Tech Note

embarcadero®

Reasons to Migrate to Delphi XE2

What you might have missed since Delphi 7

Andreano Lanusse

November 2011

TABLE OF CONTENTS

INTRODUCTION

Many Delphi users wonder whether they'll find compelling reasons to migrate to Delphi XE2. Here they are: a plethora of new features allied to unparalleled developer productivity, all aimed at your ability to create higher-quality applications with improved performance. This article gives a few good reasons to migrate, along with an overview of all the new features added to Delphi product releases since version 7, where Delphi was an IDE to build Windows 32-bit applications and became an IDE for Windows 64-bit, Mac and iOS development.

The following table gives a quick snapshot of the new, top level, capabilities in each of the Delphi product releases since Delphi 7. This paper provides the deep technical information for all of these features and more.

Delphi 2005

- Multi-unit namespaces, for ... in ... do loops, inline functions and other code optimizations.
- Heterogeneous database access, multi-tier database applications
- Refactoring, Source code History view
- Unit testing

Delphi 2006

- Code block completion/Surround, Editor Change Bars
- Live Code Templates
- UML Modeling, Audits, Metrics, Doc Gen
- Design Patterns

Delphi 2007

- MSBuild, Build Configurations
- VCL - AJAX, Vista compatibility
- Vista and XP Themes for applications
- dbExpress - new framework, delegate drivers, Unicode database support

Delphi 2009

- Unicode throughout the language, library and IDE
- Generics and Anonymous Methods
- Resource Editor, Class Explorer

- DataSnap multi-tier
- VCL – new components, Custom Hints, Ribbon Controls
- Localization – Integrated Translation Environment, External Translation Manager

Delphi 2010

- Windows 7, Multi-Touch and Gesture support, Direct-2D
- IDE Insight, Source Code Formatter, Search task bar
- Background compilation
- Enhanced RTTI
- Breakpoints in threads, freeze/thaw threads
- DataSnap – HTTP protocol support

Delphi XE

- DataSnap – HTTPS, JavaScript, REST support
- Subversion integration
- Regular Expression library
- AQtime, CodeSite, Beyond Compare, Final Builder
- Cloud Services and Cloud Deployment

Delphi XE2

- FireMonkey Platform
- Windows 64-bit, Mac and iOS development
- VCL Styles
- RTL support for Windows and Mac
- DataSnap Monitoring and Control
- DataSnap Connectors
- dbExpress drivers for Mac, Windows 64-bit and new dbExpress ODBC driver
- Cloud API support for Amazon
- Deployment Manager and Platform assistant

We understand that developers are busy creating software, and that spending time to migrate to a new version is not always possible since you always have to deliver new projects.

Possibly one of the best reasons for you to migrate to Delphi XE2, C++Builder XE2 or RAD Studio XE2, is that buying those versions gets you earlier versions at no extra cost.

For example buying Delphi XE2 also gets you Delphi 7, Delphi 2007, Delphi 2009, Delphi 2010 and Delphi XE.

Also the technology has changed so much during the last 10 years, and end users are no longer restricted to getting the information through desktop applications, they use the web, smartphones, tablet, other OS's like Mac, iOS, Android, and other several ways to have access to the information they need.

This article goes through all the features required to build and enable Delphi applications to share information across other platforms.

WHAT'S NEW IN THE IDE

SUBVERSION INTEGRATION – VERSION INSIGHT

With Delphi XE you can easily use the popular Subversion version control system to manage source code revisions for your own code or among your team. Features include:

- Integration into the project manager and history manager
- Support for common version control tasks like import, update, commit and show log.
- Difference and merge viewer
- Source code for the integration using the Open Tools API available as an open source project

This integration is Open Source and is hosted on SourceForge (http://sourceforge.net/projects/radstudioverins/), but there's actually an easier way to access it if you already have XE. We ship a version of the source in the RAD Studio samples directory (which is at C:\Users\Public\Documents\RAD Studio\9.0\Samples).

PROJECT MANAGER

The new Project Manager introduces a lot of new capabilities, which will bring more productivity like:

- Sorting the Contents of the Project Manager: The new Sort By toolbar button enables you to sort the items in the Project Manager by name, timestamp, path name, or file type. You can also specify Auto Sort, which means that future additions to the project or project group are to be added in the current specified sort order. See Project Manager.
    - Compile All From Here and Build All From Here: The project context menu on the Project Manager contains a new From Here command that enables you to perform the following:
    - Compile All From Here
    - Build All From Here
    - Clean All from Here: These three commands start a compile, a build, or a clean operation, respectively, beginning at the selected node in the project. These commands are described in Project Context Menu.

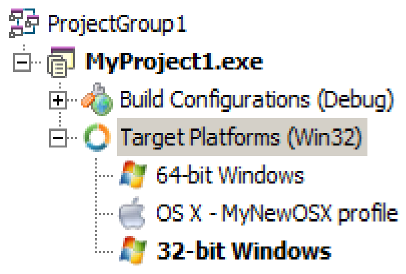
- Compile All, Build All, and Clean All: These new context menu commands are available for Project Groups that contain more than one project. See Project Group Context Menu.
- The Project context menu on the Project Manager contains a new Install|Uninstall command that enables you to either install or uninstall a design-time package.

TARGET PLATFORMS

In the Project Manager, a Target Platforms node exists for any project type that potentially supports cross-platform applications. It was introduced in Delphi XE2

When you create a new project, only the native platform (32-bit Windows) is listed as a target platform (the name of the active platform is boldfaced in the Project Manager). You add cross-platform targets for your application by using the new Target Platforms node. If your application only targets the default Win32 platform, you do not need to make any changes in the Target Platforms node.

For iOS projects the Target Platforms is not enabled, because the project is exclusively for iOS.



GALLERY

The Gallery has been augmented with a search feature. And as an extra productivity enhancement, all gallery items show up but the ones that used to be invisible are grayed out. This should really help those customers migrating from Delphi 7 where all COM wizards were visible but you had to know the order in which to create them. Now, you can run any wizard that is enabled and there won't be any question of where the COM wizards went.

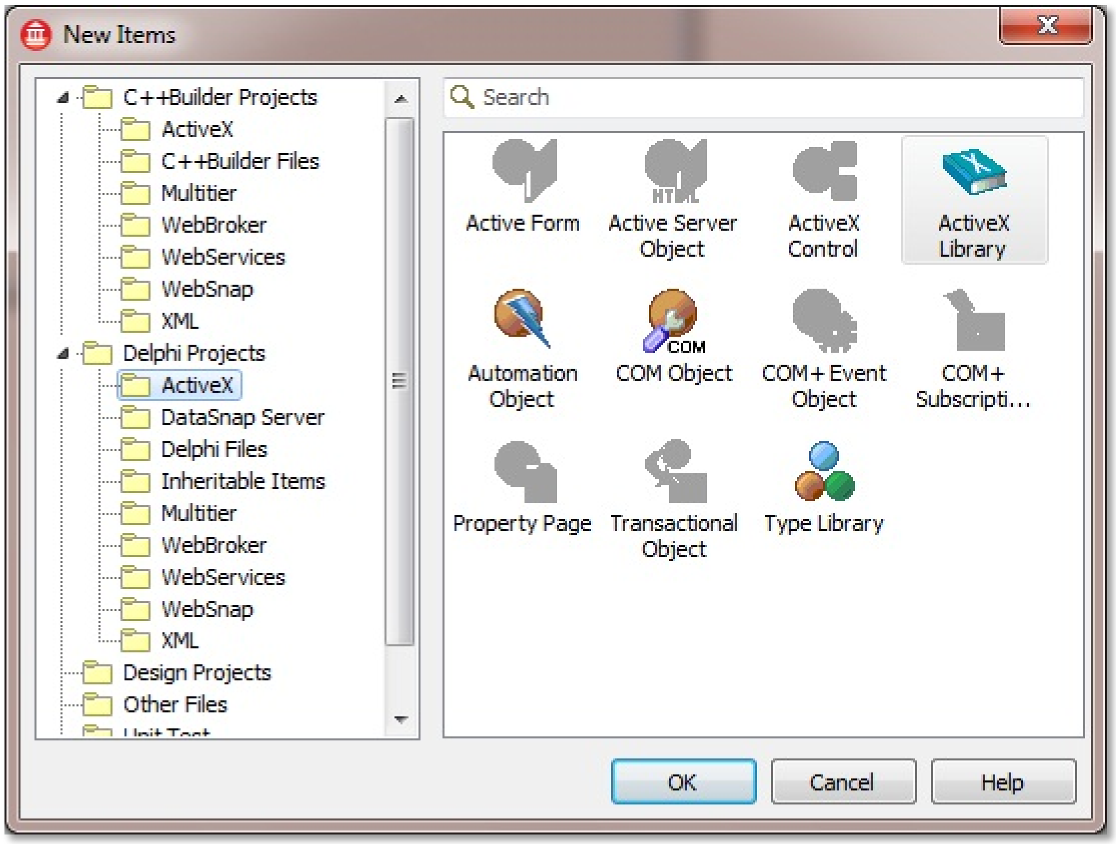
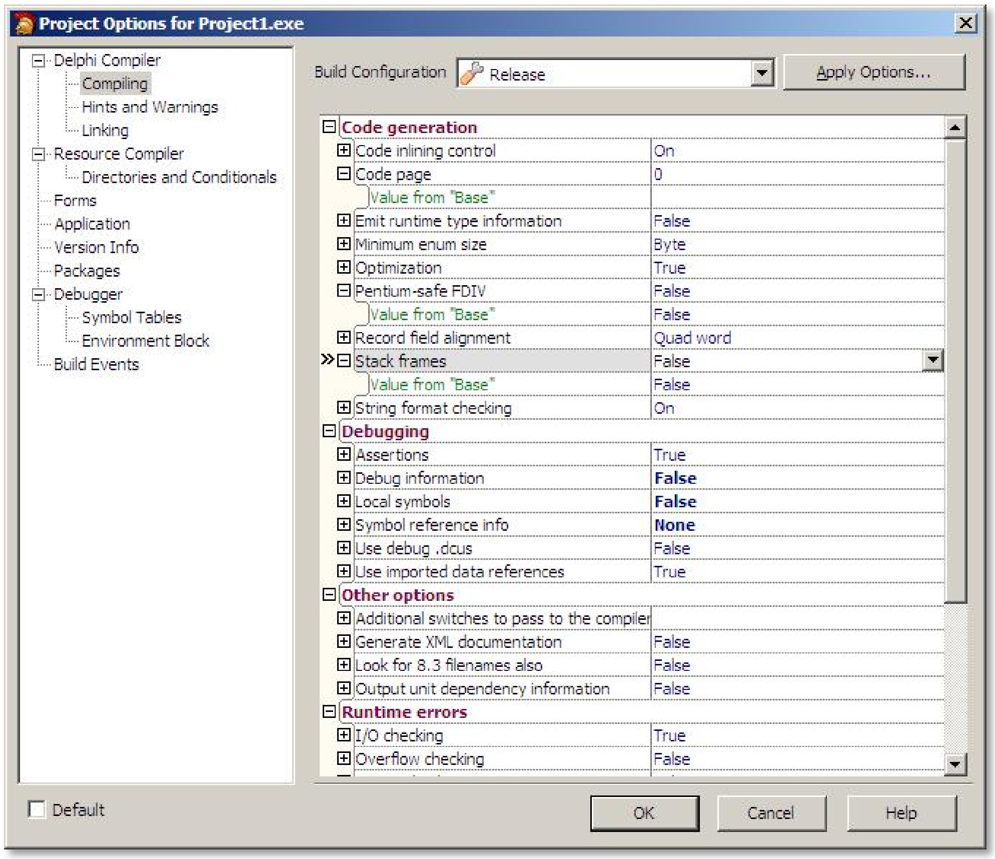


Figure 1. New project Gallery, search integrated

NEW PROJECT OPTIONS

We've changed the IDE in many different ways in order to make development faster and easier. The project compilation options are now displayed in columns and grouped by categories in a friendly manner. It's also now possible to save your project's configuration options, or build configurations, as you'll see in Figure 2.



Build Configuration: Release

Code generation	
Code inlining control	On
Code page	0
Value from "Base"	
Emit runtime type information	False
Minimum enum size	Byte
Optimization	True
Pentium-safe FDIV	False
Value from "Base"	False
Record field alignment	Quad word
Stack frames	False
Value from "Base"	False
String format checking	On
Debugging	
Assertions	True
Debug information	**False**
Local symbols	**False**
Symbol reference info	**None**
Use debug .dcus	False
Use imported data references	True
Other options	
Additional switches to pass to the compiler	
Generate XML documentation	False
Look for 8.3 filenames also	False
Output unit dependency information	False
Runtime errors	
I/O checking	True
Overflow checking	False

Figure 2. Build Configurations

BUILD CONFIGURATIONS

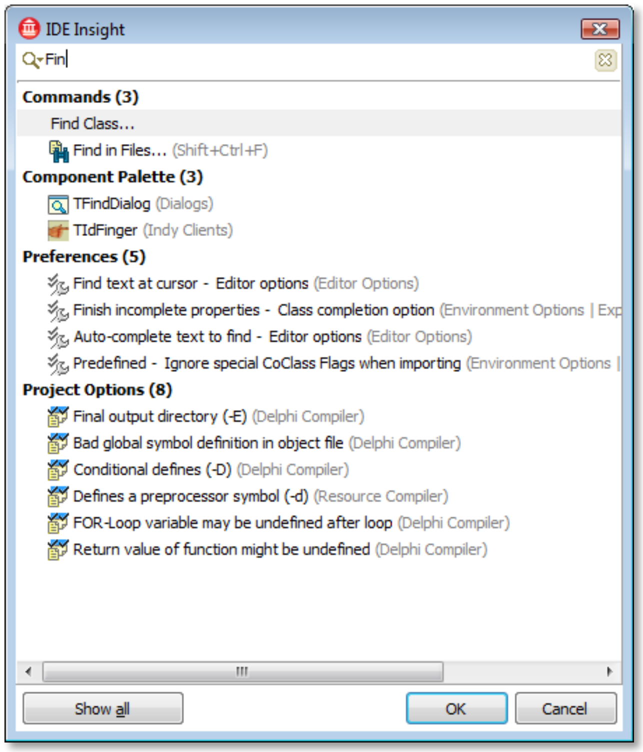
Compiling and debugging projects are regular tasks for developers. However, the project options that are used to run the final version (release) are not always the same project options you use when debugging. Having to constantly change your project's options is a time-consuming task that you can now avoid, never again being forced to spend lots of time working with the Project Manager. In Delphi XE the build configuration options are seamlessly integrated to the Project Manager.

In addition, project configurations can be saved in XML-format OPTSET files. Working with these files you're able to reuse project options from previous projects, no longer having to set them each time a new project is started. Named option sets and further configure build configuration files.

IDE INSIGHT

The new IDE Insight search box enables you to type in a string and then select from a list of all the matching items in the IDE and in your current project environment. The IDE Insight box contains a list of categories such as Commands, Files, Components, Project Options, and so forth.

As you type your search string, IDE Insight performs an incremental search: the IDE Insight box displays only the categories that contain matching items, along with the one "best" match from each category. You can press Alt+A or a button on the IDE Insight dialog box to toggle between showing all categories (with one "best" match per category) or all matches (which might require you to scroll through the list to find the match you seek).


IDE Insight

Fin

Commands (3)
Find Class...
Find in Files... (Shift+Ctrl+F)
Component Palette (3)
TFindDialog (Dialogs)
TIdFinger (Indy Clients)
Preferences (5)
Find text at cursor - Editor options (Editor Options)
Finish incomplete properties - Class completion option (Environment Options | Exp
Auto-complete text to find - Editor options (Editor Options)
Predefined - Ignore special CoClass Flags when importing (Environment Options |
Project Options (8)
Final output directory (-E) (Delphi Compiler)
Bad global symbol definition in object file (Delphi Compiler)
Conditional defines (-D) (Delphi Compiler)
Defines a preprocessor symbol (-d) (Resource Compiler)
FOR-Loop variable may be undefined after loop (Delphi Compiler)
Return value of function might be undefined (Delphi Compiler)

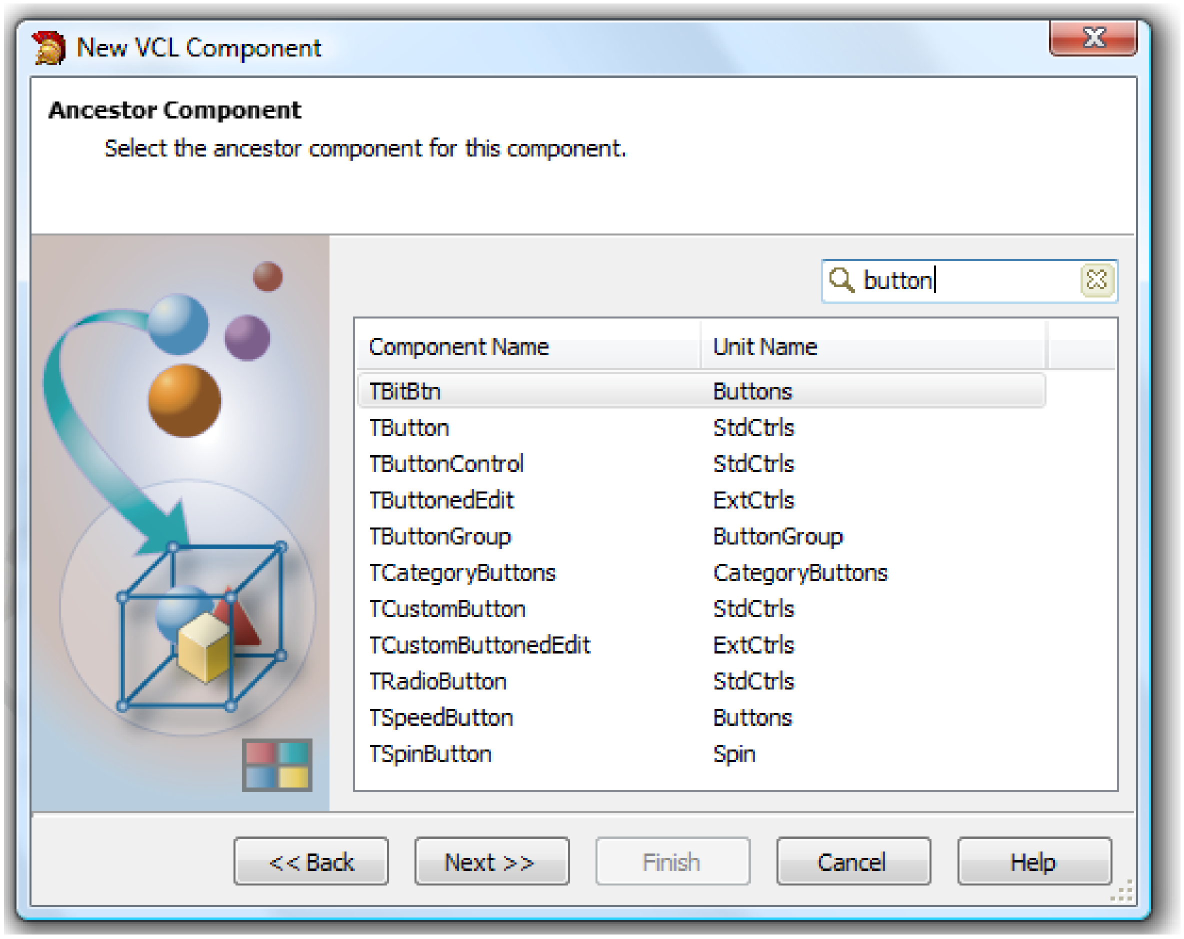
Figure 3. IDE Insight

When you double-click an item in the IDE Insight box, the IDE automatically invokes or performs the associated action. For example, if you type "open", the list displays all the currently available items that contain the string "open". If you double-click the name of a dialog box, the dialog box is invoked. If you double-click a component (such as TOpenDialog), the component is automatically added to the active form.

COMPONENT CREATION WIZARD

The Component Creation and Import Wizards have been redesigned to include type libraries, ActiveX controls and assemblies. Both wizards can now install into an existing package or in a new package.

As you see in Figure 4, a new field was added to filter components, making it easier for you to locate the component you want to inherit.



New VCL Component

Ancestor Component

Select the ancestor component for this component.

button

Component Name	Unit Name
TBitBtn	Buttons
TButton	StdCtrls
TButtonControl	StdCtrls
TButtonedEdit	ExtCtrls
TButtonGroup	ButtonGroup
TCategoryButtons	CategoryButtons
TCustomButton	StdCtrls
TCustomButtonedEdit	ExtCtrls
TRadioButton	StdCtrls
TSpeedButton	Buttons
TSpinButton	Spin

Figure 4. Ancestor Component

COM

COM wizards and the entire type library have been restructured. In fact, the COM Object Creation Wizards are all brand new.

What has changed? A new file type - RIDL (Restricted Interface Definition Language) – was added to the COM architecture. RIDL files work as recording devices projects use to save type libraries. Therefore, the Type Library (.TLB) binary file becomes an intermediary file, like .DCU, .RES, .OBJ, and so on. This means developers are now able to recompile tlb files using the command line prompt, and even edit a tlb file using a text editor, while still keeping track of its version.

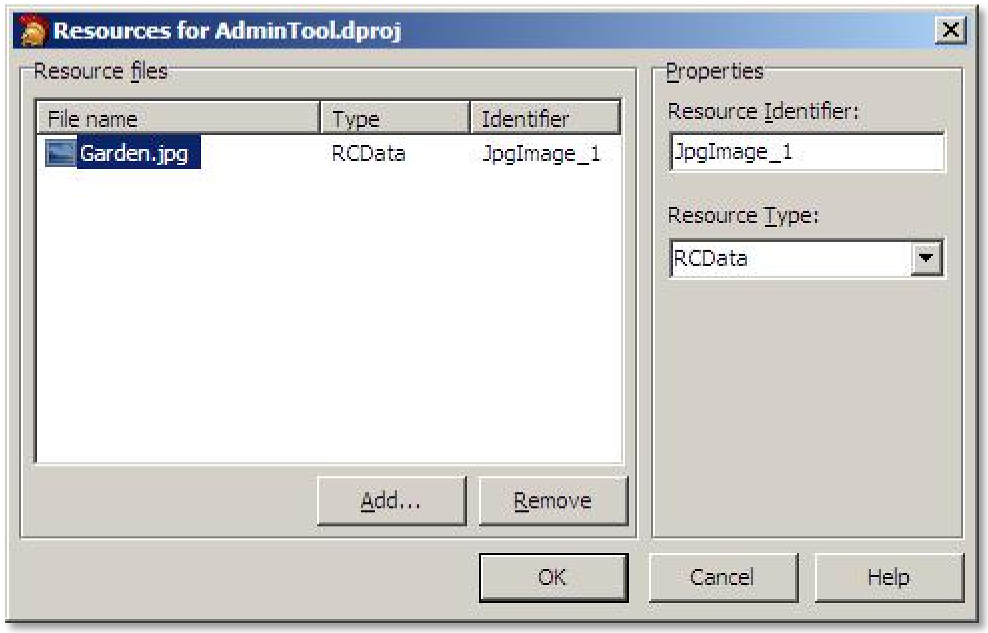
The type library now uses a text file (the RIDL file), not TLB. This is beneficial because:

- You no longer need to check the tlb file in. It's now automatically generated based on the last RIDL file.
- Different developers can work with the same type library. This is so because the text file can now be merged, something that couldn't be done with the binary file used previously.
- The RIDL format provides the Type Library editor with much higher flexibility.
- You can easily compare different RIDL files.

NEW RESOURCE MANAGER

The Resource Compiler allows you to choose between compiling your resources with BRCC32.exe or RC.exe (Microsoft Platform SDK resource compiler). RC supports the use of Unicode characters in resource files and file names. It also supports the new Windows Vista types (e.g., icons and alpha channel). When you use RC you must define #include <winresrc.h> explicitly both for Delphi and C++.

The New Resource Manager allows you to add many resource files (bitmaps, icons, fonts…) to your project.


Resources for AdminTool.dproj

Resource files

File name	Type	Identifier
Garden.jpg	RCData	JpgImage_1

Properties

Resource Identifier: JpgImage_1

Resource Type: RCData

Figure 5. Resource Editor

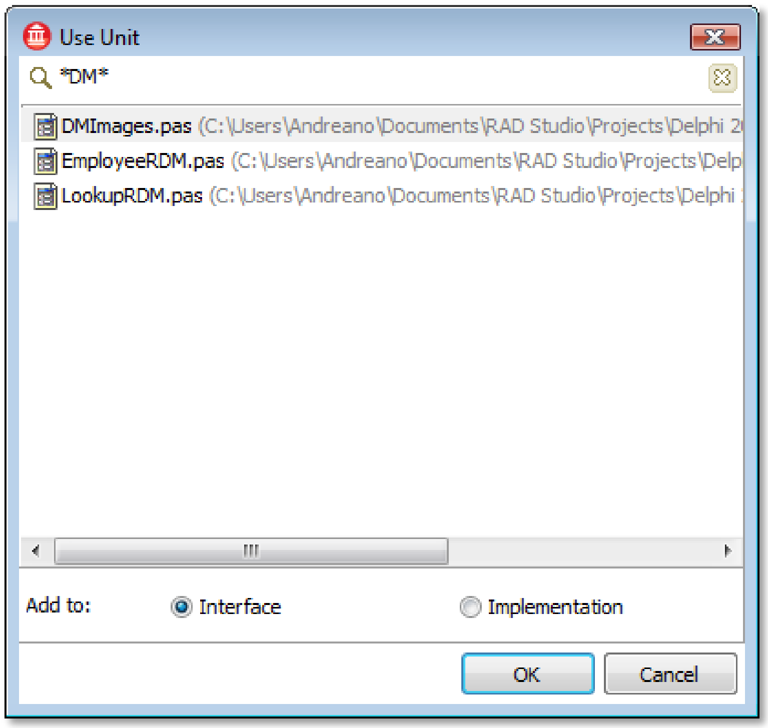
Managing the menu Reopen Files

It is now possible to control the number of files and projects that appear on the File → Reopen menu. You can now specify the amount of projects and files that you want to appear in the list, as well as clean up old files/projects you no longer want in the list.

Use Unit – Interface/Header

Until Delphi 2009 the Use Unit option declared the unit on the section Implementation, now you can define where it will be declared, Interface or Implementation for Delphi code.

In addition, our projects bring tens, hundreds or even thousands of Units, when we need to declare the unit in the code it becomes difficult to do through the Use Unit option, not anymore, now you can use masks to Filter the units and make it easy to find the unit as shown in Figure 6.



DM

DMImages.pas (C:\Users\Andreano\Documents\RAD Studio\Projects\Delphi 2
EmployeeRDM.pas (C:\Users\Andreano\Documents\RAD Studio\Projects\Delp
LookupRDM.pas (C:\Users\Andreano\Documents\RAD Studio\Projects\Delphi

Add to: [✓ Interface] Implementation

Figure 6. New Use Unit window

CLASS EXPLORER

The Class Explorer is a very useful tool that enables you to visualize a project's class hierarchy and its interfaces, as well as add properties, methods and variables to it. These operations can be performed by means of UML, through the use of class models. UML is one of the many resources that were incorporated to Delphi.

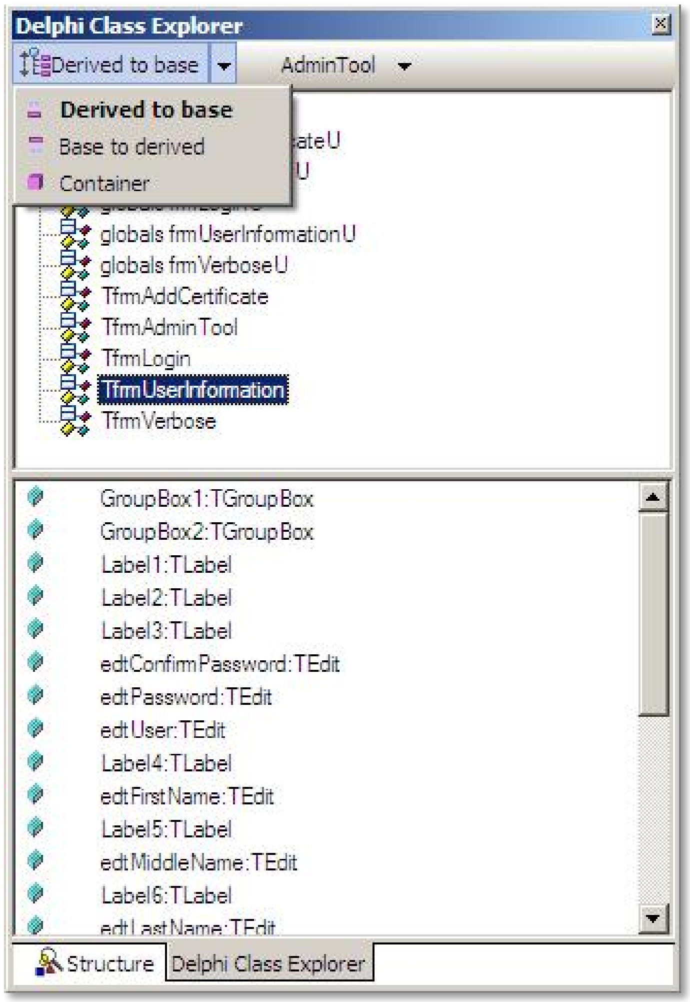


Figure 7. Class Explorer

Component Search in the Tool Palette

In Delphi 2006 you could filter components typing the first couple of letters of their names in the Tool Palette. In Delphi 2007 this feature was enhanced and you were then able to type in any portion of the component name. In Delphi XE the Edit field is used to achieve the same result, making this feature clearer and easy to recognize at first glance.

Users who prefer Delphi 7's layout (i.e., components displayed at the upper portion of the IDE) will be glad to know Delphi XE's IDE can look just like Delphi 7's.

However, before switching to the old Delphi 7 layout, give the Tool Palette in the new version a try. Locating components with ease, the orderly arrangement of categories, etc., can provide great productivity gains.

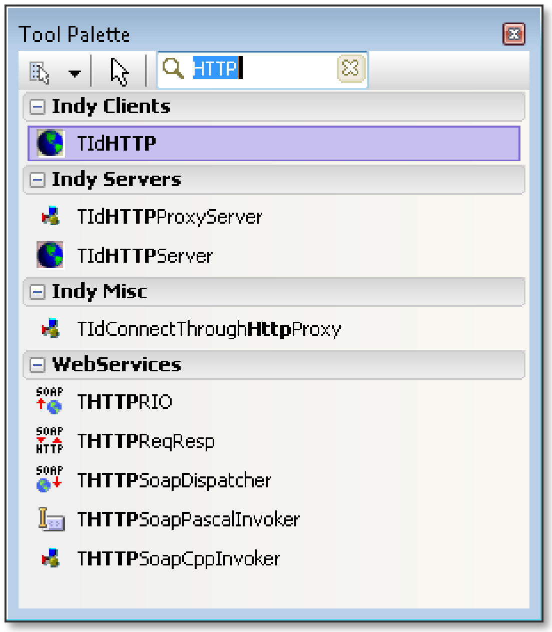


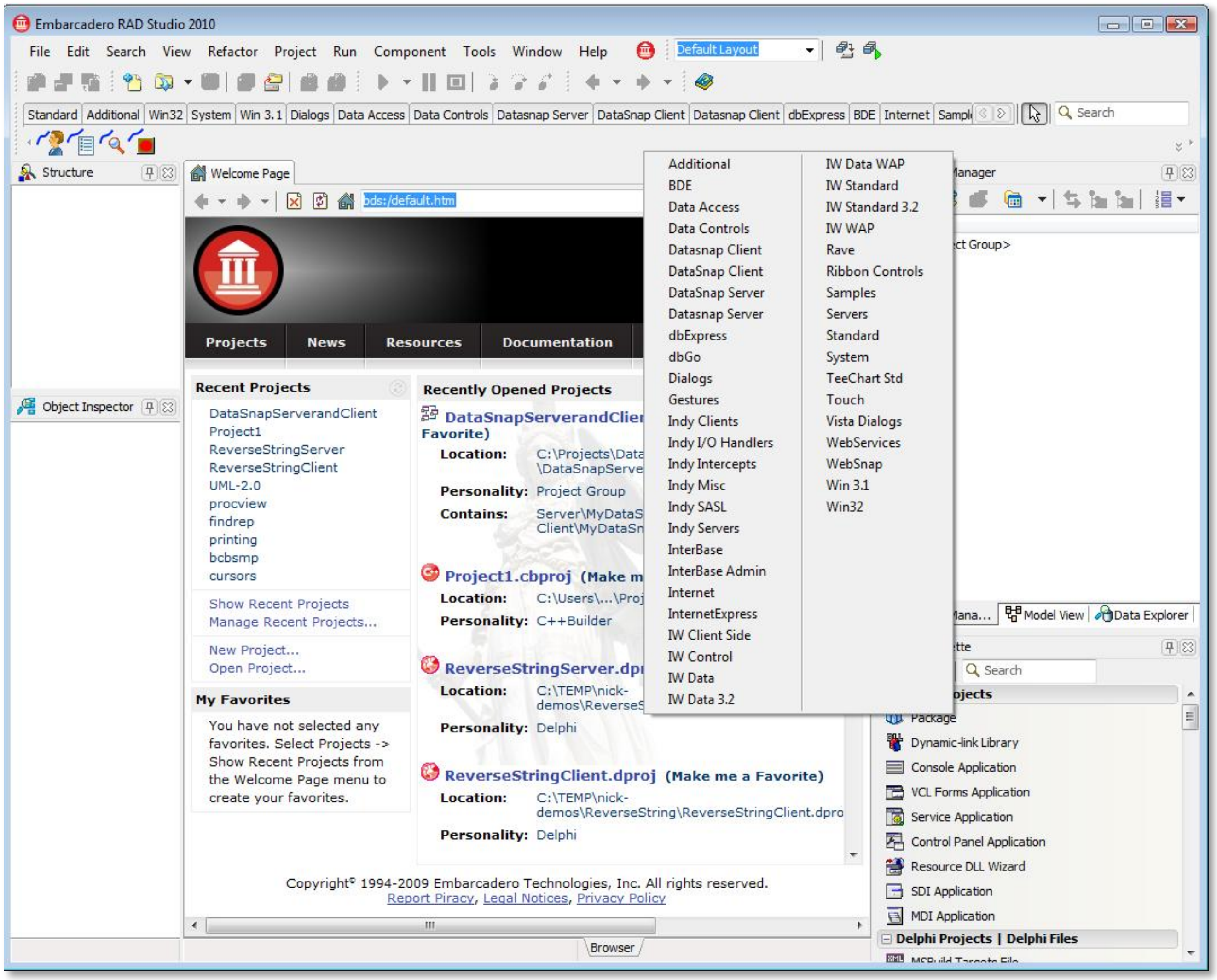
Figure 8. Component Search

The old Component Toolbar is back

We have heard that a lot of developers liked the old Component Toolbar as it was in Delphi 7, and consider that one of the reasons not to move to the new release. In Delphi 2010 (and later versions) you can use the old Component Toolbar and/or the new Tool Palette.

To activate the Component Toolbar just right click on the main toolbar and select Component. After that you can right click on the Component Toolbar to see the list of all categories available or just navigate through the tabs and chose one.

The configuration of the Tool Palette and Component Toolbar are independent, and you can for example reorder the categories. There is also a search box for the components on the Component Toolbar.



Recent Projects
DataSnapServerandClient
Project1
ReverseStringServer
ReverseStringClient
UML-2.0
procview
findrep
printing
bcbsmp
cursors
Show Recent Projects
Manage Recent Projects...
New Project...
Open Project...

My Favorites
You have not selected any favorites. Select Projects -> Show Recent Projects from the Welcome Page menu to create your favorites.

Recently Opened Projects
DataSnapServerandClie
Favorite)
Location: C:\Projects\Data \DataSnapServe
Personality: Project Group
Contains: Server\MyDataS Client\MyDataSn

Project1.cbproj (Make m
Location: C:\Users\...\Proj
Personality: C++Builder

ReverseStringServer.dp
Location: C:\TEMP\nick-demos\ReverseS
Personality: Delphi

ReverseStringClient.dproj (Make me a Favorite)
Location: C:\TEMP\nick-demos\ReverseString\ReverseStringClient.dpro
Personality: Delphi

Copyright© 1994-2009 Embarcadero Technologies, Inc. All rights reserved.
Report Piracy, Legal Notices, Privacy Policy

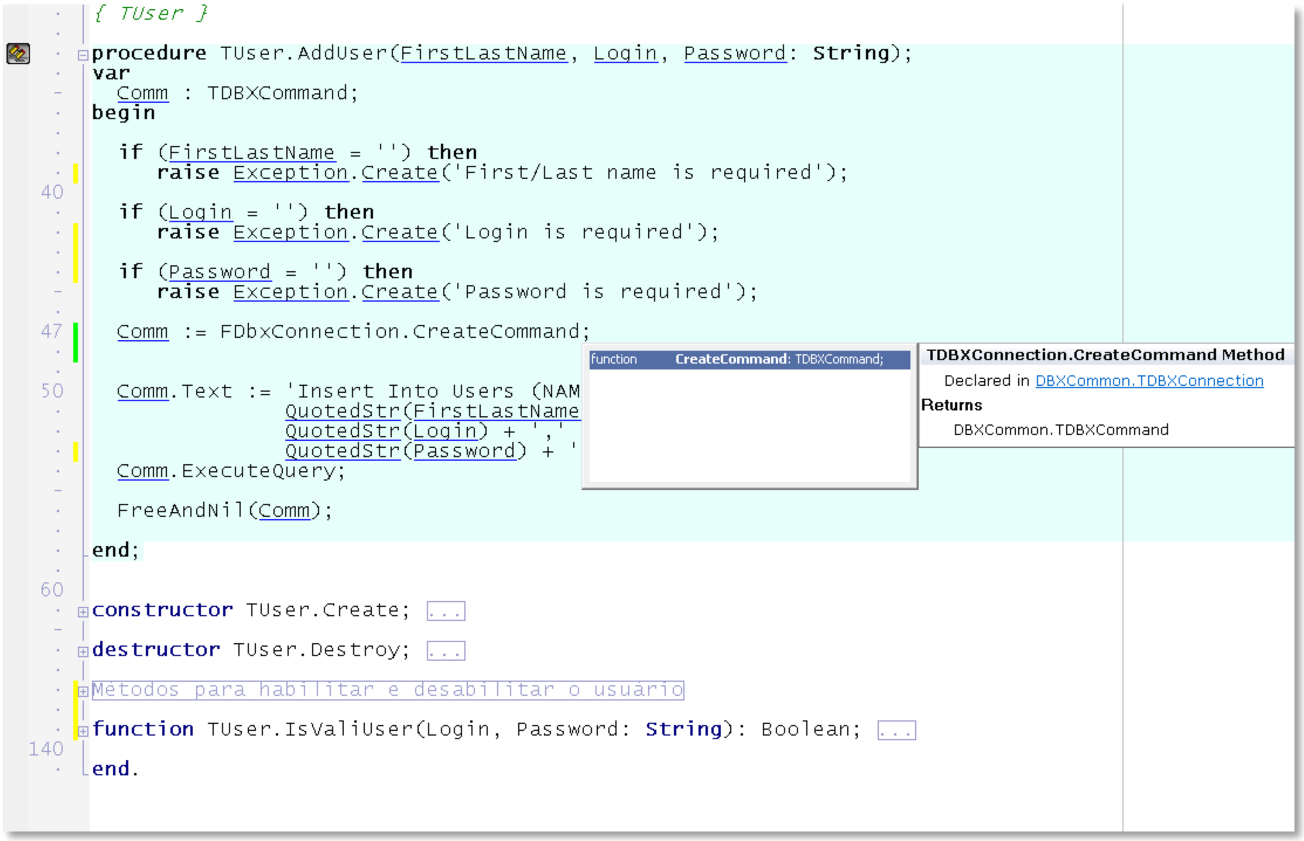
Figure 9. Delphi IDE using Delphi 7 layout

Code Editor

The new *Live Templates* feature implemented in Delphi 2006 extends your ability to create code templates in Delphi. These are created as XML files, and help you program with less code.

