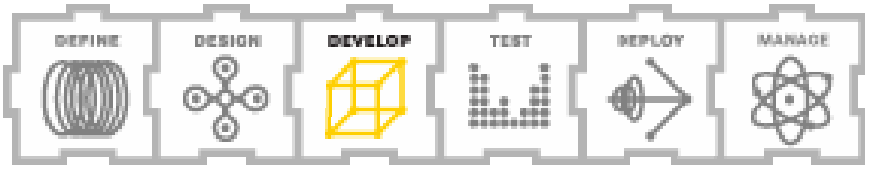
You can now choose how Delphi 2005 will encode your source files. Your options include ANSI, Binary, UTF8, and so on.

To set the file encoding for the current source file in the editor, right-click in the editor and select File Format from the context menu. Select the encoding you want to use from the displayed menu.

Being able to select the source file encoding is particularly valuable when you are writing source files using non-US locales. For example, source files encoded using UTF-8 will correctly maintain the identity of the individual characters even when opened in a different locale. By comparison, special characters in a source file may change if the source file is encoded in ANSI and then opened with a different ANSI codepage.

The VCL for .NET

Borland®
Delphi™ 2005



The VCL for .NET

The Delphi visual component library for .NET (which for the purpose of this discussion, includes the runtime library for .NET, or RTL for .NET), is a 100% .NET managed code equivalent of the Delphi VCL for Win32. Several features of the VCL for .NET are notable. For one thing, VCL for .NET is the largest, 100% managed collection of classes, types, and functions for the .NET framework outside the .NET framework class library itself. And is only available in Delphi 2005 (or its immediate predecessor, Delphi 8 for the Microsoft.NET Framework).

The second characteristic of the VCL for .NET is its remarkable compatibility with the Win32 versions of the VCL. In fact, you use this compatibility to migrate your Win32 Delphi code to .NET with little or no effort.

There are several updates to the VCL for .NET added to Delphi 2005. These are described in the following sections.

Virtual Library Interfaces

Delphi provides extensive support for interoperability between Win32 and .NET applications, including COM interop through runtime callable wrappers (RCWs) and platform invoke (PInvoke). With Delphi 2005, this support has been advanced through the support for virtual library interfaces (VLI). Virtual library interfaces permit you to call routines in Win32 DLLs from your .NET applications much more easily than the mechanism provided by .NET's PInvoke.

Normally, managed code in the .NET framework can call routines in unmanaged libraries through the .NET platform invoke service, or PInvoke. With PInvoke, you import the exported routines of an unmanaged DLL by using the [DLLImport] attribute to identify the DLL in which the function is located, as well as other characteristics of the exported function.

There are several drawbacks to using PInvoke. First, using the [DLLImport] attribute you cannot resolve the DLL name or location (path) at runtime. Second, if the specified routine in the DLL cannot be loaded, for whatever reason, a runtime exception is raised.

Third, the [DLLImport] attribute is somewhat verbose and repetitive, especially when you have many routines that you are importing from a single DLL.

Consider the following two functions, which are implemented and exported from a Win32 DLL created using Win32 Delphi:

```
function ConvertCtoF(CentValue: Integer): Integer; stdcall;
function ConvertFtoC(FahrValue: Integer): Integer; stdcall;
```

A unit that imports these routines using PInvoke has, at a minimum, an implementation block that looks something like the following (assuming that these routines were exported from a DLL named Win32DLL.dll):

```
function ConvertCtoF; external;
[DllImport('Win32DLL.dll', CharSet = CharSet.Auto,
  EntryPoint = 'ConvertCtoF')]
function ConvertFtoC; external;
[DllImport('Win32DLL.dll', CharSet = CharSet.Auto,
  EntryPoint = 'ConvertFtoC')]
```

With virtual library interfaces, importing routines from an unmanaged DLL is easier, is less prone to raising exceptions, and permits your code to resolve the name and/or location of the DLL at runtime. There are three steps to importing one or more routines from an unmanaged DLL using virtual library interfaces. These are:

- Adding the Borland.Vcl.Win32 namespace to your uses clause
- Creating an interface declaration where each method in the interface maps to one of the routines exported from the DLL
- Calling the Supports function from the Borland.Vcl.Win32 unit, passing to it the name of the DLL (including an optional path if the DLL is not located in a location where Windows will find it), the Interface you created in the preceding step, and a variable of that interface type

If the Supports function determines that the methods of your interface map to functions exported from the named DLL, the variable you pass in the third parameter of the call to Supports will point to an object that implements the interface you passed in the second parameter. You can then use this object reference to execute the unmanaged routines of the DLL.

If one or more of the methods of the interface are not exported by the named DLL, or the named DLL does not exist or is somehow compromised, Supports returns a Boolean false without raising an exception.

Here is a sample interface that declares the two exported functions of the unmanaged DLL example used earlier in this section:

```
type
IWin32DLLInt = interface
  function ConvertCtoF(CentValue: Integer): Integer;
  function ConvertFtoC(FahrValue: Integer): Integer;
end;
```

If Win32DLL.dll is located in the mylib subdirectory of the application's executable, the following code returns an implementation of IMyWin32DLL, after which one of the methods (ConvertCtoF) of the returned object is executed:

```
var
  MyDLL: String;
  MyWin32DLL: IWin32DLLInt;
begin
MyDLL := ExtractFilePath(Application.ExeName) +
  '\mylib\Win32DLL.dll' ;
if not Supports(MyDLL, IWin32DLLInt, MyWin32DLL) then
  MessageBox.Show(self, 'Could not load Win32DLL.dll')
else
  NewInt := MyWin32DLL.ConvertCtoF(100);
```

Support for Partially Trusted Callers

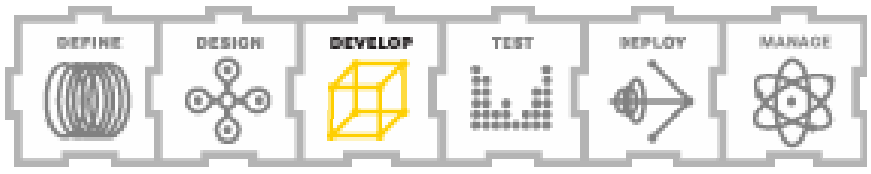
The VCL for .NET assemblies now support partially trusted callers. A partially trusted caller is an application that does not reside on the same workstation as a managed assembly that it calls.

For example, an .exe being executed from a network share or from a URL is a partially trusted caller. By default, the .NET security model prevents a partially trusted caller from invoking unmanaged code, such as that in the Windows API, unless that caller includes the appropriate declarations and checks.

The Delphi 2005 assemblies of the VCL for .NET now include the additional security declarations and checks that permit the VCL for .NET to be called from a partially trusted caller without violating .NET security.

The Delphi Compilers

Borland®
Delphi™ 2005



The Delphi Compilers

Borland compilers are legendary for their speed and compatibility, and this legacy continues with Delphi 2005. Actually, Delphi 2005 ships with three compilers. One of these compilers, the C# compiler, is licensed from Microsoft. Consequently, C# applications you build in Delphi 2005 generate the same intermediate language (IL) code as those built with Visual Studio.

The other two compilers are Delphi compilers, one for compiling traditional 32-bit Windows executables and the other for generating IL for the .NET Framework. Both of these compilers have received significant updates in the Delphi 2005 release.

This section begins with a discussion of features added to both the Win32 and the .NET versions of the Delphi compilers. Later in this section, you will learn about the new features that are specific to one or the other of these compilers.

Updates for Both Win32 and .NET Delphi Compilers

Several new features have been added to both of Delphi 2005's Delphi compilers. Thes most significant of these include the new **for…in** loop and Unicode support. These new compiler features are described in the following sections.

The For…In Loop

The Delphi language has been updated to include a new looping control structure similar to the C# **foreach** keyword. In Delphi, this new loop is referred to as a **for…in** loop. Unlike traditional for loops in Delphi, the **for…in** loop does not require an ordinal control variable. Instead, the **for…in** loop systematically retrieves a reference to the next object in a collection of like objects.

For example, the following code segment can be used to iterate through the DataRows of a DataTable's Rows property (this property is of the type DataRowCollection):

```
var
  Row: DataRow;
begin
//…
for Row in MyDataTable.Rows do
  ListBox1.Items.Add(Row[0].ToString);
```

For the .NET Delphi compiler, **for…in** can be used with any object that satisfies at least one of the following conditions: it implements the IEnumerable interface, has a public GetEnuermerator function, or is an array, set, or string. For the Win32 compiler, **for…in** can be used with any class that has a public GetEnumerator function, or is an array, a set, or a string. Classes that implement a GetEnumerator function include TList, TCollection, TStrings, TMenuItem, TFields, to name a few.

Support for Unicode and UTF8 Formats

Both of Delphi's compilers can now compile UTF8 and Unicode source files. Previously, only ANSI source files were supported. For the Delphi for .NET compiler, this feature supports CLS (common language specification) standard Unicode identifiers in both metadata and in source code.

The Delphi for .NET Compiler

Borland's Delphi for .NET compiler made its first debut with the release of Delphi 8 for the Microsoft .NET Framework. In addition to the updates listed in the preceding section, this compiler has received a number of updates that apply specifically to .NET applications. These include a revision to how namespaces are created and managed, forward-declared record types, and support for weak packaging in VCL for .NET applications. The updates to the Delphi for .NET compiler are described in the following sections.

Delphi Code and Namespaces

The biggest change to the .NET compiler is in how it generates namespaces for the symbols defined in your units. Under the previous version of the compiler, the unit name was the namespace.

For some developers, particularly those accustomed to using classes defined in C#, the namespaces created by Delphi appeared awkward. Specifically, these namespaces revealed the physical structure of the underlying code, which is irrelevant from the perspective of the person using your classes, and can be distracting.

The Delphi 2005 compiler takes a new approach to namespace generation, allowing multiple units, and even multiple applications, to contribute to a common namespace, if desired. At the same time, it is just as easy to make each unit contribute to a separate namespace.

Here is how it works. If your unit names do not use dot notation, the unit name is the namespace. This is how it worked before.

If a unit includes a multipart name, using dot notation, the namespace is defined by dropping the last part of the unit name. For example, if a unit has the name YourCompany.Data.Unit1, the classes within that unit will reside in the YourCompany.Data namespace. Classes that appear in the YourCompany.Data.Unit2 and YourCompany.Data.Unit3 units will be in the YourCompany.Data namespace as well.

Global variables, constants, functions, and procedures declared in Delphi code represent something of a challenge, in that .NET requires all declarations to be associated with a class. Therefore, the global symbols of a Delphi unit named YourCompany.Data.Unit1 are implemented in .NET metadata as members of a class named Unit1 within the namespace YourCompany.Data.Units.

How Delphi symbols appear in .NET metadata has no effect on your Delphi source code. You only need to consider how your Delphi code will appear in the .NET metadata for the portion of your code that you want developers using other .NET languages to use. In general, you should avoid using global variables, global constants, or global procedures and functions when writing Delphi code that you intend to be used by other .NET languages.

Support for Weak Packaging in VCL for .NET Applications

A runtime package in the VCL for .NET is a managed .NET assembly — it contains declarations that the application can load and use at runtime. Under normal circumstances, if

you compile a VCL for .NET application to use a runtime package, you are required to deploy that package, just as you are required to ensure the deployment of all assemblies (DLLs) that are referenced in your application.

Weak packaging of a unit addresses a problem that arises when a runtime package contains one or more units that statically link to an external DLL, in particular, a DLL that is not commonly available. Under normal conditions, this situation requires that you deploy both the runtime package and the DLL.

Consider the Microsoft DLL PenWin.dll for pen device input, which is not distributed with Microsoft operating systems. The PenWin unit in Delphi statically links to the DLL PenWin.dll. If your unit uses PenWin, and includes calls to one or more functions in the statically linked PenWin.dll, adding your unit to a runtime package without weak packaging would require that the PenWin.dll be available from any application that loaded that runtime package. By making this unit weakly packaged, only applications that actually call PenWin.dll functions will require PenWin.dll.

Weak packing permits an application to link a non-packaged version of the unit into the executable instead of using the runtime package that contains this unit. As a result, applications that need the features of the weakly packaged unit will link the non-packaged version of the unit (that stored in the compiler-generated DCPIL file), and as a result, require the DLL. Applications that do not use the unit will not require the DLL, even if they are compiled to use the package that contains the weakly packaged unit.

Weak packaging has been available for some time in the Delphi Win32 compiler. Weakly packaged unit semantics are now supported by the Delphi for .NET compiler.

Forward Declared Record Types

Record types can now be forward declared in Delphi VCL for .NET and FCL applications. A forward declared record instructs the compiler to recognize the record as a valid type, even though its formal declaration appears later in the same type block.

Forward declared record types permit two type declarations, specifically records, classes, and interfaces, appearing in the same type block to reference one another in their member fields, properties, or methods. You create a forward declared record type by declaring the record type symbol but omitting the record's field lists.

The Delphi Win32 Compiler

The degree of compatibility between the Delphi Win32 and .NET compilers is one of the truly remarkable Delphi 2005 features. This compatibility permits single projects to be compiled as true Win32 applications and then effortlessly migrated to 100% .NET managed code applications. In many cases, a single set of source files can be compiled by both the Win32 and the .NET versions of the Delphi compiler. No other development environment lets you do this as easily.

Equally compelling for developers is Borland's continued support for the Win32 platform with the most modern IDE on the market. While Borland is committed to the .NET platform as the future of Windows development, Borland also knows that the majority of desktop developers maintain applications on the Win32 platform, and Borland is just as committed to providing those developers with the advanced features that they need.

Although the bulk of the enhancements to the Win32 compiler have already been described earlier in this guide (in the section "Updates for Both Win32 and .NET Compilers"), the following sections discuss some of the unique features added to the Delphi Win32 compiler in Delphi 2005.

Function Inlining

Function inlining is an operation performed by the Win32 compiler at compile time. When a function is inlined, the compiler replaces a call to the subroutine (a method, function, or procedure) with the compiled instructions defined within the subroutine. Function inlining can increase application performance by eliminating the overhead associated with function, procedure, and method calls.

There are two ways to influence whether the compiler will inline a function or not. One way is to include the **inline** directive in the function, procedure, or method declaration. This directive is a request to the compiler to consider whether or not to inline the function. If inlining has not been disabled, and the compiler determines that the function can be safely inlined, the inlining will be performed.

The second way is to use the {$INLINE} compiler directive. This directive can be passed with one of three parameters, ON, OFF, and AUTO. With the ON parameter, the default, the compiler will inline functions declared using the **inline** directive, whenever the compiler determines that inlining is safe. No inlining takes place when you specify the OFF parameter.

When you use the {$INLINE} compiler directive with the AUTO parameter, the compiler attempts to inline, if possible, any small function — one whose code size is roughly 32 bytes or less.

