Tech Note

embarcadero

Преимущества перехода на Delphi XE

Что нового по сравнению с Delphi 7

Андреано Лануш (Andreano Lanusse)

Ноябрь 2010 г.

Embarcadero Technologies Россия, СНГ

129343 Россия, Москва, проезд Серебрякова, 6; т. +7(495) 708-43-93

Содержание

ВВЕДЕНИЕ

Многие пользователи Delphi хотят знать основные преимущества перехода на Delphi XE. Это – большое число новых возможностей, обеспечивающих небывалый рост производительности при создании высококлассных приложений. В этой статье представлено обсуждение основных причин перехода на версию XE благодаря новым возможностям, появившимся в Delphi, начиная с версии 7.

Далее приводится краткая справка по самым важным новым функциям с разбивкой по релизам продукта после Delphi 7. А затем в основной части статьи проводится углубленный технический анализ перечисленных, а также других новых возможностей.

Delphi 2005

- Пространства имен для нескольких модулей, цикл for ... in ... do, директива inline для функций и другие возможности оптимизаций кода.
- Доступ к гетерогенным базам данных, многозвенная архитектура приложений
- Рефакторинг, History view для исходного кода
- Модульное тестирование

Delphi 2006

- Code block completion/Surround (сервисы обрамления/завершения кода), Editor Change Bars
- Live Code Templates (автоматизированные шаблоны кода)
- UML-моделирование, аудиты, метрики, генератор документации
- Шаблоны проектирования

Delphi 2007

- MSBuild, Build Configurations
- VCL - AJAX, совместимость с Vista
- темы Vista и XP для приложений
- новый dbExpress framework, delegate drivers («драйверы-делегаты»), поддержка баз данных в формате Unicode

Delphi 2009

- Сквозная поддержка Unicode на уровне языка, библиотеки и среды разработки
- Generics и анонимные методы

- Редактор ресурсов (Resource Editor), обозреватель классов (Class Explorer)
- многозвенная архитектура DataSnap
- новые компоненты VCL (Custom Hints, Ribbon Controls и т.д.)
- локализация: встроенные и внешние сервисы для управления переводом

Delphi 2010

- Поддержка Windows 7, множественного сенсорного ввода и ввода при помощи жестов, Direct-2D
- IDE Insight, Source Code Formatter (сервис форматирования исходного кода), Search task bar
- Фоновая компиляция
- Расширенная RTTI
- Точки останова в потоках, «заморозка/разморозка» потоков
- DataSnap: поддержка протокола HTTP

Delphi XE

- DataSnap: поддержка HTTPS, JavaScript, REST
- Интеграция с Subversion
- Библиотека регулярных выражений
- AQTime, CodeSite, Beyond Compare, Final Builder
- Возможности взаимодействия с облачными сервисами и размещение в облачной инфраструктуре

Очевидно, что миграция проектов на новые версии средств разработки требует времени, которое не всегда есть у разработчика.

Одной из основных причин миграции на Delphi XE, C++ Builder XE или RAD Studio XE является то, что при приобретении последних версий появляется возможность получить предыдущие без дополнительной платы.

Например, при покупке Delphi XE пользователь получает Delphi 7, Delphi 2007, Delphi 2009 и Delphi 2010.

Что нового в IDE

ИНТЕГРАЦИЯ С SUBVERSION – VERSION INSIGHT

Вместе с Delphi XE появляется возможность использовать популярную систему контроля версий Subversion для управления ревизиями исходного кода, как на уровне индивидуального разработчика, так и при групповой разработке.

Данная возможность включает:

• Интеграция в Project Manager и History Manager

• Поддержка основных команд для управления версиями, такими как import, update, commit и show log.

• Сервис отображения различий и выполнения слияний (difference and merge viewer)

• Исходный код для поддержки функции интеграции с использованием Open Tools API в виде проекта с открытым исходным кодом

Указанный проект доступен в исходном коде на сервисе SourceForge:
http://sourceforge.net/projects/radstudioverins/.
Но есть более простой путь получить доступ к нему, если версия XE уже приобретена. В поставку продукта входит исходный код в директорию с примерами RAD Studio (RAD Studio samples): «C:\Users\Public\Documents\RAD Studio\8.0\Samples)» (естественно, путь может быть другим).

Project Manager

Новый Project Manager содержит много новых возможностей, которые помогут значительно повысить производительность:

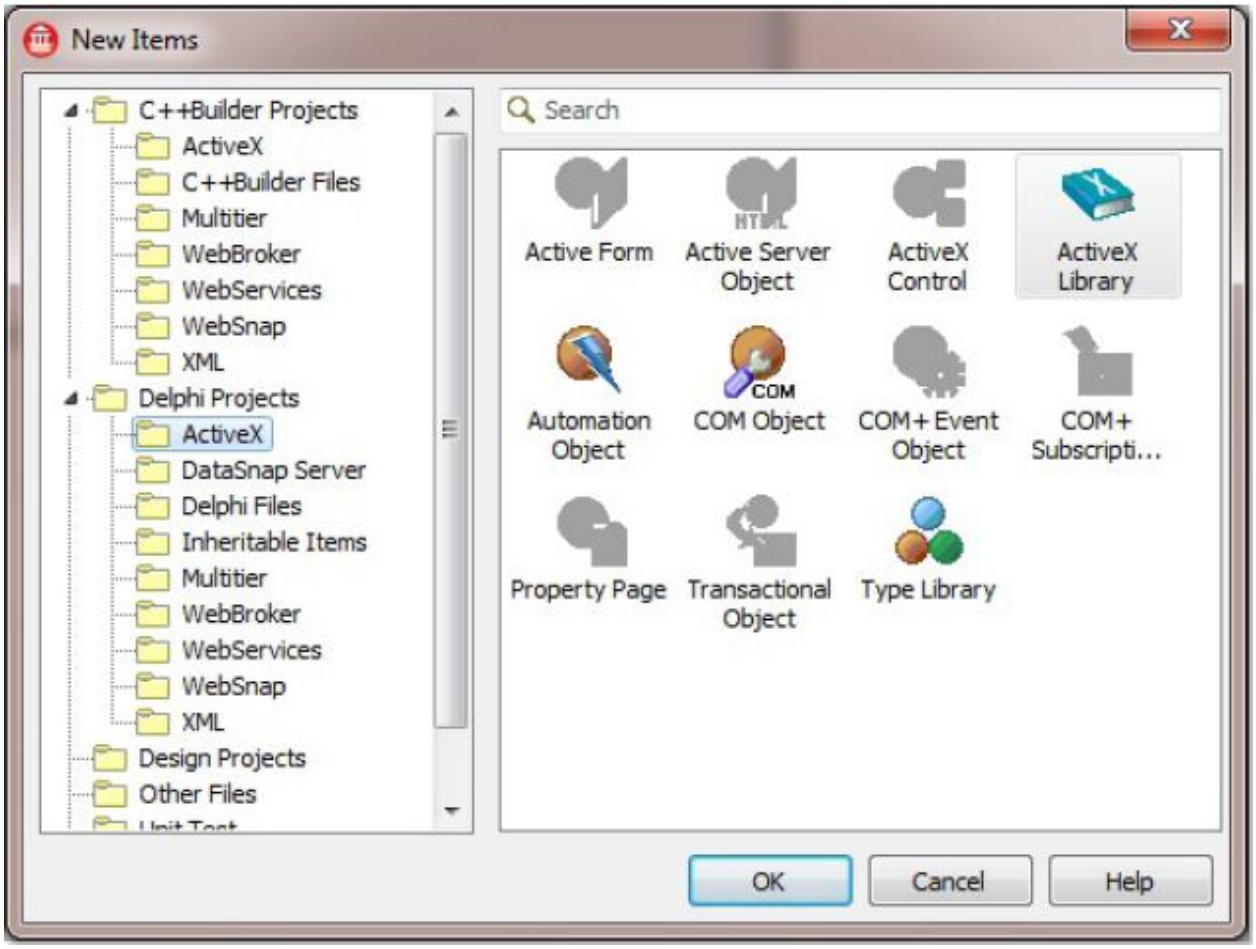
• Сортировка содержимого в Project Manager: новая кнопка на инструментальной панели Sort By дает возможность упорядочивать элементы по имени, дате/времени, директории или типу файла. Также можно задать опцию Auto Sort, которая означает, что новые добавления в проект или в группу проектов будет производиться с учетом заданного параметра сортировки. Рассмотрим еще некоторые функции Project Manager.

- Compile All From Here (компилировать всё с данного места) и Build All From Here (выполнить сборку всего с данного места): новое контекстное меню в Project Manager содержит новую команду, которая позволяет выполнять следующее:
  - Compile All From Here (компилировать всё с данного места)
  - Build All From Here (выполнить сборку всего с данного места)
  - Clean All From Here (очистить всё с данного места)
    эти три команды запускают компиляцию, сборку или операцию очистки, которые начинаются, естественно, с текущего выделенного узла в проекте. Эти команды описаны в контекстном меню проекта.

• Compile All, Build All и Clean All: Эти новые команды контекстного меню доступны для групп проектов, которые содержат более одного проекта.
• Контекстное меню проекта содержит новую команду Install|Uninstall которая позволяет инсталлировать или удалять пакеты design-time.

Галерея (GALLERY)

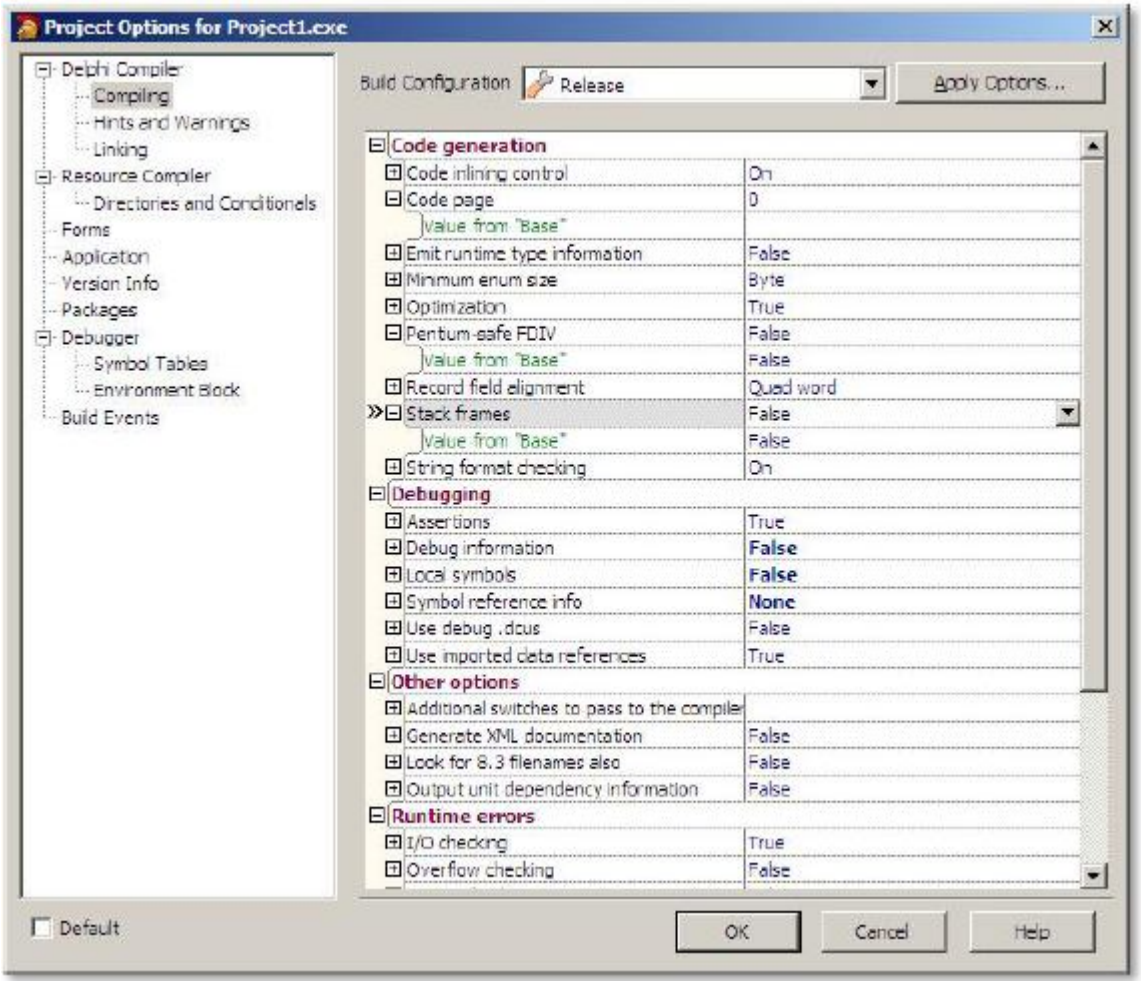
Галерея была расширена возможностью поиска для повышения производительности. Все элементы галереи появляются, но те, которые раньше были невидимыми, теперь просто отображаются серым цветом. Эта возможность реально поможет пользователям, которые переводят свои проекты с Delphi 7, где все мастера COM отображались, но нужно было знать, в каком порядке пользоваться ими. Теперь можно запускать любые мастера, которые доступны, а остальные также отображаются в недоступном виде. Соответственно, вопрос «куда делись остальные мастера» больше не возникает.



Фигура 1. Новая галерея проектов (Project Gallery) со встроенным поиском

Новые опции проекта (PROJECT OPTIONS)

В интегрированную среду разработки было внесено множество изменений, чтобы упростить и повысить скорость разработки. Теперь опции компиляции проекта отображаются в колонках и группируются по категориям в более наглядном виде. Также стало гораздо проще сохранять параметры конфигурации проекта или опции сборки, как показано на Фигуре 2.



Фигура 2. Параметры конфигурации сборки

Параметры конфигурации сборки (BUILD CONFIGURATIONS)

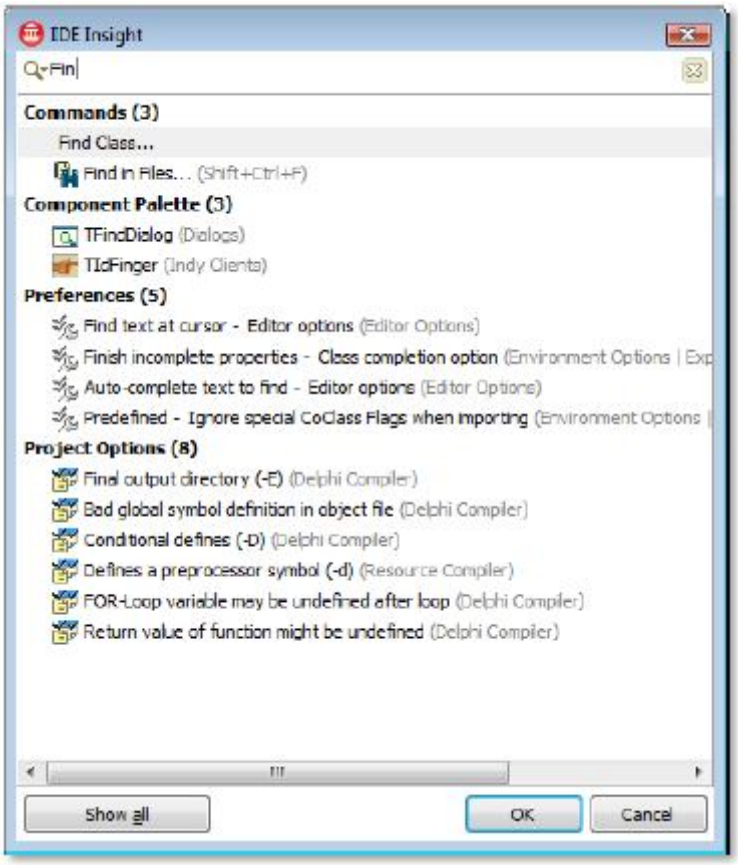
Компиляция и отладка проектов являются типовыми задачами для разработчиков. Однако опции проекта, которые используются для построения финальной версии (релиза) не всегда совпадают с опциями проекта, используемыми при отладке. На постоянную смену опций проекта приходится тратить много времени. Теперь этого можно избежать при использовании нового менеджера проектов (Project Manager). В Delphi XE функции управления параметрами конфигурации сборки встроены непосредственно в него.

Кроме того, параметры конфигурации проекта могут быть сохранены в OPSET-файлах формата XML. Это позволяет многократно использовать именованные опции предыдущего проекта.

IDE INSIGHT

Новый сервис IDE Insight дает возможность вводить какое-либо название и затем выбирать из списка подходящих вариантов опцию проекта или параметр среды разработки. Поле ввода IDE Insight содержит множество категорий, таких как Commands, Files, Components, Project Options и т.д.

При вводе строки поиска сервис IDE Insight выполняет инкрементный поиск: IDE Insight отображает только категории, в которых находятся найденные варианты вместе с наиболее точно совпадающим вариантом в каждой из них. Также можно нажать комбинацию кнопок Alt+A или кнопку в диалоговом окне IDE Insight для переключения между показанными категориями (с одним наилучшим совпадением на каждую категорию) или всеми совпадениями (что может потребовать прокручивание списка для выбора нужного элемента).



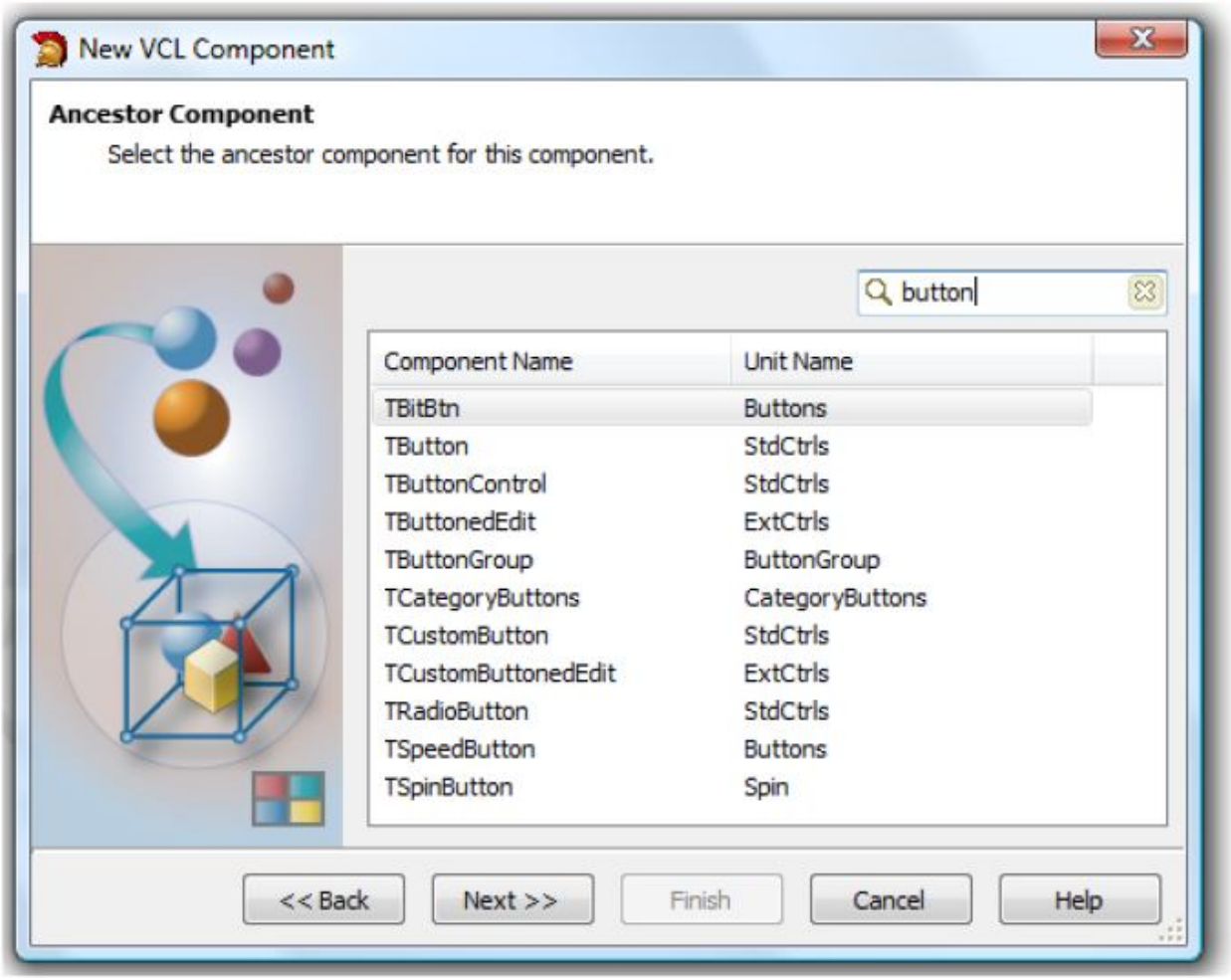
Фигура 3. IDE Insight

При двойном клике на элементе в окне IDE Insight система автоматически вызывает или выполняет ассоциированное действие. Например, если введено слово «open», то в списке показаны все доступные элементы, название которых содержит подстроку «open». Если дважды кликнуть на название, соответствующее определенному диалогу, то он появится на экране. Если два раза кликнуть на названии компонента, содержащем слово «open» (например, TOpenDialog), то компонент автоматически добавится на активную форму.

Мастер создания компонентов (COMPONENT CREATION WIZARD)

Дизайн мастера создания и импорта компонентов был модифицирован для включения библиотеки типов, элементов управления ActiveX и сборок. Оба мастера могут производить инсталляцию в существующий или новый пакет (package).

Как показано на Фигуре 4, добавлено новое поле ввода для фильтрации компонентов, что облегчает поиск класса компонента, выбранного в качестве базового.



Фигура 4. Выбор компонента-предка

COM

Мастера COM и библиотеки типов полностью изменены. Действительно, дизайн мастеров создания COM-объектов полностью изменился.

Рассмотрим, что появилось нового. В архитектуру COM был добавлен новый типа файлов – RIDL (Restricted Interface Definition Language). RIDL-файлы работают как «записывающие устройства» для сохранения библиотек типов проекта. Следовательно, бинарный файл

библиотеки типов (.TLB) становится промежуточным, таким как .DCU, .RES, .OBJ и т.д. Это означает, что разработчики получили возможность перекомпилировать tlb-файлы из командной строки и даже редактировать tlb-файлы с использованием текстового редактора, отслеживая его версии.

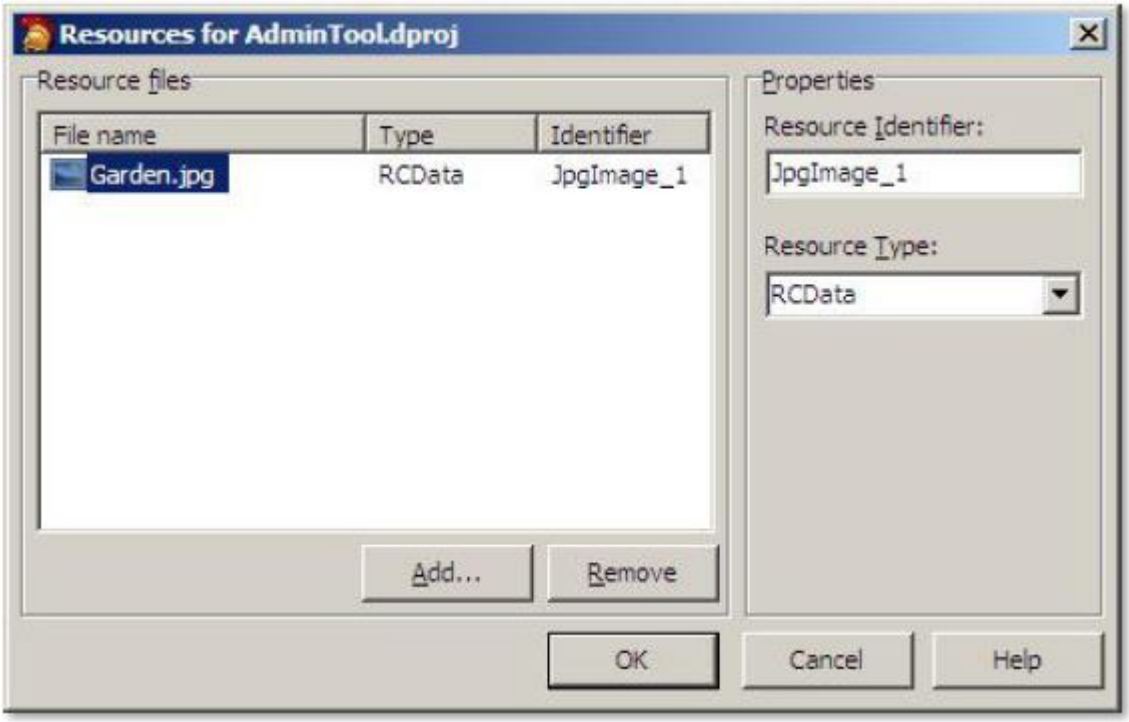
Теперь библиотека типов использует текстовой файл (файл RIDL), а не TLB. Это – очень полезно:

- Нет необходимости перегенерировать tlb-файлы. Теперь они автоматически генерируются на основе последней версии RIDL-файла.
- Различные разработчики могут работать с одной и той же библиотекой типов. Это возможно, т.к. текстовые файлы можно «сливать» (merge), что было невозможно в случае с бинарными файлами, используемыми ранее.
- RIDL-формат для библиотеки типов позволяет использовать более гибкий редактор.
- Можно легко сравнивать разные RIDL-файлы.

Новый менеджер ресурсов (NEW RESOURCE MANAGER)

Компилятор ресурсов позволяет выбрать между методом компиляции при помощи BRCC32.exe или RC.exe (компилятор ресурсов Microsoft Platform SDK). Компилятор ресурсов поддерживает символы Unicode в файлах ресурсов и именах файлов. Также он поддерживает новые типы ресурсов для Windows Vista (например, иконки и альфа-канал). Если используется компилятор ресурсов, нужно явно указать #include <winresrc.h> как для Delphi, так и C++.

Новый менеджер ресурсов позволяет добавлять множество файлов ресурсов (растровые изображения, иконки, шрифты…) в любой проект.



Фигура 5. Редактор ресурсов

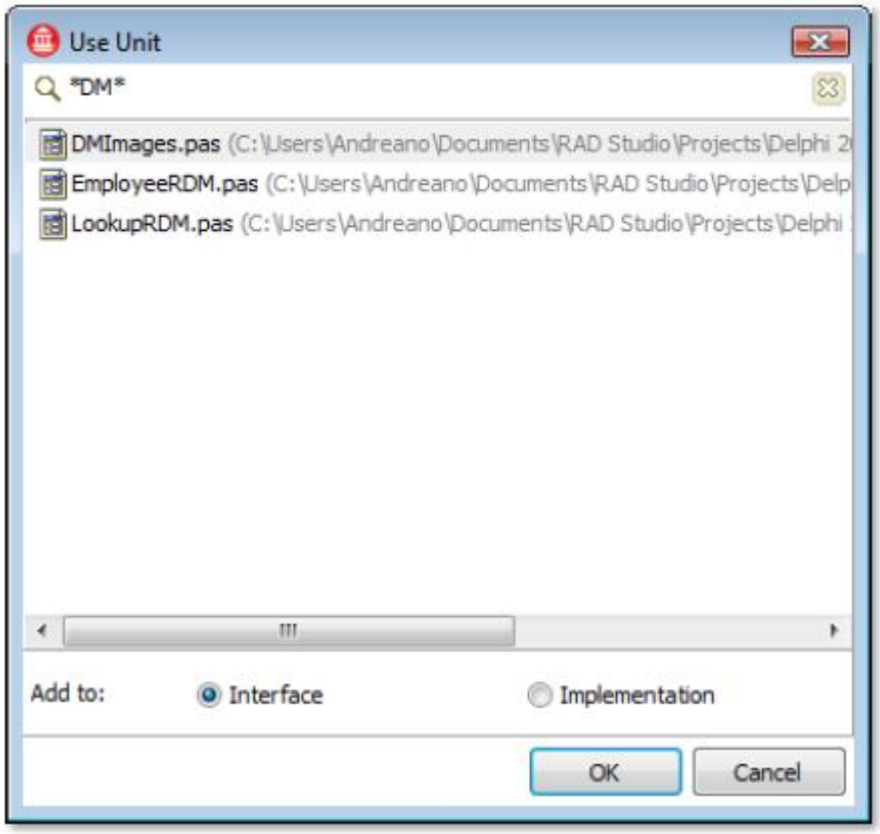
Управление опцией меню REOPEN FILES

Теперь есть возможность управлять количеством файлов и проектов, которые отображаются в меню File→Reopen menu. Можно задать количество проектов и файлов, которые будут появляться в списке, а также очистить этот список в случае необходимости.

USE UNIT – INTERFACE/HEADER

До выхода Delphi 2009 опция Use Unit декларировала модуль в разделе Implementation. Теперь можно выбирать, где декларация будет располагаться, Interface или Implementation для кода Delphi.

