

## Описание программы

**Дизель-Паскаль** - это платформа для разработки и выполнения приложений, главным образом платформа создавалась для создания корпоративных приложений способных работать в гетерогенной среде на разных платформах без каких либо переделок под конкретную платформу, как средство автоматизации бизнеса, как свободная альтернатива таких продуктов как 1С. При разработке преследовалась цель запуска программ без перекомпиляции на разных платформах, внесение изменений в приложение без перекомпиляции. Интерфейс и функционал среды аналогичен интерфейсу Lazarus, имеет встроенные интерпретаторы языков Паскаль и Дизель-Паскаль. Оба языка поддерживают ООП. Можно объявлять классы, поддерживается наследование и полиморфизм. Встроенный интерпретатор Паскаль довольно тесно совместим с оригиналом (Free Pascal, Delphi) за исключением некоторых нюансов. Приложения Дизель-Паскаль представляют из себя XML файл содержащий ресурсы (формы, фреймы, модули данных) и код для интерпретатора Паскаль / Дизель-Паскаль. Приложения выполняются машиной, компиляция кода во внутренний формат происходит при запуске приложения, поэтому приложения Дизель-Паскаль будут работать без перекомпиляции и переделок на любой платформе, под которую удастся собрать сам Дизель-паскаль. Можно сказать что Дизель-Паскаль является дополнением к Lazarus и предназначен для разработки бизнес программ либо других программ автоматизации какой либо деятельности, но подразумевающей немного иной подход в построении и выполнении приложения. Дизайнер и машина Дизель-Паскаль могут загружать приложения как локально, так и по протоколу ftp или http.

## Поддерживаемые компоненты и технологии

При разработке Дизель-Паскаль использовались только свободные компоненты, все кросс-платформенно и бесплатно. Исходный код Дизель-Паскаль открыт, Вы всегда можете сделать модификацию для собственных нужд и добавить поддержку любых доступных компонент и технологий.

-Дизель-Паскаль может работать с любыми базами данных через компоненты ZEOS, но основной упор делается на FireBird SQL server или Red Database. Для

доступа к FireBird используется форк компонент IBX собственной модификации по аналогии с FIBPlus. Смотреть описание IBX.

-Работа с ftp, http, pop, smtp, imap через компоненты Synapse (можно обмениваться данными с web сервисами использующими архитектуру REST или протокол SOAP)

-Встроенные парсеры JSON и XML (XML с поддержкой чтения UTF-8 и cp1251, запись только в UTF8).

-Работа с DBF в кодировке cp1251 и 866

-MemoryDataset, BufDataSet, SDFDataSet

-Визуальные и не визуальные компоненты: Standard, Additional, Common Control, Dialogs, System, Misc, DataControls

-Собственная палитра компонентов VisualTech - это компоненты с расширенными свойствами и сектой TxDBGrid с измененным оформлением (поддерживает вывод до 2х полей в одном столбце, перенос текста в ячейках, вывод иконок, подсветку строк и ячеек) Смотреть описание

-Электронные таблицы в формате xls, xlsx, ods без использования Excel или LibreOffice с помощью компонент fpSpreadSheet, имеется встроенное средство отображения электронных таблиц с поддержкой формул fpSpreadSheet\_Visual.

-Графики (TChart)

-Генератор отчетов LazReport (модифицированная версия с поддержкой матричных штрихкодов QRCode, DataMatrix, Aztec, PDF417 и экспортом в PDF). Смотреть описание.