While function inlining can produce performance improvements, Borland is quick to note that it should be applied judiciously, and does not recommend using the AUTO parameter with the {$INLINE} compiler directive. Inlining can produce larger executables, even some that are dramatically larger. Also, inlined functions do not always produce performance benefits. In some cases, inlining can actually reduce performance.

There are a number of conditions that prevent a subroutine from being inlined. For example, subroutines that include inlined assembly instructions cannot be inlined. Similarly, methods of a class that access one or more of that class's private members cannot be inlined into a method in another class.

Borland has applied the inline directive to some of the smaller routines in the VCL and RTL, where deemed appropriate. As a result, code that uses these routines will execute faster than before, but with slightly larger executables.

Support for Nested Types

A nested type is a type declaration inside another type declaration. The Delphi for .NET compiler already supports nested types. Delphi's Win32 compiler does now, too.

The following is an example of a class that contains a nested type. This example is taken from the Delphi 2005 Help, and can be found under the heading Nested Type Declarations.

```
type
  TOuterClass = class
  strict private
    myField: Integer;
  public
    type
      TInnerClass = class
    public
      myInnerField: Integer;
      procedure innerProc;
    end;
    procedure outerProc;
    end;
```

Nested Type Constants in Class Declarations

Nested type constants are constant class member declarations inside of a class type declaration. Nested type constants are somewhat similar to class functions, in that they can be referenced using a class reference without an instance of the class. Unlike class functions, however, nested type constants always return a constant value.

Nested type constants are already available for your .NET projects. Now you can use them in your Win32 applications as well. Nested type constants can be of any simple type, such as ordinal, real, and String. You cannot declare a nested constant to be a value type, such as TDateTime.

The following is an example of a class that includes a nested type constant declaration:

```
type
  TTemperatureConverter = class(TObject)
    public
      const AbsoluteZero = -273;
```

```
      procedure ConvertFtoC(Temp: Integer): Integer;
//…
```

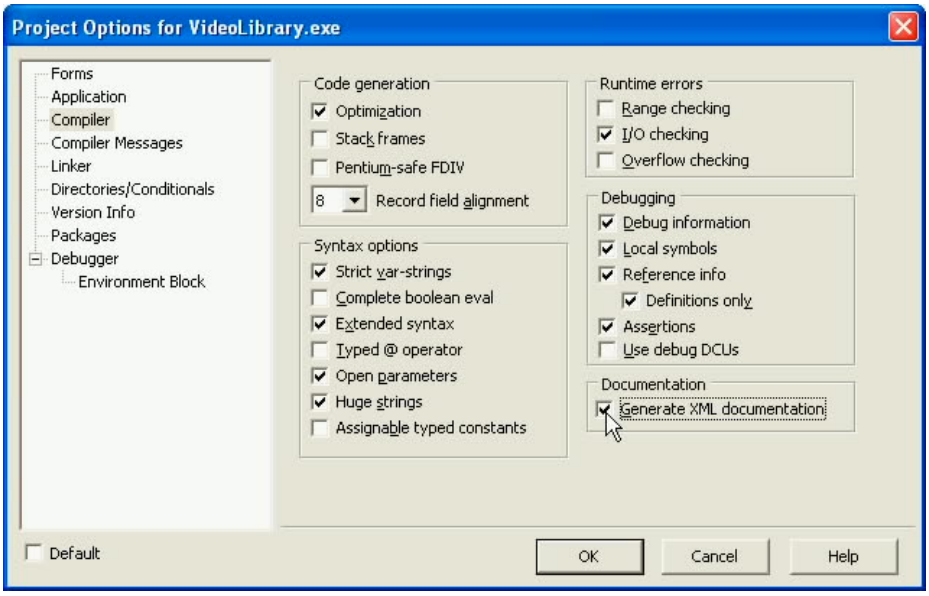
Support for Pentium 4 SSE3 and SSE2 Instruction Op Codes and Data Types

If you need to get close to the silicon, Delphi's Win32 compiler now permits you to include Pentium 4 SSE3 and SSE2 op codes and data types in your inline assembly routines.

XML Document Generation

XML document generation was introduced in the Delphi 8 for .NET and C#Builder compilers. You can now generate XML documentation files for your Win32 source code.

To enable XML Doc generation, enable the Generate XML Documentation check box on the Compiler page of the Project Options dialog box. You display the Project Options dialog box by selecting Project | Options from the Delphi 2005 main menu.

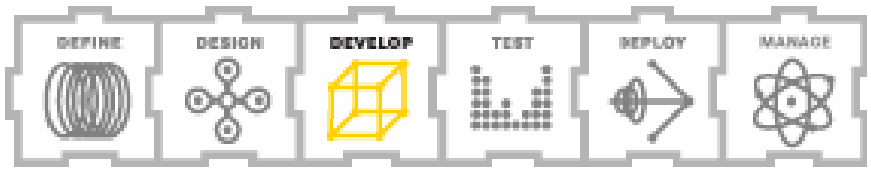


When Generate XML Documentation is enabled, the compiler produces one XML file for each of your source files. This file has the same name as the source file, but with the .xml extension. If you have included custom XML Documentation comments in your source files, these will be inserted into the generated XML file.

The XML files generated when you compile with Generate XML documentation enabled can be used with widely available documentation generating tools. Alternatively, you can write your own XML parser to use this information any way you see fit.

The Delphi Debuggers

Borland®
Delphi™ 2005



The Delphi Debuggers

A good debugger is one of the essential tools for successful software engineering. Whether it is used to help you learn the values of your various variables and objects as your code executes, or to inspect the contents of your application's stack, a debugger lets you do the nearly impossible — peer into the black box and make sense of what's going on.

This section provides you with insight into Delphi 2005's support for debugging your Win32 and .NET applications.

Multiple Debugger Support

Delphi 2005 doesn't just have a world-class debugger — it has two. One of these is for your .NET applications that you have compiled to IL, and the other is for your Win32 applications that you've compiled to machine language.

Delphi 2005 selects which of these debuggers to use based on the type of compiler that created your executable. For example, if you are debugging an ASP.NET Web application, a Windows Forms application, or a VCL for .NET application, Delphi 2005 uses the Borland .NET Debugger. By comparison, if you are debugging a VCL client/server application, a COM (component object model) server, or a traditional Win32 DLL, Delphi 2005 uses the Borland Win32 Debugger.

Just as Borland provides you with a consistent set of features when it comes to compiling, Borland's debuggers do a remarkable job of giving you a rich, dependable, and consistent set of tools for debugging your applications, whether you are compiling for .NET, Win32, or both. For example, each of Delphi 2005's debuggers permits you to set breakpoints, view the call stack, change the values of variables and objects, access local variable values, switch between your application's current threads, view CPU (central processor unit) data, examine the event log, as well as access the list of loaded modules. You can even use these debuggers to attach to existing processes, giving you insight into how they are functioning.

While the features offered by these two debuggers are consistent, they are not identical. Specifically, each debugger provides you with options appropriate for the associated executable.

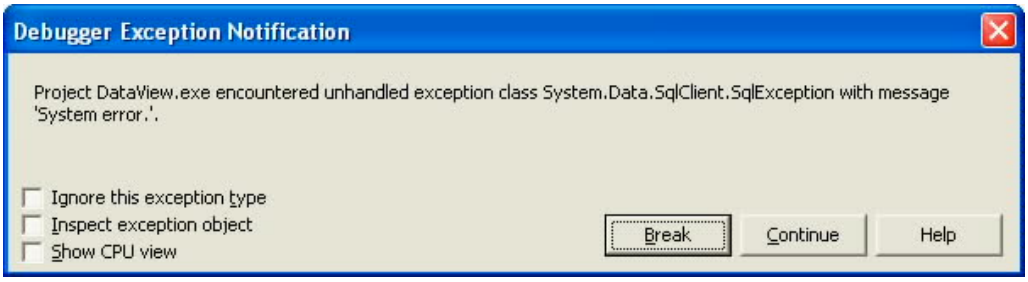
For example, with Win32 applications you can create Data breakpoints, breakpoints that trigger when the data stored in a particular memory address changes. Data breakpoints don't make sense in the .NET world, since the physical address in which data is stored cannot be predicted.

On the other hand, the CPU window displayed by the .NET debugger can include the IL (intermediate language) the .NET compiler emitted. Win32 compilers don't generate IL, so this feature does not apply to Win32 executables.

The following sections provide you with information about new features that appear in the debuggers for Delphi 2005.

Exception Dialog Enhancements

An exception is an error generated at runtime by your application. Unless you have specifically configured your debugger to ignore the exception (or have disabled the debugger), several things happen when an exception occurs when you are running an application from within the Delphi 2005 IDE: your program stops executing, the appropriate debugger is loaded, and the Debugger Exception Notification dialog box is shown. An example of the exception dialog box is shown in the following figure.

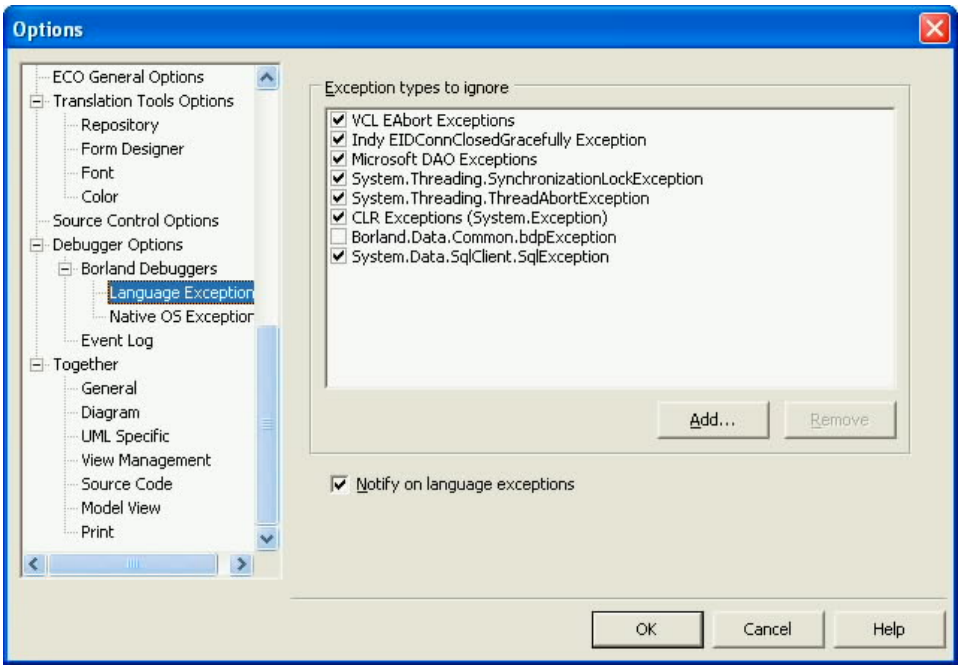


The Debugger Exception Notification dialog box in Delphi 2005 includes a number of new features. You can choose whether to stop your program's execution temporarily or close the

debugger and continue executing the program using the Break and Continue buttons, respectively, located in the lower-right corner of this dialog box.

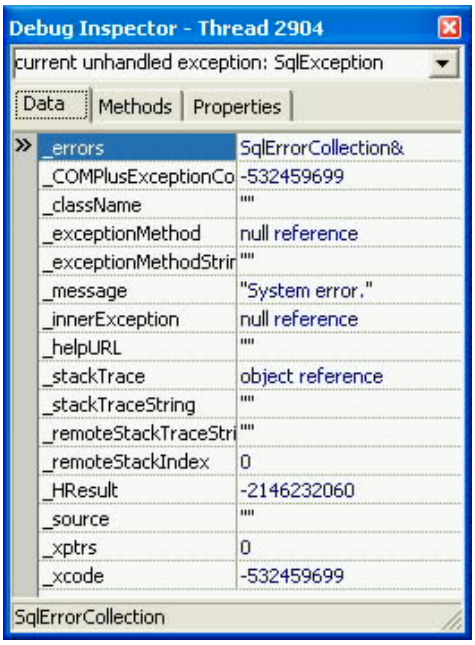
In addition to these options, you may also see one or more of the check boxes that appear on the left side of the dialog box in the preceding figure. If you click the Ignore this exception type check box, the class of exception that occurred is added to the Exception types to ignore list on the Language Exception tab of the Options dialog box. From that point on, this particular exception class, as well as any class that descends from it, will no longer load the integrated debugger.

If you later want to restore the default behavior of having this exception load the integrated debugger, either uncheck the check box next to the corresponding exception in the Exception types to ignore list, or select the exception and click the Remove button. (You display the Options dialog box by selecting Tools | Options from the main menu.)



If you check the Inspect exception object check box on the Debugger Exception Notification dialog box, and then click the Break button, the Debug Inspector becomes available, as shown in the following figure. The Debug Inspector allows you to view, and drill down into the

instance of the raised exception. In this case, detailed information about the exception can be discovered by double-clicking the _errors property of the SqlException object, and then inspecting the SqlErrorCollection& which contains the detailed information about the problem that was encountered.

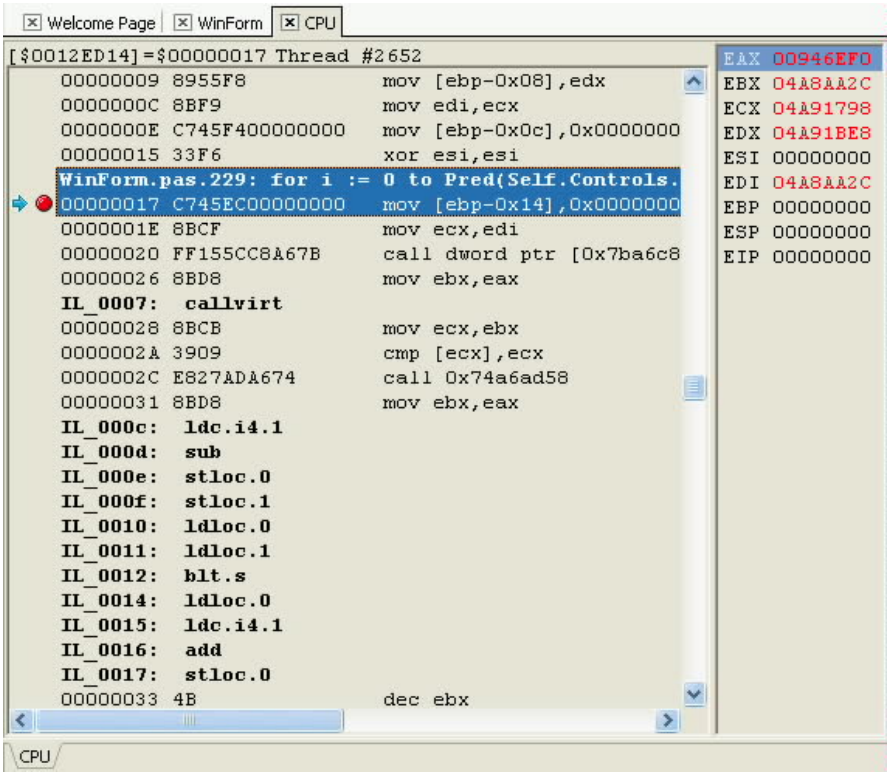


If the raised exception does not correspond to a source location, the Show CPU view check box is available on the Debugger Exception Notification dialog box. Checking this check box, then clicking Break, loads the CPU window, displaying the disassembled view of the executing code, the CPU registers, and possibly other information, depending on the debugger.

