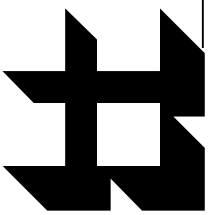

ГЛАВА 15

# Операции над файлами

Вероятно, вы сталкивались с необходимостью преобразовать лист книги Microsoft Excel в таблицу формата DBF или, наоборот, сохранить данные из таблицы или курсора в текстовом файле для последующей обработки информации другими приложениями. В этой главе рассматриваются средства, предоставляемые Visual FoxPro для импорта и экспорта данных. Кроме того, здесь вы познакомитесь с функциями низкоуровневого ввода-вывода, а также узнаете, как "устроены" таблицы формата DBF.

## Импорт данных

Под импортом данных подразумевается возможность преобразования информации, хранимой во внешних файлах, в таблицы Visual FoxPro. Внешними называются файлы, использующие для хранения табличных данных форматы, отличные от формата DBF, например, электронные таблицы Microsoft Excel или Lotus, а также текстовые файлы, информация в которых структурирована соответствующим образом.

Вы можете импортировать данные в интерактивном режиме при помощи диалогового окна **Import** или Мастера импорта данных (**Import Wizard**), а также программно, используя команды **Import From** или **Append From**.

### Диалог *Import*

Для вызова диалога **Import** в меню **File** выберите пункт **Import…**. Диалоговое окно показано на рис. 15.1.

Список **Type** диалога содержит перечень приложений, документы которых могут быть импортированы в формат DBF.

Поле **From** предназначено для ввода имени экспортируемого файла. Нажмите на расположенную правее этого поля кнопку, чтобы вызвать стандартный диалог **Open** для выбора файла.

Если вы импортируете электронную таблицу Microsoft Excel, то в списке **Sheet** появится список листов книги.

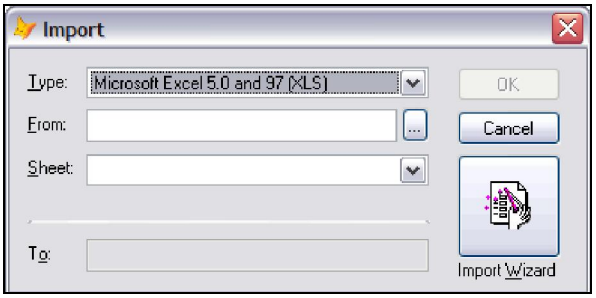


**Рис. 15.1.** Диалоговое окно **Input**

Заполните все поля и нажмите на кнопку **OK**. На основании полученных данных Visual FoxPro сгенерирует команду **Import From**, которая и выполнит импорт данных. Таблица будет создана в той же папке, в которой расположен исходный файл; ее имя будет совпадать с именем исходного файла, расширение — dbf.

## Мастер импорта данных

Мастер существенно расширяет возможности импорта данных. В частности, он поддерживает работу с текстовыми файлами, а также может добавлять данные в существующую таблицу. Большим достоинством Мастера является возможность определения типов данных и имен полей таблицы.

Для запуска Мастера вызовите диалоговое окно **Import** (рис. 15.1) и нажмите в его окне кнопку **Import Wizard**.

Мастер выполняет импорт данных за четыре шага. На первом шаге определяется тип импортируемых данных, источник и адресат (рис. 15.2).

В списке **File Type** выберите тип импортируемых данных. Поле **Source File** (файл с исходными данными) недоступно для ввода; воспользуйтесь расположенной правее этого поля кнопкой **Locate…**, чтобы вызвать стандартное диалоговое окно **Open** для выбора файла.

В качестве адресата вы можете указать как новую, так и существующую таблицу. Если вы выберете опцию **New table**, то Мастер создаст новую таблицу с указанным именем и загрузит в нее данные. В случае выбора опции **Existing table** Мастер добавит данные в существующую таблицу. И в том и в другом случае вы должны указать файл таблицы, используя соответствующие кнопки **Locate…** для вызова стандартных диалогов сохранения файла.

Если вы собираетесь импортировать данные в новую таблицу, то Мастер перейдет к шагу 1а, на котором предложит вам уточнить, будет ли создаваемая таблица свобод-

ной (**Create my table as a stand-alone free table**) или она будет добавлена в базу данных (**Add my table to the following database**).

Шаг 2 (рис. 15.3) предназначен для определения формата данных. Секция **Data Format** доступна только в том случае, если источником данных является текстовый