Block completion is one of the resources involved in enabling automatic begin and end. And who can honestly say he's never had a hard time with begin..end?

Consider this context: you want to change the name of all variables in a selected part of the code. Find..Replace is not a good practice for this situation. It doesn't guarantee only variable names will be changed in the process. Since Delphi 8 you can use Sync Edit to edit different portions of code simultaneously, provided they share the same identifier. Taking the code below as an example, you could select the entire block, make sure sync edit is active and then change the "Comm" variable a single time.



```
{ TUser }

procedure TUser.AddUser(FirstLastName, Login, Password: String);
var
  Comm : TDBXCommand;
begin

   if (FirstLastName = '') then
      raise Exception.Create('First/Last name is required');

   if (Login = '') then
      raise Exception.Create('Login is required');

   if (Password = '') then
      raise Exception.Create('Password is required');

  Comm := FDbxConnection.CreateCommand;

  Comm.Text := 'Insert Into Users (NAM
               QuotedStr(FirstLastName
               QuotedStr(Login) + ','
               QuotedStr(Password) + '
  Comm.ExecuteQuery;

  FreeAndNil(Comm);

end;

constructor TUser.Create; ...

destructor TUser.Destroy; ...

Métodos para habilitar e desabilitar o usuário

function TUser.IsValiUser(Login, Password: String): Boolean; ...

end.
```

function CreateCommand: TDBXCommand;

TDBXConnection.CreateCommand Method
Declared in DBXCommon.TDBXConnection
Returns
DBXCommon.TDBXCommand

Figure 10. Live Templates, sync edit, code folding and other source code editor features

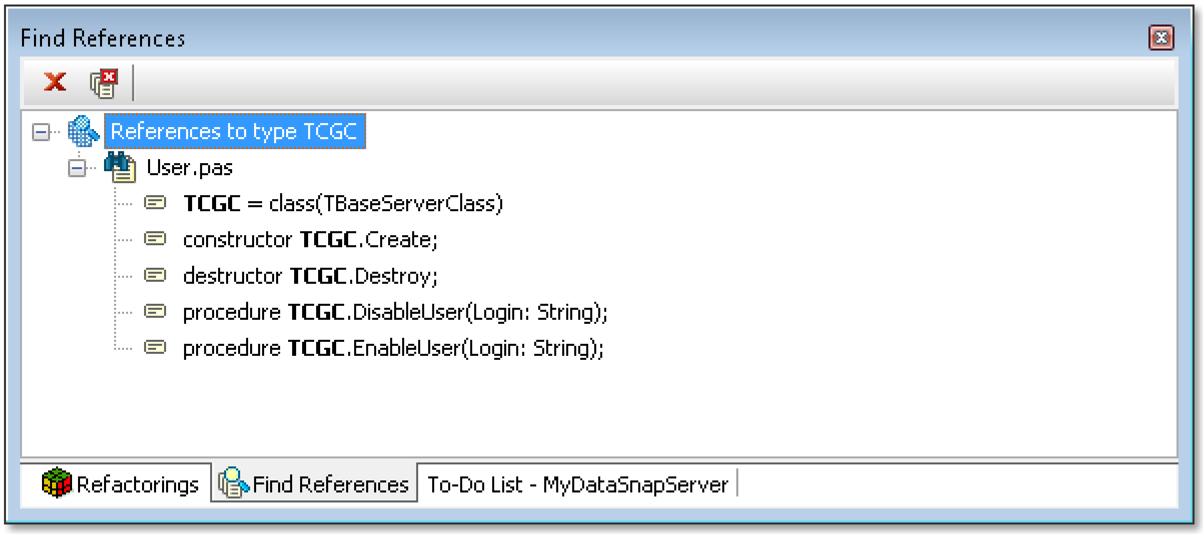
Alongside the code block you see yellow and green marks. The yellow ones are shown for lines that have changed since the last Save. Green marks, in turn, indicate lines that were recently changed and saved.

You also see the smart code line numbering. And you're able to either expand or collapse a method or a class right within the block.

Think about a unit with tens of methods. You hope one day you'll have enough time to stop and set it in order, but never really find time to do so. The code above holds a region called "Methods for enabling/blocking user access". This region has two methods for either enabling or blocking users. You can't see those methods, unless their region is expanded. A question remains: wouldn't it be easier to organize and visualize the code with the assistance of such features?

Help from within the code (Help Insight): Press F1 in order to see the documentation of a method, type, class, etc. As you can see, the code above displays the CreateCommand method help info. The same thing happens with any method, type or class, provided it has a description available.

Find references for a method, class, variable, or any other specific item. Imagine your code holds a class named TCGC, which you want to rename to TCNPJ. Assuming you're a careful developer, tell me where the TCGC class is referenced in the project. Find..Replace won't work this time. Instead, try pressing Shift + CTRL + Enter over class TCGC. The IDE then looks for all references in the project, as seen below.



```
TCGC = class(TBaseServerClass)
constructor TCGC.Create;
destructor TCGC.Destroy;
procedure TCGC.DisableUser(Login: String);
procedure TCGC.EnableUser(Login: String);
```

Figure 11. Find References with Live Template

If you're now wondering how to rename all classes to TCNPJ, wait until we discuss Refactoring.

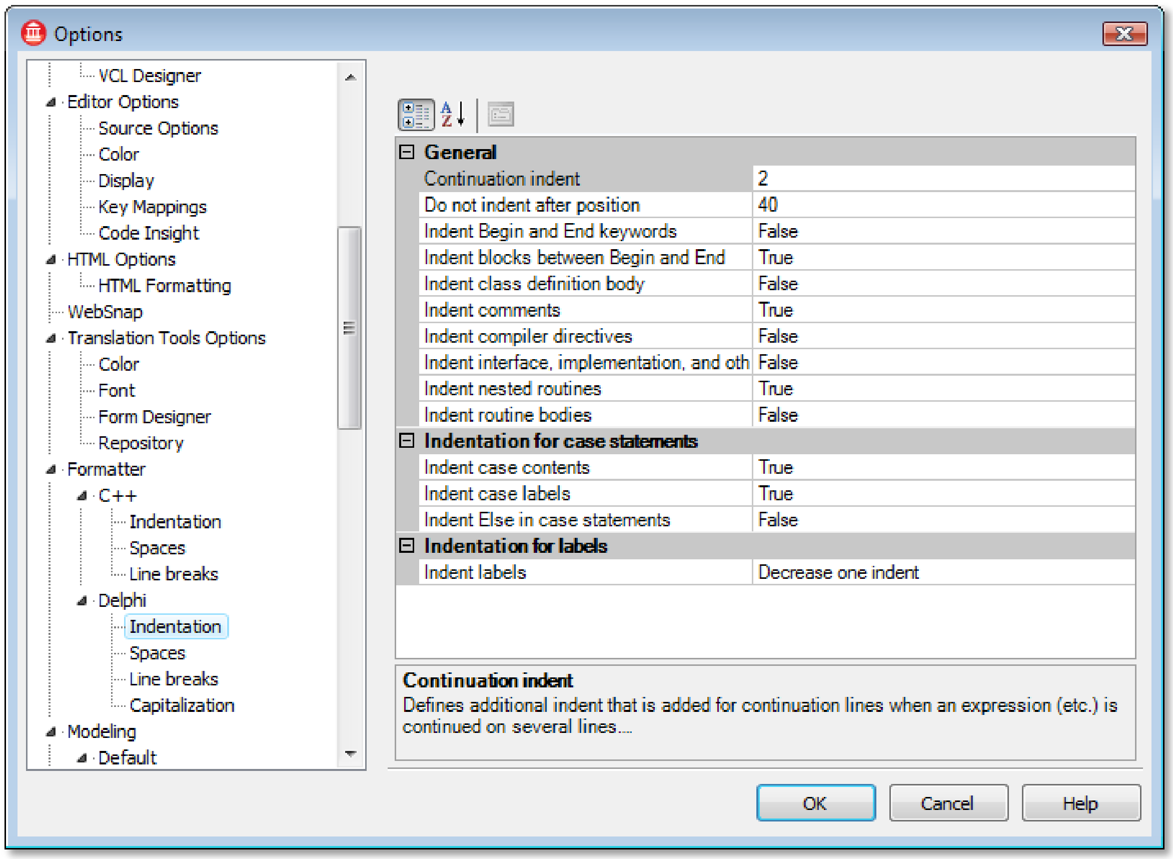
Another useful feature is named Surround. It works basically by allowing you to add begin/end, if/begin/end, try/finally, try/except, etc., to a block of code.

SOURCE CODE FORMATTER

Delphi has its default formatting code, but we all know that many developers follow this pattern and many have their own style, this is cause for great discussion, but that is no longer a problem.

The IDE now provides a fully customizable code formatter for accessible via CTRL + D. This allows you to have your unit formatted according to the desired settings. Furthermore, you can use the Project Manager to format all units included in the project.

The IDE Insight Formatter (Figure 12) allows you to set your own options for Indentation, Spaces, Line breaks and Capitalization, and your code will be formatted according to these options when you invoke the code formatter.



General	
Continuation indent	2
Do not indent after position	40
Indent Begin and End keywords	False
Indent blocks between Begin and End	True
Indent class definition body	False
Indent comments	True
Indent compiler directives	False
Indent interface, implementation, and oth	False
Indent nested routines	True
Indent routine bodies	False
Indentation for case statements	
Indent case contents	True
Indent case labels	True
Indent Else in case statements	False
Indentation for labels	
Indent labels	Decrease one indent

Continuation indent
Defines additional indent that is added for continuation lines when an expression (etc.) is continued on several lines....

Figure 12. Code Formatter configuration window

There is also Profile support, which allows you to maintain and switch between several sets of formatting options used by the source code formatter. The formatting options are stored in configuration files. By selecting a profile you activate the set of formatting options stored in this profile.

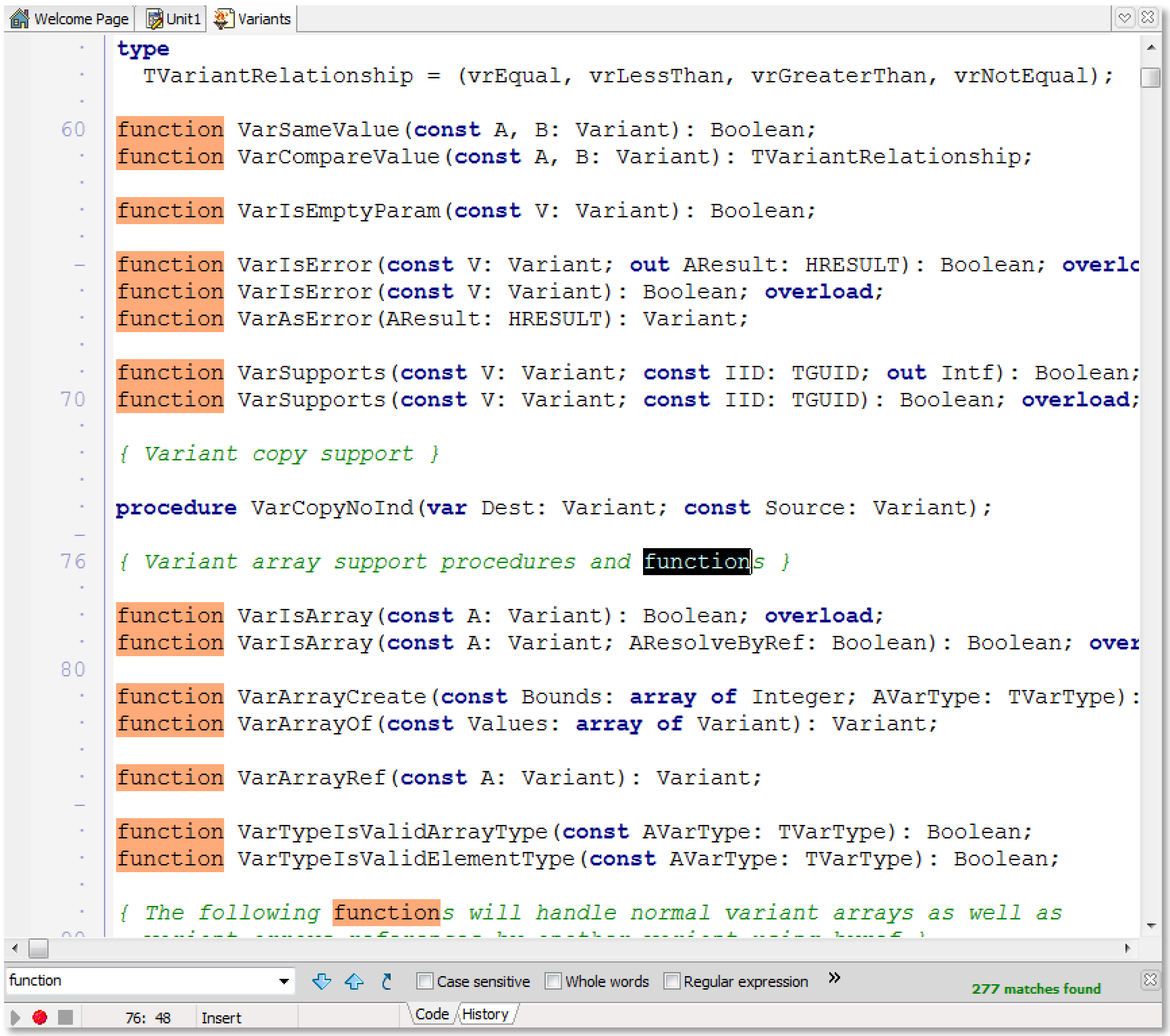
To create a configuration file use the Save As command button. By default, configuration files have the .config extension.

Profiles are special kinds of configuration files. Profiles have the following additional properties:

- Profiles should match the Formatter_*.config file mask (profile mask).
- Profiles should be located in the RAD Studio working directory. By default, this is the "%APPDATA%\Embarcadero\RAD Studio\VersionNumber" directory.

Source Code Editor SEARCH

The search feature in the code editor has been overhauled and is based on other popular search implementations (e.g. Firefox and Internet Explorer).Now when you press CTRL + F and type a search word you will see results as shown in Figure 13, where the first content from the cursor will be highlighted in black and the remaining orange.



```
type
  TVariantRelationship = (vrEqual, vrLessThan, vrGreaterThan, vrNotEqual);

function VarSameValue(const A, B: Variant): Boolean;
function VarCompareValue(const A, B: Variant): TVariantRelationship;

function VarIsEmptyParam(const V: Variant): Boolean;

function VarIsError(const V: Variant; out AResult: HRESULT): Boolean; overlc
function VarIsError(const V: Variant): Boolean; overload;
function VarAsError(AResult: HRESULT): Variant;

function VarSupports(const V: Variant; const IID: TGUID; out Intf): Boolean;
function VarSupports(const V: Variant; const IID: TGUID): Boolean; overload;

{ Variant copy support }

procedure VarCopyNoInd(var Dest: Variant; const Source: Variant);

{ Variant array support procedures and functions }

function VarIsArray(const A: Variant): Boolean; overload;
function VarIsArray(const A: Variant; AResolveByRef: Boolean): Boolean; over

function VarArrayCreate(const Bounds: array of Integer; AVarType: TVarType):
function VarArrayOf(const Values: array of Variant): Variant;

function VarArrayRef(const A: Variant): Variant;

function VarTypeIsValidArrayType(const AVarType: TVarType): Boolean;
function VarTypeIsValidElementType(const AVarType: TVarType): Boolean;

{ The following functions will handle normal variant arrays as well as
```

function | Case sensitive [ ] | Whole words [ ] | Regular expression [ ] | 277 matches found

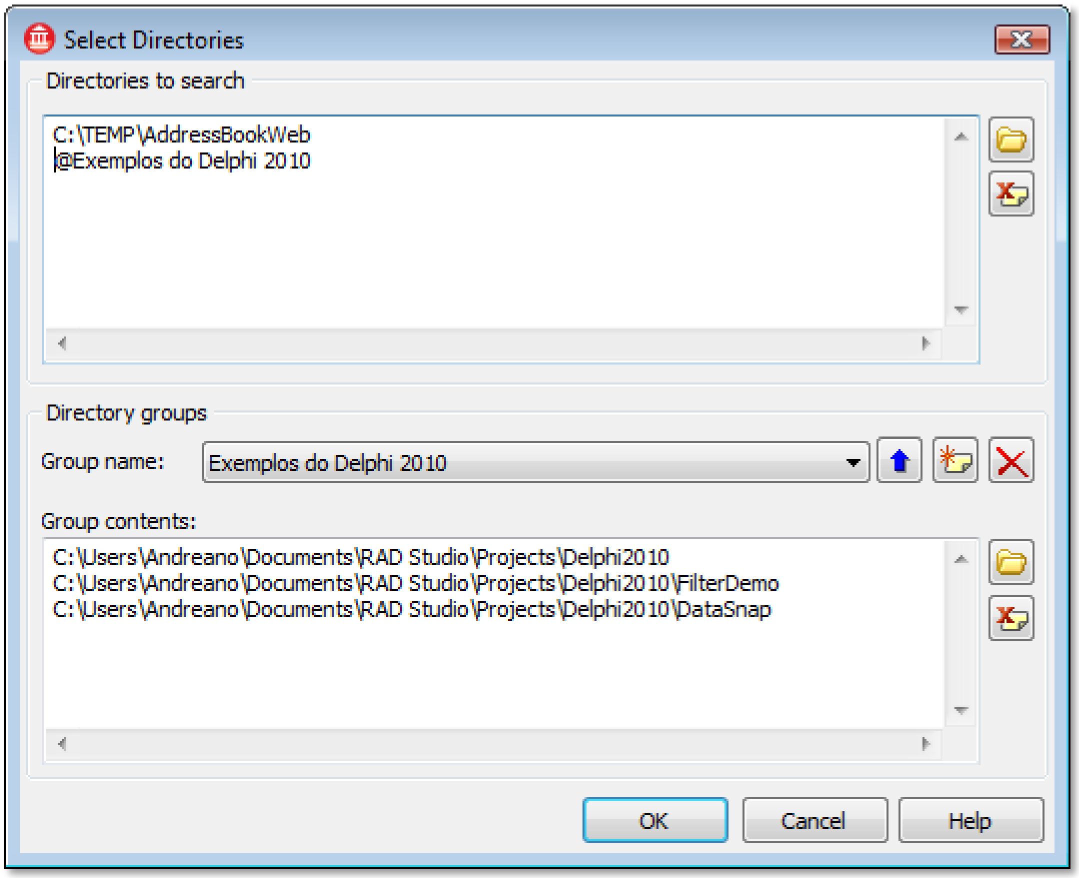
76: 48 Insert | Code | History

Figure 13. Searching for the word "function", yields the above result.

SEARCH IN FILE

Find content in files is a common task and take time, almost time we try to find content in directories that are not part of the project, repeating this operation many times is stressful.

In Delphi 2010 we added a feature to enable you to select directories for "Find in files" easier. You can now save groups of directories that you use on a regular basis in conjunction with "Find in Files" (See Figure 14).



```
C:\TEMP\AddressBookWeb
@Exemplos do Delphi 2010
```

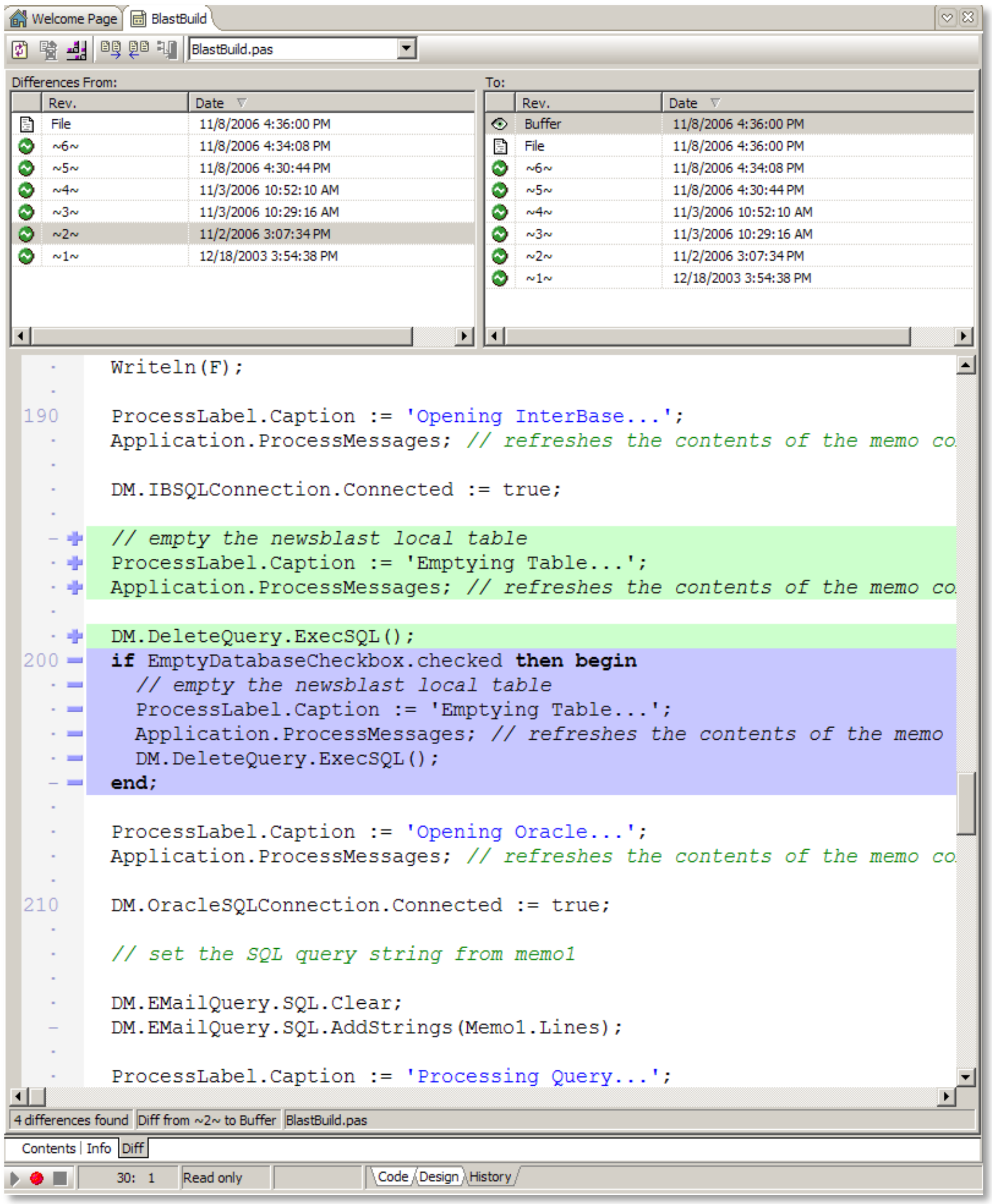
Group name: Exemplos do Delphi 2010

```
C:\Users\Andreano\Documents\RAD Studio\Projects\Delphi2010
C:\Users\Andreano\Documents\RAD Studio\Projects\Delphi2010\FilterDemo
C:\Users\Andreano\Documents\RAD Studio\Projects\Delphi2010\DataSnap
```

Figure 14. Search in File window

CHANGE HISTORY

Files are locally versioned whenever they're changed, even in the absence of version control, thus allowing you to make comparisons between them.



Differences From:

	Rev.	Date
	File	11/8/2006 4:36:00 PM
	~6~	11/8/2006 4:34:08 PM
	~5~	11/8/2006 4:30:44 PM
	~4~	11/3/2006 10:52:10 AM
	~3~	11/3/2006 10:29:16 AM
	~2~	11/2/2006 3:07:34 PM
	~1~	12/18/2003 3:54:38 PM

To:

	Rev.	Date
	Buffer	11/8/2006 4:36:00 PM
	File	11/8/2006 4:36:00 PM
	~6~	11/8/2006 4:34:08 PM
	~5~	11/8/2006 4:30:44 PM
	~4~	11/3/2006 10:52:10 AM
	~3~	11/3/2006 10:29:16 AM
	~2~	11/2/2006 3:07:34 PM
	~1~	12/18/2003 3:54:38 PM

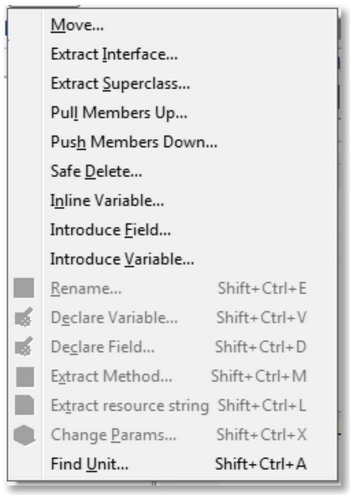
```
Writeln(F);

ProcessLabel.Caption := 'Opening InterBase...';
Application.ProcessMessages; // refreshes the contents of the memo co

DM.IBSQLConnection.Connected := true;

// empty the newsblast local table
ProcessLabel.Caption := 'Emptying Table...';
Application.ProcessMessages; // refreshes the contents of the memo co

DM.DeleteQuery.ExecSQL();
if EmptyDatabaseCheckbox.checked then begin
  // empty the newsblast local table
  ProcessLabel.Caption := 'Emptying Table...';
  Application.ProcessMessages; // refreshes the contents of the memo
  DM.DeleteQuery.ExecSQL();
end;

ProcessLabel.Caption := 'Opening Oracle...';
Application.ProcessMessages; // refreshes the contents of the memo co

DM.OracleSQLConnection.Connected := true;

// set the SQL query string from memo1

DM.EMailQuery.SQL.Clear;
DM.EMailQuery.SQL.AddStrings(Memo1.Lines);

ProcessLabel.Caption := 'Processing Query...';
```

Figure 15. Change History

If your project is integrated with Subversion, the Change History will list the file revisions from your version control system.

REFACTORING

Delphi 7 users will surely love to try this resource. Refactoring is a technique you can use to restructure and modify your existing code in such a way that the intended behavior of your code stays the same. Refactoring allows you to streamline, simplify, and improve both performance and readability of your application code.


Move...
Extract Interface...
Extract Superclass...
Pull Members Up...
Push Members Down...
Safe Delete...
Inline Variable...
Introduce Field...
Introduce Variable...
Rename... Shift+Ctrl+E
Declare Variable... Shift+Ctrl+V
Declare Field... Shift+Ctrl+D
Extract Method... Shift+Ctrl+M
Extract resource string Shift+Ctrl+L
Change Params... Shift+Ctrl+X
Find Unit... Shift+Ctrl+A

Delphi includes a refactoring engine that evaluates and executes the refactoring operation. The engine also displays a preview of the changes that will occur in a refactoring pane that appears at the bottom of the Code Editor. The potential refactoring operations are displayed as tree nodes, which can be expanded to show additional items that might be affected by the refactoring, if they exist. Warnings and errors also appear in this pane. You can access the refactoring tools from the Main menu and from context-sensitive drop down menus.

UNIT TESTING

Delphi integrates an open-source testing framework, DUnit, for developing and running automated test cases for your applications. This framework simplifies the process of developing tests for classes and methods in your application. Using unit testing in combination with refactoring can improve your application's stability. Running a standard set of tests every time a small change is made throughout the code makes it more likely that you will catch any problems early in the development cycle.

DATA EXPLORER

It's now easier to retrieve data from databases. Using the Data Explorer along with drag-and-drop capabilities you can access tables, views, stored procedures and other database items. Besides that, you can also search for data using SQL.

Connections are established through dbExpress. This means Data Explorer supports every database dbExpress supports.

Each connection is assigned an alias which is saved in the dbxconnections.ini file (dbExpress' configuration file). Aliases are treated as shared information, thus facilitating the use of Data Explorer.

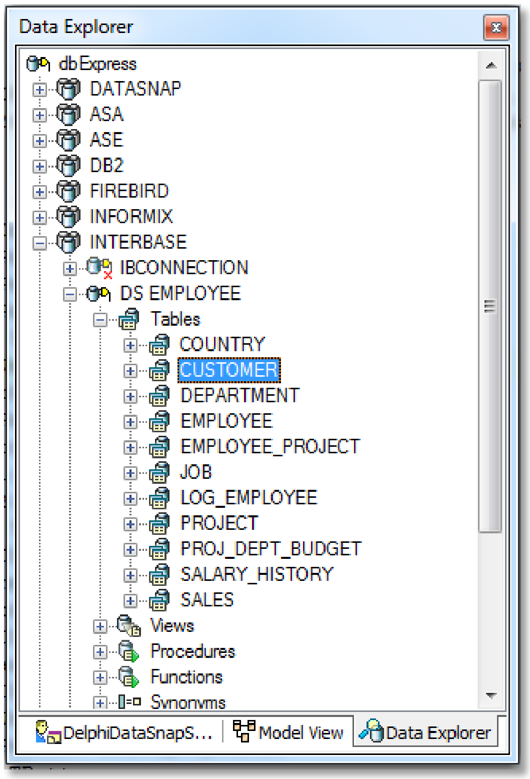
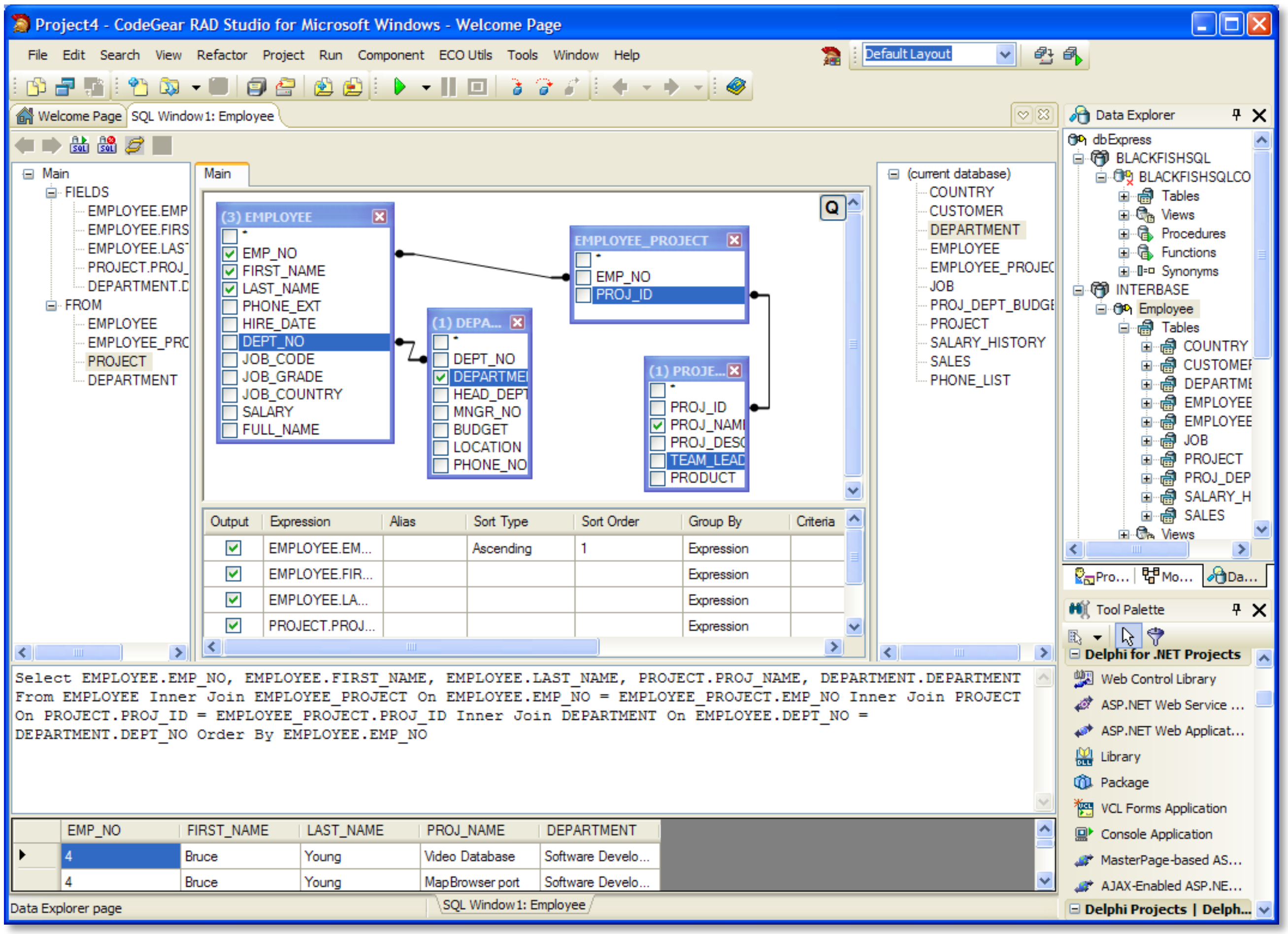


Figure 16. Data Explorer

SQL Window - Query Builder

The Data Explorer allows developers to easily build complex SQL queries via an intuitive visual query builder interface.



Output	Expression	Alias	Sort Type	Sort Order	Group By	Criteria
✓	EMPLOYEE.EM...		Ascending	1	Expression	
✓	EMPLOYEE.FIR...				Expression	
✓	EMPLOYEE.LA...				Expression	
✓	PROJECT.PROJ...				Expression	

```
Select EMPLOYEE.EMP_NO, EMPLOYEE.FIRST_NAME, EMPLOYEE.LAST_NAME, PROJECT.PROJ_NAME, DEPARTMENT.DEPARTMENT
From EMPLOYEE Inner Join EMPLOYEE_PROJECT On EMPLOYEE.EMP_NO = EMPLOYEE_PROJECT.EMP_NO Inner Join PROJECT
On PROJECT.PROJ_ID = EMPLOYEE_PROJECT.PROJ_ID Inner Join DEPARTMENT On EMPLOYEE.DEPT_NO =
DEPARTMENT.DEPT_NO Order By EMPLOYEE.EMP_NO
```

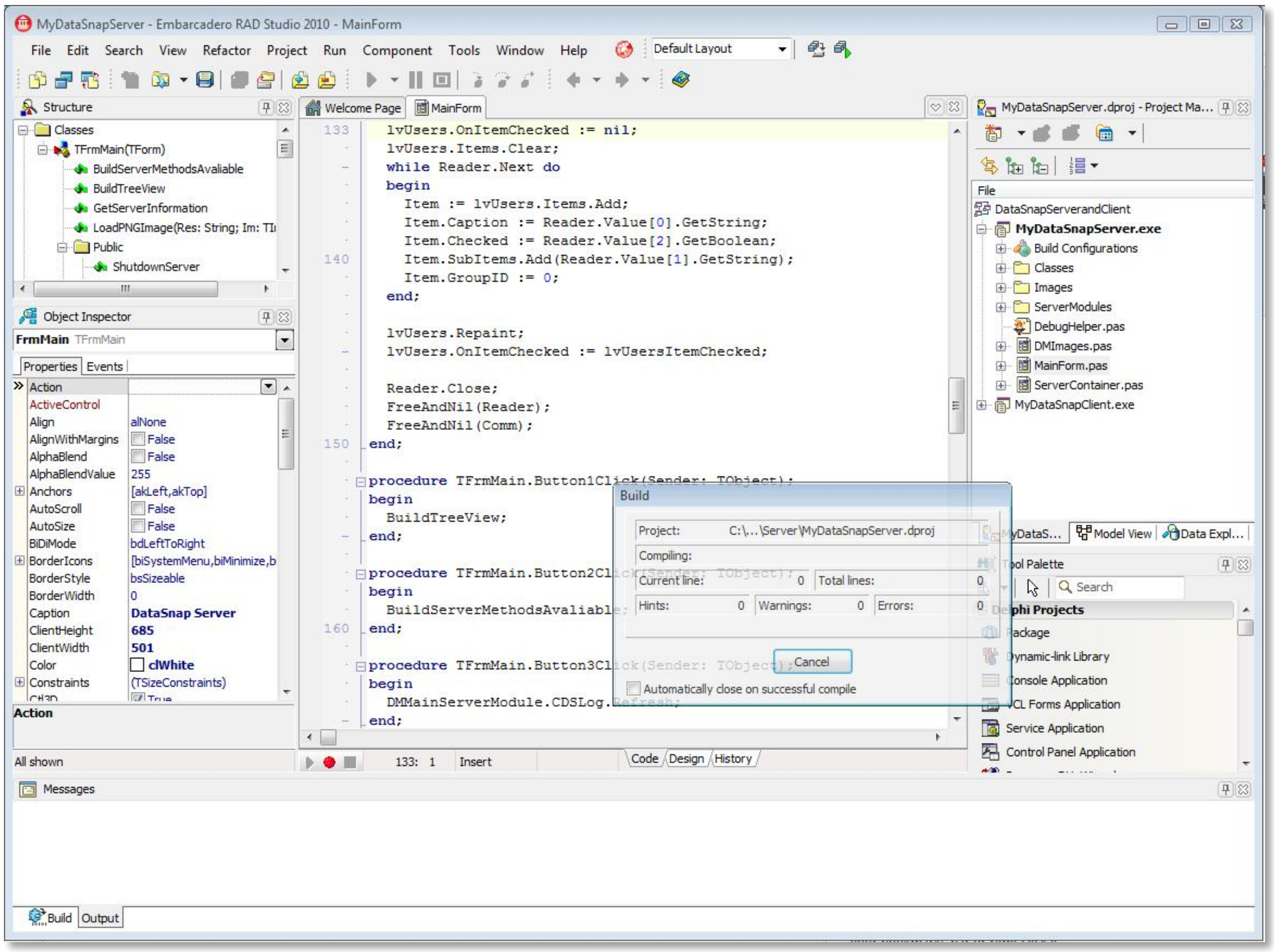
EMP_NO	FIRST_NAME	LAST_NAME	PROJ_NAME	DEPARTMENT
4	Bruce	Young	Video Database	Software Develo...
4	Bruce	Young	MapBrowser port	Software Develo...

