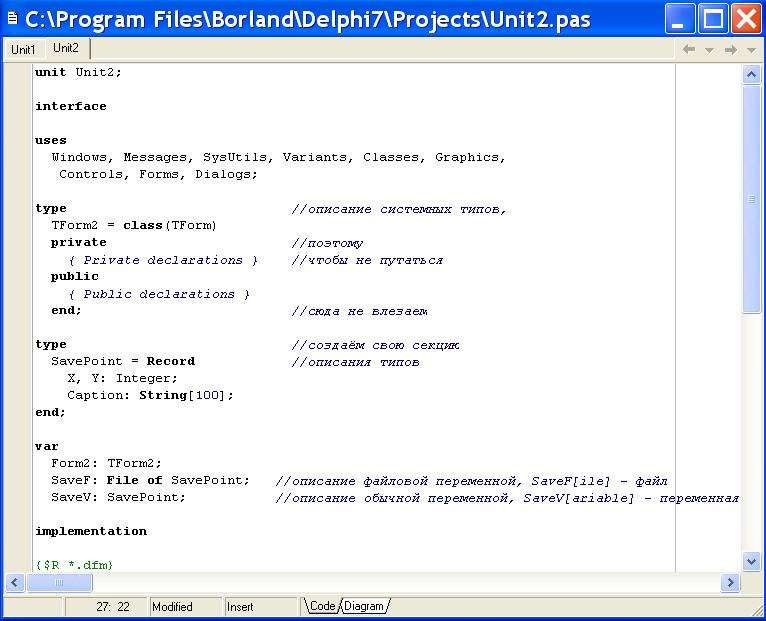
   Технология **работы с файлами** в системе **Delphi** требует определённого порядка действий:

1. Прежде всего **файл** должен быть **открыт**. Система следит, чтобы другие приложения не мешали **работе с файлом**. При этом определяется, в каком режиме открывается **файл** - для изменения или только считывания информации. После открытия **файла** в программу возвращается его идентификатор, который будет использоваться для указания на этот **файл**во всех процедурах обработки.
2. Начинается **работа с файлом**. Это могут быть запись, считывание, поиск и другие операции.
3. **Файл** закрывается. Теперь он опять доступен другим приложениям без ограничений. Закрытие**файла** гарантирует, что все внесённые изменения будут сохранены, так как для увеличения скорости работы изменения предварительно сохраняются в специальных буферах операционной системы.

В **Delphi** реализовано несколько способов **работы с файлами**. Познакомимся со классическим способом, связанным с использованием **файловых переменных**. Файловая переменная вводится для указания на файл. Делается это с помощью ключевого слова **File** :  
  
   var F**:**  **File** **;**  
  
Описанная таким образом файловая переменная считается ***нетипизированной***, и позволяет работать с файлами с неизвестной структурой. Данные считываются и записываются побайтно блоками, размер которых указывается при открытии файла, вплоть от 1 байт.  
  
Но чаще используются файлы, состоящие из последовательности одинаковых записей. Для описания такого файла к предыдущему описанию добавляется указание типа записи:  
  
   var F**:**  **File of** тип\_записи **;**  
  
В качестве типа могут использоваться базовае типы, или создаваться свои. Важно только, чтобы для типа был точно известен фиксированный размер в байтах, поэтому, например, тип **String** в чистом виде применяться не может, а только в виде **String[N]**, как указывалось в [уроке Delphi 5](http://www.delphi-manual.ru/lesson5.php).  
  
Данные, считанные из файла или записываемые в файл, содержатся в обычной переменной, которая должна быть того же типа, что и файловая. Поэтому сначала в программе лично я описываю нужный тип, а затем ввожу две переменные этого типа - файловую и обычную:



   Для текстовых файлов отдельно укажу, что тип файловой переменной в этом случае **TextFile**, а тип обычной - **String**.  
  