The Disassembled View

Speaking of the disassembled view, Borland has introduced updates to both the Win32 and .NET versions of this part of the CPU window. For a .NET executable, you now have the option of viewing the generated IL, the source code that compiled to the IL, or both.

An example of the CPU view displayed by the Borland .NET Debugger is shown in the following figure. This particular CPU view is associated with a source breakpoint. The highlighted statement in the disassembled pane (the left pane) is the Delphi source on which the breakpoint was placed. Beneath this code, you can see both the IL instructions that were emitted by the Delphi for .NET compiler, as well as the resulting assembly language instructions that the JIT compiler produced.



You control whether IL and/or source code appears in the disassembled pane of the CPU view using the disassembled pane's context menu. When Mixed Source is checked, source code is displayed. When Mixed IL Code is checked, IL is displayed. Turn both of these options off to view only the code generated by the JIT compiler.

Breakpoints

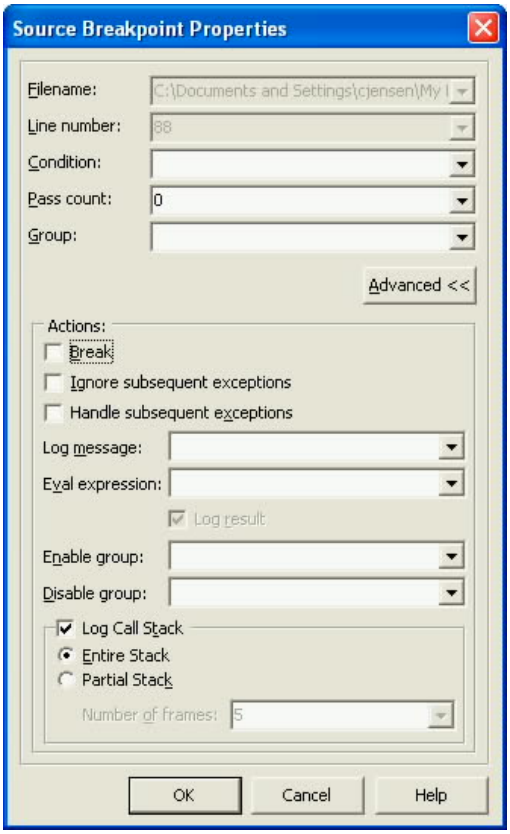
Breakpoints are event-driven markers that can be configured to cause the integrated debugger to perform a task. In most cases, this task is to temporarily stop executing your code and load the integrated debugger, permitting you to examine features of the execution environment. On the other hand, the task might not include stopping your code's execution, but instead perform some action, such as writing a message to the event log.

Delphi 2005 introduces two new features that specifically apply to breakpoints. These are described in the following sections.

The Log Call Stack Breakpoint Option

Source, address, and data breakpoints can now be configured to write the call stack to the event log. The call stack stores the current methods, functions, and procedures in the call chain, in the order in which they were entered. Breakpoints that write the call stack to the event log permit you to more easily track and document the events that lead to your code's execution.

To write call stack information to the event log, enable the Log Call Stack check box. Use the available radio buttons to configure the breakpoint to either write the entire call stack to the event log, or only a specific number of frames.

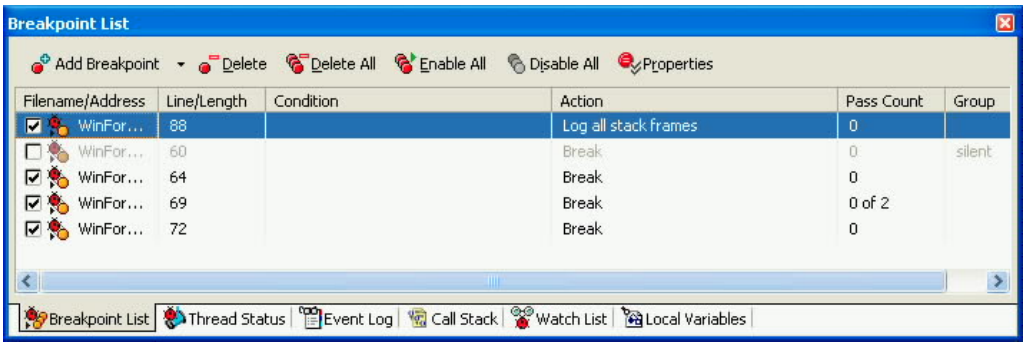


Typically, when you write the call stack to the event log, you do not need the breakpoint to load the integrated debugger. If that is the case, make sure that the Break check box is left unchecked for this breakpoint.

Breakpoint Dialog Box Updates

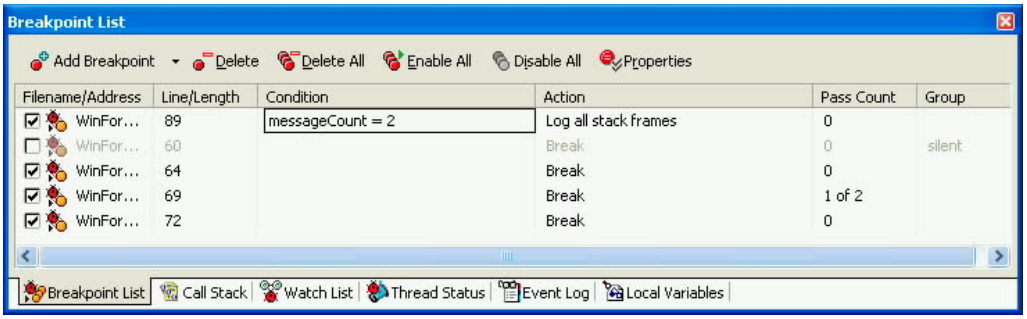
The Breakpoint dialog box has received several updates in this release. First, a new toolbar is available, allowing you to more easily enable, disable, remove, and configure your

breakpoints.



The Breakpoint dialog box has also been upgraded to permit in-place editing of a number of breakpoint properties, without having to view a particular breakpoint's Breakpoint Properties dialog box. Using the Breakpoint dialog box, you can directly edit the Enabled, Condition, Pass Count, and Group properties of individual breakpoints.

The following figure shows the Condition property of a breakpoint being edited using the Breakpoint dialog box.

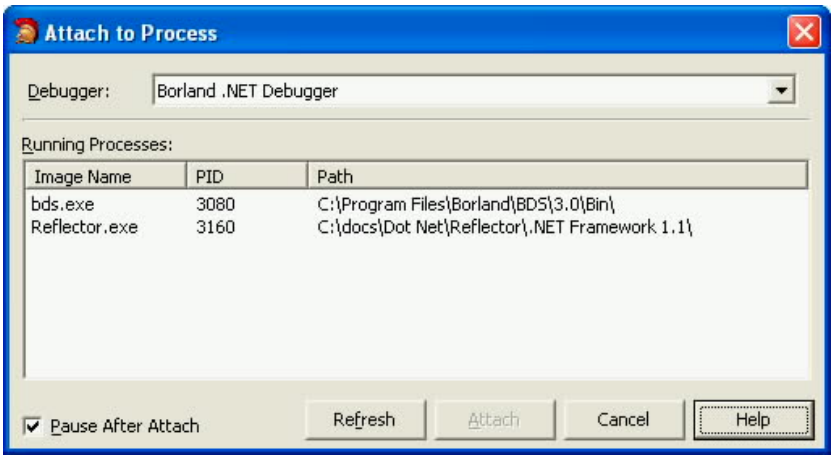


Updated Attach to/Detach from Process

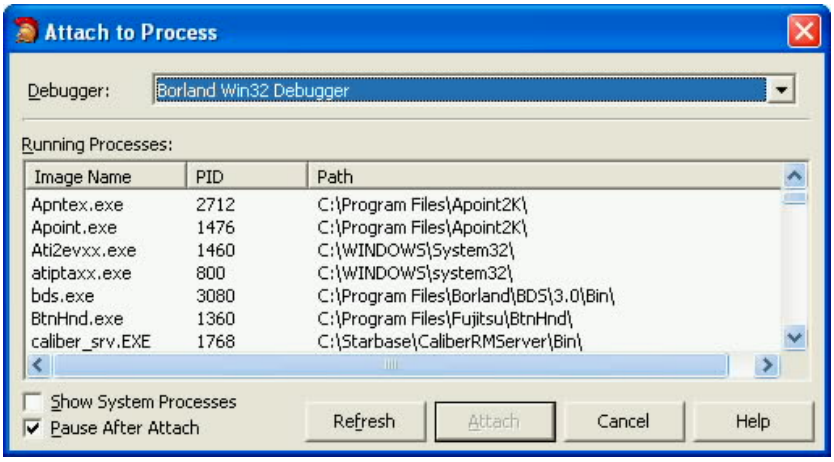
Previous versions of Borland compilers have permitted you to attach to a running process. Once attached to a process, you can use the debugger's features to inspect the process execution environment.

Attaching to a running process is even more powerful in Delphi 2005. For starters, when you select Run | Attach to Process, Delphi 2005 asks you to select which debugger to use to attach

to the process. If you select the Borland .NET Debugger, only processes hosted by the CLR (common language runtime) are displayed for your selection.



If you select the Borland Win32 Debugger, traditional Win32 processes are shown.

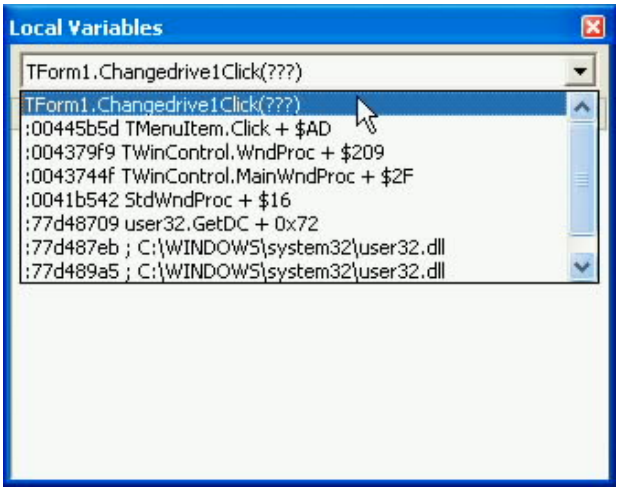


Also new is the option to detach from a process. If you have previously used one of the Borland debuggers to attach to a process, select Run | Detach from Process from Delphi 2005's main menu to detach from the process.

Evaluator Frame Support for Win32 Local Variables

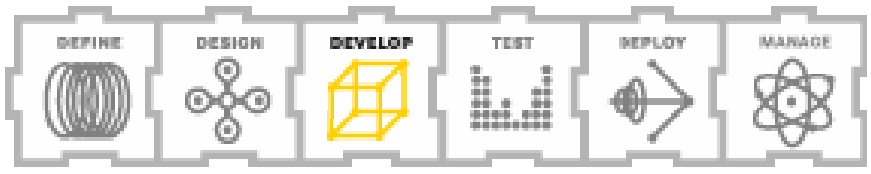
A popular debugging feature in Delphi 8 and C#Builder is the capability to select a particular frame from the call stack using the Local Variables dialog box. This feature is now available for the Borland Win32 Debugger.

With the Borland Win32 Debugger loaded, view the Local Variables dialog box. (If this dialog box is not already visible, select View | Debug Windows | Local Variables, or press Ctrl-Alt-L, to display it.) Initially, the values of variables local to the current function that the debugger is in are shown. To view local variables in one of the methods earlier in the call chain, select the method name from the drop-down menu.



Database Development

Borland®
Delphi™ 2005



Database Development

Delphi has long been considered the leading environment for database development. Currently, Delphi 2005 provides you with more data access options than any other environment.

For Win32 development, in addition to a number of industry standard data access mechanisms, such as ODBC (open database connectivity) and Ole Db Providers, developers have a wide range of Borland technologies that they can employ, including the BDE (Borland Database Engine), dbExpress, IBExpress (InterBase® Express), dbGo™ for ADO, MyBase (ClientDataSet), and DataSnap; Borland's multi-tier, distributed database environment.

Delphi for .NET developers can use the same technologies as Delphi Win32 developers. The .NET implementation of the Win32 data access mechanisms uses what Borland calls its compatibility data access technologies. These are all found in VCL for .NET

In addition, both Delphi for .NET and C# developers can access their data through ADO.NET, the data access framework of the FCL. Borland also provides an advanced custom data provider for ADO.NET for both Delphi for .NET and C# developers. This technology, which is called Borland Data Providers, or BDP for ADO.NET, offers many enhancements and extensions to ADO.NET, including live data views at design time, useful component designers, greater portability between underlying databases, and more.

What's especially impressive about Delphi 2005 is that Borland has added significant new database functionality in addition to the extensive features available in Delphi 8 for .NET and C#Builder. These additions and enhancements are described in the following sections.

RAD for ADO.NET

ADO.NET is the portion of the .NET framework associated with database development. While ADO.NET is very powerful, it fails to provide the design time ease-of-use Delphi

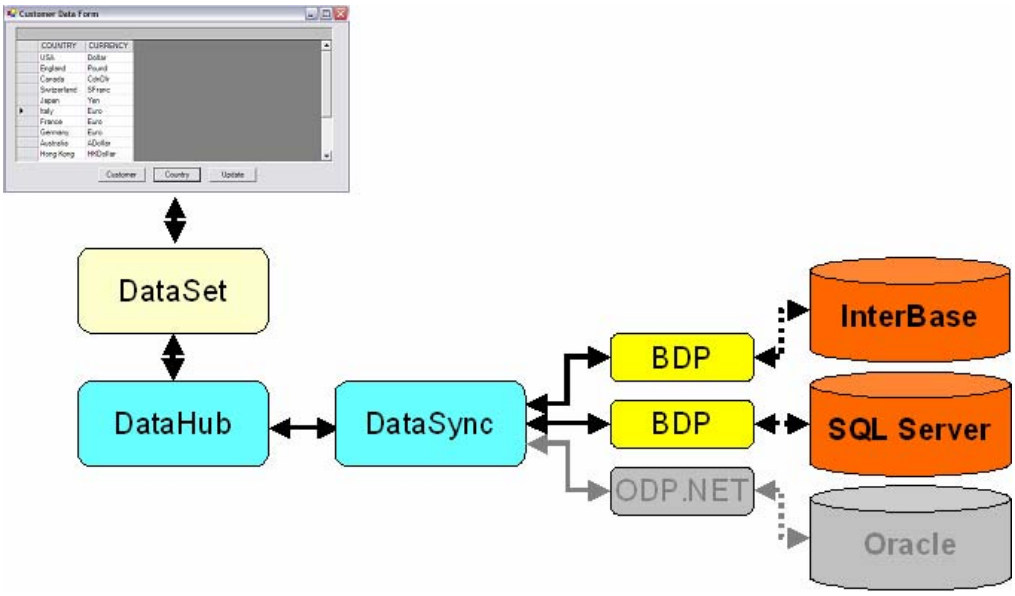
developers expect. RAD for .NET is Borland's answer, bringing the convenience and speed of Delphi database development to the world of ADO.NET.