-Звуковая библиотека UOS

## Как работает Дизель-Паскаль

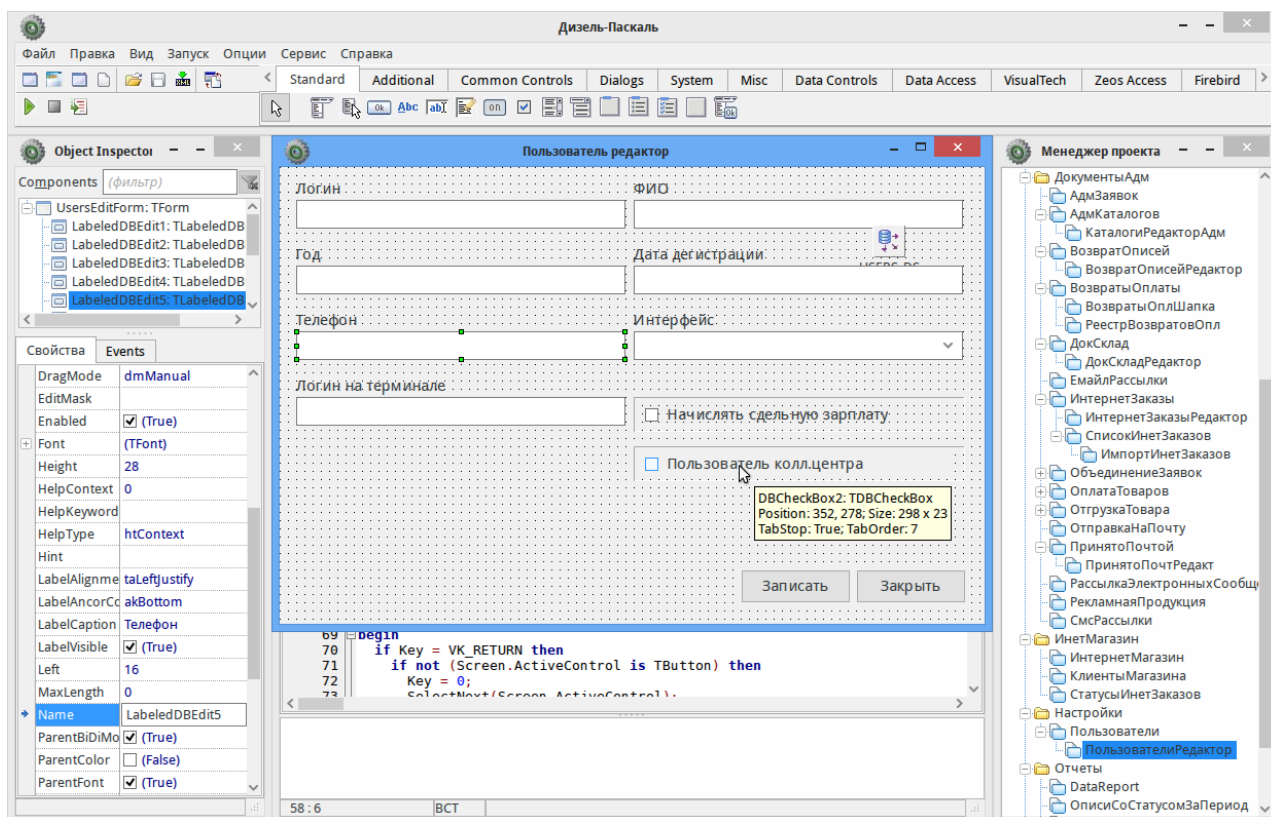
Приложение Дизель-Паскаль разрабатывается в дизайнере (CrossDesigner), визуально строятся формы, фреймы и модули данных, пишется код связующий компоненты и задающий логику работы приложения, все точно так-же как в Lazarus или Delphi. Затем, всё приложение с формами и кодом сохраняется в один XML файл, т.е. по сути приложение Дизель-Паскаль это текстовый файл, при необходимости он может быть защищен шифрованием от несанкционированного чтения и изменения. Этот файл размещается на любом доступном ресурсе, на ftp, http, smb или локально. Для выполнения приложения требуется запустить машину Дизель-Паскаль (CrossMachine), в качестве параметра ей передается полный путь к приложению, машина загрузит приложение с указанного ресурса и всё целиком за один раз скомпилирует во внутренний формат и затем запустит на выполнение. Машина не транслирует приложение в какой-либо байткод а создает после разбора кода синтаксическое дерево, в узлы которого добавлены инструкции для выполнения, такой подход может потребовать значительных ресурсов памяти, хотя мне удалось запустить корпоративное приложение состоящее из 180 форм и модулей и примерно 50тысяч строк кода, работающее с базой данных, на машине с Pentium4 + WinXP+256мб памяти. Для запуска приложения удобно использовать Менеджер приложений (DManager) он сохраняет пути и названия приложений, которые можно запустить на

выполнение или загрузить в дизайнер. Могу добавить что приложения компилируются и запускаются довольно быстро, то-же приложение из 180 модулей и 50тыс. строк кода на машинах Core2Duo с 1гб ОЗУ запускается за 2-3 секунды.