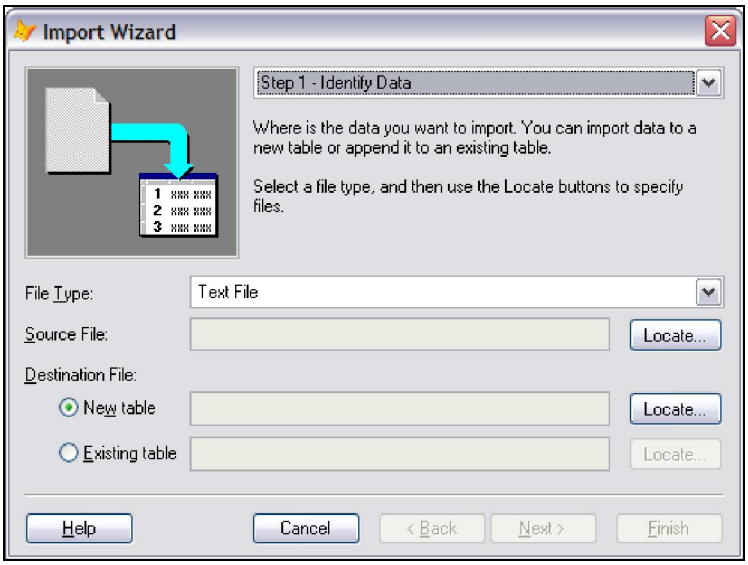


**Рис. 15.2.** Шаг 1 Мастера импорта

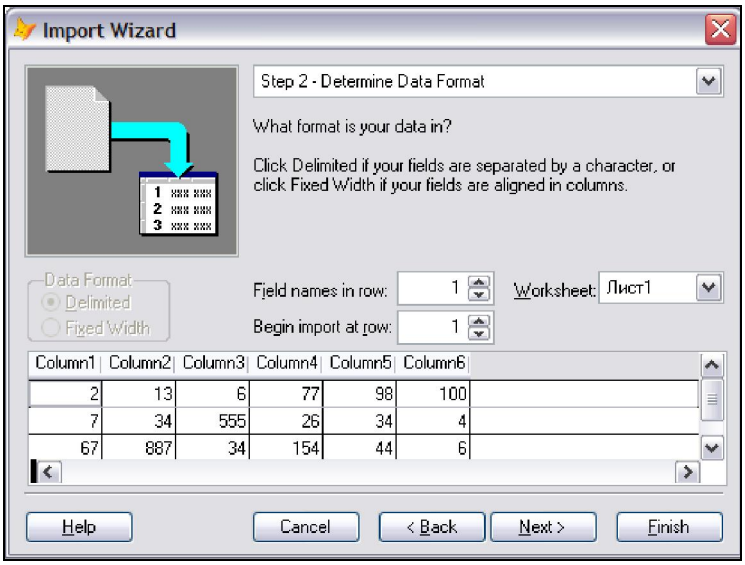


**Рис. 15.3.** Шаг 2 Мастера импорта данных

файл. В ней вы можете указать, каким образом отформатированы данные. Опция **Delimited** предполагает, что поля отделяются друг от друга специальными символами-разделителями, например, запятыми. Символ, используемый в качестве разделителя, должен быть указан в шаге 2а. Опция **Fixed Width** предполагает, что поля в файле выровнены по столбцам фиксированной длины.

В поле **Field names in row** вы можете указать номер строки (считается от начала импортируемого файла), содержащей имена полей. Если в импортируемом файле отсутствует информация об именах полей, то установите это поле в ноль.

В поле **Begin import at row** укажите номер строки, с которой начинаются импортируемые данные. Если источником данных является книга Microsoft Excel, то в списке **Work sheet** вы можете указать конкретный лист этой книги.

В нижней части окна Мастера выводится таблица, в которой отображаются импортируемые данные. Вид отображаемых данных зависит от их типа. Так, для текстового файла это будет нумерованный список строк в том виде, как они указаны в текстовом файле. Для документа Microsoft Excel или Lotus данные будут показаны в виде таблицы. Вы можете здесь отредактировать импортируемые данные.

Если в качестве источника данных выбран текстовый файл, то мастер перейдет к шагу 2а, на котором вы должны указать разделительный символ для полей данных (если в шаге 2 выбрана опция **Delimited**), либо указать размер каждого поля.

В шаге 3 (рис.15.4) определяются имена, типы и размеры полей формируемой таблицы.

Для установки параметров поля таблицы выделите столбец, щелкнув по нему мышью, после чего в поле **Name** введите наименование, а в полях **Type**, **Width** и **Decimals** укажите тип и размер данных.

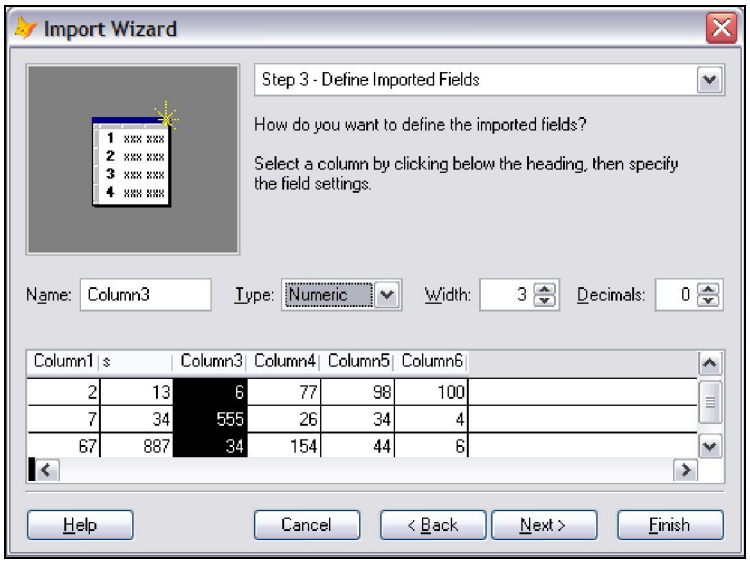


**Рис. 15.4.** Шаг 3 Мастера импорта данных

Шаг 3а предназначен для определения кодовой страницы, формата представления даты, символа-разделителя целой и дробной части числа и некоторых других параметров.

На последнем, четвертом шаге Мастера, нажмите на кнопку **Finish**; Мастер выполнит импорт данных и завершится.

## Команда *Import From*

При помощи этой команды вы можете создать таблицу и выполнять импорт данных непосредственно в процессе выполнения вашего приложения. Таблица будет создана в той же папке, в которой находится исходный файл; ее имя будет совпадать с именем исходного файла, расширение — dbf.

Ниже показан пример использования команды для создания таблицы на основе данных листа книги Accounts Microsoft Excel:

```
IMPORT FROM "d:\My Excel Books\Accounts.xls" XL8 SHEET "Лист1"
```

Ключевое слово `XL8 SHEET` указывает, что импортируемый документ является книгой Microsoft Excel 97 или более поздних версий. Если книга содержит всего один лист, слово `SHEET` можно опустить.

В процессе выполнения команды в папке d:\My Excel Books будет создана таблица Accounts.dbf.

При импорте данных из документа Lotus 1-2-3 вместо ключевого слова `XL8` используйте ключевые слова `WK1`, `WK3` или `WKS`. Значения ключей для импорта документов других форматов вы можете найти в справочной документации.

## Команда *Append From*

Эта команда обладает гораздо большей функциональностью по сравнению с командой **Import From**. Она может импортировать данные как из внешних файлов, так и из таблиц Visual FoxPro.

В следующем примере показано, как импортировать данные из текстового файла, в котором в качестве разделителей полей используются табуляторы:

```
APPEND FROM customer.txt DELIMITED WITH TAB
```

## Экспорт данных

Под экспортом понимается преобразование данных из таблиц или курсоров Visual FoxPro во внешние файлы, использующие для хранения табличных данных форматы, отличные от формата DBF. Вы можете экспортировать данные в интерактивном режиме при помощи диалогового окна **Export** или программно, используя команды **Export To** и **Copy To**.

## Диалоговое окно *Export*

Для вызова диалогового окна **Export** выберите в меню **File** пункт **Export…**. Диалоговое окно показано на рис. 15.5.

