джерело

ЗМІСТ

Урок 1. Введение в Delphi. 1

Короткий конспект –Висновок 4

Урок 2. Первое знакомство с Delphi IDE 4

Короткий конспект –Висновок 9

Урок 3. Работа с компонентами в дизайнере форм 9

Короткий конспект –Висновок 11

Урок 4. Свойства в Delphi 11

Короткий конспект –Висновок 14

Урок 5. Управление проектом 15

Короткий конспект –Висновок 19

Урок 6. Обзор свойств компонент 19

Короткий конспект –Висновок 25

Урок 7. Обзор палитры компонент: Standard, Additional 27

Короткий конспект –Висновок 30

Урок 8. Pascal - первое знакомство 30

Короткий конспект –Висновок 33

Урок 1. Введение в Delphi.

Введение в Delphi

С чего начать?

Delphi, основой которого является язык Pascal, отлично подходит для того, чтобы начать учиться программировать. Сам Pascal постепенно уходит в прошлое и на него всё меньше обращают внимание. Это и понятно - на нём пишутся приложения для MS-DOS, а эту однозадачную операционную систему все пытаются забыть как страшный сон. Если быть более точным, то языком Delphi является Pascal не в том виде, в каком его используют для написания приложений MS-DOS, а в другой модификации - Object Pascal. В этом языке программирование как бы "привязывается" к определённым объектам - как визуальным, так и невизуальным, просто находящимся в памяти. Программирование простых приложений с интерфейсом командной строки (т.е. когда на экран последовательно выводятся строки текста и при этом пользователь

вводит какие-либо данные) советуют также начинать с Pascal. Также многие начинают с QBasic, но этот язык один из самых "древних" и возможностей у него немного. Однако для понимания общих принципов программирования он также подойдёт. Дело в том, что в этих языках нет каких-либо хитроумных конструкций и наборов знаков - написанный код легко читается и воспринимается. Этого нельзя сказать, например, о C++. Есть шутки на эту тему - "то, что ночью программист писал на C++, утром он прочитать не сможет". Конечно, это не так, но синтаксис языка C++ достаточно сложен. Мы начнём изучение Delphi практически с нуля. Pascal будет изучаться попутно.

Почему Delphi?

Delphi - это нечто иное, нежели Pascal, это совершенно другой качественный этап среды программирования. С помощью Delphi создаются приложения для операционной системы Windows, но помимо этого с помощью дополнительных средств можно написать, например, программы и для Linux. Среда Delphi легко расширяется установкой дополнительных модулей. Пользовательский интерфейс также хорошо настраиваемый - каждый организует рабочее пространство так, как ему будет удобно.

Краткие сведения о Delphi

Delphi — результат развития языка Турбо Паскаль, который, в свою очередь, развился из языка Паскаль. Delphi оказал огромное влияние на создание концепции языка C# для платформы .NET. Многие его элементы и концептуальные решения вошли в состав C#. Одной из причин называют переход Андерса Хейлсберга, одного из ведущих разработчиков Дельфи, из компании Borland Ltd. в Microsoft Corp.
Версия 1 была предназначена для разработки под 16-ти разрядную платформу Win16;
Версии со второй компилируют программы под 32-х разрядную платформу Win32;
Вместе с 6-й версией Delphi вышла совместимая с ним по языку и библиотекам среда Kylix, предназначенная для компиляции программ под операционную систему Linux;
Версия 8 способна генерировать байт-код исключительно для платформы .NET. Это первая среда, ориентированная на разработку мультиязычных приложений (лишь для платформы .NET);
Последующие версии (обозначаемые годами выхода, а не порядковыми номерами, как это было ранее) могут создавать как приложения Win32, так и байт-код для платформы .NET;

Delphi for .NET — среда разработки Delphi, а так же язык Delphi (Object Pascal), ориентированные на разработку приложений для .NET.

Что нам потребуется...

Подразумевается, что Вы знакомы с общими правилами работы в системе Windows и работали в каких-либо приложениях хотя бы примитивного уровня вроде Блокнот или Калькулятор. Из программного обеспечения нам потребуется сама среда Delphi. Процесс установки описан не будет, так как он довольно стандартный. На сайте дистрибутивов Delphi Вы не найдёте - любая из версий имеет объём не менее 200-300 Мб, а хранить такие файлы на сайте просто невыгодно. Кроме того, Delphi не является официально бесплатным продуктом. Поэтому, если у Вас ещё нет дистрибутива, постарайтесь его как можно быстрее найти. Можете купить в магазине, либо возьмите у кого-нибудь из знакомых. Из бесплатных аналогов Delphi можно отметить Lazarus, однако в этих статьях речь будет идти именно о Delphi.

Какую версию Delphi установить?

Это один из самых частозадаваемых вопросов. По большому счёту, все они очень похожи и в большинстве случаев программы будут одинаково работать независимо от версии Delphi, в которой они были созданы. Однако кое-какие советы я всё же дам. Не устанавливайте версии ниже **Delphi 5** - они очень старые и имеют существенные расхождения с более новыми. Среда Delphi 5 содержит все основные возможности, но в ней нет некоторых удобных вещей, которые появились в следующих версиях. Наиболее оптимальный вариант - **Delphi 6** или **Delphi 7**. Эти версии наиболее популярны среди "населения". Самой стабильной считается Delphi 6. Delphi 7 - мало чем отличается от Delphi 6, разве что большей совместимостью с Windows XP (имеется ввиду совместимость написанных приложений). Это НЕ означает, что программы, написанные в Delphi 6, будут некорректно работать в WinXP. Всё будет замечательно. Но Delphi 7 всё же менее стабильна, нежели Delphi 6. Дальнейшие версии - **Delphi 8**, **Delphi 9** я вообще не рекомендую устанавливать. Это самые неудачные из всех. Были сделаны попытки интегрировать средства для написания приложений на технологии .NET, но в ответ сами среды получились довольно неудачными ("глючными"). Далее стоит упомянуть **Delphi 2005 Enterprise Edition**. Эта версия тоже не получила особенного широкого распространения и большинство программистов её просто "перешагнули". Следующая по счёту - **Borland Developer Studio 2006**. Да, это уже целый программный комплекс, включающий помимо Delphi и другие средства разразботки. Среда удобная, но очень ресурсоёмкая. На старых компьютерах с объёмом оперативной памяти менее 1 ГБ не рекомендую её использовать. На этом Borland остановились и дальнейшее производство стало вестись от имени **CodeGear** (а далее - **Embarcadero**). **Delphi 2007**, входящая в **RAD Studio 2007**, мало чем отличается по возможностям от BDS 2006, но зато она очень хорошо оптимизирована и пригодна для использования на маломощных машинах, в отличие от своего предшественника. **Delphi 2009** - это новый большой шаг в развитии Delphi. В этой версии появилась полноценная

поддержка Юникода (этого все ждали и оно свершилось). Конечно, наравне с новыми просторами для деятельности это вскрыло и новые проблемы: некоторые старые программы, компоненты и модули перестали корректно работать. Однако в большинстве случаев все конфликты решаются правкой нескольких строк. Не стоит этого пугаться. Если Вы только начинаете своё "путешествие", то для Вас разницы нет никакой и потому лучше установить сразу более новую версию. Все примеры, которые будут приводиться в данных уроках, работают корректно как в старых, так и в новых версиях Delphi. Ну и наконец, самая свежая версия - **Delphi 2010**. О ней пока не могу много сказать. Устанавливать её или нет - решайте сами. Возможно, есть смысл пока что освоиться с более проверенными версиями.

Поводя итог, советую выбирать из трёх вариантов: Delphi 7, Delphi 2007 или Delphi 2009. На 7-ой версии и сейчас работают многие, утверждая, что лучше неё нет ничего на свете. Но стоит помнить, что рано или поздно всё старое устаревает до такой степени, что становится непригодным. Delphi 2007 - это уже ближе к современности. Ну и Delphi 2009 - если не хотите отставать от всего остального мира. Решать Вам, но я бы выбрал именно последний вариант.

Заключение

В этой статье мы поговорили о происхождении Delphi и его функциях. Далее мы начнём изучение самой среды и языка программирования.

Короткий конспект –Висновок

1.

Урок 2. Первое знакомство с Delphi IDE

Первое знакомство с Delphi IDE

Запуск Delphi

Способов запустить среду существует множество (как и любой другой программы впрочем). Ярлык на рабочем столе, иконка на панели быстрого запуска, пункт в главном меню (Пуск - Программы - Borland Delphi n - Delphi n, где n - номер версии). Также есть удобный способ запустить Delphi через окно Пуск - Выполнить - ввести в этом окне delphi32. Более новые версии (2007, 2009, 2010) выпускаются уже не Borland, а CodeGear,

поэтому в главном меню группа называется CodeGear (Delphi входит в состав RAD Studio, поэтому может быть и название CodeGear RAD Studio). Из командной строки запуск осуществляется командой bds.

Delphi IDE

Вот и первое, возможно новое, для Вас слово. IDE (Integrated Development Environment) - интегрированная среда разработки программного обеспечения. После запуска Delphi перед Вами предстаёт эта самая среда. Состоит она из нескольких окон. Сейчас мы разберём, что это за окна и каково назначение каждого из них. В разных версиях Delphi эти окна могут выглядить немного по-разному, а некоторые и вообще могут отсутствовать. В данной статье будут приведены иллюстрации окон Delphi 7.

Итак, после запуска, наверное, Вы сразу обратите внимание, что среда в целом практически не отличается от других Windows-приложений. Все элементы стандартные. Главным окном можно считать то, которое содержит строку меню и панели инструментов. Вот строка меню:

Многие из этих пунктов стандартны. Если Вы установили русскую версию Delphi, то пункты будут называться примерно так: Файл, Правка, Поиск, Вид, Проект, Запуск, Компонент, База данных, Инструменты, Окно, Справка.