Приложение можно запускать из под дизайнера, предварительно сохранив, в этом случае приложение запускается в режиме отладки и обменивается с дизайнером информацией. **В ы можете ставить точки останова и выполнять программу по шагам для обнаружения ошибок.**

## Описание дизайнера приложений

Дизайнер имеет интерфейс аналогичный Lazarus и использует некоторые его элементы, это было сделано специально, чтобы не требовались дополнительные затраты на освоение и переучивание разработчиков знакомых с Delphi/Lazarus. Если Вы знакомы со средой Lazarus, разобраться в Дизель-Паскаль не составит труда. Дизайнер содержит главный модуль с палитрой компонентов, инспектор объектов, редактор кода, менеджер проекта и дизайнер форм.



### Дизайнер в ОС Linux

После первого запуска дизайнера, разместите элементы (инспектор, редактор кода, менеджер проекта) как вам удобно, затем, чтобы дизайнер запомнил их расположение, выполните в меню **Опции->Сохранить расположение**

элементов.

Меню **Опции->Параметры** задают параметры работы дизайнера форм, подсветки синтаксиса и параметры клавиш. По умолчанию для автозавершения кода используется сочетание клавиш <Ctrl>+<Shift>+<C> и если у Вас используется переключение раскладки клавиатуры как <Ctrl>+<Shift> то автозавершение работать не будет пока Вы не поменяете сочетание клавиш для автозавершения.

Меню **Опции->Опции проекта** (рис ниже), здесь указывается имя программы (без пробелов и специальных символов), заголовок приложения, автор, заставка и иконка. Раздел "Проверять версию движка" нужен для того, чтобы при запуске на движке с версией ниже указанной, пользователь получил сообщение "Приложение не может работать с этой версией Дизель-Паскаль, требуется обновить программу"

**Опции проекта**

Имя программы (без пробелов и символов)  
Predprinimatel

Заголовок  
Предприниматель

Автор  
Копнин Юрий

Иконка

Заставка

Проверять версию движка

Старшая 1 Младшая 5 Ревизия 9

OK Отмена

**Редактор кода** работает так-же как и редактор Lazarus, имеется помощник набора текста, который можно вызвать нажав на клавиши на клавиатуре <Ctrl> + <Пробел>, работает связь мышью, удерживая <Ctrl> вы можете щелкать мышью на доступных объектах которые связаны между собой, например щелкнув на имени метода можно перейти от объявления к реализации или наоборот, связанные объекты подсвечиваются при наведении на них мыши, когда удерживается клавиша <Ctrl>.

Объявив метод в классе, Вы можете с помощью клавиш автозавершения кода создать его реализацию нажав <Ctrl>+<Shift>+<C>. Чтобы быстро перейти от реализации метода к объявлению или от объявления к реализации нажмите на клавиатуре <Ctrl>+<Shift>+<стрелка вверх>.

Если у вас для переключения раскладки клавиатуры используется сочетание <Ctrl>+<Shift>, то <Ctrl>+<Shift>+<C> в редакторе работать не будет, Вам нужно в настройках дизайнера (Опции->Параметры->Управляющие кнопки) задать другое сочетание клавиш для автозавершения.

## Получение информации о встроенных процедурах и функциях, импортированных типах данных

---

Чтобы узнать список встроенных процедур и функций, в дизайнере выполните **Сервис->Список импортированных процедур и функций**.