Figure 17. SQL Window

BACKGROUND COMPILATION

Since Delphi 2010 you can perform background compilation — that is, you can start a compile running as a separate and parallel thread and continue working in the IDE while the compiler compiles your project.

You can continue to work in the IDE while a background compilation runs. For example, you can edit files, even the files you are compiling, and you can set and modify breakpoints.



```
    lvUsers.OnItemChecked := nil;
    lvUsers.Items.Clear;
    while Reader.Next do
    begin
      Item := lvUsers.Items.Add;
      Item.Caption := Reader.Value[0].GetString;
      Item.Checked := Reader.Value[2].GetBoolean;
      Item.SubItems.Add(Reader.Value[1].GetString);
      Item.GroupID := 0;
    end;

    lvUsers.Repaint;
    lvUsers.OnItemChecked := lvUsersItemChecked;

    Reader.Close;
    FreeAndNil(Reader);
    FreeAndNil(Comm);
  end;

procedure TFrmMain.Button1Click(Sender: TObject);
begin
  BuildTreeView;
end;

procedure TFrmMain.Button2Click(Sender: TObject);
begin
  BuildServerMethodsAvaliable;
end;

procedure TFrmMain.Button3Click(Sender: TObject);
begin
  DMMainServerModule.CDSLog.Refresh;
end;
```

Action	
ActiveControl	
Align	alNone
AlignWithMargins	False
AlphaBlend	False
AlphaBlendValue	255
Anchors	[akLeft,akTop]
AutoScroll	False
AutoSize	False
BiDiMode	bdLeftToRight
BorderIcons	[biSystemMenu,biMinimize,b
BorderStyle	bsSizeable
BorderWidth	0
Caption	DataSnap Server
ClientHeight	685
ClientWidth	501
Color	clWhite
Constraints	(TSizeConstraints)

Build

Project: C:\...\Server\MyDataSnapServer.dproj

Compiling:

Current line: 0   Total lines: 0

Hints: 0   Warnings: 0   Errors: 0

Cancel

Automatically close on successful compile

Figure 18. Compiler dialog transparent during the compiler process running in background

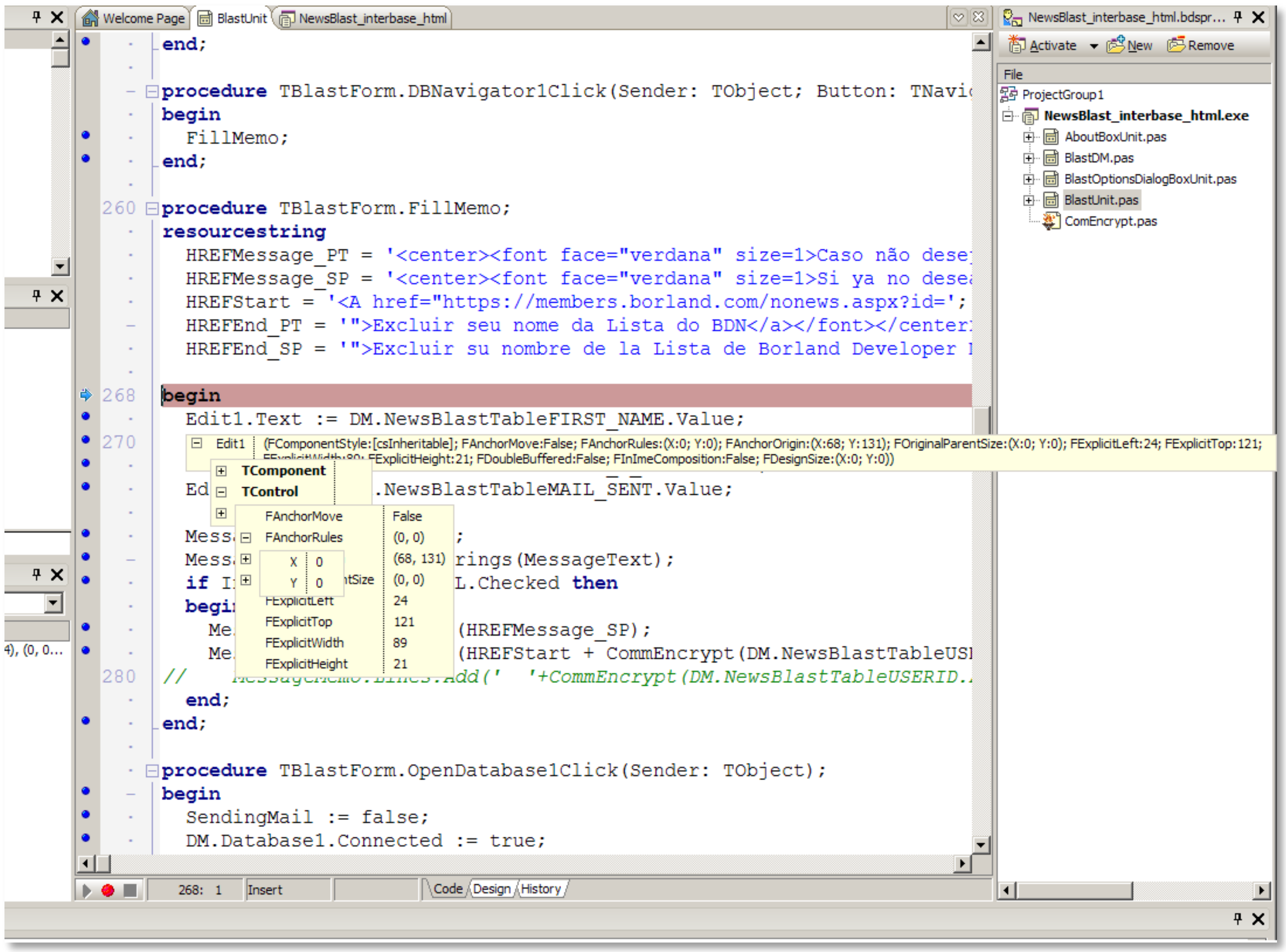
DEBUGGER

The debugger now comes with the "Thread View and Wait Chain Traversal" feature, available only for Windows Vista and Windows 7. This resource helps you locate deadlocks and thread contentions.

During the debug process it is helpful to visualize the content of variables. The Watch List comes in handy for that, but the higher the number of items added the more confusing the visualization gets. Developers can now group Watch List variables based on custom names. Custom variable groups are then represented as tabs in the Watch List.

After the debug is finished, all units that are opened during the process are automatically closed; only units that remain open are the ones that were open before the debug process was started.

There are a lot of improvements with the visualization and usability of local variable window, call stack and others. There is also a new tree view for the content of objects that are being debugged, shown in Figure 19:



```
end;

procedure TBlastForm.DBNavigator1Click(Sender: TObject; Button: TNavi
begin
  FillMemo;
end;

procedure TBlastForm.FillMemo;
resourcestring
  HREFMessage_PT = '<center><font face="verdana" size=1>Caso não dese
  HREFMessage_SP = '<center><font face="verdana" size=1>Si ya no dese
  HREFStart = '<A href="https://members.borland.com/nonews.aspx?id=';
  HREFEnd_PT = '">Excluir seu nome da Lista do BDN</a></font></center
  HREFEnd_SP = '">Excluir su nombre de la Lista de Borland Developer 

begin
  Edit1.Text := DM.NewsBlastTableFIRST_NAME.Value;
```

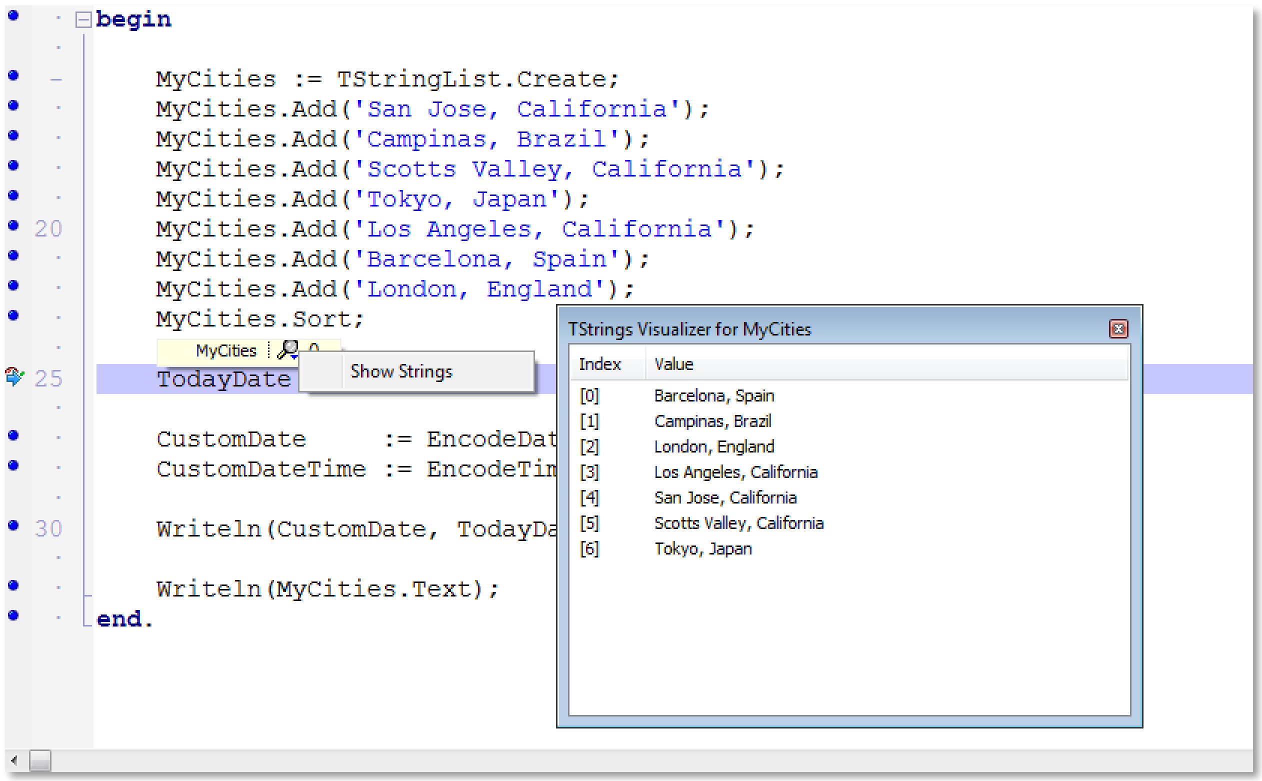
```
Edit1 | (FComponentStyle:[csInheritable]; FAnchorMove:False; FAnchorRules:(X:0; Y:0); FAnchorOrigin:(X:68; Y:131); FOriginalParentSize:(X:0; Y:0); FExplicitLeft:24; FExplicitTop:121;
      FExplicitHeight:21; FDoubleBuffered:False; FInImeComposition:False; FDesignSize:(X:0; Y:0))
  TComponent
  TControl
    FAnchorMove     False
    FAnchorRules    (0, 0)
      X  0
      Y  0
                    (68, 131)
                    (0, 0)
    FExplicitLeft   24
    FExplicitTop    121
    FExplicitWidth  89
    FExplicitHeight 21
```

```
  Ed                 .NewsBlastTableMAIL_SENT.Value;

  Mess                ;
  Mess                rings(MessageText);
  if I                L.Checked then
  begi
    Me                (HREFMessage_SP);
    Me                (HREFStart + CommEncrypt(DM.NewsBlastTableUS
//                   '+CommEncrypt(DM.NewsBlastTableUSERID..
  end;
end;

procedure TBlastForm.OpenDatabase1Click(Sender: TObject);
begin
  SendingMail := false;
  DM.Database1.Connected := true;
```

Figure 19. Debugger

Starting with Delphi 2010 it's much easier to debug a multi-threaded application. New features in the debugger allow for step-by-step debugging in threads, as well as freezing and thawing of threads. This way you can isolate only the threads that want to debug, freezing all irrelevant ones. There are options in the “Thread Status View” to "Freeze" "Freeze All Other Threads", "Thaw" and "Thaw All Threads". Also, there is now a breakpoint option to only break on a specific thread.

Debugger Visualizers make it much easier to debug your data. How many times have you needed to debug a variable of type TDate, TTime TDateTime? The IDE now shows these as you would expect – human readable dates and times, instead of floating point values. Similarly for TStringLists – the list of strings is shown in plain text
.
Furthermore, you can add your own Debugger Visualizers for your own custom types.



```
begin

    MyCities := TStringList.Create;
    MyCities.Add('San Jose, California');
    MyCities.Add('Campinas, Brazil');
    MyCities.Add('Scotts Valley, California');
    MyCities.Add('Tokyo, Japan');
    MyCities.Add('Los Angeles, California');
    MyCities.Add('Barcelona, Spain');
    MyCities.Add('London, England');
    MyCities.Sort;

    TodayDate

    CustomDate     := EncodeDat
    CustomDateTime := EncodeTim

    Writeln(CustomDate, TodayDa

    Writeln(MyCities.Text);
end.
```

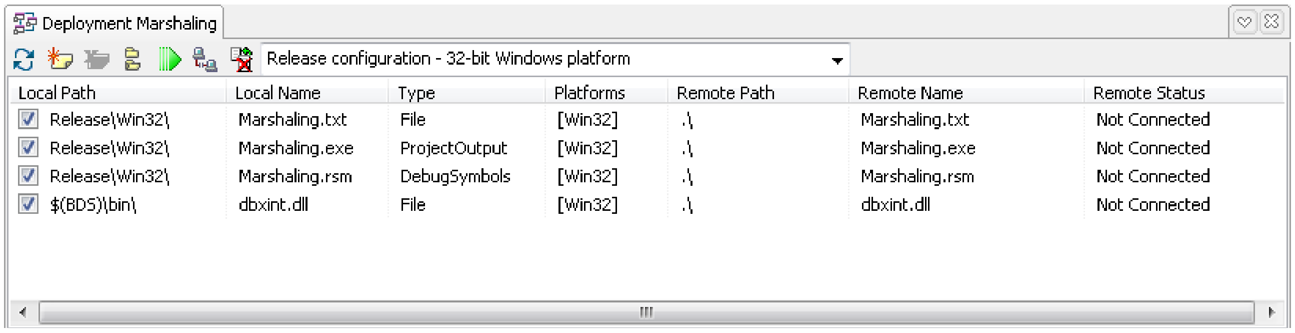
TStrings Visualizer for MyCities

Index	Value
[0]	Barcelona, Spain
[1]	Campinas, Brazil
[2]	London, England
[3]	Los Angeles, California
[4]	San Jose, California
[5]	Scotts Valley, California
[6]	Tokyo, Japan

Figure 20. Data Visualizer for TStrings data type

DEPLOYMENT MANAGER

The Deployment Manager can deploy a cross-platform application as well as native 32-bit and 64-bit Windows applications. The Deployment Manager works with the Platform Assistant, so the application being deployed must use the Platform Assistant and a remote profile. The Deployment Manager allows you to enable, view, add, delete, or edit the files that are being deployed. You can also use the Deployment Manager to add other necessary "feature files" to the deployment, such as database drivers for the target platform.



Release configuration - 32-bit Windows platform

Local Path	Local Name	Type	Platforms	Remote Path	Remote Name	Remote Status
✓ Release\Win32\	Marshaling.txt	File	[Win32]	.\	Marshaling.txt	Not Connected
✓ Release\Win32\	Marshaling.exe	ProjectOutput	[Win32]	.\	Marshaling.exe	Not Connected
✓ Release\Win32\	Marshaling.rsm	DebugSymbols	[Win32]	.\	Marshaling.rsm	Not Connected
✓ $(BDS)\bin\	dbxint.dll	File	[Win32]	.\	dbxint.dll	Not Connected

Figure 21. Deployment Manager

The Deployment Manager is not needed for every project that is to be deployed to a remote system. In most of the cases when you create a project, everything works flawlessly when the project is deployed to the target platform without the Deployment Manager. But projects differ, and the Deployment Manager gives you control over these special cases.

With the Deployment Manager, you can:

- Enable or disable the files that are to be deployed to the target machine
- For all applications, add other files to your deployment list
- Add feature files from a generated list of ready-to-deploy files
  Important: For cross-platform database applications, you must use the Deployment Manager to enable the necessary database drivers for your specific target machine. You enable the database drivers using the Add Feature Files dialog box. When you actually run, debug, or deploy your application, the IDE copies the files currently enabled in the deployment list (including the database drivers and any other feature files you have added) to the target platform.

PLATFORM ASSISTANT

The Platform Assistant (PA) enables developers to deploy and debug applications on remote Windows and Mac machines. As already mentioned, PA is integrated with the Deployment Manager interface, allowing you to deploy the applications and external references, such as DLLs.

Imagine a scenario where you need to deploy or debug an application at your customer's site, but you are hundreds of miles away and don't have any idea about the problem reported by your customer. Remote debugging through PA allows you to search and find the problem directly on the customer's machine. After you find and fix the problem in your code, when you need to deploy the new version of your application, the integration between PA and Deployment Manager will tell you which files changes, and you just deploy that set of files and not the whole application.

FIREMONKEY

FireMonkey is a brand new application platform for building stunning visual applications for Windows, Mac and iOS. It was introduced in XE2 release and supports Delphi and C++Builder. FireMonkey uses native graphic libraries to abstract from the underlying OS, offers powerful HD (High Definition) and 3D graphics, flexible styles, support for rich GUIs, and a new data binding model for business applications.

FireMonkey brings the most fantastic effects, animations and interactive applications with high performance. In addition, FireMonkey is the first native platform for business applications development. It's based on CPU and GPU, enabling you to easily develop applications that are visually stunning and highly connected to multiple platforms and devices.

FireMonkey abstracts the operating system API to render the UI. Different technologies are being used for graphic processing, depending on the operating system:

- Windows
    - For HD applications it uses Direct2D (or GDI+ in case Direct2D is not available)
    - For 3D applications it uses Direct3D
- Mac
    - For HD applications it uses Quartz
    - For 3D applications it uses OpenGL
- For iOS applications it uses OpenGL

FireMonkey uses the GPU for 3D, filters/effects and 2D drawing. The GDI+ software fallback is available only for 2D. This means the application needs to run in a computer with decent video card, which most of computers today have. The first computer to use a GPU was the Commodore Amiga in 1985.

Since FireMonkey is only responsible for the UI, other functionality like access to the file system, calling Web Services, etc, will go through the RTL, which in Delphi XE2 became cross-platform. For database access you have the dbExpress components and drivers available as cross-platform, and the DataSnap architecture as well.

In addition to all the graphics, hundreds of components such as Edit, ListBox, ComboBox, Grid and others are available. They have many similarities with VCL components including almost the same component, properties and event names. The behavior is very similar as well, but different code, units and they are cross-platform.

FireMonkey components are containers. That means that any component can be embedded in any other component. That's how all the actual FireMonkey components work today. In order to illustrate that, let's use the VCL button (TButton) as an example. TButton can receive an image through the image property. In FireMonkey the TButton component doesn't have an Image property. Instead, you drag and drop an image inside the button and align it. From there you can create a style for this button and reuse it through your application.

The Button component is composed of 9 components. When together, they look and behave like a Button, let's see how it works.

- To start, a TLayout component, helps to organize all of the controls
- Three TRectangle are responsible for the button look and feel, like: border, background and foreground color
- A Label represents the Button text
- Finally, it uses two animation components and another two for effects. Why effects and animations? Animations will happen when you move the mouse over/out the button. Effects will happen when button is pressed or has a focus. The glow effect is responsible for that.

FireMonkey uses the concept of Styles to customize components and user interface. Imagine a web site that uses HTML and CSS. In order to customize the web site, you just change the CSS style files, right? In FireMonkey we customize the components and store the changes in a StylesBook. Therefore the StylesBook is yours CSS for desktop applications and any StyleBook changes will be reflected throughout the application.

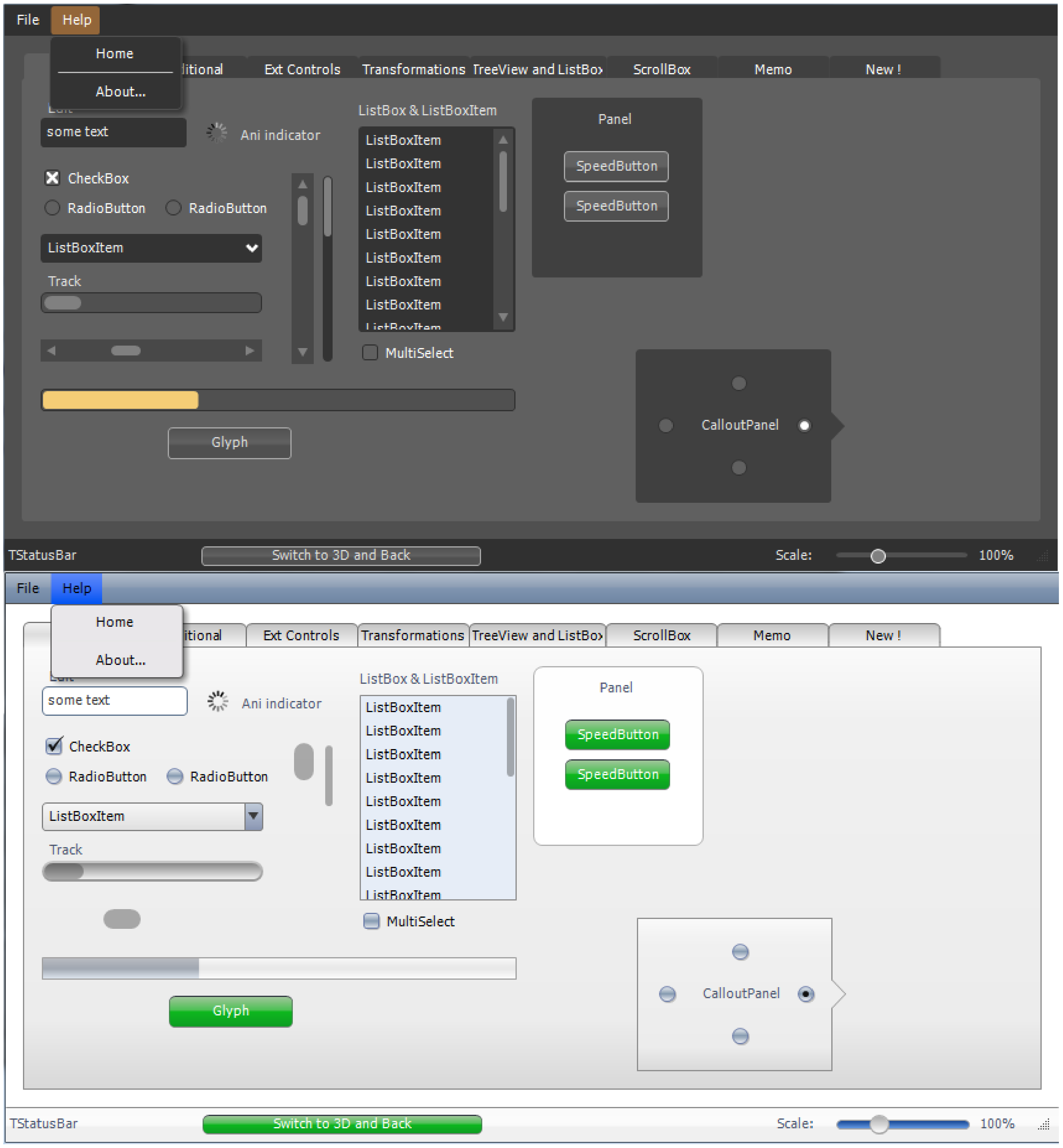


Figure 22. FireMonkey application using two different styles

Style is very powerful and goes beyond the change of interface, you'll find out when you start using FireMonkey, you will often create a style instead of create a new component.

FireMonkey goes beyond of delivery a standard user interface to multiple platforms, but delivery a rich user experience on the platforms supported, styles is one way to do that, but we also can use animations and image effect to create a much better user interface.

Animations are components that can modify property values over time. They can be started automatically or manually, both with an optional delay. After the animation has run its course over the defined time period, it can stop, start over, or do the same but in reverse.

Effects are components as well and modified Pixels, either individually or in concert with others, to achieve various visual effects. These effects are not limited to bitmap image data; they can be applied to the pixels of any 2D control in the user interface. Image effects can be both triggered and animated.

As Animations and effects are simply FireMonkey components, new effects can be created just like any other component, by using the New Component wizard and using TEffect or TAnimation (or one of the existing effects) as the ancestor.

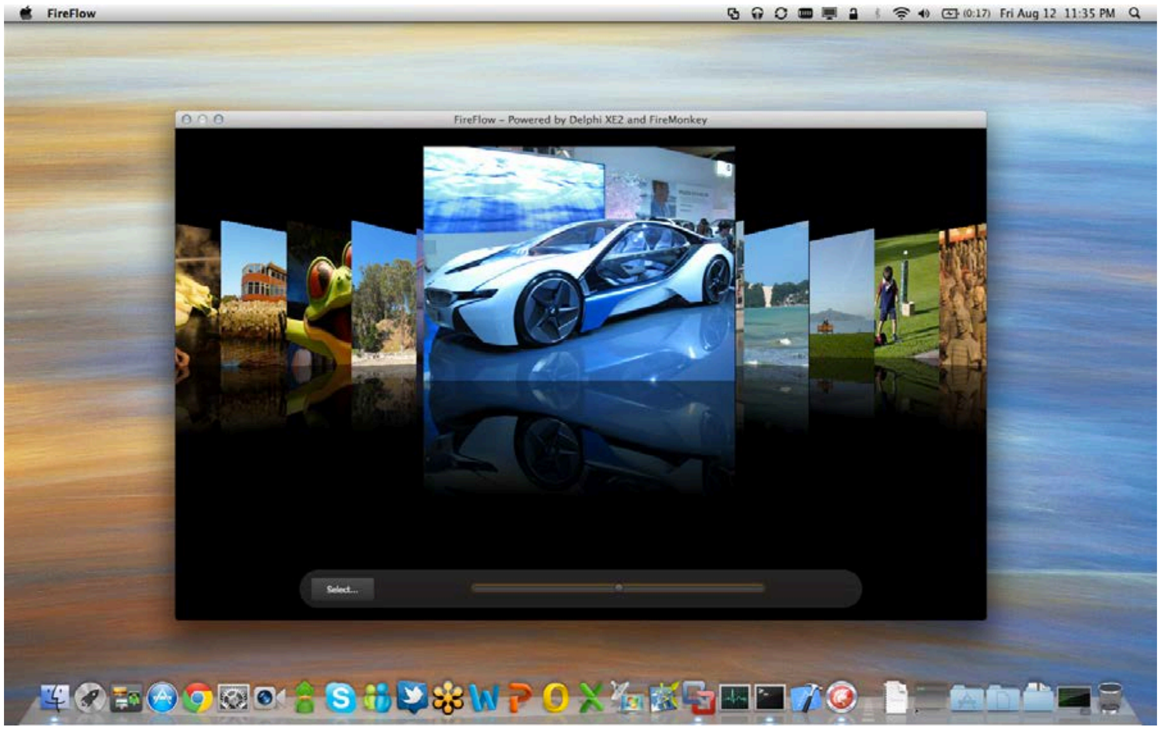


Figure 23. FireMonkey Application using animations and effects on Mac OS X

In addition to Windows and Mac support, you can develop native iOS application with FireMonkey and Delphi - creating a new iOS HD or 3D projects.

The way you develop your FireMonkey iOS application is almost the same as creating a FireMonkey application for Windows or Mac, beyond the platform differences. The Tool Palette will show only the components compatible with iOS. Components like menus, dialogs and few others are not available since they don't exist for iOS. Considering database access, dbExpress components are currently available for iOS as well, but you can have access to databases like SQLite, the standard for iOS using FPC components or using the iPhoneAll unit to access the iOS SDK directly.

You can run and debug applications on Windows. At some point, you will need to test the application in a iOS simulator or real device. The process to compile your application for iOS happens through the Apple Xcode IDE. Delphi has a command line tool (dpr2xcode.exe) that exports your project to the Xcode project file format. After that you need to open the Xcode project in the Xcode IDE and build and run it. Even in the Xcode IDE you will be able to debug your app.

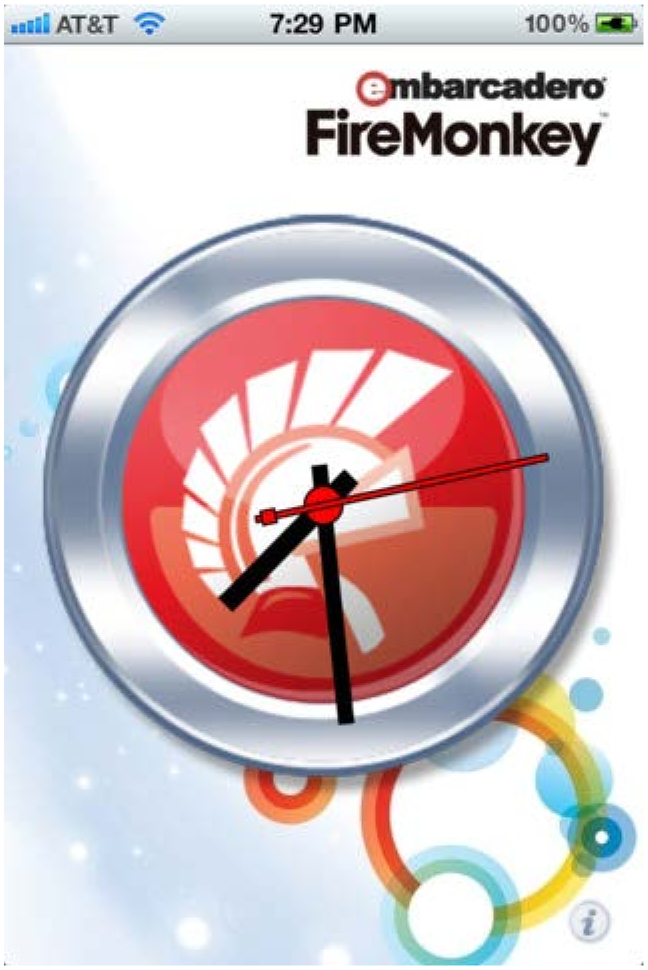


Figure 24. Analog Clock app for iPhone build with Delphi

The compilation of iOS FireMonkey application in Xcode is done using the Free Pascal compiler and uses the FPC RTL as well. In the future, we expect to have our compiler and RTL ready for iOS.

You can obtain more technical information and learn more about FireMonkey at http://www.embarcadero.com/products/firemonkey and http://www.embarcadero.com/rad-in-action/firemonkey.

LIVEBINDING

LiveBinding is a data-binding feature supported by both the VCL and FireMonkey in Delphi XE2. LiveBinding is expression-based, which means it uses expressions to bind objects to each other, by means of their properties.

LiveBinding is based on relational expressions, called binding expressions, that can be either unidirectional or bidirectional. LiveBindings is also about control objects and source objects. By means of binding expressions, any source object can be bound to itself (it becomes both source and control object) or to any other control object, simply by defining a binding expression involving one or more properties of the objects you want to bind together. For example, you can bind a TEdit control to a TLabel so that, when the text changes in the edit box, the caption of the label is automatically adjusted to the value evaluated by your binding expression. Another example is binding a tracker control to a progress bar so that the progress rises or lowers as you move the track bar.

In the same manner, you can connect to databases, alter one or more properties of different objects, and so on. Because LiveBindings propagate, you can even alter properties of objects that are connected to other objects that are bound to a control object.

FireMonkey doesn't have data-aware components link in the VCL. What we have are an Edit, Grid and other components which can be connected to dataset through the LiveBinding technology.

The LiveBinding Engine does a have use of RTTI, this is only possible because of the enhancements made in the RTTI since Delphi 2009 and you can use all of that to improve your application.

The following links show some great examples of how to use LiveBindings:

- Fill a TListBox from a TClientDataSet at design time - http://blogs.embarcadero.com/jimtierney/2011/09/30/31559
- Code to create TBindLink and fill a Listbox - http://blogs.embarcadero.com/jimtierney/2011/10/03/31601
- FillList using enumerable objects - http://blogs.embarcadero.com/jimtierney/2011/10/14/31608

If you are using or planning to use an ORM framework, LiveBinding and the new RTTI improvements provide everything you need to start using ORM in a more effective and productivity way.

WHAT'S NEW IN THE VCL AND RTL

Since Delphi 7, the VCL and RTL have been continuously enhanced. Besides complete support to Windows XP, 2000, Vista, Windows 7 and Unicode, VCL Styles, new components have been added and existing components have been improved by the addition of new functionalities.

All these add up to better component usability, easy creation of rich interfaces with VCL Styles and the ability to use Windows Vista and Windows 7 new functionalities, as well the full support for touch and gesture. In this section, we focus on what's new in existing components and the changes in RTL classes.

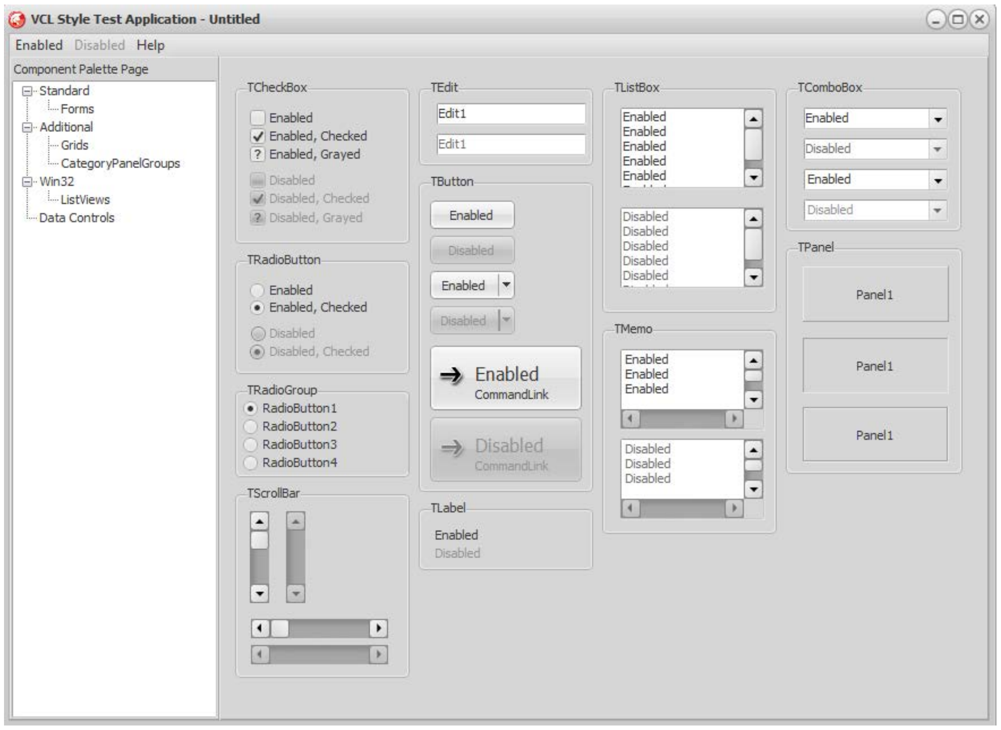
VCL STYLES

VCL Styles is one of the most impressive new feature in Delphi XE2. VCL Styles allow developers to change the entire application appearance using Style files and it takes only one line of code.