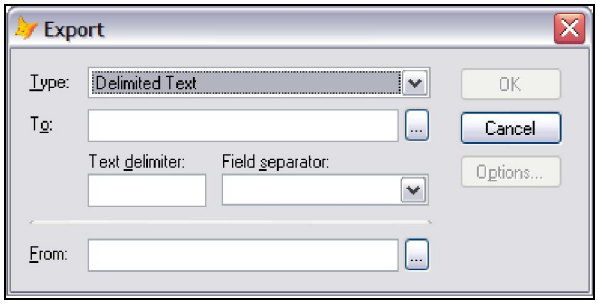


**Рис. 15.5.** Диалоговое окно **Export**

Выберите в поле **Type** диалога тип экспортируемого документа. В поле **To** введите имя и расширение файла, в который будут экспортированы данные. При необходимости воспользуйтесь стандартным диалогом **Save As**, который можно вызвать, нажав на расположенную правее поля кнопку.

Поле **Text delimiter** и список **Field separator** будут отображаться только в том случае, если выберете экспорт в текстовый файл с разделителями (Delimited Text). Введите в поле **Text delimiter** символ, который будет использоваться для ограничения символьных строк (обычно это кавычки). Вы можете выбрать из списка **Field separator** пробел или табулятор; вы так же можете ввести здесь любой другой символ, который будет использоваться как разделитель полей.

В поле **From** укажите имя таблицы, из которой будут экспортироваться данные. Кнопка, расположенная правее этого поля, позволяет воспользоваться для открытия таблицы стандартным диалогом **Open**.

Когда все поля диалогового окна **Export** будут заполнены, станет доступной кнопка **Options…**. При нажатии на эту кнопку будет выведено диалоговое окно, в котором вы можете настроить критерии выборки данных (рис. 15.6).


**Рис. 15.6.** Диалоговое окно **Export Options**

Нажимая на кнопки **Scope…**, **For…**, **While…** и **Fields…**, при помощи построителя выражений создайте список экспортируемых полей и условия выборки данных.

При нажатии на кнопку **OK** диалогового окна **Export** Visual FoxPro сгенерирует команду `Export`, которая и выполнит экспорт данных.

## Команда *Export To*

При помощи этой команды вы можете экспортировать данные в процессе выполнения вашего приложения. Команда экспортирует данные из таблицы или курсора, открытого в текущей рабочей области.

Вы можете указать в команде список экспортируемых полей и условия выборки, как это показано в следующем примере:

```
EXPORT TO "d:\My Excel Books\Accouts.xls" ;
   FIELDS acc_code, acc_name, debet, credit ;
   FOR debet > 0 XL5
```

Команда экспортирует в файл документа Microsoft Excel данные из полей `acc_code`, `acc_name`, `debet` и `credit` таблицы, которые перечислены в предложении `FIELDS`.

Предложение FOR определяет условие выборки, а ключевое слово XL5 указывает тип формируемого документа.

Значения ключей для экспорта документов других форматов вы можете найти в справочной документации.

***ЗАМЕЧАНИЕ***

При экспорте данных в книгу Microsoft Excel максимальное количество экспортируемых записей не может превышать 16 384.

## Команда *Copy To*

Команда может использоваться как для копирования таблиц Visual FoxPro, так и для экспорта данных. Она включает в себя все возможности команды Export To.

В следующем примере показано использование команды Copy To для экспорта данных в текстовый файл с разделителями, в котором для ограничения текстовых строк используется символ "|", а поля разделяются точкой с запятой:

```
COPY TO customer.txt ;
   FIELDS account_code, account_name, debet, credit ;
   FOR debet > 0 ;
   DELIMITED WITH '|' WITH CHARACTER ';'
```

При экспорте в таблицы формата DBF, использовавшиеся в FoxPro 2.x, вы должны учитывать, что некоторые типы данных полей Visual FoxPro не поддерживаются в предыдущих версиях языка. В табл. 15.1 показано, как преобразуются типы данных Visual FoxPro при экспорте в FoxPro 2.x.

***Таблица 15.1.*** *Преобразование типов данных при экспорте в формат FoxPro2.x*

| Тип поля в Visual FoxPro | Тип поля в FoxPro 2.x |
|---|---|
| Blob | Memo |
| Currency | Float |

***Таблица 15.1*** *(окончание)*

| Тип поля в Visual FoxPro | Тип поля в FoxPro 2.x |
|---|---|
| DateTime | Date |
| Double | Float |
| Integer | Numeric |
| VarChar | Memo |
| VarBinary | Memo |

В следующем примере данные из таблицы, открытой в рабочей области, копируются в таблицу account.dbf в формате, поддерживаемом в FoxPro 2.x и с кодовой страницей 866.

```
COPY TO "c:\Fox\account.dbf" FOX2X AS 866
```

Допускается использование символов "*" и "?" в качестве шаблонов в именах полей, перечисляемых в предложении `FIELD`. Например, в следующей команде будут экспортированы все поля, имена которых начинаются на букву "a":

```
COPY TO "c:\Fox\account.dbf" FIELDS LIKE a* FOX2X AS 866
```

# Копирование и удаление файлов

Копировать любые файлы в Visual FoxPro можно командой `Copy File`:

```
COPY FILE ExistingFileName TO NewFileName
```

Параметр *`ExistingFileName`* определяет имя существующего файла, а параметр *`NewFileName`* — имя нового файла, в который будут скопированы данные. Вы можете использовать символы "*" и "?" в качестве шаблонов в именах файлов. Например, в следующем примере создаются копии всех программных файлов текущей папки:

```
COPY FILE *.PRG TO *.BAK
```

Функция `FileToStr()` копирует содержимое файла в переменную памяти:

```
cString = FILETOSTR("c:\ My Folder \data.dat")
```

Функция `StrToFile()` копирует содержимое переменной памяти в файл. Функция может добавлять данные к существующему файлу.

В следующем примере создается файл data.txt, в который копируется содержимое поля таблицы Visual FoxPro:

```
nBytes = STRTOFILE(mytable.textfield, "c:\My Folder\data.txt")
```

Функция возвращает количество записанных в файл байтов. В случае ошибки функция возвращает ноль.

При вызове функции ей может быть передан третий параметр, который может иметь как логическое, так и численное значение. Допустимые значения этого параметра и определяемое им поведение функции показаны в табл. 15.2.