RAD for ADO.NET simplifies the process of using ADO.NET from within your applications in two distinct ways. First, the DataSync and DataHub components provide a flexible provider/resolver mechanism that uses industry-standard ADO.NET data providers for data access. Second, the RemoteServer and RemoteConnection components permit you to extend these capabilities to a distributed environment. These technologies, and the components that implement them, are described in the following sections.

Providing and Resolving with DataSync and DataHub

Delphi 2005 introduces two new provider/resolver components that simplify how you work with your ADO.NET-related data access objects: DataSync and DataHub. You can use these components with any ADO.NET data providers to provide design-time views of your data, simplify data access, as well as apply updates back to your underlying database.

The relationship between the DataSync and DataHub components and the traditional classes of ADO.NET development is shown in the following figure. Here the DataSync and DataHub components mediate between the ADO.NET DataSet and IDbConnection classes to provide services lacking in ADO.NET alone. These services include live, design time views of data, the management of multiple database connections, as well as flexible and optimized data resolution services.



In addition, when used with Borland's new data remoting components, DataSync and DataHub simplify the process of creating distributed applications in the .NET framework. The data remoting components are discussed later in this section.

DataSync

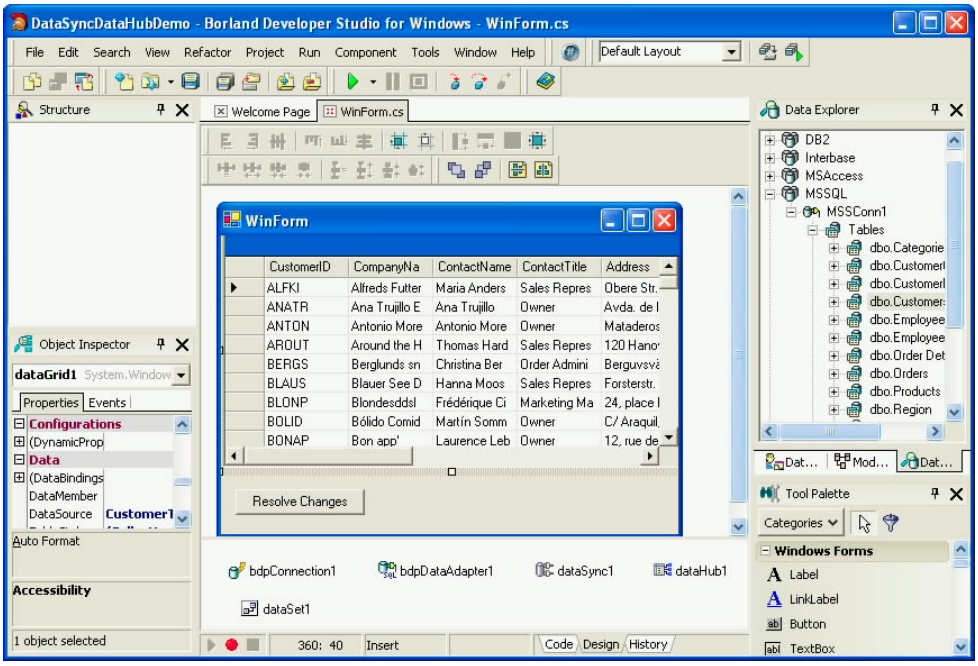
The DataSync component maintains a list of data providers, that is, classes that descend from DbDataAdapter. For each data adapter, the DataSync keeps track of the provider name, the name of the DataTable that the DataSync will create for the provider, as well as how changes to the DataTable will be applied to the underlying database.

That the DataSync relies on descendants of DbDataAdapter means that it can work with any data provider, not just the Borland Data Providers. As a result, you can use a DataSync with classes such as SqlDataAdapter and OdbcDataAdapter, which are included in the FCL, as well as data adapters from third-party vendors, such as IBM.Data.DB2 and Oracle.Data.Provider.

DataHub

You use the DataHub component in conjunction with a DataSync to feed data from the DataSync's data adapters to a DataSet, as well as initiate the resolution of changes back to the underlying database. Importantly, the DataHub can be activated at design time, which means that the combination of a DataSync and DataHub provide you with live data views at design time, a feature that is otherwise unavailable from non-BDP data adapters.

The following figure shows a C# project in which a DataSync and DataHub are used to populate a DataSet at design time. The DataGrid on the form shown in the designer is displaying the data obtained through the DataSync/DataHub combination.



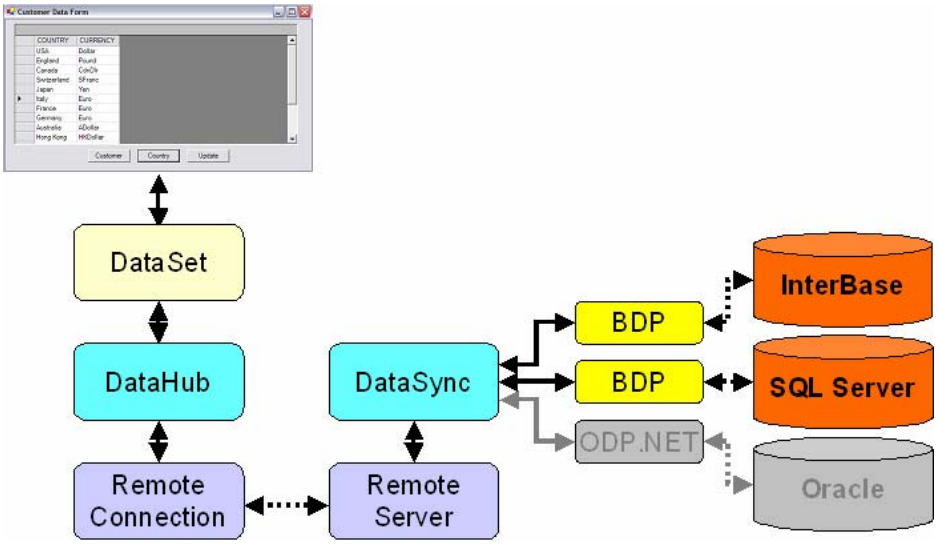
Another important feature of a DataHub is that it provides a single point of control for applying changes back to the underlying databases. Simply call the DataHub's ApplyChanges method, and it communicates to the DataSync, which responds by generating and executing the appropriate queries, based on the changes found in the associated DataTables. In the project shown in the preceding figure, the single line of code that is associated with the Click event of the button whose caption reads Resolve Changes is shown here:

```
dataHub1.ApplyChanges();
```

Data Remoting with RemoteServer and RemoteConnection

The .NET framework provides extensive support for working with remote objects through its .NET remoting services. One of the more practical applications of this technology is for implementing distributed database applications where DataSets in one process are accessed from applications in another, even when the applications are on separate computers on the Internet. However, .NET remoting is a general service, which means that using it to work with remote DataSets often requires a lot of custom code.

Delphi 2005 makes working with remote data easy with two new components that encapsulate .NET remoting services, permitting you to effortlessly work with DataSync and DataHub components in a distributed environment. These components, RemoteServer and RemoteConnection, permit you to build applications where the DataSync and DbDataAdapters reside on one machine, and the DataHub and its associated DataSet component reside on another. How RemoteServer and RemoteConnection extend the capabilities DataSync and DataHub is depicted in the following diagram.



RemoteServer

The RemoteServer component permits you to publish DataSync objects in one process to applications using a RemoteConnection component in another process. The RemoteServer and RemoteConnection components can communicate using either HTTP or TCP.

When you place a RemoteServer component into a project, you set its DataSync property to the DataSync instance containing the providers that you want to expose. You also set its ChannelType (Http or Tcp), Port to listen on, and URI (the specific resource that a client requests over the specified port).

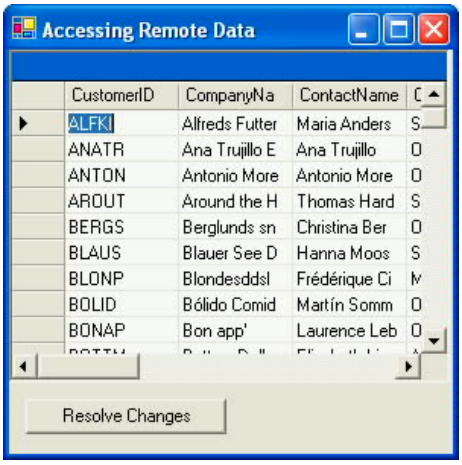
RemoteConnection

You use the RemoteConnection component in an application to obtain data through a remote DataSync. After placing a RemoteConnection, you specify the ChannelType, Port, URL, and URI that identifies where your remote server resides. You then set the RemoteConnection's ProviderType property to point to a particular provider on the remote server.

Once the RemoteConnection object is configured, you connect a DataHub in your client application to the RemoteConnection. This provides the DataHub with access to the DataSync on the server to which the RemoteConnection is attached.

From this point on, you configure and use the DataHub just as you would if the DataSync were in the same process. The RemoteConnection and RemoteServer objects use .NET remoting to transparently move the data between the remote DataSync and the local DataHub.

The following figure shows a DataGrid that displays data obtained through a remote DataSync. It is interesting to note that this client application was built using Delphi, while the server was built using C#. You could have just as easily done this the other way around. On the other hand, both the client and the server could have been built using the same language.



Borland Data Provider for ADO.NET

The Borland Data Provider for ADO.NET is a set of concrete classes and associated types that implement the data access interfaces of ADO.NET. These classes, which are part of what Borland calls BDP for ADO.NET, provide you with a powerful and portable solution for connecting to a wide variety of different databases at the same time extending the already substantial capabilities of ADO.NET.

The Borland Data Provider for ADO.NET also includes powerful component editors that you use to work with the BDP data access classes, as well as additional classes that specifically bind to BDP, such as DataSync and DataHub, which provide data services that go well beyond those found in ADO.NET alone.

Delphi 2005 includes a number of updates to BDP for ADO.NET. For example, BDP now supports connections to Sybase databases, as well as support for Oracle packages.

There is also a new BDP for ADO.NET component — BdpCopyTable. This component provides your applications with the ability to copy a table and its primary index from one supported BDP for ADO.NET provider to another, giving you the runtime equivalent of the new Copy Table feature in the BDP Data Explorer (which is described in the following section).

There is another update to BDP for ADO.NET that is not so obvious. BDP for ADO.NET has introduced additional interfaces for BDP providers that expose schema retrieval methods. BDP uses these interface implementations to discover information about the structure of database objects beyond what is currently supported in ADO.NET alone.

These behind-the-scene interfaces are responsible for BDP's ability to copy tables, discover stored procedure parameters, and migrate data. These features are available to you at design time through the newly enhanced Data Explorer.

The BDP Data Explorer

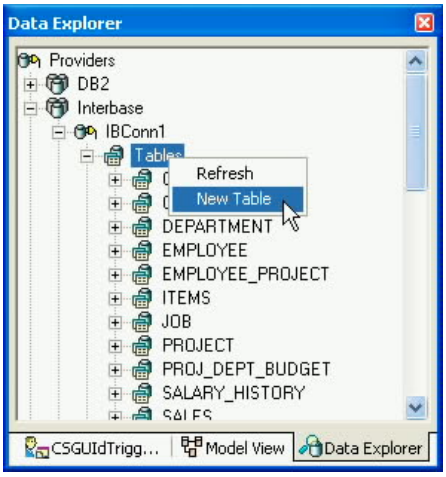
The Data Explorer allows you to work with ADO.NET at design time through BDP for ADO.NET supported databases, such as Oracle®, DB2®, MS SQL™ Server, InterBase, and MS Access. With the Data Explorer, you can inspect database objects, such as tables, views, and stored procedures from within the Delphi 2005 IDE. The Data Explorer also lets you easily create and configure BDP-related data access components, such as BdpConnections and BdpDataAdapters.

The Data Explorer has received a significant upgrade in Delphi 2005. Features now available from the Data Explorer permit you to create, alter, and drop database tables, test stored procedures, and copy data between BDP for ADO.NET-supported databases. Each of these features is discussed in the following sections.

Managing Tables

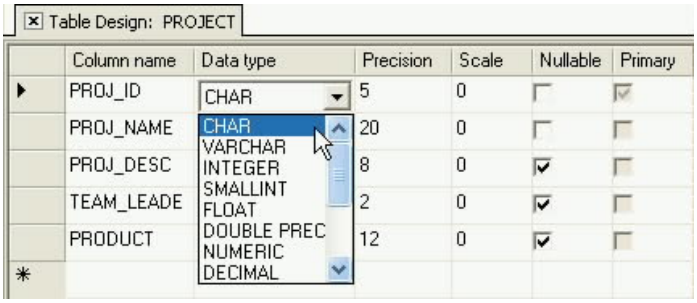
You can use the Data Explorer to create, modify, and delete database tables without having to leave the Delphi 2005 IDE. These capabilities are made available through BDP for ADO.NET's schema discovery services. These services, which debut in Delphi 2005, extend the already powerful capabilities of ADO.NET.

For example, to create a new table, open a connection in the Data Explorer. Next, right-click the Tables node and select New Table.



You use the Table Designer in Delphi 2005 to define the structure of your new table. You use this same designer when you want to modify an existing structure.

To modify a table's structure, right-click the table name under the Tables node in an open connection, and select Alter. (To delete a table, you select Drop from this same context menu.) The following figure shows a table named PROJECT being altered in the Table Designer.



You can see from the preceding figure that you can define or change the data type of a field in a table using a drop-down list of the applicable data types. Once again, this information is available through BDP for ADO.NET's schema discovery capabilities.

Data Migration

You can use the Data Explorer to migrate tables from one supported BDP for ADO.NET database to another simply by copying and pasting. When you copy a table, you copy the table's structure, data, and primary indexes.

To copy a table, right-click the table in the Data Explorer and select Copy. Next, select the connection into which you want to paste the table, right-click and select Paste. Delphi 2005 will respond with the New Table Name dialog box, as shown in the following figure.

Enter the name for the copied table and click OK.