Чтобы получить информацию о типе данных, которые не объявлены в скрипте программы а импортированы из FPC-Lazarus, установите курсор на имени типа и затем выполните **Сервис->Информация о импортированном типе**. Для примера, если поставить курсор в слово TCanvasState получим следующий список с возможными значениями:

TCanvasState

Множество (Set)

Множество из TCanvasStates

TCanvasStates

Перечисление (Enum)

MinVal 0

MaxVal 4

(csHandleValid, csFontValid, csPenvalid, csBrushValid, csRegionValid)

## Шифрование приложения

---

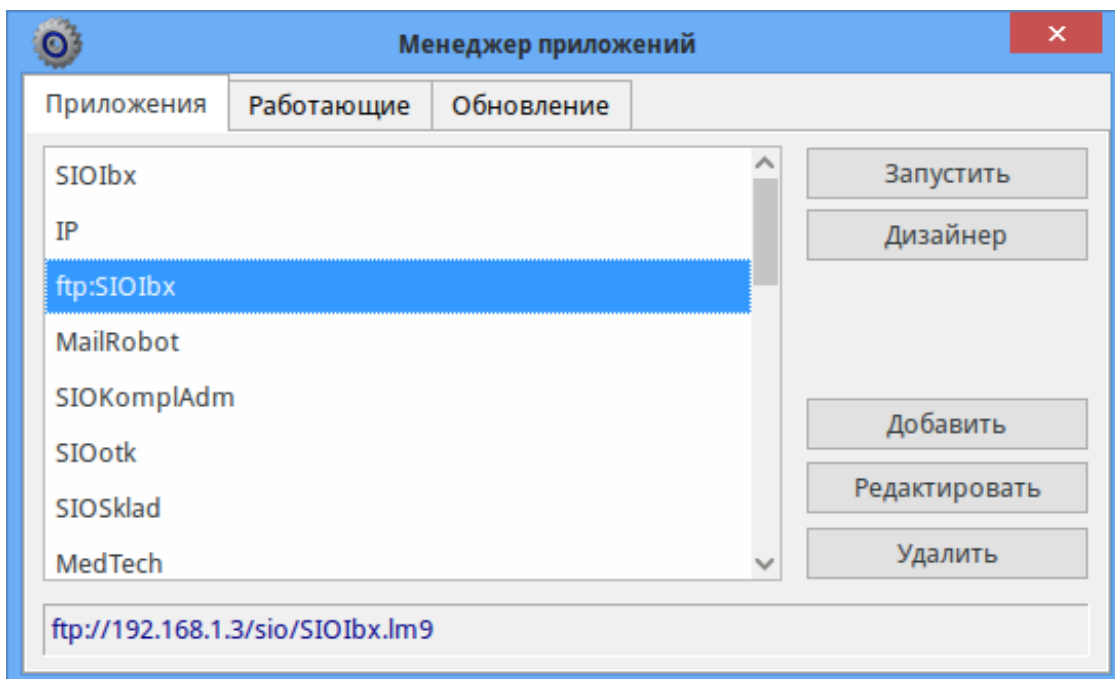
Чтобы зашифровать приложение, его не нужно загружать в дизайнер. В дизайнере в меню выполните Файл->Зашифровать приложение, программа попросит указать путь к приложению, после этого создаст рядом с приложением файл с тем-же именем, но расширением \*.sm9. По расширению система понимает