Как и во многих приложениях здесь есть панели инструментов. Они небольшие, кнопок на них немного, но всё самое основное как раз здесь и собрано. Панели инструментов выглядят примерно так:

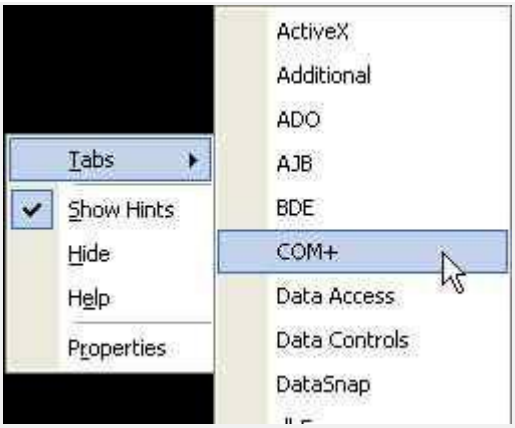
Теперь рассмотрим те элементы, которых в обычных приложениях нет.

Палитра компонент (Component palette)

Палитра компонент - это набор вкладок, на каждой из которых расположены элементы. Именно с помощью этих элементов создаются интерфейсы программ. Все эти элементы принято называть компонентами. Среди компонент бывают как визуальные, так и невизуальные, но об этом мы поговорим позже. Вот как выглядит палитра компонент:

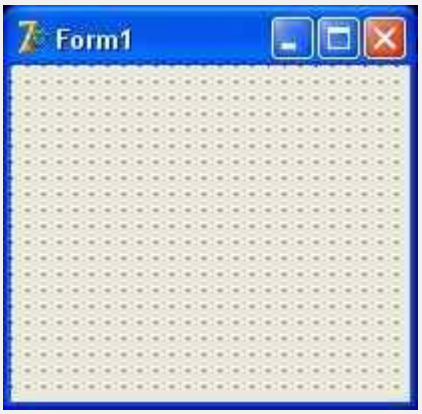
Её внешний вид практически одинаков во всех версиях Delphi. Да что там Delphi, такие же вкладки есть в любой среде объектно-ориентированного программирования (ООП), ибо это самый удобный способ предоставить выбор из сотен (а иногда даже тысяч) различных элементов.

Переключение между вкладками осуществляется стандартным способом - щелчком по названию одной из вкладок. Сразу после установки в Delphi Вы можете видеть огромное количество вкладок. Они даже не помещаются на экране - для прокрутки созданы горизонтальные кнопки со стрелками. Также есть ещё один удобный способ перемещаться по вкладкам - можно щёлкнуть правой кнопкой мыши по палитре компонент и в появившемся меню выбрать пункт Tabs - в результате откроется меню, где будут названия всех существующих вкладок в алфавитном порядке:



Дизайнер форм (Form Designer)

Это самое большое окно всей среды, которое изначально пустое. Именно это - заготовка окна Вашей программы. Здесь и будут размещаться все компоненты. Удобной составляющей дизайнера форм является сетка (множество точек). С помощью этой сетки компоненты удобно размещать на одном уровне, делать их одинаковых размеров и т.д. Это сделано для того, чтобы приложения соответствовали стандартам, установленным Microsoft. На этом мы ещё остановимся в одной из статей. Сетка является настраиваемой - можно изменить расстояние между точками, а можно её и вовсе отключить.



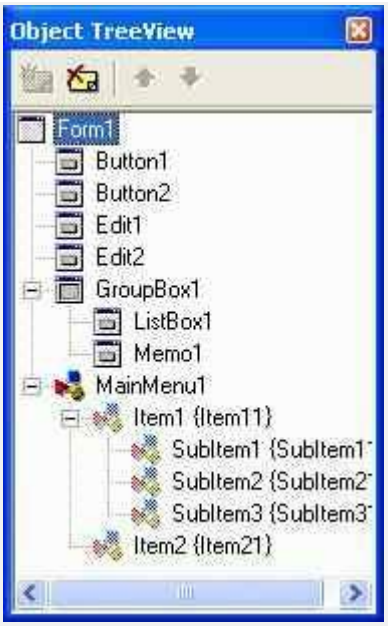
Инспектор объектов (Object Inspector)

Это окошко с двумя вкладками, каждая из которых состоит из двух колонок. В этом окне можно настроить параметры выбранного элемента и задействовать установленные события. Вкладки - Properties и Events (Свойства и События соответственно). Что это за свойства и что же такое события? По этому вопросу можно сказать очень много, это тема для отдельной статьи. А вкратце вот о чём речь. Допустим, у нас есть кнопка. Обыкновенная, какая используется в большинстве приложений. Примерами свойств этой кнопки могут быть её размеры (ширина, высота), текст, расположенный на ней и т.д. События - это предопределённые моменты реакции кнопки на какие-либо действия пользователя (либо действия со стороны операционной системы, внешних устройств и т.п.). Самый простой пример - щелчок по кнопке (так называемый "клик" - от слова Click). Очевидно, что это событие произойдёт тогда, когда пользователь щёлкнет по кнопке, т.е. нажмёт её. У большинства компонент предусмотрены стандартные события. Как правило, среди них есть все необходимые, которые могут понадобиться при создании программы. Однако можно создать и свои события как реакции на что-либо.

Дерево объектов (Object TreeView)

Это окошко появилось только в Delphi 6, в более ранних версиях его не было. В этом окне отображаются все элементы, какие есть на данной форме. Это сделано с целью упростить выбор компонентов для изменения их свойств в Object Inspector (далее - OI). Помимо того, что отображаются названия компонентов, рядом находятся маленькие графические значки, по которым можно определить, что это за объект. Элементы на форме не всегда автономны, т.е. самостоятельны, поэтому образуются иерархические связи - "деревья". Из-за этого окно и называется деревом объектов. В качестве простейшего примера иерархии объектов можно привести меню. Меню - это самостоятельный компонент, а вот его пункты - это "подчинённые" объекты. Пункт меню не может "висеть в воздухе" - он создан в конкретном меню.

Примечание: при динамическом создании пунктов меню они всё же могут просто находиться в памяти и не быть привязанными к какому-либо меню; данный пример приведён лишь для общего представления о связях между объектами.

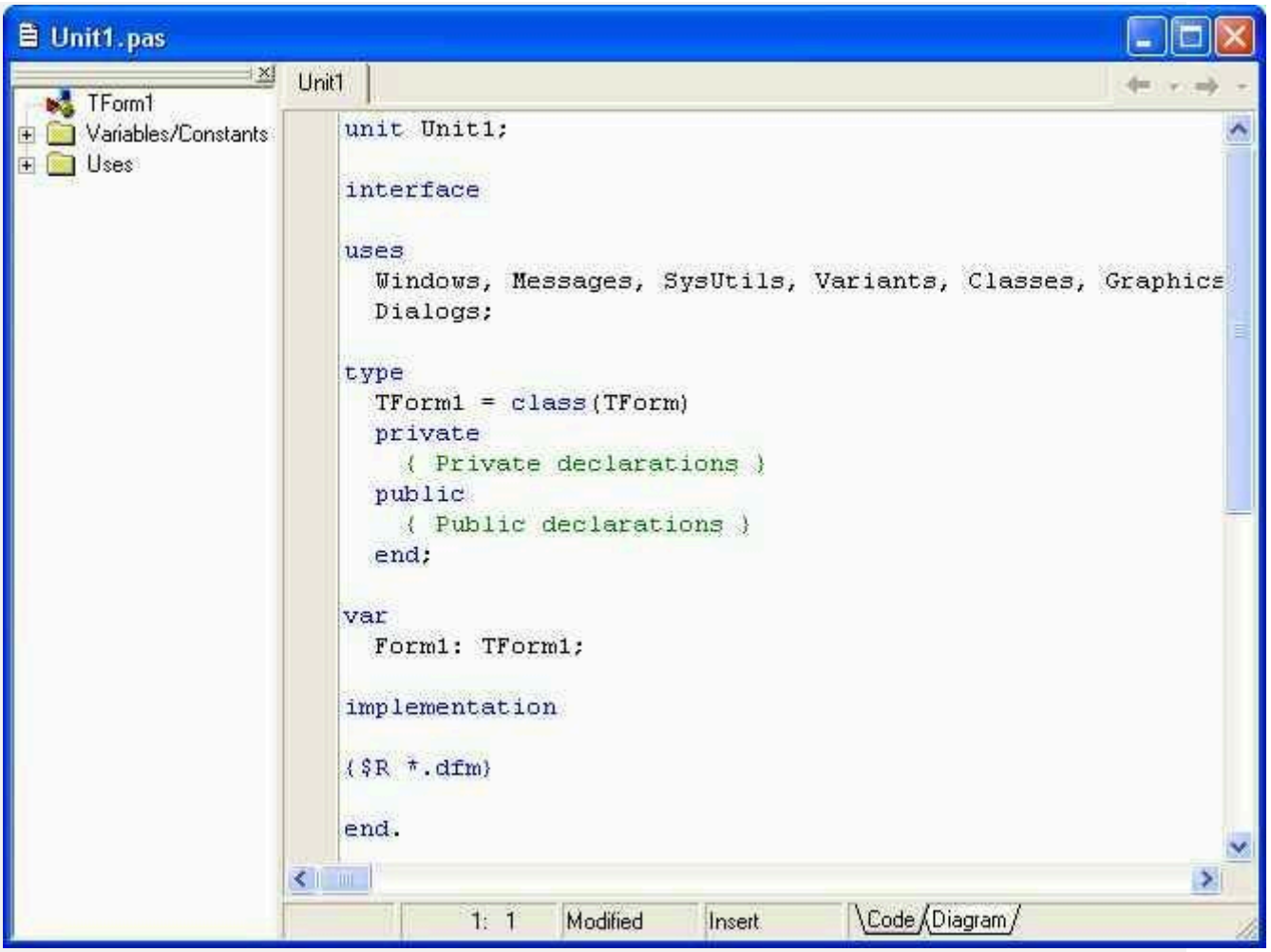


Редактор кода

Редактор кода представляет собой окно, похожее на обычный текстовый редактор за исключением некоторых дополнительных элементов. Основная область этого окна - поле редактирования. Именно здесь пишется текст программы. В отличие от языков программирования, работающих в текстовом режиме (Pascal, QBasic и т.п.) код программы здесь не пишется "с нуля". Как только Вы запустите Delphi и создадите новый проект, то, открыв редактор кода, увидите там уже часть написанной программы. Эти строки удалять ни в коем случае нельзя!

Окно редактора кода может содержать сразу несколько открытых файлов - переключение между ними осуществляется по закладкам вверху окна (на рисунке открыт только один файл - Unit1, поэтому закладка одна-единственная). В левой части окна расположено поле, называемое Code Explorer (Обозреватель кода). Здесь в виде дерева отображаются все типы, классы, свойства, методы, глобальные переменные и другие блоки, находящиеся в данном файле (модуле). Дело в том, что содержимое модуля состоит из отдельных участков. Назначение каждого из них мы рассмотрим чуть позже.