***Таблица 15.2.*** *Значения третьего параметра функции `STRTOFILE()`*

| Значение | Описание |
|---|---|
| `.F.` или 0 | Создает новый или перезаписывает существующий файл (используется по умолчанию) |
| `.T.` или 1 | Данные будут добавлены в существующий файл |

| | |
|---|---|
| 2 | Записывает в начало файла метку — признак использования формата Unicode (Byte Order Mark — BOM) байты 0xFF и 0xFE. Функция не выполняет конвертирование в формат Unicode; используйте функцию `STRCONV()` для преобразования данных в формат Unicode |
| 4 | Записывает в начало файла метку — признак использования формата UTF-8 (Byte Order Mark — BOM) байты 0xEF, 0xBB и 0xBF. Функция не выполняет конвертирование в формат UTF-8; используйте функцию `STRCONV()` для преобразования данных в формат UTF-8 |

Для удаления файла или группы файлов в Visual FoxPro применяются команды `Delete File` и `Erase`. Эти команды абсолютно идентичны.

Первый параметр, передаваемый команде, определяет имя удаляемого файла. Вы можете использовать символы "*" и "?" как шаблоны в именах файлов. Так, при выполнении следующей команды из текущей папки будут удалены все файлы с расширением bak:

```
DELETE FILE *.bak
```

Если вы в качестве первого параметра укажете вопросительный знак, то команда выведет диалоговое окно **Delete** для выбора удаляемого файла:

```
ERASE ?
```

Команды выполняют физическое удаление файла с диска. Если вы хотите поместить удаляемые файлы в Корзину Windows, то используйте в команде предложение `Recycle`:

```
DELETE FILE *.txt RECYCLE
```

Помещенные в Корзину файлы вы сможете восстановить; файлы, физически удаленные с диска, восстановить невозможно.

> ***Замечание***
> Файлы, указываемые в командах копирования или удаления, не должны быть открыты вашим или любым другим приложением. Функции `FileToStr()` и `StrToFile()` открывают файлы в режиме разделяемого доступа.

# Использование функции *ALINES()*

Эта функция замечательная тем, что с ее помощью можно преобразовать содержимое текстовой строки в массив строк, который в последующем достаточно просто экспортировать в курсор. Хотя функция `ALINES()` непосредственно не работает с файлами, тем не менее стоит рассмотреть ее возможности по импорту данных.

Предположим, что вам необходимо перенести содержимое текстового файла в таблицу формата DBF, причем каждая запись таблицы должна содержать текст, ограниченный символами возврата каретки и возврата строки. Первое, что вы должны сделать, — это прочитать текстовый файл и сформировать из него массив строк:

```
LOCAL lcString, lcCRLF, lnCount, laStringArray[1]
lcCRLF = CHR(13) + CHR(13)           && Возврат каретки + перевод строки
lcString = FileToStr("MyFiles.txt")  && Читаем файл
lnCount = ALINES(laStringArray, lcString, lcCRLF)
```

В результате выполнения этого кода строки из файла MyFile.txt будут помещены в массив `laStringArray`, причем каждый элемент массива будет содержать отдельную строку из файла. Третий параметр, передаваемый функции `ALINES()`, — разделитель строк, который может представлять собой как одиночный символ, так и некоторую строку, длина которой не должна превышать 23 байт.

Следующий шаг — создание таблицы и размещение в ней информации из массива:

```
CREATE TABLE mytable (String m)
INSERT INTO mytable (String) FROM ARRAY laStringArray
```

Так же просто при помощи функции `ALINES()` решается и более сложная задача, возникающая в случае, если каждая строка текстового файла содержит несколько значений (полей данных), разделенных одним или несколькими заранее определенными символами. В листинге 15.1 показан пример кода, в котором выполняется импорт данных из текстового файла, каждая строка которого содержит разделенные символами табуляции четыре поля, из которых первые два — обычные символьные строки, а вторые два — целочисленные значения. В результате выполнения показанного в листинге кода данные из текстового файла будут импортированы в курсор.

```
LOCAL lcString, lcCRLF, lnCount, laStringArray[1], laFields[1]
lcCRLF = CHR(13) + CHR(13)           && Возврат каретки + перевод строки
lcString = FileToStr("MyFiles.txt")  && Читаем файл
lnCount = ALINES(laStringArray, lcString, lcCRLF)
CREATE CURSOR mycursor (name c(100), title c(10), amount I, cost i)
FOR I = 1 TO lnCount
   = ALINES(laStringArray[1], laFields, CHR(13))
   laFields[3] = VAL(laFields[3])    && Преобразуем символьные значения
   laFields[4] = VAL(laFields[4])    && в числа
   INSERT INTO mycursor FROM ARRAY laFields
ENDFOR
```

Проанализируем код. Сначала выполняется считывание содержимого текстового файла в переменную `lcString`. Затем при помощи функции `ALINES()` заполняется массив `laStringArray`, каждый элемент которого содержит по одной строке из текстового файла, причем в качестве ограничителя (разделителя) строк используется последовательность символов `CR` + `LF`. Количество выбранных строк сохраняется в переменной `lnCount`.

Как и большинство встроенных функций Visual FoxPro, работающих с массивами, функция `ALINES()` может работать как с объявленным массивом (что и сделано в рассматриваемом листинге), либо, если массив не существует, то функция создаст его.

Соответственно, функция изменяет размер ранее объявленного массива, устанавливая его равным количеству выбранных из символьного выражения строк.

В цикле `FOR` в качестве первого параметра функции `ALINES()` передается очередной элемент массива `laStringArray`; теперь `ALINES()` "расщепляет" каждую считанную строку, рассматривая как разделитель символ табуляции (CHR(0)), помещая полученные поля в массив `laFields`. Так как по условию последние два поля — целочисленные, то выполняется преобразование строки в число. В команде `INSERT` выполняется запись данных из массива в курсор `mycursor`.

# Низкоуровневый доступ к файлам

Файл — это совокупность связанных между собой байтов, расположенных на диске или ином запоминающем устройстве, поддерживающем файловую структуру. Когда приложение считывает данные из файла, оно получает строку байтов, и именно на приложение ложится функция определения, какая именно информация получена из файла (символы, строки, числа или иные данные). В зависимости от организации информации внутри файла различают текстовые файлы с последовательным доступом и двоичные файлы.