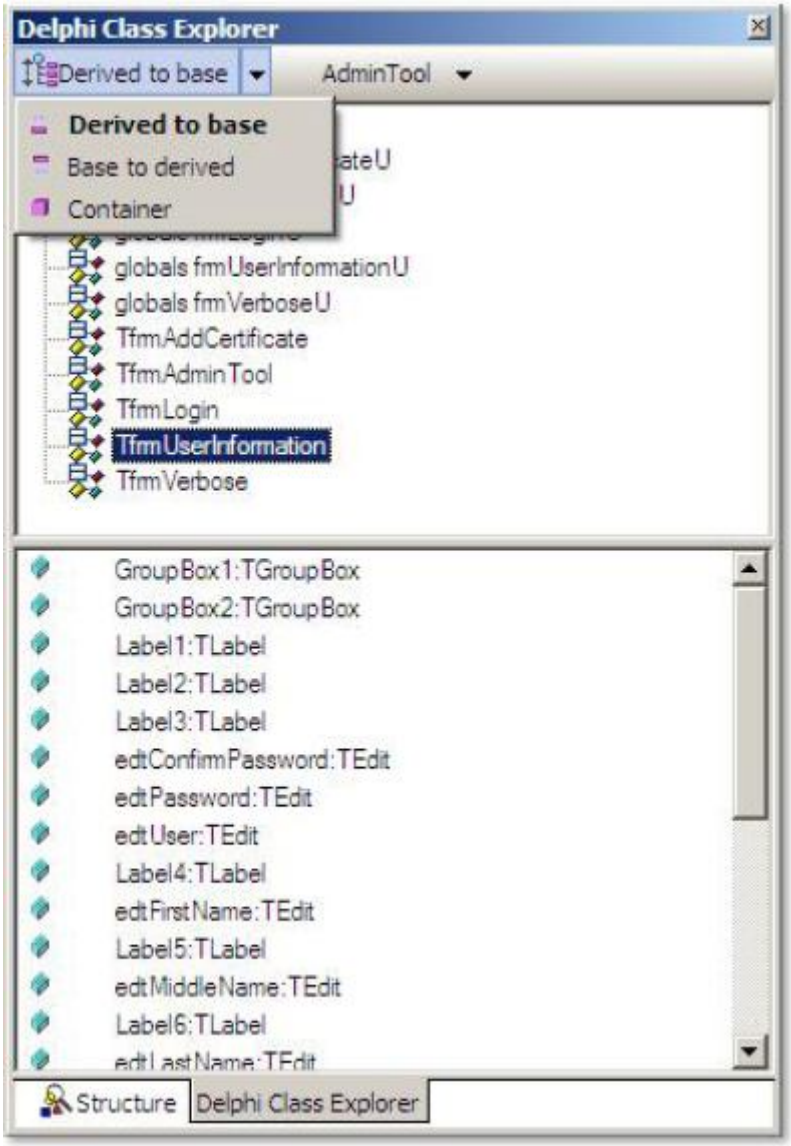
Кроме того, проекты могут включать десятки, сотни или даже тысячи модулей, то их декларирование через опцию Use Unit становится сложно. В этом случае появилась возможность использовать маски для фильтрации модулей и облегчения поиска, как показано на Фигуре 6.



Фигура 6. Новое окно Use Unit

Обозреватель классов (CLASS EXPLORER)

Обозреватель классов является очень мощным инструментом, который позволяет визуализировать иерархию классов и интерфейсов в рамках проекта, также как и добавлять свойства, методы и переменные в них. Эти операции можно выполнить посредством UML с помощью моделей классов. Возможности UML стали неотъемлемой частью Delphi, наряду с многими другими сервисами.



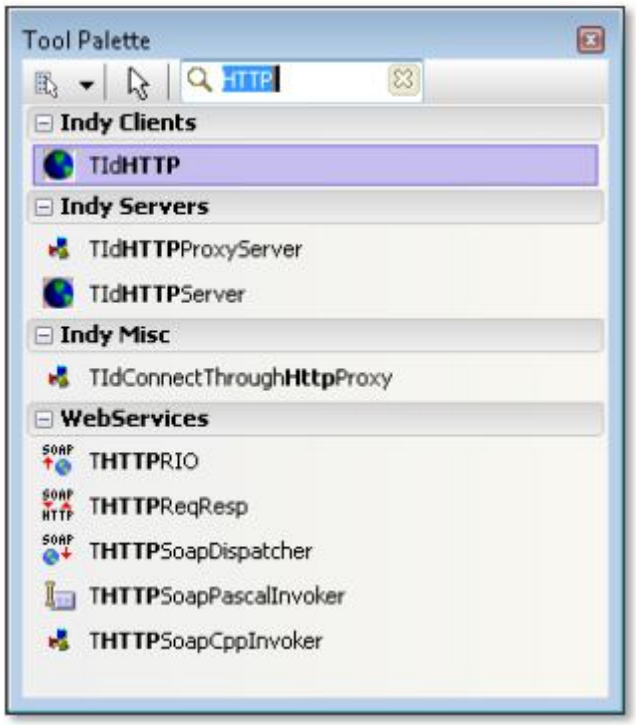
Фигура 7. Обозреватель классов (Class Explorer)

Поиск компонентов в палитре

В Delphi 2006 можно было фильтровать компоненты при вводе первых нескольких символов из названия в палитре компонентов. В Delphi 2007 эта возможность была расширена, и можно было вводить любую часть названия компонента. В Delphi XE используется поле ввода для достижения подобного результата, но более простым и ясным способом.

Пользователи, которые предпочитают использовать модель интерфейса как в Delphi 7 (т.е. компоненты отображаются в верхней части среды разработки), будут рады узнать, что Delphi XE может выглядеть также как и Delphi 7.

Однако, перед переключением режима интерфейса к устаревшему виду Delphi 7, следует попробовать новую палитру инструментов в действии. Наглядное расположение компонентов, упорядоченное размещение по категориям и т.д. обеспечивают более высокую производительность.



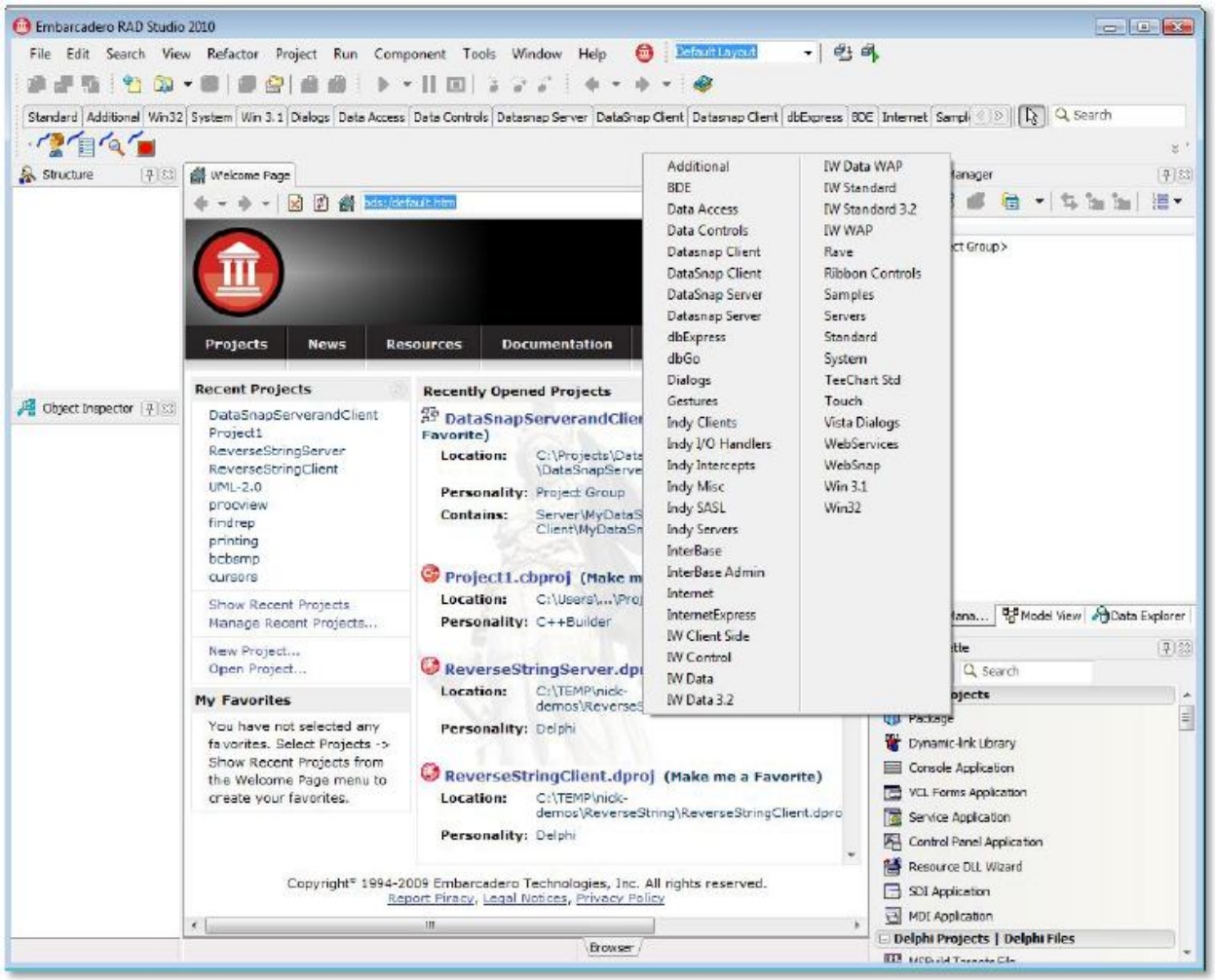
Фигура 8. Поиск компонентов

Классическая панель компонентов

Многие разработчики предпочитают использовать классический вариант панели компонентов как в Delphi 7, что являлось причиной отказа от новых релизов. В Delphi 2010 (и, естественно, последующих версиях) можно пользоваться как классической, так и новой панелью компонентов.

Для активации панели компонентов нужно просто щелкнуть правой кнопкой мыши на главной панели и выбрать Component. После этого можно кликнуть правой кнопкой на инструментальную панель, чтобы получить список всех доступных категорий или просмотреть и выбрать конкретную.

Конфигурации палитры компонентов и панели компонентов независимы, поэтому можно реорганизовать категории. Это также касается поля для поиска компонентов на панели компонентов.

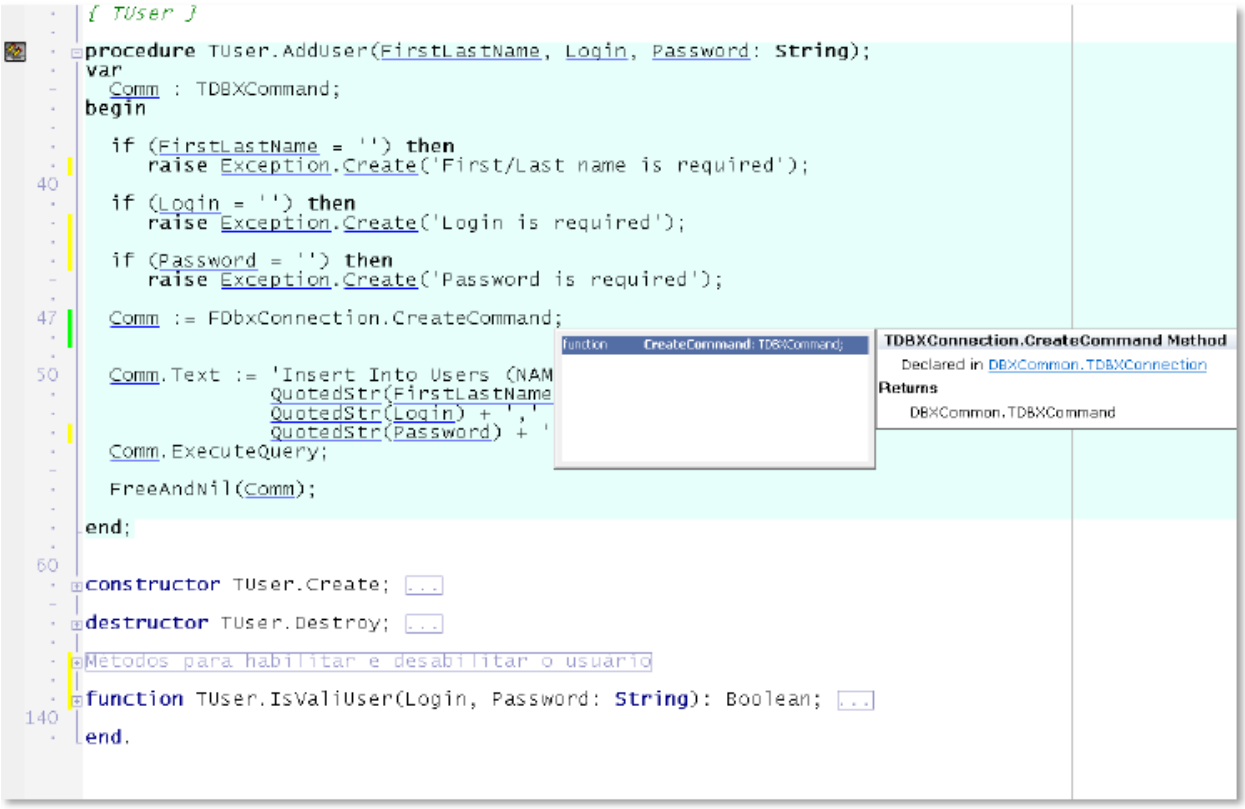


Фигура 9. Современная Delphi в стиле Delphi 7

РЕДАКТОР КОДА (CODE EDITOR)

Новая возможность в виде механизма *Live Templates* появилась в Delphi 2006 и позволяла создавать пользовательские шаблоны. Они сохраняются в виде XML-файлов и помогают уменьшать объем кода, вводимого вручную. *Block completion* автоматически вставляет begin и end. Даже такой простой сервис помогает значительно сэкономить время.

Представим ситуацию: необходимо изменить имя всех переменных в выделенной части исходного кода. Команда Find..Replace не совсем удобна в этой ситуации. Она не гарантирует изменение только имен переменных. Начиная с Delhi 8 можно использовать инструмент Sync Edit для редактирования различных фрагментов кода одновременно, воздействуя на один и тот же идентификатор. В качестве примере рассмотрим код внизу, где выбирается блок кода, активируется сервис Sync Edit, а затем изменяется переменная «Comm» только один.



Фигура 10. *Live Templates*, *Sync Edit*, свёртка кода и другие возможности редактора кода

Сбоку от блока кода можно видеть желтые и зелёные полоски. Жёлтые показывают участки, которые были измененные с последнего сохранения. Зелёные полоски, в свою очередь, показывают строчки, которые недавно были измененные и сохранены.

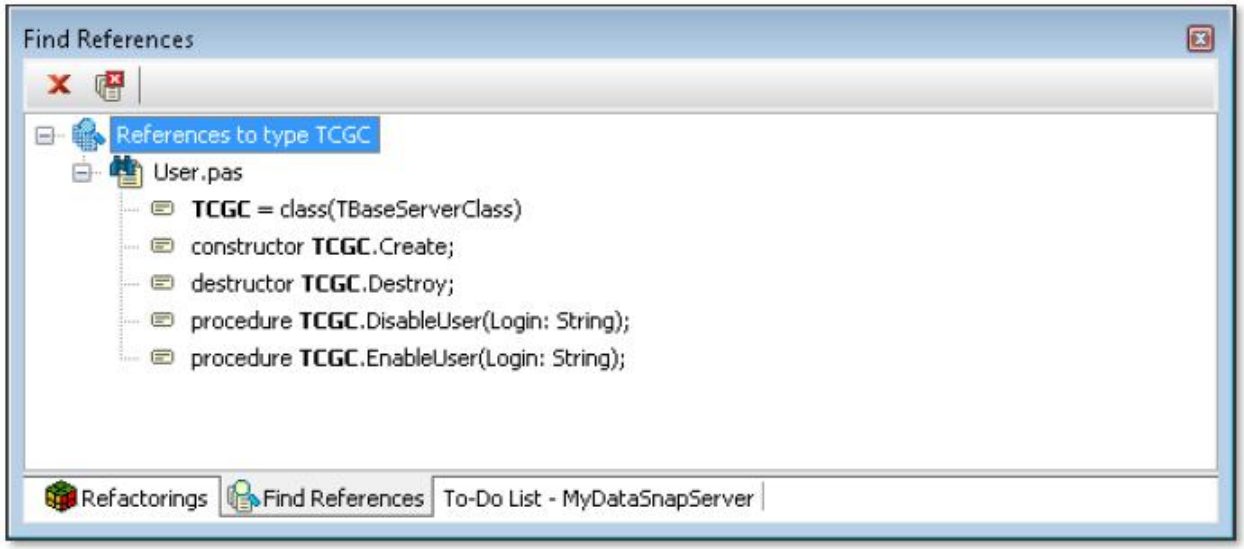
Также можно заметить «умную» нумерацию строк кода. Кроме того, есть функция сворачивания и разворачивания метода или класса непосредственно внутри блока.

Например, есть модуль, в котором представлены десятки методов. Оптимальным является выбрать время и привести код в порядок, однако это достаточно сложно. Код на картинке выше имеет область, обозначенную как «Metodos para habilitar e desabilitar o usuario». Эта

область имеет два метода, объединенных общим смыслом. Методы невидимы, пока область не раскрыта. В ряде случаев эта возможность является полезной.

Вызов справки непосредственно из кода (Help Insight): нужно нажать F1 для просмотра документации по методу, типу, классу и т.д. Как можно видеть, на рисунке выше отображается информация из справочной системе по методу CreateCommand. То же самое срабатывает для любого метода, типа или класса, если для него имеется описание.

Как работает поиск ссылок на методы, классы, переменные или другие идентификаторы? Например, в коде проекта есть класс, названный TCGC, который нужно переименовать в TCNPJ. Как определить, где в рамках проекта есть ссылки на данный класс TCGC? Find..Replace не будет работать в этом случае. Вместо этого нужно нажать Shitf + CTRL + Enter в классе TCGC. Среда разработки найдет все ссылки в коде проекта, как показано ниже.



Фигура 11. Поиск ссылок

Если теперь возникает необходимость переименовать все ссылки на TCNPJ, то эта возможность описана в разделе, посвященном рефакторингу.

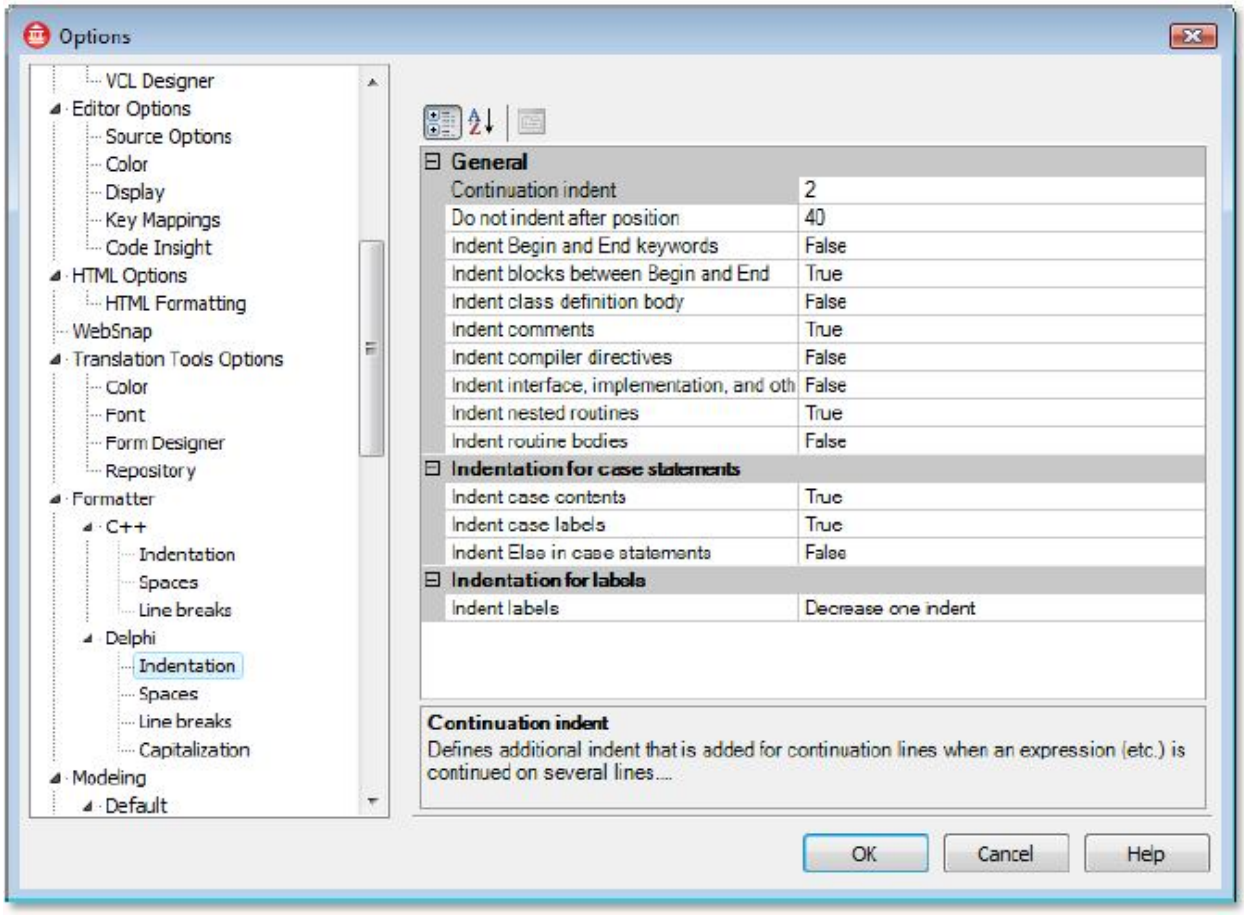
Другая полезная возможность называется Surround («обрамление»). Она позволяет «обрамлять» блок кода конструкциями типа begin/end, if/begin/end, try/finally, try/except и т.д.

Форматирования исходного кода (SOURCE CODE FORMATTER)

Среда Delphi дает возможность форматировать код в рамках шаблона по-умолчанию. Многие разработчики пользуются этим средством, однако многие предпочитают свой собственный стиль форматирования, что порождает большие дискуссии. Однако теперь это перестало быть проблемой.

Теперь среда разработки обеспечивает полностью настраиваемый сервис форматирования кода, который активируется при нажатии CTRL + D. Это позволяет отформатировать модуль в соответствии с заданными настройками. Более того, можно использовать Project Manager для форматирования всех модулей, входящих в состав проекта.

Сервис IDE Insight Formatter (Фигура 12) позволяет задавать пользовательские опции для отступов, переходов на новую строку и изменение букв на заглавные, тогда исходный код будет отформатирован соответствующим образом.



Фигура 12. Окно задания опций сервиса форматирования

Также реализована поддержка профилей, что позволяет переключаться между различными наборами параметров форматирования. Опции форматирования хранятся в конфигурационных файлах. Выбором профиля загружаются опции профиля, хранящиеся в соответствующем файле.

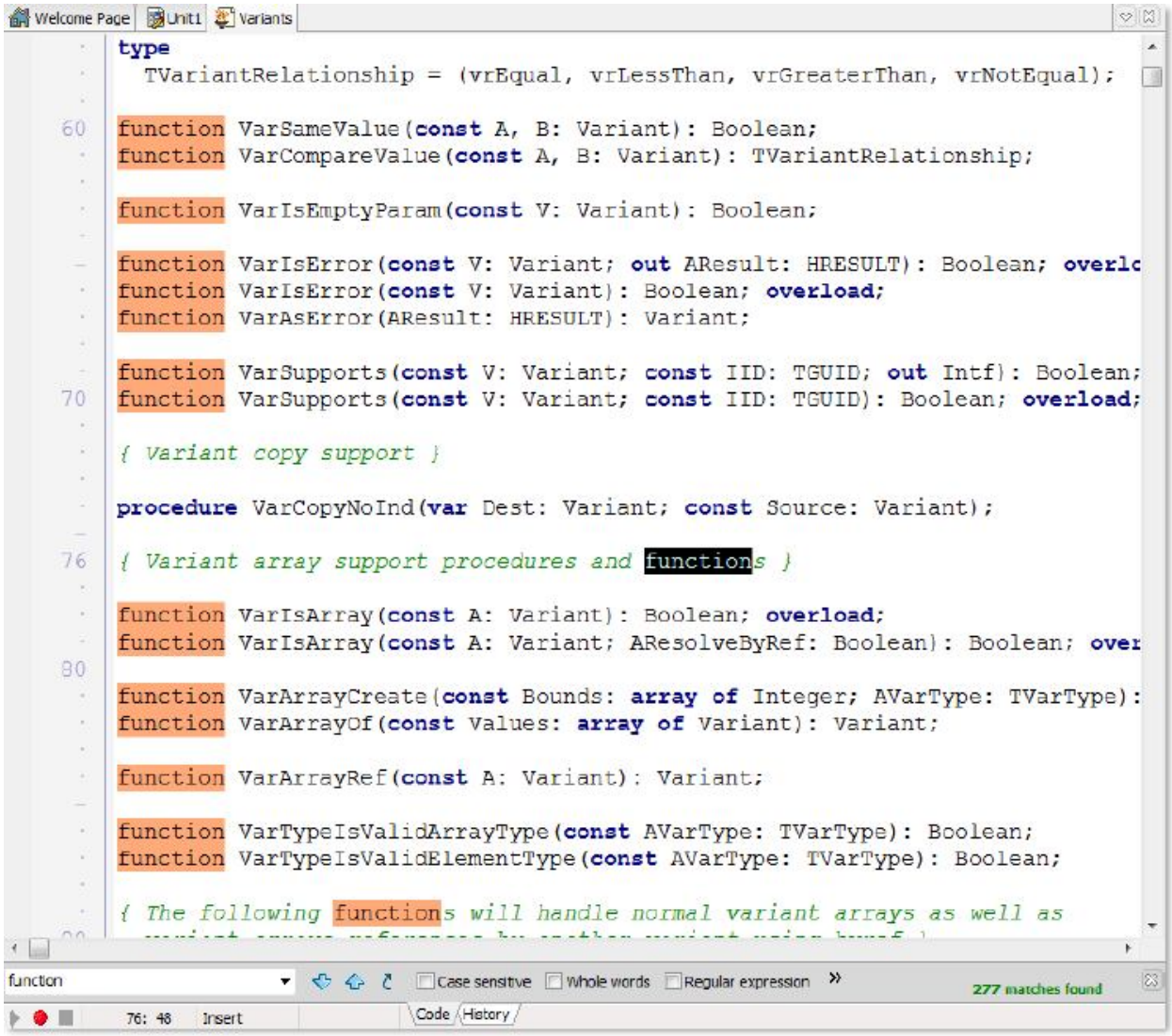
Для создания конфигурационного файла нужно использовать кнопку Save As. По-умолчанию конфигурационный файл имеет расширение .config.

Профили представляют собой особый вид конфигурационного файла. Профили имеют следующие дополнительные свойства:

- Профили должны соответствовать маске Formatter_*.config (маска профиля).
- Профили должны быть расположены в рабочей директории RAD Studio. По-умолчанию это папка «%APPDATA%\Embarcadero\RAD Studio\VersionNumber».

Поиск в редакторе кода

Сервис поиска в редакторе кода был усовершенствован и теперь представляет собой вариант современную популярную реализацию (как, например, в Firefox и Internet Explorer). Теперь можно нажать CTRL + F и вводить слово для поиска, а результаты будут отображаться так, как показано на Фигуре 13. Первое вхождение слово от положения курсора подсвечивается черным, а все остальные вхождения – оранжевым.

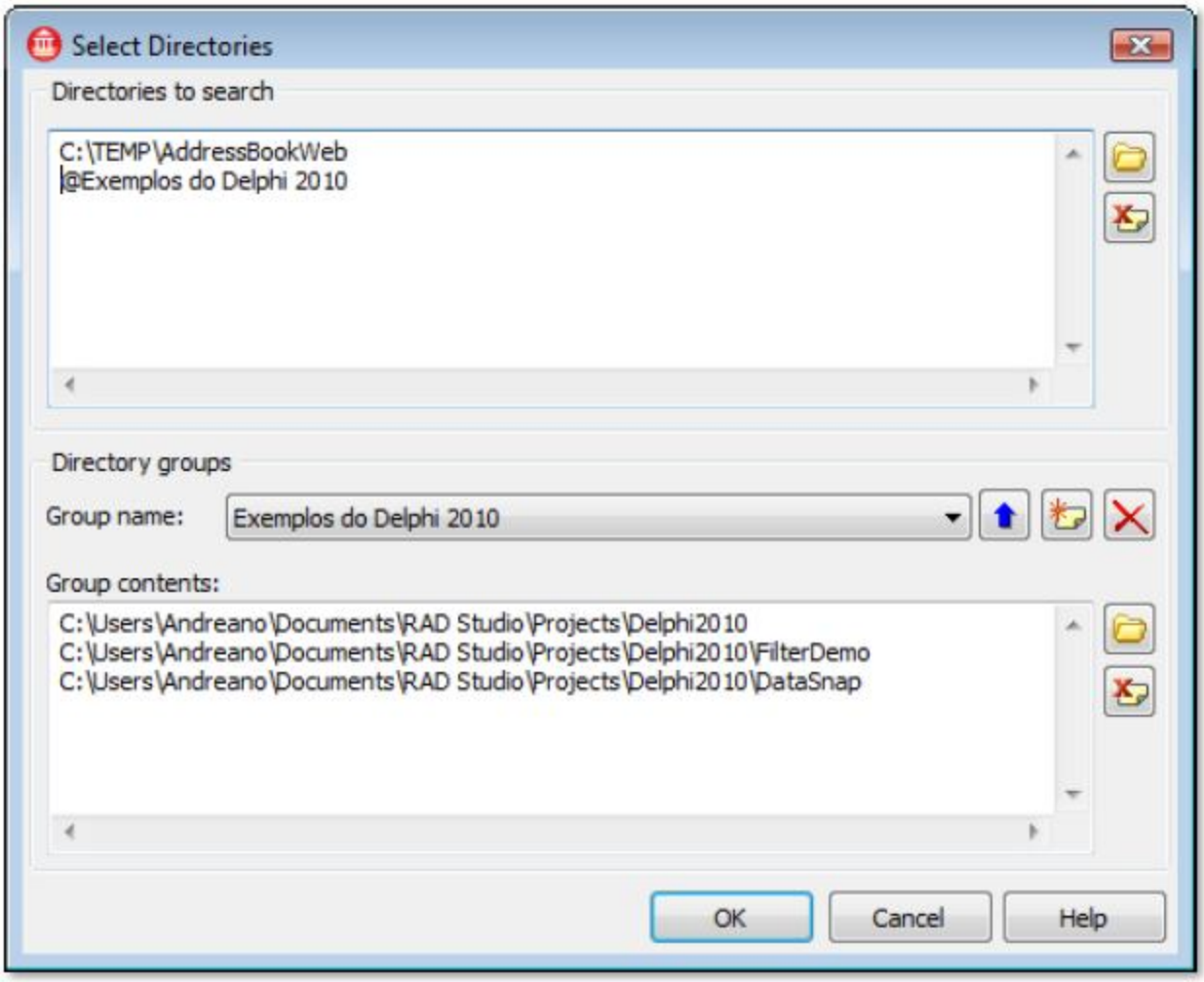


Фигура 13. Поиск по слову «function» дает такие результаты

Поиск в файле

Поиск какого-либо содержимого в файлах является обычной задачей, которая требует значительного времени, тем более что не всегда поиск производится в нужных папках, которые имеют отношение к текущему проекту.