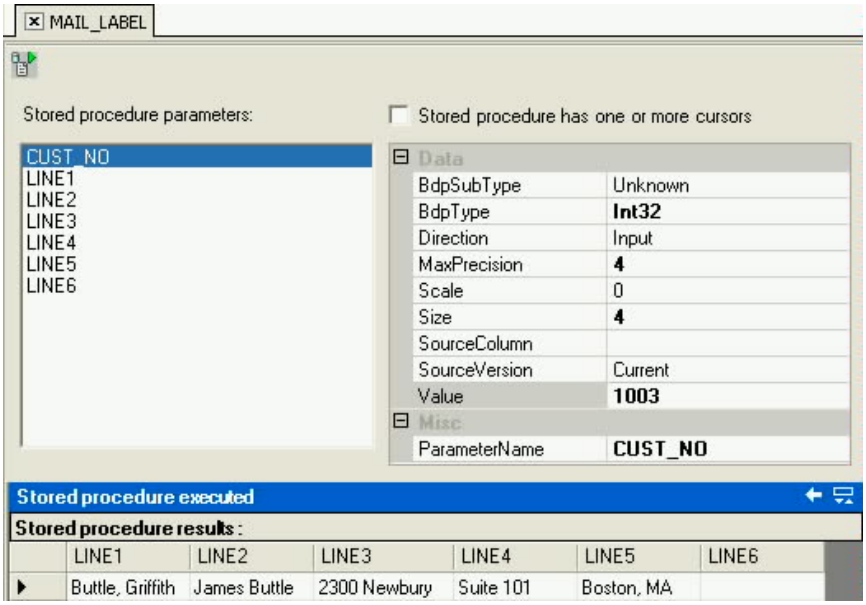
Delphi 2005 also includes components you can use in your applications to provide these same data migration capabilities to your users.

Testing Stored Procedures

Another important enhancement to the Data Explorer is its support for testing stored procedures. To test a stored procedure, right-click the name of the stored procedure that you want to test in the Data Explorer and select View Parameters.

Delphi 2005 examines the stored procedure's parameters, determining each parameter's data type, direction, and name. You can then test the stored procedure by assigning a value to each input parameter and clicking the Execute button, which appears in the top-left corner of the stored procedure pane. After executing the stored procedure, Delphi 2005 displays the output parameters in a data grid beneath the stored procedure pane (given that the stored procedure has output parameters).

The following figure shows the stored procedure pane with a result set, which displays the mailing label lines for customer number 1003.



Creating Reports in Delphi 2005

Reports are the tools that you use to turn data into information. Delphi 2005 includes two powerful reporting tools for you to use. For your .NET applications written in either Delphi or C#, Delphi 2005 includes Crystal Reports for Borland Delphi from Business Objects. For your Delphi VCL applications, both VCL Forms (Win32) and VCL for .NET, Delphi 2005 includes Rave Reports Borland Edition from Nevrona Designs.

Added VCL for .NET Data Access Components

Delphi 8 for the .NET Framework was notable for its extensive support for data access mechanisms compatible with Win32 Delphi. With Delphi 2005, this support has been extended further.

One of the biggest additions is the support for dbGo for ADO. dbGo for ADO is a set of components that implement the standard VCL TDataSet interface through which you can

communicate to ActiveX Data Objects using installed Ole Db Providers. Delphi 2005 now includes the full compliment of dbGo for ADO components in VCL for .NET.

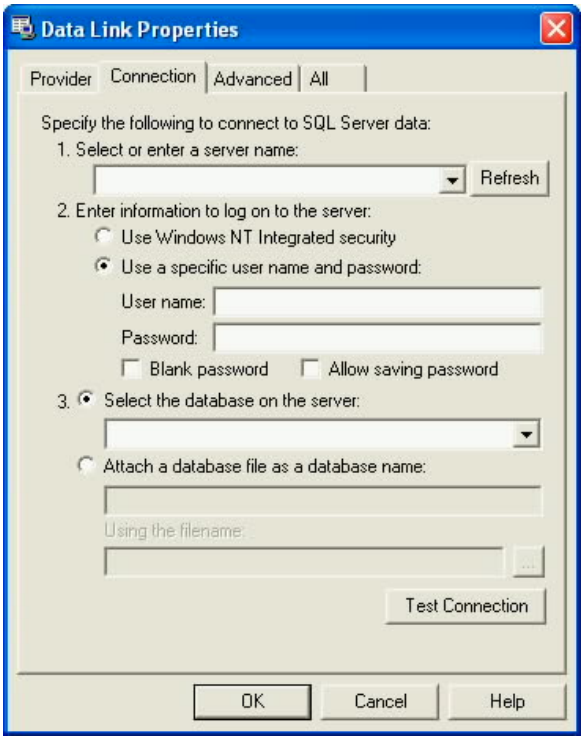
Other compatibility components that have now been added to VCL for .NET include the following: TStoredProc, TSimpleDataSet, TNestedDataSet, and TUpdateSql.

Additional compatibility components for DataSnap clients have also been added in Delphi 2005. DataSnap is Borland's multi-tier architecture for building thin clients and their associated application servers. These new VCL for .NET components include TConnectionBroker, TSharedConnection, and TLocalConnection.

ADO.NET Connection String Editor

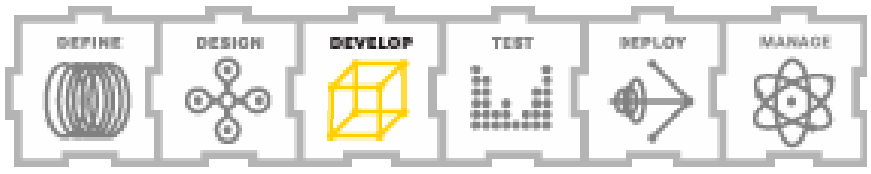
Delphi 2005 now provides you with an ADO.NET connection string editor for SqlConnection, OdbcConnection, and OleDbConnection components. (Previously, unless you were using BDP for ADO.NET, constructing your connection string for an ADO.NET connection generally involved referring to the documentation for your .NET data provider.)

When you need to configure one of these components in the Delphi 2005 IDE, select the ConnectionString property in the Object Inspector and click the ellipsis button to display the Connection String editor. The Connection String editor for a SqlConnection ConnectionString property is shown in the following figure.


Data Link Properties
Provider
Connection
Advanced
All
Specify the following to connect to SQL Server data:
1. Select or enter a server name:
Refresh
2. Enter information to log on to the server:
Use Windows NT Integrated security
Use a specific user name and password:
User name:
Password:
Blank password
Allow saving password
3.
Select the database on the server:
Attach a database file as a database name:
Using the filename:
Test Connection
OK
Cancel
Help

Web and Internet Development

Borland®
Delphi™ 2005



Web and Internet Development

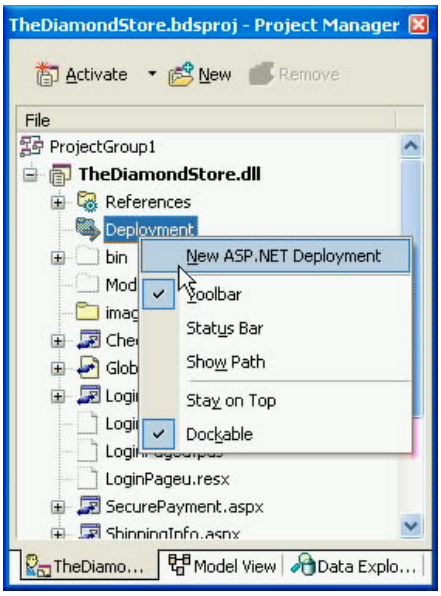
Delphi was one of the first IDEs to give you component-based, event driven tools for building dynamic Web sites for the World Wide Web. In addition, Delphi was also one of the first development tools to provide high-level wizards, tools, and services for creating Web Service servers and clients.

In Delphi 2005, Borland continues its tradition of providing you with the best tools for building standards-based applications for the Web. In fact, Delphi 2005 gives you more options than ever before for creating and deploying Internet-based applications. Technologies included in Delphi 2005 include ASP.NET Web Applications, ASP.NET Web Service Applications, Win32 Web Service servers, Win32 Web Service clients, Web Broker Web server extensions, WebSnap Web server extensions, and both Win32 and .NET IntraWeb applications. No other environment even comes close to this much Internet development support.

Borland updated and improved many of the tools that you use to build Web-based applications. For example, the what-you-see-is-what-you-get (wysisyg) designer and the drag-and-drop capabilities of the Web Forms designer have been updated. In addition, new features and components have been enhanced. The following sections discuss some of the new and enhanced Web and Internet-related features that you will find in Delphi 2005.

Deployment Manager

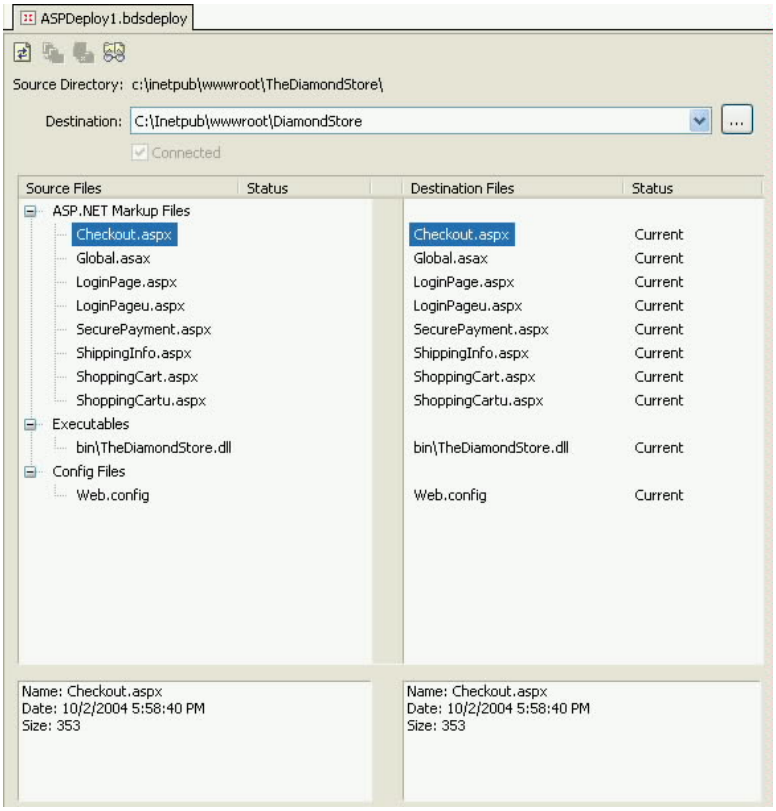
You can now deploy your ASP.NET Web applications, ASP.NET Web Service Applications, and IntraWeb (both Win32 and .NET) applications directly from Delphi 2005's Project Manager. To do this, right-click the Deployment node in the Project Manager and select New Deployment from the displayed context menu.



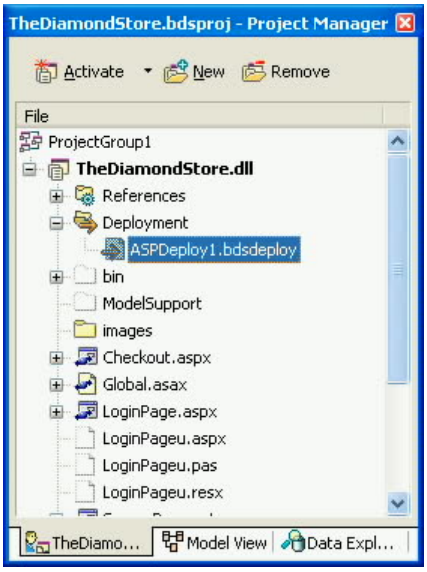
You can deploy your application's files using either XCOPY or FTP (file transfer protocol). Use XCOPY when the directory to which you want to copy your files is visible from your local machine. For example, you can use XCOPY if your Web server is on the same local area network as your development machine.

FTP is useful when the location where you want to deploy your files is available somewhere on the Internet, but is not on the local network. In order to deploy using FTP, the server to which you want to deploy your files must be running an FTP server.

Once you select the directory or FTP server to deploy your files to, select which files you want to deploy, right-click, and then select Copy Selected Files to Destination or Copy All New and Modified Files to Destination.

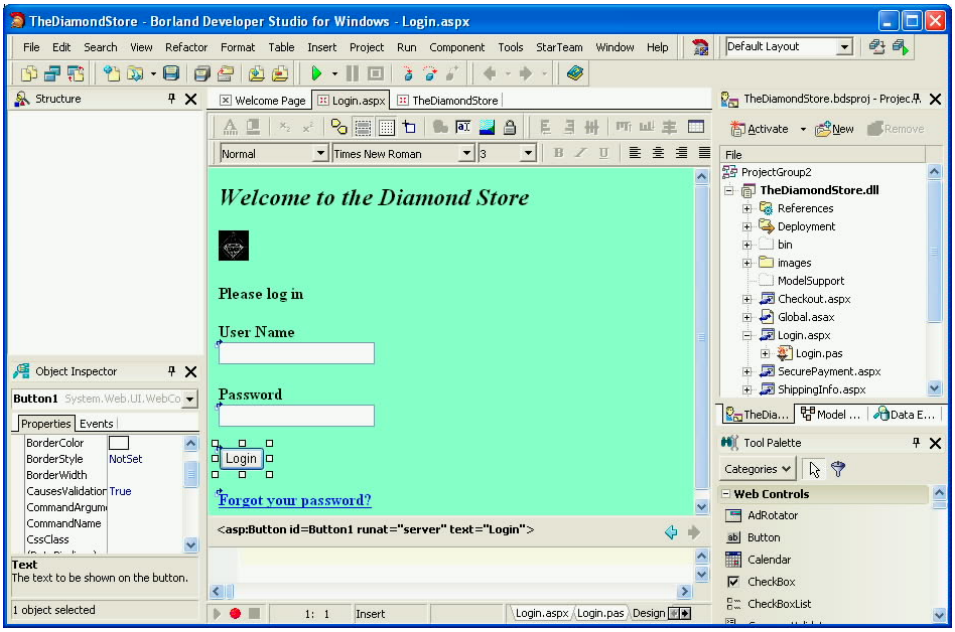


Once you have created a deployment, that deployment appears as a new node beneath the Deployment node in the Project Manager. You can re-deploy some or all of your files using the deployment configuration that you created earlier by selecting the associated node. If you want, you can have multiple deployment configurations for any of your Web-related projects.