Файлы с последовательным доступом используются для хранения текстовых строк произвольной длины. Каждая строка заканчивается парой символов возврата каретки и перевода строки. Особенностью таких файлов является то, что для получения доступа к любой очередной строке внутри этого файла необходимо прочитать все предшествующие ей строки.

В двоичных файлах разбиение на строки отсутствует, поэтому в них могут храниться любые данные. Если информация в двоичном файле представлена последовательностью блоков фиксированной (неизменной) длины, то для таких файлов может использоваться прямой доступ к любому произвольному блоку по его адресу (смещению) от начала файла.

Таблицы формата DBF являются двоичными файлами, в которых области, состоящей из блоков фиксированной длины (записей таблицы), предшествует так называемый заголовок — блок байтов, определяющий структуру таблицы. Размер заголовка изменяется в зависимости от количества полей таблицы.

Для определения позиции внутри файла, начиная с которой выполняется чтение или запись очередной порции данных, применяется *указатель*. Фактически указатель определяет величину смещения в байтах от начала файла. Так как в Visual FoxPro для хранения указателя используется 32-разрядная переменная, то его максимальное значение не может превышать 2 147 483 648; именно это значение и определяет ограничение на максимальный размер файлов. Значение указателя изменяется после выполнения каждой операции чтения или записи; если вы открываете файл при помощи одной из рассмотренных ниже функций низкоуровневого доступа, то у вас также появляется возможность непосредственного управления указателем.

## Открытие и закрытие файлов

Функция FCREATE() создает новый, а функция FOPEN() открывает существующий файл. Обе эти функции возвращают дескриптор файла. Дескриптор файла — это целочисленное положительное значение, идентифицирующее файл. Если функция возвращает отрицательное значение, то это означает, что файл не может быть создан (открыт). Используйте функцию FERROR() для определения причины возникшей ошибки.

Все остальные файловые функции низкоуровневого доступа обращаются к файлу только через его дескриптор.

При создании нового файла вы можете определить его атрибуты:

```
nHandle = FCREATE(MyFile)     && Файл доступен для записи и чтения
nHandle = FCREATE(MyFile, 2) && Скрытый файл
nHandle = FCREATE(MyFile, 4) && Системный файл
nHandle = FCREATE(MyFile, 6) && Скрытый системный файл
```

При открытии существующего файла вы можете установить для него вид доступа и определить использование буферизации:

```
nHandle = FOPEN(MyFile)      && Открыть только для чтения (с буферизацией)
nHandle = FOPEN(MyFile, 1)   && Только запись (с буферизацией)
nHandle = FOPEN(MyFile, 2)   && Чтение и запись (с буферизацией)
nHandle = FOPEN(MyFile,10)   && Только чтение (без буферизации)
nHandle = FOPEN(MyFile,11)   && Только запись (без буферизации)
nHandle = FOPEN(MyFile,12)   && Чтение и запись (без буферизации)
```

Буферизация обеспечивает более высокое быстродействие операций чтения/записи. Если вам необходимо сбросить данные из буфера на диск, воспользуйтесь функцией FFLUSH():

```
lResult = FFLUSH(nHandle)     && nHandle – дескриптор файла
```

Функция возвращает "истину" при успешном завершении. Не путайте эту функцию с командой Flush, которая работает с таблицами и индексами.

Функция FCLOSE() закрывает файл, открытый функциями FCREATE() или FOPEN():

```
lResult = FCLOSE(nHandle)     && nHandle – дескриптор файла
```

Эта функция также в случае успеха возвращает "истину".

## Работа с файлами последовательного доступа

Функция FPUTS() записывает в файл строку, добавляя к ее концу символы возврата каретки и перевода строки. Функция FGETS() возвращает строку, считанную из последовательного файла. При вызове этой функции вы можете указать максимальную длину строки; функция возвратит указанное число байтов, если раньше не встретятся символы возврата каретки и перевода строки. Функция FEOF() возвращает "истину",

если достигнут конец файла. Все эти функции в качестве первого параметра получают дескриптор файла.

В листинге 15.2 показан пример кода, в котором записи из текстового файла заносятся в поля таблицы Visual FoxPro.

```
nHandle = FOPEN(MyTextFile)
IF nHandle > 0
   CREATE TABLE MyTable (ID I AUTOINC, textfield c(255))
   DO WHILE !FEOF(nHandle)
      INSERT INTO MyTable (textfield) VALUES (FGETS(nHandle))
   ENDDO
   = FCLOSE(nHandle)
ENDIF
```

Если при вызове функции `FGETS()` вы не указываете второй параметр, определяющий количество считываемых байтов, то по умолчанию подразумевается, что длина считываемой строки не превышает 254 байта. Вы должны явно указывать второй параметр, если строки в файле имеют большую длину, как, например, в следующем примере:

```
cString = FGETS(nHandle, 4096)
```

Максимальная длина строки в текстовом файле последовательного доступа не может превышать 8192 байтов.

В листинге 15.3 показан пример кода, в котором данные из таблицы записываются в текстовый файл. Поле **textfield** таблицы должно иметь тип данных `Character`.

```
nHandle = FCREATE(MyTextFile)
IF nHandle > 0
   USE MyTable
   SCAN
      = FPUTS(nHandle, MyTable.textfield)
   ENDSCAN
   = FCLOSE(nHandle)
   USE IN MyTable
ENDIF
```

## Работа с двоичными файлами

Для чтения и записи блоков данных указанной длины в двоичный файл применяются функции `FREAD()` и `FWRITE()`. Для определения размера файла, а также для управления

указателем используется функция `FSEEK()`. В качестве первого параметра эти функции так же получают дескриптор файла.

В листинге 15.4 показан пример кода, в котором содержимое переменной помещается в двоичный файл. В переменной могут находиться данные любого типа — символьные, числовые, логические и т. д.

```
nHandle = FCREATE(MyFile)
IF nHandle > 0
   = FWRITE(nHandle, MyData, LEN(MyData))
   = FCLOSE(nHandle)
ENDIF
```

А в листинге 15.5 показан код, демонстрирующий использование функции `FWRITE()` для записи символьных строк в файл последовательного доступа. Поле **textfield** таблицы должно иметь тип данных `Character`.

```
CRLF = CHR(13) + CHR(10)      && Перевод каретки и возврат строки
nHandle = FCREATE(MyTextFile)
IF nHandle > 0
   USE MyTable
   SCAN
      cString = MyTable.textfield + CRLF
      = FWRITE(nHandle, cString, LEN(cString))
   ENDSCAN
   = FCLOSE(nHandle)
   USE IN MyTable
ENDIF
```

Записи из такого файла можно читать при помощи функции `FGETS()`. Как видите, разница между текстовыми файлами последовательного доступа и двоичными файлами достаточно условна.