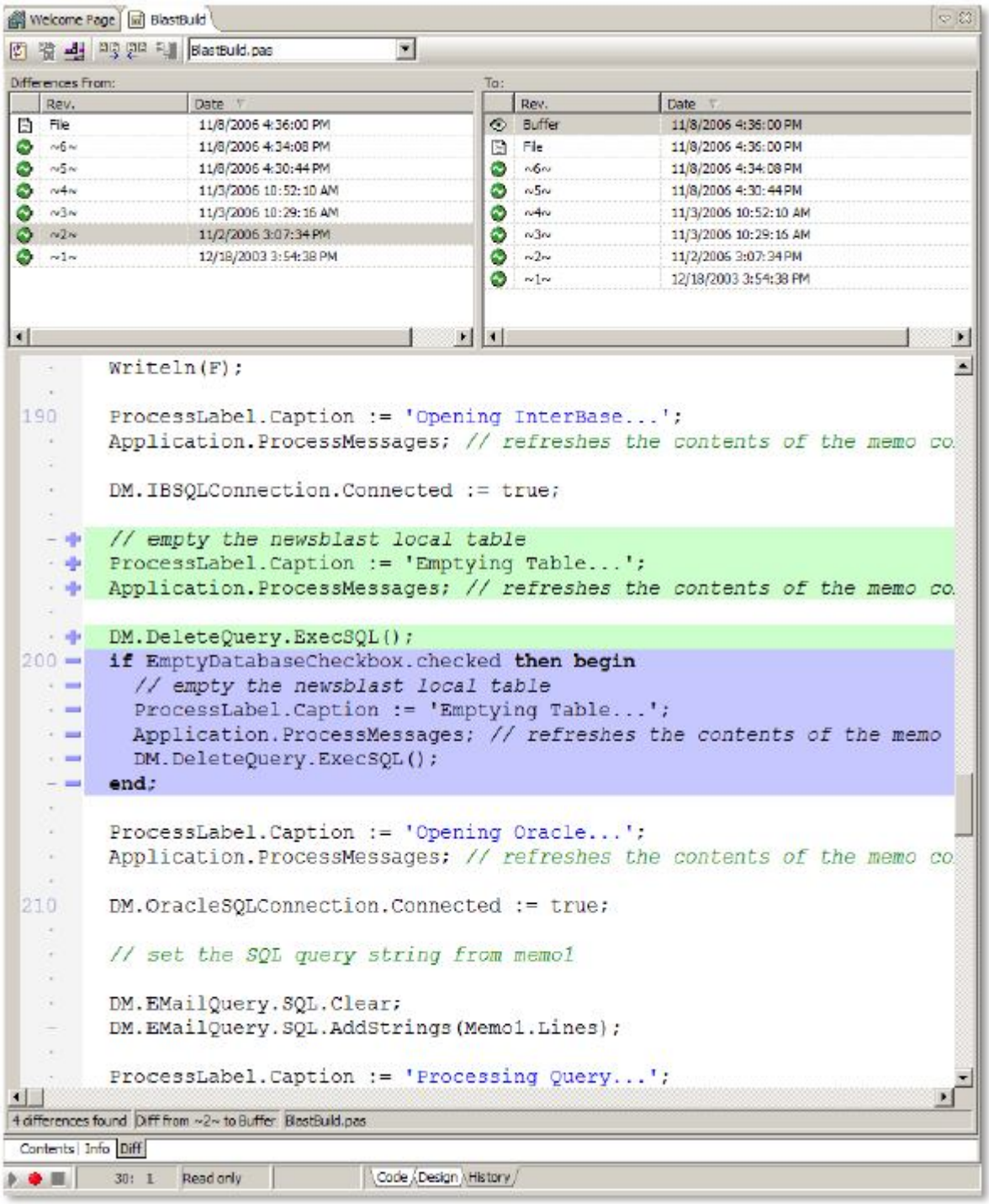
Начиная с Delphi 2010, была добавлена возможность поиска в выбранных директориях для операции «Find in Files» боле удобным способом. Можно сохранить список выбранных директорий, которые затем будут использоваться вместе с «Find in Files» на регулярной основе (см. Фигура 14).



Фигура 14. Окно поиска в файлах

История изменений (CHANGE HISTORY)

По мере изменения локальных файлов происходит сохранение их версий даже в отсутствии специализированной системы. Пользователь всегда имеет возможность сравнивать различные версии файла.

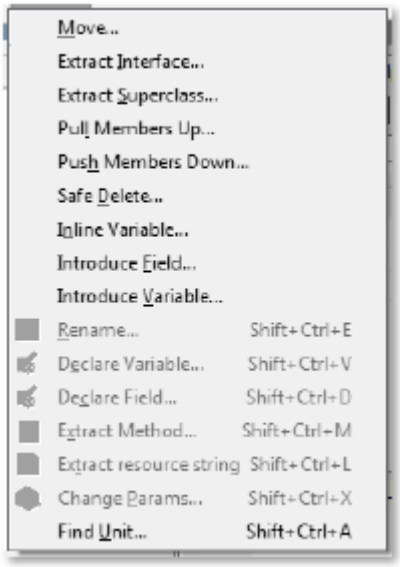


Фигура 15. История изменений (Change History)

Если проект находится под управлением Subversion, то сервис истории изменений также отобразит различия между двумя ревизиями выбранного файла.

Рефакторинг (REFACTORING)

Пользователям Delphi 7 однозначно понравится данная возможность. Рефакторинг – это технология для реструктуризации и модификации существующего кода при сохранении его функциональности. Рефакторинг позволяет убыстрить, упростить и улучшить и производительность, и читабельность кода приложения.



Delphi включает в себя сервис рефакторинга, который анализирует и выполняет операции реструктуризации. Сервис также отображает изменения в режиме предварительного просмотра, которые появятся на панели рефакторинга в нижней части редактора кода. Потенциальные операции рефакторинга отобразятся в узлах дерева, которые можно открыть, чтобы просмотреть дополнительные элементы, подверженные рефакторингу. Предупреждения и ошибки также отобразятся на этой панели. К сервису рефакторинга можно получить доступ из главного, а также контекстно-зависимого меню.

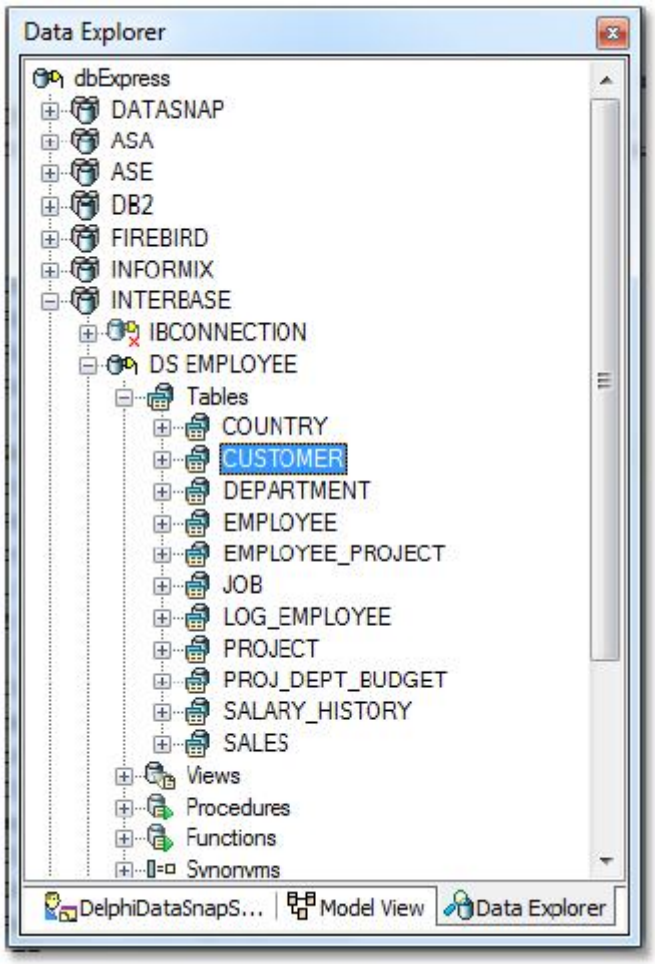
Модульное тестирование (UNIT TESTING)

Delphi включает в себя фреймворк DUnit для тестирования с открытым исходным кодом, позволяющий создавать и запускать автоматизированные тесты. Этот фреймворк упрощает процесс создания тестов для классов и методов приложения. Используя данную возможность в комбинации с рефакторингом, можно значительно повысить стабильность приложения. Частое выполнение набора тестов по мере внесения изменения в исходный код проекта позволяет выявлять и исправлять ошибки на ранних стадиях процесса разработки.

Обозреватель данных (DATA EXPLORER)

Теперь связь с базами данных для получения данных на этапе разработки стала гораздо проще. С использованием обозревателя данных с функцией «перетаскивания» можно получать доступ к таблицам, просмотрам и другим элементам в базе данных. Кроме этого, можно обрабатывать данные с использованием SQL. Соединения осуществляются посредством библиотеки dbExpress. Это означает, что обозреватель данных поддерживает все те платформы баз данных, которые поддерживает dbExpress.

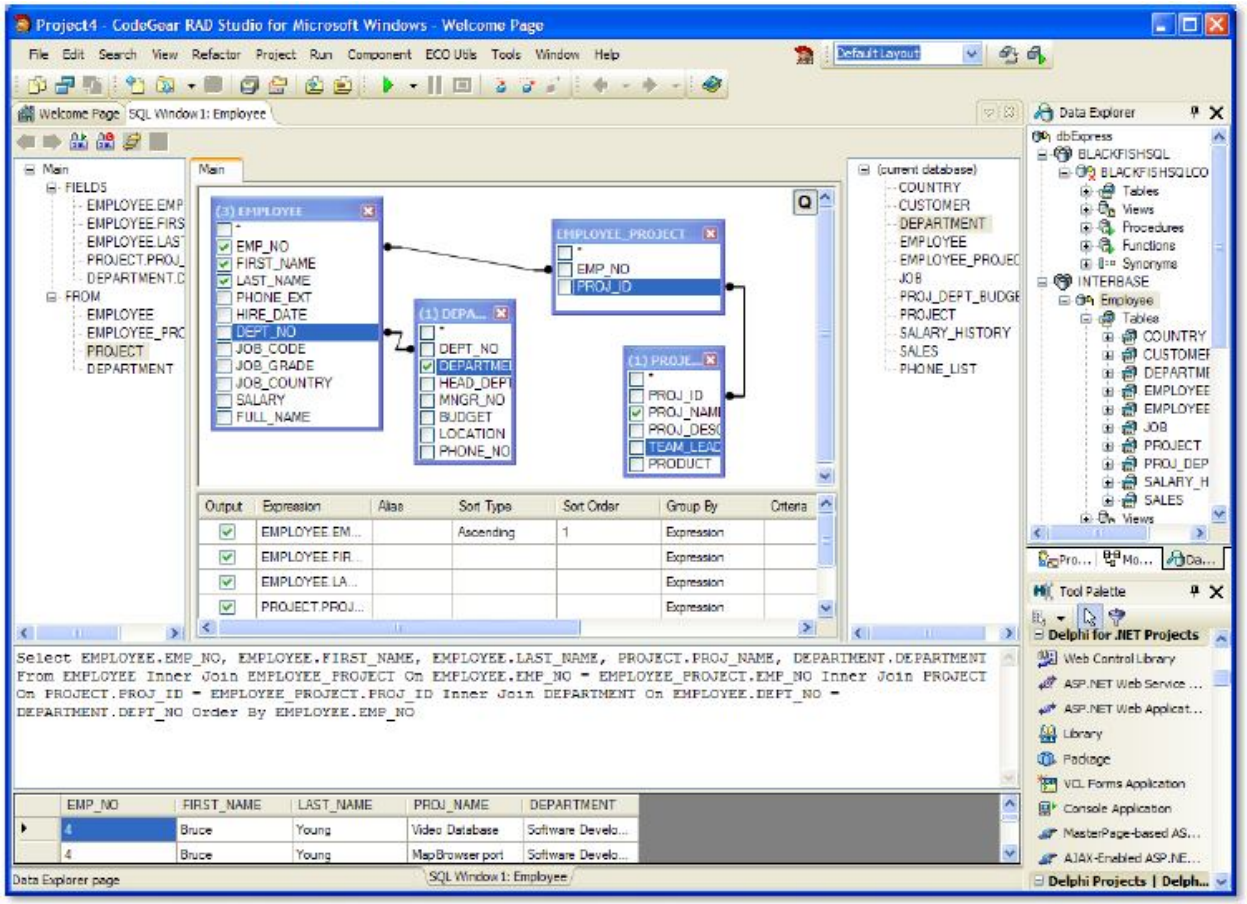
Каждое соединение получает свой псевдоним, который сохраняется в файле dbxconnections.ini (конфигурационный файл для dbExpress). Информация о псевдонимах является доступной для всех подключений через данную библиотеку, поэтому обозреватель данных очень удобен в использовании.



Фигура 16. Обозреватель данных (Data Explorer)

Окно SQL – построитель запросов (SQL WINDOW - QUERY BUILDER)

Обозреватель данных позволяет строить сложные SQL-запросы с использованием интуитивно-понятного визуального интерфейса.

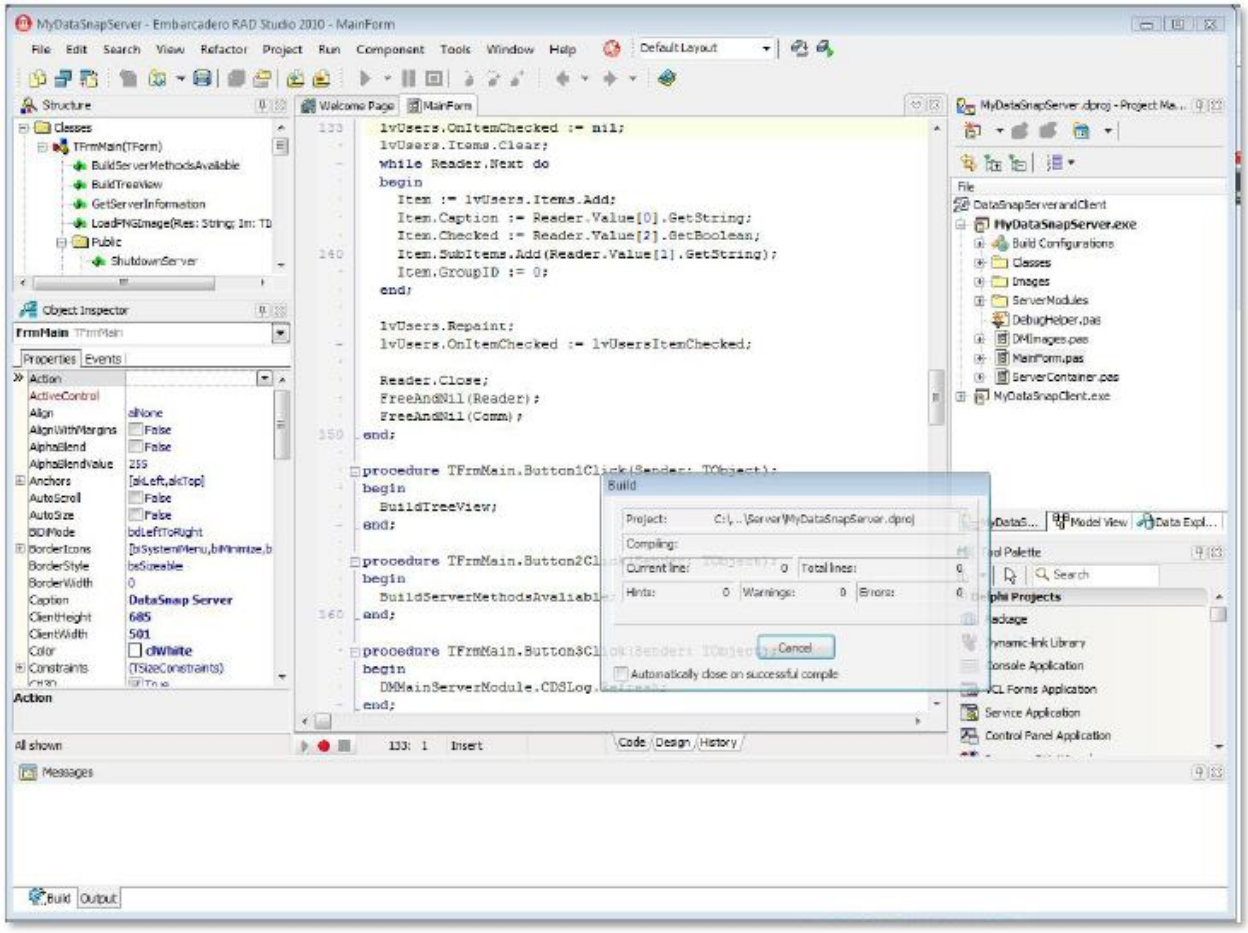


Фигура 17. Окно SQL

Фоновая компиляция (BACKGROUND COMPILATION)

Начиная с Delphi 2010 можно осуществлять фоновую компиляцию. Таким образом, можно начать процесс компиляции в отдельном или параллельном потоке и продолжить работать в интегрированной среде, в то время как проект будет компилироваться.

Можно продолжить работу в интегрированной среде в течение компиляции проекта. Например, можно редактировать конкретный файл даже в процессе его компиляции, а также задавать и модифицировать точки останова.



Фигура 18. Полупрозрачное диалоговое окно с фоновой компиляцией

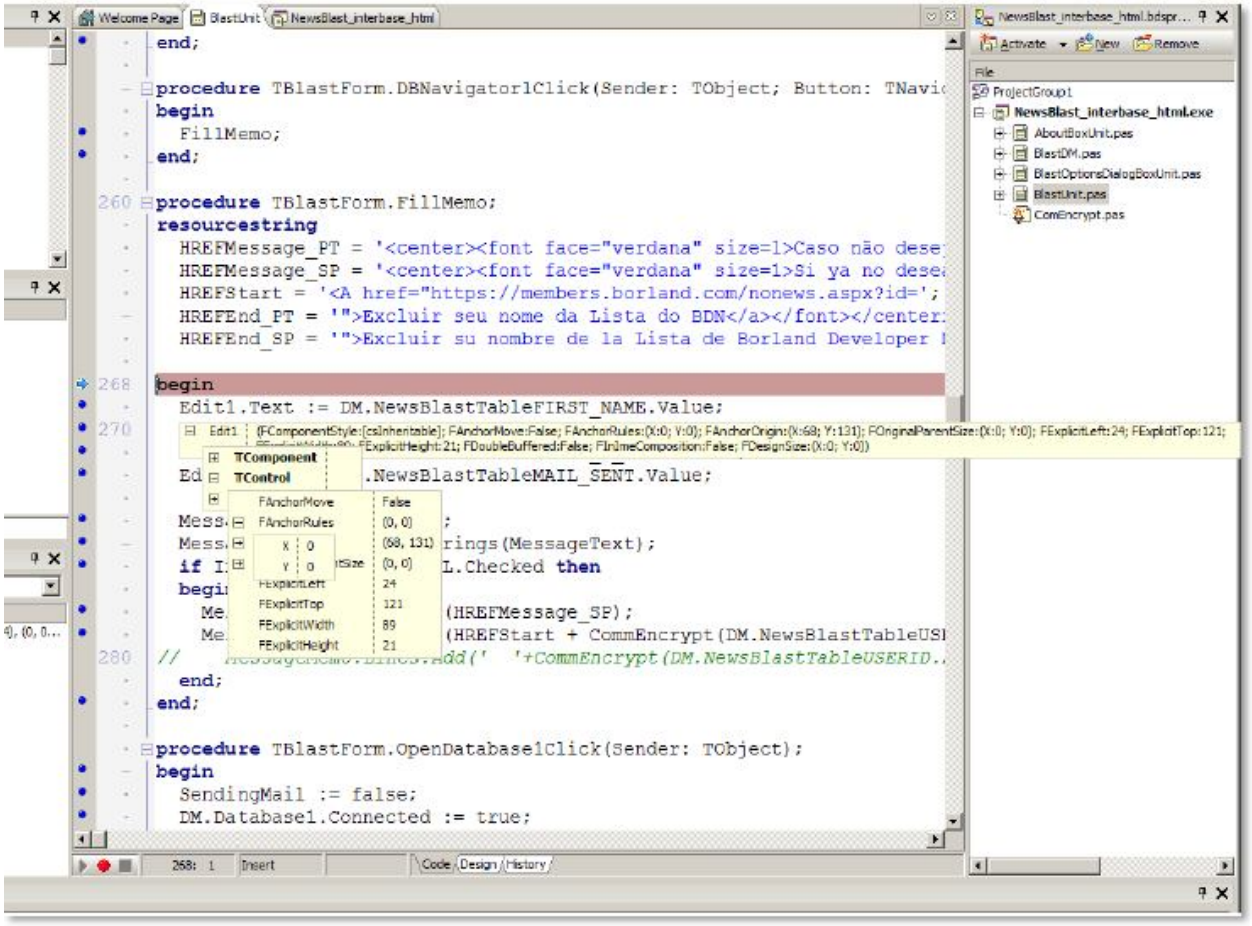
Отладчик

Теперь отладчик имеет опцию «Thread View and Wait Chain Traversal» («просмотр потоков и отслеживание цепочек ожиданий»), которая имеет смысл только для Windows Vista и Windows 7. Эта возможность позволяет выявлять взаимные блокировки и конфликты между потоками.

В течение процесса отладки бывает полезным визуализировать значения переменных. Для этого очень хорошо подходит Watch List (список просмотра), но чем больше значений добавлено, тем более запутанной становится визуализация. Теперь разработчики могут группировать переменные в Watch List по именам. Пользовательские группы отображаются закладками в Watch List.

После того как отладка завершена, все модули, которые открыты в процессе, автоматически закрываются; только тем модули остаются открытыми, которые были таковыми перед запуском процесса отладки.

Появилось множество улучшений с точки зрения эргономики в окне визуализации локальных переменных, в окне call stack (стек вызовов) и др. Также появилось новое дерево просмотров содержимого отлаживаемых объектов, как показано на Фигуре 19:



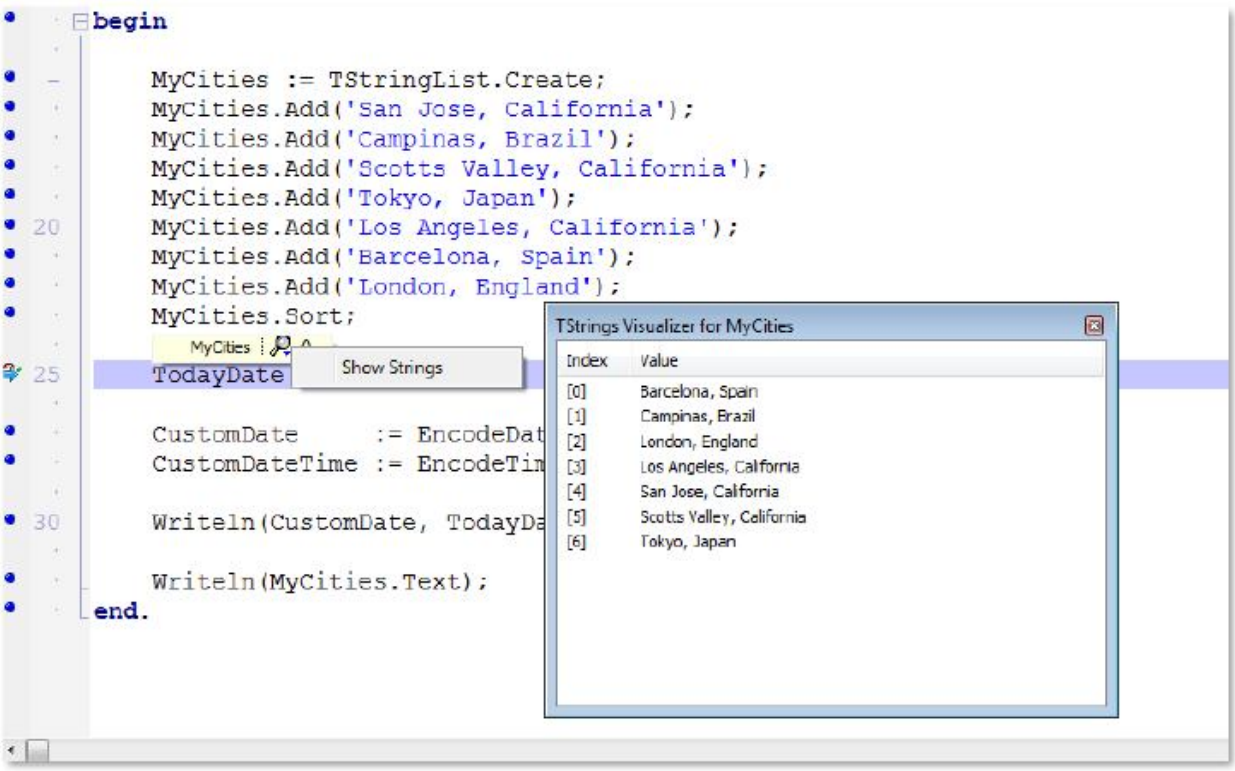
Фигура 19. Отладчик

Начиная с Delphi 2010, стало гораздо проще отлаживать многопоточные приложения. Новые возможности в отладчике позволяют выполнять пошаговую отладку в потоках, также как «замораживать» и «размораживать» потоки. Это позволяет изолировать потоки, которые нужно отладить, временно «замораживая» ненужные. Теперь есть опции в «Thread Status View», такие как «Freeze» («заморозить»), «Freeze All Other Threads» («заморозить все другие потоки»), «Thaw» («разморозить») и «Thaw All Threads» («разморозить все потоки»). Также

теперь есть возможность задавать точки останова (breakpoint) для приостановки отдельного потока.

Визуализаторы в отладчике значительно упрощают отладку. Достаточно часто приходится визуализировать данные типа TDate, TTime и TDateTime. Среда разработки отображает значения этих типов в удобочитаемом формате – привычное отображение даты и времени вместо значений с плавающей точкой. То же самое справедливо для TStringList – список строк показывается в виде обычного текста.

Более того, можно добавлять свои собственные визуализаторы в отладчике для пользовательских типов данных.



Фигура 20. Визуализатор для типа данных TStringList

Что нового в VCL и RTL

Начиная с Delphi 7, библиотеки VCL и RTL постоянно расширяются. Помимо полной поддержки Windows XP, 200, Vista, Windows 7 и Unicode, были добавлены новые и расширены существующие компоненты за счет добавления новой функциональности.

Все эти нововведения повышают возможности компонентов в плане эргономики, облегчают процесс создания расширенных интерфейсов и позволяют использовать новую функциональность Windows Vista и Windows 7, также как и обеспечивают поддержку управления и ввода при помощи касаний и жестикуляции. В этом разделе рассматриваются нововведения в существующих компонентах и изменения в классах библиотеке RTL.

VCL Direct2D и Windows 7

В Microsoft Windows 7 появилась возможность использовать Direct2D: API для 2-D графического вывода с аппаратным ускорением, что обеспечивает повышенную производительность при отображении двухмерных объектов, растровых изображений и текста. Программный интерфейс Direct2D API предназначен для взаимодействия с GDI, GDI+ и Direct3D. Direct2D перенаправляет все операции рисования в GPU (Graphic Processing Unit) вместо CPU, а это дает больше ресурсов приложению. Эта часть посвящена обсуждению того, как получить максимум преимущества от поддержки Direct2D в Delphi.