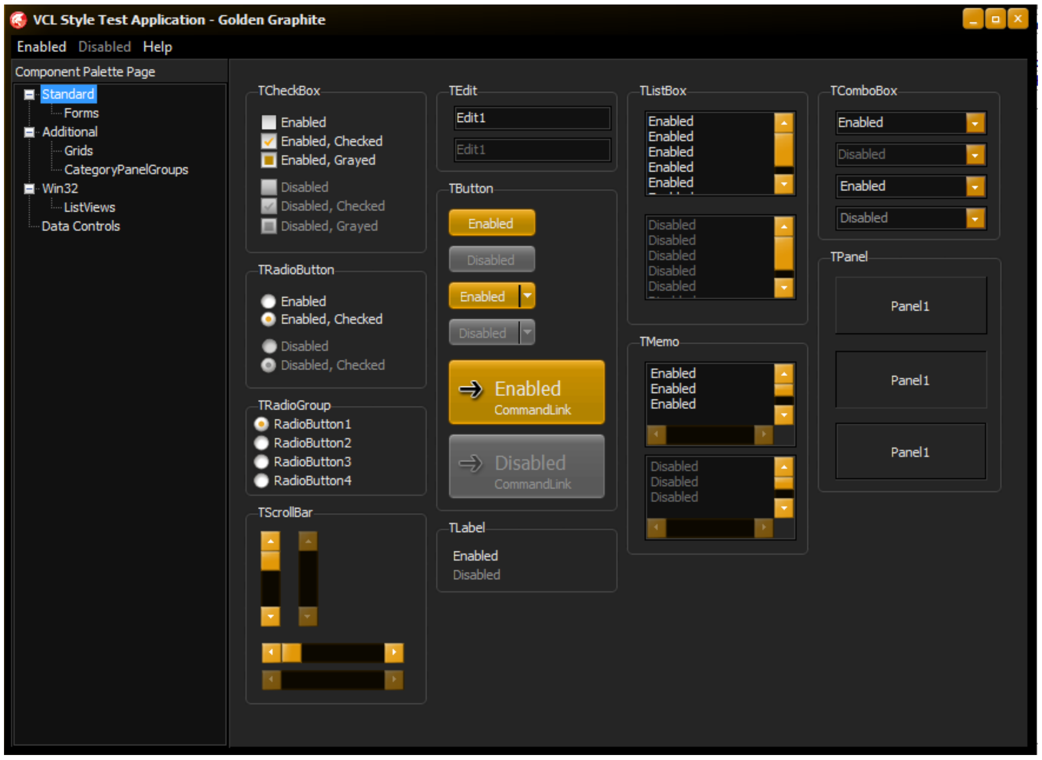
A style is a set of graphical details that define the look and feel of a VCL application. It corresponds to a theme in Windows.

VCL controls are made up of parts and states. A VCL style consists of a set of values for each of these parts and states. For instance, a scroll bar has the following parts: frame, slider, and the two side buttons for each direction. The side buttons, for example, have the following states: pressed, disabled, hot, normal. A style permits you to change the appearance of every part and state of a control.

The following images show the use of two different Styles available in Delphi XE2.



Delphi XE2 includes 5 default styles. You can find them in C:\Users\Public\Documents\RAD Studio\Styles\Embarcadero\. To set one of these styles for your application, set it in Project options > Application > Appearance.



You can create your own styles, or modify the existing ones by using the VCL Styles Designer, available in the Tools menu. The VCL Style Designer is a standalone tool that can be distributed, so your designer can use that to create a new style. The result is a new style file that you will load in your application.

VCL DIRECT2D AND WINDOWS 7

Microsoft Windows 7 introduces Direct2D a hardware-accelerated, immediate-mode, 2-D graphics API that provides high performance and high-quality rendering for 2-D geometry, bitmaps, and text. The Direct2D API is designed to interoperate well with GDI, GDI+, and Direct3D. Direct2D forwards all drawing operations to the GPU (Graphics Processing Unit) instead of the CPU and that means more resource available to your application. This topic discusses how to take advantage of the new screen Direct2D Delphi in your application.

Direct2D is only supported on Windows 7, along with Delphi 2010 (or later) is 100% compatible with Windows 7 and it uses several features of the new Microsoft operating system, Direct2D is one. To make sure you're developing your application using Direct2D, you must include the following units in your application:

- Direct2D, which exposes the wrapper classes as TDirect2DCanvas VCL.
- D2D1, which contains the header translations to Microsoft Direct2D API

The follow example shows how to override the Paint method of the form, using the class TDirect2Canvas.

```
procedure T2D2Form.FormPaint(Sender: TObject);
var
  LCanvas: TDirect2DCanvas;
begin
  LCanvas := TDirect2DCanvas.Create(Canvas, ClientRect);
  LCanvas.BeginDraw;

  try
    { Drawing goes here }
    LCanvas.Brush.Color := clRed;
    LCanvas.Pen.Color := clBlue;
    LCanvas.Rectangle(100, 100, 200, 200);
  finally
    LCanvas.EndDraw;
    LCanvas.Free;
  end;
end;
```

Direct2D will be the natural replacement of Canvas, the difference in graphics quality is very high, as shown in Figure 21.

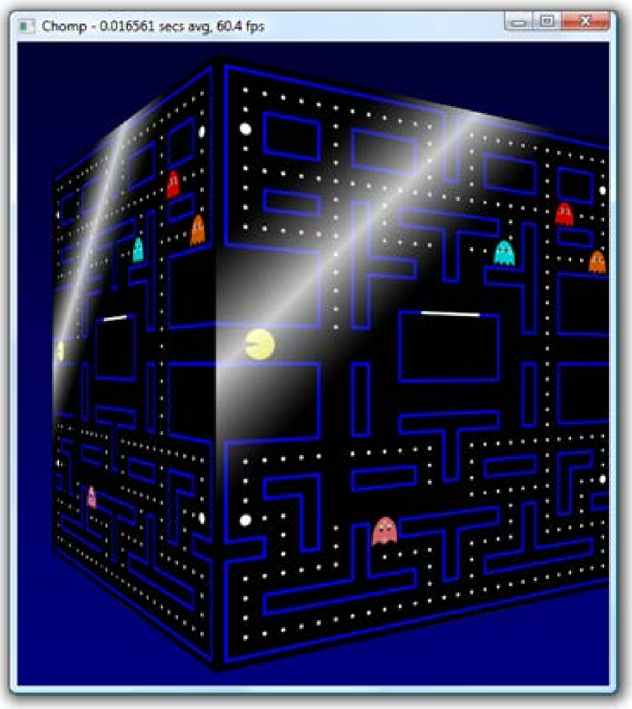


Figure 25. Graphic using Direct2D

TOUCH AND GESTURES

The application development market is not the same after the year of 2009; today the emergence of applications for cameras, phones, GPS-based Touch and Gesture is very large and growing rapidly. Windows 7 introduce the support for Multi-Touch where users can interact with the application by two or more taps on the screen.

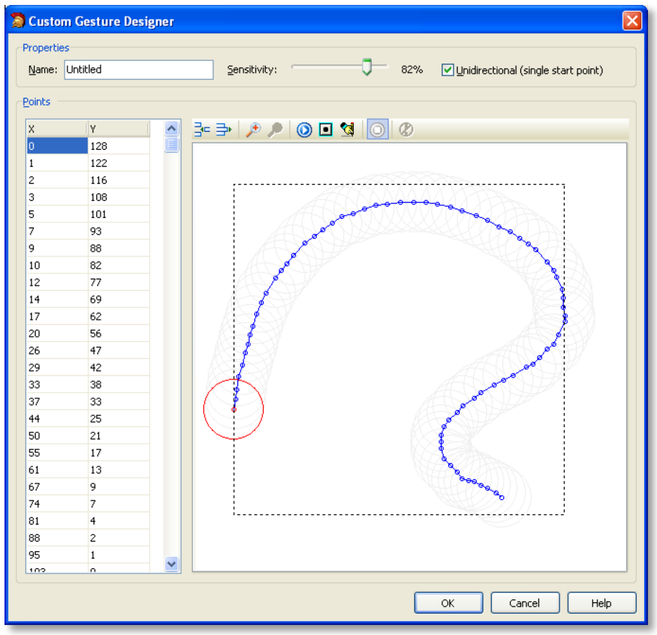
Before start developing touch applications is important to understand the 3 kinds of touch:

- Basic Touch → already used in the market, fingers replaces the right mouse button, e.g. ATMs, point of sale, etc.
- Multi-touch → Application elements with extensive interactions via touch, example: movie Minority Report and iPhone. Only supported by Windows 7 and newer hardware.
- Gestures → Act with the movement of a finger or mouse, firing an event. May be intuitive, allowing customization, more inputs and is supported by almost all systems touch.

Delphi has an architecture that allows users to plug other gesture engine and it will run on all supported versions of Windows, not only Windows 7. It's also backwards compatible with the available hardware and enables users to emulate mouse movements or touch. The VCL brings more than 30 standard pre-defined gestures, as well the gesture editor for writing, testing and reproduction of recorded GESTURE. In addition, Delphi includes a component to simulate a virtual keyboard, the TTouchKeyboard.

All visual components have a Touch property, which has a sub-property, named Gesture Manager, which will be connected to the respective component. The Gesture Manager will manage all the gestures made in the application, it triggers an action for each gesture, and all actions are managed by the Action Manager, so even if you use Action Manager simply connect their Actions to the Gesture Manager.

Figure 22 shows the Gesture Designer that allows recording, playback and testing of the generated motion, the movement associated with the VCL component will trigger an action by the Action Manager, the movement has to be close as possible, and you can adjust sensitivity for each gesture.



Name: Untitled
Sensitivity: 82%
[✓] Unidirectional (single start point)

X	Y
0	128
1	122
2	116
3	108
5	101
7	93
9	88
10	82
12	77
14	69
17	62
20	56
26	47
29	42
33	38
37	33
44	25
50	21
55	17
61	13
67	9
74	7
81	4
88	2
95	1
103	0

Figure 26. The Gesture Designer, allow developers to create and test gesture

RIBBON CONTROLS

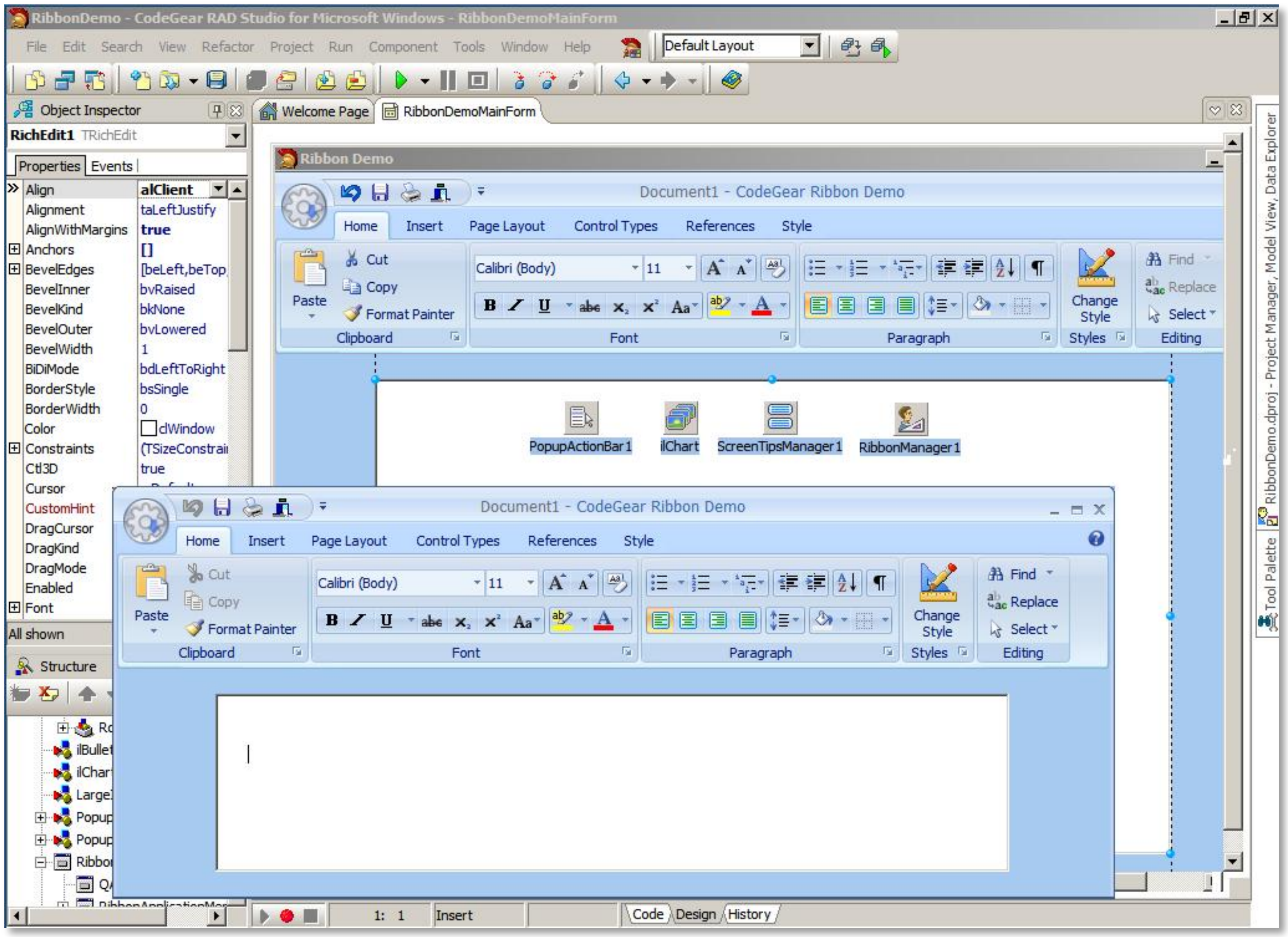
Most of you already know about the new Ribbon interface (used in Office 2007). Such interfaces are designed to facilitate user access to your application's menu options.

The VCL now comes with Ribbon Controls, a group of components that allows you to create Ribbon-style Delphi interfaces.

Ribbon Controls integrate with the Action Manager, which means that applications with Actions and the traditional menus can be easily migrated to Ribbon.

The architecture behind Ribbon Controls is very simple. From a Ribbon control you are able to add a Tab (which contains groups). Each of these groups contains buttons with customized appearance. Additionally, the Ribbon control includes the *Quick Access Toolbar* and the *Application Menu*.

Figure 23 provides an example application with Ribbon Controls.


RibbonDemo - CodeGear RAD Studio for Microsoft Windows - RibbonDemoMainForm
File Edit Search View Refactor Project Run Component Tools Window Help
Default Layout
Object Inspector
Welcome Page
RibbonDemoMainForm
Ribbon Demo
Document1 - CodeGear Ribbon Demo
Home Insert Page Layout Control Types References Style
Paste
Cut
Copy
Format Painter
Clipboard
Calibri (Body)
11
Font
Paragraph
Change Style
Styles
Find
Replace
Select
Editing
PopupActionBar1
ilChart
ScreenTipsManager1
RibbonManager1
Structure
Tool Palette
RibbonDemo.dproj - Project Manager, Model View, Data Explorer
1: 1
Insert
Code Design History

RichEdit1 TRichEdit

Properties | Events

Align	alClient
Alignment	taLeftJustify
AlignWithMargins	true
Anchors	[]
BevelEdges	[beLeft,beTop
BevelInner	bvRaised
BevelKind	bkNone
BevelOuter	bvLowered
BevelWidth	1
BiDiMode	bdLeftToRight
BorderStyle	bsSingle
BorderWidth	0
Color	clWindow
Constraints	(TSizeConstrai
Ctl3D	true
Cursor	
CustomHint	
DragCursor	
DragKind	
DragMode	
Enabled	
Font	

All shown

Figure 27. Ribbon Controls

WINDOWS VISTA AND WINDOWS 7 SUPPORT

Applications compiled with Delphi 2007 (or later) are 100% compatible with Windows Vista and Windows 7. The VCL has been updated to support the new characteristics of this OS, while new components were also added: TFileOpenDialog, TFileSaveDialog and TTaskDialog.

New classes have also been created; for instance:

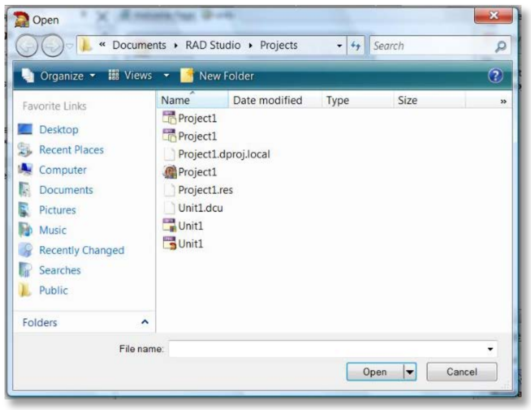
- TCustomFileDialog
- TCustomFileOpenDialog
- TCustomFileSaveDialog
- TCustomTaskDialog
- TFavoriteLinkItem
- TFavoriteLinkItems
- TFavoriteLinkItemsEnumerator
- TFileTypeItem
- TFileTypeItems
- TTaskDialogBaseButtonItem
- TTaskDialogButtonItem
- TTaskDialogButtons
- TTaskDialogButtonsEnumerator
- TTaskDialogProgressBar
- TTaskDialogRadioButtonItem

Dialog box components are now displayed in a Vista-like fashion. You are now probably wondering what happens to applications when you run them on Windows XP. There's no reason to be concerned about it. VCL recognizes the OS, using its specific resources and the appropriate interface.

The TaskMessageDlg function was designed to support Windows Vista. It has the same functionality seen in MessageDlg, with additional parameters that support Windows Vista's characteristics. When you run your application on Windows XP, MessageDlg is automatically executed. VCL is there to ensure it.

The UseLatestCommonDialogs global variable defines that all dialog components (TOpenDialog, TSaveDialog, TOpenPictureDialog and TSavePictureDialog) should follow Windows Vista's design whenever it receives a TRUE statement.

For example, this is how Open and Save Dialogs look in Windows Vista and Windows 7:



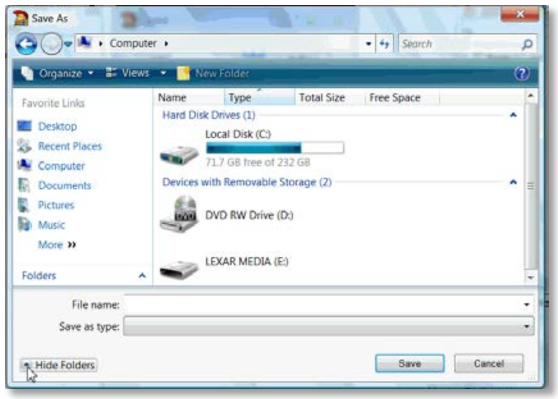


Figure 28. Open Dialog in Windows 7

Figure 29. Save Dialog in Windows 7

The units below were enhanced to support Window's new APIs.

- UxThemes – new
- DwnApi – new
- ActiveX – updated
- Windows – updated
- Messages – updated
- CommCtrl – updated
- ShlObj - updated

NEW AND ENHANCED VCL COMPONENTS

TACTIONMANAGER

New properties - DisabledImages, LargeImages and LargeDisabledImages - that allow you to define large and disabled images, based on the TImageList component.

PNG SUPPORT

The Image component supports the PNG format natively.

TBITMAP

Support to 32-bit alpha bitmaps, along with the addition of the AlphaFormat property.

TBUTTONGROUP

This component allows you to group many different buttons in a panel.

TBUTTONEDIT

The new TButtonEdit component allows you to add images within the Edit field. Images can be placed either at the right or left side. You can also use events to control image clicking – by means of the onLeftClick and onRightClick events.

TLinkLabel

LinkLabel allows you to add HTML tags that affect the appearance of the component.

TPopupActionBar

Now supports ActionBar styles.

THeaderControl and THeaderSection

New checkbox support.

TButton

Windows Vista now has two new button styles, both supported by Delphi with its TButton class.

CommandLink has a different, friendlier design. You can use it to add a more detailed description of the button functionality.

SplitButton opens a list of options when clicked. This list is presented as a PopMenu. You can also assign images to the items.

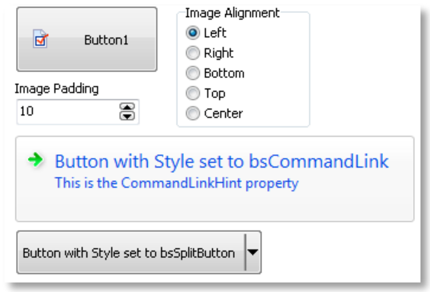
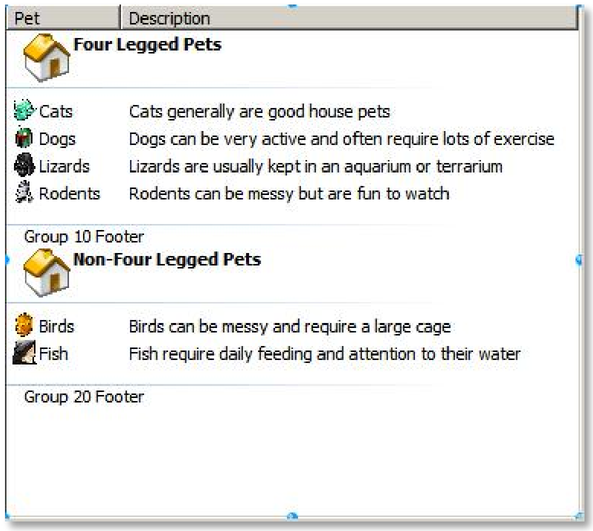

Button1
Image Padding
10
Image Alignment
Left
Right
Bottom
Top
Center
Button with Style set to bsCommandLink
This is the CommandLinkHint property
Button with Style set to bsSplitButton

Figure 30 New Button Styles

TLISTVIEW AND TTREEVIEW

TListView now allows you to define basic and advanced groups. The advanced group support enables deeper customization of groups (requires Vista), allowing you to define images for each of them.

TTreeView allows enabling/disabling nodes and images for expanded items.

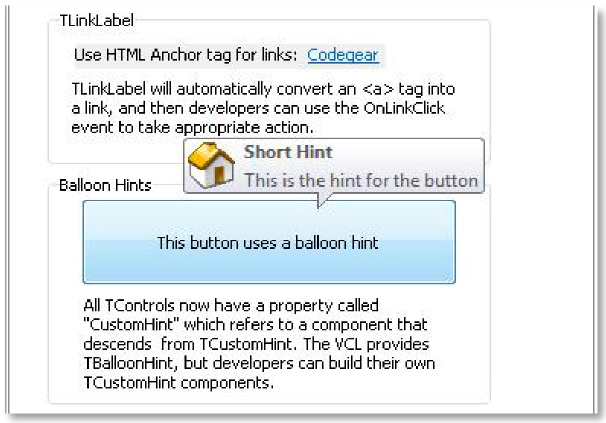


Pet	Description
Four Legged Pets	
Cats	Cats generally are good house pets
Dogs	Dogs can be very active and often require lots of exercise
Lizards	Lizards are usually kept in an aquarium or terrarium
Rodents	Rodents can be messy but are fun to watch
Group 10 Footer	
Non-Four Legged Pets	
Birds	Birds can be messy and require a large cage
Fish	Fish require daily feeding and attention to their water
Group 20 Footer	

Figure 31. TListView

TBALLOONHINTS

Hints now come in Windows Vista style and allow the addition of titles, descriptions and images, which will make your user notifications much friendlier.



TLinkLabel

Use HTML Anchor tag for links: Codegear

TLinkLabel will automatically convert an <a> tag into a link, and then developers can use the OnLinkClick event to take appropriate action.

Short Hint

This is the hint for the button

Balloon Hints

This button uses a balloon hint

All TControls now have a property called "CustomHint" which refers to a component that descends from TCustomHint. The VCL provides TBalloonHint, but developers can build their own TCustomHint components.

Figure 32. Balloon Hints

TCategoryPanelGroup

This new component is very useful. It works like an Outlook bar, to which you can add many different panels. Each of these panels can contain any VCL component. You can define a title, an image, the alignment and an icon for each of the panels, being able to expand and collapse them.

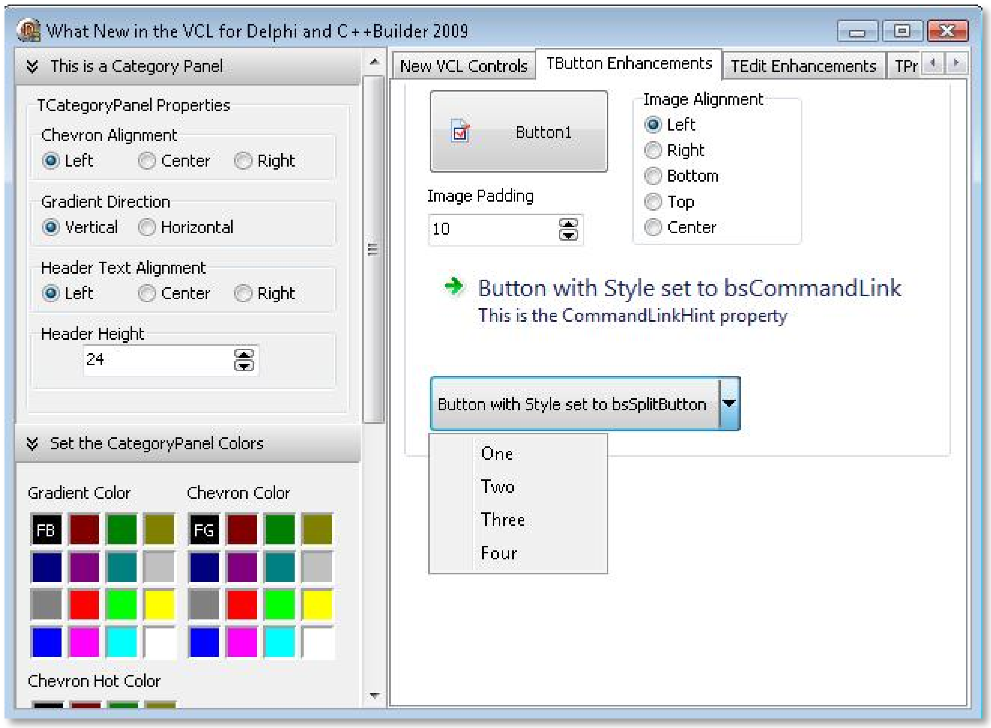


Figure 33. Panel Group

New Panels

The traditional TPanel is a visual container for other components. Within TPanel you are able to accommodate visual controls wherever you want. In other words, it works with absolute positioning (the Top and Left coordinates of the control refer to the panel's upper left corner).

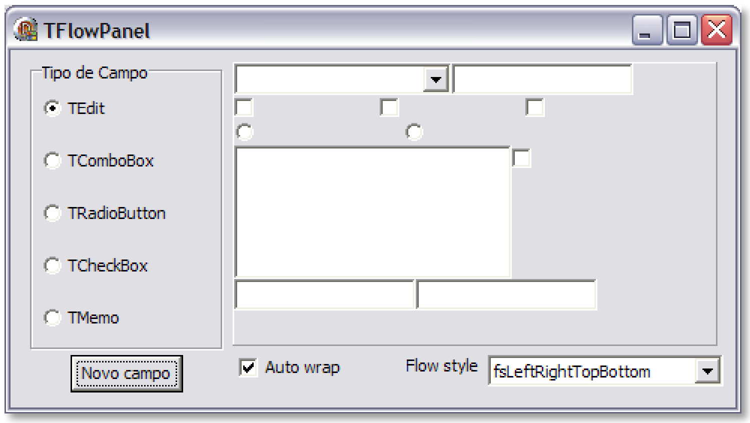
Inspired by similar Java concepts (namely, the Layout Manager, which defines how controls are distributed within the container) we can say we now have three kinds of layout managers:

- TPanel: absolute type, or XY. Components are placed in fixed, precise positions.
- TFlowPanel: components are placed in a sequence, according to a given order (similar to an HTML page, neither using tables nor CSS stylesheets).
  - The flow is determined by the FlowStyle property, which accepts the options you see below. In order to understand the naming convention, keep in mind that components are accommodated according to the direction defined by the first pair (e.g., LeftRight). When

there's no space left in the panel, the direction is redefined by the second pair (e.g., TopBottom):

- fsLeftRightTopBottom: left to right, top down (default)
- fsRightLeftTopBottom: right to left, top down
- fsLeftRightBottomTop: left to right, bottom up
- fsRightLeftBottomTop: right to left, bottom up
- fsTopBottomLeftRight: top down, left to right
- fsBottomTopLeftRight: bottom up, left to right
- fsTopBottomRightLeft: top down, right to left
- fsBottomTopRightLeft: bottom up, right to left

- Another relevant property is AutoWrap. When True, indicates the flow will be "broken" towards the other direction in case the panel runs out of space. When False, components outside the boundaries of the panel will not be visible.
- You can use this panel to automatically generate forms, with fields being dynamically defined either in a database or a file. This way you do not need to be concerned about positioning each of the fields.



Auto wrap [✓]
Flow style: fsLeftRightTopBottom

Figure 34. Form Auto Generation

- TGridPanel : the panel is partitioned by lines and columns, with each of the resulting cells holding a component (similar to the use of HTML tables).

  - Components are arranged according to the order of the lines (top down) and, within each line, column-wise (left to right).
  - The number of lines is determined by the RowCollection property, which can contain various objects of the TRowItem class. Each item has two properties:

    - SizeStyle: determines the standard by which line height is specified in the Value property:

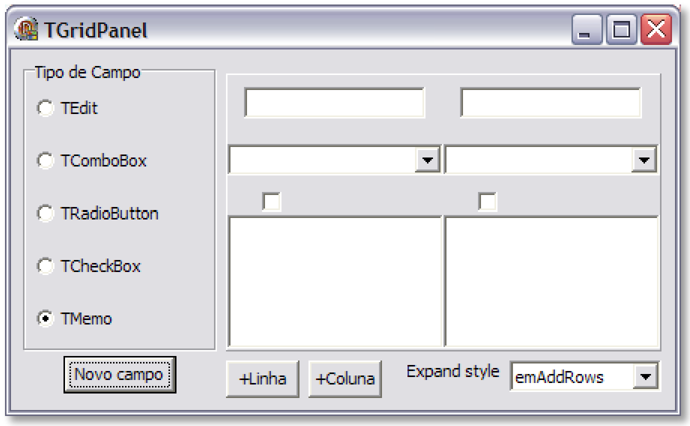
      - ssAbsolute: number of pixels
      - ssAuto: the number is disregarded, with line height being automatically calculated

- ssPercent: percentage in comparison to the panel height

- Value: a number expressing the height, according to the SizeStyle property.

- Similarly, the number of column is determined by the ColumnCollection property, which contains objects of the TColumnItem class. TColumnItem has the same properties TRowItem does, differing only in that they relate to the column width, not the rows.
- The ExpandStyle property determines an action for whenever someone tries to add a component into a panel that is out of free cells. It's possible values are:

  - emAddRows: a new line is added to the panel to accommodate the components
  - emAddColumns: new columns are added to accommodate the components
  - emFixedSize: an exception is raised when there's no more free space to accommodate new components.



Expand style: emAddRows

Figure 35. Expand Property Style

TCATEGORYBUTTONS

This component allows you to create buttons and group them into categories, similar to what's seen in an Outlook bar. It helps you refine the design of your applications.

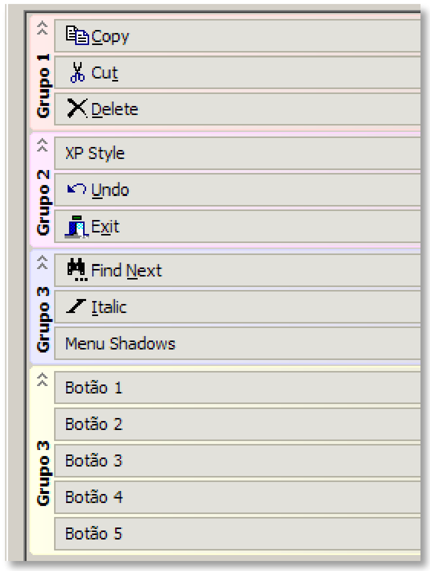


Figure 36. TCategoryButtons

TDRAWGRID, TSTRINGGRID, TDBGRID

Themed and gradient styles have been added to the grid components.

TTRAYICON

Think of those icons you see beside the taskbar clock, in the Windows tray. What if you could place your application right there too? It's now more than simple to do so. All you have to do is place a TTrayIcon component (Win32 tab) in the main form. Set just a few properties and you're done:

- Icon: Stores the icon that is displayed in the tray. You can use your application's icon or an icon that describes a status or situation. This icon can be changed at anytime.
- Icons: references a TImageList containing a bunch of bitmaps or icons which will be used in the animation.
- Animate: When True, keeps swapping the icons in the Icons list. The index of the icon that is being currently displayed can be either retrieved or changed using the IconIndex property.
- AnimateInterval: millisecond interval of the icon swapping process. The OnAnimate event is generated after each iteration, allowing you to define an action to be taken.
- BalloonTitle and BalloonHint: Title and text for the balloon, displayed by the ShowBalloonHint method. The balloon can be closed by a simple click (either on it or over the dialog's X). However, it goes away automatically after the interval predefined in the BalloonTimeout property (milliseconds).

- PopupMenu: it's common to associate a popup menu to the application's icon; enabling users to quickly access the most commonly used commands. All it takes for you to do so is reference the menu in this property. To access it, left-click the icon.

You can hide the main form (using methods Hide or Visible := False) without halting the application. In this case it's essential to provide the tray icon with a menu or event (OnMouseClick, for instance) to ensure the control is passed back to you right after.

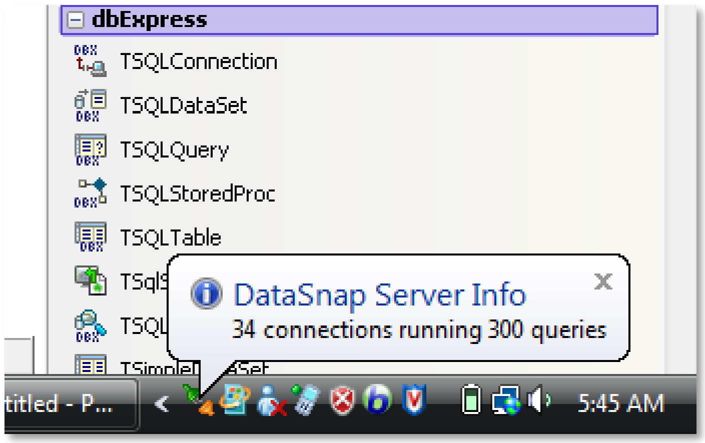


Figure 37. Tray Icon

VCL – MARGINS AND PADDING

Three seems to be a magic number these days… Besides three new visual components, the VCL has also been improved with three additional classes:

- TMargins
- TPadding
- TCustomTransparentControl

The TControl class now has an additional property (Margins, of TMargins class). TMargins is used in the Margins property of TControl and its descendants. TMargins helps define the relative position between components on a form, and between the edges of the form and the component. For example, when you set the left margin for a component to 10 pixels, the component will not come closer than 10 pixels to the edge of the container or to another component on the left edge. The number of pixels by which two components are separated is the sum of the pixels of both components.

The TWinControl class adds an extra property – Padding and Margins.

Padding adds space along the edge the control. Child controls that are aligned to the parent are positioned inside the control according to this spacing. Padding does not affect child controls which are not aligned to the parent control, nor does it affect the size of the ClientArea.

Padding is the opposite of Margins. Margins affects the positioning of the control itself inside the parent control, but Padding affects how all aligned child controls are positioned with respect to the parent control.

TRANSPARENT CONTROLS

The new TCustomTransparentControl class can be used for components that need to be there while pretending they're not. Uh oh. Well... think of it as the glass of a window or door. You know it's there, although you're unable to visually perceive it. To see the difference, you can do this test: create a new VCL application (File | New | VCL Forms Application – Delphi for Win32). Place two TButtons and two TLabels in it, see **Error! Reference source not found.**

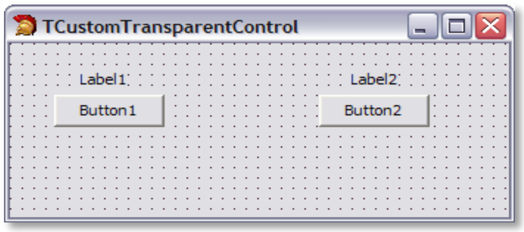
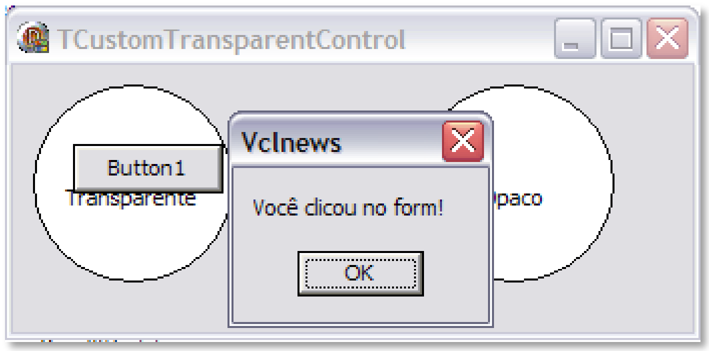


Figure 38. Transparent Controls

Set both buttons' Top property to 40; Button 1 and 2's Left properties to 30 and 210, respectively. When you're done, type the code as below. Note that the form implements the events OnCreate, OnDestroy and OnClick, with both buttons sharing the same OnClick event.

I've created a TCustomTransparentControl descendent – TTransparentControl – and a TCustomControl descendent - TOpaqueControl. They are dynamically created by the form's OnCreate event, being positioned right above the other existing controls. I've also added an OnClick event to our customized controls for you to observe their behavior. The result can be seen in Figure 35.



Você clicou no form!