Как уже говорилось, функция `FSEEK()` предназначена для управления позицией указателя внутри файла. Кроме этого, при помощи данной функции можно устанавливать указатель в начало и конец файла. Познакомимся подробнее с ее синтаксисом:

```
nBytes = FSEEK(nHandle, nBytesMoved [, nRelativePosition])
```

Параметр *nBytesMoved* определяет величину изменения значения указателя; он может принимать как положительные, так и отрицательные значения. Если необязательный параметр *nRelativePosition* опущен, то значение в *nBytesMoved* определяет смещение от начала файла (т. е. собственно значение указателя).

В табл. 15.3 приведены возможные значения параметра *nRelativePosition*.

***Таблица 15.3.*** *Значения параметра* `nRelativePosition` *функции* `FSEEK()`

| nRelativePosition | Описание |
|---|---|
| 0 | Указатель перемещается в начало файла (используется по умолчанию). Функция возвращает ноль |
| 1 | Указатель перемещается из текущей позиции на количество байтов, указанных в *nBytesMoved* |
| 2 | Указатель перемещается в конец файла. Функция возвращает количество байтов в файле |

Функция возвращает смещение в байтах от начала файла.

Следующий пример иллюстрирует применение функции `FSEEK()`.

```
nBytes = FSEEK(nHandle, 0, 0)   && Указатель в начало файла; nBytes = 0
nBytes = FSEEK(nHandle, 100)    && nBytes = 100
nBytes = FSEEK(nHandle, -10, 1) && nBytes = 90
nBytes = FSEEK(nHandle, 20, 2)  && nBytes = 110
```

В листинге 15.6 показан пример пользовательской функции `ReadFile()`, являющейся функциональным аналогом функции `FileToStr()`.

```
FUNCTION ReadFile(tcFileName)
LOCAL lnHandle, lnRecords
lnHandle = FOPEN(tcFileName, 10)
IF lnHandle > 0
   lnRecords = FSEEK(lnHandle, 0, 2)  && В lnRecords – длина файла
   = FSEEK(lnHandle, 0, 0)            && Указатель — в начало файла
   RETURN FREAD(lnHandle, lnRecords)
ENDIF
RETURN ""
```

В листинге 15.7 показан код, демонстрирующий возможности чтения данных по произвольному адресу.

```
lnHandle = FCREATE("Test.txt")
lcString = "Пример произвольного доступа к байтам внутри файла"
= FWRITE(lnHandle, lcString, LEN(lcString))
= FSEEK(lnHandle,20)     && Указатель – на 20-й байт
? FREAD(lnHandle,20)     && " доступа к байтам вн"
= FSEEK(lnHandle,-10,1)  && Указатель – на 30-й байт
```

```
? FREAD(lnHandle,20)      && " байтам внутри файла"
= FSEEK(lnHandle,0,0)     && Указатель – в начало файла
? FREAD(lnHandle,20)      && " Пример произвольного"
= FCLOSE(lnHandle)
```

Обратите внимание на то, что функция `FREAD()` также изменяет значение указателя.

## Обработка ошибок

Функция `FERROR()` возвращает номер последней ошибки функции низкоуровневого доступа к файлам. Список ошибок приведен в табл. 15.4.

***Таблица 15.4.*** *Значения, возвращаемые функцией `FERROR()`*

| Возвращаемое значение | Описание |
|---|---|
| 0 | Нет ошибок |
| 2 | Файл не найден |
| 4 | Открыто слишком много файлов |
| 5 | Нет прав на доступ к файлу |
| 6 | Задан неверный дескриптор файла |
| 8 | Недостаточно памяти |
| 25 | Ошибка позиционирования начала файла |
| 29 | Переполнение диска |
| 31 | Ошибка открытия файла |

Функция не имеет параметров.

## Изменение размера файла

Функция `FCHSIZE()` изменяет размер файла, открытого функциями `FCREATE()` или `FOPEN()`. Функции передается два параметра, первый из которых — дескриптор файла, а второй — новый размер файла в байтах. В случае неудачи функция возвращает –1.

Когда размер файла увеличивается, Visual FoxPro выделяет для него сектора на том же диске, где файл был открыт. Вы должны иметь в виду, что, поскольку `FCHSIZE()` не инициализирует новое пространство файла, в него могут попасть старые данные, хранившиеся на диске в добавляемых секторах.

# Проверка наличия файла и даты изменения

Следующие три функции позволяют определить наличие файла на файловом томе и дату его последней модификации.

Функция `FILE()` получает два параметра, первый из которых — имя файла, а второй (необязательный) — флаг, определяющий "доступность" системных и скрытых файлов. Возвращаемое функцией значение зависит от наличия второго параметра.

В следующем примере функция возвращает "истину", если файл test.txt существует; но если этот файл создан как системный или скрытый, функция вернет "ложь":

```
lResult = FILE("test.txt")
```

Следующий вызов функции вернет "истину", даже если файл — скрытый или системный:

```
lResult = FILE("test.txt", 1)
```

Функция `FDATE()` возвращает значение типа `Date` или `DateTime`, определяющее дату последней модификации файла. Функция `FTIME()` возвращает символьную строку, содержащую время последнего изменения файла.

Тип значения, возвращаемого функцией `FDATE()`, определяется вторым параметром. Если этот параметр опущен или равен нулю, возвращается значение типа `Data`. Если параметр равен 1, возвращается значение `DateTime`.

Формат представления даты и времени зависит от установок, сделанных в окне **Options** или командами настройки среды окружения.

Примеры применения функций:

```
SET DATE GERMAN
? FDATE("test.dbf")     && 02.04.2006
? FDATE("test.dbf", 1)  && 02.04.2006 16:04:32
? FTIME("test.dbf")     && 16:04:32
```

# Структура таблиц формата DBF

Для хранения данных в Visual FoxPro используются таблицы, поддерживающие формат DBF. Этот формат был предложен еще во времена разработки системы dBase (80-е годы прошлого века) и с тех пор все изменения касались только добавления новых типов данных для полей таблиц.