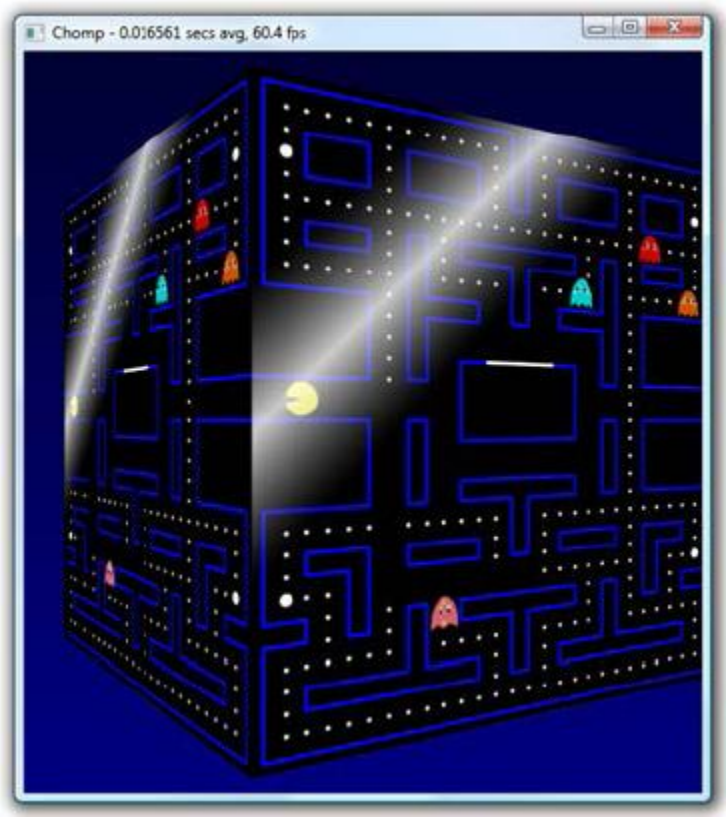
Direct2D поддерживается только в Windows 7 для проектов, сделанных в Delphi 2010 (или более поздней версии). Такие приложения на 100% совместимы с Windows 7 и используют все возможности этой новой операционной системы Microsoft, такие как Direct2D. Чтобы обеспечить поддержку Direct2D в разрабатываемом приложении, нужно включить следующие модули:

- Direct2D, который экспонирует классы-оболочки в виде TDirect2DCanvas VCL
- D2D1, который содержит трансляцию заголовочных файлов в Microsoft Direct2D API

Следующий пример демонстрирует, как переопределяется метод Paint формы с использованием класса 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 является естественной заменой Canvas, разница в качестве графического отображения очень велико, что и показано на Фигуре 21.



Фигура 21. Графика с использованием Direct2D

Касания и жестикуляция (Touch and Gestures)

Рынок разработки программных продуктов отличается от ситуации 2009 года. Теперь стали появляться приложения для камер, телефонов, приборов на основе GPS с использованием сенсорного ввода и управления на основе касаний и жестикуляции. Windows 7 ввела поддержку множественного сенсорного ввода (multi-touch), когда пользователи могут взаимодействовать с экранными приложениями двумя и более пальцами.

Перед тем, как начать разрабатывать приложение с сенсорным вводом, важно правильно классифицировать предполагаемый тип сенсорного ввода:

- Базовый сенсорный ввод (Basic Touch) → уже применяется в коммерческих приложениях, когда пальцы заменяют правую кнопку мыши, например, платежные терминалы и т.д.

- Множественный сенсорный ввод → элементы интерфейса приложения обеспечивают интенсивный ввод посредством касаний, например, iPhone и эпизод в фильме «Особое мнение» («Minority Report»). Данная возможность поддерживается только в Windows 7 и на новом аппаратном обеспечении.

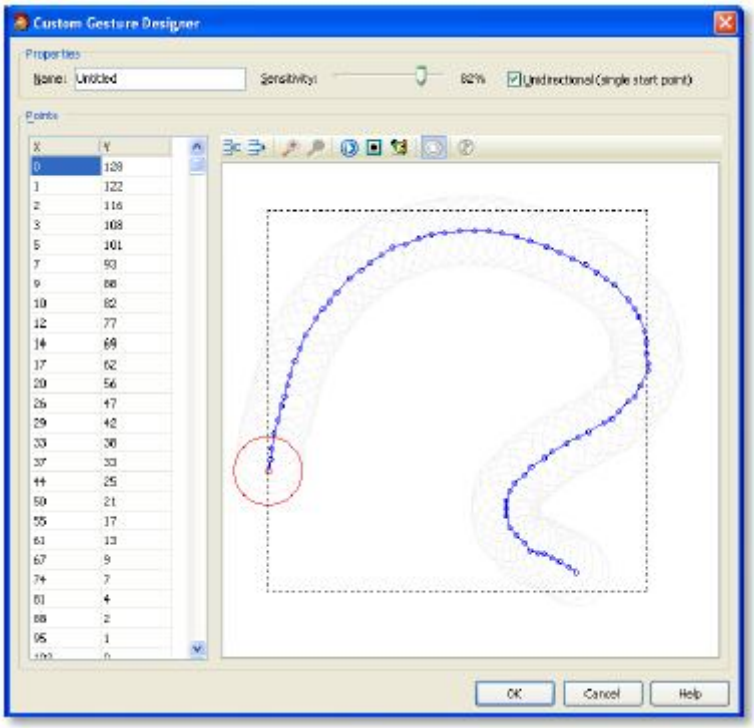
- Ввод и управление при помощи жестов → перемещения пальцев или «мыши» генерируют события.

Delphi обладает архитектурой, которая дает возможность пользователю подключить механизм интерпретации жестов. Данный механизм будет работать во всех поддерживаемых версиях Windows, а не только в Windows 7. Он поддерживает обратную совместимость с существующим аппаратным обеспечением и дает возможность пользователям эмулировать движение «мыши» или касание.

Библиотека VCL поддерживает интерпретацию более 30 стандартных заранее определенных жестов, также как и редактор для их создания, тестирования, хранения и последующего распознавания в виде элементов GESTURE. Кроме этого, Delphi включает компонент для эмуляции виртуальной клавиатуры (TTouchKeyboard).

Все визуальные компоненты имеют свойство Touch, который имеет вложенное свойство под названием Gesture Manager («менеджер жестов») для связи с соответствующим компонентом. Компонент Gesture Manager инкапсулирует в себе механизм управления при помощи жестов для всего приложения, он активирует действия для каждого жеста при помощи Action Manager. Это требует лишь простого связывания компонента Gesture Manager с компонентом Action Manager.

Фигура 22 демонстрирует окно Gesture Designer, которое позволяет записывать, воспроизводить и тестировать движения для создания жеста. Записанный жест потом может быть использован в компонентах VCL для активации соответствующего действия. Редактор также позволяет также изменять жест и варьировать чувствительностью при его восприятии.



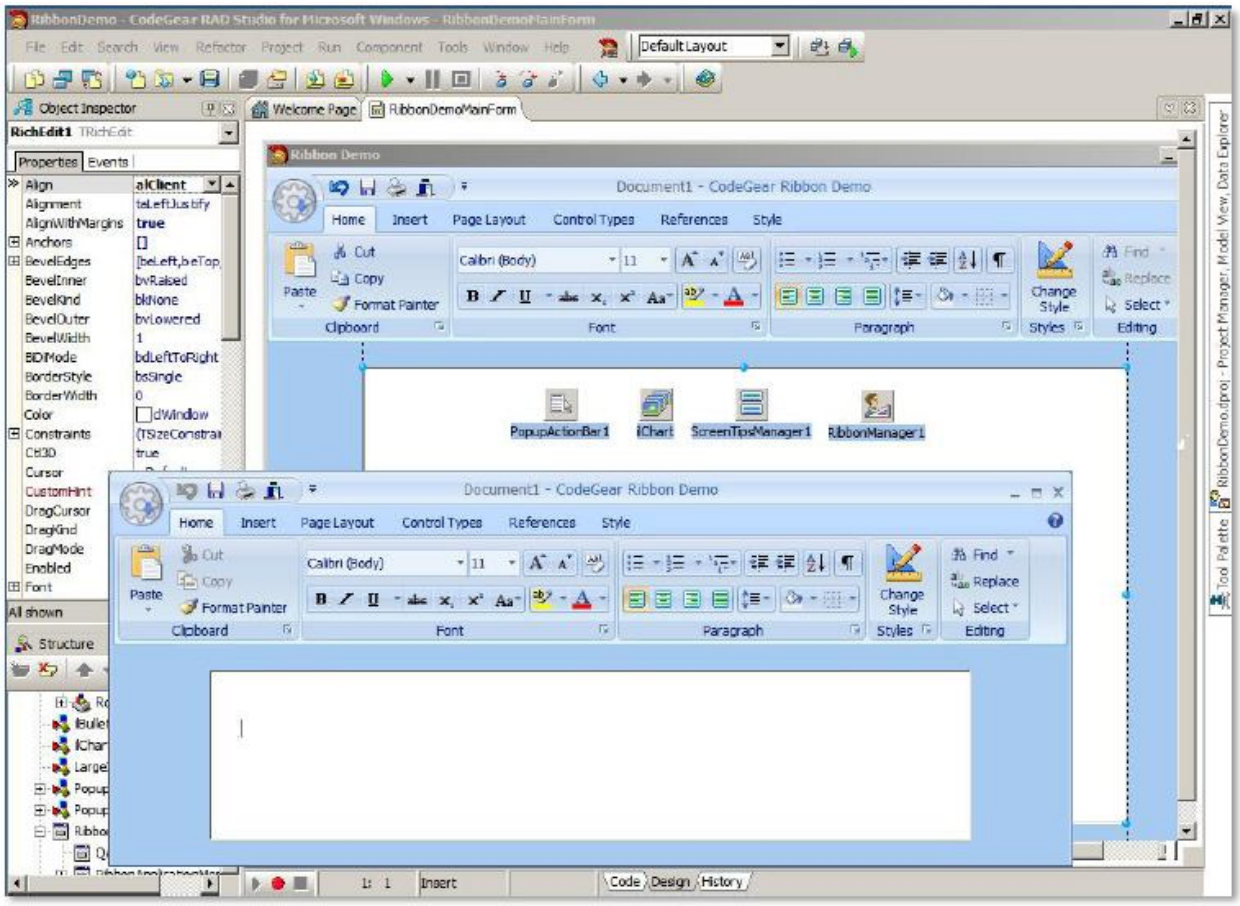
Фигура 22. Gesture Designer позволяет записывать и редактировать жесты

Ленточные элементы управления (RIBBON CONTROLS)

«Ленточные» интерфейсы стали широко известны благодаря Office 2007. Такой тип интерфейса отлично комбинирует главное меню и инструментальную панель приложения.

Теперь библиотека VCL содержит группу «ленточных элементов управления» (Ribbon Controls) для создания интерфейсов Delphi-приложений в стиле Ribbon.

Архитектура в основе Ribbon Controls очень проста. На компонент Ribbon можно добавить закладку (Tab), которая содержит группы. Каждая группа содержит кнопки с управляемым внешним видом. Кроме того, компонент Ribbon включает *Quick Access Toolbar u Application Menu.* Фигура 23 демонстрирует пример приложения с Ribbon Controls.



Фигура 23. Ribbon Controls

Поддержка WINDOWS VISTA и WINDOWS 7

Библиотека VCL была обновлена для поддержки новых характеристик операционных систем Windows Vista и Windows 7, поэтому в её состав вошли новые компоненты TFileOpenDialog, TFileSaveDialog и TTastDialog.

Также были добавлены новые классы, например:

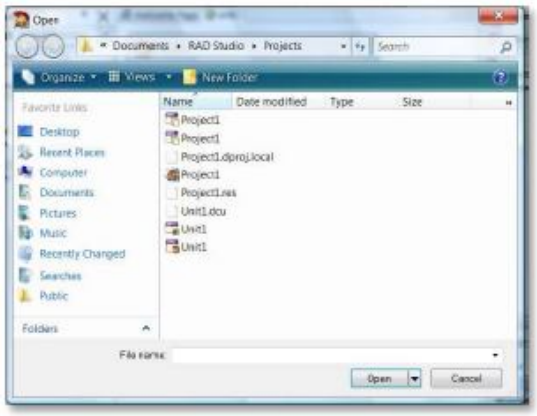
- TCustomeFileDialog
- TCustomeFileOpenDialog
- TCustomFileSaveDialog
- TCustomTaskDialog
- TFavoriteLinkItem
- TFavoriteLinkItems
- TFavoriteLinkItemsEnumerator
- TFileTypeItem
- TFileTypeItems
- TTaskDialogBaseButtonItem
- TTaskDialogButtonItem
- TTaskDialogButtons
- TTaskDialogButtonsEnumerator
- TTaskDialogProgressBar
- TTaskDialogRadioButtonItem

Компоненты типа Dialog отображаются в стиле Vista. Возможно, возникает вопрос, а что будет, если запустить приложение под управлением Windows XP. Здесь причин для беспокойства нет. Компоненты VCL распознают версию ОС, используют её особенности и формируют соответствующий вид.

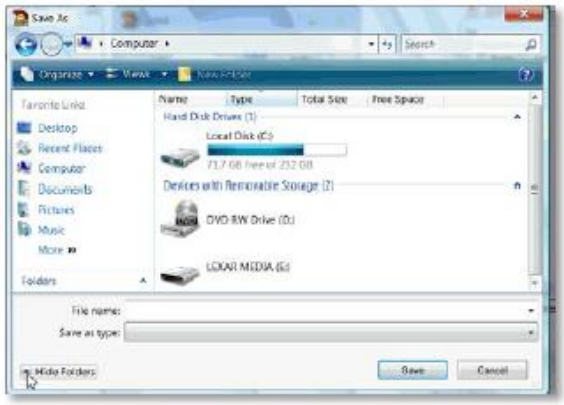
Добавлена функция TaskMessageDlg для поддержки Windows Vista. Она обладает такой же функциональностью, что и MessageDlg, но также содержит особенности, характерные для Windows Vista. Когда приложение запускается под управлением Windows XP, автоматически выполняется MessageDlg. VCL берет на себя ответственность за решение данной проблемы.

Глобальная переменная UseLatestCommonDialogs определяет, что все компоненты типа Dialog (TOpenDialog, TSaveDialog, TOpenPictureDialog и TSavePictureDialog) должны отображаться в стиле Vista, как только она получает значение TRUE.

Например, ниже показано, как диалоги Open и Save выглядят в Windows Vista и Windows 7:



Фигура 24. Open Dialog в Windows Vista



Фигура 25. Save Dialog в Windows Vista

Перечисленные ниже модули были расширены для поддержки API новых версий Windows.

- UxThemes – новый
- DwnApi – новый
- ActiveX – обновлен
- Windows – обновлен
- Messages – обновлен
- CommCtrl – обновлен

Новые и расширенные компоненты VCL

TActionManager

Новые свойства – DisabledImages, LargeImages и LargeDisabledImages – позволяют задавать большие и неактивные изображения на основе компонента TImageList.

PNG Support

Компонент Image поддерживает формат PNG.

TBitMap

Компонент поддерживает 32-битные растровые изображения с альфа-каналом, а также дополнительное свойство AlphaFormat.

TButtonGroup

Данный компонент позволяет группировать различные кнопки на панели.

TButtonEdit

Новый компонент TButtonEdit дает возможность добавлять изображения к полю ввода Edit. Изображения могут размещаться как справа, так и слева. Также можно управлять событиями изображений: onLeftClick и onRightClick.

TLinkLabel

LinkLabel позволяет использовать тэги HTML, что изменяет внешний вид компонента.

TPopupActionBar

Теперь поддерживает стили ActionBar.

THeaderControl и THeaderSection

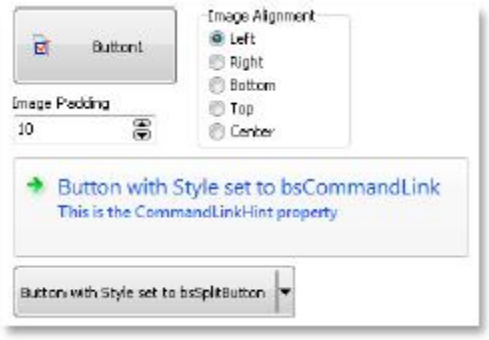
Поддержка нового checkbox.

TButton

Windows Vista теперь определяет два новых стиля кнопки, которые оба поддерживаются в Delphi в классе TButton.

ComandLink имеет другой, более дружественный дизайн. Можно использовать его для добавления более тонкой настройки к функциональности кнопки.

SplitButton предлагает список опций при клике на нем. Список представлен как всплывающее меню. К пунктам меню можно добавить изображения.



Фигура 26. Новые стили кнопок

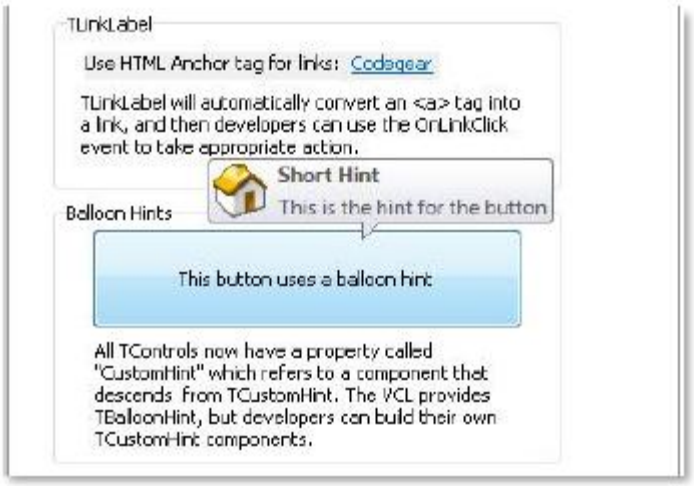
TListView и TTreeView

TListView теперь позволяет определять основную и дополнительную группы. Дополнительная группа поддерживает более глубокую настройку (для Vista), позволяя задавать изображения для каждой из них. TTreeVeiw позволяет включать/выключать узлы и изображения для раскрываемых элементов.

Фигура 27. TListView

TBalloonHints

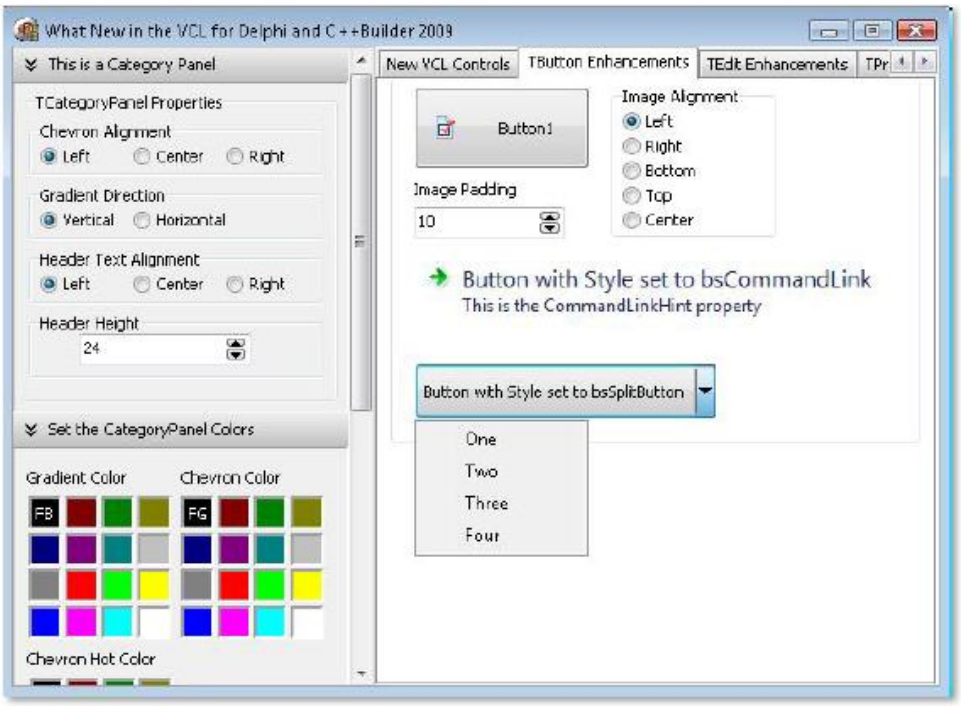
«Подсказки» (hints) теперь имеют вид Windows Vista и позволяют добавлять заголовки, описания и изображения, которые делают пользовательские уведомления более дружественными.



Фигура 28. Balloon Hints

TCategoryPanelGroup

Этот новый компонент очень полезен. Он работает как панель Outlook, к которой можно добавить много других различных панелей. Каждая из этих панелей может содержать любые компоненты VCL. Можно задавать заголовок, изображение, выравнивание и иконку для каждой из панели, а потом разворачивать и сворачивать их.



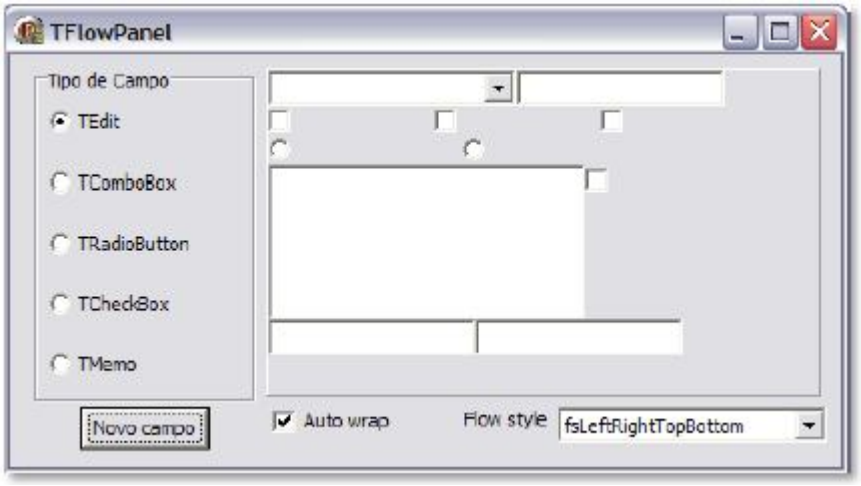
Фигура 29. Panel Group

Новые панели

Традиционные панели являются визуальным контейнерами для других компонентов. Внутри панели можно размещать визуальные элементы управления в любом порядке и количестве. Другими словами, он работает по принципу абсолютного позиционирования (координаты Top и Left конкретного элемента управления задают положение относительно верхнего левого угла панели).

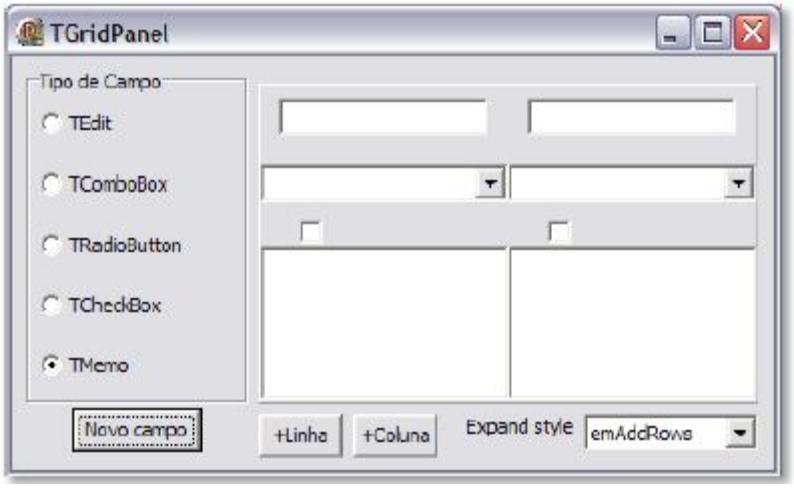
По аналогии с концепцией, принятой в средствах разработки для Java (а именно, «менеджер расположения» - Layout Manager), которая предопределяет, как элементы управления размещаются внутри контейнера, теперь в Delphi есть три типа менеджеров расположения:

- TPanel  : абсолютная расположение или XY. Компоненты размещаются в фиксированных, точных позициях.
- TFlowPanel  : компоненты размещаются последовательно в соответствии с определенным порядком (наподобие страницы HTML, но без таблиц или каскадных таблиц стилей CSS).
    - Размещение определяется свойством FlowStyle, которое может принимать различные значения, указанные ниже. В порядке понимания соглашения об именах, следует понимать, что компоненты располагаются в соответствии с направлением, заданным первой парой (например, LeftRight). Когда на панели больше нет места, направление переопределяется второй парой (например, TopBottom):
        - fsLeftRightTopBottom: слева направо, сверху вниз (по-умолчанию)
        - fsRightLeftTopBottom: справа налево, сверху вниз
        - fsLeftRightBottomTop: слева направо, снизу вверх
        - fsRightLeftBottomTop: справ налево, снизу вверх
        - fsTopBottomLeftRight: сверху вниз, слева направо
        - fsBottomTopLeftRight: снизу вверх, слева направо
        - fsTopBottomRightLeft: сверху вниз, справа налево
        - fsBottomTopRightLeft: снизу вверх, справа налево
    - Еще одно полезное свойство – AutoWrap. Когда его значение true, это означает, что размещение будет «перенаправлено» на другое направление, если панель не умещается в отведенное ей пространство. Когда его значение false, компоненты на панели, которые выходят за границы, будут невидимыми.
    - Можно использовать эту панель для автоматической генерации форм, когда поля будут задаваться динамически либо из базы данных, либо из файла. В этом случае не надо беспокоиться о позиционировании каждого из полей.



Фигура 30. Автоматическая генерация формы

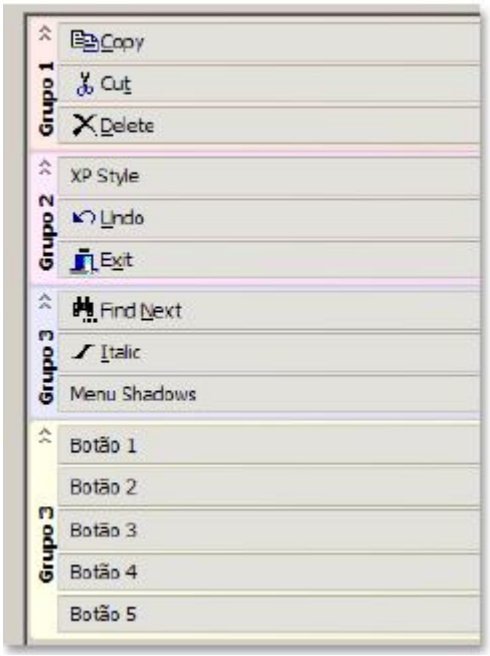
- TGridPanel  : эта панель разделена на участки по вертикали и горизонтали (строками и колонками). Каждая образованная ячейка может содержать компонент (подобно таблице HTML).
    - Компоненты размещаются в соответствии с положением их ячеек (сверху вниз) в колонках и строках (слева направо).
    - Количество строк задается свойством RowCollection, которое может содержать различные объекты класса TRowItem. Каждый элемент имеет два свойства:
        - SizeStyle: определяет стандарт, по которому высота строки задается свойством Value:
            - ssAbsolute: количеством пикселей
            - ssAuto: значение игнорируется, т.к. высота строки будет вычислена автоматически
            - ssPercent: в процентном отношении от высоты панели
        - Value: значение задает высоту в соответствии со значением свойства SizeStyle.
    - Подобно вышеизложенному, число колонок задается свойством ColumnCollection, которое содержит объекты класса TColumnItem. TColumnItem имеет такие же свойства, что и TRowItem, отличающиеся только тем, что они относятся к ширине колонки, а не высоте строки.
    - Свойство ExpandStyle задает действие, которое происходит, когда происходит попытка добавить новый компонент на панель, а свободных ячеек уже нет. Возможные значения:
        - emAddRows: добавляется новая строка к панели для размещения нового компонента
        - emAddColumns: новая колонка добавляется для размещения нового компонента
        - emFixedSize: возникает исключение, когда больше нет места для размещения нового компонента.



Фигура 31. Expand Property Style

TCategoryButtons

Этот компонент позволяет создавать кнопки и группировать их по категориям, подобно тому, как это происходит на панели Outlook. Это позволяет добиться более высокого уровня эргономичности дизайна интерфейсов приложений.



Фигура 32. TCategoryButtons

TDrawGrid, TStringGrid, TDBGrid

Поддержка «тем» и стилей градиента введена в «сеточные» компоненты.

TTrayIcon

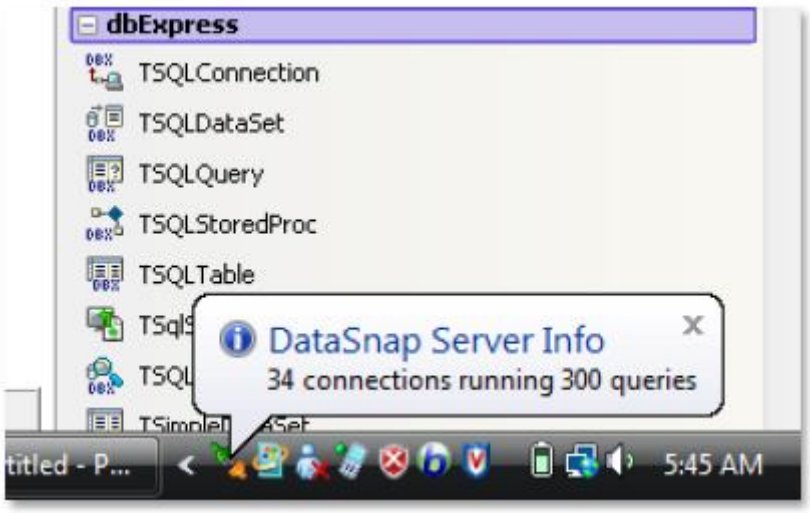
Этот компонент служит для создания интерактивных иконок в области «Windows tray». Теперь можно использовать эту возможность для обычных приложений гораздо более простым способом. Для этого нужно всего лишь разместить компонент TTrayIcon (закладка Win32) на главную форму. Далее нужно задать небольшое количество свойств:

- **Icon**: предназначено для хранения иконки, которая будет отображаться в области tray. Можно использовать иконку приложения или любую другую для описания статуса или ситуации. Эта иконка может быть изменена в любое время.
- **Icons**: содержит ссылку на TImageList, содержащий несколько растровых изображений или иконок, которые будут использованы в анимации.
- **Animate**: когда значение этого свойства True, то происходит автоматическая ротация иконок в списке Icons. Индекс иконки, которая отображается в текущий момент, может быть как получен, так и задан при использовании свойства IconIndex.
- **AnimateInterval**: интервал в миллисекундах, определяющий процесс ротации иконок. Событие OnAnimate генерируется после каждой итерации, позволяя управлять выполняемым действием.
- **BaloonTitle** и **BaloonHint**: заголовок и текст подсказки в стиле «balloon», отображаемой методом ShowBalloonHint. Данная подсказка может быть закрыта простым «кликом»

(как в любом месте подсказки, так и на кнопке [x]). Однако подсказка будет скрыта автоматически по прошествии интервала времени, заданного свойством BalloonTimeout (в миллисекундах).