   Для открытия файла нужно указать, где он расположен. Для этого файловая переменная должна быть ассоциирована с нужным файлом, который определяется его адресом. Адрес файла может быть абсолютным, с указанием диска и каталогов ('C:\Мои документы\Мои рисунки\FileName.ini'), или относительным, тогда он создаётся в папке с **.exe** файлом программы. Для задания относительного адреса достаточно указать имя файла с нужным расширением. Делается это оператором **AssignFile :**   
  
   AssignFile(SaveF, 'C:\Мои документы\Мои рисунки\FileName.ini');  
   AssignFile(SaveF, 'FileName.ini');  
  
Теперь файл должен быть открыт.  
   Открытие файла оператором **Rewrite** приведёт воссозданию файла заново, т.е. существующий файл будет ***без предупреждения*** уничтожен, и на его месте будет создан новый пустой файл заданного типа, готовый к записи данных. Если же файла не было, то он будет создан.  
   Открытие файла оператором **Reset** откроет существующий файл к считыванию или записи данных, и его указатель будет установлен на начало файла **:**  
  
   Rewrite(SaveF);  
   Reset(SaveF);  
  
Каждый из этих операторов может иметь второй необязательный параметр, имеющий смысл для нетипизированных файлов, и указывающий длину записи нетипизированного файла в байтах:  
  
   Rewrite(SaveF, 1);  
   Reset(SaveF, 1);  
  
Чтение файла производится оператором **Read :**  
  
   Read(SaveF, SaveV);  
  
Запись в файл производится оператором **Write :**  
  
   Write(SaveF, SaveV);  
  
При этом чтение и запись производится с текущей позиции указателя, затем указатель устанавливается на следующую запись. Можно проверить, существует ли нужный файл, оператором **FileExists :**  
  
   **if** FileExists('FileName.ini')  
      **then** Read(SaveF, SaveV);  
  
Принудительно установить указатель на нужную запись можно оператором **Seek(SaveF, N)**, где N - номер нужной записи, который, как и почти всё в программировании, отсчитывается от нуля:  
  
   Seek(SaveF, 49); - установка указателя на 50-ю запись.  
  
При последовательном чтении из файла рано или поздно будет достигнут конец файла, и при дальнейшем чтении произойдёт ошибка. Проверить, не достигнут ли конец файла, можно оператором **EOF** (аббревиатура **E**nd **O**f **F**ile), который равен **true**, если прочитана последняя запись и указатель находится в конце файла:  
  
   **while** (not EOF(SaveF)) **do**  
      Read(SaveF, SaveV);  
  
   Для текстовых файлов вместо **Read** и **Write** используются операторы **Readln** и **Writeln**, умеющие определять конец строки. [В коментариях](http://www.delphi-manual.ru/les7next.php?com=yes#textfile) приведена процедура чтения текстового файла.  
  
Оператор **Truncate(SaveF)** позволяет отсечь (стереть или, если хотите, удалить!) все записи файла, начиная от текущей позиции указателя, и до конца файла.  
  
В конце работы с файлом его необходимо закрыть. Это делается оператором **CloseFile(SaveF) ;**  
  
   Теперь можно изменить программу из [первой части](http://www.delphi-manual.ru/lesson7.php) урока, запоминающую своё положение на экране. Описание переменных опускаю, оно приведено на рисунке выше.  
  
  Создаём обработчик события Формы **OnCreate** со следующим содержимым:  
  
**procedure** TForm1.FormCreate(Sender: TObject) ;  
**begin**  
   AssignFile(SaveF, 'Init.ini') ;  
   **if** FileExists('Init.ini') **then**  
      **begin**  
         Reset(SaveF) ;  
         Read(SaveF, SaveV) ;  
         Form1.Left := SaveV.X ;  
         Form1.Top := SaveV.Y ;  
         Form1.Caption:=SaveV.Caption ;      //Наши переменные дополнительно сохраняют заголовок Формы!  
      **end ;**  
**end ;**  
  
Теперь необходимо создать обработчик события **OnClose :**  
  
**procedure** TForm1.FormClose(Sender: TObject; var Action: TCloseAction) ;  
**begin**  
   Rewrite(SaveF) ;      //Нет необходимости проверять наличие файла, создадим его заново!  
   SaveV.X := Form1.Left ;  
   SaveV.Y := Form1.Top ;  
   SaveV.Caption := Form1.Caption ;  
   Write(SaveF, SaveV) ;  
   CloseFile(SaveF) ;  
**end ;**  
  
В данном случае файл считывается и записывается туда, куда мы ему указали. Но необходимо также уметь выбрать нужный файл в работающей программе. Описание этого - в [заключительной части](http://www.delphi-manual.ru/les7end.php) урока.