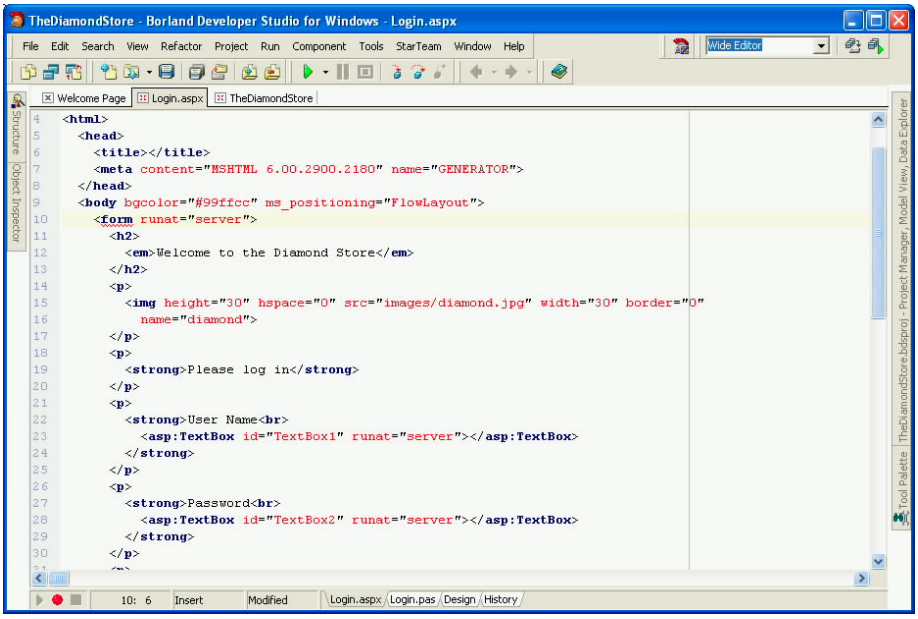


HTML Editing in the Web Forms Designer

You use HTML (hypertext markup language), either in code or wysiwyg, to describe the Web pages that you create in your ASP.NET applications. Delphi 2005 provides you with a number of options for creating and modifying the HTML that defines your various ASP.NET pages. For example, the following figure shows a login page being designed using the ASP.NET Web application designer.

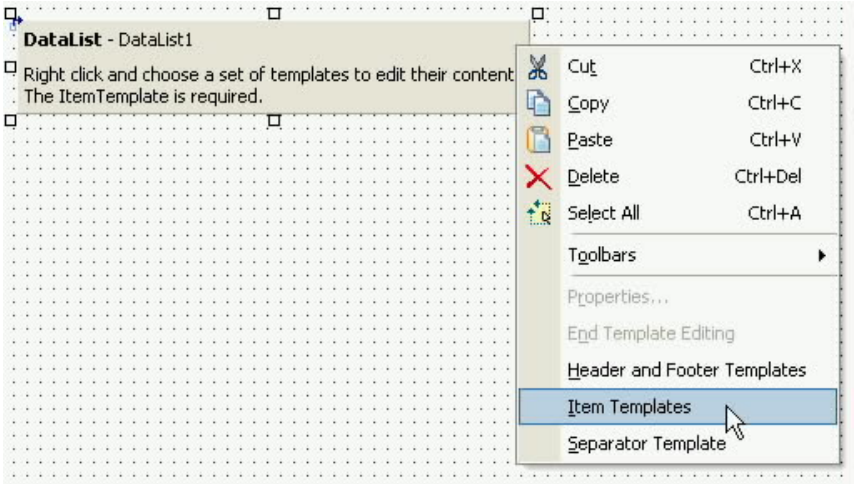


When you drag HTML Controls, Web Controls, and DB Web Controls from the Tool Palette onto the Web Forms designer, HTML is inserted into the associated .aspx file in your project. You can edit this .aspx file directly, modifying what the Web Form designer generated, or you can insert your own custom HTML. The following figure shows a portion of the editable .aspx file that was created as the preceding login page was being designed visually.



Template Editing

Delphi 2005's Web Form designer now permits template editing within the form designer. Certain Web controls, such as a DataList, support templates for the formatting of the header, footer, and displayed items. To edit a template in Delphi 2005's Web Form designer, right-click a template-supporting control and select the template that you want to edit. For example, the following figure shows the context menu that is displayed when you select a DataList.



After selecting which type of template you want to edit, the designer re-draws the control, permitting you to enter the template text directly. For example, the following figure shows a DataList with its Item templates available for editing.

When you are through editing your control's templates, right-click the control again and select End Template Editing.

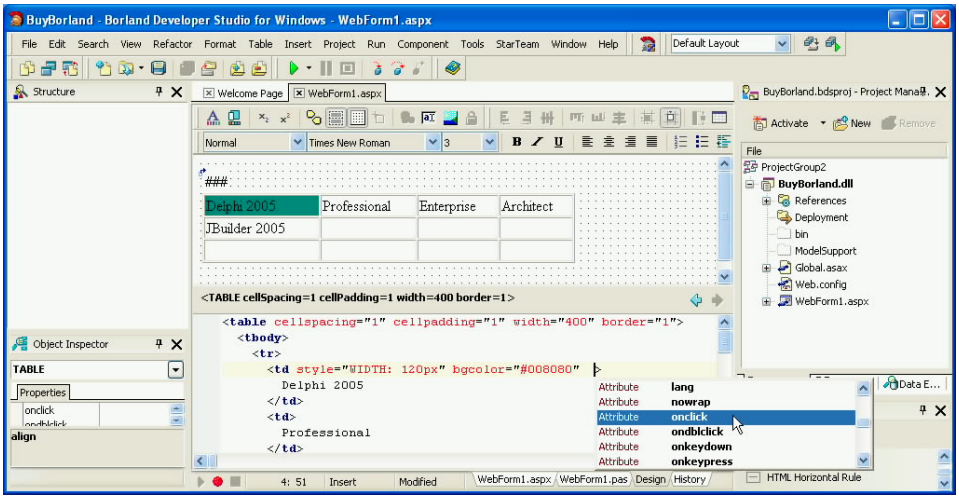
Updated Code Completion and Syntax Highlighting

While Delphi has supported code completion and syntax highlighting of HTML in past versions, Delphi 2005 has extended this support. Code completion and syntax highlighting are now available for cascading style sheets (CSS) and XHTML.

Updated Tag Editing

The Tag editor has also been improved in Delphi 2005. The Tag editor is the small window that appears below the Web Form Designer, and it provides you with a context sensitive, editable view of the HTML that underlies your Web page. While earlier versions of the Tag editor permitted you to edit the inner HTML, you can now edit the outer HTML as well.

The following figure shows attributes of a <td> tag being edited in the Tag edit. Note that both Code Insight and syntax highlighting are visible in this figure.



Additional ASP.NET Project Manager Support

ASP.NET Web applications, more so than other types of applications, often rely on external files to operate. For example, while the HTML in your .aspx file may include an <IMG> (image) tag, the image itself is typically a .jpg or .gif resource whose location is referred to in the src attribute of the <IMG> element.

In addition to supporting application deployment, as discussed earlier in this section, the Project Manager has also been updated to better manage the external resources used in your ASP.NET applications. For example, you can right-click an ASP.NET project in the Project Manager and select New | Folder. The newly added folder will be created as a subdirectory of the ASP.NET application folder.

Once you've added a new folder, you can right-click it and select Add. This brings up a browser dialog box that you can use to add support files, such as images, cascading style sheets, JavaScript files, and so forth, to the folder. The resources that you add to this folder can then be included in your configured deployments.

With these enhancements to the Project Manager for ASP.NET applications, you no longer need to leave Delphi 2005 in order to manage your application's files.

New and Enhanced DB Web Controls

DB Web controls are special data-aware Web controls that you can use in your ASP.NET applications. Like the Web controls that ship with the .NET framework, you add DB Web controls to your Web Forms, and they participate in the generation of the content that is provided to the requesting browser at runtime.

Compared with the standard Web controls of the FCL, DB Web controls offer better support for ASP.NET applications, making it even easier for you to build great Web sites faster. For starters, DB Web controls are data-aware, and in many cases, provide you with automated read/write access to the data to which they are bound. As a result, they greatly simplify the process of creating sophisticated Web-based applications.

New DB Web Controls

With this release of Delphi 2005, Borland has added a number of new and enhanced DB Web controls. The following are the DB Web controls introduced in Delphi 2005: DBWebAggregateControl, DBWebNavigatorExtender, DBWebSound, and DBWebVideo.

The DBWebAggregateControl is similar to a DBWebTextBox, but automatically calculates and displays an aggregate statistic, such as Sum, Min, and Count.

The DBWebSound and DBWebVideo controls allow you to easily add sound and video to your ASP.NET applications. The sound or video resource can either be contained in a blob field of a database, or the database field can contain a string that specifies the URL of the external sound or video resource.

Finally, the DBWebNavigationExtender permits you to configure standard Web control Buttons to perform navigation operations against BDP for ADO.NET data sources without additional code. Simply place a DBWebNavigationExtender component on a Web Form, and any Buttons that you place will display three additional properties: DBDataSource, TableName, and DataSourceAction. The DataSourceAction property indicates what type of navigation operation the button will perform on the table accessed through the DBDataSource.

Updated DB Web Controls

There are two updated DB Web controls in Delphi 2005. These are the DBWebImage and DBWebDataSource. The DBWebImage has been updated to include a feature of the newly added DBWebSound and DBWebVideo controls described earlier in this section. The DBWebImage can be linked to either a blob field in an underlying database that contains the image to display, or a string field containing the URL of the image resource. Previously, the DBWebImage control could only refer to a blob field containing the image to display.

The remaining updates to DB Web can be found in the DBWebDataSource. DBWebDataSource can now be configured to support auto-updates, as well as cascading updates and cascading deletes for master-detail relationships.

DBWebDataSource now also supports XML files for the storage of the data used by DB Web controls. You can use this feature in a number of interesting ways. For example, an XML file can be used instead of an underlying database during development, providing a convenient substitute for a database connection. Alternatively, an XML file can be used as a local, readonly data source for managing static information, such as images or other resources. Alternatively, if user authentication is being used, a DBWebDataSource can be configured to generate a unique XML file name for each user. That XML file can be used to persist data on a per user basis between sessions.

IntraWeb Support

IntraWeb is a sophisticated, RAD component-based Web development tool that automatically maintains server-side state between Web page requests. As a result, IntraWeb has advantages over ASP.NET for creating Web sites that require the type of state persistence typically associated with traditional client applications.

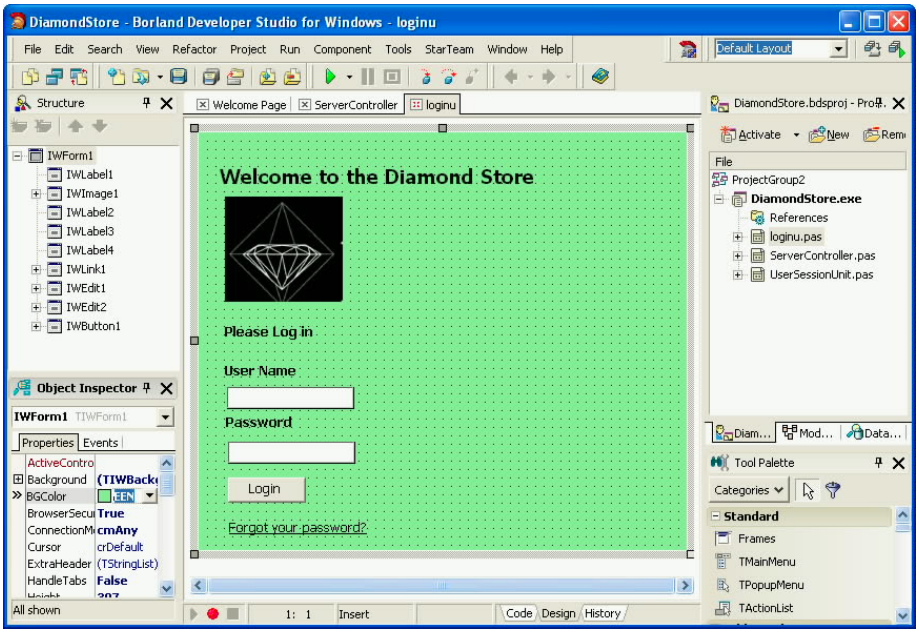
There are a number of features that make IntraWeb an attractive alternative to ASP.NET Web site development. As mentioned previously, IntraWeb supports several convenient levels of state maintenance between Web page requests. At the application level, you can use the TIWServerController to share objects between sessions.

At the session level, each IntraWeb session can have its own persistent data module that remains in memory for the duration of the session. This data module can be used to store objects and data that are used by two or more Web pages for a particular end user. Finally, unlike ASP.NET, where ASP.NET Web forms are created and destroyed for each page request, IntraWeb pages persist on the server between requests, until the page is no longer needed.

The second aspect of IntraWeb that makes it attractive is its "Delphi" way of doing things. You design your user interface using Delphi components from the Tool Palette, just as you would design any VCL or VCL for .NET application. The difference is that these components participate in the IntraWeb form rendering process to emit HTML, WAP (wireless access protocol), or HTML 3.2.

Finally, you have a variety of choices for deploying IntraWeb applications. An IntraWeb application can be deployed as an ISAPI (Internet Server application programming interface) Web server extension, or it can be used as a self-contained HTTP server. In other words, if you are already running IIS (Internet Information Server), you can use your IntraWeb application with it. On the other hand, if you do not already have a Web server, you can design your IntraWeb application to be a Web server, providing all the features necessary to serve Web pages to any Web browser or Web-enabled device using the HTTP protocol.

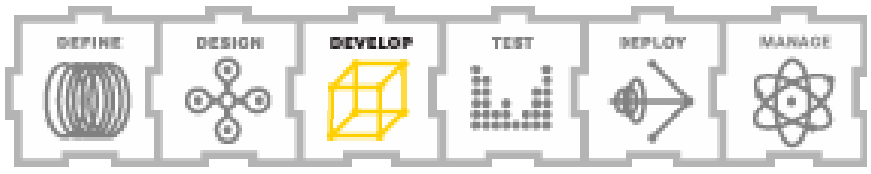
The following figure shows an IntraWeb Web page being designed in Delphi 2005. Unlike ASP.NET applications, there is no code-behind .aspx file. Instead, the IntraWeb components used to build the page render the appropriate HTML at runtime in response to an appropriate HTTP Web page request.



Delphi 2005 includes both Win32 Delphi and Delphi for .NET versions of IntraWeb.

Integrated Application Lifecycle Management

Borland®
Delphi™ 2005



Integrated Application Lifecycle Management

In today's world of software development, most developers are part of a larger process of application definition, design, testing, deployment, and management. Consistent with Borland's commitment to providing you with the solutions you need to ensure the success of your projects, Delphi 2005 provides tight integration to essential support tools.