- **PoupMenu**: служит для добавления контекстного меню к иконке для ускорения пользовательского доступа к наиболее часто используемым командам. Единственное, что нужно сделать, это ассоциировать уже имеющееся контекстное меню с этим свойствам. Для доступа к этому меню нужно «кликнуть» левую кнопку «мыши» на иконке.

Можно скрыть главную форму (с использованием метода Hide или свойства Visible := False) без остановки приложения. В этом случае важно обеспечить иконку tray контекстным меню или определить событие (например, OnMouseClick), чтобы снова передать управление главной форме приложения.



Фигура 33. Tray Icon

VCL – границы и выравнивание

Помимо улучшений существующих и добавления новы компонентов в VCL, также появились три новых класса:

- **TMargins**
- **TPadding**
- **TCustomeTransparentControl**

Теперь в класс TConrol добавлены дополнительные свойства (Margins, класс TMargins). TMargins используется в свойстве Margins класса TConrol и его потомках. TMargins помогает задавать относительное взаиморасположение между компонентами на форме и границами формы и компонентами. Например, если установить левое поле (margin) для компонента как 10 пикселей, никакой из компонентов не сможет расположиться ближе, чем на 10 пикселей к границе контейнера или другого компонента по левой границе. Количество пикселей, которые разделяют два компонента, есть сумма пикселей полей этих компонентов.

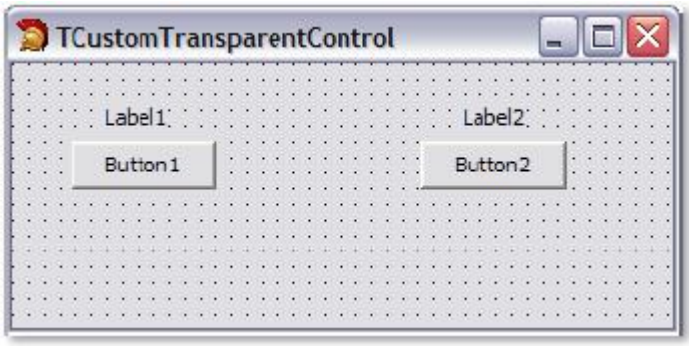
Компонентам класса TWinControl добавлено новое свойство – Padding, класс TPadding, потомок TMargins.

Padding добавляет пространство вдоль границы элемента управления. Дочерние компоненты выравниваются относительно родительского внутри последнего в соответствии с этим свойством. Свойство Padding не воздействует на дочерние элементы управления, которые не выровнены (aligned) относительно родительского компонента, а также размер ClientArea.

Свойство Padding по своей сути противоположно Margins. Margins воздействует на расположение элементов управления внутри родительского, тогда как Padding определяет выравнивание дочерних элементов управления относительно родительского.

Прозрачные элементы управления (Transparent Controls)

Новый класс TCustomTransparentControl может быть использован для компонентов, которые нужно временно скрыть. Хорошей метафорой здесь является стеклянная дверь или окно. Такие компоненты будут присутствовать, но визуально определить их наличие будет нельзя. Чтобы почувствовать разницу нужно выполнить следующий тест: создайте новое VCL-приложение (File | New | VCL Forms Application – Delphi for Win32). Разместите две кнопки TButton и две метки TLabel на форме.



Фигура 34. Прозрачные элементы управления

Обеим кнопкам установите свойство Top как 40; для свойства Left компонента Button 1 и Button 2 как 30 и 210, соответственно. Далее нужно ввести код, как показано ниже. Заметьте, что форма реализует события OnCreate, OnDestroy и OnClick, а обе кнопки имеют одинаковое событие OnClick.

Здесь созданы: потомок TCustomTransparentControl – TtransparentControl и потомок TCustomControl – TOpaqueControl. Они создаются динамически в событии OnCreate формы и располагаются над существующими элементами управления. Также добавлено событие OnClick для демонстрации их поведения. Результат показан на Фигуре 35.

Фигура 35. Пример прозрачного элемента управления (Transparent Control)

```
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;
```

Пользовательское сообщение CM_INPUTLANGCHANGE было добавлено в компоненты VCL для уведомления, когда изменяется язык в операционной системе, так что это может быть учтено в приложении без перезапуска операционной системы.

Иконки (icons) могут теперь быть ассоциированными с растровыми изображениями (bitmaps), а TImage поддерживает TIFF.

Новый выпадающий список в Month Calendar для задания свойства типа «дата» в Инспекторе объектов (Object Inspector) помогает задавать значения соответствующих свойств, также как новый редактор свойство для значений типа Boolean.

Другие новые возможности, реализованные в Delphi: изменение имен компонентов TCategoryBottons, метод CheckAll(cbUnchecked, True, True) для TCheckListBox, функция PtInCircle в модуле “Types” (подобная “PtInRect”), свойство ActiveLineNo, которое возвращает правильную позицию курсора в компоненте TRichEdit и новый модуль IOUtils.pas, реализующий 3 новых статических класса TDirectory, TPath и TFile, которые содержат много полезных статических методов для ввода/вывода.

Кроме того, есть еще десятки улучшений в VCL, которые здесь не описаны, но информацию о которых можно получить посредством использования справочной системы.

В завершение этого раздела следует сказать, что некоторые методы VCL стали inline-методами и, следовательно, их производительность возросла, что также благоприятно сказалось на быстродействии всей библиотеки VCL.

Поддержка IntelliMouse

IntelliMouse – так теперь называется поддержка прокрутки при помощи колеса «мыши» в разрабатываемых приложениях. Эта технология была впервые введена в VCL версии Delphi 2006. Для ее использования нужно задекларировать модуль IMOUSE в коде приложения.

TObject

«Прородитель» всех компонентов в Delphi также был расширен:

- Новые методы
  - Class function UnitName: string
  - Function Equals(Obj : TObject) : Boolean; virtual
  - Function GetHashcode : Integer; virtual
  - Function ToString; virtual

- Несколько дополнительных перегрузок следующих методов
  - MethodAddress
  - FieldAddress

- Тип возвращающих значений функций, представленных ниже, изменен с ShortString на String для поддержки Unicode
  - ClassName
  - MethodName

Другие компоненты – TPanel, TProgressBar, TTrayIcon, TScreen и TRadioGroup также содержат множество улучшений.

Новый менеджер памяти и новые функции RTL

Многие функции RTL были улучшены для повышения производительности. Использование FASTMM является одним из ключевых этих улучшений. Новый менеджер памяти FASTMM для приложений Win32 дает возможность приложениям показывать более высокую производительность, начиная с Delphi 2006, и идентифицирует утечки памяти простой декларацией ReportMemoryLikeonShudown := True в любой части кода программы. Естественно, рекомендуется добавлять данную строчку в разделе initialization.

Сложно переоценить факт того, что в Delphi 2006 и более поздних выпусках приложения работают существенно быстрее в дополнение к возможности определять утечки памяти.

Поддержка клиентов SOAP 1.2

В Delphi 2010 введена поддержка клиентов SOAP 1.2 при помощи компонента THTTPRIO. В Delphi XE компонент THTTPRIO экспонирует новые свойства, что позволяет разработчику выбирать Client Certificate в design-time.

Регулярные выражения (Regular Expression)

В Delphi XE введена поддержка регулярных выражений (модуль RegularExpressions). Регулярные выражения обеспечивают прямые и гибкие средства проверки соответствия строк текста, такие как частичные совпадения символов, слов или шаблонов символов.

Следующий пример показывает, как использовать регулярные выражения для проверки корректности IP-адресов.

```
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;
```

Если передать параметр 200.100.2.21 (корректный IP), то результатом будет:

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

Классы для объектно-ориентированного ввода/вывода в файлы и директории

В Delphi 2010 появился новый модуль IOUtils, содержащий три статических класса: TDirectory, TPath и TFile. Эти классы экспонируют ряд статических методов, полезных для задач ввода/вывода. Большинство методов по функциональности и сигнатуре совместимо с классами .NET: System.IO.Directory, System.IO.Path и System.IO.File.

Следующий код демонстрирует, как следует выполнять чтение всех файлов в папке.

```
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

Одной из наиболее амбициозных задач перед командой разработчиков Delphi оказалась поддержка Unicode на уровне языка, VCL, RTL и IDE, а также всей экосистемы сообщества разработчиков, партнеров и пользователей.

При подготовке процесса перехода на Unicode было проведено множество встреч с программистами, преподавателями, консультантами и технологическими партнерами. Основной целью данных рабочих встреч было подготовить наших технологических партнеров к тому, что их разработки должны были полностью соответствовать Unicode-версиям Delphi, а программистам показать, как они смогут работать с Unicode.

Unicode представляет собой стандарт, позволяющий получить компьютерное представление и работать с любой системой записи.

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

Многие наборы символов, такие как китайский, японский и русский, наряду с другими азиатскими вариантами, представимы при помощи Unicode. Наиболее часто употребимые

кодировки – UTF (Unicode Transform Format) и UCS (Universal Character Set). Чтобы получить подробную информацию по Unicode, следует обратиться по адресу: http://en.wikipedia.org/wiki/Unicode.

Результатом всего этого явилось то, что теперь Delphi поддерживает Unicode на 100%. Возникает вопрос, а на сколько проста миграция? Миграция достаточно проста, а VCL и компилятор хорошо обрабатывают сложные ситуации. В случае необходимости получить исчерпывающую информацию по проблеме перехода на Unicode, рекомендуется проработать технический документ под названием “Delphi Unicode Migration for Mere Mortals”, которая доступна по адресу http://edn.embarcadero.com/article/40307.

Одним из главных изменений в современных версиях Delphi является то, что теперь строковые типы основаны на UNICODE. AnsiString и WideString на основе стандарта ANSI продолжают работать как и ранее с учетом размера строк в байтах.

Краткий список изменений для поддержки Unicode:

- String теперь означает UnicodeString, а не AnsiString
- Char теперь означает WideChar (2 байта, а не 1), представляющий собой символ UTF-16
- PChar означает PWideChar
- AnsiString означает “старый” тип String

Изменения не коснулись:

- AnsiString
- WideString
- AnsiChar, PAnsiChar
- Short string содержит элементы AnsiChar
- Неявное преобразование продолжает работать.

Активная кодовая страница управляет режимом (ANSI или Unicode), так что строки ANSI по-прежнему поддерживаются.

Операции, которые не зависят размера символа:

- Конкатенация строк
- Стандартные функции для работы со строками. Например, Length, Copy, Pos и т.д.
- Операторы. Например, <string> <comparison> <string>, CompareStr(), CompareText(), etc.
- FillChar (<struct or memory>)
- Windows API

Операции, которые подразумевают использование размера символа в байтах, могут потребовать изменений. Никаких особых сложностей нет, но главным советом является следующее: уделяйте особое внимание коду, который:

1. Подразумевает, что Sizeof(Char) возвращает 1.
2. Подразумевает, что размер строки равен числу байтов в строке.
3. Прямое обращение String или PChar
4. Сохранение или чтение строки из/в файл.

Пункт 1 и 2 не будут работать в Unicode, т.к. Sizeof(Char) есть 2 байта, а размер строки в байтах в два раза больше ее длины. Кроме того, программный код, который выполняет сохранение и чтение из файлов, должен оперировать правильным числом байтов для выполнения операции, т.к. символ больше не представляется 1 байтом.

Миграция не представляет особых проблем в большинстве случаев. Основным преимуществом от поддержки Unicode является то, что разработчики могу распространять свои приложения по всему миру. Бразилия представляет собой одну из самых больших территорий, где производится создание программного обеспечения для глобального рынка. Многие бразильские компании распространяют свои приложения и/или обмениваются информацией с Китаем, Японией, Россией и другими странами, где поддержка Unicode является критической.

В 2007 году правительство России приобрело 1 миллион лицензий Delphi для использования в обучении в начальных и высших школах программированию на Delphi. Следовательно, поддержка Unicode жизненно необходима в этой стране.

Новые языковые возможности и ресурсы компилятора

Расширенная RTTI

Система динамической идентификации типов данных RTTI обеспечивает информацию об объектах, таким образом, позволяя взаимодействовать объектам в приложении. Сама IDE в Delphi является одним большим примером использования RTTI, когда мы используем инспектор объектов, редактор кода, средства моделирования и другие ресурсы.

Эволюция и развитие других языков программирования изменили методы и способы кодирования. Приложения, созданные при помощи Java и .NET, хорошо иллюстрируют эти нововведения, когда современный язык программирования предоставляет новый уровень динамического взаимодействия. Вместе с расширенной поддержкой RTTI в Delphi 2010, Delphi for Win32 в данный момент обладает всей мощью .NET и Java. Новая система RTTI (RTTI.pas) является полностью объектно-ориентированной и позволяет создавать и обеспечивать взаимодействие между объектами более динамичным способом.

Атрибуты (Attributes)

Как известно, техника применения пользовательских атрибутов широко применяется программистами Java и .NET. Пользовательские атрибуты представляют собой очень интересную возможность, которая успешно применяется в различных фреймворках, таких как Hibernate, .NET и Java.

Атрибуты позволяют определять классы и особые свойства для их элементов. Есть несколько примеров того, как могут быть полезны атрибуты. Самый лучший пример использования атрибутов – создание фреймфворка “объектно-реляционного отображения” (O/R Mapping). Атрибуты определяются обязательным образом в классах. Классы атрибутов наследуют TCustomAttribute. Приемер: TableAttribute – есть класс, который будет использован для отображения классов приложения на базу данных, а свойство TableName задает название таблицы.

```
1. TableAttribute = class(TCustomAttribute)
2. private
3.   FTableName: string;
4.
5. public
6.
  { TRttiType can be used as a parameter type; TypeInfo() supplies the argument. }
7.   property TableName: string read FTableName;
8.   constructor Create(ATableName: string); overload;
9. end;
10.
11. implementation
12.
13. constructor TableAttribute.Create(ATableName: string);
14. begin
15.   FTableName := ATableName;
16. end;
```

Зададим таблицу отображения атрибутов и будем использовать ее в своих классах.

Например: класс TClient реализует отображение на таблицу CUSTOMER (CLIENTE) базы данных.

```
1. [Table('CLIENTE')]
2. TCliente = class
3. public
4.   { public declarations }
5.   property Nome : String read FNome write FNome;
6.   property Endereco : String read FEndereco write FEndereco;
7. end;
```

В этом примере мы используем атрибуты для отображения только класса, но мы также можем отобразить переменные, свойства, параметры методов и т.д.

Атрибуты пользователя были добавлены в Delphi 2010 и помогли выйти языку программирования на новый уровень, открыв перспективу дальнейшего развития. Уже сейчас можно использовать всю мощь, предоставляемую атрибутами.

Функция Exit

Функция Exit теперь может принимать параметр, означающий результат. Параметр функции System.Exit должен быть такого же типа, что и возвращаемое результирующее значение.

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

Директива Inline

Компилятор Delphi позволяет маркировать процедуры и функции директивной inline, наличие которой повышает быстродействие. Если процедура или функция удовлетворяет определенным критериям, то компилятор вставляет код непосредственно в точку вызова, а не генерирует обычный вызов. Метод “подстановки в строку” (inlining) позволяет оптимизировать производительность, что порождает более быстрый код, однако за счет увеличение его размера. Бинарный файл, генерируемый компилятором, в этом случае будет больше по размеру. Директива inline используется при декларировании и определении функций или процедур, как любая другая директива.

```
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.
```

Когда выполняется код, представленный выше, метод Max вызывается 10 миллионов раз. Показанные ниже цифры могут варьироваться в зависимости от конкретного компьютера. При использовании Pentium M 1.8 ГГц с оперативной памятью 2 Гб, получены следующие результаты:

С директивой **inline**	Без директивы **inline**
25 миллисекунд	68 миллисекунд

Директива inline носит рекомендательный характер для компилятора. Не существует абсолютных гарантий того, что компилятор сделает iniline-подстановку для конкретной

функции, т.к. есть ряд обстоятельств, когда строчная подстановка не может быть осуществлена. Следующий список показывает условия, когда inline-подстановка не будет выполнена:

- Подстановка в строку не будет осуществлена в любом случае метода позднего связывания: virtual, dynamic или message.
- Процедуры или функции, содержащие ассемблерный код, никогда не будут подставлены в строку.
- Конструкторы и деструкторы никогда не будут подставлены в строку.
- Основной программный блок, раздел initialization и раздел finalization не могут быть подставлены в строку.
- Процедуры или функции, которые не объявлены заранее, не могут быть подставлены в строку.
- Процедуры или функции, которые принимают параметры open array, не могут быть подставлены в строку.
- Код внутри пакета (package) может быть подставлен в строку, однако действие данной директивы не преодолевает границы пакетов (packages).
- Подстановка в строку невозможна, если модули связаны кольцевой зависимостью. Это также включает непрямые кольцевые ссылки, например, модуль A использует модуль B, а модуль B использует модуль C, который, в свою очередь, использует модуль A. В этом примере, при компиляции модуля A, никакой код ни из модуля B, ни из модуля C, не будет подставлен в строку в модуле A.
- Компилятор может выполнять подстановку в строку, когда модуль связан кольцевой зависимостью, если код, подлежащий подстановке, берется из не вовлеченного в зависимость модуля. В примере, представленном выше, если модуль A использует модуль D, код из модуля D может быть подставлен в модуль A, т.к. он не вовлечен в кольцевую зависимость.
- Если процедура или функция определена в секции interface и осуществляет доступ к чему-либо, определенному в разделе implementation, эта процедура или функция не может быть подставлена в строку.
- Если процедура или функция обозначена как inline и использует что-то из другого модуля, все эти модули должны быть перечислены в разделе uses, иначе процедура или функция не может быть подставлена в строку.
- Процедура или функция, в которой используется условные выражения в операторах while-do и repeat-until, не могут быть подставлены в строку.
- Внутри модуля тело inline-функции должно быть определено перед ее вызовом. В противном случае, тело функции, которое не известно компилятору в момент обработки точки вызова, не будет подставлено inline.

Если производится модификация реализации inline-процедуры или функции, то это приведет к перекомпиляции всех модулей. Это несколько отличается от традиционной практики перестроения всего проекта, когда перестройка вызвана изменением только интерфейсной части модуля.

Перегрузка операторов

Delphi позволяет перегружать определенные функции или “операторы” в рамках декларации записей (records). Название функции оператора соответствует символьному представлению исходном коде. Например, оператор Add соответствует символу +. Компилятор генерирует вызов перегруженного варианта, соответствующему данному контексту (например, типу

возвращаемого значения, типу параметров, используемых в вызове) с точки зрения сигнатуры операторной функции.

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

Ниже представлен пример работы с операторами складывания, вычитания, а также операторами явного и неявного преобразования:

```
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;
```

Никакие другие операторы помимо тех, что перечислены в таблице, не могут быть определены в записи.

Перегруженные методы операторов не могут быть вызваны по имени в исходном коде. Для доступа к специфическому операторному методу специфического класса или записи, нужно выполнить явное преобразование типов над операндами. Идентификаторы операторов не включаются в список методов записи. Нельзя делать никаких предположений относительно дистрибутивных или коммутативных свойств операции. Для бинарных операций, первый параметр всегда есть левый операнд, а второй параметр – всегда правый операнд. Ассоциации подразумеваются «слева направо» в отсутствии явных скобок.

Разрешение операторных методов делается по совокупности доступных операторных типов, используемых в операции (заметим, что это включает унаследованные операторы). Для операций, подразумевающих два различных типа A и B, если тип A имеет неявное преобразование в тип B, а тип B имеет явное преобразование в A, то возникнет неоднозначность (ambiguity). Неявное преобразование нужно реализовывать только в случае абсолютной необходимости и стараться избегать рефлексивности. Самое лучше, это разрешить неявное преобразования типа B в тип A, а тип A не должен ничего знать о типе B (или наоборот).

Главным правилом является следующее: операторы не должны изменять операнды. Вместо этого, они должны возвращать новое значение, созданное в процессе выполнения операции над параметрами. Перегруженные операторы чаще всего используются в записях (т.е. value types).

“Помощники” классов (Class Helpers)

“Помощник” класса представляет собой тип, который, будучи ассоциированным с другим классом, вводит дополнительные методы и свойства. Эти методы и свойства могут быть использованы в контексте ассоциированного класса (или его потомков). Помощники класса представляют собой способ расширить класс без наследования. Помощник класса просто вводит более широкую область для компилятора при разрешении имен идентификаторов.

Когда происходит декларация помощника класса, нужно указать имя помощника, а также имя класса, который будет расширен при помощи помощника. Можно использовать помощника класса в любом месте, где можно сделать расширение класса классическим способом. При разрешении область включает в себя базовый класс, а также помощника класса.

Помощники классов обеспечивают еще один способ расширить класс, но их нельзя рассматривать в качестве средства проектирования, которое следует использовать при разработке объектно-ориентированного кода. Их следует использовать только по прямому назначению, т.е. для связи языковой и платформенной RTL. Ниже представлен наглядный пример.

```
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.
```

Обратим внимание, что ссылка всегда указывает на TMyClass. Компилятор сам понимает, когда следует вызов из TMyClassHelper.

Strict Private и Strict Protected

В Delphi есть две опции, которые определяют видимость атрибутов класса: strict private и strict protected.

- **Strict private**: атрибуты класса являются видимыми только внутри класса, в котором они декларированы. Эти атрибуты не видны из методов, декларированных в том же модуле, или тех, которые не являются частью класса.
- **Strict protected**: определяет, что атрибуты класса доступны в потомках.

Записи поддерживают методы

Тип данных “запись” (record) в Delphi представляет собой смесь или набор элементов. Каждый элемент называется “полем” (field), и в декларации типа “запись” должны содержаться название и тип каждого поля.

Начиная с Delphi 2006, записи получили новые возможности, которые раньше были доступны только в классах. Здесь представлен список новых возможностей записей в Delphi 2006:

- Конструкторы
- Перегрузка операторов
- Декларация невиртуальных методов
- Статические методы и свойства

Следующий пример показывает использование записи с новыми возможностями:

```
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;
```

Теперь записи могут пользоваться функциональностью, ранее считавшейся эксклюзивной привилегией классов.

Однако записи всё-таки не классы, поэтому сохраняются некоторые различия:

- Записи не поддерживают наследование
- Записи могут иметь вариабельные части; классы – нет.
- Записи являются типом данных, поэтому их можно копировать; классы – нельзя.
- У записей не может быть деструкторов.
- Записи не поддерживают виртуальные методы.

Class Abstract, Class Sealed, Class Const, Class Type, Class Var, Class Property

Существует много способов декларации классов, типов, переменных и свойств.

- Class abstract → означает абстрактный класс
- Class sealed → классы не могут расширяться за счет наследования – такие классы нельзя использовать в качестве базовых для других (производных)
- Class const → означает, что константа является «классовой», т.е. к ней можно получить доступ без создания экземпляра
- Class type → означает, что к классовому типу можно получить доступ без создания экземпляра
- Class var → означает, что в области класса к переменной можно получить доступ без создания экземпляра
- Class property → дает доступ к свойству без необходимости создавать экземпляр

Вложенные классы

Вложенные типы используются повсеместно в объектно-ориентированном программировании. Они позволяют хранить концептуально связанные типы вместе и, таким образом, избегать коллизий имен. Именно такой синтаксис можно использовать и в Delphi. Пример такого класса представлен ниже:

```
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;
```

Следующий пример иллюстрирует доступ к методу вложенного класса:

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

Методы final

Компилятор Delphi также поддерживает концепцию «финальных» виртуальных методов (final virtual methods). Когда виртуальный метод маркируется ключевым словом final, в производных классах данный метод не может быть переопределен. Использование ключевого слова final является важным решением с точки зрения проектирования, что является прямым указанием о назначении класса. Кроме того, это является подсказкой компилятору для оптимизации генерируемого кода.

Методы Static Class

Когда классы помечены как Static, для их использования не нужно создавать экземпляр.

For … In

Начиная с Delphi 2007 была введена поддержка с новым стилем кодирования для итерации вида «элемент-в-коллекции» при использовании контейнеров.

Пример: итерации на массиве

```
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;
```

Пример: итерация на строке

```
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

Поддержка generics была введена, начиная с Delphi 2009.

«Generics» - это термин для обозначения родовых типов. Он подразумевает использование языковой конструкции для предварительного указания типа данных в массиве, коллекции или другом каком-либо контейнере. С использованием generics можно написать обобщенный

код, который будет работать конкретным типом данных – классами или классовыми методами. Также возможно задавать типы во время исполнения.

При использовании generics чаще всего приходится иметь дело с Collections. Delphi RTL обеспечивает ряд уже готовых коллекций (определенных в модуле “Generics.Collections”), включающих:

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

Если связь элемента коллекции с классом добавляемого элемента происходит по TObject, то контейнер будет работать только на объектах одного типа, что также требует приведения типов для каждого элемента. Это дает дополнительную нагрузку на структуру кода приложения, скорость компиляции и качество приложения. Значительная часть такого кода может быть переработана на использование generics. Ниже приводится пример класса, когда его свойство Key является строкой, а свойство Value – целочисленной переменной. В данном случае generics не используются.

```
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;
```

С использованием generics можно задать свойства Key и Value как свойства любого типа.

```
type
    TPair<TKey,TValue>= class // declares the TPair type with 2 parameters
private
    FKey: TKey;
    FValue: TValue;
```

```
type
    TPair<TKey,TValue>= class // declares the TPair type with 2 parameters
private
    FKey: TKey;
    FValue: TValue;
```

Теперь можно использовать данный класс различными способами:

```
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>;
```

Generics находят массу полезных применений, а показанные выше примеры лишь иллюстрируют принцип их применения ограниченным образом.

Анонимные методы

Как следует из названия, анонимные методы представляют собой процедуры или функции, которые не имеют ассоциированного имени. Анонимный метод есть блок кода, который может быть ассоциирован с переменной или быть использован в качестве параметра для другого метода. Кроме того, анонимные методы могут использовать переменные из контекста, где определен метод. Объявление и использование анонимных методов не требует сложного синтаксиса. Их синтаксис аналогичен конструкции замыканий (closures), характерных для других языков программирования.

Пример:

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

Функция MakeAdder возвращает безымянную функцию, т.е. анонимный метод. Тип анонимного метода декларируется как ссылка на метод.

Пример: выполнение функции MakeAdder

```
var
    adder: TFuncOfInt;
begin
```

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

Показанная выше конструкция означает, «анонимный метод»:

- представляет собой функцию
- получает целочисленное значение
- возвращает целочисленное значение

В качестве анонимных методов могу выступать и процедуры, и функции.

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

Перехват виртуальных методов (Virtual Method Interception)

Delphi XE определяет новый тип в Rtti.pas, названный TVirtualMethodInterceptor. Он динамически во время исполнения создает производный метакласс, который переопределяет любой виртуальный метод в «предке» за счет создания новой таблицы виртуальных методов и распространения её с заглушками. Эта таблица перехватывает вызовы и аргументы. Когда ссылка на метакласс для любого экземпляра «предка» заменяется этим новым метаклассом, пользователь может перехватить вызов виртуальных функций, изменять аргументы «на лету», изменять возвращаемое значение, перехватывать и подавлять исключения или возбуждать новые исключения, а также полностью заменять вызов основных методов. Концептуально это похоже на динамические «прокси» в .NET или Java. Это сродни наследованию класса во время исполнения, переопределение методов (но не добавление новых полей) и, таким образом, изменение типа экземпляра во время исполнения до нового производного класса.

Рассмотрим случай, когда данный подход может оказаться полезным. Тут следует запомнить две очевидные вещи: тестирование и удаленный доступ. «Объекты-заглушки» (mock objects) эффективно используются при тестировании в других языках. При перехвате вызовов методов можно более легким способом проверить, что конкретная подсистема вызывает правильные методы с корректными аргументами в правильной последовательности; подобным же образом подсистема осуществляет возврат значений после вызова этих методов без необходимости обращения к базе данных, сетевым ресурсам и т.д., для чего и служат техники модульного тестирования. Удаленный доступ на основе вызова методов не всегда эффективен, особенно когда сетевой доступ ненадежен или перегружен защитой, но это не единственный вариант использования. Логика перехвата виртуальных методов была изначально реализована для использования как часть механизма аутентификации в

технологии DataSnap, так что вызов, исходящий из сети, можно проверить, т.к. поток выполнения кода распределен по графу объектов.

В любом случае, здесь для начала приводится простой пример:

```
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.
```

Ниже представлен вывод на экран:

```
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:
```

Примечание: в данном примере идет перехват всех виртуальных методов, включая тех, которые вызываются при разрушении объекта, а не только те, которые были задекларированы. (Сам деструктор не включен).

Новая директива $POINTERMATH {ON - OFF}

Директива $POINTERMATH делает возможным математические операции с указателями.