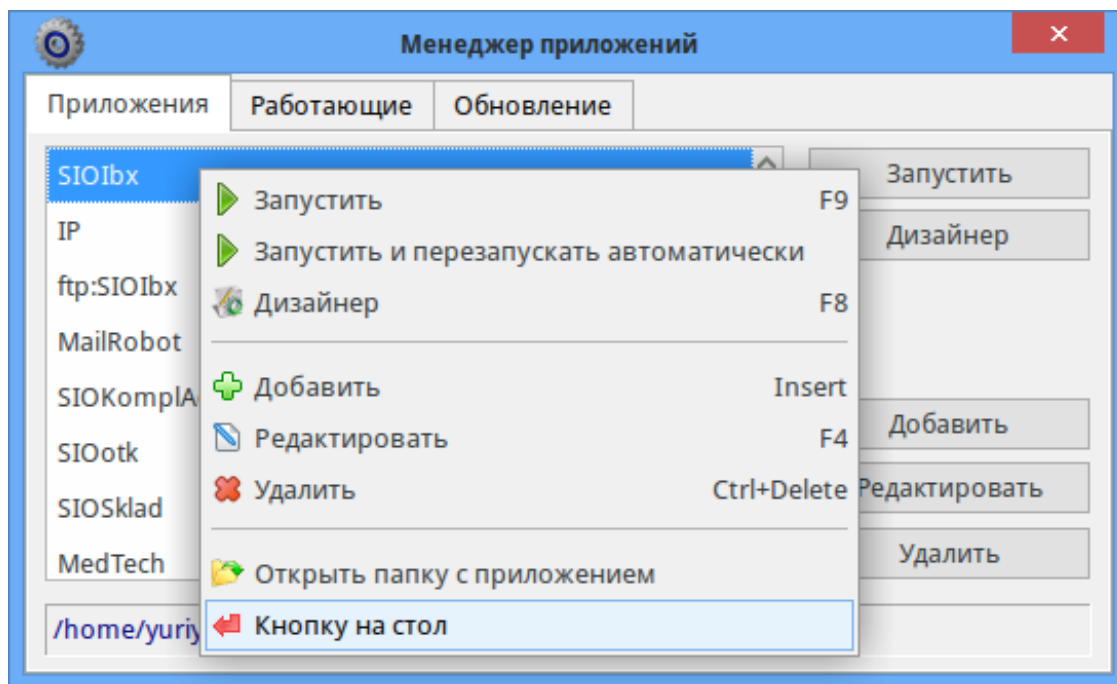


зашифрован файл или нет.