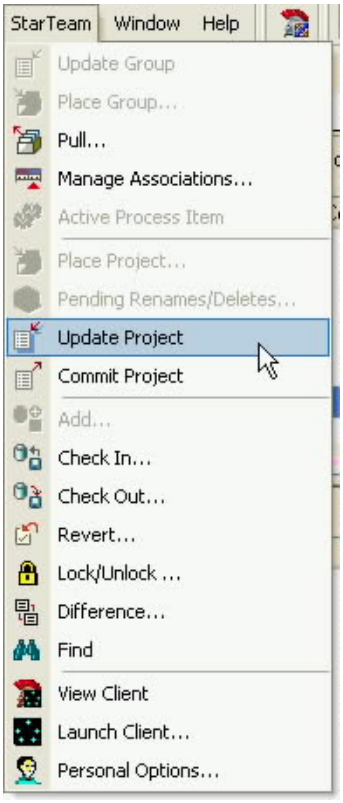
Depending on the version of Delphi 2005 that you have installed, the IDE provides you with integrated access to StarTeam for superior, team-based asset management, Janeva for access to Enterprise JavaBeans and CORBA servers, ECO for UML model driven development, and Optimizeit for performance profiling.

Two of Delphi 2005's application lifecycle management (ALM) solutions deserve particular attention. These are StarTeam and unit testing. These tools are described in the following sections.

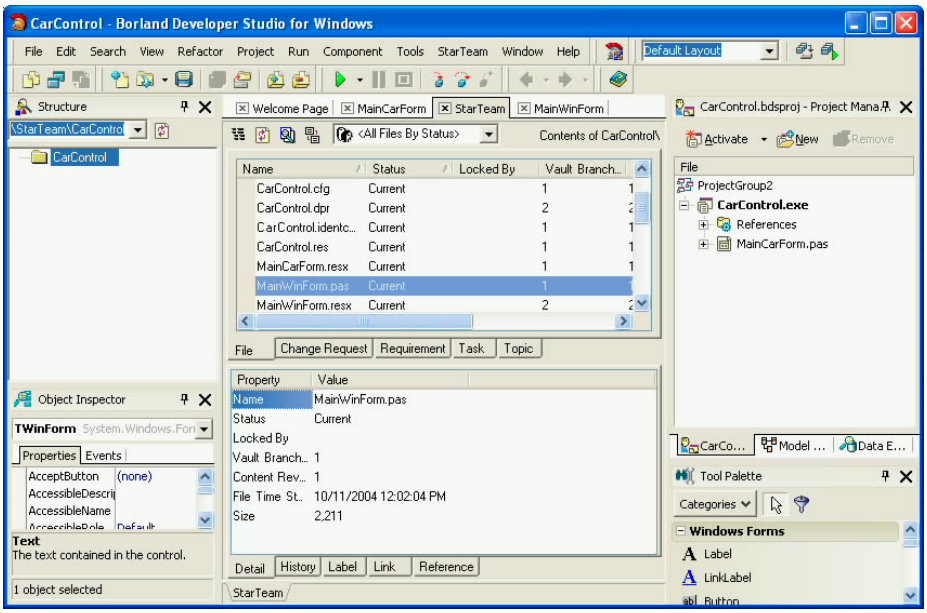
Delphi 2005 and StarTeam

StarTeam is Borland's comprehensive project asset management system. If you are part of a development team, StarTeam provides a highly reliable, server-based system for source code version control, requirements management, defect tracking, threaded discussion groups, and distributed collaboration. Even if you are the sole developer on a project, StarTeam provides you with an indispensable environment for managing every aspect of your applications.

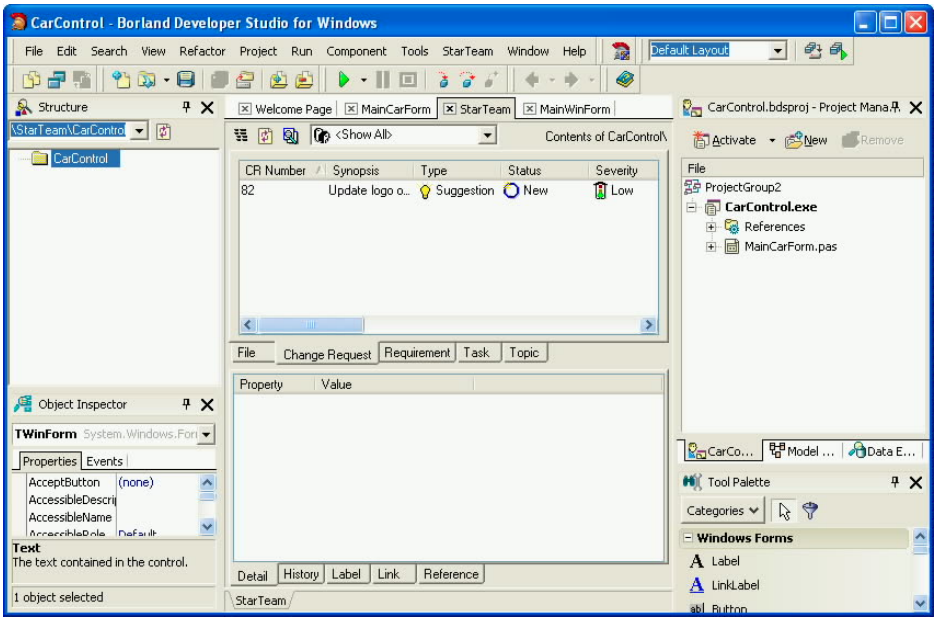
Delphi 2005 provides you with seamless integration with StarTeam, enabling access to all of StarTeam's capabilities without leaving the Delphi 2005 IDE. The Delphi 2005 main menu includes a StarTeam main menu item, as well as a submenu on the Project Manager context menu. This menu is shown in the following figure.



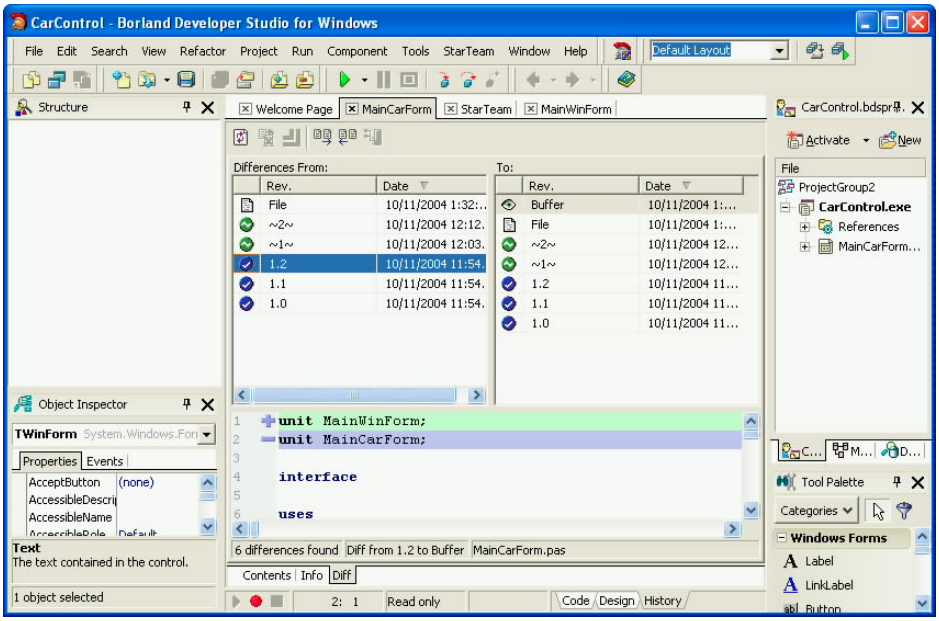
As you can see, the StarTeam menus permit you to place a project into a StarTeam repository, check in and check out files, locate managed assets, launch the integrated StarTeam client, and manage your personal StarTeam options. The following figure shows the StarTeam client active in the Delphi IDE. You launch the StarTeam client by selecting StarTeam | View Client from Delphi 2005's main menu or StarTeam | View Client from the Project Manager's context menu.



With the StarTeam client active within Delphi 2005, you can work with every aspect of your managed resources. For example, you can track defects, view and contribute to threaded discussions, submit change requests, and more. The following figure shows a change request that has been logged into the StarTeam server for this project.



When you are working with a StarTeam managed project, the Delphi 2005 History Manager makes use of the StarTeam repository. For example, the following figure shows the Diff pane of the History Manager. Here the Diff pane displays source code versions based on changes that have been checked into the StarTeam repository. With the StarTeam enabled History Manager, even changes to source code file names are tracked, as shown in the following figure.

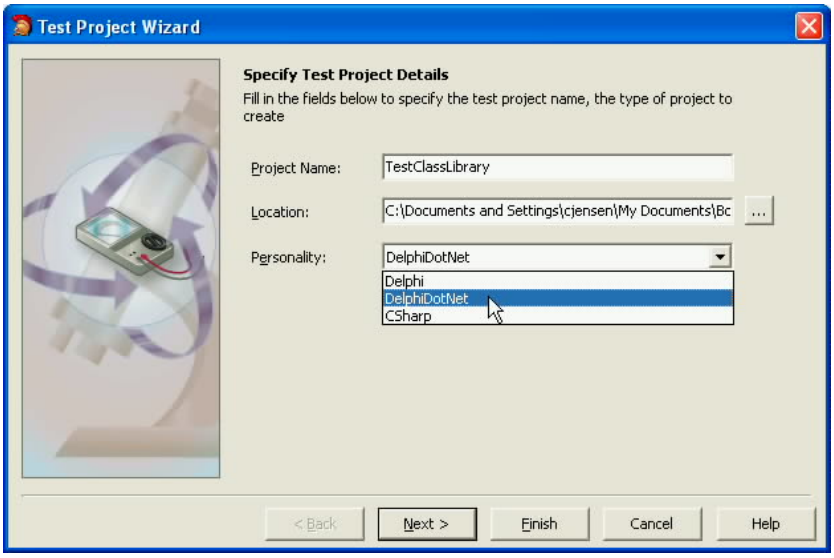


Unit Testing

Unit testing is the process of writing code to test the methods, functions, and procedure of your software. While unit testing is a corner stone of the approach to software development called extreme programming, many developers find it useful to employ some form of unit testing as part of their everyday software development.

Delphi 2005 includes unit testing support for all three of its personalities: Delphi Win32, Delphi for the .NET Framework, and C# for .NET. You establish a unit testing by first creating a test project. The Test Project Wizard asks you to select which of Delphi 2005's

personalities was used to create the code you want to test.



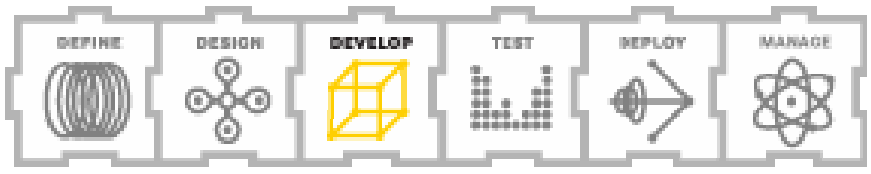
After creating your test project, you add one or more test cases to your test project. Each test case requires you to select the source file (.pas or .cs) that contains the methods or routines you want to test. The Test Case Wizard then generates a simple framework for testing that file. This framework includes a Setup and a Teardown procedure, as well as a method stub for each of the subroutines in your selected source file.

You modify the code generated by the Test Case Wizard to implement the Setup and Teardown procedures, as well as the individual tests. For example, you will typically call the constructor of the class in which the methods you want to test are implemented from the Setup procedure, as well as define any variables or objects that are needed for the parameters of your test methods. Likewise, you will free the class created in the constructor, and release any allocated resources, from the Teardown procedure.

The actual tests are performed within the stubbed out methods generated by the Test Case Wizard. You implement each of these methods to invoke the method they are testing, validating either the data resulting from the method execution, or the class of exception thrown when your method detects a problem.

Enterprise Core Objects (ECO) II

Borland®
Delphi™ 2005



Enterprise Core Objects II

Borland's Enterprise Core Objects, or ECO (pronounced ee'ko), is Borland's new model-powered framework for .NET. ECO is an object-oriented framework from Borland for the .NET framework that uses UML (universal modeling language) diagrams to drive application development. This approach to building applications is often referred to as model driven architecture, or MDA.

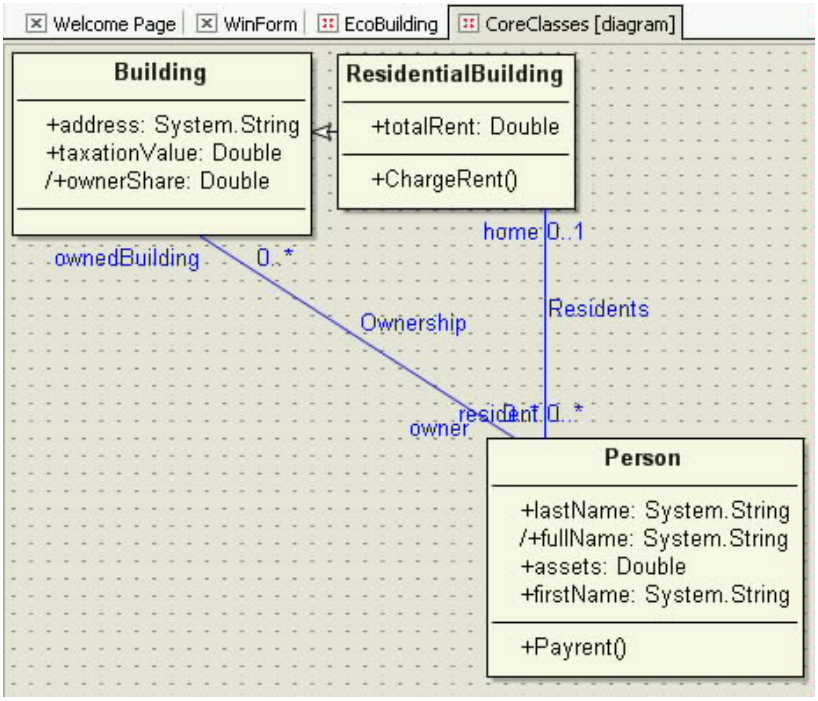
One of the more notable features of ECO concerns how the UML models are used. In many development environments, UML models simply provide you with a road map, defining the classes that you need to implement in your application. In other words, UML diagrams are used as guidelines for software development.

With ECO, UML models are not just used to guide development; they are tightly integrated into the development process. Models are used to generate classes and support code that represents the core of your application logic. When changes need to be made to the application, you return to the model, modifying its attributes, associations, and constraints, after which your application's code is updated.