Figure 39. Transparent Control Example

```
type
```

```
  TTransparentControl = class(TCustomTransparentControl)
  protected
    procedure Paint; override;
  end;

  TOpaqueControl = class(TCustomControl)
  protected
    procedure Paint; override;
  end;

  TfCustomTransparentControl = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    InvCon : TTransparentControl;
    VisCon : TOpaqueControl;
    procedure ControlClick(Sender: TObject);
  end;

var
  fCustomTransparentControl: TfCustomTransparentControl;

implementation

{$R *.dfm}

{ TTransparentControl }
procedure TTransparentControl.Paint;
const
  txt = 'Transparent';
begin
  Canvas.Ellipse(0,0,Width,Height);
  Canvas.TextOut((Width - Canvas.TextWidth(txt)) div 2, Height div 2, txt);
end;

{ TOpaqueControl }
procedure TOpaqueControl.Paint;
const
  txt = 'Opaque';
begin
  Canvas.Ellipse(0,0,Width,Height);
  Canvas.TextOut((Width - Canvas.TextWidth(txt)) div 2, Height div 2, txt);
end;

{ Form }
procedure TfCustomTransparentControl.FormCreate(Sender: TObject);
begin
  InvCon := TTransparentControl.Create(Self);
  InvCon.Parent := Self;
  InvCon.SetBounds(10,10,100,100);
  InvCon.OnClick := ControlClick;
  VisCon := TOpaqueControl.Create(Self);
  VisCon.Parent := Self;
  VisCon.SetBounds(200,10,100,100);
  VisCon.OnClick := ControlClick;
end;

procedure TfCustomTransparentControl.FormDestroy(Sender: TObject);
begin
  InvCon.Free;
  VisCon.Free;
end;

procedure TfCustomTransparentControl.Button1Click(Sender: TObject);
begin
  ShowMessage('You have clicked on ' + (Sender as TButton).Caption);
end;

procedure TfCustomTransparentControl.ControlClick(Sender: TObject);
begin
  ShowMessage('You have clicked on control ' + (Sender as
TControl).ClassName);
end;

procedure TfCustomTransparentControl.FormClick(Sender: TObject);
begin
  ShowMessage('Form clicked!');
end;
```

The custom message CM_INPUTLANGCHANGE was added to the VCL components for notification when the operating system language has changed, so you can reflect changes in language in its application without restarting the system.

Icons can now be associated with bitmaps using TIcon.AssignTo, moreover the TImage component supports TIFF.

New Month Calendar drop down box to define dates of properties in the Object Inspector define as Date data type, in addition to the new Property Editor added to the Object Inspector, which allows users to use checkbox for Boolean properties.

Other new features was implemented in Delphi adding the capability to rename TCategoryButtons components, method CheckAll (cbUnchecked, True, True) for TCheckListBox, function PtInCircle added to unit "Types" (similar to "PtInRect"), ActiveLineNo property which returns the correct position of the cursor TRichEdit component and the new unit IOUtils.pas which implement 3 new statics classes, TDirectory, and TPath TFile, these classes expose many statics methods useful for interacting with I/O.

Although there are dozens of improvements in VCL not here mentioned, but you can get a complete list of these improvements through help system.

To conclude this topic, several VCL methods have become inline methods, and hence the performance of these methods when used improved, as result the VCL gain in speed.

INTELLIMOUSE SUPPORT

IntelliMouse is how we refer to support for mouse-wheel scrolling in your application. Support for this technology was first introduced to VCL in Delphi 2006. In order to use it, simply declare the IMOUSE unit in your application.

TOBJECT

The father of all components in Delphi has also been enhanced:

- New methods
    - class function UnitName : string
    - function Equals(Obj : TObject) : Boolean;virtual
    - function GetHashcode : Integer; virtual
    - function ToString ; String;virtual

- A few additional Overloads for the following methods:
    - MethodAddress
    - FieldAddress

- The type of return of the functions below has changed from ShortString to String in order to support Unicode
    - ClassName
    - MethodName

Other components - TPanel, TProgressBar, TTrayIcon, TScreen, and TRadioGroup – also provide a lot of improvements.

NEW MEMORY MANAGER AND NEW RTL FUNCTIONS

Many RTL functions have been updated to improve the application performance. FASTMM is the most relevant of such improvements. A new memory manager aimed at Win32 applications, FASTMM enables applications to have better performance by performing the compilation in Delphi 2006, and identifies memory leaks just declaring ReportMemoryLikeonShutdown := True in any part of your program, in general I recommend to add at the initialization section.

It can't be overemphasized that simply by compiling your applications in Delphi 2006 or later, you experience performance gains, while also being able to detect memory leakages.

SOAP 1.2 CLIENT SUPPORT

Delphi 2010 introduced the SOAP 1.2 client support through the THTTPRIO component. In Delphi XE THTTPRIO exposes new properties to allow the developer to select a Client Certificate at design-time.

REGULAR EXPRESSION

Delphi XE introduces RTL support for regular expressions (unit RegularExpressions). Regular expressions provide a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters.

The following example shows how to use regular expression to validate IP address.

```
program RegExpIP;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  RegularExpressions;

var
  ipRegExp : String;
begin
  try

    ipRegExp := '\b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.’ +
                ‘(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25’ +
                ‘[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]’ +
                ‘|2[0-4][0-9]|[01]?[0-9][0-9]?)\b';

    if TRegEx.IsMatch(paramstr(1), ipRegExp) then
      Writeln('Text DOES match the regular expression')
    else
      Writeln('Text DOES NOT match the regular expression');

  except
    on E: Exception do
      Writeln(E.ClassName, ': ', E.Message);
  end;
end.
```

Passing the parameter 200.100.2.21 (valid IP) the result will be:

```
Text DOES match the regular expression
```

Passing the parameter 200.100.2.263 (invalid IP) the result will be:

```
Text DOES NOT match the regular expression
```

OBJECT-ORIENTED FILE AND DIRECTORY I/O CLASSES

Delphi 2010 brings a new unit IOUtils contains three static classes: TDirectory, TPath and TFile. These classes expose a number of static methods useful for I/O tasks. Most methods are functionally and signature compatible with .NET classes System.IO.Directory, System.IO.Path and System.IO.File.

The follow code shows how to read all files from specific folder.

```
procedure TForm2.Button2Click(Sender: TObject);
var
  Path : string;
begin
  if not TDirectory.Exists(edtPath.Text) then
    Caption := 'Invalid Path'
  else
    Caption := edtPath.Text;

  ListBox1.Clear;

  for Path in TDirectory.GetFiles(edtPath.Text, ‘*.*’) do
    Listbox1.Items.Add(Format('Name = %s', [Path]));
end;
```

100% UNICODE

One of the greatest challenges faced by our R&D team was incorporating Unicode support throughout the Delphi language, VCL, RTL, and the IDE and the Delphi ecosystem of tool and component partners..

As part of the Unicode development process we had meetings with customers, authors, consultants and technology partners Working directly with our tool and component partners proved essential in allowing third-party components to be available to support the latest version of Delphi, as well as in keeping developers updated about how to work with Unicode.

Unicode is a standard that allows computers to consistently represent and handle text from any existing system of writing.

- *The Unicode Standard: Version 5.0. 5. ed. Addison-Wesley Professional, 2006. 1472 p*

Many character sets – like Chinese, Japanese, and Russian, along with others of Asian background – are represented by means of Unicode. The most commonly used encodings are UTF (Unicode Transform Format) and UCS (Universal Character Set). To learn more about Unicode, visit: http://en.wikipedia.org/wiki/Unicode.

The result of all this is a Delphi that is 100% Unicode. You’re now probably wondering if the migration is that easy. Definitely yes, the VCL and the compiler handle many things. In case you want more detailed information about Unicode techniques I recommend that you to read the white paper, "Delphi Unicode Migration for Mere Mortals", available at http://edn.embarcadero.com/article/40307.

One of the most relevant changes in this version is that String types are now based on UNICODE. Previously based on the ANSI standard, the AnsiString and WideString types still work the same way, except regarding their data size in bytes.

Unicode changes, in short:

- String maps UnicodeString, no longer AnsiString
- Char now maps WideChar (2 bytes, not 1 byte) being a UTF-16 character
- PChar maps PWideChar
- AnsiString maps the old String type

No changes were applied to:

- AnsiString
- WideString
- AnsiChar, PAnsiChar
- Short String contains AnsiChar elements
- Implicit conversions still work.

The user's active code page controls the mode (ANSI vs. Unicode), so that ANSI strings are still supported.

Operations that do not depend on character size:

- String concatenation
- Standard String functions. E.g., Length, Copy, Pos, and so on.
- Operators. E.g., <string> < comparison> <string>, CompareStr(), CompareText(), etc.
- FillChar (<struct or memory>)
- Windows API

Operations involving the character size (measured in bytes) may require a few changes. Nothing too complicated, but here's a tip: pay special attention to code in which you:

1. Assume Sizeof (Char) is 1.
2. Assume the size of a string equals the number of bytes in the string.
3. Handle String or PChars directly.
4. Saves or reads a string from/to a file.

Items 1 and 2 do not apply to Unicode, because in Unicode the Sizeof (Char) is 2 bytes and the size of a string is twice as big as the number of bytes. Besides, the code that reads and saves files must understand the right number of bytes to perform those operations, for a character is no longer represented as 1 byte.

As you can see, migrating is very easy. The benefit of having Unicode support is that of allowing Delphi developers to distribute their applications worldwide. Brazil is currently one of the most relevant software developers in the global market. Many Brazilian companies distribute their applications and/or interchange information with China, Japan, Russia and other countries where Unicode is crucial.

In 2007, the Russian government acquired 1 million Delphi licenses to be used in teaching primary and high school students to develop software with Delphi. Therefore, Unicode support is vital for that country.

NEW LANGUAGE FEATURES AND COMPILER RESOURCES

64-BIT COMPILER

Delphi XE2 brings full support for 64-bit platform, not just the compiler support, but for the entire RTL, VCL and FireMonkey.

Delphi XE2 introduced the development of 64-bit Windows applications on either a native Win32 or a Win64 development system using RTL, VCL and FireMonkey..

The VCL and the RTL have been modified to work with 64-bit applications in the same way they work with 32-bit applications. This means that if you are using only the VCL and RTL, you can expect to use the same source code for both Win64 and Win32 platforms.

For those who have their applications based on VCL, migration is basically done by recompiling the application. You may need to review and handle some issues (mostly related to pointer operations, NativeInt size, and Assembly code), for example:

- Use of sizeof (Integer) should be replaced by SizeOf (Pointer)
- Inline Assembly code, mixing of assembly statements with Pascal code is not supported in 64-bit applications
- Winapi
  - If you pass pointers to SendMessage/PostMessage/TControl.Perform, the wParam and lParam parameters should be type-casted to the WPARAM/LPARAM type and not to Integer/Longint.
- Check external references (DLL, ActiveX, etc..), the documentation has all information about that..

For more details about the migration to 64-bit and code samples, visit http://docwiki.embarcadero.com/RADStudio/en/Converting_32-bit_Delphi_Applications_to_64-bit_Windows.

Win64 application development in Delphi is by definition cross-platform development, because the IDE is a Win32 application. This means that when you run an application that has the target platform 64-bit Windows, you are essentially deploying the application to the Win64 platform. Thus, at run time your development system needs to either be 64-bit Windows or be connected to a Win64 system.

If you are creating 64-bit components, packages, or libraries, you need to have 32-bit design-time versions of these if you want to use these components, packages, and libraries in the IDE during application development. This requirement exists because the IDE is a 32-bit Windows program.

If your development PC is a 64-bit machine running 64-bit Windows, you can run, debug, and deploy on your development PC, just as you would debug a 32-bit Windows application, without using the Platform Assistant and a remote profile.

When compiling for 64-bit, all the DLLs and other external dependencies should be 64-bit. 32-bit and 64-bit cannot coexist in the same process. For example, if a 64-bit application tries to access a database through dbExpress you will need the database client 64-bit version. dbExpress in XE2 has a new parameter called vendorlib64 that points to the 64-bit database client DLL.

Now you can create a Windows Shell Extension for Windows 64-bit, for example, adding two new context menu items in Windows Explorer to allow users to upload files to Microsoft Azure and Amazon S3. The same code for this extension can be compiled to 32-bit Windows with no changes.

Unit Scope Names

Starting in Delphi XE2 VCL, FireMonkey and RTL units now use a dotted-prefix naming convention, such as System.Types and Vcl.Styles. If you have existing code that uses qualified identifiers (such as Types.IStream), code changes may be required in order to compile. The wizards and templates in RAD Studio automatically insert the properly unit-scoped unit names in uses and includes.

For example, the SysUtils unit is now part of the System unit scope, as follows:

```
System.SysUtils
```

and the Controls unit is now part of the Vcl unit scope:

```
Vcl.Controls
```

Unit scope names will help to:

- Classify units into basic groups such as Vcl, System, Fmx, and so forth (unit scopes are classified in Unit Scopes).
- Ensure compatibility of the code that you write using the IDE.
- Differentiate members whose names are ambiguous (that is, ensure correct name resolution when a member's name matches the name of a member of another unit).

ENHANCED RTTI

The RTTI is largely responsible for providing us with information about our objects, thereby allowing the application and interaction of objects. One great example of the use of RTTI is the Delphi IDE itself, where we use the Object Inspector, code editor, modeling and other resources.

The evolution and development in other languages have changed the way we do programming. Java and. NET applications are good examples in this direction, where the language is now offering a high level of dynamic interaction. With the enhanced RTTI support added in Delphi 2010 and evolved in every following release, Delphi for Win32 now has the same power of reflection that. NET and Java have. The new RTTI (RTTI.pas) is fully object oriented and allows the creation and interaction with the applications in a much more dynamic way.

Visiting the RAD Studio wiki and you can learn all of the great new RTTI features - http://docwiki.embarcadero.com/RADStudio/en/Working_with_RTTI_Index

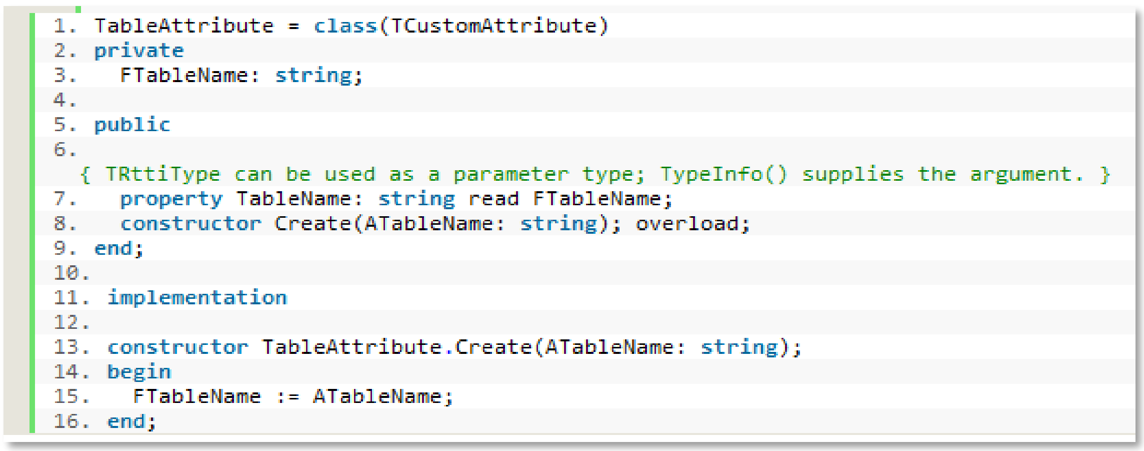
ATTRIBUTES

How many times have you heard your fellow users of Java and. NET comment on attributes, I believe that many, actually attributes is a very interesting feature, often used in frameworks such as Hibernate, .NET and Java.

Attributes let you define classes and features for their respective elements. There are several examples that show the usefulness of the attributes. The best example for attributes is to create framework object relational (O/R Mapping).

Attributes are defined through mandatory classes, they inherit from TCustomAttribute.

Example: TableAttribute – a class that will be used to map the application classes to the database, the TableName property defines the name of the table.



```
TableAttribute = class(TCustomAttribute)
private
  FTableName: string;

public

 { TRttiType can be used as a parameter type; TypeInfo() supplies the argument. }
  property TableName: string read FTableName;
  constructor Create(ATableName: string); overload;
end;

implementation

constructor TableAttribute.Create(ATableName: string);
begin
  FTableName := ATableName;
end;
```

Set the attribute mapping table, and use it in their classes.

Example: A class is mapping TClient to the CUSTOMER (CLIENTE) table.

```
[Table('CLIENTE')]
TCliente = class
public
  { public declarations }
  property Nome : String read FNome write FNome;
  property Endereco : String read FEndereco write FEndereco;
end;
```

In this example we use attributes to map only the class, but we can also map methods, variables, properties, method parameters, etc.

Attributes added in Delphi 2010 takes the language to another level and undoubtedly enables us to provide great improvement in the future, as well as allow you to implement several new features in your applications TODAY.

Exit Function

The Exit function can take a parameter specifying a result. The parameter of the System.Exit function must be of the same type as the result.

```
function doValidate( I : Integer) : boolean;
begin
  if I = 0 then
    exit(False)
  else
  begin
    …..
    // result something
  end;
end;
```

Inline Directive

The Delphi compiler allows functions and procedures to be tagged with the inline directive to improve performance. If the function or procedure meets certain criteria, the compiler will insert code directly, rather than generating a call. Inlining is a performance optimization that can result in faster code, but at the expense of space. Inlining always causes the compiler to produce a larger binary file. The inline directive is used in function and procedure declarations and definitions, like other directives.

```
program InlineDemo;
{$APPTYPE CONSOLE}

uses
  MMSystem;
  function Max(const X,Y,Z: Integer): Integer;inline
  begin
    if X > Y then
      if X > Z then Result := X
               else Result := Z
    else
      if Y > Z then Result := Y
               else Result := Z
  end;

const
  Times = 10000000; // 10 million

var
  A,B,C,D: Integer;
  Start: LongInt;
  i: Integer;

begin
  Random; // 0
  A := Random(4242);
  B := Random(4242);
  C := Random(4242);
  Start := TimeGetTime;
  for i:=1 to Times do
    D := Max(A,B,C);
  Start := TimeGetTime-Start;
  writeln(A,', ',B,', ',C,': ',D);
  writeln('Calling Max ',Times,' times took ',Start,' milliseconds.');
  readln

end.
```

When the above code is executed, the Max method is called 10 million times. The numbers below vary depending on your machine. Using a Pentium M 1.8GHz with 2GB RAM we've obtained the following results:

With inline	Without inline
25 milliseconds	68 milliseconds

The inline directive is a suggestion to the compiler. There is no guarantee the compiler will inline a particular routine, as there are a number of circumstances where inlining cannot be done. The following list shows the conditions under which inlining does or does not occur:

- Inlining will not occur on any form of late-bound method. This includes virtual, dynamic, and message methods.
- Routines containing assembly code will not be inlined.
- Constructors and destructors will not be inlined.
- The main program block, unit initialization, and unit finalization blocks cannot be inlined.

- Routines that are not defined before use cannot be inlined.
- Routines that take open array parameters cannot be inlined.
- Code can be inlined within packages, however, inlining never occurs across package boundaries.
- No inlining will be done between units that are circularly dependent. This included indirect circular dependencies, for example, unit A uses unit B, and unit B uses unit C which in turn uses unit A. In this example, when compiling unit A, no code from unit B or unit C will be inlined in unit A.
- The compiler can inline code when a unit is in a circular dependency, as long as the code to be inlined comes from a unit outside the circular relationship. In the above example, if unit A also used unit D, code from unit D could be inlined in A, since it is not involved in the circular dependency.
- If a routine is defined in the interface section and it accesses symbols defined in the implementation section, that routine cannot be inlined.
- If a routine marked with inline uses external symbols from other units, all of those units must be listed in the uses statement, otherwise the routine cannot be inlined.
- Procedures and functions used in conditional expressions in while-do and repeat-until statements cannot be expanded inline.
- Within a unit, the body for an inline function should be defined before calls to the function are made. Otherwise, the body of the function, which is not known to the compiler when it reaches the call site, cannot be expanded inline.

If you modify the implementation of an inlined routine, you will cause all units that use that function to be recompiled. This is different from traditional rebuild rules, where rebuilds were triggered only by changes in the interface section of a unit.

OPERATOR OVERLOADING

Delphi allows certain functions, or "operators" to be overloaded within record declarations. The name of the operator function maps to a symbolic representation in source code. For example, the Add operator maps to the + symbol. The compiler generates a call to the appropriate overload, matching the context (i.e. the return type, and type of parameters used in the call), to the signature of the operator function.

The following table shows the Delphi operators that can be overloaded:

Operator	Category	Declaration Signature	Symbol Mapping
Implicit	Conversion	Implicit(a : type) : resultType;	implicit typecast
Explicit	Conversion	Explicit(a: type) : resultType;	explicit typecast
Negative	Unary	Negative(a: type) : resultType;	-
Positive	Unary	Positive(a: type): resultType;	+
Inc	Unary	Inc(a: type) : resultType;	Inc
Dec	Unary	Dec(a: type): resultType	Dec

LogicalNot	Unary	LogicalNot(a: type): resultType;	not
Trunc	Unary	Trunc(a: type): resultType;	Trunc
Round	Unary	Round(a: type): resultType;	Round
In	Set	In(a: type; b: type) : Boolean;	in
Equal	Comparison	Equal(a: type; b: type) : Boolean;	=
NotEqual	Comparison	NotEqual(a: type; b: type): Boolean;	<>
GreaterThan	Comparison	GreaterThan(a: type; b: type) Boolean;	>
GreaterThanOrEqual	Comparison	GreaterThanOrEqual(a: type; b: type): Boolean;	>=
LessThan	Comparison	LessThan(a: type; b: type): Boolean;	<
LessThanOrEqual	Comparison	LessThanOrEqual(a: type; b: type): Boolean;	<=
Add	Binary	Add(a: type; b: type): resultType;	+
Subtract	Binary	Subtract(a: type; b: type) : resultType;	-
Multiply	Binary	Multiply(a: type; b: type) : resultType;	*
Divide	Binary	Divide(a: type; b: type) : resultType;	/
IntDivide	Binary	IntDivide(a: type; b: type): resultType;	div
Modulus	Binary	Modulus(a: type; b: type): resultType;	mod
LeftShift	Binary	LeftShift(a: type; b: type): resultType;	shl
RightShift	Binary	RightShift(a: type; b: type): resultType;	shr
LogicalAnd	Binary	LogicalAnd(a: type; b: type): resultType;	and
LogicalOr	Binary	LogicalOr(a: type; b: type): resultType;	or
LogicalXor	Binary	LogicalXor(a: type; b: type): resultType;	xor
BitwiseAnd	Binary	BitwiseAnd(a: type; b: type): resultType;	and
BitwiseOr	Binary	BitwiseOr(a: type; b: type): resultType;	or
BitwiseXor	Binary	BitwiseXor(a: type; b: type): resultType;	xor

Here is an example that implements addition, subtraction, implicit and explicit operators:

```
TMyClass = record
  class operator Add(a, b: TMyClass): TMyClass; // Addition
  class operator Subtract(a, b: TMyClass): TMyClass; // Subtraction
  class operator Implicit(a: Integer): TMyClass; // integer to TMyClass
  class operator Implicit(a: TMyClass): Integer; // TMyClass to integer
  class operator Explicit(a: Double): TMyClass; // Double to TMyClass
end;
// Method implementation example. Add
class operator TMyClass.Add(a, b: TMyClass): TMyClass;
begin
   // ...
end;
var
  x, y: TMyClass;
begin
   x := 12;      // Implicit conversion of Integer, executes Implicit method
   y := x + x;   // Executes TMyClass.Add(a, b: TMyClass): TMyClass
   b := b + 100; // Executes TMyClass.Add(b, TMyClass.Implicit(100))
end;
```

No operators other than those listed in the table may be defined on record.

Overloaded operator methods cannot be referred to by name in source code. To access a specific operator method of a specific class or record, you must use explicit typecasts on all of the operands. Operator identifiers are not included in the class or record's list of members.

No assumptions are made regarding the distributive or commutative properties of the operation. For binary operators, the first parameter is always the left operand, and the second parameter is always the right operand. Association is assumed to be left-to-right in the absence of explicit parentheses.

Resolution of operator methods is done over the union of accessible operators of the types used in the operation (note this includes inherited operators). For an operation involving two different types A and B, if type A has an implicit conversion to B, and B have an implicit conversion to A, an ambiguity will occur. Implicit conversions should be provided only where absolutely necessary, and reflexivity should be avoided. It is best to let type B implicitly convert itself to type A, and let type A have no knowledge of type B (or vice versa).

As a general rule, operators should not modify their operands. Instead, return a new value, constructed by performing the operation on the parameters. Overloaded operators are used most often in records (i.e. value types).

CLASS HELPERS

A class helper is a type that - when associated with another class - introduces additional method names and properties which may be used in the context of the associated class (or its descendants). Class helpers are a way to extend a class without using inheritance. A class helper simply introduces a wider scope for the compiler to use when resolving identifiers. When you declare a class helper, you state the helper name, and the name of the class you are going to extend with the helper. You can use the class helper any place where you can legally use the extended class. The compiler's resolution scope then becomes the original class, plus the class helper.

Class helpers provide a way to extend a class, but they should not be viewed as a design tool to be used when developing new code. They should be used solely for their intended purpose, which is language and platform RTL binding. You can see an example below.

```
type
  TMyClass = class
    procedure MyProc;
    function  MyFunc: Integer;
  end;

...
```

```
procedure TMyClass.MyProc;
var
  X: Integer;
begin
  X := MyFunc;
end;

function TMyClass.MyFunc: Integer;
begin
  ...
end;

type
  TMyClassHelper = class helper for TMyClass
    procedure HelloWorld;
    function MyFunc: Integer;
  end;

procedure TMyClassHelper.HelloWorld;
begin
  Writeln(Self.ClassName); // Self refers to TMyClass, not TMyClassHelper
end;

function TMyClassHelper.MyFunc: Integer;
begin
  ...
end;

var
  X: TMyClass;

begin
  X := TMyClass.Create;
  X.MyProc;    // Executes TMyClass.MyProc
  X.HelloWorld; // Executes TMyClassHelper.HelloWorld
  X.MyFunc;    // Executes TMyClassHelper.MyFunc
end.
```

Note that the reference is always pointed to TMyClass. The compiler recognizes when it's appropriate to execute the call in TMyClassHelper.

STRICT PRIVATE AND STRICT PROTECTED

In Delphi, you have two options for determining the visibility of a class' attributes: strict private and strict protected.

- **Strict private**: class attributes are visible only within the class in which it is declared. Those attributes can't be seen by methods declared in the same unit, or by those that are not part of the class.
- **Strict protected**: determines that class attributes and their descendents are visible.

RECORDS SUPPORT METHODS

Record data types in Delphi represent a mixed set of elements. Each element is called a field and the declaration of a record type specifies a name and type for each field.

Records as of Delphi 2006 are even more powerful, bringing features supported only in class. Here's a list of the new record features in Delphi 2006:

- Constructors
- Operator overload
- Non-virtual methods declaration
- Static methods and properties

The following example is the implementation of a record with the new characteristics:

```
type
  TMyRecord = Record
  type
    TColorType = Integer;
    var
      pRed : Integer;
    class var
          Blue : Integer;
        Procedure printRed();
        Constructor Create(Val : Integer);
        Property Red : TColorType Read pRed Write pRed;
        Class Property  pBlue : TColorType Read Blue Write Blue;
      End;

Constructor TMyRecord.Create(Val: Integer);
Begin
  Red := Val;
End;

Procedure TMyRecord.printRed;
Begin
  WriteLn('Red: ', Red);
End;
```

Now the record can use many of the functionalities that were exclusive to classes. However, records are not classes, meaning they still have many differences:

- Records do not support inheritance.
- Records may have variable parts; classes may not.
- Records are data types and, as so, can be copied. Classes cannot.
- Records have no destructors.
- Records do not support virtual methods.

CLASS ABSTRACT, CLASS SEALED, CLASS CONST, CLASS TYPE, CLASS VAR, CLASS PROPERTY

There are many ways for you to declare classes, types, variables and properties.

- Class abstract → defines an abstract class
- Class sealed → classes cannot be extended through inheritance - such a class cannot be used as a base class for some other (derived) class
- Class const → defines a class constant that can be accessed without having to instantiate the class
- Class type → defines a class type that can be accessed without having to instantiate the class
- Class var → defines a variable from the scope of the class which you can access without having to instantiate the class
- Class property → grants access to the property without requiring the class to be instantiated

NESTED CLASSES

Nested types are used throughout object-oriented programming in general. They allow you to keep conceptually related types together, and to avoid name collisions. The same syntax for declaring nested types may be used with the Delphi compiler. Class sample below:

```
type
      TOuterClass = class
       strict private
          myField: Integer;

       public
          type
             TInnerClass = class
              public
                myInnerField: Integer;
                procedure innerProc;
             end;

         procedure outerProc;
       end;

This is how the method is implemented:

procedure TOuterClass.TInnerClass.innerProc;
begin
   ...
end;
```

To access the method within the Nested class, see the next example:

```
var
   x: TOuterClass;
   y: TOuterClass.TInnerClass;

begin
   x := TOuterClass.Create;
   x.outerProc;
   ...
   y := TOuterClass.TInnerClass.Create;
```

FINAL METHODS

The Delphi compiler also supports the concept of a final virtual method. When the keyword final is applied to a virtual method, no descendent class can override that method. Use of the final keyword is an important design decision that can help document how the class is intended to be used. It can also give the compiler hints that allow it to optimize the code it produces.

STATIC CLASS METHOD

When these classes are declared as Static they do not need to be instantiated.

FOR … IN

Delphi 2007 brought support for element-in-collection (collections, arrays, string expressions and set-type expressions) style iteration over containers.

Example: Iteration in an Array

```
var
  IArray1: array[0..9] of Integer   = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
  I: Integer;
begin

  for I in IArray1 do
  begin
     // do something here...
  end;
```

Example: Iteration in a String

```
var
  C: Char;
  S1, S2: String;

  OS1, OS2: ShortString;
  AC: AnsiChar;
```

```
begin

  S1 := 'New resources in Delphi 2009';
  S2 := '';

  for C in S1 do
    S2 := S2 + C;

  if S1 = S2 then
    WriteLn('SUCCESS #1');
  else
    WriteLn('FAIL #1');

  OS1 := 'Migrating from Delphi 7 to Delphi 2009...';
  OS2 := '';

  for AC in OS1 do
    OS2 := OS2 + AC;

  if OS1 = OS2 then
    WriteLn('SUCCESS #2');
  else
    WriteLn('FAIL #2');

end.
```

GENERICS

Support for *generics* was introduced in Delphi 2009.

What are generics? 'Generics' is the defining term for generic types. It is a language construct that allows you to predefine any type of data from arrays, collections and other sorts of lists. Using Generics allows you to write code in a generic way and have it work with a specific type of data - classes or class methods. It's also possible to define types at runtime.

While using Generics, you will often also work with Collections. The Delphi RTL provides several pre-defined Collections (defined in the "Generics.Collections" unit) see the of data or objects including

- TList
- TQueue
- TStack
- TDictionary
- TObjectList
- TObjectQueue
- TObjectDictionary
- TThreadedQueue

Items added to these classes are of TObject type, meaning you can only add a single type to the list. Whenever you need to read the items you must perform the type cast for each of them, leaving all processing for the compiler. This is nothing but additional effort for your application, or something that can potentially impact its performance. A relevant portion of our code can be adapted to work with generics. Below you see a sample class whose Key property should be a String and whose Value property should be an Integer. In this case you do not use Generics.

```
type
  TSIPair = class
  private
    FKey: String;
    FValue: Integer;
  public
    function GetKey: String;
    procedure SetKey(Key: String);
    function GetValue: Integer;
    procedure SetValue(Value: Integer);
    property Key: TKey read GetKey write SetKey;
    property Value: TValue read GetValue write SetValue;
  end;
```

Using generics it's possible to define the Key and Value properties as being of any type. This is how you do so:

```
type
  TPair<TKey,TValue>= class   // declares the TPair type with 2 parameters

private
    FKey: TKey;
    FValue: TValue;
  public
    function GetKey: TKey;
    procedure SetKey(Key: TKey);
    function GetValue: TValue;
    procedure SetValue(Value: TValue);
    property Key: TKey read GetKey write SetKey;
    property Value: TValue read GetValue write SetValue;
  end;
```

You can now use this class in many different ways:

```
type
  TSIPair = TPair<String,Integer>; // declares it with types String and Integer
  TSSPair = TPair<String,String>; // declares it with other types
  TISPair = TPair<Integer,String>;
  TIIPair = TPair<Integer,Integer>;
```

You'll find many uses for generics. Examples are provided here as a simple, limited reference.

ANONYMOUS METHODS

As the name suggests, an anonymous method is a procedure or function that does not have a name associated with it. An anonymous method treats a block of code as an entity that can be assigned to a variable or used as a parameter to a method. In addition, an anonymous method can refer to variables and bind values to the variables in the context in which the method is defined. Anonymous methods can be defined and used with simple syntax. They are similar to the construct of closures defined in other languages.

Example:

```
function MakeAdder(y: Integer): TFuncOfInt;
begin
  Result := { START anonymous method } function(x: Integer)
  begin
    Result := x + y;
    end; { END anonymous method }
end;
```

The MakeAdder function returns a nameless function; that is, an Anonymous Method.

Note that MakeAdder returns a value of TFuncOfInt type. The type of the Anonymous Method is declared as a reference to the method.

Example: Executing the MakeAdder function

```
var
  adder: TFuncOfInt;
begin
  adder := MakeAdder(20);
  Writeln(adder(22)); // prints 42
end.
type
  TFuncOfInt = reference to function(x: Integer): Integer;
```

The above statement indicates this Anonymous Method:

- Is a function
- Receives an Integer value
- Returns an Integer value

You can declare both procedures and functions as Anonymous Methods.

```
type
  TSimpleProcedure = reference to procedure;
  TSimpleFunction = reference to function(x: string): Integer;
```

Anonymous Methods offer much more than just a simple execution point in code:

- Binding in variables
- Ease to use and define methods
- Ease to parameterize the code

Virtual Method Interception

Delphi XE has a new type in Rtti.pas called TVirtualMethodInterceptor. It creates a derived metaclass dynamically at runtime that overrides every virtual method in the ancestor, by creating a new virtual method table and populating it with stubs that intercepts calls and arguments. When the metaclass reference for any instance of the "ancestor" is replaced with this new metaclass, the user can then intercept virtual function calls, change arguments on the fly, change the return value, intercept and suppress exceptions or raise new exceptions, or entirely replace calling the underlying method. In concept, it's somewhat similar to dynamic proxies from .NET and Java. It's like being able to derive from a class at runtime, override methods (but not add new instance fields), and then change the runtime type of an instance to this new derived class.

Why would you want to do this? Two obvious purposes spring to mind: testing and remoting. Mock objects have been in vogue in the testing space in other languages for some time. By intercepting method calls, one may more easily verify that a particular subsystem is calling all the right methods, with the correct arguments, in the expected order; similarly, the subsystem can proceed with the return values from these method calls, without necessarily having to hit the database, the network, etc. for what should be a unit test. Remoting on the basis of method calls is somewhat less useful, especially when an unreliable and latency-prone network gets into the stack, but that's not the only usage point. The virtual method interceptor logic was originally implemented to be used as part of DataSnap authentication, so that a call that comes in from the network can still be checked as its code flow spreads throughout the graph of objects.

Anyhow, here's a simple example to get started:

```
uses SysUtils, Rtti;
{$apptype console}
type
  TFoo = class
    // Frob doubles x and returns the new x + 10
    function Frob(var x: Integer): Integer; virtual;
  end;

function TFoo.Frob(var x: Integer): Integer;
begin
  x := x * 2;
  Result := x + 10;
end;
```

```
procedure WorkWithFoo(Foo: TFoo);
var
  a, b: Integer;
begin
  a := 10;
  Writeln('  before: a = ', a);
  try
    b := Foo.Frob(a);
    Writeln('  Result = ', b);
    Writeln('  after:  a = ', a);
  except
    on e: Exception do
      Writeln('  Exception: ', e.ClassName);
  end;
end;

procedure P;
var
  foo: TFoo;
  vmi: TVirtualMethodInterceptor;
begin
  vmi := nil;
  foo := TFoo.Create;
  try
    Writeln('Before hackery:');
    WorkWithFoo(foo);

    vmi := TVirtualMethodInterceptor.Create(foo.ClassType);

    vmi.OnBefore := procedure(Instance: TObject; Method: TRttiMethod;
      const Args: TArray<TValue>; out DoInvoke: Boolean; out Result: TValue)
    var
      i: Integer;
    begin
      Write('[before] Calling ', Method.Name, ' with args: ');
      for i := 0 to Length(Args) - 1 do
        Write(Args[i].ToString, ' ');
      Writeln;
    end;

    // Change foo's metaclass pointer to our new dynamically derived
    // and intercepted descendant
    vmi.Proxify(foo);

    Writeln('After interception:');
    WorkWithFoo(foo);
  finally
    foo.Free;
    vmi.Free;
  end;
end;

begin
  P;
end.
```

Here's what it outputs:

```
Before hackery:
  before: a = 10
  Result = 30
  after:  a = 20
After interception:
  before: a = 10
[before] Calling Frob with args: 10
  Result = 30
  after:  a = 20
[before] Calling BeforeDestruction with args:
[before] Calling FreeInstance with args:
```

You'll notice that it intercepts all the virtual methods, including those called during destruction, not just the one I declared. (The destructor itself is not included.)

NEW $POINTERMATH {ON – OFF } DIRECTIVE

The $POINTERMATH directive enables mathematic operations with pointers.

NEW WARNINGS

When you compile your application with Delphi XE, four new warnings might appear in your IDE's message window. These messages regard the use of the new UnicodeString type. These new warnings include:

- 1057 Implicit string cast from '%s' to '%s'
- 1058 Implicit string cast with potential data loss from '%s' to '%s'
- 1059: Explicit string cast from '%s' to '%s'
- 1060 Explicit string cast with potential data loss from '%s' to '%s'

DBEXPRESS

FRAMEWORK

One of the most significant changes in Delphi 2007, dbExpress architecture has been restructured, and you can now count on a framework that is totally written in Delphi. We have performed lab tests to simulate the most diverse situations with varied databases; in some of those tests the performance was improved by 100%.

DbExpress 4 is also a milestone for applications developed in Delphi that require database connectivity. The new architecture was designed to support Win32 and other platforms. For example the dbExpress DataSnap driver can be used for Win32, Win64 and Mac OS X, and it allows developers to create DataSnap client applications in Delphi and deploy on Windows and Mac.