В нижней части окна расположена строка состояния, содержащая полезную информацию. В ней представлена текущая позиция курсора в тексте (номер строки : номер символа), текущий режим режима замены (Insert/Overwrite), информация о том, были ли внесены изменения в модуль с момента последнего сохранения и т.п.



Когда Вы попытаетесь запустить программу, то внизу появится информационное поле, в котором будут показаны сообщения об ошибках в тексте программы. Также в этом окне показываются полезные советы по оптимизации кода. В одной из статей мы разберём все эти сообщения более подробно. Если программа написана "идеально", т.е. ошибок нет и оптимизировать нечего, то окошко даже и не появится на экране.

Заключение

Итак, мы рассмотрели все основные элементы оболочки Delphi, которые используются в процессе работы. Конечно же, в Delphi существует множество других окон, но их назначение и способы вызова на экран мы будем рассматривать в процессе работы. Интерфейс в более новых версиях, чем Delphi 7, отличается, но все основные элементы те же самые, просто расположены они несколько иначе. При желании можно настроить оболочку по своему вкусу.

Короткий конспект –Висновок

1.

Урок 3. Работа с компонентами в дизайнере форм

Работа с компонентами в дизайнере форм

В прошлый раз мы познакомились в основными элементами среды Delphi. Теперь пришло время научиться создавать интерфейсы для программ хотя бы на начальном уровне.

Размещение компонент на форме

Совершенно очевидно, что интерфейс программы создаётся самим программистом. В Delphi (а также в большинстве других сред объектно-ориентированного программирования) элементы размещаются на форме очень просто. Для этого нужно выбрать интересующий компонент на одной из вкладок палитры компонент (Component Pallete), щёлкнув по нему и второй раз щёлкнуть по форме. В месте щелчка появится выбранный компонент и с этого момент он принадлежит данной форме.

Выбор объектов в дизайнере форм

Выбрать объект очень просто - достаточно щёлкнуть по нему. Также объекты можно выбирать с помощью клавиатуры - достаточно нажать клавишу [Tab]. С помощью этой же клавиши можно далее последовательно выбирать все объекты, расположенные на форме. Иногда требуется выбрать сразу несколько объектов. Мышью это делается стандартным способом: не отпуская кнопку мыши очертить прямоугольную область. Все объекты, попавшие в эту область, окажутся выделенными. Для выбора произвольных объектов, не очерчивая область, достаточно удерживать клавишу [Shift] и щёлкать по нужным объектам.

Перемещение элементов на форме

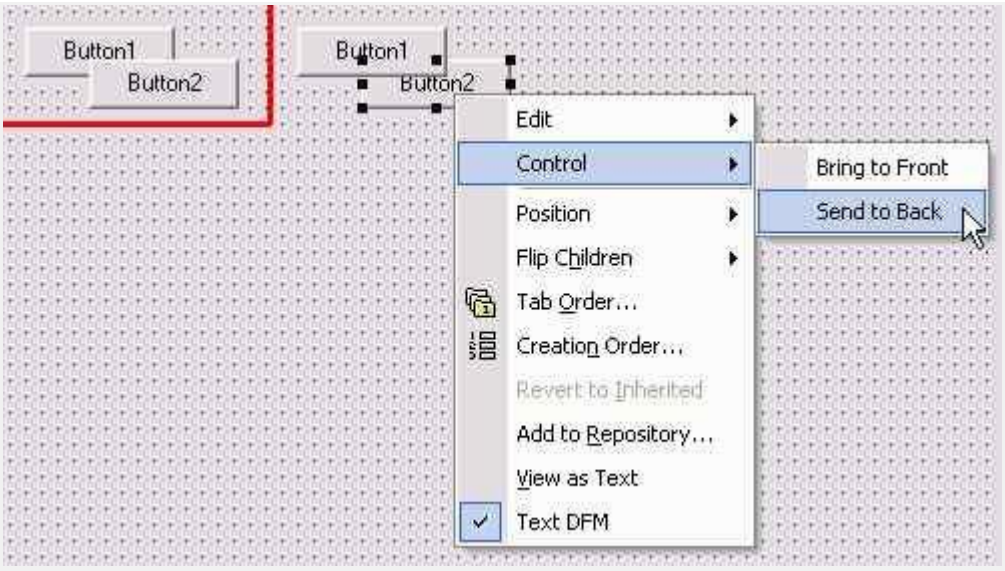
Компоненты можно легко перемещать по форме. Наиболее быстрый способ - "захватить" компонент мышью и перетащить в нужно место. Обратите внимание, что в дизайнере форм есть специальная сетка (точки, стоящие на равном расстоянии друг от друга). С помощью этой сетки удобно выравнивать компоненты относительно краёв формы или относительно друг друга. При перемещении мышью компонент перемещается именно по этой сетке. Также перемещение можно выполнять с помощью клавиатуры. Если компонент выделен, то перемещать его по сетке можно с помощью комбинаций клавиш [Ctrl]+[Shift]+[стрелка]. [стрелка] - одна из клавиш "вверх", "вниз", "влево" или "вправо". Но не всегда сетка позволяет разместить компоненты так, как задумано, поэтому перемещать компоненты можно и с более высокой точностью - по одной экранной точке (пикселю). Для этого служит комбинация [Ctrl]+[стрелка].

Изменение размеров элементов

Если помещать компоненты на форму "стандартным" методом - просто щёлкая в произвольном месте формы, то компоненты принимают размер, установленный стандартом. Однако стандартные размеры очень часто оказываются неудобными. Помещая компоненты на форму, можно сразу указывать их размеры. Делается это очень просто - вместо щелчка по форме нужно нажать кнопку мыши и, не отпуская её, растянуть прямоугольную область. В результате компонент примет размер очерченной области. Изменять размеры установленных на форме объектов не менее просто - если выделить объект, то на его контуре появляются так называемые узлы, потянув за которые как раз можно изменить размеры. С помощью клавиатуры размеры изменяются при помощи комбинаций [Shift]+[стрелка].

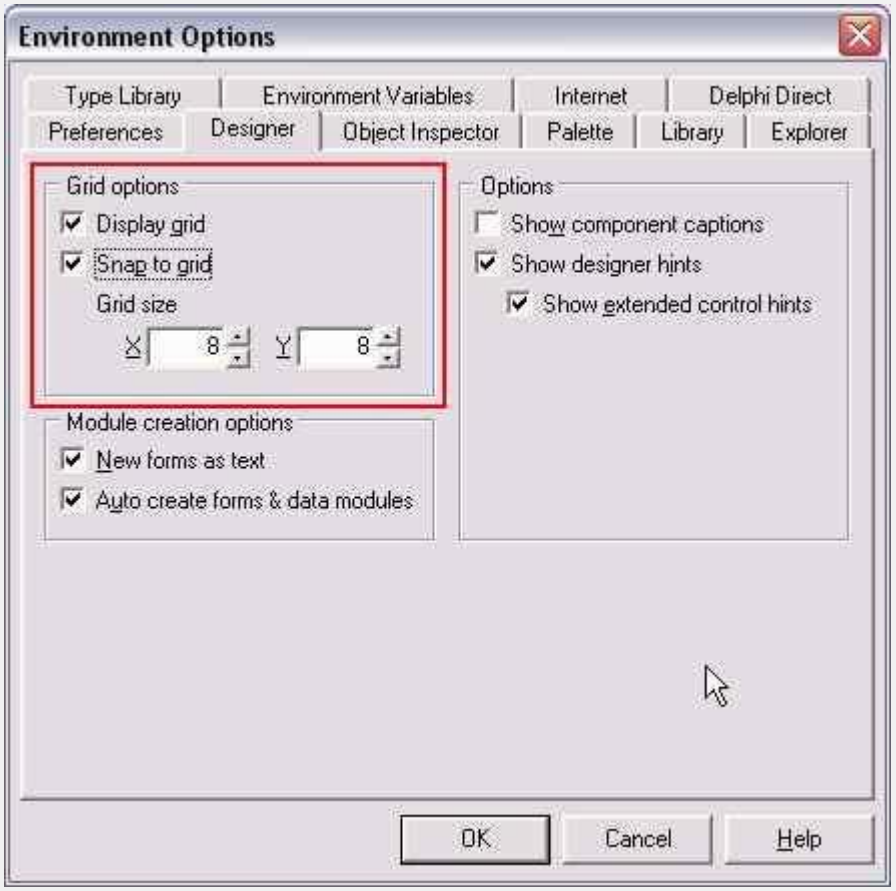
Перекрывание объектов

Ничто не мешает делать элементы такими, что они будут перекрывать друг друга. Например, разместите на форме кнопку (Button1), а затем ещё одну (Button2) прямо поверх первой. Вероятно, вторая кнопка частично перекроет первую (см. рисунок). Но как быть, если Вы хотите, чтобы наверху была именно первая? Всё очень просто. Откройте контекстное меню второй кнопки (щёлкните по ней правой кнопкой мыши) и выберите в меню Control - Send to Back (Отправить назад). Вторая кнопка уйдёт за первую. В том же меню есть команда Bring to Front (Перенести вперёд) - перемещает элемент на уровень выше.



Настройка сетки дизайнера форм

Не всегда стандартная сетка в дизайнере форм удобна. Некоторые приложения изначально создаются с нестандартными элементами нестандартных размеров. Настроить сетку можно в окне Tools - Environment Options, вкладка Designer, блок Grid options. Параметры Grid size - X и Y позволяют изменить шаг сетки по горизонтали и вертикали соответственно. Опция Display grid отвечает за отображение сетки вообще, т.е. если сетка не нужна, её можно отключить совсем. Опция Snap to grid указывает на то, как компоненты будут себя вести при помещении на форму и при перемещении. Если выключить эту опцию, то компоненты будут просто "не замечать" сетку и двигаться не по точкам, а произвольно.



Заключение

В данном уроке мы рассмотрели все основные манипуляции с объектами на форме. Однако размещать элементы на форме "как попало" и произвольно изменять их размеры (например делать огромные кнопки) крайне нежелательно. Существуют специальные стандарты и в одном из уроков мы с ними познакомимся.