**Урок 15. Работа с файлами**

В этом уроке рассмотрим очень важную тему в программировании - работа с файлами.   
Практически в любой программе нужно копировать, открывать, изменять, сохранять, иногда удалять файлы.

Как известно, файлы имеют различные форматы и к ним применяются различные программы для обработки. Например, файлы рисунков BMP, текстовые файлы TXT и т.п. Файлы имеют различные имена и определенное расширение, имеющее в среде MS-DOS максимально три символа. Если у файла расширение RAR, то пользователь и операционная система знает, что это архиватор, причем windosw для обработки этого файла применяет средства, указанные в специальном разделе реестра.

**Копирование файлов**

Для копирования файлов применяется функция CopyFile. Формат ее такой:

CopyFile(Исходный\_файл,Записуемый\_файл,Флаг\_перезаписи);

где: Исходный\_файл – полный путь и название файла, который копируется;

Записуемый\_файл – полный путь и название файла, куда копируется исходный файл;

Флаг\_перезаписи – будет или нет перезаписан файл, если таковой уже существует (true - не будет, false - будет перезаписан).

CopyFile является функцией, которая возвращает флаг успешной или нет операции копирования.

Например, следующая строка:

if not CopyFile('c:\command.com','c:\1.com',true) then ShowMessage('Ошибка копирования');

скопирует файл command.com в файл 1.com только в том случае, если последнего нет, в противном случае будет выдаваться сообщение об ошибке копирования.

**Перемещение файлов**

Для перемещения файлов применяется функция MoveFile. Она имеет следующий формат:

MoveFile(Исходный\_файл,Записуемый\_файл);

Ее параметры аналогичны вышерассмотренной команде за исключением отсутствия флага перезаписи.

Пример:

if not MoveFile('c:\1.com','c:\2.com') then ShowMessage('Ошибка перемещения');

**Переименование файлов**

Функция RenameFile

if not RenameFile('c:\2.com','c:\3.com') then ShowMessage('Ошибка переименования');

У всех вышесказанных командах параметры исходного и конечного файла имеет тип PChar. Это строковый тип с завершающимся нулем. Возможно, вы раньше слышали о таком формате строковых данных. В такую переменную можно занести строку очень большой длины. Ее размер ограничивается наличием нулевого байта в конце. Теоретически такая переменная может иметь бесконечный размер. На практике она ограничивается размерами выделяемой памяти для программы (2Гб).

Преобразование обычной строковой переменной, типа String в PChar производится функцией:

PChar(Строковая\_переменная).

При этом, для команд файловых операций возможны объединение строк. Например:

procedure TForm1.Button1Click(Sender: TObject);

Var InDir,OutDir:String; // объявление строковых переменных

begin

InDir:='c:\1\'; // каталог исходных файлов

OutDir:='c:\2\'; // каталог записуемых файлов

CopyFile(PChar(InDir+'1.txt'),PChar(OutDir+'1.txt'),false);

CopyFile(PChar(InDir+'2.txt'),PChar(OutDir+'2.txt'),false);

end;

Здесь налицо операция соединения двух строковых переменных.

В итоге этой процедуры будет копирование файлов c:\1\1.txt и c:\1\2.txt в каталог c:\2. Файлы копируются под теми же именами.

**Удаление файлов**

Функция DeleteFile

if not DeleteFile('c:\3.com') then ShowMessage('Ошибка удаления');

**Работа с текстовыми файлами. Чтение из текстового файла**

На прошлых уроках мы как, в компоненте Memo процесс загрузки и записи текстового файла делался следующим образом:

Memo1.Lines.LoadFromFile(Имя\_файла); // загрузка

Memo1.Lines.SaveToFile(Имя\_файла); // сохранение

Все это благодаря свойству Lines, в котором хранятся строки.

Но на практике иногда необходимо прочитать только определенную строку или совершить операцию добавления строки в уже существующий файл.