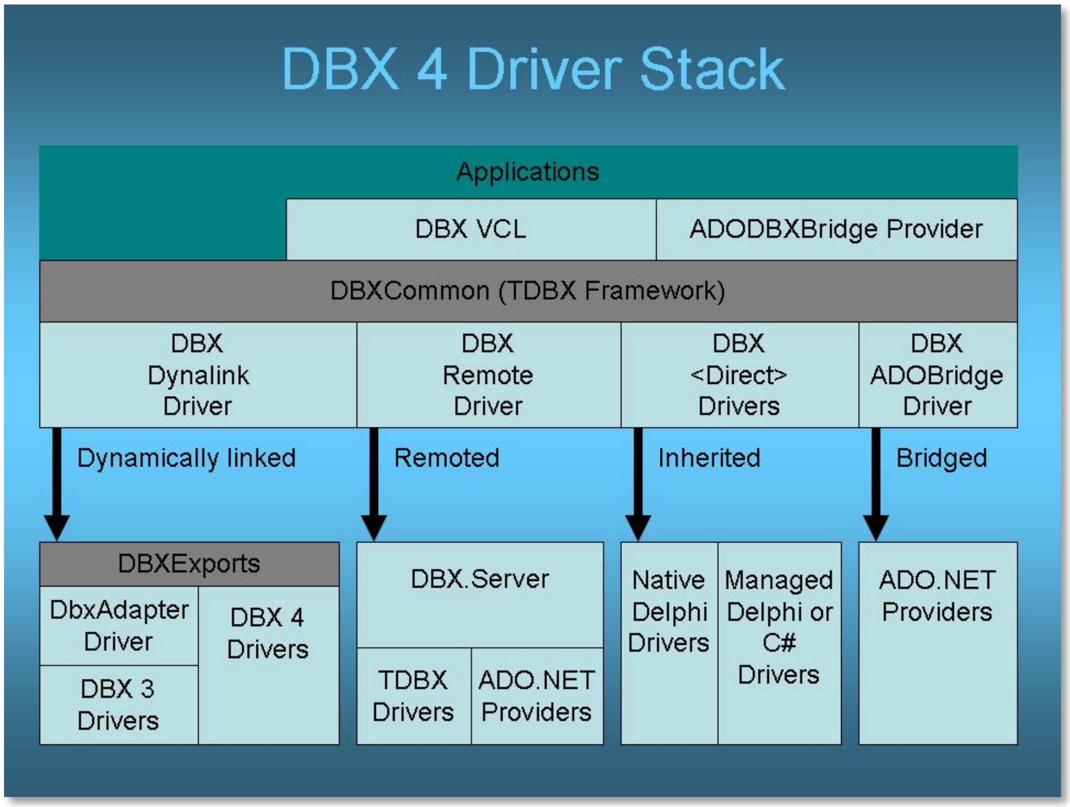


Figure 40 dbExpress 4 Architecture

Remember that applications developed in prior versions are 100% compatible with Delphi XE.

The dbExpress Framework comes with a new group of classes that facilitate the task of accessing and otherwise handling databases. Now you can find all the information regarding the database within the framework. Previously you'd have to use components SQLConnection, SQLDataSet, SQLQuery, and others instead.

The following example represents a console application that uses database connection resources, reads connections parameters, sends a query and displays its result – all in a single transaction.

```
program DBX4Example;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  DBXDynalink,
  Dialogs,
  DBXCommon;

var

  aConnName: string;
  aDBXConn: TDBXConnection;
  aDBXTrans : TDBXTransaction;
  aCmnd: TDBXCommand;
  aReader: TDBXReader;
  i, aColCount: integer;

begin

    aDBXConn := TDBXConnectionFactory.GetConnectionFactory.GetConnection(
'EMPLOYEE', 'sysdba','masterkey');

    // Write the all connection parameters
    Writeln( '================= Connection Properties ============' );
    WriteLn(  aDBXConn.ConnectionProperties.Properties.Text);
    Writeln( '====================================================' );
    Writeln( '' );

    if aDBXConn <> nil then
    begin

      aCmnd := aDBXConn.CreateCommand;

      // Start transaction
      aDBXTrans:= aDBXConn.BeginTransaction(TDBXIsolations.ReadCommitted);

      // Prepare and execute the SQL Statement
      aCmnd.Text := 'SELECT * FROM Country';
      aCmnd.Prepare;
      aReader := aCmnd.ExecuteQuery;

      aColCount := aReader.ColumnCount;
      Writeln( 'Results from Query: ' + aCmnd.Text );
      Writeln( 'Number of Columns:  ' + IntToStr(aColCount) );

      while aReader.Next do
      begin
        Writeln( aReader.Value['Country'].GetAnsiString );
      end;

      Writeln( '=====================================================' );
      Writeln( '' );

      end;
      // Commit transaction
      aDBXConn.CommitFreeAndNil(aDBXTrans);

      Readln;
      aReader.Free;
      aCmnd.Free;
      aDbxConn.Free;

    end;

end.
```

DBEXPRESS METADATA

The new metadata support is used extensively by the Data Explorer pane of the Delphi IDE, but can also be used by any application. In short, you'll not only be able to browse the database structure, but also be able to use classes and objects to modify it, rather than relying directly on the native database SQL commands for creating and modifying data structures. Not only will the code look more object-oriented, but it will be also easier to target different database servers with the same code, as dbExpress abstracts the metadata capabilities of each server.

The unit DBXMetaDataNames has been provided to read metadata. The dbExpress class TDBXMetaDataCommands provides a set of constants to read various types of metadata. Set the TDBXCommand.CommandType property to DBXCommandTypes.DBXMetadata and set TDBXCommand.Text to one of the constants in TDBXMetaDataCommands to acquire the designated metadata TDBXCommand.ExecuteQuery returns a TDBXReader to access the metadata. The new classes in DBXMetaDataNames describe and provide access to this metadata's columns. Below a list of metadata you can read using dbExpress:

- Data types
- Tables
- Columns (from tables, views, etc.)
- Indexes
- Fields from those indexes
- Foreign key

- Fields from Foreign keys
- Stored Procedures
- Stored Procedures' parameters
- User list
- Catalogs
- Schemas
- Views
- Synonyms
- Stored Procedures' source code
- Packaged Stored Procedures
- Packaged Stored Procedures' source code
- Packaged Stored Procedures' parameters
- Roles
- Reserved words

Data Explorer includes support for creating SQL dialect sensitive CREATE, ALTER, and DROP statements. dbExpress also exposes a DbxMetaDataProvider class that surfaces this capability for applications. This slightly increases the size of application, since the metadata writers must be included. The ability to generically create tables is useful for many applications. The interface allows you to describe what a table and its columns look like and pass this description to the TdbxMetaDataProvider.CreateTable method.

The following example on how to create tables, primary keys, foreign keys, and indexes using dbExpress Framework's classes.

```
var
  Provider: TDBXDataExpressMetaDataProvider;
  Country, State: TDBXMetaDataTable;
  IdCountryField,
  IdField: TDBXInt32Column;
  StrField : TDBXUnicodeVarCharColumn;
begin
  Provider := DBXGetMetaProvider(conn.DBXConnection);

  // Country
  Writeln('Creating Table - Country ....................');
  Country := TDBXMetaDataTable.Create;
  Country.TableName := 'COUNTRY';

  IdCountryField := TDBXInt32Column.Create('COUNTRYID');
  IdCountryField.Nullable := false;
  IdCountryField.AutoIncrement := true;
  Country.AddColumn(IdCountryField);

  StrField := TDBXUnicodeVarCharColumn.Create('COUNTRYNAME', 50);
  StrField.Nullable := False;
```

```
    Country.AddColumn(StrField);

    Provider.CreateTable(Country);
  end;
    // Defines COUNTRYID as Primary Key
    AddPrimaryKey(Provider, Country.TableName, IdCountryField.ColumnName);

    // Defines Unique Index as COUNTRYNAME
    AddUniqueIndex(Provider, Country.TableName, StrField.ColumnName);

    // State
    Writeln('Creating Table - State ....................');
    State := TDBXMetaDataTable.Create;
    State.TableName := 'STATE';

    IdField := TDBXInt32Column.Create('STATEID');
    IdField.Nullable := false;
    IdField.AutoIncrement := true;
    State.AddColumn(IdField);

    StrField := TDBXUnicodeVarCharColumn.Create('SHORTNAME', 2);
    StrField.Nullable := False;
    State.AddColumn(StrField);

    StrField := TDBXUnicodeVarCharColumn.Create('STATENAME', 50);
    StrField.Nullable := False;
    State.AddColumn(StrField);

    State.AddColumn(IdCountryField);

    Provider.CreateTable(State);

    // Defines STATEID as Primary Key
    AddPrimaryKey(Provider, State.TableName, idField.ColumnName);

    // Defines Unique Index as STATENAME
    AddUniqueIndex(Provider, State.TableName, StrField.ColumnName);

    AddForeignKey(Provider, State.TableName, IdCountryField.ColumnName,
                  Country.TableName, IdCountryField.ColumnName);

    FreeAndNil(Provider);
    FreeAndNil(Country);
```

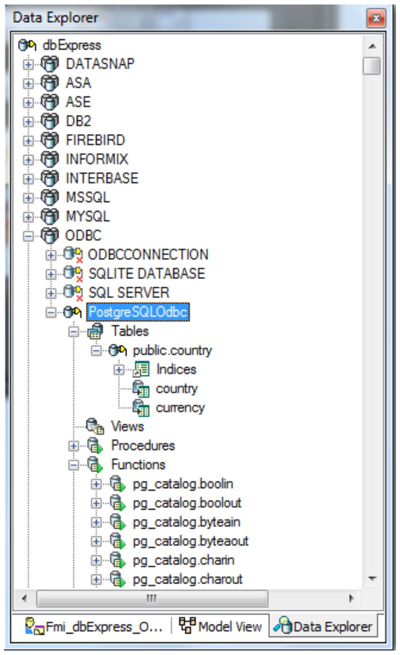
The source code for this example is available at http://cc.embarcadero.com/Item/26210.

DBEXPRESS DRIVERS

Support for the latest versions of databases: InterBase XE2, MySQL 5.1, Oracle 10g/11g and two new dbExpress drivers:

- New dbExpress SQL Server 2008 driver includes support the new data type datetime offset
- New dbExpress ODBC driver support

The dbExpress ODBC driver introduced in Delphi XE2 allows your application to connect with other databases that doesn't have dbExpress native drivers, like SQLite, PostgreSQL and others.



```
Data Explorer
dbExpress
  DATASNAP
  ASA
  ASE
  DB2
  FIREBIRD
  INFORMIX
  INTERBASE
  MSSQL
  MYSQL
  ODBC
    ODBCCONNECTION
    SQLITE DATABASE
    SQL SERVER
    PostgreSQLOdbc
      Tables
        public.country
          Indices
          country
          currency
      Views
      Procedures
      Functions
        pg_catalog.boolin
        pg_catalog.boolout
        pg_catalog.byteain
        pg_catalog.byteaout
        pg_catalog.charin
        pg_catalog.charout
Fmi_dbExpress_O...  Model View  Data Explorer
```

Figure 41. Data Explorer connected to PostgreSQL via dbExpress ODBC driver

The drivers for Oracle, InterBase and MySQL now come with Unicode support.

As Delphi XE2 supports Windows and Mac development, the drivers for the following databases are available for Mac: Informix, Oracle, SQL Anywhere, Firebird, InterBase and MySQL.

Delphi 2010 brought a dbExpress driver for Firebird, which was the number one request at that time. The dbExpress driver for Firebird supports Firebird version 1.5 and 2.x, and not only that, the DBX Framework is fully compatible with Firebird so you can create tables, primary keys, foreign keys, indexes and more through the framework. Data Explorer fully supports Firebird as well.

BIGINT fields are 100% supported and are mapped as TLargeIntField in the VCL, and the BLOB type is mapped as TBlobField.

Certainly the question arises, support for the blob and bigint as it gets. BIGINT fields are 100% supported and are mapped as TLargintField the VCL, the BLOB works perfectly and are mapped as TBlogField.

New concepts, called “Delegate Driver” and “Pools Connections”, are available in dbExpress and require simple parameter configuration.

You can also extend the dbExpress framework to write delegation drivers, which provide an extra layer between the application and the actual driver. Delegate drivers are useful for connection pooling, driver profiling, tracing, and auditing DBXTrace is a delegate driver used for tracing.

Below you see the log result generated by the Delegate in Delphi language. It’s easy to read, understand, and even execute operations once again.

Trace configuration. The following example captures events according to the TraceFlags parameter, saving the log file at c:\dbxtrace.txt. The dbExpress connection has a parameter to indicate the trace configuration; for instance: DelegateConnection= DBXTraceConnection

```
[DBXTraceConnection]
DriverName=DBXTrace
TraceFlags=EXECUTE;COMMAND;CONNECT
TraceDriver=true
TraceFile=c:\dbxtrace.txt
```

Generated log result:

```
Log Opened ==========================================
{CONNECT        } ConnectionC1.Open;
{COMMAND      } CommandC1_1 := ConnectionC1.CreateCommand;
{COMMAND      } CommandC1_1.CommandType := 'Dbx.SQL';
{COMMAND      } CommandC1_1.CommandType := 'Dbx.SQL';
{COMMAND      } CommandC1_1.Text := 'select * from employee';
{PREPARE          } CommandC1_1.Prepare;
{COMMAND      } ReaderC1_1_1 := CommandC1_1.ExecuteQuery;
{COMMAND      } CommandC1_2 := ConnectionC1.CreateCommand;
{COMMAND      } CommandC1_2.CommandType := 'Dbx.MetaData';
{COMMAND      } CommandC1_2.Text := 'GetIndexes "localhost:C:\
database\employee.ib"."sysdba"."employee" ';
{COMMAND      } ReaderC1_2_1 := CommandC1_2.ExecuteQuery;
{READER         }  { ReaderC1_2_1 closed.  6 row(s) read }
{READER         } FreeAndNil(ReaderC1_2_1);
{COMMAND    } FreeAndNil(CommandC1_2);
```

You can also use connection pools with dbExpress natively. Below you see an *alias* (Pool_DelegateDemo) passing to DBXPoolConnection the control over the connection pool (where the maximum number of connections is set).

```
[DBXPoolConnection]
DriverName=DBXPool
MaxConnections=16
```

```
MinConnections=0
ConnectTimeout=0
```

```
[Pool_DelegateDemo]
DelegateConnection=DBXPoolConnection
DriverName=Interbase
DriverUnit=DBXDynalink
DriverPackageLoader=TDBXDynalinkDriverLoader
DriverPackage=DBXCommonDriver110.bpl
DriverAssemblyLoader=Borland.Data.TDBXDynalinkDriverLoader
DriverAssembly=Borland.Data.DbxCommonDriver,Version=11.0.5000.0,Culture=neutra
l,PublicKeyToken=a91a7c5705831a4f
Database=localhost:C:\database\employee.ib
RoleName=RoleName
User_Name=sysdba
Password=masterkey
ServerCharSet=
SQLDialect=3
BlobSize=-1
CommitRetain=False
WaitOnLocks=True
ErrorResourceFile=
Interbase TransIsolation=ReadCommited
Trim Char=False
```

Cloud API

Delphi XE2 includes expanded cloud computing integration with new data and deployment options. This API is designed to be easier to understand and easier to use. It is fully code commented, and should be pretty self explanatory. It still works with the same services provided in XE release, the Azure API: Azure Queue Service, Azure Table Service and Azure Blob Service.

The Cloud API in XE2 includes anew Amazon API, which interacts with Amazon's AWS services, similar to Azure's: Amazon Simple Queue Service (SQS), Amazon SimpleDB and Amazon Simple Storage Service (S3.) Again, this API is designed to be easy to understand and use, and is fully code commented.

You can use both of these Cloud APIs, along with your login credentials to access your Azure and Amazon accounts. You can create queues, add messages to queues, and pop messages from queues. For example,yYou can work with both cloud services' NOSQL Database service, to store and retrieve data from tables stored on the cloud. And you can use the blob/storage service cloud APIs to upload files to the clouds and download files from the clouds.

The following code uses the Cloud API to connect to Amazon and upload a file to Amazon S3, but before the upload I can list all the buckets (folders), create one if I need and them upload the file to the bucket I chose.

```
function TCloud.UploadtoAmazon(FileName: String): TCloudResponseInfo;
var
  StorageService: TAmazonStorageService;
  BucketList, Metadata: TStrings;
  Content: TBytes;
  ResponseList: TCloudResponseInfo;
  FrmList: TFrmContainerList;
  Act: TContainerAction;
  BucketName: String;
  I: Integer;
begin
  Result := TCloudResponseInfo.Create;

  if (FileName = EmptyStr) then
    Exit;

  StorageService := TAmazonStorageService.Create(AmazonConnection);

{$REGION 'Define the Bucket'}
  ResponseList := TCloudResponseInfo.Create;
  BucketList := StorageService.ListBuckets(ResponseList);

  if ResponseList.StatusCode = 200 then
  begin
```

```
    // Amazon return date/time information for each Bucket
    // this for is required to remove that information
    for I := 0 to BucketList.Count - 1 do
      BucketList[I] := BucketList.Names[I];

    FrmList := TFrmContainerList.Create(nil, BucketList,
TClouds[AmazonIndex]);
    try
      FrmList.ShowModal;
      Act := FrmList.Action;

      case Act of
        caCreate:
          begin
            if StorageService.CreateBucket(FrmList.Container,
amzbaNotSpecified,
              amzrNotSpecified, Result) then
              BucketName := FrmList.Container;
          end;
        caUpload:
          begin
            BucketName := FrmList.Container;
          end;
      end;
    finally
      FrmList.Free;
    end;

    if Act = TContainerAction.caNone then
      Exit;
```

end;
You can download a complete sample that use the Cloud API for Amazon and Windows Azure at:
https://radstudiodemos.svn.sourceforge.net/svnroot/radstudiodemos/branches/RadStudio_XE2/Delphi/CloudAPI/CloudUpload

DATASNAP

Integration between DataSnap and dbExpress is among the most significant new features in Delphi 2009. Later releases added even more features. Based on feedback received from our customers and user groups, we created the new DataSnap. As usual, creating a multilayer application seemed easy. However, considering its continued use and the impressive number of DataSnap applications available, many opportunities for improvement have been identified. In this section, we explore the concepts surrounding the creation of multilayer applications with the new DataSnap.

The DataSnap today is the core technology in Delphi for multi-tier development, in XE release DataSnap provide a open architecture for other technologies to execute the

Server Methods (business roles implemented on the server side), languages like: C++Builder, .NET (Delphi, C#, VB.NET, etc.), PHP, Java, JavaScript and other languages.

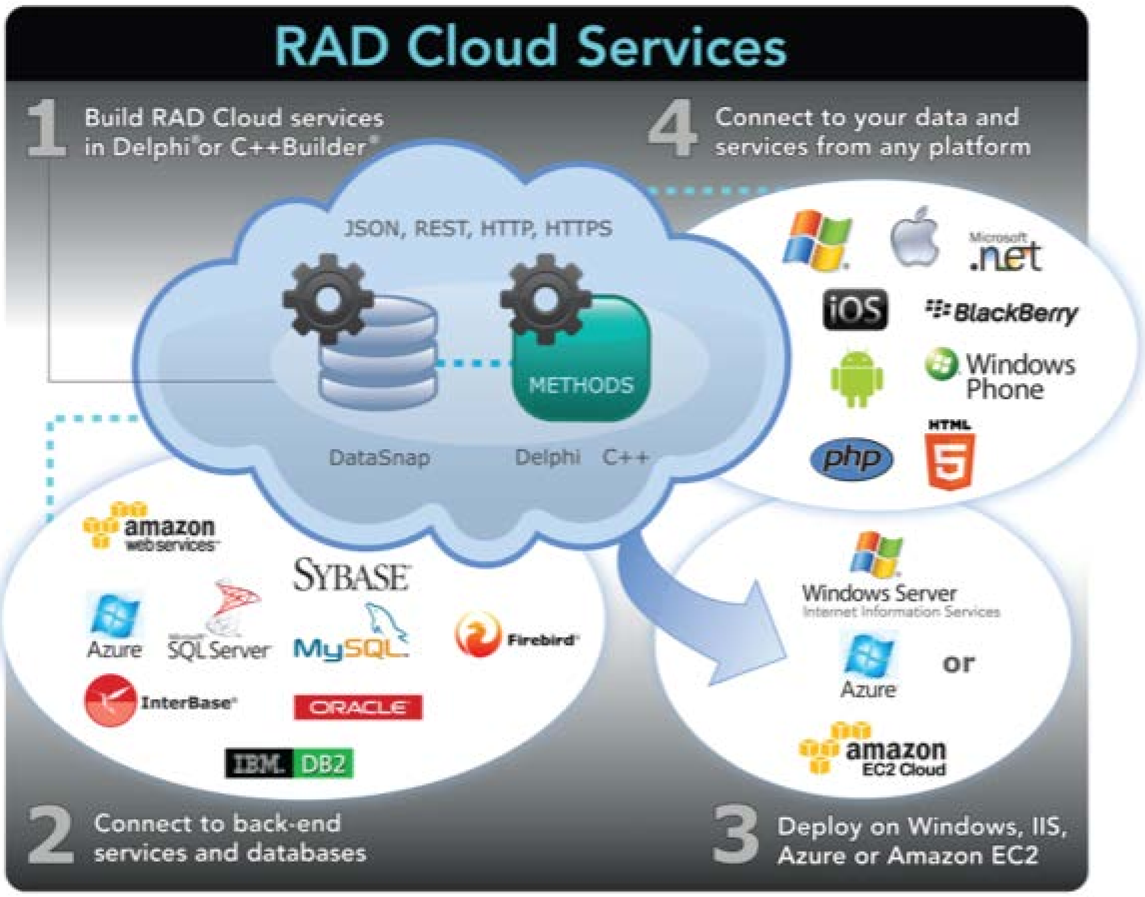

RAD Cloud Services
1 Build RAD Cloud services in Delphi® or C++Builder®
4 Connect to your data and services from any platform
JSON, REST, HTTP, HTTPS
METHODS
DataSnap
Delphi C++
Microsoft .net
iOS
BlackBerry
Windows Phone
php
HTML 5
amazon webservices
SYBASE
Azure
SQL Server
MySQL
Firebird
InterBase
ORACLE
IBM. DB2
Windows Server Internet Information Services
Azure
or
amazon EC2 Cloud
2 Connect to back-end services and databases
3 Deploy on Windows, IIS, Azure or Amazon EC2

Figure 42. RAD Cloud Services core technology in Delphi for multitier development

CONCEPTS

The new integration between DataSnap and dbExpress brought great flexibility into the world of multi-layer application development. Before that, sending and receiving data by means of ClientDataSet was the practice. Working with remote functions was something that required the use of a Type Library, consequently making you dependent on COM (present in Remote Data Module). Developers requested that we remove the COM dependency. This is a main attribute of the new DataSnap: it does not depend on any COM technology. However, this technology was not disregarded and compatibility is maintained, allowing you to use it whenever you find appropriate.

DataSnap integration with dbExpress allows dynamic execution of server methods using, for example, SQLDataSet or the new SqlServerMethod component. The parameter and the result of the method are defined using the Param properties.

One way to execute the methods on the server is to use the new SqlServerMethod component. this component inherits from CustomSQLDataSet, which means you can execute server side methods using a DataSet, input/output parameters will be represented by Params property.

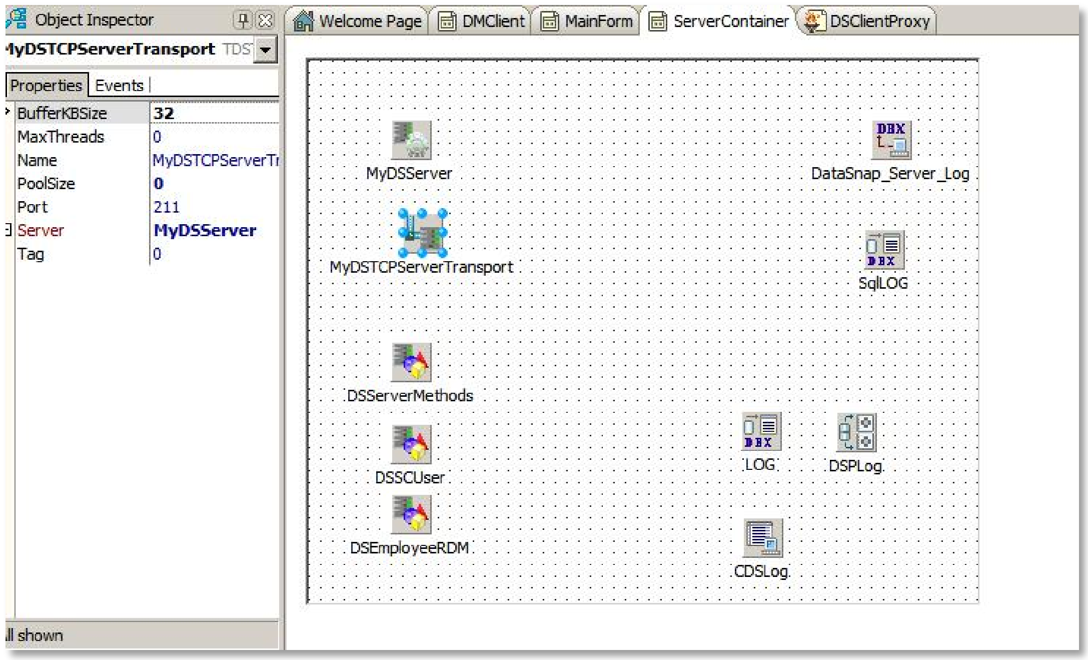
SERVER CONTAINER

DataSnap servers are defined by two components: DSServer and DSTCPTransport or DSHTTPTransport. Add these two components to your application and you have a DataSnap server. The form/datamodule that hosts these components can be called Server Container. From now on we will start using a new naming convention.

DSServer is your DataSnap server. When connected to DSTCPTransport – which is where you define the connection port, the maximum number of threads, etc. – it exposes your application as a server.

DSTCPTransport transports information by means of TCP, thus enabling you to extend and create new transport methods (e.g., HTTPS, SSL, to name a few). DSTCPTransport currently uses Indy's infrastructure for TCP connections.

If you use BSS, your application will still work in Delphi XE. However, it's recommended that you migrate to the new DataSnap.



BufferKBSize	32
MaxThreads	0
Name	MyDSTCPServerTr
PoolSize	0
Port	211
Server	MyDSServer
Tag	0

Figure 43. Server Container based on DataModule

SERVER MODULE

You most probably have many classes that hold business rules which would otherwise be of better use in a multi-layer application. Using Server Methods you can easily expose all public methods to the client side.

In order for a class to be made available as a Server Method, it must:

- Descend from TPersistent
- Have the {$MethodInfo ON} directive. This directive allows dbExpress to obtain information about the class from RTTI.
- Be registered by means of the DSServerClass component.

Finally, we use the term Server Module to define the location of the providers, Server Methods, etc. You can create a Server Module selecting File | New | Other | Delphi Files | Server Module.

The Server Module is a DataModule that comes with the directive $MethodInfo ON by default.

Each class you make available has an associated DSServerClass component. This component is responsible for registering the class and making it available to client applications. It's recommended that you keep your DSServerClass components in the Server Module. You can have as many Server Modules as you wish, which helps you to have a better organized application.

The various DSServerClass define the application lifetime, or the LifeCycle property:

- Server → One component instance is used per server (Singleton)
- Session → One component instance is used per DataSnapSession: (Statefull).
- Invocation → One component instance is used per invocation of a method (Stateless).

Additionally, DSServerClass comes with a few events that require you to use the OnGetClass event to get the class registered. Below you see an example of it, with TServerMethods working as a datamodule that holds many methods:

```
procedure TDMServerContainer.DSServerMethodsGetClass (DSServerClass :
TDSServerClass; var PersistentClass : TPersistentClass);
begin
  PersistenClass := TServerMethods
end;
```

Proceed the same way in case you have a Remote Data Module in your application.

SESSION MANAGEMENT

When a client connects to a DataSnap server, a session is created. This session is represented with a TDSSession instance or subclass. The TDSAuthSession class extends TDSSession and is itself subclassed (TDSRESTSession, TDSTCPSession, TDSTunnelSession.) These instances hold a TDSCustomAuthenticationManager instance and allow for Authorization checking.

Adding session events and iterating over available sessions was available in XE, but only worked for HTTP connections. Now this feature is enabled also for TCP/IP connections, and allows you to keep track of all active sessions, and when they are created/closed.

To add a session event, get the singleton instance of the TDSSessionManager class by calling TDSSessionManager.Instance, and then use the AddSessionEvent method to register a procedure with the signature following signature:

```
TDSSessionManager.Instance.AddSessionEvent(
  procedure(Sender : TObject; const EventType: TDSSessionEventType;
            const Session : TDSSession)
begin
  case EventType of
    SessionCreate : (* session was created *);
    SessionClose : (* session was closed *);
  end;
end);
```

The procedure takes an EventType parameter, which says if the session was created or is being closed. It also takes the Session itself, to use however you need. (such as storing data in the session, or getting the Session ID [SessionName].)

If you want to iterate over all sessions, you again do that with the singleton instance of the TDSSessionManager class. Using the ForEachSession method, you are able to iterate all of the registered sessions in a thread-safe manner.

```
TDSSessionManager.Instance.ForEachSession(
  procedure(const Session : TDSSession)
  begin
     // handle session instance
  end);
```

With XE2, sessions can also store string-TObject pairs by using the Object functions (HasObject, GetObject, PutObject and RemoveObject). This can be used for storing complex data in a session, mapping it with a user's connection. Note, however, that any object stored in the session is instance-owned by the session and will be freed by the session when it is destroyed.

DATASNAP MONITORING AND CONTROL

In Delphi XE2, DataSnap servers with TDSTCPServerTransport components are able to monitor connections, and close any TCP connection they wish. The connections are linked with a Session Id, which can be used in a server method or authentication manager to get the TCP connection for the current session. This allows both server methods and authentication managers to terminate a connection for any reason.

On the TDSTCPServerTransport component there are two new events which can be assigned; OnConnect and OnDisconnect.

On the OnConnect event you can capture and stores the Channel, which is an instance of TDSTCPChannel, as well as the connection for later use. The connection will be provided without a channel in the disconnect event, so in a way it can be used to uniquely identify a channel.

On the OnDisconnect event, which is called whenever the TDSTCPServerTransport component is still active and a TCP connection has been closed. If the transport is being disposed, then this event will not be notified. Note that, this event provides only the connection, but this connection could be used to look up a channel you've obtained with the OnConnect event.

By default, the OnDisconnect event will not be notified if the client abruptly loses his internet connection. This is because the socket remains open until an IO operation is attempted and fails. If your OS is configured to use keep-alive packets for all TCP/IP connections then based on its configuration, you will eventually see the disconnect event being notified. If you'd like to control this behavior on a per-connection basis, then you can use the EnableKeepAlive and DisableKeepAlive methods on the TDSTCPChannel:

You can download a full example including source code at
http://radstudiodemos.svn.sourceforge.net/viewvc/radstudiodemos/branches/RadStudio_XE2/Delphi/DataSnap/DataSnapMonitor/

FILTERS

A suite of filters can intercept the communication between a DataSnap client and a DataSnap server. Each filter can perform transformations over the byte stream such as encryption and/or compression; the byte stream can be intercepted by more than one filter and such the output of one becomes the input of the next filter. The filters are attached to the byte stream at design time (or coded), by setting the Filters property of the DataSnap server transport components such as DSTCPServerTransport.TDSTCPServerTransport or DSService.TDSHTTPService.

The filters are available at design time if they are present in a package registered with RAD Studio. The filter needs to be built into a package and the package needs to be installed into Delphi. Server-side design time support enables the filter to show up in the filter list editor. Client-side design time support enables design time connection using TSQLConnection.

There is no need to associate filters at the client side, as they are automatically instantiated based on a handshake protocol between client and server. Hence it is important that the client code registers the filters before connecting to a filtered server either by adding the unit name to the uses clause or in an early stage, such as initialization time.

Delphi XE has a filter based on zlib compression, where the data is compressed automatically. You can also define an encryption filter.

You can find some samples of filters at Code Central http://cc.embarcadero.com/Author/38483.

HTTP TUNNELING

DataSnap 2010 implemented a new feature, which allows developers to implement redundant solutions like Failover and Load Balancing in DataSnap applications. Questions about this subject are very frequent and this topic will give you an overview how to implement Failover. After reading this topic you will be able to understand how you can implement Failover in your DataSnap applications and then use that to implement other redundant solution, like Load Balancing.

When we started to think about application servers as part of our N-tier development we had many goals like centralized processes, business rules, hardware investments, updates, etc. When we think about a centralized process, we also need to think about redundancy, which is the duplication of critical components of a system with the intention of increasing reliability , usually via a backup or fail-safe.

Before Delphi 2010, failover or load balance was not easy to implement, but now it is another story.

DataSnap 2010 brings a feature named HTTP Tunneling which allow you to have control of the data sent and received between the client and the server. HTTP Tunneling communicates through the HTTP protocol. Thus, you have to use that for the communication between client and server, which is not a problem.

When you implement Load Balancing or Failover, you will need a middleware application to control things. This application will be responsible for receiving the data from the client application, analyzing it, and forwarding it to the appropriate Server.

Translating that to the DataSnap world, the client application connects to the Failover Server that is our proxy and it will forward the connection to the appropriate DataSnap Server in case the primary connection fails. The below example will simulate a client application sending/receiving data, while at some point the main DataSnap Server will crash, and you will see the Failover Server redirecting the connection to a secondary DataSnap Server.

You will only need to make two changes on the client application:

- Define HTTP as connection protocol in your SQL Connection
- Connect to the Failover Server and not directly to your DataSnap Server

Besides this you need to create your Failover Server, and you can use this example code as a start.

The Failover Server needs two components, the DSHTTPService which represent your server and is connected to the DSHTTPServiceAuthenticationManager for the authentication process, so that only authorized users can connect to the server.

To enable the HTTP Tunneling feature you will need to implement the following events on the HTTP Service Tunnel Service:

- DSHTTPService1.HttpServer.TunnelService.OnErrorOpenSession
- DSHTTPService1.HttpServer.TunnelService.OnErrorWriteSession
- DSHTTPService1.HttpServer.TunnelService.OnErrorReadSession
- DSHTTPService1.HttpServer.TunnelService.OnOpenSession
- DSHTTPService1.HttpServer.TunnelService.OnWriteSession
- DSHTTPService1.HttpServer.TunnelService.OnReadSession
- DSHTTPService1.HttpServer.TunnelService.OnCloseSession

These events are executed during the communication process. The event names explain what they do, and this example implements all of them. The example has a log which will help you to see what is happening during the communication.

In the case of the Failover solution, all the events need to be implemented. The events OnErrorXXX will be executed when something goes wrong. These events will allow you identify and decide what to do with the bytes transferred. I will focus on the event OnErrorOpenSession that will redirect the connection in case of error during the opening session.

The follow code implements a method named Redirect which is associated with the event OnErrorOpenSession. You can see how the session data is represented by parameters on this method, and how it includes everything you need to redirect the data.

In this sample we use the Session.UserFlag to control if the connection was already redirected.We are only allowing one redirection, and in case that the Sender parameter becomes anException, we will save the error message in our log. After that we created an instance of DBXProperties, which has the new redirection server information. I'm using the same HostName and changed the port to 213, differentiating it from the other DataSnap Server on the same machine, for demo purposes.

After that just reopen the session passing the new properties as parameter and it's done — that is all you need.

```
Procedure TForm6.Redirect(Sender: TObject; Session: TDSTunnelSession; Content:
TBytes; var Count: Integer);
var
  DBXProperties: TDBXDatasnapProperties;
  Msg: String;
begin
  if Sender is Exception then
    Msg := Exception(Sender).Message;
  Log('>>' + Msg);

  if Session.UserFlag = 1 then
    Raise  Exception.Create('Backup  session  cannot  be  redirected  once  more.'  +
Msg);

  DBXProperties := TDBXDatasnapProperties.Create(nil);
  DBXProperties.Values[TDBXPropertyNames.DriverName] := 'Datasnap';
  DBXProperties.Values[TDBXPropertyNames.HostName]  := 'localhost';
  DBXProperties.Values[TDBXPropertyNames.Port]      := '213';

  try
    try
      Session.Reopen(DBXProperties);
      Session.UserFlag := 1;

      Msg := 'Flow commuted to alternate resource.';
      Log('>>' + Msg);
    except
      Raise Exception.Create(Msg + '. Commuting to alternate resource failed.');
    end;
  finally
    DBXProperties.free;
  end;

end;
```

In case you want to run the sample code on your machine, here are the steps you need for that.

The sample includes two projects: Failover Server and DataSnap Server. We will use DataExplorer as our client application. Using the follow steps you will be able to see the Failover Server in action.

- Open and execute the Failover server, the sample use HTTP protocol and port 8080

- Execute the DataSnap Server twice, one using port 211 and other one port 213. The server using the port 213 will work as the backup server
- Create a DataSnap alias in your Data Explorer, remembering to configure it to use HTTP as protocol and port 8080
- On the Stored Procedure node, find the method EchoString, pass the value Delphi 2010 as parameter and execute. The return value will be Delphi 2010 (Server 211)
- Close the DataSnap Server that runs on the port 211
- Repeat the step 4 and look the return value, which should be Delphi 2010 (Server 213)
- Now look the log on the Failover Server, you will see the exception error and the redirection log information

You can download the source code at http://cc.embarcadero.com/Item/27391.

SECURITY

Once you have a server created with some server methods you will probably want to add in some logic to control who can invoke those methods. For this you can use the new/improved functionality in the Authentication Manager component.

To get started you simply drop this component onto your form and set it as the authentication manager to use in the TDSHTTPWebDispatcher, TDHTTPService or TDSTCPServerTransport component, depending on your server's configuration.

The authentication manager has two events; OnUserAuthenticate and OnUserAuthorize. It also has a 'Roles' collection.

The OnUserAuthenticate event is called when a user tries to connect (invoke a method) for the first time, and takes as input parameters connection info such as the user's name and password, and allows you to set a value for the in/out parameter "valid". By default this is set to true, but you can decide, based on the user information or anything else you wish, if you want to set valid to true or false. Setting it to false will deny the user connection, and therefore also deny any invocation they may have been attempting.