Короткий конспект –Висновок

1.

Урок 4. Свойства в Delphi

Свойства в Delphi

Что такое свойства? Что является свойствами в реальной жизни? Например, в физике это температура плавления, температура кипения, вязкость, плотность, растворимость и т.д. Аналогично и объектно-ориентированном программировании. Каждый объект имеет какие-либо свойства. Свойства отвечают либо за внешний вид объекта, либо за его поведение в программе во время её выполнения.

Где найти свойства

Каждый компонент, помещённый на форму, имеет своё отражение в Инспекторе объектов (Object Inspector). Попробуйте, к примеру, поместить на форму кнопку TButton и текстовое поле TEdit и выделить по очереди сначала один объект, а затем другой, наблюдая при этом за Инспектором объектов. Вы заметите, что содержимое его окна изменяется. Это связано с тем, что каждый объект имеет свои свойства. Свойство может быть у одного объекта, но его может не быть у другого. Например, у поля ввода (TEdit) есть свойство ReadOnly, отвечающее за возможность изменения текста в этом поле. Совершенно логично, что у кнопки (TButton) этого свойства нет и быть не может.

Как создаются программы

Чтобы научиться создавать программы, для начала нужно понять основные принципы их создания. Коротко на этом остановимся. Итак, Вы уже научились помещать компоненты на форму, управлять ими и просматривать свойства. Размещение компонентов на форме - один из первых этапов разработки программы. Однако, просто поместив компоненты никаких действий происходить не будет - по кнопке можно будет щёлкнуть, но при этом ничего не произойдёт. В поле ввода можно ввести текст, но этот текст никуда не отправится и никак не обработается. Поэтому совершенно логичным этапом является настройка взаимодействий между компонентами. Некоторые простые взаимодействия уже "встроены" в стандартные компоненты и легко изменяются через Инспектор объектов. Всё остальное нужно создавать и программировать вручную. Практически основным методом создания взаимодействий компонентов является написание реакций на события. Но об этом чуть позже.

Типы свойств

Прежде чем перейти к рассмотрению типов свойств, нужно понять, что же из себя вообще представляет свойство? Всё очень просто. Свойство - это поле какой-либо структуры, содержащее определённое значение. Отталкиваясь от того, каким образом задаются эти значения и какую "природу", т.е. структуру имеет свойство, выделено несколько типов свойств.

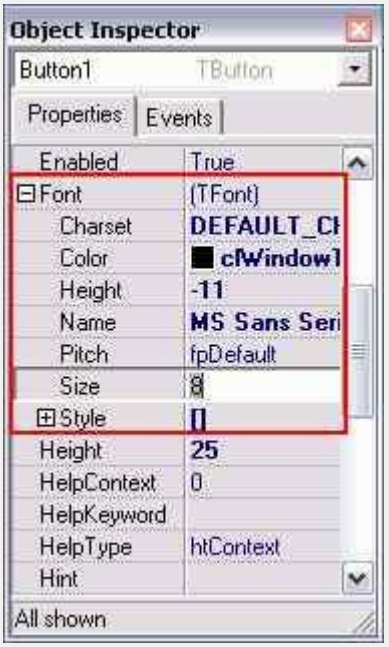
1. Простые свойства. Простыми являются те свойства, значения которых являются числами, либо строками (текстом). Примерами таких свойств могут служить Left и Top формы. Эти свойства определяют положение формы на экране (в частности, её левого верхнего угла). Значения этих свойств

- числа. Пример свойства со значением-строкой - Caption формы. Это свойство хранит заголовок формы и задаётся в виде обычного текста.

2. Перечислимые свойства. Такими являются свойства, задать значения которым в явном виде нельзя, а можно только выбрать из списка. Список возможных значений определяется заранее. Пример такого свойства - свойство AutoSize формы. Оно отвечает за то, будет ли форма автоматически изменять свой размер, чтобы отобразить все размещённые на ней компоненты. Значение - либо истина (True), либо ложь (False). Другой пример - свойство BorderStyle. Это свойство отвечает за внешний вид формы, а также за поведение её границ, т.е. можно ли изменять размеры формы во время работы программы. Это свойство принимает одно из 6 значений.
3. Вложенные свойства. Это те свойства, которые имеют внутри несколько других свойств. В Инспекторе объектов слева от названий таких свойств отображается маленькая кнопка со знаком "+", нажатие на которую раскрывает данное свойство (знак при этом меняется на "-"). Повторный щелчок "сворачивает" свойство обратно. Вложенные свойства бывают двух основных типов - это множества и комбинированные значения. Множества - это набор каких-либо значений, каждое из которых либо "включено", либо "выключено". Комбинированные значения - это набор из нескольких свойств, которые могут иметь разный тип данных. Примером множества является свойство BorderIcons у формы - оно отвечает за кнопки, которые будут показаны в строке заголовка окна. Понятно, что любая из кнопок может либо отображаться на экране, либо нет - такой набор очень удобно задавать с помощью множества. Примером комбинированного значения является свойство Font (оно есть у большинства визуальных компонентов) - задаёт шрифт для элемента. В его включено несколько других свойств - название шрифта, цвет, стиль, размер и т.п.

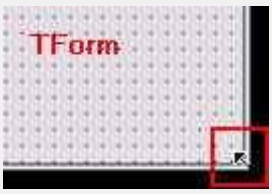
Простые свойства

Перечислимое свойство



Свойство вложенного типа

Управление свойствами



Управлять свойствами, т.е. изменять их значения можно двумя способами - в режиме проектирования программы (Design-time) и во время выполнения (Run-time). В данный момент нас интересует режим проектирования. Изменять свойства можно всё в том же Инспекторе объектов. Если это простое свойство, то достаточно щёлкнуть по строке с названием этого свойства и ввести новое значение. Если это перечислимое свойство, то значение можно выбрать из списка. Некоторые свойства (например, Left, Top, Width и Height) можно изменять простыми операциями перетаскивания и изменением

размеров с помощью мыши.
Примечание: Left, Top - положение формы на экране, Width - ширина, Height - высота формы.

Следует отметить, что в режиме проектирования через Инспектор объектов могут быть доступны не все свойства выбранного компонента. Эти свойства доступны в режиме выполнения и изменяются они программным путём.

Немного о компонентах

Чтобы у Вас не возникало путаниц и непонятных моментов, считаю необходимым сделать следующее уточнение. Все компоненты подразделяются на визуальные и невизуальные. Во время проектирования (Design-time) видны все компоненты без исключения. А вот во время выполнения (Run-time) - не все. Те, которые представляют собой какой-то видимый объект (поле, кнопка, таблица и т.д.) и являются визуальными. Невизуальные компоненты на экране не видны, но при этом используются в самой программе. В качестве примера невизуального компонента можно привести TApplicationEvents со вкладки Additional. Во время проектирования этот компонент присутствует на форме и он доступен через Инспектор объектов, но во время работы программы его не видно. За видимость объекта на экране в большинстве случаев отвечает свойство Visible (True - видимый, False - невидимый).

Короткий конспект –Висновок

1.

Урок 5. Управление проектом

Управление проектом

В каком виде сохраняются созданные программы? Этот вопрос возникает у любого начинающего программиста. Эта тема заслуживает отдельного внимания. Как известно, в Windows любая программа представляет собой exe-файл, который является исполняемым, т.е. может быть запущен как отдельное приложение. Понятно, что разрабатывая программу, в итоге нужно получить именно exe-файл, чтобы его можно было запустить на компьютере, где не установлена среда, в которой эта программа была создана. В языках программирования для DOS, в частности Turbo Pascal, всё довольно просто - весь программный код сохраняется в один-единственный файл, а в итоге получается exe-файл. Довольно просто и удобно. При переходе в Windows всё становится гораздо сложнее. Оконное приложение Windows не может быть сохранено в одном файле. Если в среде DOS программы можно называть именно программами, то в объектно-ориентированном программировании они называются проектами. Почему проектами? Всё достаточно просто - программа представляет собой совокупность некоторого числа файлов различного типа, определённым образом связанных между собой. Очевидно, что всю эту группу логично назвать проектом.

В этой статье мы познакомимся с самыми основными типами файлов, которые включены в любой проект, созданный на Delphi. На самом деле типов файлов гораздо больше, но эти - базовые.

***.dpr - файл проекта**

Это главный файл всего проекта. Если этот файл отсутствует, запустить программу будет невозможно. В файле хранятся все основные параметры приложения, а также информация об окнах, которые в приложение включены. Файл представляет собой свободно читаемый код. Посмотреть содержимое этого файла можно командой меню Project -> View Source. В более новых версиях Delphi файл проекта имеет другое расширение - .bdsproj (в Borland Developer Studio 2006), .dproj (в Delphi 2007 и далее). Обратная совместимость поддерживается.

***.dfm - файл описания формы**

В этом файле содержится описание всех объектов, которые расположены на форме, а также информация обо всех их свойствах. При этом в этот файл сохраняются только те свойства объектов, которые были принудительно изменены. Это объясняется тем, что все свойства любого объекта имеют значения по умолчанию (т.е. заранее установленные). Соответственно нет смысла хранить значения абсолютно всех свойств - достаточно запомнить только те, которые имеют значение, отлиное от начального. Все остальные свойства принимают своё исходное значение и таким образом вся форма восстанавливается.

***.pas - модуль (самостоятельный, либо модуль формы)**

Именно этот файл больше всего похож на файл программ Turbo Pascal. В этом файле находится код программы. Модули могут быть отдельными от конкретных проектов - в этом случае их можно подключить к любому проекту и использовать. Как правило, в отдельных модулях находятся вспомогательные функции, либо какие-либо объекты. Помимо этого модуль есть у каждой формы. В результате *.pas-файл неразрывно связан с файлом *.dfm, а форма соответственно описывается этими двумя файлами - один содержит её состояние и объекты, а второй - код программы, относящийся к этой форме. Следует отметить, что модули значительно облегчают процесс написания программы и ориентацию в больших модулях - отдельные элементы большого модуля можно вынести в несколько модулей и просто подключить их к проекту.

***.res - файл ресурсов**

Это дополнительный файл. В нём находятся данные различных типов - это могут быть тексты, изображения, звуки и т.д. К примеру, для проекта этот файл создаётся автоматически и в нём сохраняется иконка (значок) приложения, а также некоторая информация о проекте.