Следующий пример обработки текстового файла очень похож на аналогичную на языке Pascal.   
Знающие люди могут ощутить разницу, поскольку есть некоторые отличия.

procedureTForm1.Button1Click(Sender: TObject);

Varf:TextFile; // объявление файловой переменной

st:**String**; // строковая переменная

begin

AssignFile(f,'c:\1.txt'); // привязка названия файла к файловой переменной

{$I-} // отключение контроля ошибок ввода-вывода

Reset(f); // открытие файла для чтения

{$I+} // включение контроля ошибок ввода-вывода

ifIOResult<>0 **then** // если есть ошибка открытия, то

**begin**

ShowMessage('Ошибка открытия файла C:\1.TXT');

Exit; // выход из процедуры при ошибке открытия файла

**end**;

While **not** EOF(f) **do** // пока не конец файла делать цикл:

**begin**

ReadLn(f,st); // читать из файла строку

ShowMessage(st); // выводить строку пользователю

**end**;

CloseFile(f); // закрыть файл

end;

Прокомментирую некоторые строки этого примера.

Команда AssignFile осуществляет привязку строки пути файла к файловой переменной. Все дальнейшие операции с файловой переменной автоматически осуществляются с указанным файлом. Для избежания путаниц, указывайте полный путь к файлу.

{$I-} и {$I+} являются директивами компилятору, что в этом месту соответственно следует отключить и включить контроль ошибок ввода-вывода. В данном случае при неудачной попытке открытия файла c:\1.txt (файл отсутствует или открыт для записи другой программой) наша программа не выдаст аварийной ошибки и продолжит выполнение данной процедуры. Это свойство полезно для обработки всех возможных случаев в работе программы.

IOResult – переменная, которая хранит в себе код ошибки последней операции ввода-вывода. Если она равна нулю, то последняя операция была успешно выполнена.

EOF(Файл) – функция, возвращающая признак конца файла. Т.е. она показывает, достигнут или нет конец открытого файла.

ReadLn(Файл,Переменная) – процедура считывания переменной из файла. В отличие от команды Read производит считывание строки с завершающимся символом перевода строки под кодами 13 и 10 (клавиша Enter).

CloseFile(Файл) – процедура закрытия ранее открытого файла.

Работа с текстовыми файлами. Запись в текстовый файл

Рассмотрим пример:

procedure TForm1.Button1Click(Sender: TObject);

Var f:TextFile; // указатель на текстовый файл

begin

AssignFile(f,'c:\1.txt'); // привязка названия к переменной

{$I-}

Append(f); // открыть файл для добавления

if IOResult<>0 **then //** если ошибка открытия (напр. файла нет)

**begin**

{$I-}

Rewrite(f); // создать новый файл

{$I+}

if IOResult<>0 then // ошибка создания файла

**begin**

ShowMessage('Ошибка создания файла C:\1.TXT');

Exit;

**end**;

**end**;

WriteLn(f,'Привет'); // запись в файл строки с символами перевода строки

CloseFile(f); // закрыть файл

end;

Процедура Append(Файл) открывает файл для записи и устанавливает указатель записи в конец файла, т.е. все добавляемые к файлу строки будут записаны в конец файла.

В нашем случае в самом начале файла 1.txt может не оказаться на диске, поэтому команда открытия файла для добавления вызовет ошибку. В этом случае срабатывает наш собственный контроль ошибок и выполняется команда создания файла.

Запись и чтение из файла не текстовых данных

Рассмотрим пример, в котором программа при выходе сохраняет свои размеры и положение окна на экране в файл и при последующем запуске восстанавливает состояние, которое было при выходе. Программа состоит из двух процедур OnShow и OnClose для окна Form1 программы.