Новые предупреждения

Когда происходит компиляция приложения в Delphi XE, то могут появиться новые предупреждения в окне «message» IDE. Эти предупреждения связаны с использованием нового типа UnicodeString. Эти новые предупреждения включают:

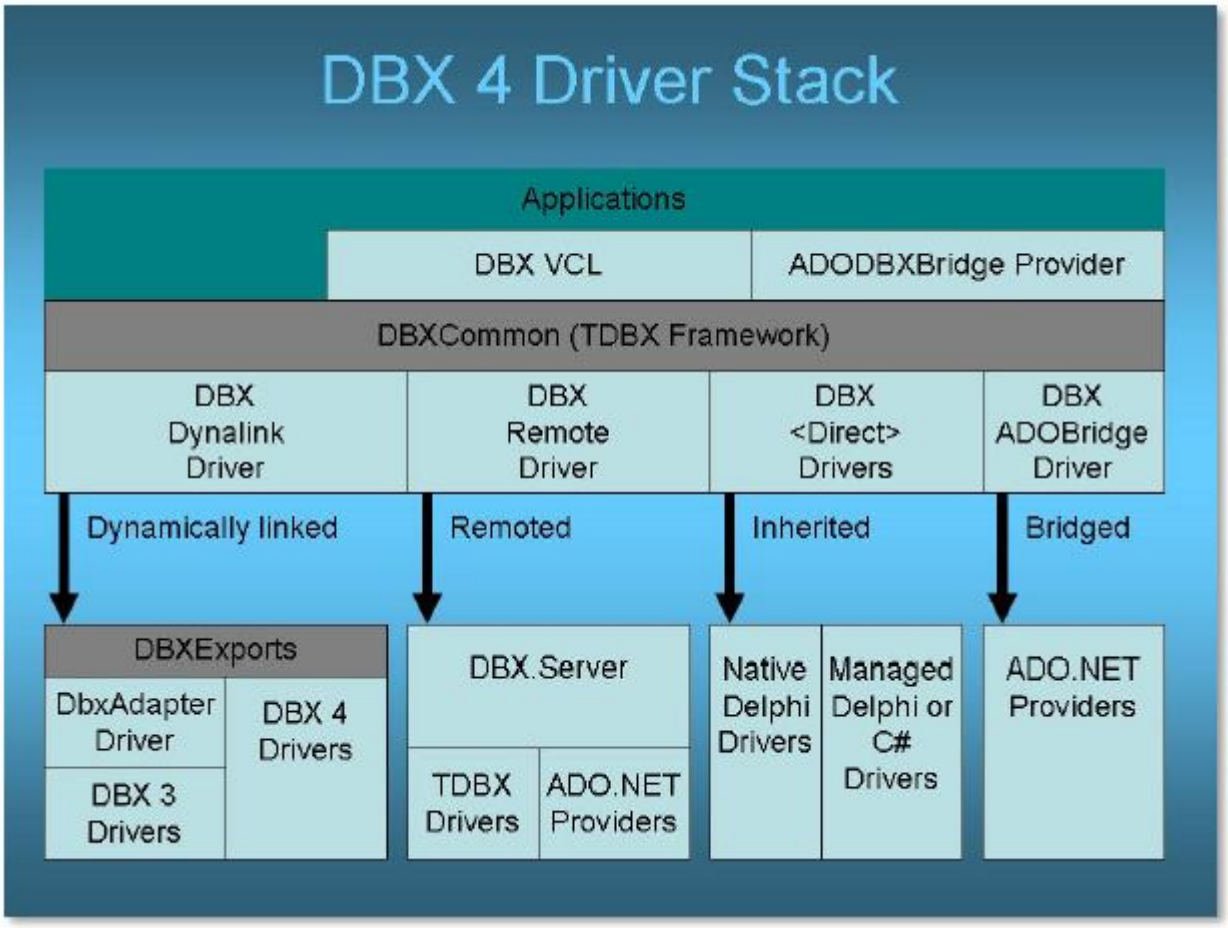
- 1057 Implicit string cast from ‘%s’ to ‘%s’ (неявное преобразование строк)
- 1058 Implicit string cast with potential data loss from ‘%s’ to ‘%s’ (неявное преобразование строк с возможной потерей данных)
- 1059 Explicit string cast from ‘%s’ to ‘%s’ (явное преобразование строк)
- 1060 Explicit string cast with potential data loss from ‘%s’ to ‘%s’ (явное преобразование строк с возможной потерей данных)

dbExpress

Фреймворк

Одним из наиболее значительны улучшений в Delphi 2007 явилось то, что архитектура dbExpress была переделана и теперь может считаться фреймворком, который полностью разработан на Delphi. Были проведены тесты для моделирования различных ситуаций на множестве платформ баз данных. Результаты некоторых тестов показали повышение производительности на 100%.

DbExpress 4 также является краеугольным камнем для приложений, созданных в Delphi и предназначенных для работы с базами данных. Новая архитектура была разработана для поддержки и Win32, и .NET, позволяя использовать те же самые драйвера для обеих платформ, например, драйвер dbExpress DataSnap может быть использован и для Win32, и для .NET. Это позволяет создавать клиентские приложения и в Delphi Win32, и в Delphi Prism (.NET).



Фигура 36. Архитектура dbExpress 4

Обратите внимание, что приложения, созданные в более ранней версии Delphi, являются на 100% совместимыми с Delphi XE в этой части.

Фреймворк dbExpress содержит в себе новую группу классов, которые обслуживают задачу доступа и работы с базами данных. Теперь можно найти информацию о работе с базами данных внутри данного фреймворка. Ранее можно было вместо этого только использовать компоненты SQLConnection, SQLDataSet, SQLQuery и другие.

Следующий пример демонстрирует консольное приложение, которое использует ресурсы соединения с базой данных, выполняет чтение параметров соединения, посылает запрос и отображает его результаты – всё это в рамках одной транзакции.

```
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.
```

Метаданные в dbExpress

Новая поддержка метаданных широко используется сервисом Data Explorer в IDE Delphi, однако она также может быть использована в любом приложении. Кратко, можно не только просматривать структуру базы данных, но также использовать классы и объекты для её модификации вместо того, чтобы полностью полагаться на использование прямых SQL-команд для решения подобных задач. В этом случае код не только становится более объектно-ориентированным, но можно использовать один и тот же исходный текст для разных платформ база данных, т.к. dbExpress абстрагирует функции работы с метаданными для каждого сервера.

Модуль DBXMetaDataNames служит для чтения метаданных. Класс TDBXMetaDataCommands из dbExpress содержит набор констант для чтения различных типов метаданных. Для выполнения этой операции нужно установить свойство TDBXCommand.CommandType в значение DBXCommandTypes.DBXMetadata а также TDBXCommand.Text в значение одной из констант в TDBXMetaDataCommands, чтобы получить нужные нужных метаданные, а TDBXCommand.ExecuteQuery вернет TDBXReader для доступа к метаданным. Новые классы в DBXMetaDataNames описывают и дают доступ к нужным колонкам метаданных. Ниже представлен список метаданных, которые можно получить с использованием dbExpress:

- Тип данных
- Таблицы
- Колонки (для таблиц, просмотров и т.д.)
- Индексы
- Поля по индексам
- Внешние ключи
- Хранимые процедуры
- Параметры хранимых процедур
- Список пользователей

- Каталоги
- Схемы
- Просмотры
- Синонимы
- Исходный код хранимых процедур
- Упакованные хранимые процедуры
- Исходный код упакованных хранимых процедур
- Параметры упакованных хранимых процедур
- Роли
- Зарезервированные слова

Data Explorer включает поддержку создания SQL запросов, чувствительных к диалекту, для CREATE, ALTER и DROP. DbExpress также экспонирует класс DbxMetaDataProvider, который обеспечивает легкий доступ к данной функциональности из приложений. Это несколько увеличивает размер приложения, т.к. должны быть включены объекты для чтения метаданных. Возможность генерировать таблицы родственным образом является полезной для многих приложений. Интерфейс позволяет описывать, что собой представляет таблица и ее колонки, и передавать это описание в метод TdbxMetaDataProvider.CreateTable.

Следующий пример показывает, как нужно создавать таблицы, первичные ключи, внешние ключи и индексы с использованием классов фреймворка dbExpress.

```
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);
```

Исходный код данного примера доступен по адресу http://cc.embarcadero.com/Item/26210.

Драйверы dbExpress

Поддержка последних версий баз данных: InterBase 2009, MySQL 5.1, Oracle 10g/11g, новый драйвер dbExpress для SQL Server 2008 включает поддержку нового типа данных datetime offset. Драйверы для Oracle, InterBase и MySQL теперь поставляются с поддержкой Unicode.

В Delphi 2010 введена поддержка драйвера dbExpress для Firebird, который на тот момент был самым востребованным для разработчиков. Драйвер dbExpress для Firebird поддерживает Firebird версий 1.5 и 2.x, но не только поддерживает, но и является полностью совместимым с Firebird, так что можно создавать таблицы, первичные ключи, внешние ключи, индексы и многое другое при работе с использованием данного фреймворка. Data Explorer также полностью поддерживает Firebird.

Поля BIGINT поддерживаются на 100% и отображаются как TLargeIntField в VCL, а тип BLOB отображается как TBlobField.

Введена новая концепция, названная “Delegate Driver” и “Pools Connections”, которая требует лишь упрощенная конфигурация параметров.

Можно расширить фреймворк dbExpress за счет написания «делегирующих драйверов» (delegation drivers), которые обеспечивают еще один промежуточный слой между приложением и реальным драйвером. Делегирующие драйвера полезны для создания пула соединений (connection pooling), профилирования драйверов, трассировки, а агент аудита DBXTrace представляет собой делегирующий драйвер для трассировки.

Ниже представлен журнал для отображения результатов, полученных от делегата в Delphi. Его легко читать, понять, что там представлено, и еще раз выполнить операции в нем.

Конфигурация трассировки (trace configuration). Следующий пример записывает события в соответствии с параметром TraceFlags в файл журнала c:\dbxtrace.txt. Соединение dbExpress обладает параметром для индикации конфигурации трассировки; например, DelegateConnection=DBXTraceConnection

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

Результаты в сгенерированном журнале:

```
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);
```

Также можно использовать пулы соединений через dbExpress напрямую. Ниже представлен *alias* (Pool_DelegateDemo), передающий DBXPoolConnection контроль над пулом соединений (где задается максимальное число соединений).

```
[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=ne
utral,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
```

Облачные вычисления (Cloud Computing)

Microsoft Windows Azure

Вместе с Delphi XE поставляются компоненты для работы с учетными записями Windows Azure. Windows Azure (не путайте с SQL Azure) позволяет хранить и работать с Blobs, очередями сообщений (Queues of messages) и таблицами данных в облаке Azure. Если создана соответствующая учетная запись, то можно использовать компоненты в разделе Microsoft Azure с палитры компонентов. Доступны следующие компоненты для работы с Azure:

- TAzureTableManagement → таблицы Windows Azure; они обеспечивают масштабируемое структурированное хранение. Тип таблиц NOSQL подразумевает, что каждая хранимая единица в таблице может иметь различный набор свойств различного типа, таких как string или int.
- TAzureQueueManagement → в отличии от blobs и tables, которые используются для хранения данных, очереди (queues) служат другим целям: хранение асинхронных сообщений, где каждое сообщение может быть длинной до 8 кб.
- TAzureBlobManagement → хранилище больших бинарных объектов (blob) Windows Azure; blob-хранилище представляет собой неструктурированное хранилище, т.к. используется для хранения больших порций данных, таких как изображения, видео, документы и т.д. внутри заданного контейнера.

Также есть компонент, называемый TAzureConnectionString, который работает с информацией, необходимой для соединения с учетной записью Windows Azure. Все эти компоненты используют Windows Azure REST API для доступа ко всем возможностям, которые предоставляют эти сервисы.

Компоненты для работы с Azure полезны для разработчиков, которые хотят обеспечить взаимодействия между своими приложениями и учетной записью Windows Azure.

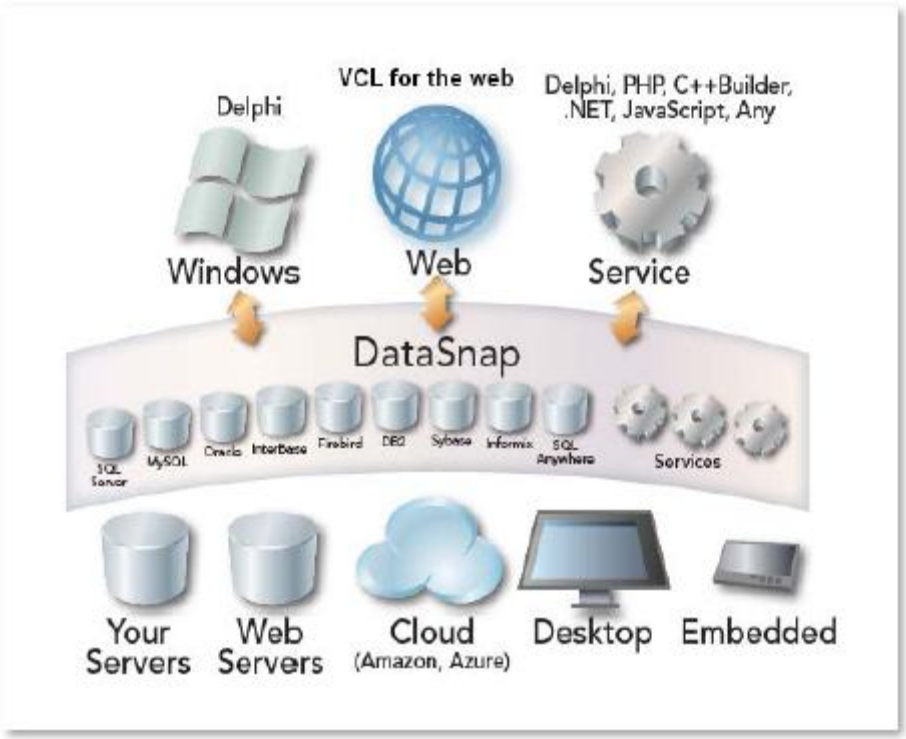
Amazon EC2

Delphi XE содержит в себе мастер для удобного размещения файлов на удаленной машине. Эта удаленная машина может быть частью инфраструктуры Amazon EC2. Следует иметь в виду, что поддерживается экземпляры только EC2 и Windows. Для этого Delphi XE предлагает простой трехступенчатый процесс для удобного размещения в облачной инфраструктуре.

DataSnap

Интеграция между DataSnap и dbExpress является одной из наиболее значимых новых возможностей, появившихся в Delphi 2009. В последующих релизах были добавлены также еще более мощные возможности. На основании отзывов, полученных от разработчиков, был создан принципиально новый DataSnap. В Delphi создание многозвенных приложений всегда было достаточно простой задачей. Однако практика реализации различных проектов показала, что есть направления дальнейшего улучшения данной технологии. В этом разделе мы познакомимся с основными концепциями, которые помогут создать современные многозвенные приложения с использованием DataSnap.

Сегодня DataSnap является ключевой технологией при создании многозвенных приложений, а в релизе XE платформа DataSnap представляет открытую архитектуру для других технологий, использующих вызов серверных методов (в которых бизнес-логика реализуется на стороны сервера). Эти дополнительные поддерживаемы технологии: C++Builder, .NET (Delphi, C#, VB.NET и т.д.), PHP и JavaScript.



Фигура 37. Ключевая технология DataSnap в реализации многозвенных Delphi-приложений

Основные концепции

Новый механизм интеграции между DataSnap и dbExpress, который многие называют «dbExpress для удаленного использования», дает значительную гибкость при создании многозвенных приложений. До его реализации удаленные функции были чем-то, что требовало использовании библиотеки типов (Type Library), а это означало зависимость от COM (как основа для Remote Data Module). Разработчики потребовали исключить зависимость от COM, что и явилось основным атрибутом новой DataSnap: она никаким образом не зависит от технологии COM. Однако данная технология поддерживает совместимость с предыдущим вариантом, которую можно использовать по необходимости.

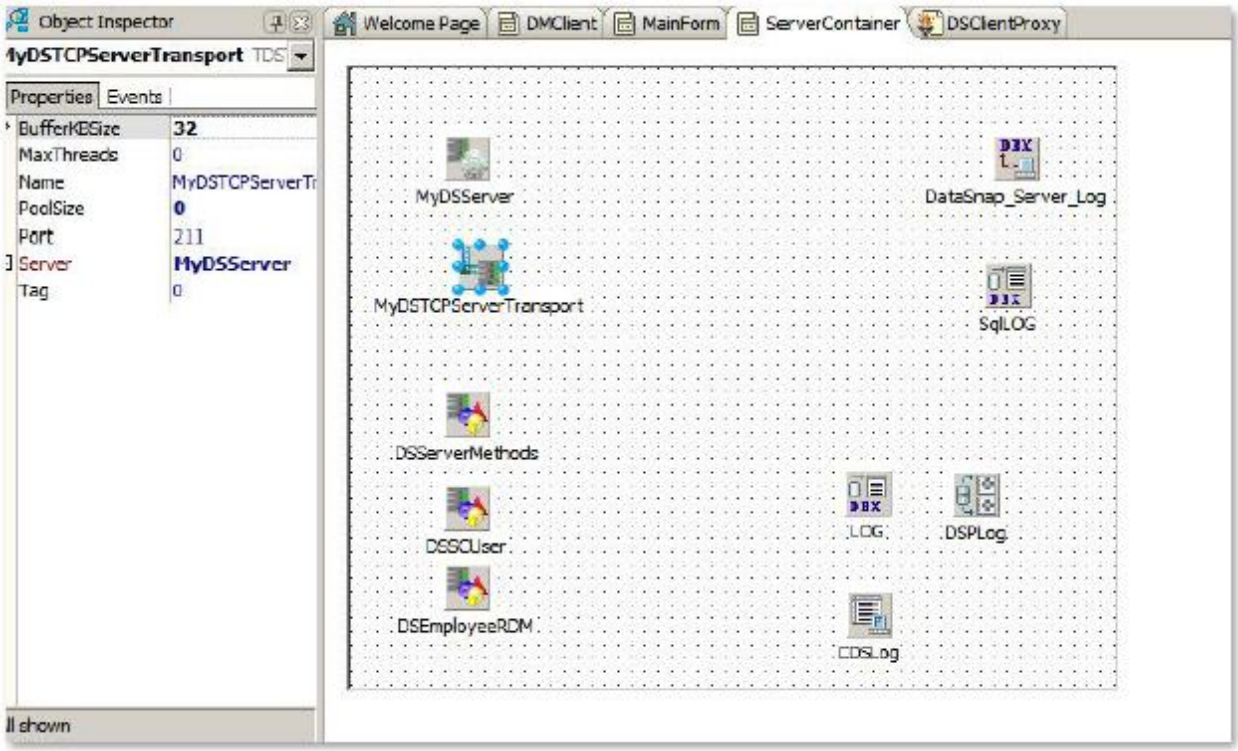
Интеграция DataSnap с dbExpress обеспечивает динамическое выполнение серверных методов, используя, к примеру, SQLDataSet или новый компонент SqlServerMethod. Параметр и результат метода задается при помощи свойства Param.

Одним из способов запуска методов на сервере является использование нового компонента SqlServerMethod. Данный компонент наследует CustomSQLDataSet, что позволяет выполнять серверные методы с использованием DataSet, а входные/выходные параметры представлены в свойстве Params.

DataSnap Server – Server Container

Серверы DataSnap представляются двумя компонентами: DSServer и DSTCPTransport или DSHTTPTransport. Для создания сервера DataSnap достаточно добавить два этих компонента. Тогда форма/модуль данных (data module), на котором размещены эти компоненты, называется Server Container (серверный контейнер). С этого момента будет использоваться новое соглашение об именах. DSServer является сервером DataSnap. Когда он связывается посредством DSTCPTransport, в котором задается порт для соединения, максимальное количество потоков и т.д., то начинается экспонирование приложения в качестве сервера. DSTCPTransort передает информацию посредством TCP, таким образом обеспечивая возможность расширять и создавать новые механизмы транспорта (например, HTTPS, SSL и т.д.). В настоящий момент DSTCPTransport использует инфраструктуру Indy для соединений TCP.

Если используется BSS, то приложение будет продолжать работать в Delphi XE. Однако рекомендуется произвести миграцию на новую DataSnap.



Фигура 38. Серверный контейнер на основе DataModule

DataSnap Server – Server Module

Вполне возможно, что в разрабатываемом приложении уже существует множество классов, которые содержат в себе бизнес-правила и которые было бы эффективнее использовать в многозвенных приложениях. С использованием серверных методов можно легко экспонировать все открытые (public) методы для доступа со стороны клиента.

Для того чтобы сделать доступным серверный метод, нужно:

- Произвести наследование от TPresistent.
- Включить директиву {$MethodInfo ON}. Эта директива позволяет dbExpress получать информацию о классе из RTTI.
- Обеспечить регистрацию с помощью компонента DSServerClass.

Окончательно термин «серверный модуль» (Server Module) используется для идентификации места размещения провайдеров, серверных методов и т.д. Можно создавать серверный модуль через File | New | Other | Delphi Files | Server Module. Server Module – это DataModule, который создается с директивой $MethodInfo ON по-умолчанию.

Каждый класс, который открывается на доступ, ассоциируется с компонентом DSServerClass. Этот компонент отвечает за регистрацию класса и доступ к нему из клиентского приложения. Рекомендуется размещать компоненты класса DSServerClass в серверном модуле. Можно иметь в проекте столько серверных модулей, сколько требуется для поддержания ясной структуры приложения.

Свойство LifeCycle компонента DSServerClass определяет «время жизни» (lifetime):

- Server → один экземпляр компонента используется для каждого сервера (Singleton).
- Session → один экземпляр компонента используется для каждой сессии DataSnap (Statefull).
- Invocation → один экземпляр компонента для каждого вызова метода (Stateless).

Кроме того, DSServerClass имеет несколько событий, которые требуют реализации события OnGetClass для получения зарегистрированного класса. Ниже представлен пример реализации, когда TServerMethods выполняет роль модуля данных, который предоставляет несколько методов:

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

Если в проекте есть Remote Data Module, то тоже можно использовать данную технику.

DataSnap Server – Filters

Может быть использован набор фильтров для применения в процессе обмена информацией между клиентом и сервером DataSnap. Каждый фильтр может выполнять трансформацию потоков байтов, какую как шифрование и/или компрессию; поток байтов может быть перехвачен более чем одним фильтром, и тогда данные на выходе из одного фильтра являются входными для следующего. Фильтры пристыковываются к потоку байтов во время проектирования посредством установки свойства Filters для сервера DataSnap в компонентах DSTCPServerTransport.TDSTCPServerTransport или DSService.TDSHTTPService. Установка фильтров возможна во время проектирования (design time), если они находятся в «пакете» (package), зарегистрированном в RAD Studio. Фильтр должен быть встроен в пакет, а пакет должен быть установлен в Delphi. Если фильтр установлен, то во время разработки он доступен в списке редактора фильтров. На стороне клиента фильтр подключается с использованием TSQLConnection.

Нет необходимости поддерживать фильтры особым образом на стороне клиента, т.к. они автоматически подключаются при взаимодействии на основе протокола «рукопожатия» между клиентом и сервером. Таким образом, важно отметить, что код клиента регистрирует фильтры перед соединением с фильтрующим сервером либо путем добавления соответствующего модуля в раздел uses, либо на ранней стадии, такой как «время инициализации».

В Delphi XE включен фильтр на основе компрессии zlib, когда данные сжимаются автоматически. Также можно задавать фильтры шифрования.

Примеры некоторых фильтров размещены на Code Central по адресу:
http://cc.embarcadero.com/Author/38483.

DataSnap Server – HTTP Tunneling

В DataSnap 2010 появилась новая возможность, которая позволяет разработчикам реализовывать решения в области дублирования, такие как «преодоление отказа» (failover) и распределение нагрузки (load balancing) в приложениях DataSnap. Вопросы в данной области представляют собой частую тему для обсуждений, поэтому данный раздел посвящен именно этому. Здесь содержится информация, которая помощет понять принципы реализации функций для преодоления отказов в приложениях DataSnap наряду с избыточностью для распределения нагрузки.

Как только сервера приложений начинают восприниматься как часть многозвенной архитектуры, разработчики сталкиваются с задачами, такими как централизация процессов, бизнес-правил, определение требований к аппаратному обеспечению, обновления и т.д. При анализе задачи о централизации процессов, также нужно принять во внимание необходимость реализации избыточности, которая предназначена для дублирования критических компонентов системы и увеличения надежности обычно за счет резервирования и механизма преодоления отказов.

До выпуска Delphi 2010, преодоления отказов и распределение нагрузки было сложно реализовать, но теперь ситуация сильно изменилась в лучшую сторону.

В DataSnap 2010 появилась новая возможность, названная HTTP Tunneling («HTTP-туннелирование»), которая обеспечивает контроль над посланными и принятыми данными между клиентом и сервером. HTTP-туннелирование реализуется в рамках протокола HTTP. Таким образом, его можно использовать при обеспечении связи между клиентом и сервером без особых проблем.

Когда реализуется распределение нагрузки или механизм преодоления отказов, то нужно передать контроль над этим связующему программному обеспечению (middleware application). Такое приложение будет отвечать за получение данных от клиентского приложения, их анализ и дальнейшая передача соответствующему серверу.

Переводя эту концепцию на язык DataSnap, можно сказать, что клиентское приложение соединяется с сервером, в функции которого входит обеспечение преодоления отказов. Этот север будет называться «прокси» (proxy), а он, в свою очередь, будет обеспечивать связь с соответствующим сервером DataSnap, когда не удается сохранить связь с первичным севером. Показанный ниже пример моделирует посылку/прием данных с клиентского приложения, в то время как основной сервер DataSnap прекращает работу, а сервер преодоления отказа перенаправляет соединение на вторичный сервер DataSnap.

Для этого нужно лишь внести два изменения в текущее клиентское приложение:

- Выбрать HTTP в качестве протокола связи в SQL Connection
- Обеспечить соединение с сервером преодоления отказов, а не напрямую с сервером DataSnap

Кроме этого, потребуется создать сервер преодоления отказов. Для этого ниже представлен код в качестве иллюстрации начальных шагов для реализации подобной архитектуры.

Реализация сервера преодоления отказов потребует двух компонентов: DSHTTPService, который представляет сервер и соединяется с DSHTTPServiceAuthenticationManager для

процесса аутентификации так, чтобы только авторизованные пользователи могли соединиться с сервером.

Для обеспечения HTTP-тунеллирования нужно будет реализовать следующие события на 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

Эти события срабатывают в течение процесса коммуникации. Названия событий объясняют то, зачем они предусмотрены, а данный пример реализует их все. В примере используется журнал (log), который помогает понять, что происходит в течение процесса коммуникации.

В случае реализации преодоления отказа все события должны быть реализованы. Эти события OnErrorXXX должны выполняться, как только что-то работает не в соответствии с нормальным режимом. Эти события позволяют идентифицировать ситуацию и решить, что делать с передаваемыми байтами информации. Основной фокус направлен на событие OnErrorOpenSession, которое перенаправит соединение в случае ошибки при открытии сессии.

Следующий код реализует метод, названный Redirect, который ассоциирован с событием OnErrorOpenSession. Можно увидеть, как данные в рамках сессии представляются параметрами данного метода, и каким образом они включают все, что необходимо для перенаправления данных.

В этом примере используется Session.UserFlag для обозначения, что соединение было уже перенаправлено. Здесь допускается только одно перенаправление, и в случае, когда параметр Sender становится Exception (исключением), производится сохранение error message (сообщения об ошибке) в журнал. После этого создается экземпляр DBXProperties, который содержит новую информацию о соединении сервера. В данном случае для демонстрационных целей используется то же самый HostName, а порт изменяется на 213, что позволяет отличить его от другого сервера DataSnap на той же самой машине.

После этого сессия переоткрывается за счет передачи новых свойств в качестве параметров, и это – всё, что нужно сделать.

```
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;
```

Если потребуется запустить этот демонстрационный код на конкретной машине, нужно выполнить следующие шаги.

Данный пример включает два проекта: сервер преодоления отказов и сервер DataSnap. В качестве клиентского приложения будет использоваться DataExplorer. Следует выполнить следующие действия для демонстрации работы демонстрационного сервера преодоления отказов.

Открываем и запускаем сервер преодоления отказов, данный пример использует протокол HTTP и порт 8080.

- Запускаем сервер DataSnap дважды, один раз используем порт 211, а другой – 213. Сервер со значением порта 213 будет работать как резервный сервер
- Создаем псевдоним (alias) для DataSnap в Data Explorer, не забывая сконфигурировать его на использование HTTP в качестве протокола и настроить порт на 8080
- В узле Stored Procedure находим метод EchoString, передаем занчение Delphi 2010 в качестве параметра и исполняем его. Возвращаемое значение будет Delphi 2010 (Server 211)
- Закрываем сервер DataSnap, который использует порт 211
- Повторяем шаг 4 и анализируем возвращенное значение, которое должно быть Delphi 2010 (Server 213)
- Теперь смотрим в журнал (log) сервера преодоления отказов (Failover Server) и обнаруживаем текст исключения (exception error) и информацию о перенаправлении

Исходный код примера доступен по адресу http://cc.embarcadero.com/Item/27391.

Безопасность в DataSnap

Как только в сервере приложений появляются серверные методы, возникает потребность реализовать некую логику для ограничения пользователей, которым разрешен вызов данных методов. Для этого следует использовать новую/улучшенную функциональность компонента Authentication Manager.

Чтобы начать его использование достаточно поместить данный компонент на форму и определить его в качестве менеджера аутентификации в компоненте TDSHTTPWebDispatcher, TDHTTPService или TDSTCPServerTransport в зависимости от конфигурации сервера.

Менеджер аутентификации имеет два события: OnUserAuthenticate и OnUserAuthorize. Он также имеет коллекцию “Roles” (роли).

Событие OnUserAuthenticate вызывается, когда пользователь пытается соединиться (вызвать метод) первый раз, и принимает входные параметры соединения, такие как имя пользователя и пароль (user’s name и password), и позволяет задать значение для in/out параметра “valid”. По умолчанию это значение стоит в true, но в зависимости от пользовательской информации или на основании чего-либо другого можно значение valid поставить в true или false. Установка значения данного параметра в false отвергнет соединение пользователя и также запретит вызов любого метода, если такой поступит.