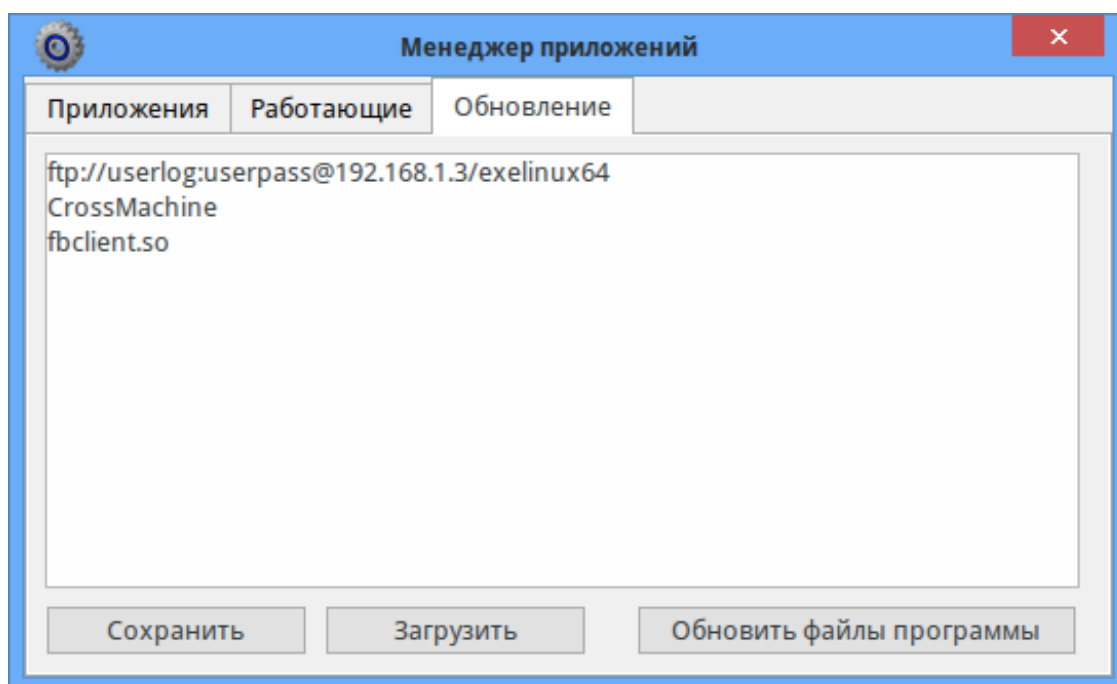
## Менеджер приложений

Менеджер приложений создан для удобства пользователя, он хранит список приложений, которыми пользуется пользователь, приложения можно запустить на выполнение или загрузить в дизайнер. Добавить, удалить, изменить приложение в менеджере можно с помощью кнопок <Добавить>, <Редактировать>, <Удалить>. Выбранное приложение запускается на выполнение кнопкой <Запустить>, с помощью кнопки <Дизайнер> приложение загружается в дизайнер приложений. Запустить приложение можно просто выбрав его в списке и нажав клавишу <Enter> или клавишу <F9> на клавиатуре. Все горячие клавиши можно увидеть в контекстном меню, которое содержит ещё и дополнительные опции. Например опция "Запустить и перезапустить автоматически", будет запускать приложение в случае если вы его закроете либо оно "вылетит" в случае ошибки, менеджер будет перезапускать приложение вновь и вновь пока открыт сам менеджер, это может быть важным если приложение должно работать круглосуточно и в автоматическом режиме выполнять какие-то действия. Если приложение падало и менеджер перезапускал его, это можно увидеть на вкладке "Работающие", там приложение которое перезапускалось, отображается с цифрой в фигурных скобках, цифра показывает сколько раз приложение было перезапущено. Вы можете сами провести эксперимент закрыв приложение запущенное таким образом и увидите как менеджер его перезапустит, а цифра в фигурных скобках на вкладке работающие изменится. Чтобы завершить приложение запущенное таким образом, есть два пути, первое, переключиться на вкладку "Работающее" и убить приложение в списке или закрыть сперва менеджер, затем приложение. Опция "Кнопку на стол" создает ярлык на рабочем столе для запуска приложения на прямую без использования менеджера приложений.





Закладка "Обновление" нужна чтобы обновлять сам движок Дизель-Паскаль (но при этом сам менеджер не обновляется). Чтобы настроить обновление движка, на вкладке обновление в первой строке указывается путь к источнику обновлений, это может быть ftp, http или папка на файловом сервере, ниже в следующих строках указываются файлы которые будут обновляться, на скриншоте ниже видно, что для обновления указаны 2 файла CrossMachine и библиотека fbclient.so. Само обновление запускается кнопкой <Обновить файлы программы>.



Среда поддерживает 2 встроенных языка - Паскаль и Дизель-Паскаль, но их нельзя использовать одновременно в одном приложении, приложение всегда пишется либо только на Паскаль либо только на Дизель-Паскаль. Паскаль довольно тесно совместим с оригиналом Free Pascal или Delphi. Но есть и существенные различия, например все импортированные функции, типы и классы добавлены в одно адресное пространство и все это доступно в любом модуле, например класс TForm доступен в любом месте кода, если в Delphi или Lazarus требуется подключать модуль forms в раздел uses, чтобы класс TForm был доступен в модуле, то в Дизель-Паскаль это не требуется. В раздел uses подключаются только модули Вашего приложения которые требуются в текущем модуле. Чтобы переопределить существующий метод в классе наследнике, не требуется его определять как virtual и override, просто переписываем метод с таким-же именем. Переопределять можно только созданные Вами методы, обработчики событий переопределять нельзя. Внутри переопределенного метода можно так-же использовать инструкцию inherited чтобы вызвать метод базового класса.

## **Что поддерживает интерпретатор в реализации языков Паскаль и Дизель-Паскаль**

---

Для хранения большинства типов используется тип Variant, но не смотря на это синтаксис языков требует указание типа данных при объявлении и совместимость типов строго проверяется. При написании кода действуйте так-же как привыкли в Lazarus или Delphi.

- Простые типы данных (integer, double, string и т.д.).
- Агрегатные типы данных class
- Массивы (инициализация массивов при объявлении не поддерживается)
- Перечисления
- Множества
- процедуры, функции, опережающее объявление процедур и функций: procedure MyProc(I: Integer); forward;
- Циклы for, while, repeat-until
- Условные операторы if then else
- Инструкцию case
- Свойства в классах (property)
- Обработка исключительных ситуаций с помощью try-finally, try-except
- Явное приведение типов TComponent(Object).Owner или MyComponent := Object as TComponent
- Указатели (используются для передачи адресов методов и назначении их на обработчики событий)

## **Язык программирования Дизель-Паскаль**

---

Не смотря на то что как казалось бы, этот язык отличается от обычного Паскаля, на самом деле синтаксис языка Дизель-Паскаль базируется на Паскале и



технически имеет минимальные отличия, что позволило код Дизель-Паскаль и Паскаль компилировать и выполнять одной и то-же машиной-интерпретатором. Поэтому здесь будут описаны только различия языка Паскаль и Дизель-Паскаль, последний заимствовал некоторые элементы из Си подобных языков а так-же из языка Oberon.