Visual FoxPro использует формат DBF не только для таблиц с данными (файлы с расширением dbf), но и для файлов, в которых хранится информация о компонентах проекта: формах (scx), метках (lbx), отчетах (frx), меню (mnx), библиотеках классов (vcx) и, собственно, о самом проекте (pjx). Вы можете использовать команду `USE` для открытия таких файлов; вы можете просматривать и корректировать данные в этих файлах при помощи тех же самых инструментов, которыми вы пользуетесь для работы с таблицами.

## Заголовок таблицы

В начале таблицы находится заголовок — блок байтов, в котором можно выделить постоянную и переменную компоненты. Постоянная компонента занимает первые 32 байта, ее структура приведена в табл. 15.5.

***Таблица 15.5.*** *Структура постоянной части заголовка таблицы*

| Смещение, в байтах | Размер, байт | Назначение |
|---|---|---|
| 0 | 1 | Определяет формат таблиц:<br>0x31 — Visual FoxPro<br>0x32 — Visual FoxPro, поддержка автоинкрементных полей<br>0x43 — Visual FoxPro, поддержка полей типов `Varchar`, `Varbinary` или `Blob`<br>Значения байта для других форматов вы можете найти в справочной документации |
| 1 | 3 | Дата последней модификации таблицы в формате YYMMDD |
| 4 | 4 | Количество записей в файле |
| 8 | 2 | Смещение в байтах до начала первой записи (размер заголовка) |
| 10 | 2 | Длина одной записи, включая байт признака удаления |
| 12 | 16 | Зарезервированы |
| 28 | 1 | Определяет использование индексов, memo-полей и принадлежность к базе данных. Значение байта определяется как сумма следующих значений:<br>0x01 — таблица использует одноименный структурный индекс;<br>0x02 — таблица имеет memo-поля<br>0x04 — таблица является контейнером базы данных |
| 29 | 1 | Признак кодовой страницы |
| 30 | 2 | Зарезервированы |

> ***Замечание***
>
> Для числовых данных значения байтов хранятся в обратной последовательности, т. е. первым следует самый младший байт, и завершает число самый старший байт. Например, если байты с 4-го по 7-й заголовка (число, определяющее количество записей в файле) имеют следующие шестнадцатеричные значения: 0x04, 0xD8, 0x07 и 0x00, то их нужно составить в обратном порядке: 0x0007D804. Полученное значение и есть количество записей в таблице.

За постоянной компонентой заголовка следует до 255 32-байтовых блоков, каждый такой блок описывает формат поля таблицы. Структура блока приведена в табл. 15.6.

***Таблица 15.6.*** *Структура блока описания поля таблицы*

| Смещение, в байтах | Размер, байт | Назначение |
|---|---|---|
| 0 | 10 | Имя поля. Если длина имени поля меньше 10 байтов, то оставшиеся байты заполняются нулями |

***Таблица 15.6*** *(окончание)*

| Смещение, в байтах | Размер, байт | Назначение |
|---|---|---|
| 11 | 1 | Определяет тип поля:<br>W — `Blob`; C — `Character`; Y — `Currency`; B — `Double`; D — `Date`; T — `DateTime`; F — `Float`; G — `General`; I — `Integer`; L — `Logical`; M — `Memo`; N — `Numeric`; Q — `VarBinary`; V — `VarChar` |
| 12 | 4 | Смещение поля в записи |
| 16 | 1 | Длина поля в байтах |
| 17 | 1 | Количество знаков в дробной части (для полей типа `Numeric` и `Float`) |
| 18 | 1 | Флаг поля:<br>0x01 — поле используется системой (невидимо для пользователя)<br>0x02 — поле может хранить значения `NULL`<br>0x04 — двоичное поле (для полей типа `Character Binary` и `Memo Binary`)<br>0x06 — поле может хранить двоичные данные и значения `NULL` |
| 19 | 4 | Последнее значение автоинкремента поля |
| 23 | 1 | Значение шага автоинкремента |
| 24 | 8 | Зарезервированы |

Байт, следующий за последним 32-байтовым блоком описания поля, является терминирующим; он определяет конец данных описания полей и всегда равен 0x0D.

Следующий блок заголовка размером 263 байта предназначен для хранения информации, связанной с базой данных. Если таблица не принадлежит никакой базе данных, то этот блок заполняется нулями. Если таблица принадлежит базе данных, то в начале блока записывается имя таблицы — контейнера базы данных и затем — специфическая информация, используемая для обратной связи с базой данных.

## Блоки записей таблицы

Первая запись расположена по адресу, указанному 16-разрядным целым числом (байты 9 и 10 заголовка) плюс 1. Каждая следующая запись начинается непосред-

ственно за предыдущей. За самой последней записью таблицы следует байт, имеющий значение 0x1A, который является признаком окончания файла таблицы.

Первый байт в каждой записи используется как признак удаления. Если запись не удалена, то байт содержит пробел; для удаленных записей байт содержит символ "*" (звездочка). Следующие за первым байтом поля записи располагаются вплотную один за другим в том формате, как они объявлены в заголовке, например, за двоичным полем типа `Double`, занимающим 8 байтов, может следовать поле типа `Character`, и т. д.

## Изменение кодовой страницы

Как известно, кодовая страница для русского языка в Windows имеет номер 1251; в то же время некоторые функции Windows API, в частности функции, управляющие раскладкой клавиатуры, для установки русского языка используют код 0x419 (десятичное значение — 1049). Так как в таблице формата DBF для определения кодовой страницы используется всего один байт, то это привело к необходимости применения своего набора кодов страниц, используемых в таблицах. Соответствие номеров некоторых кодовых страниц значениям кодирующего байта показано в табл. 15.7.

***Таблица 15.7.*** *Байт кодовой страницы для таблиц формата DBF*

| Номер кодовой страницы | Значение байта | Описание |
|---|---|---|
| 437 | 1 (0x01) | Кодировка MS-DOS, США |
| 866 | 101 (0x65) | Кодировка MS-DOS, Россия |
| 1251 | 201 (0xC9) | Кодировка Windows, Россия |
| 1252 | 3 (0x03) | Кодировка Windows, США |

Информацию о кодировке других кодовых страниц вы можете найти в справочной документации в разделе **Code Pages Supported by Visual FoxPro**.