Вот и все основные файлы, информацию о которых обязательно нужно знать. Чтобы перенести проект, достаточно следующих файлов: *.dpr, *.dfm и *.pas. Имея эти файлы проект можно восстановить, а программу - запустить. Желательно также наличие *.res-файла(ов), но тем не менее это не обязательно. Если *.res-файл, сопоставленный проекту будет отсутствовать, то при открытии проекта Delphi предупредит, что файл не найден и спросит, что нужно сделать. Достаточно проигнорировать сообщение и файл будет создан автоматически при следующем запуске программы.

***.dof, *.cfg - файлы конфигурации проекта**

В этих файлах хранятся опции текущего проекта (которые настраиваются в окне Project - Options). При отсутствии этих файлов будут использоваться стандартные параметры.

***.dcu, *.obj - объектные файлы**

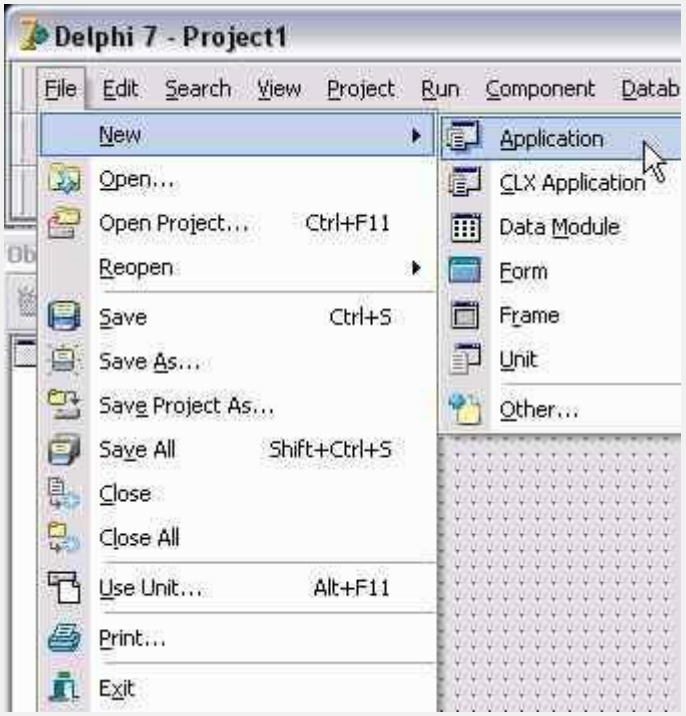
Эти файлы представляют собой уже скомпилированные модули. При очередной компиляции Delphi создаёт для каждого pas-файла соответствующий dcu-файл. Это существенно ускоряет компиляцию в дальнейшем, т.к. имеющиеся файлы просто включаются в конечный exe-файл, а не обрабатываются заново. OBJ-файлы - это тоже скомпилированные модули, но применяющиеся в C++. При работе в Delphi эти файлы не используются, но при необходимости их можно сформировать.

***.~* - временные файлы**

Файлы, расширение которых начинается со знака тильды ("~"), являются временными. Это означает, что они не несут какой-либо важной информации и используются в других целях (например, для ускорения компиляции). Эти файлы можно смело удалять, если они мешают. Чем больше проект, тем обычно больше появляется временных файлов.

Работа с проектом

Ну а теперь пришло время рассказать о базовых операциях с проектом, так как ранее речь об этом не шла. Рассмотрим всё по порядку.



Меню File содержит все основные команды управления проектом

Создание нового проекта

Для создания "чистого листа" следует выбрать в меню File -> New -> Application. В разных версиях Delphi содержимое подменю New может немного отличаться, но всё основное присутствует всегда. После этого создаётся новый проект, такой, какой появляется при запуске Delphi.

Открытие существующего проекта (или файла)

Для этого существует команда File -> Open... Выбрав файл *.dpr, откроется проект, а выбрав какой-либо другой файл (например, *.pas или *.dfm) откроется что-либо другое (например, отдельная форма).

Сохранение

С сохранением есть несколько тонкостей. Исходя из того, что проект представляет собой совокупность нескольких файлов, можно сделать вывод, что сохранять нужно все эти файлы, а не какой-то в отдельности. Выбрав File -> Save, Вы сохраните только текущую форму, но не более того, а проект останется "висеть в воздухе". File -> Save As... - стандартный пункт, который делает то же самое, что и Save, только позволяет пересохранить файл (форму) под другим именем. Команда Save Project As... сохраняет файл проекта (*.dpr). Таким образом, чтобы сохранить проект полностью, нужно сохранить каждую из форм и сам проект. Делать это по отдельности достаточно неудобно, поэтому существует команда, облегчающая этот процесс: File -> Save All. При вызове этой команды сначала появится диалог для сохранения формы (если форм несколько, то и диалогов будет несколько), а затем диалог для сохранения проекта. После того, как все диалоги отработали, можно с уверенностью сказать, что проект сохранён полностью.

Железное правило: каждый проект должен быть сохранён в отдельном каталоге!

Если в один каталог сохранить несколько проектов, то все файлы перемешаются и можно отправлять всё в корзину. Этого нельзя делать ни в коем случае!

Запуск и остановка программы

Теперь, когда проект сохранён, программу можно и запустить и посмотреть, что же получилось. Ещё одно правило, которое желательно соблюдать: перед запуском программы проект нужно сохранить. Мгновенный вопрос: зачем? Конечно, делать это или нет - решать Вам, но бывают случаи, когда программа зависает (по вине самого программиста например), а с ней зависает и оболочка Delphi. В этом случае ничего не остаётся делать, как "убивать" Delphi через Диспетчер задач Windows. Понятно, что изменения, которые были внесены в проект, теряются. Однако не всё так плохо. Случаи зависания Delphi IDE достаточно редки. Как правило, если программа зависает, Delphi позволяет её просто уничтожить из памяти и продолжить работу. Процесс сохранения проекта перед запуском можно поручить оболочке: меню Tools -> Environment Options..., вкладка Preferences, блок Autosave options - опция Editor files.

Запомните следующие основные горячие клавиши:

Ctrl+F9 - компиляция программы. Что такое компиляция? Говоря простым языком - создание выходного (exe) файла программы. Следует отметить, что имя выходного файла совпадает с именем проекта, т.е. именем *.dpr-файла и не может быть изменено. Выходной файл создаётся в том же каталоге, где расположение *.dpr-файл. Однако компиляция просто "собирает" всю программу, но не запускает её.

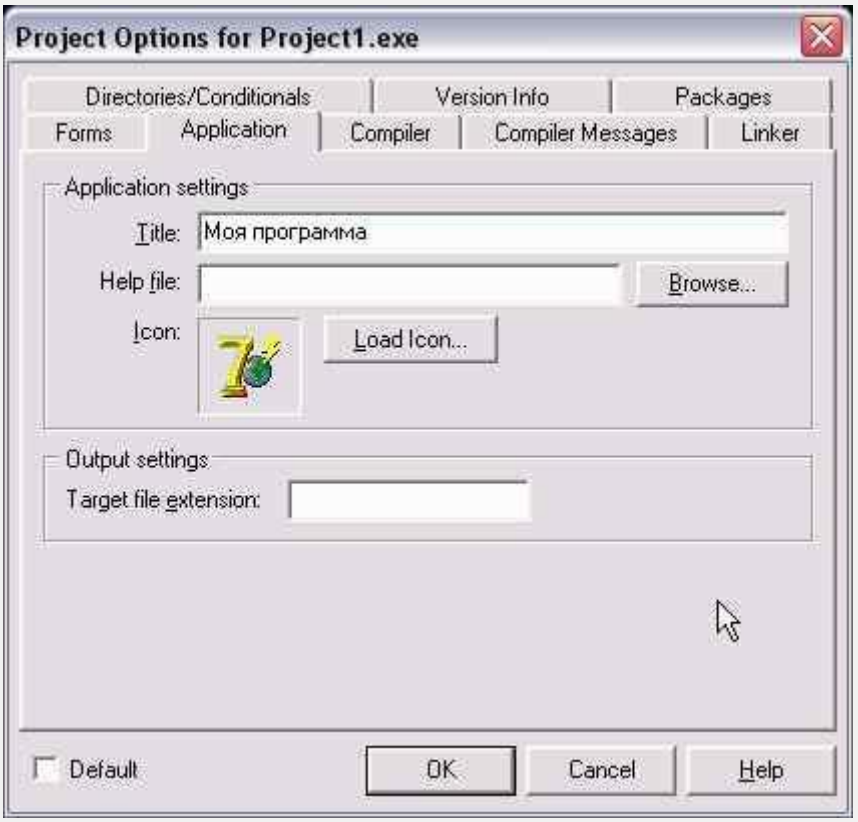
F9 - запуск. В отличие от компиляции, это уже полноценный запуск программы из оболочки Delphi, однако не совсем такой, каким является запуск приложения из Windows. Запуск происходит под отладчиком. Но об этом позже.

Ctrl+F2 - остановка выполнения программы. Это именно то, о чём сказано чуть выше. Если нужно экстренно завершить работу программы, нужно активировать какое=-либо из окон оболочки Delphi и нажать это сочетание клавиш. Программа будет остановлена и можно будет безболезненно продолжить работу.

Все эти команды доступны и напрямую из меню: Run -> Run, Project -> Compile, Run -> Program Reset.