Роли можно задавать различными способами. Можно обратиться к классу серверных методов и добавить атрибут TRoleAuth непосредственно в код (требует модуль DSAuth). Этот атрибут может быть добавлен как на уровне класса, так и на уровне метода, как показано далее:

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

Или так:

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

В первом примере и ‘EchoString’ и ‘ReverseString’ требуют от пользователя обладания ролью “admin” для вызова данных методов. Во втором случае только метод ‘EchoString’ имеет ассоциированную роль “admin”. Следует обратить внимание, что атрибут TRoleAuth имеет опционный второй параметр, который представляет собой список ‘denied roles’ (‘запрещенные роли’), назначение которого понятно из названия. Все эти параметры могут быть списком ролей, разделенных запятыми.

Событие OnUserAuthorize вызывается каждый раз, когда успешно прошедший авторизацию пользователь вызывает серверный метод. Нет необходимости реализовывать это событие. Если добавить роли в список UserRoles, то эти роли сами по себе могут быть использованы для принятия решения, допущен ли пользователь к вызову любого серверного метода. Однако если нужно осуществить более изощренное управление процессом (такое как, например, разрешение или запрещение вызовов в зависимости от времени суток), можно реализовать это событие. Объект, переданный в событие, содержит информацию, такую как имя пользователя (user name), UserRoles в событии аутонтефикации, и разрешенные/запрещенные роли для метода, которые вызывается. Можно использовать данную информацию, как и любую другу, для задания значения “valid” как true или false, что даст или запретит доступ к вызову метода.

DataSnap REST Server

Delphi XE обеспечивает поддержку REST для DataSnap. Другими словами, разработанный сервер DataSnap может экспонировать все серверные методы как REST-интерфейсы. Новая система прокси-генерации в составе XE содержит генераторы для языков программирования Delphi, C++, Delphi Prism, PHP и JavaScript. Эти прокси-генераторы берут на себя всю сложность взаимодействия с REST-сервисами, транслируя методы DataSnap и комплексные типы в реализации на «родном» языке для клиентский приложений.

Для того чтобы создать REST сервер DataSnap, Delphi XE содержит мастер DataSnap REST Application, который создает проект в качестве отправной точки для создания web-приложения на основе AJAX. Коммуникационным протоколом между серверным и клиентским приложением является HTTP, а его архитектура – REST (Representation State Transfer).

Мастер DataSnap REST Application подразумевает четыре или пять шагов в зависимости от типа приложения REST, которое выбрано (на первом шаге).

На первом шаге предлагается выбрать тип приложения REST. Возможные варианты:

- Изолированное приложение VCL
  Изолированное VCL-приложение REST представляет собой web-сервер, который отображает форму VCL. Он поддерживает HTTP с использованием компонента Indy HTTP server.

- Изолированное консольное приложение
  Изолированное консольное приложение REST представляет собой web-сервер, который имеет только текстовой интерфейс. Он поддерживает HTTP с использованием компонента Indy HTTP server.

- Динамическая библиотека ISAPI
  ISAPI и NSAPI приложения для Web-сервера представляют собой разделяемые объекты, которые загружаются web-сервером. Запрашиваемая клиентом информация передается в DLL как структура и обрабатывается TISAPIApplication. Каждое входящее сообщение обрабатывается в отдельном потоке выполнения. Когда выбирается данные тип приложения, в список uses добавляется заголовочный файл проекта библиотеки и все прочие необходимые модули. Кроме того,

формируется раздел экспорта проекта. Библиотеки ISAPI интегрируются с IIS. IIS поддерживает HTTP и HTTPS.

- Исполняемое приложение Web App Debugger
  Web App Debugger обеспечивает лёгкий путь наблюдать запросы по HTTP, ответы и время, потраченное на ответы. Web Application Debugger выполняет роль web-сервера. Как только приложение отлажено, оно легко может быть преобразовано в другой тип web-приложения, а затем поставляться в составе коммерческого web-сервера.

- Class Name
  Class Name является идентификатором того, что будет использоваться в URL для соединения с конкретным приложением Web App Debugger.

Второй шаг проходится только в случае, когда на первом шаге выбрано изолированное приложение VCL или изолированное консольное приложение. Предлагается ввести порт HTTP для связи. Мастер также позволяет протестировать доступность порта при помощи кнопки Test Port. Нажатием кнопки Find Open Port можно автоматически заполнить поле HTTP Port значением автоматически определяемого свободного порта.

Третий шаг предназначен для задания параметров REST сервера DataSnap.

Если выбрана опция Authentication, компонент TDSHTTPServiceAuthenticationManager размещается на форме. Компонент TDSHTTPWebDispatcher использует TDSHTTPService Authentication Manager как AuthenticationManager для осуществления авторизации пользователя HTTP для сервера DataSnap. Реализация состоит в определении свойства HTTPAuthenticate. Когда выбрано Authentication, клиент должен предоставить имя (user name) и пароль (password) в свойствах SQL connection.

Выберем опцию Server Method Class для добавления компонента TDSServerClass на форму сервера и для определения класса на сервере, который будет экспонировать серверные методы клиентскому приложению. Если выбрать опцию Sample Method, то модуль ServerMethodsUnit будет содержать реализацию двух простых методов, названных EchoString и ReverseString, которые возвращают Value как параметр в нормальном и, соответственно, обращенном состоянии.

На четвертом шаге мастера DataSnap REST Application запрашивается тип «предка» для класса серверного метода.

Выберите TDSServerModule для экспонирования наборов данных от сервера на клиентские приложения. Выберите TDataModule, если нужно использовать невизуальные компоненты в классе сервера. Выберите TComponent, если хотите полностью реализовать серверный класс.

На пятом шаге будет запрошено корневое размещение web-приложения REST. Это есть директория размещения исполняемого модуля проекта и размещение файлов web-приложения, таких как .js, .html и .css.

DataSnap Client – dbExpress

Начиная с Delphi 2007, когда появился фреймворк dbExpress, нашей целью стало создание инфраструктуры для различных технологий, баз данных и платформ, расширяя существующую поддержку многозвенных приложений и обеспечивая поддержку Java, .NET, PHP и других клиентский технологий для связи с серверами DataSnap.

DbxClient теперь также используется для соединения клиента с сервером DataSnap. Это означает, что для соединения с сервером DataSnap нужно использовать SQLConnection, задавая, что драйвер для связи – DataSnap и обеспечивая hostname (server) и порт связи.

Существует много способов вызывать серверные методы. Давайте предположим, что на сервере есть класс, который содержит методы HelloWorld и GetEmployee.

```
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;
```

Метод HelloWorld является очень простым. Он просто возвращает строку. Метод GetEmployee, в свою очередь, получает ID и возвращает TDBXReader. Другими словами, это – курсор, который выполняет чтение на стороне клиента в качестве объекта DBXReader или ClientDataSet.

Продолжая работать с методом HelloWorld, можно запустить его на исполнение с помощью компонента SQLServerMethod, как показано ниже:

```
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;
```

Теперь рассмотрим метод GetEmployee. В этом примере серверный метод выполняется посредством dbExpress Framework. То же самый метод может вызываться через 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;
```

Заметим, что выполнение осуществляется через TDBXCommand. Возвращаемое значение имеет тип DBXReader и копируется в ClientDataSet в строке №13, таким образом, обеспечивая визуализацию данных в VCL.

В случаях, когда нет необходимости отображать данные, можно просто выполнять чтение из DBXReader-а.

Может показаться, что если всё происходит динамически, то компилятор не в состоянии определить, когда серверные методы изменяются. Да, это так, когда серверные методы не представлены в клиентском интерфейсе. Компонент SQLConnection включает опцию под названием «Generate DataSnap Client Access» («сгенерировать клиентский доступ DataSnap»), что осуществляет генерацию модуля на клиентской стороне, который содержит все методы, предоставляемые серверной стороной. Каждый метод в клиентском классе содержит реализацию для запуска серверного метода.

Рассмотрим в качестве примера класс:

```
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;
```

Возникает вопрос, где размещается DataSetProvider: на сервере, на удаленном модуле данных (remote data module) или на обычном модуле данных? Ответ простой: на клиентской стороне используется новый компонент DSProviderConnection, который соединен с SQLConnection (DataSnap). Свойство ServerClassName отображает имя класса (DataModule/RDM, обычно), где провайдеры располагаются на сервере. ClientDataSet может использовать DSProviderConnection в качестве ProviderName.

Новые DataSnap и фреймворк dbExpress обеспечивают значительную гибкость, которая не ограничивается рамками данного раздела. Можно динамически получать доступ к списку методов вместе с их параметрами, которые позволяют разработчикам создавать компоненты, которые управляют доступом пользователей к серверу и задают правила, специфичные для каждого серверного метода и класса. Можно получить более полное представление, просмотрев пример приложений DataSnap, в моем блоге:
http://www.andreanolanusse.com/blogen/tag/datasnap/ и
http://www.andreanolanusse.com/blogen/tag/dbexpress.

DATASNAP – передача и получение объектов

Одним из ключевых вопросов при миграции проектов на новую DataSnap или разработки проектов с начала является передача объектов между клиентами и серверами DataSnap. Технология DataSnap 2009 позволяла передавать данные только типов, определяемых dbExpress. В DataSnap 2010 можно передавать любые объекты между клиентами и серверами.

В DataSnap 2010 введена поддержка нотации JSON (JavaScript Object Notation), которая представляет собой облегченный формат обмена данными, легко понятный для человека и легко интерпретируемый для компьютера. JSON также не зависит от языка программирования и платформы. Для примера передадим объект класса TCustomer между клиентом и сервером DataSnap.

```
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;
```

В DataSnap 2010 любые объекты, которые производные от TJSONObject могут быть переданы между клиентом и сервером без какой бы то ни было трансформации. Если объект не является потомком TJSONObject, то необходимо использовать классы TJSONMarshal и TJSONUnMarshal для преобразования этих объектов. Пример ниже показывает, как нужно выполнить данное преобразование.

```
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;
```

Нет необходимости реализовывать два метода преобразования для каждого класса. Можно реализовать родовой метод для класса, который использует простые типы данных, строки, числа, булевские значения. Поддержка некоторых классов достаточно сложна, поэтому конвертеры могут быть реализованы с использованием RTTI.

Класс TCustomer теперь готов для передачи между клиентом и сервером. Остается только реализовать серверный метод, который вернет значение TJSONValue после преобразования TCustomer.

```
// 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;
```

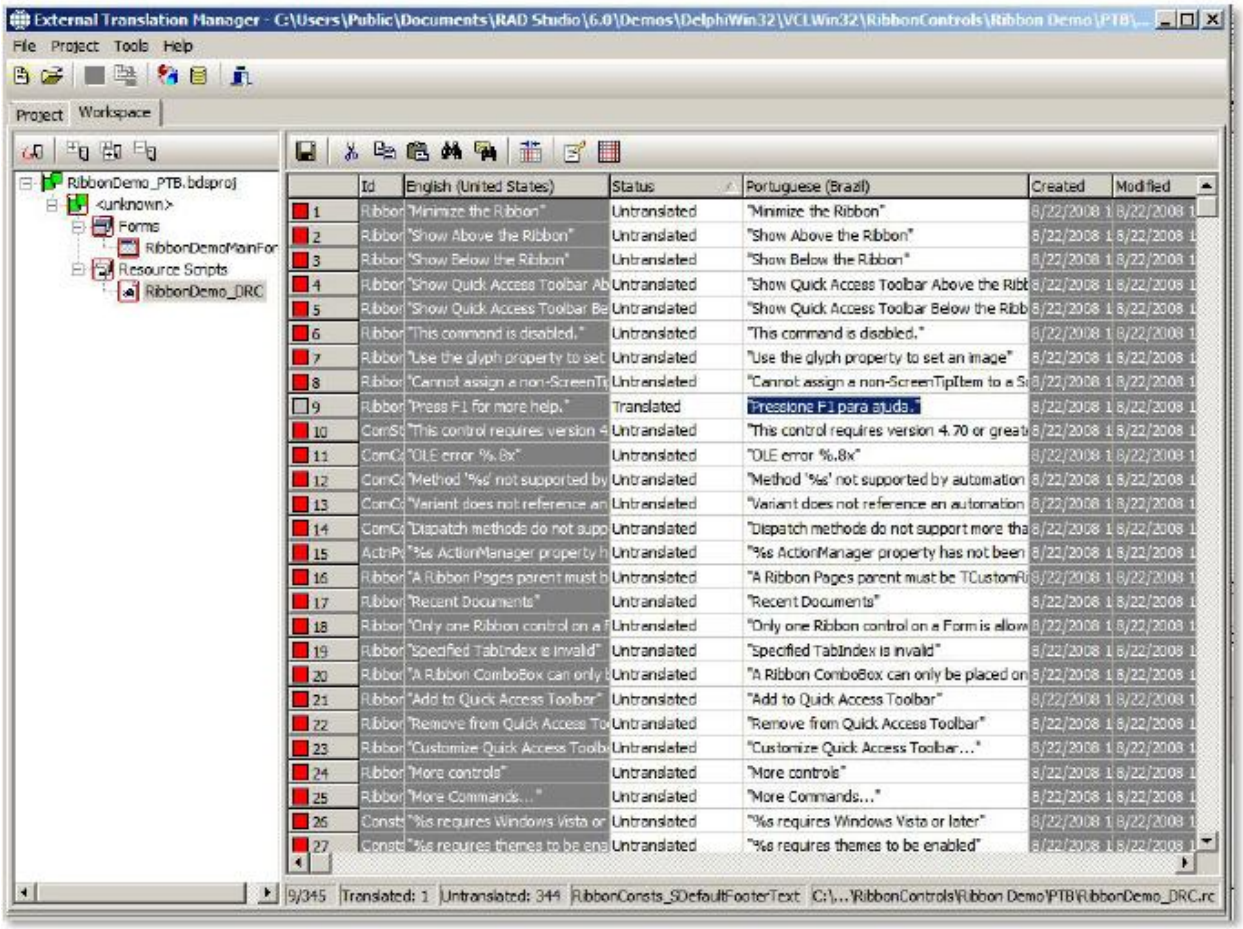
Запуск метода GetJSONCustomer со стороны клиента приведет к обязательной конвертации возвращаемого значения из TJSONValue в TCustomer с использованием метода JSONToCustomer.

```
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;
```

Можно также привести множество других примеров, когда возвращается массив объектов, комплексные классы и т.д. Исходный код этих примеров доступен по адресу: http://cc.embarcadero.com/Item/27361.

Сервис для перевода и локализации приложений в Delphi

Теперь сервис, интегрированный в IDE, стал изолированным. Это означает, что профессионалы, отвечающие за перевод проекта, могут использовать то же самый сервис, что и разработчик, без необходимости установки Delphi. Для каждого нового языка генерируется проект для перевода. Теперь гораздо легче исправлять DFM-файлы для разных языков.



Фигура 39. Сервис перевода

UML-моделирование, аудиты, метрики и документация

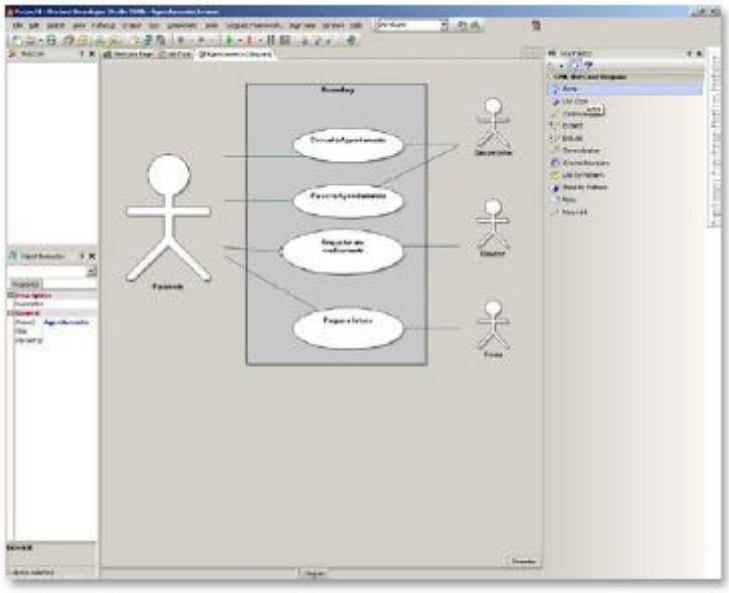
UML-моделирование

Финальная версия любого продукта должна быть наивысшего качества. Требования по качеству также жестко предъявляются, когда речь идет о программных продуктах (особенно, когда на них основаны перспективы развития компании). Программное обеспечение низкого качества может сильно затронуть интересы потребителей.

Начиная с Delphi 2006, можно использовать UML и все его диаграммы. Кроме того, можно использовать также LiveSource, что позволяет синхронизировать диаграмму классов и исходный код.

Ниже представлен список всех доступных диаграмм вместе с их функциональностью:

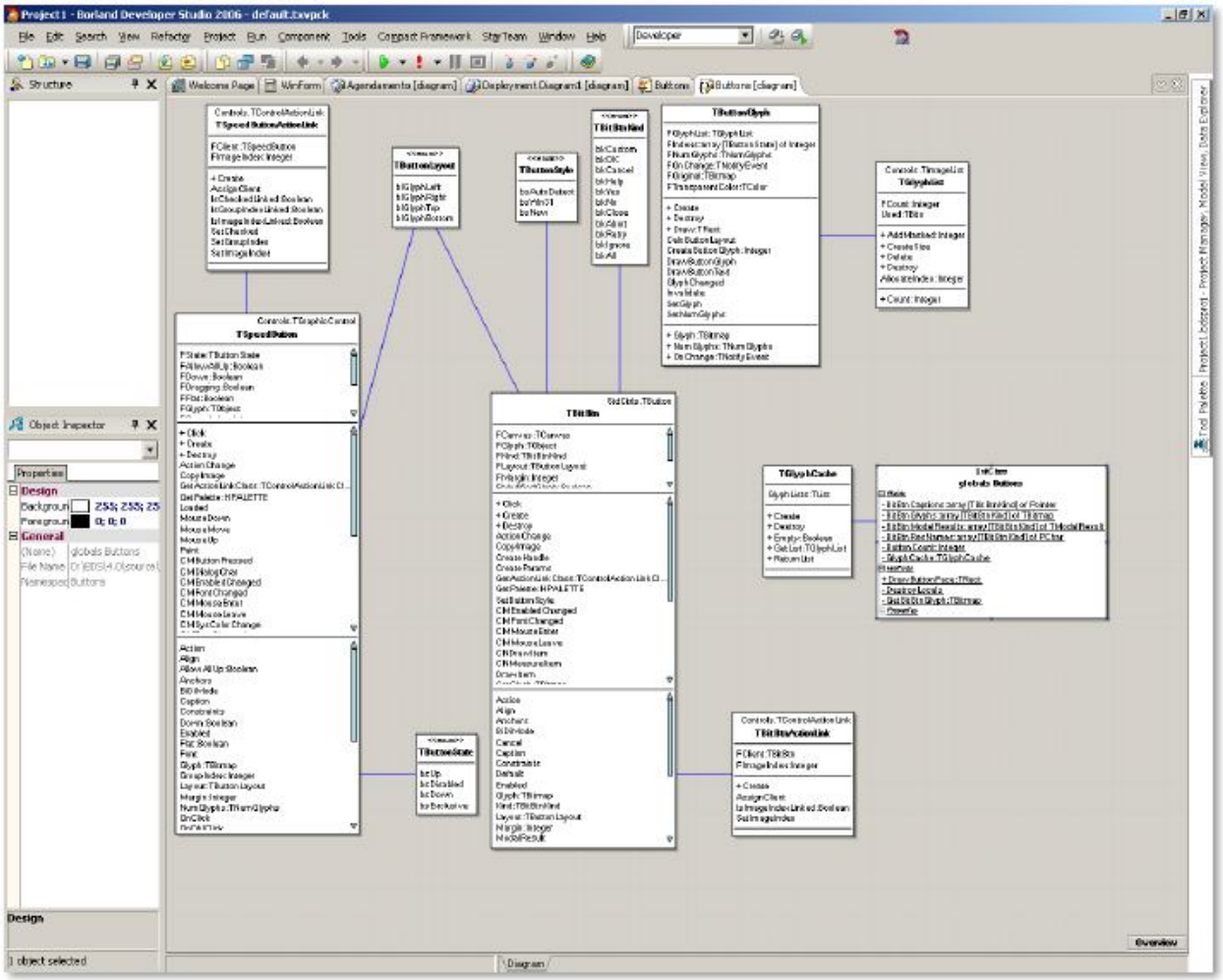
- Use Case (диаграмма использования) → это способ описать взаимодействия между системой и реальным миром. В этом случае действующие лица (как люди, так и системы) представляют реальный мир.
- Class Diagram (диаграмма классов) → представляет классы системы и отношения между ними.
- Collaboration (диаграмма сотрудничества) →используется для моделирования динамических аспектов системы или подсистемы.
- Activity (диаграмма деятельности) →позволяет представлять динамические ситуации за счет «потоков» (flows) (с использованием это диаграммы можно представить «поток» между различными объектами).
- Component (диаграмма компонентов) → используется в высокоуровневом моделировании, в случаях, когда представляются более сложные структуры. Такая диаграмма иллюстрирует системы, встроенные элементы управления и т.д.
- State (диаграмма состояний) → определяет последовательность событий для данного объекта.



Фигура 40. Диаграмма вариантов использования

Визуализация диаграммы классов значительно упрощает и облегчает понимание классов, представленных на ней (в сравнении с аналогичным анализом непосредственно кода).

Рассмотрим пример из мира Delphi: модуль Buttons.pas содержит много компонентов – TBitBtn, TSpeedButton и другие. Теперь представим, как сложно будет проанализировать 1946 строк кода, чтобы понять, какие компоненты здесь представлены и какие связи между ними установлены. Однако если применить технику обратного проектирования (reverse-engineering), то ситуация станет гораздо проще, как показано на фигуре ниже:



Фигура 41. Диаграмма классов

Диаграмма последовательности может быть сгенерирована непосредственно из исходного кода. Диаграммы последовательности детализируют, каким образом методы реализованы в исходном коде: какими сообщениями они обмениваются с какими объектами и когда. Диаграмма последовательности организована относительно времени. Объекты, вовлеченные в методы, перечислены слева направо в соответствии с тем, когда они вызываются в последовательности обмена сообщениями.

Delphi полностью поддерживает возможность проведения обратного проектирования кода. Embarcadero стремится помочь разработчикам использовать и анализировать уже существующий программный код за счет соответствующих технологий, которые обеспечивают дальнейшее развитие проектов.

Аудиты

Когда речь заходит о качестве, люди обычно подразумевают программное обеспечение с высокой производительностью. Некоторые даже говорят о том, что не важно, как разработано программное обеспечение, лишь бы оно работало и удовлетворяло требованиям по функциональности. Такой подход, фактически, будет иметь фатальные последствия в недалеком будущем. Если разработанный код неструктурирован, то снижается возможность развития приложения в будущем. Это бывает, когда все куски «свалены в кучу», а приложение развивается бессистемно с точки зрения структуры. Аудиты и метрики в Delphi помогают локализовать проблемы в коде разрабатываемого приложения.

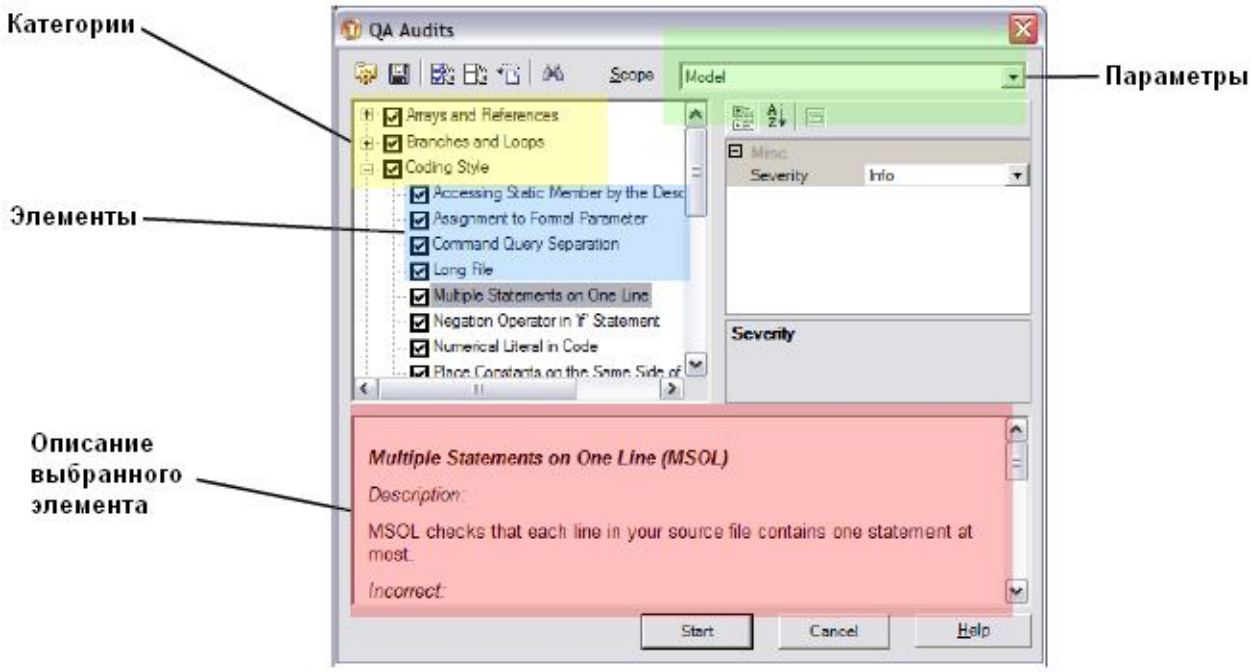
Как часто вам приходилось определять самые лучшие правила для кодирования, надеясь, что это предотвратить превращение кода, разрабатываемого вашими программистами приложения, в некий массив текста, разобрать который не предоставляется возможным?

Предполагая, что ваша команда использует правила хорошего кодирования, зададим следующий вопрос. Насколько вы уверены, что этим правилам постоянно и неукоснительно следуют?

Ответ один: необходим регулярный просмотр кода. А теперь подумаем, что реальный проект содержит несколько тысяч строк кода, причем его объем увеличивается катастрофически.

Используя аудиты кода (QA Audits) в Delphi, можно выбирать среди группы самых лучших стилей программирования, будучи уверенным, что команда разработчиков будет следовать им. Аудиты помогут вам определять проблемы в коде еще на этапе разработки.

- Массивы и ссылки
- Дублирование кода
- Избыточный код
- Производительность
- Ветвление и циклы
- Стиль кодирования
- Стиль именования
- Выражения
- Ошибки проектирования
- Вероятные ошибки



Фигура 42. Аудиты качества кода (QA Audits)

Каждый из аудитов включает описание, объясняющее корректность или некорректность его использования, что помогает разработчикам понять, как применять каждый аудит. Можно задать уровень «серьезности» аудита как Info (информация), Warning (предупреждение) или Error (ошибка).

Тело цикла никогда не выполняется (Loop Body is Never Executed (LBNE))

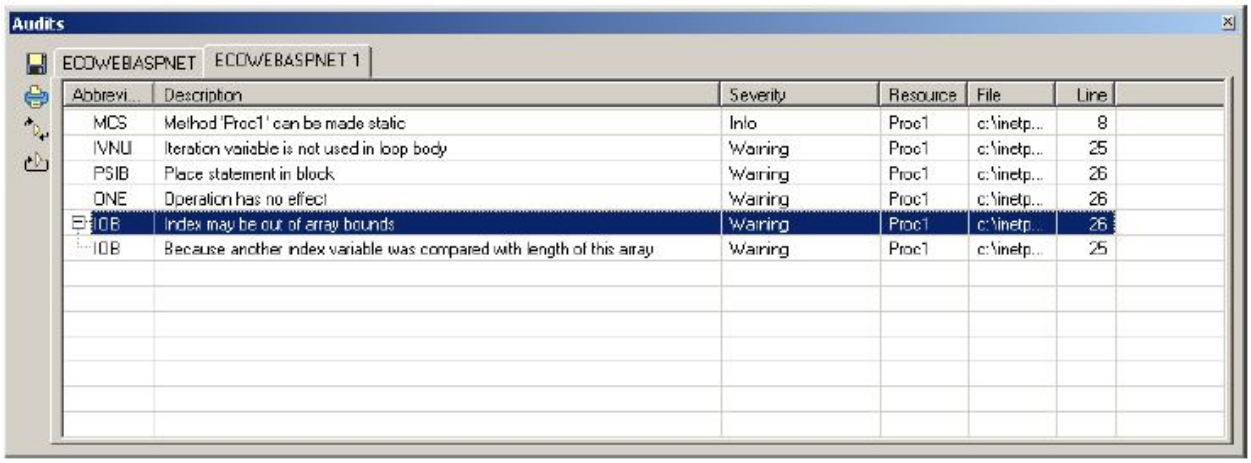
Часто приходится видеть много процедур или функций, которые требует проведения аналитической работы с использованием отладчика, чтобы подтвердить вхождение во все циклы. Аудит LBNE помогает определить такого рода нарушения. Следующий код иллюстрирует ситуацию, когда определяется LBNE. Более сложные ситуации также легко идентифицируются.

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

Индекс за границами (Index Out of Bounds (IOB))

Это – обычное сообщение, когда осуществляется попытка получить доступ к позиции массива, которой не существует. Пример внизу демонстрирует генерацию данного предупреждения.