Для того чтобы изменить кодовую страницу, достаточно записать в байт, определяющий кодовую страницу, новое (допустимое) значение. Специально для корректировки кодовых страниц в составе Visual FoxPro поставляется программа cpzero.prg, которую вы можете найти в папке tools. Но так как нас интересует возможность изменения кодовой страницы при помощи функций низкоуровневого доступа к файлам, то создадим пользовательскую функцию, выполняющую требуемую операцию (листинг 15.8).

```
FUNCTION ChCodePage(tcFile, tnCodePage)
*
```

```
* tcFile – имя файла таблицы, в которой нужно изменить кодовую страницу
* tnCodePage – номер кодовой страницы
*
LOCAL nHandle, lcByte
nHandle = FOPEN(tcFile,11)      && Открываем файл для записи
IF nHandle > 0
   DO CASE
      CASE tnCodePage = 437     && Кодовая страница 437
         lcByte = CHR(1)
      CASE tnCodePage = 866     && Кодовая страница 866
         lcByte = CHR(101)
      CASE tnCodePage = 1251    && Кодовая страница 1251
         lcByte = CHR(201)
      CASE tnCodePage = 1252    && Кодовая страница 1252
         lcByte = CHR(3)
   ENDCASE
   = FSEEK(nHandle, 29)         && Указатель – на 29-й байт
   = FWRITE(nHandle, lcByte, 1) && Записываем новое значение
   = FCLOSE(nHandle)            && Закрываем файл
ENDIF
```

# Редактор файлов HexEdit

В поставку Visual FoxPro входит специальное приложение — **HexEdit**, которое позволяет просматривать и редактировать любые файлы; особенностью приложения является то, что данные в нем выводятся как последовательность байтов в шестнадцатеричном представлении. Вызвать **HexEdit** можно следующей командой:

```
DO HOME(1) + 'tools\hexedit\hexedit.app'
```

Окно **HexEdit** показано на рис. 15.7.

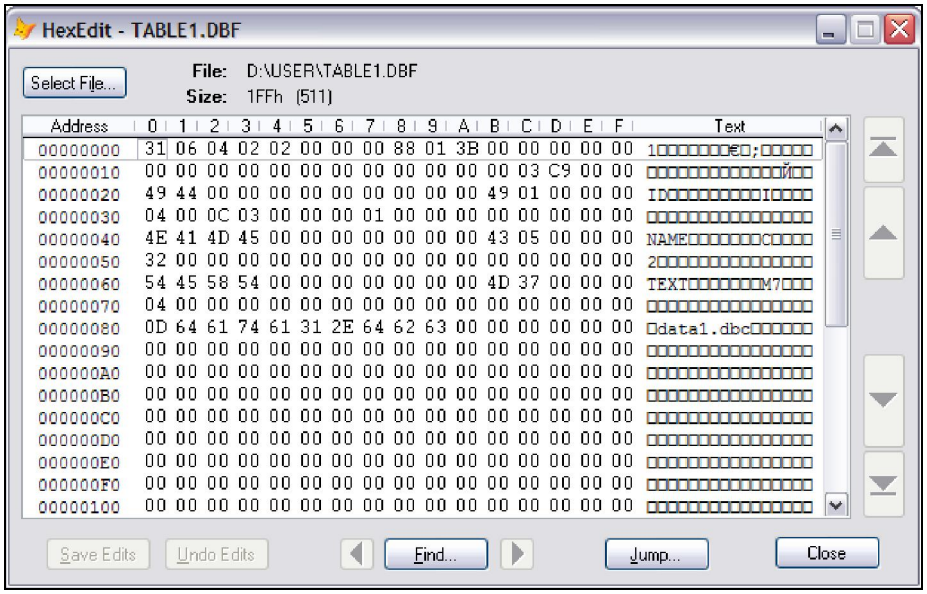


**Рис. 15.7.** Окно редактора **HexEdit**

Управлять этим редактором чрезвычайно просто; его описание вы можете найти в справочной документации.

Использование **HexEdit** может помочь вам "вылечить" таблицы, при записи данных в которые произошла ошибка, вызванная, например, сбоем устройства или пропаданием питания. Обычно в такой ситуации нарушается соответствие между значением количества записей, указанным в байтах с 4-го по 7-й заголовка таблицы, и фактическим количеством записей. Зная структуру таблицы, вам не составит труда проверить соответствие параметров, определенных в заголовке таблицы, ее фактическому состоянию, и на основании этого внести в файл необходимые исправления.

# Специальные таблицы Visual FoxPro

Как вы уже знаете, Visual FoxPro использует для хранения служебной информации, предназначенной для описания компонентов приложения, таблицы формата DBF. Так, для хранения информации о любой форме используются файлы с расширением scx (собственно таблица) и sct (memo-поля). Сведения о собственно проекте сохраняются в файлах с расширением pjx и pjt, библиотеки классов хранятся в файлах с расширениями vcx и vct, отчеты — в файлах с расширением frx и frt, и т. д. Описание полей этих таблиц вы можете найти в папке \Tools\Filespec, расположенной в корневой папке Visual FoxPro. В этой папке приведены описания таблиц форматов FoxPro 2.6, Visual FoxPro 6.0 и Visual FoxPro 9.0. В девятой версии некоторые изменения претерпели только файлы для хранения информации об отчетах; структура остальных

служебных таблиц Visual FoxPro девятой версии аналогична таблицам из шестой версии.

# Заключение

В этой главе вы познакомились с особенностями работы с внешними документами, т. е. документами, содержащими данные в форматах, отличных от формата, используемого в Visual FoxPro. Вы также узнали, что из Visual FoxPro можно работать с любыми файлами на низком уровне, вплоть до побайтовой записи/чтения.

Несмотря на отсутствие в этой главе подробного описания, мы надеемся, что вы при необходимости изучите и будете использовать поставляемый с Visual FoxPro редактор **HexEdit**, который позволит вам успешно "лечить" не только запорченные таблицы, но и специальные служебные таблицы Visual FoxPro.

Конечно, помимо рассмотренных, существуют и другие механизмы импорта и экспорта данных. Один из таких механизмов, позволяющий экспортировать данные (и не только!) в другие приложения, например, такие как Microsoft Excel, получивший название *автоматизации*, рассматривается в одной из следующих глав этой книги.