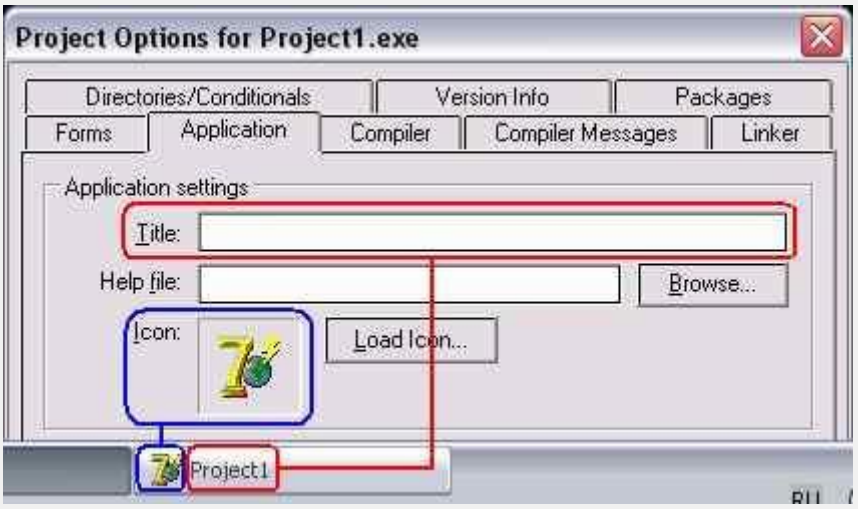
Все основные команды управления проектом вынесены также в виде кнопок на панели инструментов:

Свойства проекта



Окно свойств проекта

Для проекта можно установить множество разнообразных параметров. Все они находятся в окне Project -> Options. В частности, на вкладке Application можно указать заголовок проекта, который будет отображаться на кнопке программы на панели задач. Совет: всегда прописывайте заголовок своей программе, не оставляйте стандартного "Project1" - это резко бросается в глаза. На этой же вкладке можно выбрать иконку (значок) для приложения - файл *.ico размером 32x32 пикселя. В блоке Output settings можно указать расширение выходного файла. Заполнять это поле не обязательно - по умолчанию файлу присваивается расширение .exe. Однако в некоторых случаях эта настройка бывает полезной. Например, экранные программы-заставки представляют собой те же исполняемые exe-файлы, только имеют расширение .scr. Неудобно каждый раз после внесения изменений в программу, чтобы протестировать заставку, переименовывать файл. А прописав в указанное поле "scr" проблема мигом решится.



Вкладка Application в окне свойств проекта влияет на внешний вид кнопки программы на панели задач.

Заключение

В этой статье рассмотрены все базовые навыки для управления проектами. Теперь можно приступить непосредственно к изучению языка и исследованию объектов.

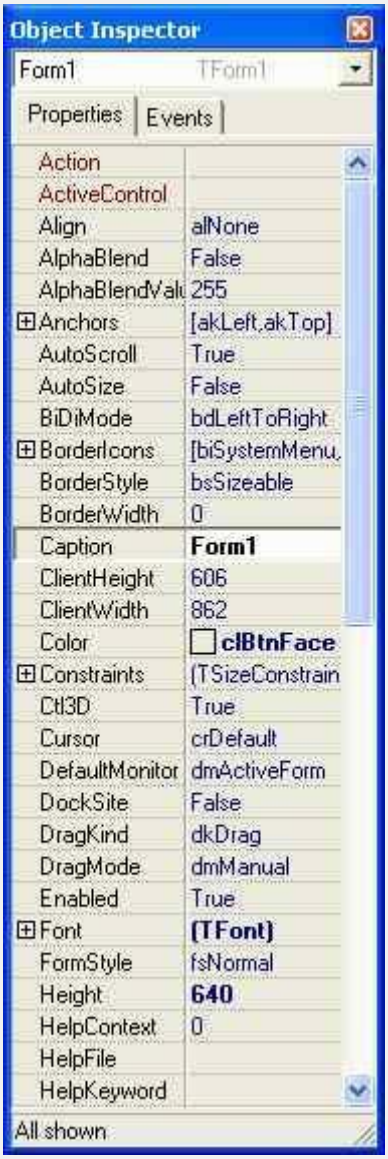
Короткий конспект –Висновок

1.

Урок 6. Обзор свойств компонент

Обзор свойств компонент

Пришло время подробнее ознакомиться со свойствами компонент. И начнём мы с базового элемента любого оконного приложения - формы. Свойств у формы достаточно много, и в разных версиях Delphi их набор может немного отличаться. Здесь будет рассмотрение свойств на основе Delphi 7.



AlphaBlend = True

BorderStyle = bsDialog

Action - определяет объект TAction. Это объект служит для быстрой привязки действий к компонентам, в особенности - к пунктам меню и панелям инструментов. Но может быть привязан и к форме. Для управления TAction служат редакторы TActionList со страницы Standard и TActionManager со страницы Additional.

ActiveControl - определяет элемент, который имеет в данный момент фокус ввода. Если выбрать какой-либо объект во время разработки (design-time), то при запуске приложения этот объект и будет иметь фокус ввода. Также свойство может быть полезно и во время выполнения (run-time) - можно узнать, какой объект "держит" фокус в данный момент, а также можно переместить фокус на любой из объектов. Пример: разместим на форме 2 кнопки - Button1 и Button2, а также TTimer (страница System). Выбрав элемент Timer1, дважды щёлкнем в Инспекторе объектов напротив надписи OnTimer на вкладке Events, т.е. создадим обработчик события и напишем следующее: ActiveControl:=Button2; Теперь, запустив программу, каждую секунду фокус будет перемещаться на Button2.

Align - определяет выравнивание формы на экране. Свойство принимает одно из следующих значений:

alBottom - по нижнему краю;

alClient - вся пользовательская (клиентская) область;

alCustom - выравнивание определяется вызовом методом объекта-родителя;

alLeft - по левому краю;

alNone - без выравнивания;

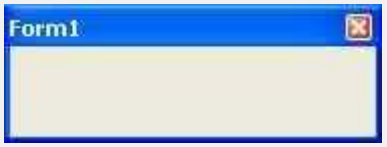
alRight - по правому краю;

alTop - по верхнему краю.

AlphaBlend - включает/выключает прозрачность формы.

AlphaBlendValue - задаёт степень непрозрачности формы: 0 - форма полностью невидима, 255 - полностью видима. Прозрачность активируется только при установке свойства AlphaBlend в True.

Anchors - определяет направления, по которым компоненты "привязываются" к форме. Пример: если установить у формы значения akLeft и akRight этого свойства в True, и точно также сделать у кнопки, то при изменении ширины формы размер кнопки (ширина) также будет изменяться.



BorderStyle = bsToolWindow

AutoScroll - включает автоматическое появление полос прокрутки (Scroll bars) на форме, когда размеров формы недостаточно для отображения всех элементов.

AutoSize - включает автоматическое изменение размеров формы согласно позициям размещённых на ней элементов.

BiDiMode - определяет двунаправленное отображение элемента. В некоторых языках письмо осуществляется не слево-направо, а наоборот. Это свойство создано как раз для этой цели.

BorderIcons - определяет множество кнопок, которые отображаются в заголовке окна:

biSystemMenu - единственный элемент, который не является кнопкой - отвечает за системное меню окна, которое вызывается комбинацией клавиш [Alt]+[Пробел].

biMinimize - кнопка сворачивания (минимизации) окна;

biMaximize - кнопка разворачивания окна;

biHelp - кнопка справки.

Если хотя бы одна из кнопок сворачивания и разворачивания включена, то независимо от состояния другой, отображаются обе (но вторая естественно неактивна). Если выключены обе, они не отображаются вообще. Это не зависит от Delphi - так устроена ОС Windows.

BorderStyle - определяет поведение границ окна и общий тип окна:

bsDialog - диалоговое окно (из кнопок - только "Закрыть", иконки в заголовке окна нет);

bsNone - "чистый лист" (отсутствие у окна границ и заголовка) - применяется обычно для создания заставок во время запуска программы;

bsSingle - обычное окно, но с запретом изменения размеров;

bsSizeable - обычное окно (по умолчанию) - размеры формы можно изменять;

bsSizeToolWin - упрощённое окно с уменьшенным заголовком;

bsToolWindow - упрощённое окно с уменьшенным заголовком без возможности изменения размеров.

BorderWidth - ширина границы окна в пикселах. Граница является невидимой и расположена в пользовательской части формы.

Caption - текст заголовка формы.

ClientHeight, ClientWidth - размер клиентской (пользовательской) части формы, т.е. той, на которой располагаются компоненты.

Color - цвет формы.

Constraints - определяет минимальные и максимальные размеры высоты и ширины формы в пикселах. 0 - любое значение, т.е. без ограничений.

Ctl3D - свойство определяет 3D-вид формы. При выключенном - "плоское" изображение.

Cursor - курсор мыши в тот момент, когда он находится над формой.

DefaultMonitor - определяет, на каком мониторе появится форма. Имеет смысл применять это свойство только при наличии более, чем одного монитора (например, если несколько экранов).

DockSite, DragKing и DragMode - определяют поведение формы при осуществлении операций Drag&Drop.

Enabled - отвечает за общую активность формы. Если установлено в False, форма недоступна.

Font - шрифт, используемый на форме.

FormStyle - стиль формы или её поведение в MDI-приложении (многооконное приложение, где дополнительные формы располагаются "внутри" основной формы). Значения:

fsNormal - обычная форма (значение по умолчанию);

fsMDIChild - дочерняя (подчинённая) форма MDI-приложения;

fsMDIForm - главная форма MDI-приложения;

fsStayOnTop - форма находится поверх всех окон на экране.

Height - высота формы в пикселах. В отличие от ClientWidth является высотой с учётом заголовка и границ формы.

HelpContext, HelpFile, HelpKeyword, HelpType - свойства для связи формы с файлом справки в формате *.hlp.

Hint - текст всплывающей подсказки.

HorzScrollBar - свойство определяет внешний вид и поведение горизонтальной полосы прокрутки окна.

Icon - значок (иконка) формы. Отображается в заголовке слева от заголовка. Задаётся файлом в формате *.ico.

KeyPreview - если свойство установлено в True, то при нажатии клавиш сначала будут вызываться обработчики формы, а только затем обработчики того компонента, который в данный момент имеет фокус ввода. События, связанные с нажатием клавиш - OnKeyDown(), OnKeyPress(), OnKeyUp().

Left - позиция формы на экране (левого верхнего угла) в пикселах.

Menu - позволяет выбрать один из компонентов-меню, который станет главным меню окна, т.е. будет отображаться вверху.

Name - имя формы как объекта. Может содержать только латинские буквы, цифры и знак подчёркивания, и не может начинаться с цифры. Фактически, это то имя, по которому в программе можно обратиться к форме.

ObjectMenuItem - используется при работе с OLE-объектами и позволяет связать пункт меню и OLE-объект: когда объект выделен, пункт меню активен и наоборот.

OldCreateOrder - определяет, когда происходят события OnCreate() и OnDestroy() формы. Если установлено в False, то OnCreate() произойдёт после вызова всех конструкторов, а OnDestroy() - после вызова всех деструкторов. Начальное значение - False, изменять не рекомендуется.

ParentBiDiMode - изменение свойства BiDiMode согласно значению объекта-предка формы.