**procedure** TForm1.FormShow(Sender: TObject);

**Var** f:**file** **of** Integer; // файловая переменная, тип integer

     i:Integer; // целочистенная переменная

**begin**

AssignFile(f,'pos.ini');

{$I-}

Reset(f);

{$I+}

**if** IOResult<>0 **then** Exit;

Read(f,i); //считать из файла значение

Form1.Top:=i; // верхнее положение окна

Read(f,i);

Form1.Left:=i; // левое положение окна

Read(f,i);

Form1.Width:=i; // ширина окна

Read(f,i);

Form1.Height:=i; // высота окна

CloseFile(f);

**end**;

**procedure** TForm1.FormClose(Sender: TObject; **var** Action: TCloseAction);

**Var** f:**file** **of** Integer;

**begin**

AssignFile(f,'pos.ini');

{$I-}

Rewrite(f);

{$I+}

**if** IOResult<>0 **then** Exit;

Write(f,Form1.Top);

Write(f,Form1.Left);

Write(f,Form1.Width);

Write(f,Form1.Height);

CloseFile(f);

**end**;

Как вы наверное заметили, имя файла, в котором храним переменные состояния окна, мы указали без пути, просто pos.ini.   
В этом случае файл создается и открывается в текущем каталоге. По умолчанию это каталог, из которого была запущена программа.

|  |  |
| --- | --- |
| **Типизированные файлы** | [**Previous**](http://delphi-faq.zoxt.net/b73.htm)  [**Home**](http://delphi-faq.zoxt.net/)  [**Next**](http://delphi-faq.zoxt.net/b75.htm) |

   
Теперь разберем типизированные файлы. Типизированный файл - это файл в котором записанны идентичные структуры. Например любой файл можно считать файлом байтов - т.е. можно его читать байт за байтом, можно перейти сразу к любому байту по его номеру, можно сразу узнать сколько байт в файле, можно заменить любой байт на другой не перезаписывая файл. Теперь все это в примерах:  
  
Объявляем файл байтов:  
  
**var** f:**file** **of** byte;  
     b:Byte;  
  
Ассоциируем файловую переменную с физическим файлом:  
  
AssignFile(f,'c:\myfile.bin');  
  
Теперь мы можем либо перезаписать/создать файл:  
  
Rewrite(f);  
  
Либо открыть существующий для чтения и записи:  
  
Reset(f);  
  
Обратите внимание, что функция Reset хотя и имеет такой же формат как и для текстовых файлов, но открытый ей файл можно и читать и писать, в отличие от текстовых.  
  
Теперь функции работы с файлом:  
  
**read**(f,b); - прочитать 1 байт   
**write**(f,b); - записать 1 байт  
seek(f,100); - поставить текущее положение считывания/записи на сотый байт  
Size(f); - прочитать количество байт в файле.   
Eof(f); - узнать не являетсмя ли байт последним  
  
Все эти функции не работают с файлами большими 2 Gb.  
  
После работы файл надо закрыть:   
  
CloseFile(f);  
  
Приведенные выше механизмы будут работать с любым файлом, так как любой файл можно считать файлом байтов. Теперь где это можно использовать? В принципе везде, но в подавляющем большинстве случаев это будет очень неудобно, ведь скорость считывания при чтении по байтам будет на порядки более низкой чем другими способами. Однако в некоторых случаях этот способ может быть очень полезен. Например в программе вам надо заменить 100й байт файла на другой, или прочитать 100й байт файла, например во всяких читерских программах, при взломе и т.п. Здесь такой доступ будет весьма удобен. Гораздо более интересным представляется дальнейшее развитие технологии типизированных файлов (их еще лет 15 назад называли "Файлы прямого доступа"). Представим себе, что файл состоит не из байт а из более сложных структур. Например мы имеем некоторую информацию в виде:  
  
**Type** MyRec=**Record**  
           **Name**:**string**[100];  
           Age:byte;  
           Membership:Boolean;  
           Accounts:**array**[1..10] **of** integer;  
        **End**;  
  
Обратите внимание, что все элементы записи точно типизированны, нет ни длинных строк, ни открытых массивов, ни объектов, другими словами, заранее точно известно сколько именно байт будет занимать переменная этого типа. Объявим переменную этого типа:  
  
**Var** MyVar:MyRec;  
  
и файл этого типа:  
  
**Var** f:**File** **of** MyRec;  
  
Теперь мы можем читать и писать сразу целую структуру, абсолютно так же как и если бы это был один байт:  
  