You can define Roles in several different ways. You can go to your server methods class and add a TRoleAuth attribute to the code (requires DSAuth unit.) This attribute can either be added at class level, or at method level, like this:

```
[TRoleAuth('admin')]
TServerMethods1 = class(TComponent)
public
```

```
  function EchoString(Value: string): string;
  function ReverseString(Value: string): string;
end;
```

Or this:

```
TServerMethods1 = class(TComponent)
public
  [TRoleAuth('admin')]
  function EchoString(Value: string): string;
  function ReverseString(Value: string): string;
end;
```

In the first example both 'EchoString' and 'ReverseString' would require the user to have the "admin" role to invoke the method. In the second example, only the 'EchoString' method has the admin role associated with it. Note that the TRoleAuth attribute has an optional second parameter, which is the 'denied roles' list which behaves as you would expect. Both of these parameters can be a comma-separated list of roles.

The OnUserAuthorize event is called whenever a user who has already been successfully authenticated tries to invoke a server method. You do not need to implement this event. If you add roles to the UserRoles list in the OnUserAuthenticate event, then those roles alone can be used to decide if the user has permission to invoke any given server method. However, if you wish to have more control over the process (such as allowing or denying invocation based on time of day,) then you can implement this event. Passed into the event is an object containing information such as the user name, the UserRoles populated in the authentication event, and the allowed/denied roles for the method being called. You can use this information, as well as anything else you like, to decide on if you want to set the value of "valid" to true or false, which will allow or deny the method invocation.

CALLBACKS

Delphi 2010 introduced two types of callback, the lightweight and heavyweight. The propose of call callback is to allow server to send notifications to the client.

Lightweight callbacks can be passed as parameters for the server method based on the new type TDBXCallback delivered with DBXJSON unit. Why there? Because callback can exchange JSON values between server and client. It is the server that is now able to send over data wrapped though a JSON value and waits for a client response. Why lightweight? Because the callbacks 'live' only as long as server method executes.

Your business process can now automatically use client side information to base its decisions on what path to follow next.

TDBXCallback requires one thing only: override its Execute method with the logic you want to be executed when server calls it. It is a class, so you can have fields in it with values specific to each instance. These stateful callbacks are managed by DataSnao, i.e. they are freed after server method completes and they are executed within the same thread used to invoke that lengthy server method.

Heavyweight callbacks allows you to, from a web page, register JavaScript functions with the server, and specify in which condition you want them to be notified by specifying the Channel ID to 'listen' to. This allows a thin client to receive nearly instantaneous notifications from the server without needing to constantly issue requests to see if anything has changed/happened.

Callbacks are useful when the server needs to communicate with the client while executing a server method. Callbacks allow a JSON parameter to be passed back to the client and the client can return a JSON value back to the server.

Delphi XE2 implemented monitoring for heavyweight callbacks through the new TDSCallbackTunnelManager class in the DSServer unit. It can be used to keep track of the creation and destruction of heavyweight callback channels, and the callbacks they contain. This allows the server to respond to any state change with the connect clients and their heavyweight callbacks. You can add and remove events with the following functions AddTunnelEvent and RemoveTunnelEvent, these functions contain the parameter that represents what type of event is happening in the tunnel, which this information you can decide what to do on the server side.

You can also register heavyweight callback events on the client side when using dbExpress, Delphi REST and JavaScript REST.

The server will be able to notify all clients connected or specific clients.

DATASNAP REST SERVER

Delphi XE provide REST support for DataSnap, in other words your DataSnap Server can expose all server methods as REST interfaces.

The new DataSnap client proxy generation system included in XE has generators for the Delphi, C++, Delphi Prism, PHP and JavaScript programming languages. The proxy generator handles the complexity of communicating with and consuming REST services by translating the DataSnap methods and complex types into "native" implementations for the target client language.

To create a DataSnap REST Server, Delphi XE provides a DataSnap REST Application Wizard which creates a project that is the starting point for building an AJAX-enabled web application. The communication protocol between server and client applications is HTTP and the architectural style is REST (Representational State Transfer).

The DataSnap REST Application Wizard consists of four or five steps, depending on the type of REST Application you select (in the first step).

In the first step, you are prompted to select the type of REST Application. The possible options are:

- Stand-alone VCL application
  A stand-alone REST VCL application is a web server that displays a VCL form. It supports HTTP/HTTPS using an Indy HTTP server component.

- Stand-alone console application
  A stand-alone REST console application is a web server that has a text-only user interface. It supports HTTP/HTTPS using an Indy HTTP server component.

- ISAPI dynamic link library
  ISAPI and NSAPI Web server applications are shared objects that are loaded by the web server. Client request information is passed to the DLL as a structure and evaluated by TISAPIApplication. Each request message is handled in a separate execution thread. When you select this type of application, the library header of the project files and the required entries are added to the uses list. Also, the clause of the project file is exported. An ISAPI library integrates with IIS. IIS has support for HTTP and HTTPS.

- Web App Debugger executable
  Web Application Debugger provides an easy way to monitor HTTP requests, responses, and response times. Web Application Debugger takes the place of the web server. Once you have debugged your application, you can convert it to one of the other web application types and install it with a commercial web server.

- Class Name
  Class Name is an identifier that will be used in the URL to connect to a particular Web App Debugger executable.

The second step is available only if you choose Stand-alone VCL application or Stand-alone console application in the first step. You are prompted to enter the HTTP communication port. The wizard also allows you to test the availability of the port using the Test Port button. Clicking the Find Open Port button automatically completes the HTTP Port field with one of the auto-detected available ports.

The third step asks for DataSnap REST server features.

If you select the Authentication option, a TDSHTTPServiceAuthenticationManager component is placed on the server form. The TDSHTTPWebDispatcher component uses TDSHTTPServiceAuthenticationManager as the AuthenticationManager to allow the implementation of HTTP user authentication for the DataSnap server. The implementation consists of implementing the HTTPAuthenticate property. When Authentication is selected, the client must provide the DataSnap user name and password as SQL connection properties.

In Delphi XE2 the TDSHTTPService and TDSHTTPWebDispatcher components have a FormatResult event, which allows for modifying the JSON Result being passed back to a client. You have access to the command being invoked by the client, and the JSON response (minus the result JSON Object) the client will be getting.

Select the Server Methods Class option to add a TDSServerClass component to the server form and to allow defining a class on the server that will expose the server methods to the client applications.

If you select the Sample Methods option, ServerMethodsUnit will contain the implementation of two simple methods called EchoString and ReverseString, which return the Value given as a parameter in normal respective reversed states.

The fourth step of the DataSnap REST Application Wizard asks you for the ancestor type of the server method class.

Choose TDSServerModule to expose the data sets from the server to the client applications. Choose TDataModule if you want to use nonvisual components in your server class. Choose TComponent if you want to entirely implement the server class.

The fifth step asks you to select the root location for the REST web application. This is the output directory of the project executable and the location of the web application files such as .js, .html and .css files.

DataSnap Mobile Connectors

DataSnap Mobile Connectors are a framework that you can use to access the services on a DataSnap REST Server, enabling the invocation of server methods in much the same way that the other DataSnap proxy generators do, it was introduced in Delphi XE2.

The following mobile platforms are currently supported by the various DataSnap Connectors:

- Android 2.1 (Java)
- BlackBerry Java SDK 5 and 6 (Java)
- Windows Phone 7 (C#Silverlight - all devices compatible with the platform)
- iOS 4.2 (Objective-C)

DataSnap Connectors provide a new Proxy Dispatcher that works together with the pre-existing proxy generator, metadata provider, and server class components. This proxy dispatcher provides functionality which allows for developers on remote machines to generate a proxy on the server and download it onto the developer's machine. This new proxy dispatching service is introduced because this mobile development will not be done within RAD Studio.

To begin mobile development with DataSnap, the first step is to properly set up your DataSnap Server for proxy dispatching, so that you can generate the mobile platform's proxy. For example, when developing an Android application, you need the DSProxy.java file which contains the code generated by the server. You also need the com.embarcadero.javaandroid package, which contains the additional static files required by the proxy. These static files are mostly the data types, such as TStream, that correspond to Delphi data types supported by DataSnap.

The DataSnap wizards have a Mobile Connectors checkbox, which when checked will generated the project and include the Connector components.

After setup you server and implemented the server methods, you can download the proxy using your server url or you can generate through command line. Every time you change the signature or add new server methods you will need to regenerate the proxy classes.

In case you generate the proxy for Java, you will get the DSProxy.java file which contains the code generated by the server, and the package com.embarcadero.javaandroid which contains the additional static files required by the proxy.

The code below shows the DSProxy.java file generated by the server. The code contains a TServerMethods1 class, which encapsulates the server methods. Our server has only one method (EchoString), and the Java Class TServerMethods1 implements it accordingly. The proxy generator has automatically generated all the Java code for connecting to the server and executing the methods:

```
public class DSProxy {
  public static class TServerMethods1 extends DSAdmin {
    public TServerMethods1(DSRESTConnection Connection) {
      super(Connection);
    }

    private DSRESTParameterMetaData[] TServerMethods1_EchoString_Metadata;
    private  DSRESTParameterMetaData[]  get_TServerMethods1_EchoString_Metadata()
{
      if (TServerMethods1_EchoString_Metadata == null) {
        TServerMethods1_EchoString_Metadata = new DSRESTParameterMetaData[]{
          new DSRESTParameterMetaData("Value", DSRESTParamDirection.Input,
 DBXDataTypes.WideStringType, "string"),
          new DSRESTParameterMetaData("", DSRESTParamDirection.ReturnValue,
DBXDataTypes.WideStringType, "string"),
        };
      }
      return TServerMethods1_EchoString_Metadata;
    }

    /**
     * @param Value [in] - Type on server: string
     * @return result - Type on server: string
     */
    public String EchoString(String Value) throws DBXException {
      DSRESTCommand cmd = getConnection().CreateCommand();
      cmd.setRequestType(DSHTTPRequestType.GET);
      cmd.setText("TServerMethods1.EchoString");
      cmd.prepare(get_TServerMethods1_EchoString_Metadata());
      cmd.getParameter(0).getValue().SetAsString(Value);
      getConnection().execute(cmd);
      return cmd.getParameter(1).getValue().GetAsString();
    }


    private DSRESTParameterMetaData[] TServerMethods1_ReverseString_Metadata;
    private                                                  DSRESTParameterMetaData[]
get_TServerMethods1_ReverseString_Metadata() {
      if (TServerMethods1_ReverseString_Metadata == null) {
        TServerMethods1_ReverseString_Metadata = new DSRESTParameterMetaData[]{
          new DSRESTParameterMetaData("Value", DSRESTParamDirection.Input,
                                               DBXDataTypes.WideStringType,
"string"),
          new DSRESTParameterMetaData("", DSRESTParamDirection.ReturnValue,
DBXDataTypes.WideStringType, "string"),
        };
      }
      return TServerMethods1_ReverseString_Metadata;
    }
}
```

The RAD Studio documentation provides a very good material which will guide you through the steps of how to use the Connectors and is available at: http://docwiki.embarcadero.com/RADStudio/en/Getting_Started_with_DataSnap_Mobile_Connectors

DATASNAP CLIENT – DBEXPRESS

Since Delphi 2007 - when the dbExpress Framework was first created - our goal was to build an infrastructure for many technologies, databases and platforms, extending the existing multi-layer support and enabling Java, .NET, PHP, and other clients to connect to DataSnap servers.

DbxClient is now being used to connect the client to the DataSnap server, as well. This means that in order to connect to a DataSnap server you must use an SQLConnection, informing that the connection driver is DataSnap and providing the hostname (server) and port

There are many ways to execute the server methods. Let's assume the server holds a class that contains the HelloWorld and GetEmployee methods.

```
function TDMServerDB.HelloWorld(IncommingMessage : WideString ): WideString;
begin
  Result := 'Hello World';
```

```
end;

function TDMServerDB.GetEmployee(ID : Integer): TDBXReader;
begin
  SQLDataSet1.Close;
  SQLDataSet1.Params[0].AsInteger := ID;
  SQLDataSet1.Open;
  Result := TDBXDataSetReader.Create(SQLDataSet1, False);
end;
```

The HelloWorld method is quite simple. It sends a string and returns another. Method GetEmployee, in turn, sends an ID and receives a TDBXReader. In other words, it's a cursor that can be read client-side either as a DBXReader or as a ClientDataSet.

Still working with the HelloWorld method, you'll now execute it using the SQLServerMethod component. See below:

```
begin
  DMDataSnapClient.DSServerConnect.Open;  //  opens  the  connection  by  means  of
SQLConnection

  SMHelloWorld.SQLConnection := DMDataSnapClient.DSServerConnect;
  SMHelloWorld.Params[0].AsString := 'Message sent from Client';
  SMHelloWorld.ExecuteMethod;

  ShowMessage(SMHelloWorld.Params[1].AsString); // shows the result
End;
```

Let's look at the GetEmployee method. In this example the server method is executed through the dbExpress Framework. The same method could be executed through SqlServerMethod.

```
 1 var
 2  Command : TDBXCommand;
 3  Reader  : TDBXReader;
 4 begin
 5  DMDataSnapClient.DSServerConnect.Open;
 6  With DMDataSnapClient.DSServerConnect.DBXConnection do begin
 7
 8   Command := DMDataSnapClient.DSServerConnect.DBXConnection.CreateCommand;
 9    Command.CommandType := TDBXCommandTypes.DSServerMethod;
10    Command.Text := TDSAdminMethods.GetServerMethodParameters;
11    Reader := Command.ExecuteQuery;
12
13    TDBXDataSetReader.CopyReaderToDataSet(Reader, ClientDataSet1);
14    ClientDataSet1.Open;
15 end;
```

Note that the execution is performed by TDBXCommand. The return of type DBXReader is copied to a ClientDataSet in line 13, thus allowing data to be visualized in the VCL.

In cases where you don't need to display the data, DBXReader can be read directly.

You might be wondering… If this is all dynamic now, wouldn't the compiler be able to detect when server methods change? The answer could be positive in case server methods were not represented by means of a client interface. SQLConnection includes an option called "Generete DataSnap Client Access", that generates a client-side unit with all the methods available at server-side. Each method in the client class contains an implementation to execute the server method.

See a sample class below:

```
  TDSServerMethodsClient = class
  private
    FDBXConnection: TDBXConnection;
    FGetServerDateTimeCommand: TDBXCommand;
    FExecuteJobCommand: TDBXCommand;
  public
    constructor Create(ADBXConnection: TDBXConnection);
    destructor Destroy; override;
    function GetServerDateTime: TDateTime;
    function ExecuteJob(JobId: Integer): Integer;
  end;
implementation

function TDSServerMethodsClient.GetServerDateTime: TDateTime;
begin
  if FGetServerDateTimeCommand = nil then
  begin
    FGetServerDateTimeCommand := FDBXConnection.CreateCommand;
    FGetServerDateTimeCommand.CommandType := TDBXCommandTypes.DSServerMethod;
    FGetServerDateTimeCommand.Text := 'TDSServerMethods.GetServerDateTime';
    FGetServerDateTimeCommand.Prepare;
  end;
  FGetServerDateTimeCommand.ExecuteUpdate;
  Result := FGetServerDateTimeCommand.Parameters[0].Value.AsDateTime;
end;

function TDSServerMethodsClient.ExecuteJob(JobId: Integer): Integer;
begin
  if FExecuteJobCommand = nil then
  begin
    FExecuteJobCommand := FDBXConnection.CreateCommand;
    FExecuteJobCommand.CommandType := TDBXCommandTypes.DSServerMethod;
    FExecuteJobCommand.Text := 'TDSServerMethods.ExecuteJob';
    FExecuteJobCommand.Prepare;
  end;
  FExecuteJobCommand.Parameters[0].Value.SetInt32(JobId);
  FExecuteJobCommand.ExecuteUpdate;
end;

constructor TDSServerMethodsClient.Create(ADBXConnection: TDBXConnection);
begin
  inherited Create;
  if ADBXConnection = nil then
    raise
      EInvalidOperation.Create('Connection cannot be nil. '
        + 'Make sure the connection has been opened.');
  FDBXConnection := ADBXConnection;
end;

destructor TDSServerMethodsClient.Destroy;
begin
  FreeAndNil(FGetServerDateTimeCommand);
  FreeAndNil(FExecuteJobCommand);
  inherited;
end;
```

How do you access the DataSetProvider from the server, from a remote data module, or from a conventional data module? That's simple: on the client-side you use the new DSProviderConnection component, which is connected to your SQLConnection (DataSnap). The ServerClassName property indicates the class name (DataModule/RDM, usually) where the providers are located in the server. The ClientDataSet then can use the DSProviderConnection as its ProviderName.

The new DataSnap and dbExpress Framework provide greater flexibility, not limited by what's presented in this section. It's possible to dynamically access a list of methods along with their parameters, which allows developers to create components that control user access to the server and define access rights specific to each of the server's methods and classes. You can take a look at a more complete DataSnap application samples by visiting my blog at http://www.andreanolanusse.com/en/tag/datasnap/ and http://www.andreanolanusse.com/en/tag/dbexpress.

DATASNAP – SENDING AND RECEIVING OBJECTS

When you start using or migrating applications to the new DataSnap one of the questions you'll have is "how do I transfer objects between my DataSnap clients and servers"? In DataSnap 2009, only dbExpress data types could be transferred. In DataSnap 2010 you can transfer any kind of object between clients and servers.

DataSnap 2010 adds support for JSON (JavaScript Object Notation), which is a lightweight data-interchange format, easy for humans to read/write and easy for machines to parse and generate. JSON is also independent of the programming language and platform. Let's define the object we would like to transfer between a DataSnap client and server, class TCustomer.

```
unit Customer;

interface

uses
   DBXJSON, DBXJSONReflect, SysUtils;
```

```
type
   TMaritalStatus = (msMarried, msEngaged, msEligible);

TCustomer = class
    private
       FName: string;
       FAge: integer;
       FMaritalStatus: TMaritalStatus;
    public
        property Name: string read FName write FName;
        property Age: integer read FAge write FAge;
        property MaritalStatus: TMaritalStatus read FMaritalStatus write
FMaritalStatus;

        function toString : string;override;
  end;
```

In DataSnap 2010, only objects that descend from TJSONObject can be transferred between client and server without any transformation. If your object does not descend from TJSONObject, then you have to use the TJSONMarshal and TJSONUnMarshal classes to convert those objects. The example below shows how to make this conversion.

```
unit Customer;

  function CustomerToJSON(customer: TCustomer): TJSONValue;
  var
    m: TJSONMarshal;
  begin
    if Assigned(customer) then
    begin
      m := TJSONMarshal.Create(TJSONConverter.Create);
      try
        exit(m.Marshal(customer))
      finally
        m.Free;
      end;
    end
    else
      exit(TJSONNull.Create);
  end;

  function JSONToCustomer(json: TJSONValue): TCustomer;
  var
     unm: TJSONUnMarshal;
  begin
    if json is TJSONNull then
      exit(nil);
    unm := TJSONUnMarshal.Create;
    try
      exit(unm.Unmarshal(json) as TCustomer)
    finally
      unm.Free;
    end;
  end;
```

You don't need to implement two transformation methods for every class. You can implement a generic method for a class that uses simple data types, strings, numbers, boolean. Due to the fact that some support classes can be quite complex and some types are fully supported by the current RTTI runtime converters can be added.

The TCustomer class is now ready to cross between client and server. We just need to implement a Server Method which will return a TJSONValue after the TCustomer transformation.

```
// protected
function TServerMethods.GetCustomer: TCustomer;
begin
  Result := TCustomer.Create;
  Result.Name := 'Pedro';
  Result.Age := 30;
  Result.MaritalStatus := msEligible;
end;

// public
function TServerMethods.GetJSONCustomer(): TJSONValue;
var
  myCustomer: TCustomer;
begin
  myCustomer := GetCustomer;
  Result := CustomerToJSON(myCustomer);
  myCustomer.Free;
end;
```

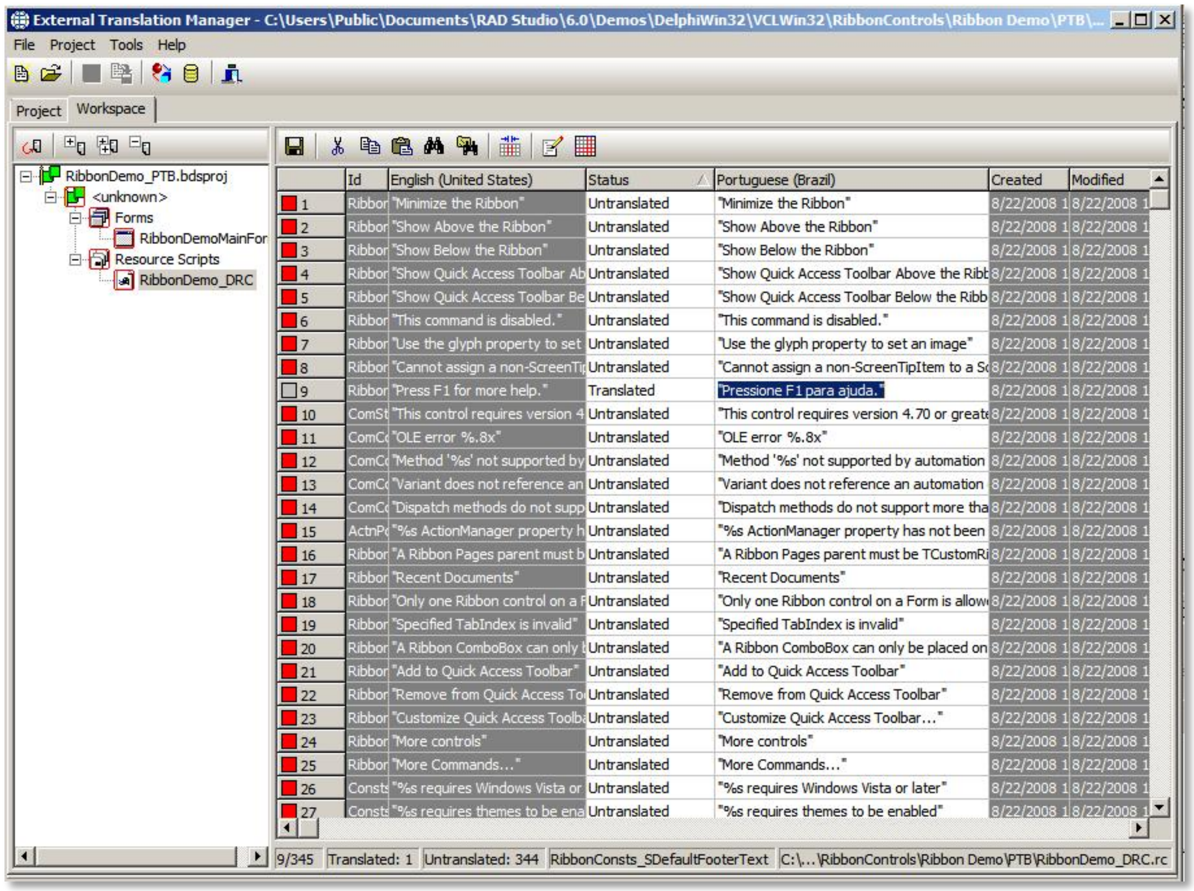
Executing the method GetJSONCustomer from the client side will be necessary to convert the method return from TJSONValue to TCustomer, using the method JSONToCustomer.

```
var
  proxy: TServerMethodsClient;
  myJSONCustomer: TCustomer;
begin

  try
    proxy := TServerMethodsClient.Create(SQLConnection1.DBXConnection);
    myJSONCustomer := JSONToCustomer(proxy.myJSONCustomer);

    Button1.Caption := myJSONCustomer.ToString;
    myJSONCustomer.Free;
  finally
    SQLConnection1.CloneConnection;
    proxy.Free;
  end;
end;
```

Much more can be done, for example return an Array of objects, complex classes, etc. You can download this sample source code at http://cc.embarcadero.com/Item/27361.

DELPHI TRANSLATION TOOLS – LOCALIZING YOUR APPLICATIONS

The IDE-integrated translation tool is now standalone. This means the professional responsible for translating your project can now use the same tool you do, without having to install Delphi himself/herself. For each new language a translation project is generated. It's then much easier to edit language-specific DFM files.



External Translation Manager - C:\Users\Public\Documents\RAD Studio\6.0\Demos\DelphiWin32\VCLWin32\RibbonControls\Ribbon Demo\PTB\...

RibbonDemo_PTB.bdsproj
- <unknown>
  - Forms
    - RibbonDemoMainFor
  - Resource Scripts
    - RibbonDemo_DRC

	Id	English (United States)	Status	Portuguese (Brazil)	Created	Modified
🟥 1	Ribbor	"Minimize the Ribbon"	Untranslated	"Minimize the Ribbon"	8/22/2008 1	8/22/2008 1
🟥 2	Ribbor	"Show Above the Ribbon"	Untranslated	"Show Above the Ribbon"	8/22/2008 1	8/22/2008 1
🟥 3	Ribbor	"Show Below the Ribbon"	Untranslated	"Show Below the Ribbon"	8/22/2008 1	8/22/2008 1
🟥 4	Ribbor	"Show Quick Access Toolbar Ab	Untranslated	"Show Quick Access Toolbar Above the Ribb	8/22/2008 1	8/22/2008 1
🟥 5	Ribbor	"Show Quick Access Toolbar Be	Untranslated	"Show Quick Access Toolbar Below the Ribb	8/22/2008 1	8/22/2008 1
🟥 6	Ribbor	"This command is disabled."	Untranslated	"This command is disabled."	8/22/2008 1	8/22/2008 1
🟥 7	Ribbor	"Use the glyph property to set	Untranslated	"Use the glyph property to set an image"	8/22/2008 1	8/22/2008 1
🟥 8	Ribbor	"Cannot assign a non-ScreenTi	Untranslated	"Cannot assign a non-ScreenTipItem to a Sc	8/22/2008 1	8/22/2008 1
☐ 9	Ribbor	"Press F1 for more help."	Translated	"Pressione F1 para ajuda."	8/22/2008 1	8/22/2008 1
🟥 10	ComSt	"This control requires version 4	Untranslated	"This control requires version 4.70 or greate	8/22/2008 1	8/22/2008 1
🟥 11	ComC	"OLE error %.8x"	Untranslated	"OLE error %.8x"	8/22/2008 1	8/22/2008 1
🟥 12	ComC	"Method '%s' not supported by	Untranslated	"Method '%s' not supported by automation	8/22/2008 1	8/22/2008 1
🟥 13	ComC	"Variant does not reference an	Untranslated	"Variant does not reference an automation	8/22/2008 1	8/22/2008 1
🟥 14	ComC	"Dispatch methods do not supp	Untranslated	"Dispatch methods do not support more tha	8/22/2008 1	8/22/2008 1
🟥 15	ActnP	"%s ActionManager property h	Untranslated	"%s ActionManager property has not been	8/22/2008 1	8/22/2008 1
🟥 16	Ribbor	"A Ribbon Pages parent must b	Untranslated	"A Ribbon Pages parent must be TCustomRi	8/22/2008 1	8/22/2008 1
🟥 17	Ribbor	"Recent Documents"	Untranslated	"Recent Documents"	8/22/2008 1	8/22/2008 1
🟥 18	Ribbor	"Only one Ribbon control on a F	Untranslated	"Only one Ribbon control on a Form is allow	8/22/2008 1	8/22/2008 1
🟥 19	Ribbor	"Specified TabIndex is invalid"	Untranslated	"Specified TabIndex is invalid"	8/22/2008 1	8/22/2008 1
🟥 20	Ribbor	"A Ribbon ComboBox can only	Untranslated	"A Ribbon ComboBox can only be placed on	8/22/2008 1	8/22/2008 1
🟥 21	Ribbor	"Add to Quick Access Toolbar"	Untranslated	"Add to Quick Access Toolbar"	8/22/2008 1	8/22/2008 1
🟥 22	Ribbor	"Remove from Quick Access To	Untranslated	"Remove from Quick Access Toolbar"	8/22/2008 1	8/22/2008 1
🟥 23	Ribbor	"Customize Quick Access Toolb	Untranslated	"Customize Quick Access Toolbar…"	8/22/2008 1	8/22/2008 1
🟥 24	Ribbor	"More controls"	Untranslated	"More controls"	8/22/2008 1	8/22/2008 1
🟥 25	Ribbor	"More Commands…"	Untranslated	"More Commands…"	8/22/2008 1	8/22/2008 1
🟥 26	Consts	"%s requires Windows Vista or	Untranslated	"%s requires Windows Vista or later"	8/22/2008 1	8/22/2008 1
🟥 27	Consts	"%s requires themes to be ena	Untranslated	"%s requires themes to be enabled"	8/22/2008 1	8/22/2008 1

9/345 Translated: 1 Untranslated: 344 RibbonConsts_SDefaultFooterText C:\...\RibbonControls\Ribbon Demo\PTB\RibbonDemo_DRC.rc

Figure 44. Translation Tool

UML MODELING, AUDITS, METRICS, AND DOCUMENTATION

UML MODELING

Any final product is expected to be first-class. Quality concerns are no different when it comes to software (especially considering they're a core element in supporting a company's growth efforts). Keep in mind that whenever you deliver low-quality software you are compromising your client's success.

Since Delphi 2006 you can use UML and all of its diagrams. In addition, you can also use LiveSource, which allows you to synchronize class diagrams and code.

Below you see a list of all diagrams available, along with their functionality:

- Use Case → it is a way to describe the interaction between a system and the real world. In this case, the actors (either persons or systems) represent the real world.
- Class Diagram → represents the classes of the system and the relationships they establish.
- Collaboration → used for modeling the dynamic aspects of a system or subsystem.
- Activity → allows you to represent dynamic situations by means of flows (using it you can represent the flow between different objects).
- Component → used in higher-level modeling, in cases where more complex structures are present. This diagram illustrates systems, embedded controls, etc.
- State → specifies the sequence of events of a given object.

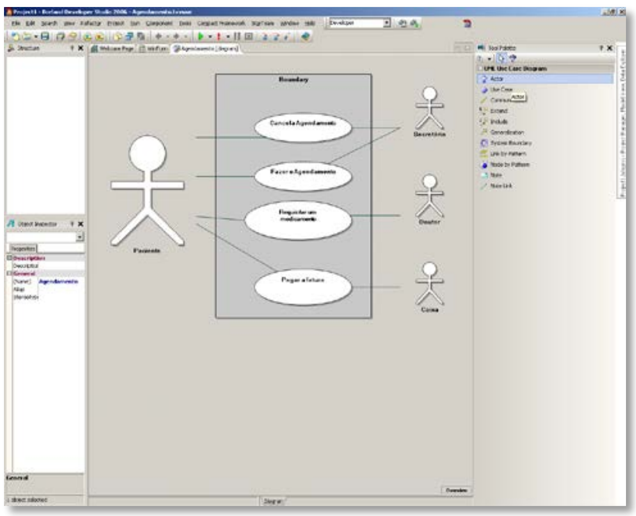


Figure 45. Use Case Diagram

Visualizing a class diagram makes it much easier for you to understand the classes in it (as compared with doing the same with code).

Let's see an example in Delphi: the Buttons.pas unit has many components - TBitBtn, TSpeedButton, among others. Now imagine how hard it would be to decipher 1946 lines of code to learn which components are there and which relationships they established. Using reverse-engineering it's not a big deal...

Let's check the following figure:

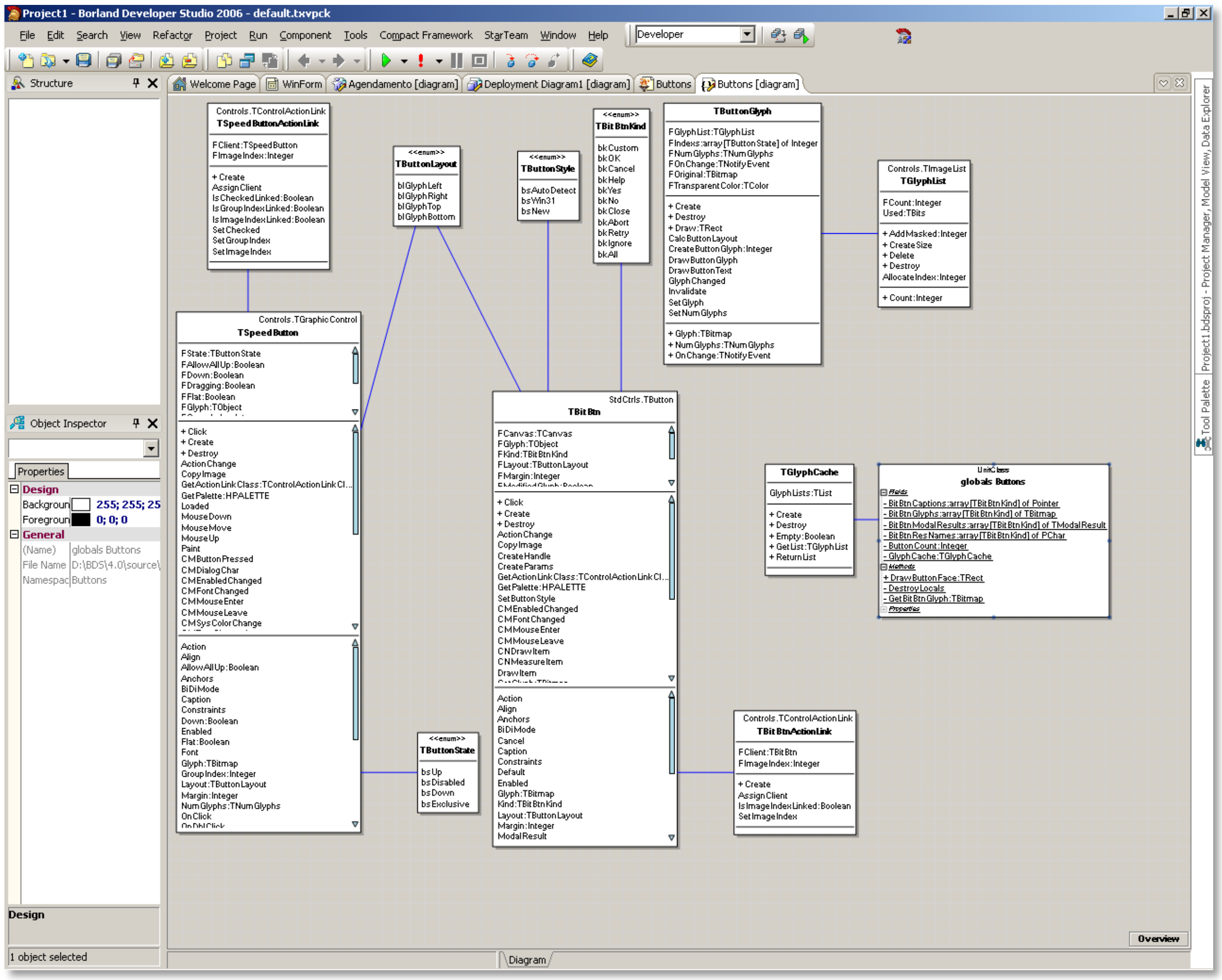


Figure 46. Class Diagram

The Sequence Diagram can be generated directly from the source code. Sequence diagrams detail how methods are implemented in the source code: what messages are sent to which objects and when. Sequence diagrams are organized according to time. The objects involved in the method are listed from left to right according to when they take part in the message sequence.

Delphi has full reverse engineering capabilities. Embarcadero is committed to helping developers evolve legacy code; a commitment that is reinforced by our efforts toward continuously providing technology that allows applications to move on.

AUDITS

When it comes to quality, people are always concerned about delivering high-performance software. Many even tell us they do not care about the way software is written, rather with its ability to work and meet their needs. In fact, this is a mistake with the potential to impact you in the not-so-distant future. When you write unstructured code you're impairing your ability to extend the application in the future. It's then all pieced together, and your application does not grow in a structured manner. Delphi's audits and metrics help you locate flaws in your application while you're still developing it.

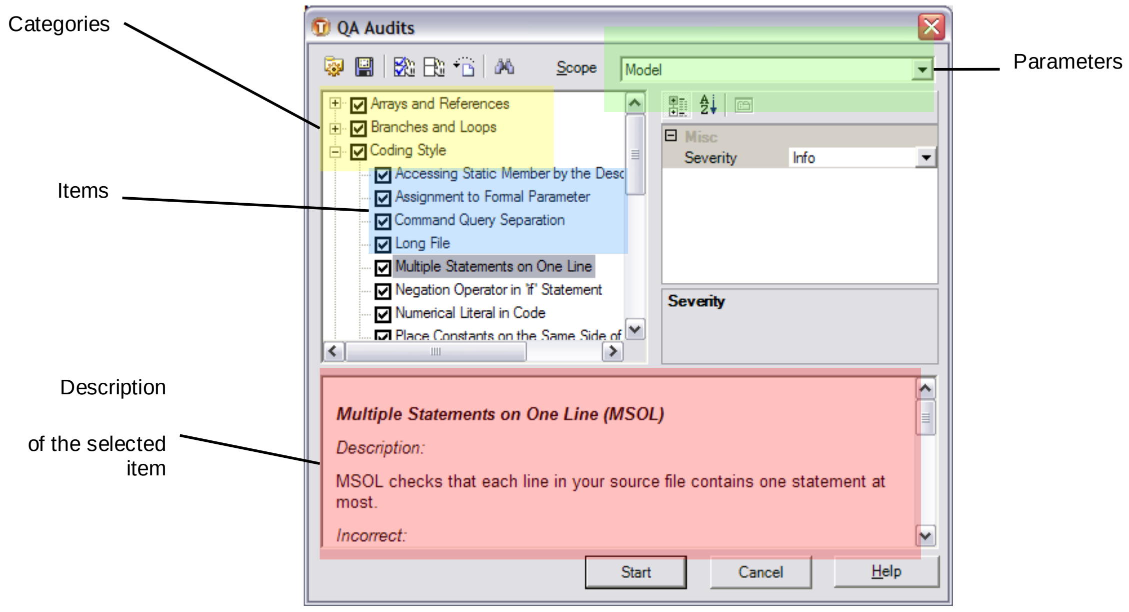
How many times have you defined best-practice guidelines for coding, hoping it would prevent your staff from making the kind of programming mistakes that turns your application into code that no one understands?