```
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;
```



Фигура 43. Аудиты

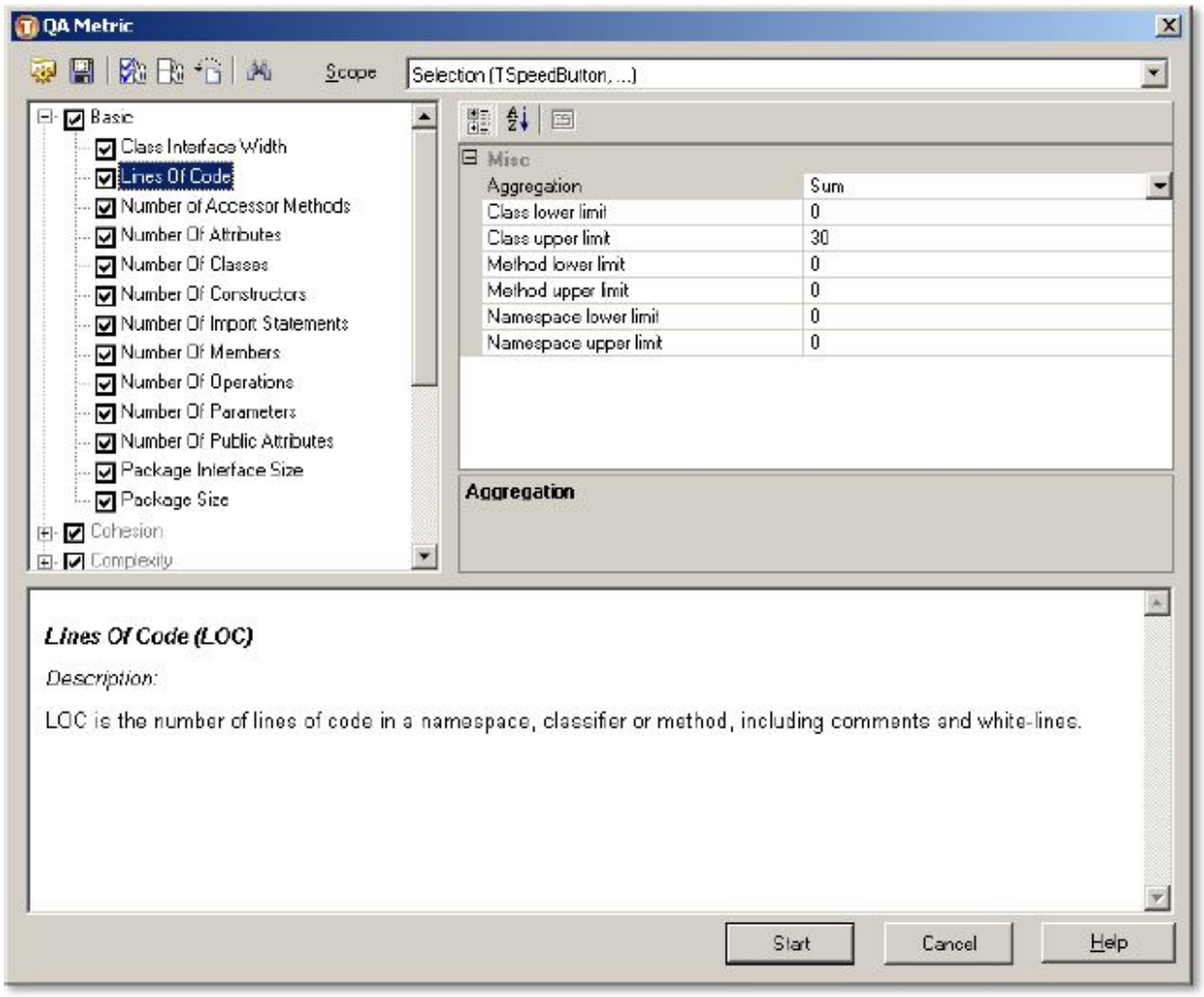
Данный аудит выявляет ошибку, информируя, что код в строке 25 пытается получить доступ с использованием переменной, которая не является частью 2-го цикла. Проведем анализ: переменная j используется в выражении ‘for’, что означает её предположительное использование для доступа к элементу массива, однако в дальнейшем используется переменная i из внешнего цикла.

Delphi XE позволяет выполнять аудиты из командной строки, что облегчает интеграцию со средством автоматизации сборки проекта.

Метрики

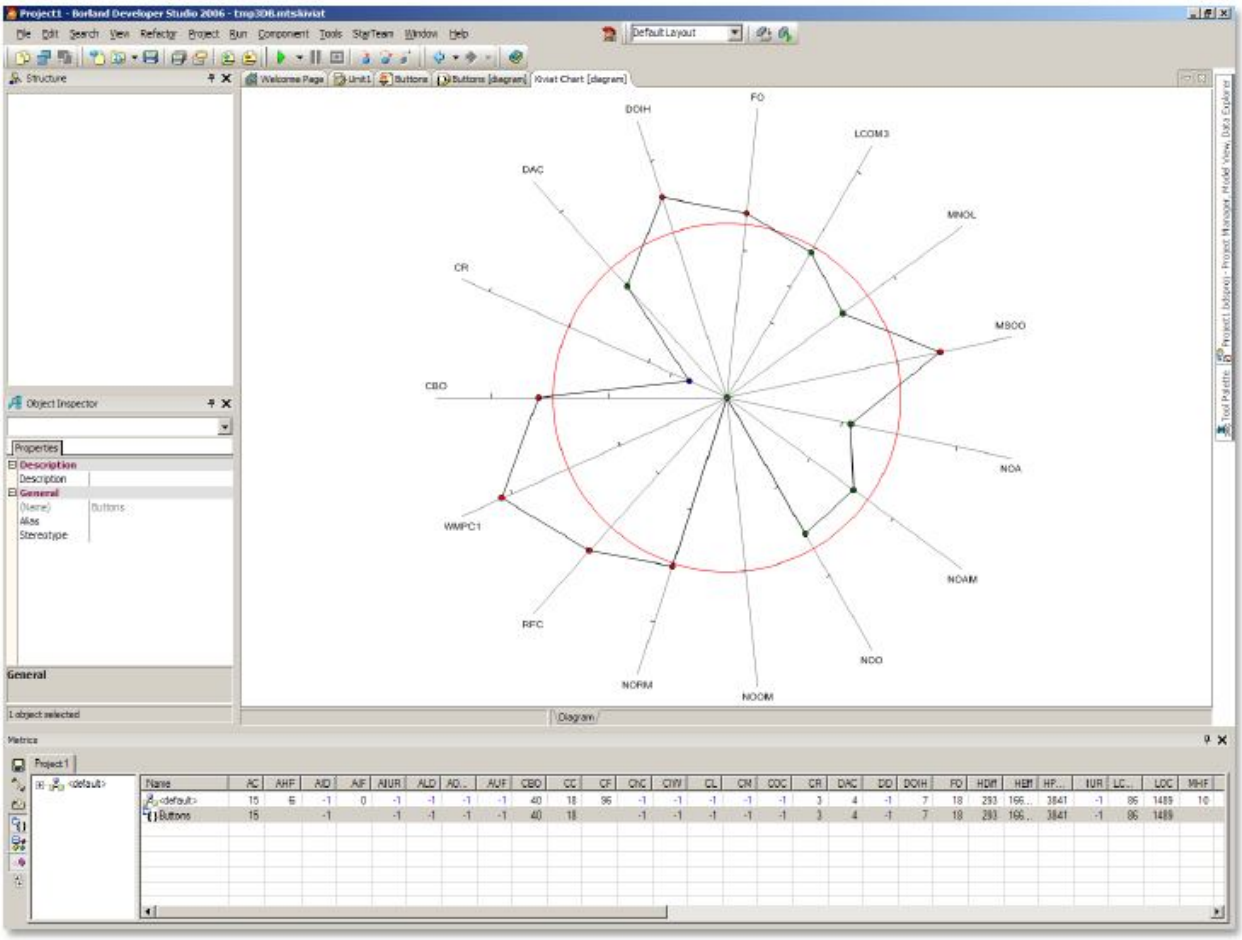
Метрики помогают идентифицировать стили кодирования, которые могут нарушать предопределенные лучшие практики в объектно-ориентированном проектировании и программировании. Наверное, каждому приходилось видеть код, содержащий 10 конструкторов в одном классе, 10 вложенных циклов, метод с 20 параметрами или что-то в подобном роде. Метрики помогают задавать стандарты, лучшие практики и ограничения для команды разработчиков в плане кодирования. Например, у класса не должно быть более 4 конструкторов и не более 400 строчек кода.

Каждая метрика задает ограничение сверху и снизу для класса, метода и пространства имен. Каждое ограничение может настраиваться:



Фигура 44. Метрики качества

После того как произведено вычисление метрик, результаты могут быть проанализированы с использованием диаграммы Kiviat chart. На данной диаграмме красная окружность означает предопределенное ограничение для метрик. Точки вне данной границы означают, что в коде присутствуют нарушения одного или нескольких правил для метрик.



Фигура 45. Анализ метрик

Можно проводить анализ каждого из классов отдельно. Это – хороший способ простой и легкой идентификации нарушения ограничений по метрикам. С использованием аудитов и метрик разработчики могут повысить качество приложений за счет применения более эффективных методов кодирования.

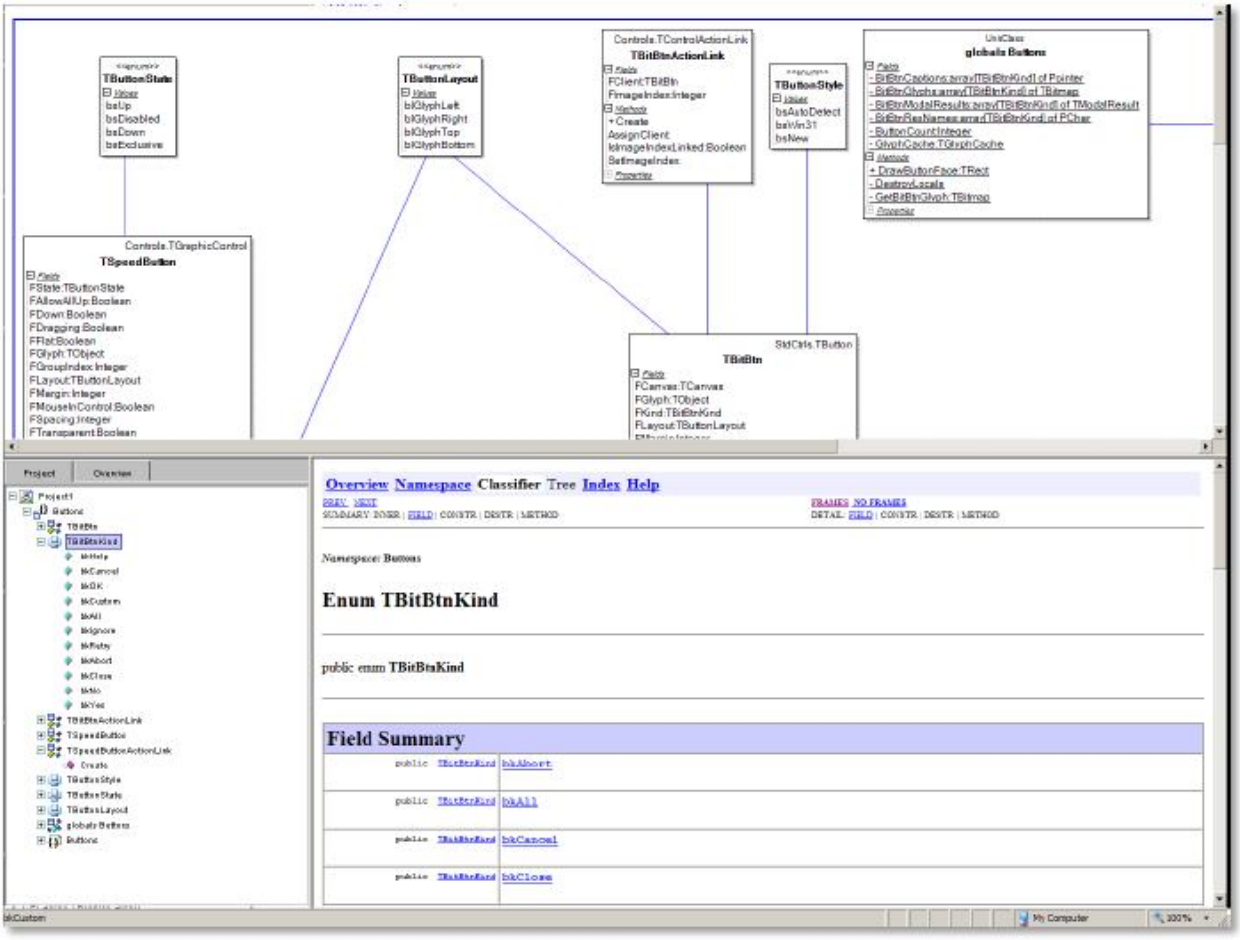
Проведение аудитов, также как и анализ метрик можно запускать из командной строки.

Также это будет полезно предпринять в процессе миграции программного кода проекта с Delphi на самый последний релиз Delphi.

Документирование

Составление и ведение документации по проекту всегда было определенным «камнем преткновения» для разработчиков. Разработчики с большим энтузиазмом занимаются разработкой качественного кода и в меньшей степени любят процесс создания документации. С помощью диаграмм Delphi разработчики, аналитики и архитекторы смогут более эффективно создавать код и документацию к нему. Подготовка документации теперь стала очень простой. Выберем в качестве примера диаграмму классов, а в ней просто кликнем на классе, переменной, методе или другом атрибуте класса. Далее кликнем правой кнопкой мыши и выберем опцию Generate Documentation из контекстного меню. Также можно сгенерировать документацию для модели из Project Manager или при помощи командной строки.

Документация генерируется в формате HTML с разделами: project overview (обзор проекта), просмотр диаграммы (diagram view) и документация подробно (documentation details).



Фигура 46. Подробная документация

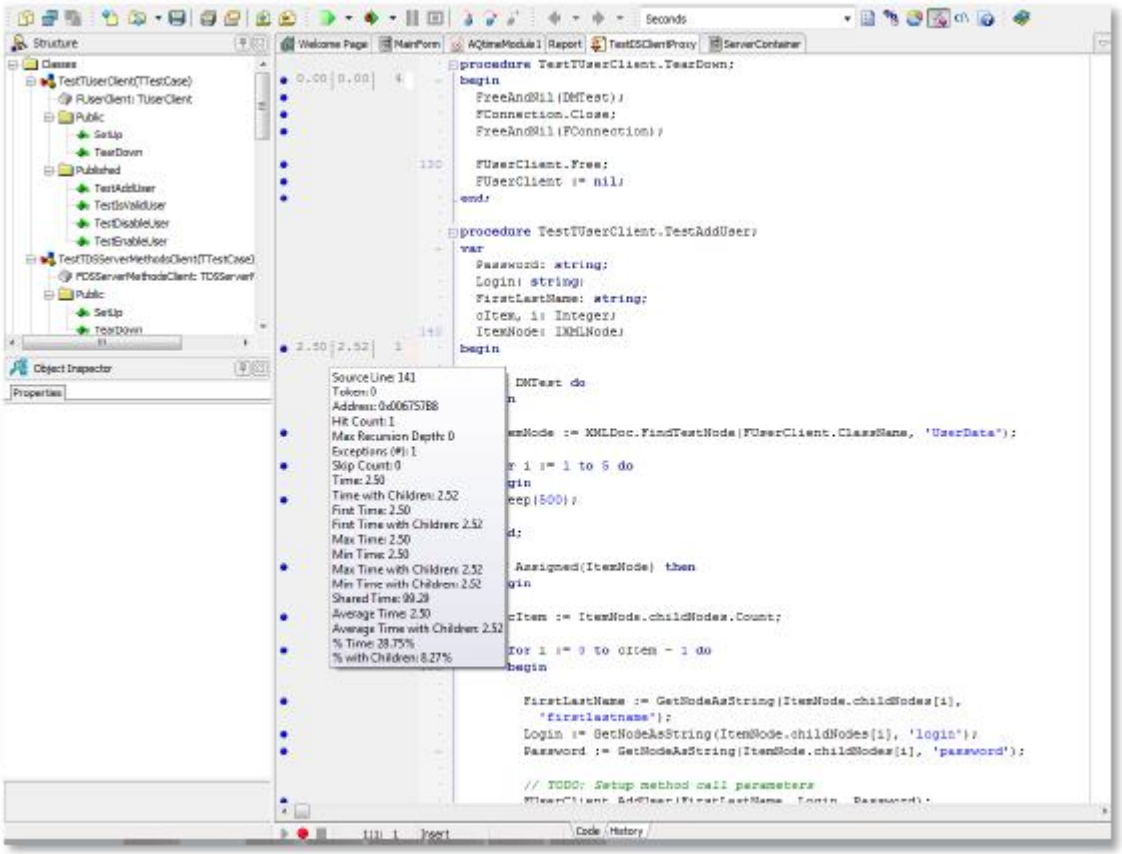
Инструменты и компоненты сторонних разработчиков

Многие инструменты и компоненты сторонних разработчиков, поставляемые вместе с Delphi, были обновлены и получили новые возможности, совместимые с существующими проектами. Если необходимо получить ответы на дополнительные вопросы по поводу отдельных инструментов и компонентов сторонних разработчиков, которые вы используете, пожалуйста, обратитесь к странице по адресу:
http://www.embarcadero.com/products/delphi/tools-and-components.

AQTime – средство профилирования производительности

AQTime представляет собой передовое средство для профилирования и отладки памяти/ресурсов. AQTime в редакции standard интегрировано в IDE, что позволяет использовать всю мощь данного средства непосредственно из IDE.

В процессе оптимизации и улучшении кода AQTime дает точную «картину» выполнения приложения. С использованием AQTime можно профилировать код с точки зрения анализа длительности выполнения каждого метода. Можно использовать AQTime для анализа покрытия кода. AQTime также может определять утечки памяти и ошибки выделения ресурсов. С использованием AQTime разработчики могут существенно сократить время, затрачиваемое на «угадывание», а вместо этого больше сил сконцентрировать на разработке.



Фигура 47. Результаты анализа метода с использованием AQTime

AQTime в редакции Pro помогает полностью понять то, как ваша программа работает в процессе исполнения.

FinalBuilder – автоматизация сборки

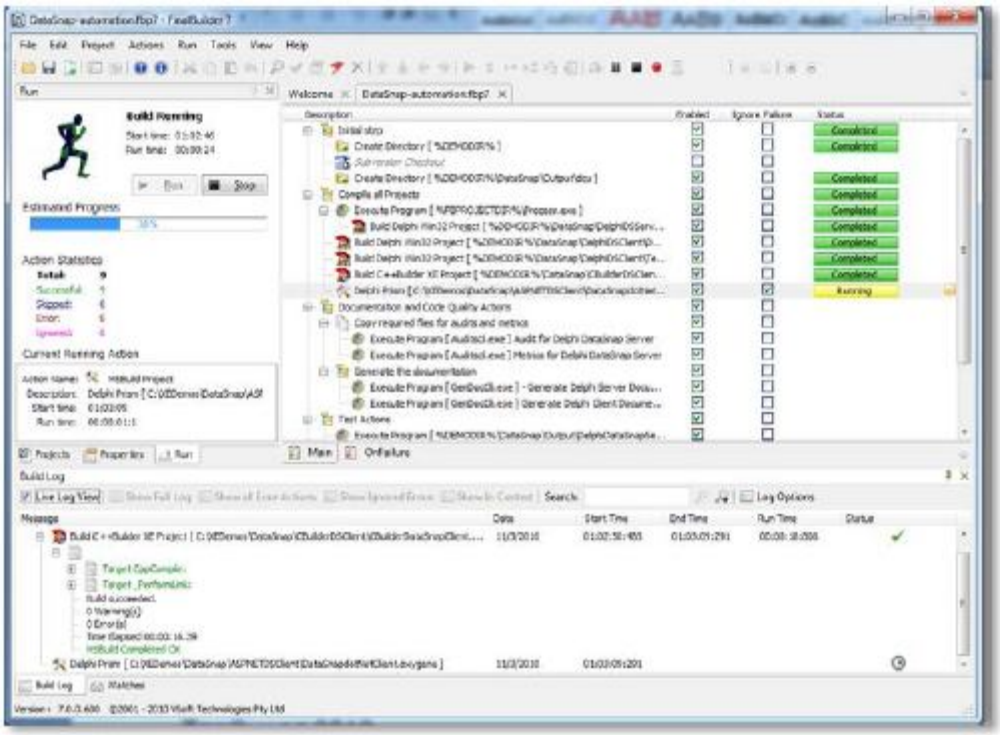
Delphi XE в редакциях Enterprise и Architect включает FinalBuilder – набор инструментов для управления автоматизацией сборки и подготовкой релиза, что эффективно при интенсивном развитии проекта.

Обычно для автоматизации сборки используются пакетные файлы, XML или скрипты. Эти методы подразумевают создания скрипта для сборки, который сложно поддерживать, тяжело понять и который страдает от недостатка механизма обработки ошибок.

В такой ситуации инструмент автоматизации сборки с функциональностью, подобной FinalBuilder, поможет облегчить создание, отладку, сопровождении и своевременный запуск процесса сборки.

FinalBuilder поможет:

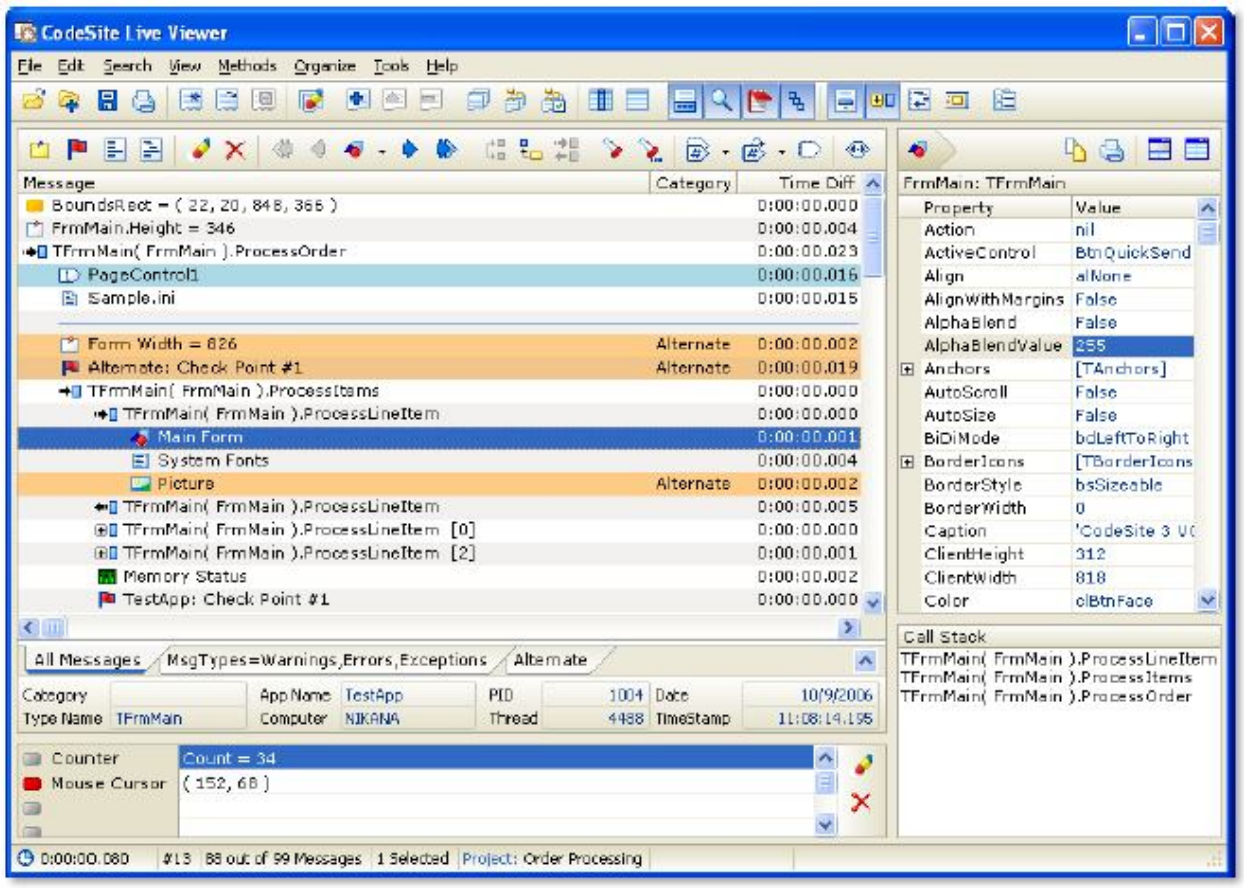
- Сэкономить время – для любого более-менее значимого проекта автоматизированная сборка быстрее, чем ручная.
- Любому члену команды выполнить сбоку – FinalBuilder легко использовать, для сборки не нужен какой-то «гуру», который только один может выполнить все операции.
- Улучшить качество релизов – FinalBuilder снижает значимость человеческого фактора за счет автоматизации задач и запуска тестов сразу после выполнения сборки.
- Вести запись, что и когда было собрано – FinalBuilder делает запись в журнал на каждом действии, который он выполняет, или на каждой операции по запуску инструмента. Журналы от предыдущих сборок архивируются и могут быть экспортированы в систему учета процессов.



Фигура 48. Средство автоматизации FinalBuilder в действии

CodeSite – комплексная система журналирования

Система журналирования CodeSite дает возможность разработчику глубже заглянуть в процесс выполнения кода. Это позволяет быстрее находить проблемы и, соответственно, решать их, гарантируя качество приложения. Система журналирования CodeSite дает возможность разработчикам извлекать всю необходимую информацию в процессе работы кода и отображать ее на экране или записывать в log-файл.



Фигура 49. CodeSite Live Viewer

IP*Works

IP*Works! представляет собой обширный фреймворк для Интернета, он устраняет сложность разработки под Интернет, предоставляя лёгкие, программируемые компоненты, облегчающие такие задачи, как посылка электронных писем (emails), передача файлов, работа по сети, просмотр web-ресурсов, использование web-сервисов и т.д.

TeeChart 2010

Одним из наиболее популярных компонентов Delphi среди разработчиков является TChart, который обновлен до версии 2010. Теперь он получил новые возможности для построения графиков в проектах Delphi.

Rave Reports 9

Delphi XE поставляется с Rave Reports, знаменитым генератором отчетов, который теперь обновлен до версии 9.

Beyond Compare

Beyond Compare позволяет разработчикам просто и быстро сравнивать файлы и выполнять операцию слияния (merge).

VCL for the Web

VCL for the Web – ранее называлась IntraWeb – позволяет создавать приложения Web 2.0 с прозрачной интеграцией с AJAX во многих компонентах VCL, а также недавно добавленной поддержкой Silverlight.

Редакции Delphi XE – Professional, Enterprise и Architect

Delphi XE имеет три редакции: Professional, Enterprise и Architect. Для понимания различия между редакциями следует обратиться по адресу:
http://www.embarcadero.com/products/delphi/product-editions.

Редакция Architect включает ER/Studio Developer Edition – средство моделирования данных. Моделирование является жизненно необходимым для разработчиков, программирующих для баз данных. ER/Studio включает поддержку обратного проектирования (reverse-engineering) для уже существующих баз данных и работу с моделями на логическом и физическом уровнях.

ER/Studio поддерживает следующие базы данных: DB2 LUW V9, Hitachi HiRDB, IBM® DB/2®, Informix, InterBase®, Microsoft® Access, SQL Server, Visual FoxPro, MySQL, NCR Teradata, Oracle, PostgreSQL, Sybase, SQL Anywhere. Доступ к другим базам данных также можно осуществить посредством ODBC.

ER/Studio обеспечивает прямое-обратное проектирование, логическое и физическое моделирование.

Заключение

Delphi IDE подверглась значительным улучшениям за последние годы. Delphi XE – не исключение. Новая IDE, интеграция с системой управления версиями, компоненты VCL, управление при помощи касаний и жестикуляции, RTL, поддержка «облачных» технологий, generics, анонимные методы и новая DataSnap – всё это поможет значительно повысить производительность разработки, позволяя не только эффективно развивать уже существующие проекты, но и создавать новые с помощью технологий будущего.

Для дополнительной информации о новых возможностях Delphi XE, пожалуйста, обратитесь по адресу:

http://www.embarcadero.com/products/delphi/whats-new.

Об авторе

Андреано Лануш (Andreano Lanusse) – эксперт в области разработки программного обеспечения и активный член сообщества программистов. Как ведущий специалист-консультант Embarcadero он много времени проводит в общении с разработчиками, участвует в конференциях, дискуссиях и обсуждениях, чтобы удостовериться в том, что инструменты компании отвечают потребностям пользователей. До Embarcadero Андреано работал тринадцать лет в компании Borland на различных должностях, включая координатор поддержки, инженер, менеджер по продуктам и менеджер по продажам. Он также работал как главный консультант в Borland Consulting Services по разработке и управлению критически важными приложениями. Первоначально он работал главным архитектором компании USS Temp (бывшая USS Soluções Gerenciadas). Андреано имеет степень бакалавра по специальности управление бизнесом, полученную в институте Sumare, и MBA по управлению проектами в FGV. Также он является сертифицированным специалистом по управлению продуктами Калифорнийского университета, Беркерли. Также Андреано имеет сертификат SCRUM Master. Вы можете узнать больше об Андреано из его блога: http://www.andreanolanusse.com/.

Также ему можно написать письмо на адрес: andreano.lanusse@embarcadero.com.

О компании Embarcadero Technologies

Компания Embarcadero Technologies, Inc. — ведущий производитель программных средств, которые позволяют разработчикам приложений и специалистам по управлению базами данных эффективно проектировать, создавать и использовать приложения и базы данных в гетерогенных ИТ-средах. Свыше 90 компаний из списка Fortune 100 и активное сообщество, насчитывающее более трех миллионов пользователей по всему миру, широко применяют отмеченные наградами продукты Embarcadero. Эти продукты позволяют им сократить затраты, оптимизировать процессы соблюдения нормативных требований, ускорить разработку решений и внедрение инноваций. Компания Embarcadero была основана в 1993 году. Ее штаб-квартира расположена в Сан-Франциско, а отделения открыты во многих странах мира. Сайт компании Embarcadero: www.embarcadero.com