ParentFont - изменение шрифта (Font) согласно значению объекта-предка.

PixelsPerInch - пропорции шрифта в системе (точек на дюйм).

TransparentColor = True

PopupMenu - позволяет указать контекстное меню (объект TPopupMenu) для формы. Это меню вызывается нажатием правой кнопки мыши.

Position - определяет начальную позицию формы на экране, т.е. в момент её появления. Основные значения:

poDesigned - появление в том месте, в каком форма расположена в design-time;

poDesktopCenter - по центру рабочего стола (рекомендуемое значение);

poScreenCenter - по центру экрана;

poMainFormCenter - по центру главной формы приложения (для главной формы не имеет смысла).

PrintScale - определяет размеры формы при выводе её изображения на печать.

Scaled - включает масштабирование формы в соответствии с заданным значением свойства PixelsPerInch.

ScreenSnap - если установлено в True, то форма будет автоматически "прилипать" к краям экрана в момент перемещения.

SnapBuffer - определяет расстояние (в пикселах), на котором форма будет "прилипать" к краю экрана.

ShowHint - включает/выключает показ всплывающей подсказки (Hint).

Tag - специальное свойство, которое есть у всех объектов. Специального применения для этого свойства нет, поэтому оно используется для разных целей в конкретной ситуации. Свойство удобно в том случае, если нужно хранить некоторое целое число - не придётся заводить дополнительную переменную.

Top - позиция формы (левого верхнего угла) на экране в пикселах.

TransparentColor - включает/выключает прозрачность определённого цвета формы.

TransparentColorValue - задаёт цвет, который будет прозрачным.

UseDockManager - используется при реализации Drag&Drop технологии, предоставляя дополнительные возможности этого метода взаимодействия.

VertScrollBar - определяет внешний вид и поведение вертикальной полосы прокрутки окна.

Visible - определяет видимость формы на экране.

Width - ширина окна в пикселах, включая границы.

WindowMenu - свойство-аналог свойства Menu, но используемое при создании MDI-форм.

WindowState - одно из состояний окна:

wsNormal - обычное состояние (занимает часть экрана);

wsMinimized - окно свёрнуто;

wsMaximized - окно развёрнуто на весь экран.

В итоге мы получаем огромное количество свойств, способных изменить как внешний вид формы, так и её поведение, а также поведение компонент, расположенных на ней. Но данная статья незаметно познакомила Вас не только со свойствами формы, но и со свойствами большинства компонент. Дело в том, что компоненты имеют общих "предков", т.е. тех объектов, от которых они образованы, поэтому свойства компонент очень похожи и большая их часть просто-напросто совпадает. Если посмотреть на свойства кнопки (TButton), то сразу можно заметить, что большинство свойств - те же самые, что и у формы. Это позволяет быстро научиться работать с любым незнакомым объектом.

Примечания

Стоит сделать несколько примечаний насчёт свойств.

Свойства прозрачности формы (AlphaBlend, AlphaBlendValue, TransparentColor и TransparentColorValue) корректно работают только на ОС Windows XP и следующих версиях. В предыдущих версиях ОС изменение значения этих свойств не производит визуального изменения формы.

Свойства, названия которых начинаются со слова Parent (англ. - родитель), в большинстве случаев связывают значения некоторых свойств со значениями соответствующих свойств объекта-родителя. Так, кнопка (TButton) имеет свойство ParentFont и свойство Font, отвечающее за шрифта текста на этой кнопке. Но и сама форма имеет свойство Font. В результате, если у кнопки установить ParentFont в True, а затем изменить шрифт у формы, то шрифт у кнопки изменится соответствующим образом. Это позволяет быстро изменять одни и те же свойства у большого числа компонент. Другие подобные свойства - ParentShowHint, ParentColor, ParentBiDiMode.

Свойство Cursor, отвечающее за курсор, есть у большинства компонент. Но при перемещении курсора его вид изменяется на тот, который задан у самого "дальнего" объекта. Т.е. если и формы и у кнопки заданы разные формы курсора, то при перемещении над кнопкой будет использоваться курсор, заданный у самой кнопки. Число "вложений" одних компонент в другие может быть довольно большим.

Заключение

В этой статье рассмотрены свойства формы и основные свойства компонент. Изменяя свойства, можно настроить объекты так, как это требуется для реализуемой программы. Объектно-ориентированное программирование в основном и сводится к управлению свойствами объектов.

Короткий конспект –Висновок

1.

Урок 7. Обзор палитры компонент: Standard, Additional

Обзор палитры компонент: Standard, Additional

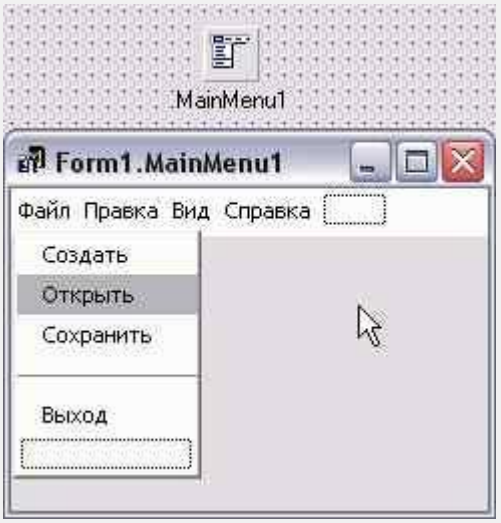
Чтобы создать интерфейс программы, нужно иметь выбор из различных компонент. В Windows существует множество элементов, и все они доступны в Delphi для использования. Познакомимся с двумя первыми вкладками Палитры Компонент - Standard и Additional. Именно на них расположено большинство элементов, наиболее часто используемых в программах.

Standard

Из названия вкладки следует, что компоненты, представленные на ней, являются стандартными, системными. Так и есть.

Frames - позволяет разместить на форме так называемый "фрейм". Фрейм из себя представляет другое окно. Чтобы создать окно-фрейм, следует выбрать пункт меню File -> New -> Frame, либо выделить значок Frame в окне File -> New -> Other на вкладке New. До тех пор, пока в приложении не будет ни одного фрейма, использовать данный объект не удастся. Фреймы удобны в том случае, когда какие-либо настройки запрашиваются во время работы программы в виде отдельной формы, а также, например, на одной из вкладок основной формы.

MainMenu - основное меню окна. Связать его с формой можно через свойство Menu формы. При двойном щелчке по значку MenuMenu появляется дизайнер меню, в котором можно добавлять и удалять пункты. В Caption задаётся текст пунта меню. Чтобы создать черту-разделитель, следует в свойство Caption прописать знак "минус" ("-", без кавычек).



PopupMenu - контекстное меню (вызывается правой кнопкой мыши). Его можно привязать ко многим компонентам (как правило, это делается через свойство PopupMenu у компонента, которому ассоциируется это меню). Одно и то же меню может быть привязано к нескольким компонентам.

Label - текстовая метка (надпись) на форме. Используется для отображения любого текста в окне. Текст задаётся в свойстве Caption. Свойство Font позволяет настроить шрифт текста.

Edit - поле ввода. Используется для ввода любых данных (текста, числа и т.д.), представленных одной строкой. Свойство ReadOnly позволяет запретить редактирование текста в поле. Текст хранится свойством

Text. Свойство MaxLength позволяет задать максимальное число символов, которое может быть введено в поле. Значение 0 означает, что ограничение не установлено.

Memo - многострочный Edit. Используется для ввода больших объёмов текста. Свойство ScrollBars позволяет указать полосы прокрутки, которые будут отображаться у поля при недостатке места для всего текста:

- **ssNone** - без полос прокрутки;
- **ssHorizontal** - горизонтальная полоса прокрутки;
- **ssVertical** - вертикальная;
- **ssBoth** - обе: и горизонтальная, и вертикальная.
- **Button** - кнопка, самая что ни есть обыкновенная. Caption - текст на кнопке.
- **CheckBox** - флажок-переключатель. Состояние хранится в свойстве Style:
- **cbUnchecked** - не отмечен;
- **cbChecked** - отмечен;
- **cbGrayed** - затемнён.

Само состояние следует изменять с помощью свойства Checked. Таких флажков может быть несколько и каждый может быть отмечен независимо от остальных.

RadioButton - радио-переключатель. Аналогичен CheckBox, но при наличии нескольких переключателей может быть выбран только один из них. Свойство Checked позволяет изменить состояние переключателя.

ListBox - список текстовых строк. Используется для выбора одного из вариантов. Строки задаются в свойстве Items. Чтобы изменить исходные значения, следует щёлкнуть по кнопке с тремя точкам ("...") в строке свойства Items и в открывшемся редакторе ввести требуемые значения. Свойство MultiSelect позволяет включить/выключить одновременный выбор нескольких строк из списка.

ComboBox - ещё один список для выбора, но выпадающий - на экране видна всего одна строка, а сам список появляется при нажатии на кнопку со стрелкой. Установив свойство Style в csDropDownList можно запретить ввод произвольного значения в ComboBox. Текст выбранной строки доступен в свойстве Text.

ScrollBar - полоса прокрутки. Свойство Kind определяет ориентацию полосы на форме: sbHorizontal - горизонтально, sbVertical - вертикально.

GroupBox - "контейнер" для компонент. Используется для объединения элементов в отдельные области. При перемещении GroupBox'а все компоненты, расположенные на нём, также перемещаются. Для размещения компонент в этом контейнере следует после выбора их на Палитры компонент щёлкнуть по самому контейнеру (а не по форме, как обычно). У GroupBox можно задать текст заголовка (свойство Caption).

RadioGroup - группа из нескольких RadioButton'ов. Тексты строк задаются в свойстве Items. Номер выбранной строки - свойство ItemIndex (нумерация строк начинается с нуля!).

Panel - практически такой же контейнер, что и GroupBox, однако без заголовка и с возможностью отключения рельефных границ.

ActionList - позволяет управлять действиями (Actions), которые привязываются к пунктам меню, кнопкам и т.д.

Additional

Компоненты с этой вкладки используются реже, но в некоторых случаях без них не обойтись. Среди них есть как компоненты для создания украшений, так и компоненты-модификации стандартных компонент.

BitBtn - кнопка, на которой помимо текста можно разместить изображение. За изображение отвечает свойство Glyph. Формат - *.bmp. Чтобы добавить картинку, следует щёлкнуть по кнопке с тремя точками в строке свойства Glyph, в открывшемся окне нажать Load... и указать файл с картинкой.