Assuming your team uses a best-practices manual, the second question that comes to mind is: how can you ensure that the practices are being followed?

The answer, again: code review. Now, think of a context where your project is coded in no less than thousands of lines. What you have here is a scheduled catastrophe.

Using Delphi's code audits (QA Audits), you can select from a group of best coding practices, making sure your team is following them. Audits can help you detect flaws in your code throughout the development process.

- Arrays and References
- Duplicated code
- Superfluous content
- Performance
- Branches and Loops
- Coding style
- Naming style
- Expressions
- Design flaws
- Possible Errors



Multiple Statements on One Line (MSOL)

Description:

MSOL checks that each line in your source file contains one statement at most.

Incorrect:

Figure 47. QA Audits

Each of the audits includes a descriptive note explaining the correct and incorrect ways of using it helping developers understand how to use each audit. You can set the audit severity level set to Info, Warning, or Error.

LOOP BODY IS NEVER EXECUTED (LBNE)

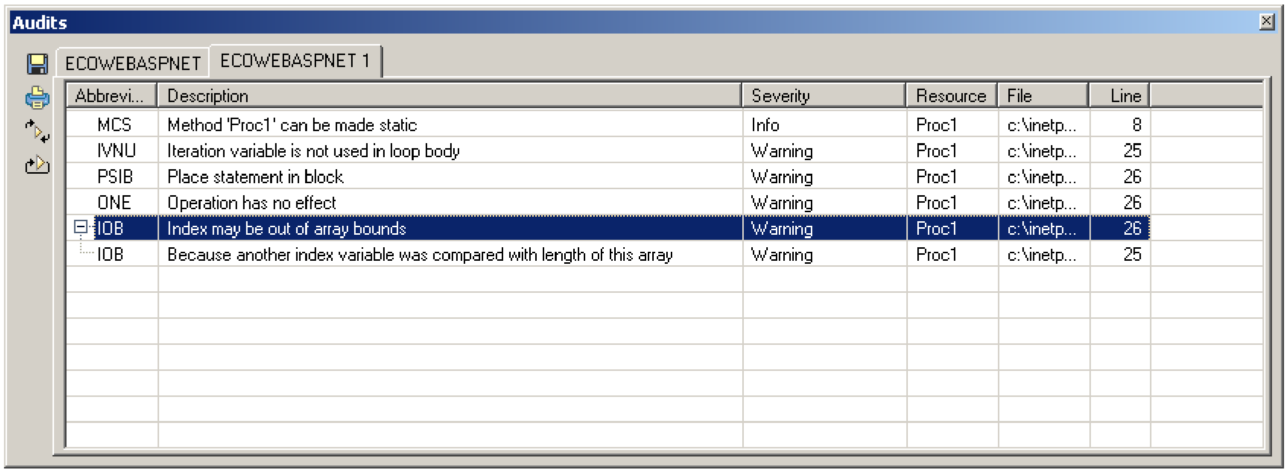
You often see routines that involve many loops, requiring you to us a debugger to make sure all of the loops are executed. The LBNE audit helps detect this type of coding style. The following code is a simple example of something LBNE would detect. Complex examples, which involve more conditions, are also easily detected.

```
var
  x: boolean;
begin
  x := false;
  while s do
  begin
     ....
  end;
end;
```

Index Out of Bounds (IOB)

This is a common message for when you try to access an array position that does not exist. The snippet below generates this warning.

```
var
  nloops,
  i,
  j :integer;
  matriz : array of integer;
  somatorio : double;
begin
  for i := 0 to nloops do
  begin
      somatorio := 0;
      for j := 0 to High(matriz) do
          somatorio := somatorio + matriz[i];

  end;
end;
```



Abbrevi...	Description	Severity	Resource	File	Line
MCS	Method 'Proc1' can be made static	Info	Proc1	c:\inetp...	8
IVNU	Iteration variable is not used in loop body	Warning	Proc1	c:\inetp...	25
PSIB	Place statement in block	Warning	Proc1	c:\inetp...	26
ONE	Operation has no effect	Warning	Proc1	c:\inetp...	26
IOB	Index may be out of array bounds	Warning	Proc1	c:\inetp...	26
IOB	Because another index variable was compared with length of this array	Warning	Proc1	c:\inetp...	25

Figure 48. Audits

The audit has located the error, informing you that the code in line 25 tries to access a variable that is not part of the 2nd loop. Drilling down a little:

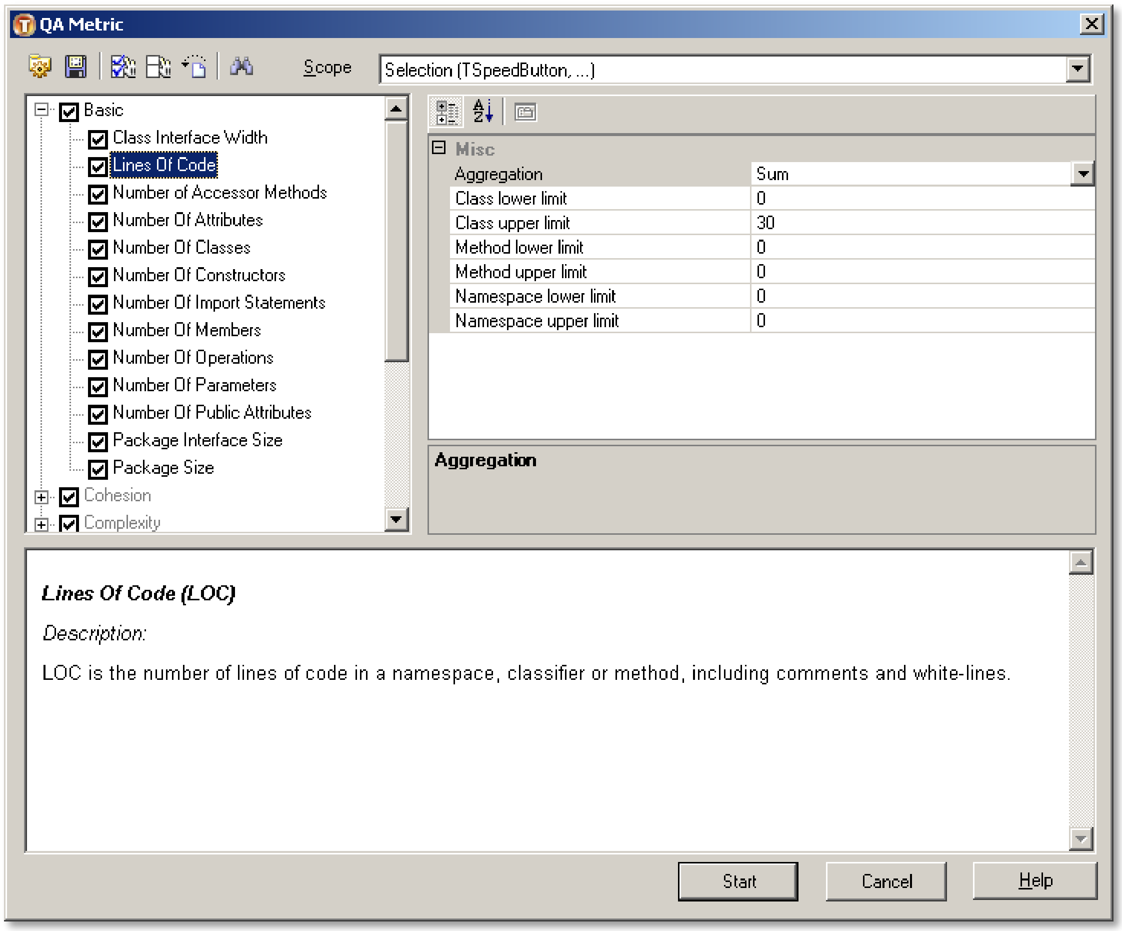
In variable J's 'for' statement I'm trying to access one of the positions in array ARR, aiming at position I of the previous loop, while variable J's 'for' statement is the place where array ARR is being read.

Delphi XE allows you to execute audits through command line, making easy the integration with automated build process.

METRICS

Metrics help identify coding styles that might be violating defined best practices in object oriented design and programming. Who hasn't come across code that has 10 constructors for a class, if statements nested 10 deep, methods with 20 parameters, and other practices that make code incomprehensible? Metrics can help you define standards, best practices and limits for your company to follow. An example: a class must not have more than 4 constructors and having no more than 400 lines of code.

Each metric has defined lower and upper limits for a class, method and namespace. Each limit can be customized:



Aggregation	Sum
Class lower limit	0
Class upper limit	30
Method lower limit	0
Method upper limit	0
Namespace lower limit	0
Namespace upper limit	0

Aggregation

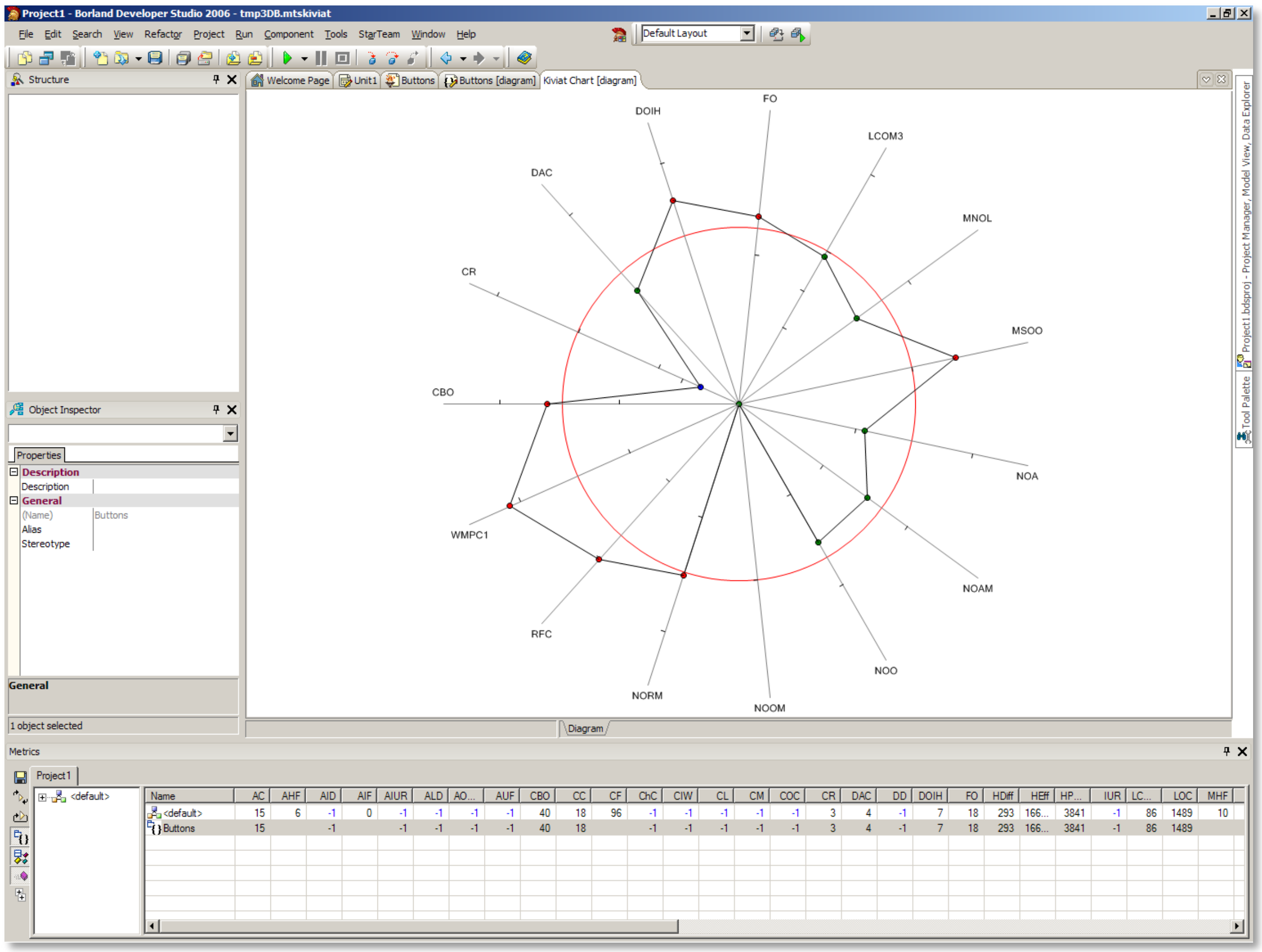
Lines Of Code (LOC)

Description:

LOC is the number of lines of code in a namespace, classifier or method, including comments and white-lines.

Figure 49. QA Metrics

After metrics are executed, their results can be analyzed with the assistance of the Kiviat chart. In a Kiviat chart, the red circle represents the predefined limits for the metrics. Points outside of this boundary denote code that is breaking one or more metric rules.



Description	
Description	
General	
(Name)	Buttons
Alias	
Stereotype	

General

1 object selected

Name	AC	AHF	AID	AIF	AIUR	ALD	AO...	AUF	CBO	CC	CF	ChC	CIW	CL	CM	COC	CR	DAC	DD	DOIH	FO	HDiff	HEff	HP...	IUR	LC...	LOC	MHF
<default>	15	6	-1	0	-1	-1	-1	-1	40	18	96	-1	-1	-1	-1	-1	3	4	-1	7	18	293	166...	3841	-1	86	1489	10
Buttons	15		-1		-1	-1	-1	-1	40	18		-1	-1	-1	-1	-1	3	4	-1	7	18	293	166...	3841	-1	86	1489	

Figure 50. Metrics Analysis

You can analyze each of the classes separately. This way it's easier to identify any metrics violations. Using audits and metrics, developers are able to deliver higher-quality applications, just as good externally as they are in their core.

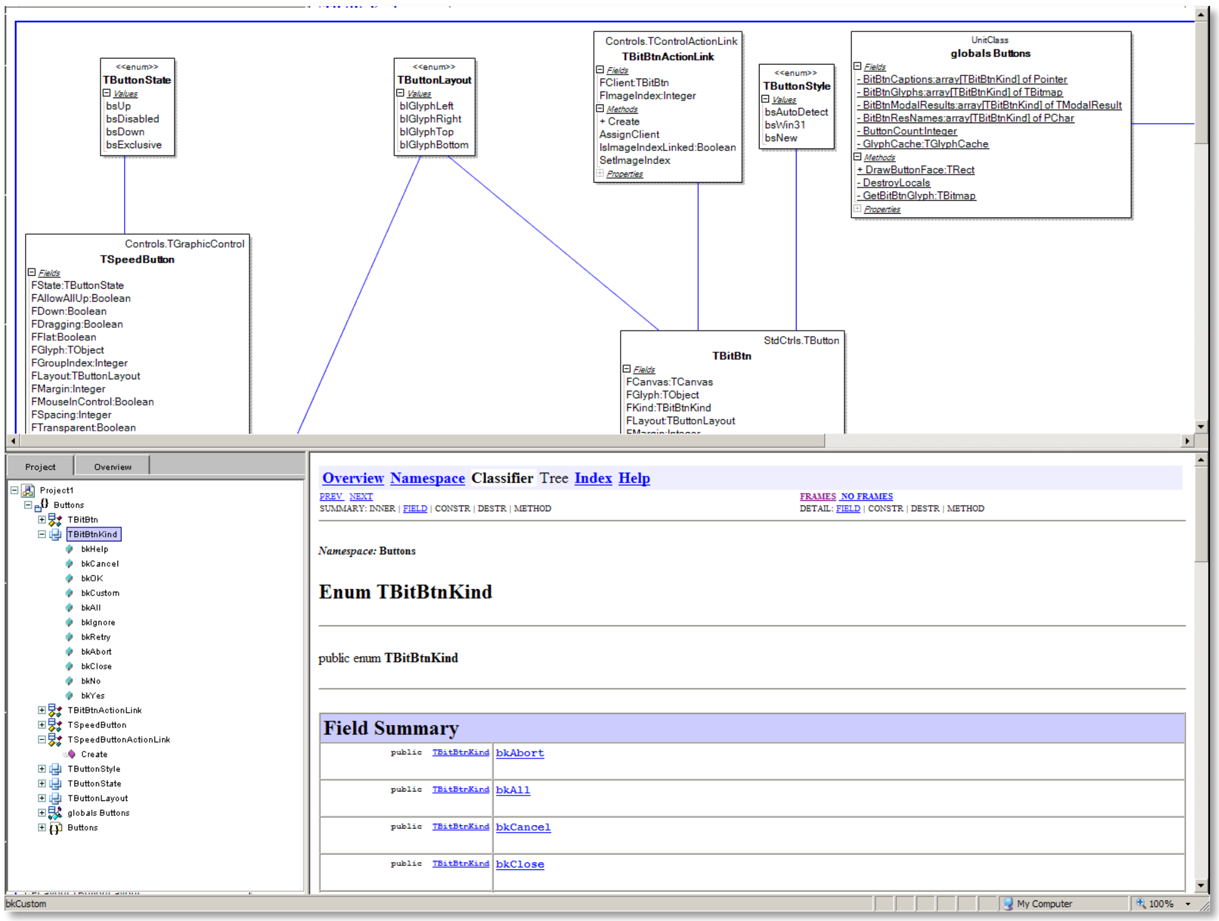
Like audits, you can execute the metrics using the command line as well.

These are the things you are able to perform when your code is migrated from Delphi 7 to the most recent Delphi release.

DOCUMENTATION

Few things are as difficult as getting developers to document their applications. Developers develop, that's what they do best. With the assistance of Delphi's diagrams, developers, analysts, and architects learn how easy it can be to write code and document the entire application. It's as simple as getting into the diagram and documenting. Taking a class diagram, as an example, you simply click on the class, variable, method, and other class attributes, then right mouse click and choose the Generate Documentation menu item. You can also generate documentation for the model from the Project Manager and from the command line.

The documentation is generated as HTML, separated as project overview, diagram view and documentation details.



Overview **Namespace** **Classifier** Tree **Index** **Help**

PREV NEXT FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | DESTR | METHOD DETAIL: FIELD | CONSTR | DESTR | METHOD

Namespace: **Buttons**

Enum TBitBtnKind

public enum **TBitBtnKind**

Field Summary		
public	TBitBtnKind	bkAbort
public	TBitBtnKind	bkAll
public	TBitBtnKind	bkCancel
public	TBitBtnKind	bkClose

Figure 51. Thorough Documentation

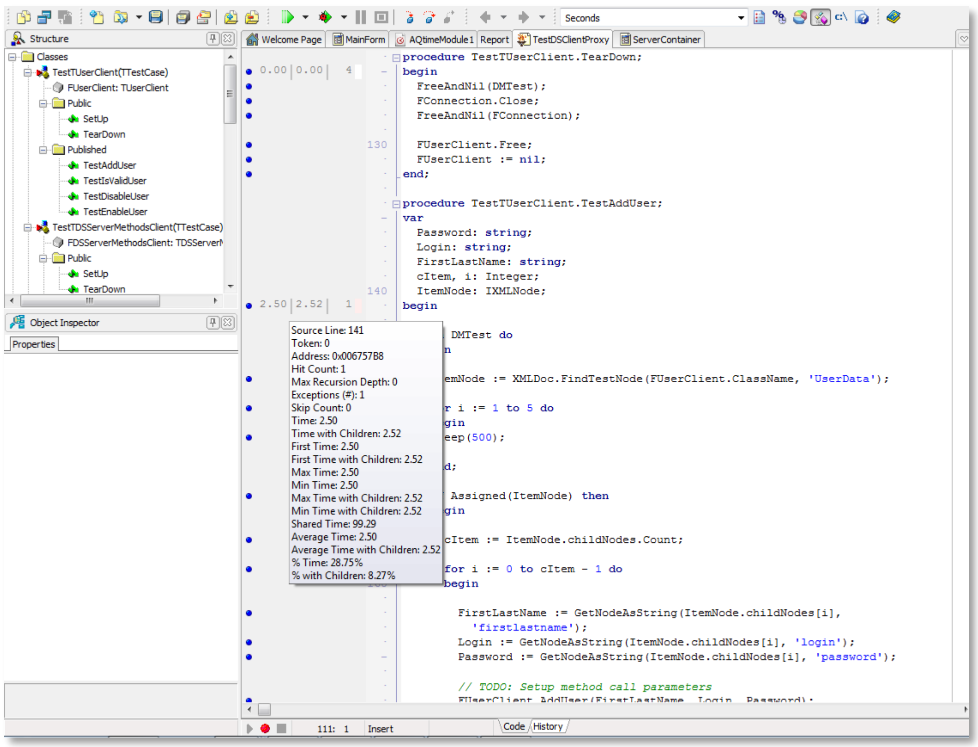
THIRD-PARTY TOOLS AND COMPONENTS

Many of the third-party tools and components packaged with Delphi have been updated and bring new resources and compatibility with existing applications. If you have any additional questions about specific third-party tools and components that you use, the Delphi product page has a list of many third tools and components available today (http://www.embarcadero.com/products/delphi/tools-and-components).

AQTIME – PERFORMANCE PROFILING

AQtime is an award-winning performance profiling, and memory/resource debugger. AQtime standard edition is integrated into the IDE giving you the power to optimize your code without leaving the IDE.

As you optimize and improve your code, AQtime provides an exact and accurate "picture" of your application's execution. Using AQtime you can profile your code to see how long each method takes to execute. You can use AQtime for code coverage analysis. AQtime will also detect memory leaks and resource allocation errors, With AQtime you can eliminate guesswork during development and deliver rock solid software products.



```
0.00|0.00|  4   procedure TestTUserClient.TearDown;
                begin
                  FreeAndNil(DMTest);
                  FConnection.Close;
                  FreeAndNil(FConnection);

                  FUserClient.Free;
                  FUserClient := nil;
                end;

                procedure TestTUserClient.TestAddUser;
                var
                  Password: string;
                  Login: string;
                  FirstLastName: string;
                  cItem, i: Integer;
                  ItemNode: IXMLNode;
2.50|2.52|  1   begin
```

```
Source Line: 141
Token: 0
Address: 0x006757B8
Hit Count: 1
Max Recursion Depth: 0
Exceptions (#): 1
Skip Count: 0
Time: 2.50
Time with Children: 2.52
First Time: 2.50
First Time with Children: 2.52
Max Time: 2.50
Min Time: 2.50
Max Time with Children: 2.52
Min Time with Children: 2.52
Shared Time: 99.29
Average Time: 2.50
Average Time with Children: 2.52
% Time: 28.75%
% with Children: 8.27%
```

```
 DMTest do
n
emNode := XMLDoc.FindTestNode(FUserClient.ClassName, 'UserData');
r i := 1 to 5 do
gin
eep(500);
d;
 Assigned(ItemNode) then
gin
cItem := ItemNode.childNodes.Count;
for i := 0 to cItem - 1 do
begin
  FirstLastName := GetNodeAsString(ItemNode.childNodes[i],
    'firstlastname');
  Login := GetNodeAsString(ItemNode.childNodes[i], 'login');
  Password := GetNodeAsString(ItemNode.childNodes[i], 'password');

  // TODO: Setup method call parameters
```

Figure 52. AQtime results by method

AQtime Pro helps you completely understand how your program performs during execution.

FINALBUILDER – BUILD AUTOMATION

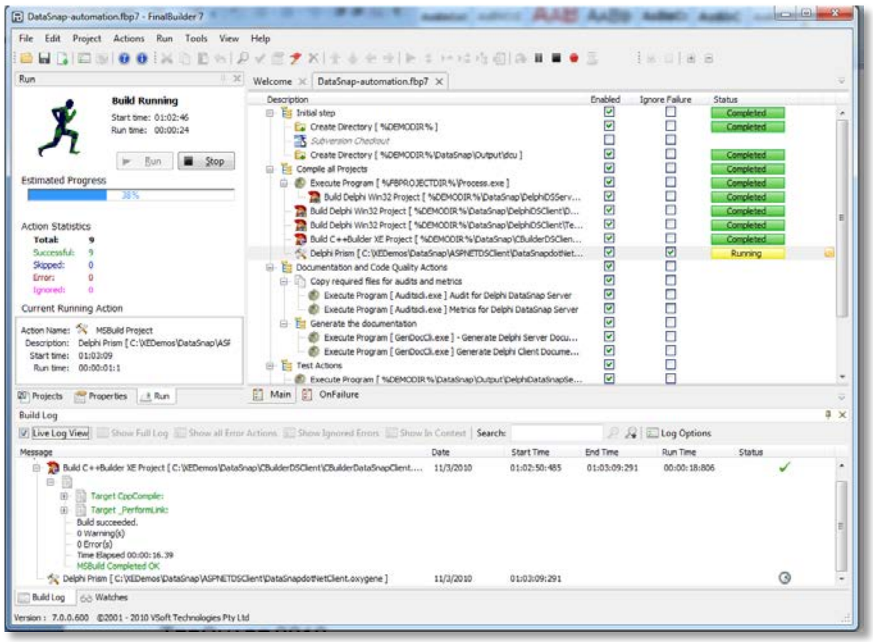
Delphi XE Enterprise and Architect Edition includes FinalBuilder automated build and release management toolset to define and manage repeatable build processes

Traditionally, batch files, XML and script have been used to automate builds. These methods create a build script that is difficult to maintain, difficult to understand and which suffers from a lack of proper error handling.

A visual build tool with the breadth of functionality of FinalBuilder makes it easy to define, debug, maintain and run a reliable build process.

FinalBuilder will:

- Save you time - for any substantial software project, automated builds are much quicker than manual builds.
- Allow anyone in the team to run a build - FinalBuilder is so easy to use, you'll no longer need a single build guru to create, maintain and run your builds.
- Improve the quality of your releases - FinalBuilder cuts the human error factor substantially by automating tasks and running your tests every time your source code is built.
- Record what was built and when - FinalBuilder logs the output from every action it performs and tools it calls. Logs from previous builds are archived and can be exported for better record keeping.



Build Running
Start time: 01:02:46
Run time: 00:00:24

Estimated Progress 38%

Action Statistics

Total:	9
Successful:	9
Skipped:	0
Error:	0
Ignored:	0

Current Running Action

Action Name: MSBuild Project
Description: Delphi Prism [C:\XEDemos\DataSnap\ASF
Start time: 01:03:09
Run time: 00:00:01:1

Description	Enabled	Ignore Failure	Status
Initial step	✓		Completed
Create Directory [%DEMODIR%]	✓		Completed
Subversion Checkout			
Create Directory [%DEMODIR%\DataSnap\Output\dcu]	✓		Completed
Compile all Projects	✓		Completed
Execute Program [%FBPROJECTDIR%\Process.exe]	✓		Completed
Build Delphi Win32 Project [%DEMODIR%\DataSnap\DelphiDSServ...	✓		Completed
Build Delphi Win32 Project [%DEMODIR%\DataSnap\DelphiDSClient\D...	✓		Completed
Build Delphi Win32 Project [%DEMODIR%\DataSnap\DelphiDSClient\Te...	✓		Completed
Build C++Builder XE Project [%DEMODIR%\DataSnap\CBuilderDSClien...	✓		Completed
Delphi Prism [C:\XEDemos\DataSnap\ASPNETDSClient\DataSnapdotNet...	✓	✓	Running
Documentation and Code Quality Actions	✓		
Copy required files for audits and metrics	✓		
Execute Program [Auditsdi.exe] Audit for Delphi DataSnap Server	✓		
Execute Program [Auditsdi.exe] Metrics for Delphi DataSnap Server	✓		
Generate the documentation	✓		
Execute Program [GenDocCli.exe] - Generate Delphi Server Docu...	✓		
Execute Program [GenDocCli.exe] Generate Delphi Client Docume...	✓		
Test Actions	✓		
Execute Program [%DEMODIR%\DataSnap\Output\DelphiDataSnapSe...	✓		

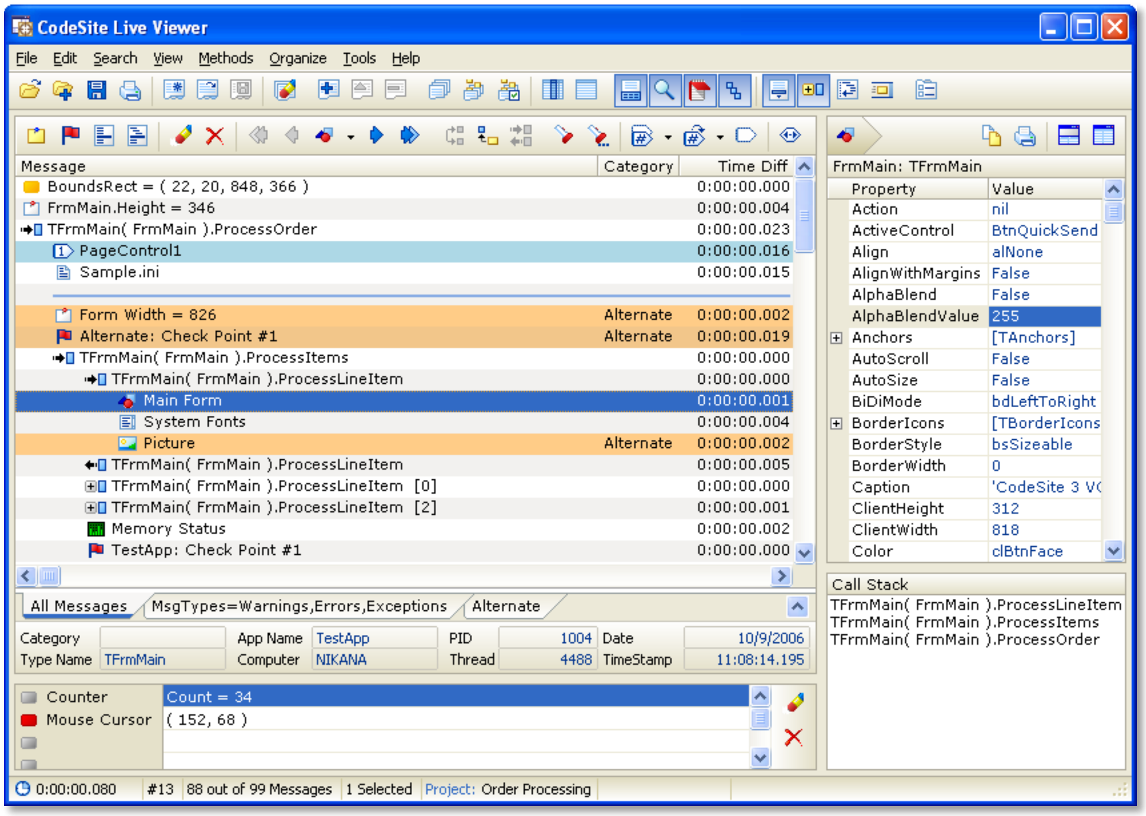
```
Message                                                                              Date       Start Time     End Time       Run Time      Status
Build C++Builder XE Project [ C:\XEDemos\DataSnap\CBuilderDSClient\CBuilderDataSnapClient....  11/3/2010  01:02:50:485   01:03:09:291   00:00:18:806  ✓
    Target CppCompile:
    Target _PerformLink:
    Build succeeded.
    0 Warning(s)
    0 Error(s)
    Time Elapsed 00:00:16.39
    MSBuild Completed OK
Delphi Prism [ C:\XEDemos\DataSnap\ASPNETDSClient\DataSnapdotNetClient.oxygene ]     11/3/2010  01:03:09:291
```

Version : 7.0.0.600 ©2001 - 2010 VSoft Technologies Pty Ltd

Figure 53. FinalBuilder build automation in action

CodeSite – Advanced Logging System

The CodeSite Logging System gives developers deeper insight into how their code is executing. This enables you to locate problems more quickly and ensure that your application is running correctly. CodeSite's logging classes let developers capture all kinds of information while their code executes and send that information to a live display or to a log file.


CodeSite Live Viewer
File Edit Search View Methods Organize Tools Help
All Messages
MsgTypes=Warnings,Errors,Exceptions
Alternate

Message	Category	Time Diff
BoundsRect = (22, 20, 848, 366)		0:00:00.000
FrmMain.Height = 346		0:00:00.004
TFrmMain(FrmMain).ProcessOrder		0:00:00.023
PageControl1		0:00:00.016
Sample.ini		0:00:00.015
Form Width = 826	Alternate	0:00:00.002
Alternate: Check Point #1	Alternate	0:00:00.019
TFrmMain(FrmMain).ProcessItems		0:00:00.000
TFrmMain(FrmMain).ProcessLineItem		0:00:00.000
Main Form		0:00:00.001
System Fonts		0:00:00.004
Picture	Alternate	0:00:00.002
TFrmMain(FrmMain).ProcessLineItem		0:00:00.005
TFrmMain(FrmMain).ProcessLineItem [0]		0:00:00.000
TFrmMain(FrmMain).ProcessLineItem [2]		0:00:00.001
Memory Status		0:00:00.002
TestApp: Check Point #1		0:00:00.000

Category		App Name	TestApp	PID	1004	Date	10/9/2006
Type Name	TFrmMain	Computer	NIKANA	Thread	4488	TimeStamp	11:08:14.195

Counter	Count = 34
Mouse Cursor	(152, 68)

FrmMain: TFrmMain

Property	Value
Action	nil
ActiveControl	BtnQuickSend
Align	alNone
AlignWithMargins	False
AlphaBlend	False
AlphaBlendValue	255
Anchors	[TAnchors]
AutoScroll	False
AutoSize	False
BiDiMode	bdLeftToRight
BorderIcons	[TBorderIcons
BorderStyle	bsSizeable
BorderWidth	0
Caption	'CodeSite 3 V(
ClientHeight	312
ClientWidth	818
Color	clBtnFace

Call Stack
TFrmMain(FrmMain).ProcessLineItem
TFrmMain(FrmMain).ProcessItems
TFrmMain(FrmMain).ProcessOrder

0:00:00.080 | #13 | 88 out of 99 Messages | 1 Selected | Project: Order Processing

Figure 54. CodeSite Live Viewer

IP*Works

IP*Works! is a comprehensive framework for Internet , it eliminates the complexity of Internet development providing easy-to-use, programmable components that facilitate tasks such as sending email, transferring files, managing networks, browsing the web, consuming web services, etc.

TEECHART

One of the components Delphi developers use the most, TeeChart, has been updated to version 2010 and is compatible with VCL and FireMonkey. It now includes new resources aimed at working with Charts in Delphi.

FASTREPORT

Delphi XE2 includes the first-class reporting tool, FastReport, which provides all the necessary tools to develop reports, including a visual report designer, a reporting core, and a preview window.

RAVE REPORTS

Delphi XE2 comes with a second reporting tool, an updated version of Rave Reports..

BEYOND COMPARE

Beyond Compare allows developers to quickly and easily compare files and merge changes. By using simple, powerful commands you can focus on the differences you're interested in and ignore those you're not. Beyond Compare is integrated with Delphi XE IDE.

INTRAWEB

IntraWeb, called VCL for the Web in Delphi 2009 and 2010 - allows you to create Web 2.0 applications, with transparent AJAX integration in many VCL components, Delphi XE2 includes IntraWeb XII.

Delphi XE2 Editions – Professional, Enterprise, Ultimate and Architect

Delphi XE2 is offered in four editions: Professional, Enterprise, Ultimate and Architect. You can check the list of features available in each edition and other product information at: http://www.embarcadero.com/products/delphi/product-editions.

Thew new Delphi XE2 Ultimate edition is designed for software developers and teams building, managing, and tuning database-intensive applications with enterprise database systems. In addition to RAD application development, Delphi Ultimate includes SQL development, database change management, SQL profiling, and SQL tuning tools with DB PowerStudio®. Delphi Ultimate includes everything in the Enterprise edition, plus the following:

- Write high quality SQL faster with Rapid SQL
- Visually tune SQL with DB Optimizer
- Simplify database change management with DB Change Manager

The Architect edition includes ER/Studio Developer Edition data modeling tool. Database modeling is vital for developers working with applications that rely on databases. ER/Studio includes support for reverse-engineering existing database and working with logical and physical database models.

ER/Studio supports the following databases: DB2 LUW V9, Hitachi HiRDB, IBM® DB/2®, Informix, InterBase®, Microsoft® Access, SQL Server, Visual FoxPro, MySQL, NCR Teradata, Oracle, PostgreSQL, Sybase, SQL Anywhere. Other databases can also be accessed through ODBC.

ER/Studio provides resources as reverse-engineering, logical and physical modeling.

Conclusion

Delphi's IDE has been continually improved over the years and Delphi XE2 is the most important new Delphi release since the beginning of Delphi. The new FireMonkey platform reinvents the way that developers create business applications and expands the possibilities for Delphi developers to build applications for Windows 64-bit, Mac and iOS.

The long list of Delphi improvements and new features since Delphi 7 have helped countless developers raise development productivity, quality of applications, and level of integration between native applications and any other platform to share information across multiple devices.

For additional detail drill down into what's new in Delphi XE2, please visit: http://www.embarcadero.com/products/delphi/whats-new.

ABOUT THE AUTHOR

Andreano Lanusse is a software development expert and industry enthusiast. As a technical lead evangelist for Embarcadero worldwide, he spends a great deal of his time with developers, both onsite and at conferences and user groups, to ensure the company's tools meet the expectations of customers. Prior to Embarcadero, Andreano worked at Borland for thirteen years in numerous roles including, support coordinator, engineer, product manager, and product line sales manager. He has also worked as a principal consultant for Borland Consulting Services on the development and management of critical applications. Previously, he served as chief architect for USS Tempo (formerly USS Soluções Gerenciadas). Andreano holds a bachelors degree in business administration from Sumare Institute and MBA in project management from FGV, and Certificate for Product Management Course from University of California, Berkeley. Andreano is also a certified SCRUM Master. You can read more about Andreano Lanusse on his blog, http://www.andreanolanusse.com/ and reach him at his e-mail address: andreano.lanusse@embarcadero.com.