AssignFile(f,'c:\MyFile.rec');  
Rewrite(f);  
MyVar.**Name**:='Vitaly';  
MyVar.Age:=33;  
MyVar.Membership:=True;  
MyVar.Accounts[1]:=12345;  
MyVar.Accounts[2]:=34985;  
**Write**(f,MyVar);  
Closefile(f);  
  
Все остальные функции приведенные в предыдущей статье будут работать так же, только одно отличие - Seek и Size оперируют не с количеством байт, а с количеством записей.

|  |  |
| --- | --- |
| **Нетипизированные файлы** | [**Previous**](http://delphi-faq.zoxt.net/b74.htm)  [**Home**](http://delphi-faq.zoxt.net/)  [**Next**](http://delphi-faq.zoxt.net/b76.htm) |

   
Идем дальше. Есть такое понятие как нетипизированный файл. Это такой файл который содержит разнородные элементы. Например файл EXE - вначале он имеет заголовок, затем двоичный код, в конце какие-то ресурсы. Все части файла имеют разную длину и разную структуру. Тут уже обратится к произвольному элементу сложно, обычно надо вначале узнать где этот элемент находится, подчас это записано в предыдущем куске информации. Работа с такими файлами достаточно сложна и требует вручную разработки алгоритмов его чтения, но в связи гибкостью структуры и компактностью такие файлы составляют большинство. Для работы с нетипизированными файлами используют процедуры BlockRead и BlockWrite, которые позволяют читать/писать произвольное количество байт. Привожу пример пользования этими функциями из справки по Дельфи:  
  
**var**

|  |  |
| --- | --- |
|  | FromF, ToF: **file**; |

|  |  |
| --- | --- |
|  | NumRead, NumWritten: Integer; |

|  |  |
| --- | --- |
|  | Buf: **array**[1..2048] **of** Char; |

|  |  |
| --- | --- |
|  |  |

**begin**

|  |  |
| --- | --- |
|  | **if** OpenDialog1.Execute **then** *{ Display Open dialog box }* |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | **begin** |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | AssignFile(FromF, OpenDialog1.FileName); |

|  |  |
| --- | --- |
|  | Reset(FromF, 1); *{ Record size = 1 }* |

|  |  |
| --- | --- |
|  | **if** SaveDialog1.Execute **then** *{ Display Save dialog box}* |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | **begin** |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | AssignFile(ToF, SaveDialog1.FileName); *{ Open output file }* |

|  |  |
| --- | --- |
|  | Rewrite(ToF, 1); *{ Record size = 1 }* |

|  |  |
| --- | --- |
|  | Canvas.TextOut(10, 10, 'Copying ' + IntToStr(FileSize(FromF))+ ' bytes...'); |

|  |  |
| --- | --- |
|  | **repeat** |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | BlockRead(FromF, Buf, SizeOf(Buf), NumRead); |

|  |  |
| --- | --- |
|  | BlockWrite(ToF, Buf, NumRead, NumWritten); |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | **until** (NumRead = 0) **or** (NumWritten <> NumRead); |

|  |  |
| --- | --- |
|  | CloseFile(FromF); |

|  |  |
| --- | --- |
|  | CloseFile(ToF); |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | **end**; |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | **end**; |

|  |  |
| --- | --- |
|  |  |

**end**;  
  
Этот код копирует из одного файла в другой. Замечания по поводу этого метода работы с файлами - плюсы - очень высокая скорость, особенно если размер буффера увеличить до 64kb-512kb, что позволит считывать файл достаточно большими кусками, чтобы обеспечить отсутствие простоев винчестера, к тому же обеспечивается очень высокая гибкость в работе. Минусы - сложность разработки, необходимость вручную писать все детали механизма чтения/записи и интерпретации данных.  
  
Пожалуй на этом можно было бы и завершить описание работы с файлами средствами Паскаля и файловых переменных, но заглянув в Help Дельфей я обнаружил еще несколько функций достойных упоминания.  
  
Erase(f) - удаляет файл  
FilePos(f) - возвращает текущую позицию чтения/записи в файл  
Flush(f) - сбрасывает кэшированные файловые операции на диск  
Rename(f, 'MyNewFileName.txt') - переименование файлов  
Truncate(f) - файл обрезается до текущей позиции чтения/записи 