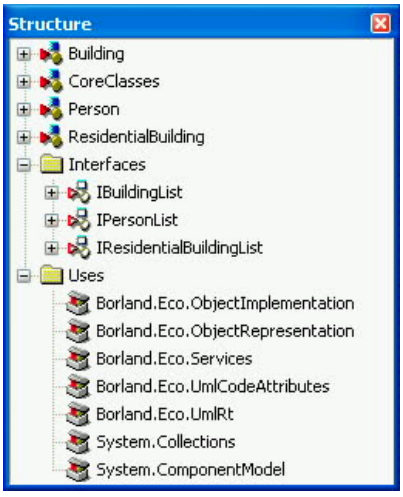
Rapid MDA

In this respect, ECO is really "Rapid MDA." ECO dramatically reduces the amount of code that you need to write manually, reducing your time to deployment and improving the overall maintainability of your applications. More importantly, the applications you build with ECO are based on the enterprise-aware architecture of your UML designs.

The following figure shows the UML class diagram for a simple ECO application. As you can see, there are three classes defined here: Building, ResidentialBuilding, and Person.



This diagram is used to generate the business objects that this application will work with at runtime. The Structure pane shown in the following figure shows you the classes and associated interfaces that ECO generated from this model.



Because the model is the central focus of your development efforts, there is an inherent synchronization between your UML model and the application created with it. In other environments where UML simply guides development, the model often quickly becomes out-of-date.

In ECO, the UML model defines the core business objects that are the focus of your development efforts. For example, if you build an ECO application to manage inventory, the objects that you work with will represent the entities of your application, such as items, employees, orders, storage facilities, and the like. In other words, your code operates in the domain of the business objects that you are using. Compare this approach to the type of development that you typically see in GUI applications, where code operates in the realm of the user interface, with items such as buttons, text boxes, list boxes, and menus.

In most ECO applications, the business objects defined by your UML models map to an underlying ADO.NET relational database structure. This database of your choosing is used to persist and restore your business objects, as needed. You can even map your ECO objects to XML files, though most developers prefer the security and transaction support provided by a remote database server.

In traditional database development, you spend considerable time designing your database and writing the code needed to store and retrieve your data. With ECO, the underlying database schema can be created for you, based on your UML models. Alternatively, you can map an existing database to your UML models, permitting you to use the power of ECO with your current databases.

ECO Space and Persistence Mapping

The ease with which you work with objects using ECO is particularly noteworthy. Object persistence is provided in ECO through an ECO space, a factory-like container that provides both an object cache as well as a transparent interface to the underlying data store. The ECO space creates your objects, as you need them, and persists changes that you make, if persistence is required.

For example, if you ask for an object that represents an existing employee, the ECO space creates an employee object and populates its attributes with data from an underlying database. Any changes made to the employee object can likewise be saved back to the database. This capability is provided by an ECO persistence mapper, which performs the required data-related tasks for you.

ECO and OCL

In addition to UML, ECO employs OCL, the object constraint language, an Object Management Group (OMG) standard for defining expressions for UML models. You use OCL to create declarative rules that calculate or control the values of attributes of your objects. As is the case of UML, the OCL you employ in your ECO applications reduces the amount of code that you have to write and maintain.

What's New in ECO II

Delphi 2005 ships with ECO II, a major update to Enterprise Core Objects. ECO II improves and extends your support for building enterprise-level model driven applications in the .NET framework. The updates found in ECO II are described in the following sections.

A Highly Scalable Enterprise Object Cache

ECO II includes two important enhancements to ECO spaces that improve how and where your applications can be used, as well as their scalability. The first of these is that a single process can now include two or more ECO spaces. This capability is particularly valuable for ASP.NET applications where ECO spaces can be pooled and reused for increased application performance.

The second improvement permits multiple ECO spaces to be synchronized, a capability supplied by the ECO persistence mapper components. Synchronized ECO spaces permit changes in one ECO space to be more easily resolved with changes that appear in another ECO space.

The ECO persistence mapper classes are thread-safe and remotable. In fact, two or more ECO spaces on separate computers can use .NET remoting to share a common persistent mapper, permitting those ECO spaces to be synchronized. This capability permits ECO applications to be easily scaled up to a multi-tier architecture, as you application's needs change.

Extended Object Capabilities

ECO spaces provide more support for object persistence than ever. Added features include undo/redo, versioning, and transactions.

ECO II Support for Web Forms and Web Services

ECO II provides extensive support for building Web-based applications using rapid MDA. Delphi 2005 includes wizards for creating ECO ASP.NET Web Form applications and ECO ASP.NET Web Service applications for both C# and Delphi for .NET.

Delphi 2005 also includes the ECODataSource component, which you can use to bind your DB Web components to ECO-based business objects. This data source implements DbDataSource, which means that you can assign it to the DataSource property of any DB Web control.

Two new enhancements to ECO spaces are particularly valuable to ASP.NET developers. The first is that an ECO space can be maintained on a per session basis, providing automatic state maintenance between page requests. The second provides for a pool of ECO spaces. These features can be used individually or in conjunction with one another to enhance the features and performance of your ECO-based ASP.NET applications.

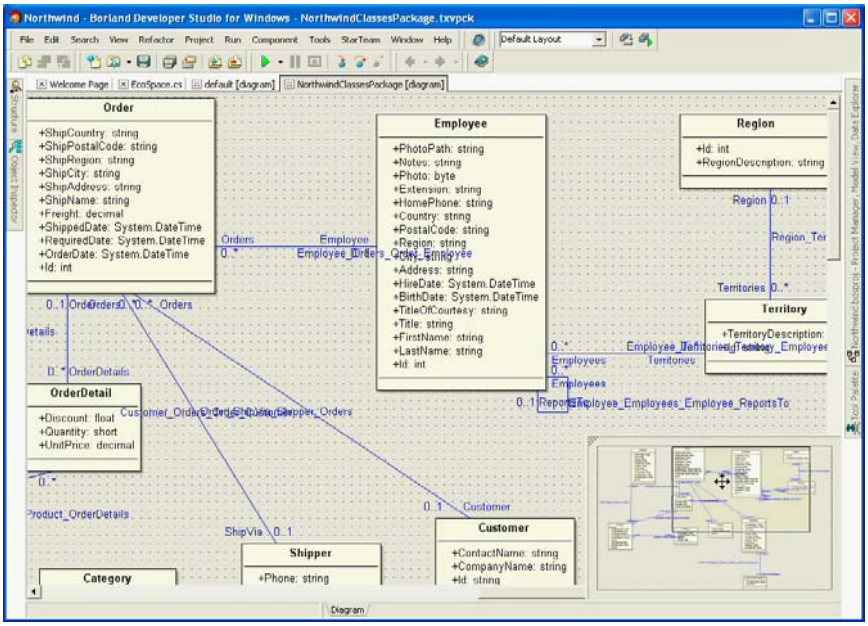
Each ASP.NET application contains an EcoSpaceProvider, which controls the caching of ECO spaces creating within the application. You use this provider to control whether an ECO space is maintained between page requests for a particular session or not. Options include never maintaining state, always maintaining state, or only maintaining state when unresolved changes appear in the ECO space. While maintaining state requires more server resources, it simplifies how you work with your objects in an ASP.NET application.

ECO space pooling permits ECO spaces to be easily reused in ASP.NET applications. For those ASP.NET applications that do not persist ECO space between page requests, each time an ASP.NET page is destroyed, its ECO space is returned to the ECO space pool. Performance is enhanced since a new ECO space does not need to be created for each page request. For those ASP.NET applications that maintain ECO space for each session, the ECO space is returned to the ECO space pool when the session terminates.

ECO II Support for Existing Databases

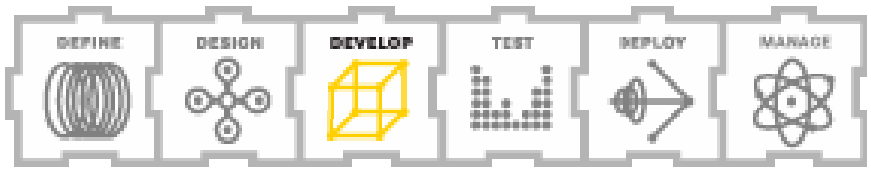
ECO II can examine the schema of your existing database and use this information to generate your initial UML diagrams. Alternatively, you can manually map your UML diagrams to an existing database. Previously, you had to create your UML diagrams first, and generate your database from these diagrams. With ECO II, you can now bring the power of ECO to your existing databases.

The following figure shows a UML diagram that ECO created from the sample SQL Server database Northwind. In addition to the various classes and their attributes, ECO infers the relationships between the classes based on field names and indexes.



Integrated and Included Partner Tools

Borland®
Delphi™ 2005



Integrated and Included Partner Tools

All versions of Delphi 2005 include licenses for other valuable Borland products that support software development and application lifecycle management, as well as products from Borland partner companies. Which products are included with Delphi 2005 depend on the version that you are using. All of the products listed in this section are included in Delphi 2005 Architect. Delphi 2005 Enterprise and Professional include some, but not all, of these products.

The following sections provide you with a short description of the associated integrated or included partner tool. For more information, please use the URL provided to learn more.

Borland InterBase 7.5 Developer Edition

InterBase 7.5 Developer Edition enables you to develop and test your applications running against InterBase, an enterprise-quality remote database management system (RDBMS). Borland InterBase is a small-footprint database server that minimizes maintenance while providing support for mission-critical applications. For more information on InterBase 7.5, please visit: http://www.borland.com/interbase/

Borland Janeva

Janeva provides you with a seamless and cost-effective solution for integrating your J2EE and CORBA back-end systems with your client and Web applications. For more information about Janeva, please visit: http://www.borland.com/janeva/
*NOTE Janeva requires a runtime license to deploy your application. This is available from a Borland sales representative.

Borland Optimizeit™ Profiler for the Microsoft .NET Framework

Identify and remove performance bottlenecks in your .NET managed code through CPU and memory usage analysis with Borland Optimizeit Profiler for the Microsoft .NET Framework. For more information about Borland Optimizeit for the Microsoft .NET Framework, please visit: http://www.borland.com/opt_profiler/

Borland StarTeam 6.0 Standard Edition

StarTeam provides you with a rich and automated system for managing the assets and application lifecycle tasks from within a single repository. For more information about StarTeam 6.0 Standard Edition, please visit: http://www.borland.com/starteam/

Component One Studio Enterprise for Borland Delphi 2005

Component One Studio Enterprise for Borland Delphi 2005 is a special edition of Studio Enterprise that includes a development license for eleven .NET (Windows Forms) and six ASP.NET (Web Forms) controls. For more information about Component One Enterprise Studio, please visit: http://www.componentone.com/

Crystal Reports Borland Edition

Crystal Reports Borland Edition is a .NET version of the world's leading reporting tool for use in your C# and Delphi for .NET applications. For more information about Crystal Reports Borland Edition, please visit:
http://www.businessobjects.com/products/reporting/crystalreports/net/default.asp

glyFX Borland Special Edition

glyFX Borland Special Edition is a collection of 95 high-quality images for use in toolbars, buttons, or any control that supports bitmap files. For more information on glyFX Borland Special Edition, please visit: http://www.glyfx.com

IBM DB2 Universal Developers Edition

IBM DB2 Universal Developers Edition provides you a DB2 database and associated tools for designing, building, and prototyping applications for deployment on any DB2 client or server platform.

InstallShield Express for Borland Delphi

InstallShield Express for Borland Delphi provides you with an easy-to-use graphical interface for building custom installers for your Windows software. For more information on InstallShield Express for Borland Delphi, please visit: http://www.installshield.com/Borland

Internet Direct (Indy)

Internet Direct (Indy) is an open-source Internet component suite comprised of popular Internet protocols written in Delphi and based on blocking sockets. For more information about Internet Direct, please visit: http://www.atozed.com/indy

IntraWeb

IntraWeb is a complete RAD solution for building Web applications, dynamic Web sites that go well beyond the capabilities of regular ASP.NET Web applications and ISAPI Web server extensions. For more information on IntraWeb, please visit:
http://www.atozed.com/intraWeb/

Microsoft SQL Server 2000 Desktop Engine (MSDE 2000)

Microsoft SQL Server 2000 Desktop Engine provides your small workgroup and low-volume Web applications with data storage capabilities that easily scale to Microsoft SQL Server 2000 as your needs grow.

Microsoft SQL Server 2000 Developer Edition

Microsoft SQL Server 2000 Developer Edition provides you with a developer license for designing, building, and prototyping applications that you can deploy with Microsoft SQL Server 2000.

Rave Reports Borland Edition

Rave Reports Borland Edition is a powerful and scalable suite of VCL and VCL for .NET reporting components for creating sophisticated Delphi reports. For more information about Rave Reports Borland Edition, please visit: http://www.nevrona.com/rave/

Wise Owl Demeanor for .NET Borland Edition

Wise Owl Demeanor for .NET Borland Edition is a .NET obfuscator, a tool that helps prevent others from reverse-engineering your managed code applications and assemblies. For more information about Wise Owl Demeanor for .NET Borland Edition, please visit: http://www.wiseowl.com/

Other Resources

Please also visit the Borland Developer Network, where you will find timely articles as well as links to a wide variety of resources that support your software development needs. The Borland Developer Network is located at http://bdn.borland.com/.

You should also consider visiting Code Central, Borland's online repository for code samples, demonstration applications, and other resources for developers using Borland products. Code Central is located at http://cc.borland.com/ccweb.exe/.

Summary

More than twenty years in the making, Delphi 2005 achieves what no other development environment can, providing you with state-of-the-art tools that preserve your investment in today's software as you migrate towards tomorrow's new standards. With integrated tools that support every aspect of the application lifecycle, Delphi 2005 truly is the ultimate Windows development solution.

About Borland Software Corporation

Founded in 1983, Borland Software Corporation (NASDAQ: BORL) is the global leader in platform independent solutions for software delivery optimization. The company provides the software and services that align the teams, technology and processes required to maximize the business value of software. To learn more about delivering quality software, on time and within budget, visit: http://www.borland.com.

About the Author

Cary Jensen is President of Jensen Data Systems, Inc., a software development, training, and consulting company (http://www.jensendatasystems.com). He is an award-winning, best-selling author of nineteen books, a featured columnist on the Borland Developer Network (http://bdn.borland.com), and a popular speaker at conferences, workshops, and training seminars around the world. Cary has a Ph.D. in Human Factors Psychology, specializing in human-computer interaction, from Rice University in Houston, Texas. You can contact Cary at cjensen@jensendatasystems.com.

Made in Borland® Copyright © 2004 Jensen Data Systems, Inc. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Microsoft, Windows, and other Microsoft product names are trademarks or registered trademarks of Microsoft Corporation in the U.S. and other countries. All other marks are the property of their respective owners. Corporate Headquarters: 100 Enterprise Way, Scotts Valley, CA 95066-3249 • 831-431-1000 • www.borland.com • Offices in: Australia, Brazil, Canada, China, Czech Republic, Finland, France, Germany, Hong Kong, Hungary, India, Ireland, Italy, Japan, Korea, Mexico, the Netherlands, New Zealand, Russia, Singapore, Spain, Sweden, Taiwan, the United Kingdom, and the United States. •