В Дизель-Паскаль используется:

- Оператор присвоения = вместо :=
- Операторы ++ , -- , += , << , >> (побитовый сдвиг)
- Процедуры и функции объявляются со словом **method**, если **method** возвращает значение то это функция. Процедуры и функции класса давно называются методами, поэтому почему бы это слово не использовать?
- Инструкции **if**, **while**, **for** сами открывают блок, **begin** для открытия блока писать не надо, **но они всегда должны заканчиваться словом end**.

```
while i >= 0 do
    Оператор;
    Оператор;
    I--;
end;

for a = 1 to 10 do
    Оператор;
    Оператор;
end;

if a = 0 then
    Оператор;
    Оператор;
elseif a = 10 then
    Оператор;
    Оператор;
elseif a = 20 then
    Оператор;
    Оператор;
else
    Оператор;
    Оператор;
end;
```

**Операторы защищенного выполнения кода try finally, try except**

Синтаксис операторов немного отличается от Free Pascal или Delphi в обоих языках.

```
try
  Оператор;
  Оператор;
finally
  Оператор;
  Оператор;
end;

try
  Оператор;
  Оператор;
except on E: Exception do
  Оператор;
  Оператор;
end;
```

Ветвления в эксерт не поддерживаются, может быть только один раздел **on E: Exception do**

Возможные варианты использования:

```
try
  Оператор;
  Оператор;
except on E: Exception
  Оператор;
  Оператор;
end;
```

Здесь опущен **do** после **on E: Excetion**

Или так:

```
try
  Оператор;
  Оператор;
except
  Оператор;
  Оператор;
end;
```

Указатели являются важной составляющей всего функционала системы. Они отличаются от указателей в Delphi или Free Pascal и их главное предназначение участвовать в технологии позднего связывания, передавать адреса методов скрипта и назначать их на обработчики событий. Вот простой пример, его код приведен ниже, в нем из MainForm создается Form1 и в Form1 через конструктор с помощью указателя передается адрес скрипта, который назначается на обработчик события TForm1.Button1.Click. Всегда существует необходимость из документа открыть справочник и выбрать какую-то запись которая должна подставиться в документ. Пример можно скачать и выполнить в Дизель-Паскаль [Загрузить](#) (эту ссылку можно скопировать и в виде URL вставить в DManager или открыть в дизайнера Дизель-Паскаль).

-----

```

unit Main;

interface

type
  TMainForm = class(TForm)
    Button1: TButton;
    ListBox1: TListBox;
procedure Button1Click(Sender: TObject);
private
  protected
  public
procedure MyActExec(Sender: TObject);
end;

var MainForm: TMainForm;

implementation

uses Unit1;

procedure TMainForm.Button1Click(Sender: TObject);
var Form1: TForm1;
begin
  Form1 := TForm1.Create(Self, @MyActExec);
  Form1.Show;
end;

procedure TMainForm.MyActExec(Sender: TObject);
var Form1: TForm1;
begin
  //Получаем указатель на Form1
  Form1 := TForm1(TComponent(Sender).Owner);
  ListBox1.Items.Append(Form1.Edit1.Text);
  Form1.Close;
end;

end.

```

---

```

unit Unit1;

interface

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Label1: TLabel;
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
  private
  protected
  public
    constructor Create(AOwner: TComponent; P: Pointer);
  end;

implementation

procedure TForm1.FormClose(Sender: TObject; var CloseAction: TCloseAction);
begin
  CloseAction := caFree;
end;

constructor TForm1.Create(AOwner: TComponent; P: Pointer);
begin
  inherited Create(AOwner);
  Button1.OnClick := P;
end;

end.

```

---

## Видеоролик как быстро создать приложение для работы с базой FireBird SQL

<http://youtu.be/EES8aXxwAoI>

И ещё ролик про Дизель-Паскаль:

<http://youtu.be/KuSIs9bILO4>

## Ссылки

Исходный код и компоненты собственной разработки, которые нужны для сборки



Дизель-Паскаль, а т.ж. последнюю бинарную сборку под разные платформы можно загрузить здесь.