SpeedButton - ещё один тип кнопки. Также поддерживает добавление изображения. Помимо этого кнопку можно сделать "плоской", т.е. без границ - за это отвечает свойство Flat.

MaskEdit - аналог Edit, но используется для ввода текста по маске. Например, можно указать ввод только цифр. Маска задаётся в свойстве EditMask. Имеется несколько стандартных вариантов.

StringGrid - таблица, в каждой ячейке которой может быть расположен текст. Часть ячеек можно сделать фиксированными (свойства FixedCols и FixedRows). Количество строк и столбцов таблицы задаётся соответственно свойствами RowCount и ColCount.

DrawGrid - таблица, но в ячейках помимо текста могут располагаться и другие объекты, например изображения.

Image - изображение, которое можно разместить на форме. Изображение задаётся в свойстве Picture. Свойство Center позволяет отцентрировать изображение, если размеров недостаточно для его полного отображения. Stretch позволяет задать сжатие/растяжение картинки под размеры компонента, Proportional указывает, следует ли при этом сохранять пропорции исходного изображения.

Shape - один из вариантов геометрических фигур. Тип фигуры определяется в свойстве Shape:

- **stCircle** - окружность;
- **stEllipse** - эллипс;
- **stRectangle** - прямоугольник;
- **stRoundRect** - прямоугольник с закруглёнными углами;
- **stRoundSquare** - квадрат с закруглёнными углами;
- **stSquare** - квадрат.
- Свойства **Pen** и **Brush** позволяют задать стиль границ фигуры и её внутреннюю заливку.

Bevel - компонент для создания рельефа на форме - линий, окошек и т.д.

ScrollBox - прокручиваемая область, на которой можно разместить другие элементы.

CheckListBox - аналог ListBox, но к каждой строке добавляется флажок. Таким образом более удобно выбирать значения из списка.

Splitter - разделитель для создания областей изменяемых размеров. Компонент следует разместить между двумя элементами, которые должны быть изменяемого размера. После этого следует настроить выравнивание (Align) элементов и Splitter автоматически начнёт работать.

StaticText - текстовая метка, аналог Label, но в отличие от него является полноценным элементом управления Windows. Использовать StaticText имеет смысл в том случае, если нужно разместить текст поверх какого-либо другого компонента. С Label этого сделать не удастся.

ControlBar - один из типов панелей инструментов. Автоматически перетаскивается по области, отведённой для панелей инструментов.

ApplicationEvents - используется для доступа к некоторым свойствам и событиям объекта TApplication. Этот объект - и есть само приложение. Работать с этим объектом можно и программно, но с помощью этого компонента всё же удобнее. Он является невизуальным.

ValueListEditor - таблица из двух колонок - поля и значения. Некое подобие StringGrid.

LabeledEdit - Label и Edit "в одном флаконе". Оба компонента являются полностью настраиваемыми.

ColorBox - выпадающий список для выбора цвета.

Chart - мощный объект для построения диаграмм и графиков.

ActionManager - используется для управления действиями TAction. Аналогичен ActionList со страницы Standard, но имеет намного больше возможностей. При создании приложения на основе этой технологии рекомендуется использовать именно ActionManager.

ActionMainMenuBar - меню, работающее на основе TAction.

ActionToolBar - аналогично, панель инструментов на основе TAction.

XPColorMap, StandardColorMap, TwilightColorMap - стандартные цветовые схемы для объектов на основе TAction (кстати, они называются Action-band компонентами).

CustomizeDlg - диалог для настройки Action-band компонентов. Такой диалог используется, например в Microsoft Word для индивидуальной настройки панелей инструментов и меню. Помимо этого, Action-band компоненты позволяют сохранять и восстанавливать состояние всех объектов. Это делает приложение полностью настраиваемым.

Заключение

Вот мы кратко и познакомились с компонентами первых двух вкладок Палитры компонент - Standard и Additional. Теперь Вы можете поэкспериментировать с созданием интерфейса для своей программы. В следующей главе мы начнём изучение языка Object Pascal.

Короткий конспект –Висновок

1.

Урок 8. Pascal - первое знакомство

Pascal - первое знакомство

Итак, достаточно кратко мы познакомились со способами создания интерфейсов программ. Теперь пришло время начать изучать сам язык программирования - Pascal. В Delphi используется его более усовершенствованная и нацеленная на объектно-ориентированное программирование версия - **Object Pascal**.

Язык программирования — формальная знаковая система, предназначенная для описания алгоритмов в форме, которая удобна для исполнителя (например, компьютера).

Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах.

Ближе к делу...

В Pascal любая программа пишется по определённой заготовке. Вот она:

```
program название_программы;

  {раздел описаний}

begin

  {раздел реализации}

end.
```

В проекте Delphi этой заготовке соответствует файл проекта - *.dpr (открывается с помощью меню Project » View Source). Если после создания нового проекта сразу открыть

этот файл, то всё основное там уже будет написано - писать это собственноручно не требуется. К примеру, в "обычном" Pascal такой заготовки нет и всё пишется "с нуля".

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Заготовка кода проекта (*.dpr)

Идентификаторы.

Идентификатор (identifier) - одно из самых важных понятий языка программирования. Что такое идентификатор? Всё верно, это то, по чему идентифицируется (узнаётся, находится, определяется) что-либо.

Фактически, идентификатор - это уникальное имя. Нужно чётко усвоить правила создания имён идентификаторов. Они довольно просты. При их несоблюдении, т.е. ошибочном вводе имени, программа просто не скомпилируется.

1. Имя идентификатора может состоять из латинских букв, причём как верхнего, так и нижнего регистра (A, B, ..., Z; a, b, ..., z), цифр (0, 1, ..., 9) и знака подчёркивания (_). Никакие другие символы не могут быть использованы.
2. Первый символ в имени не должен быть цифрой, т.е. он может быть латинской буквой или знаком подчёркивания. Все последующие символы могут быть теми, которые указаны в п.1.
3. Максимальная длина имени - 255 символов. Следует отметить, что это ограничение формально. Вы можете задать имя и из большего числа символов, но компилятор воспримет только первые 255.
4. Язык Pascal нечувствителен к регистру символов (имеется ввиду вообще весь язык) и регистру имён идентификаторов в частности. Т.е. и ABC и abc и даже aBc - это абсолютно одно и то же. Многие языки чувствительны к регистру. Хорошо это или плохо - утверждать не стоит. Однако большинство мнений сходятся на том, что нечувствительность лучше. По крайней мере, это не порождает большого числа ошибок при написании имён разным регистром.

Переменные и типы данных - первое знакомство.

Понятие переменной в языке программирования похоже на понятие переменной в математике. Переменная задана каким-либо именем и хранит некоторое значение. Переменная (variable) - именованная ячейка памяти, имя которой можно использовать для осуществления доступа к данным, находящимся по данному адресу.

Переменная определяется:

- именем;
- типом данных;
- значением.

Мы подошли к одному из практических применений идентификаторов. Каждая переменная имеет свой идентификатор, т.е. имеет своё уникальное имя. Двух переменных с одним именем быть не может (в общем

случае, с частными случаями мы столкнёмся в дальнейшем). Тип данных - это диапазон (набор) всех значений, которые может принимать данная переменная. Pascal - язык со строгой типизацией данных. Это означает, что мы не сможем использовать переменную, не указав тип данных для неё. Ну и наконец, значение - это непосредственно то, что хранится в конкретный момент времени в переменной. Это может быть число, символ, текст и т.д.

Визуально переменную можно представить как некую коробочку, в которую можно что-то положить. Например, у нас есть коробочка для разноцветных шаров, на которой написано "Шар №1". В этом случае "Шар №1" - это имя нашей переменной (на других коробочках написано что-то другое) и по этому имени мы можем найти именно эту коробочку среди остальных. Значением в данном случае выступает шар (точнее - его цвет), который мы положили в эту коробочку (например, красный, зелёный, синий). Ну а тип данных - цвет шара, т.е. мы не знаем, какой именно шар окажется в коробочке, но тем не менее знаем, что он будет какого-то цвета.

Константы

Понятие константы используется в математике, физике и других науках. Оно означает, что данная величина не изменяется с течением времени. Абсолютно такое же значение константы и в программировании. Константа (constant) - это переменная, значение которой не меняется во время выполнения программы. Следует отметить, что значение константы должно быть известно ещё до запуска программы, т.е. до этапа компиляции. Пример константы: g ≈ 9.8 (ускорение свободного падения тел на Земле). В Delphi "прописаны" также общеизвестные константы, самым ярким примером которых является число "пи", имя этой константы - PI. Таким образом, эту константу можно свободно использовать в своей программе, не задавая предварительно её значение. Ещё одно из достоинств - константа "пи" занесена с достаточно большой точностью, в то время как для "ручного" объявления придётся где-то "подсмотреть" цифры после запятой (в расчётах малых значений применение пи = 3.14 может давать некоторые погрешности результата).

Примечание: Помимо имён переменных, идентификаторы встречаются практически повсюду. Имена всех компонент (свойство Name); названия модулей (а соответственно и имена файлов, составляющих проект); название приложения, указываемое после ключевого слова program в файле проекта - всё это тоже идентификаторы, а соответственно, их имена также должны подчиняться правилам, описанным выше.

Зарезервированные слова.

Зарезервированные слова - слова языка, которые играют служебную роль. Использовать часть из этих слов в качестве имён, т.е. идентификаторов, запрещено. В редакторе кода эти слова отображаются цветом, отличным от цвета основного кода. Вот перечень все зарезервированных слов Object Pascal:

and	export	label	resourcestring
array	exports	library	set
as	file	mod	shl
asm	finalization	nil	shr
begin	finally	not	string
case	for	object	then
class	function	of	threadvar
const	goto	or	to

constuctor	if	out	try
destructor	implementation	packed	type
dispinterface	in	procedure	unit
div	inherited	program	until
do	initialization	property	uses
downto	inline	raise	var
else	interface	record	while
end	is	repeat	with
			xor

Заключение

В этом уроке мы познакомились с некоторыми элементами языка программирования, а также особенностями языка Pascal. В следующем уроке мы продолжим изучение и начнём "испытывать" язык на конкретных примерах.

Короткий конспект –Висновок

1.