Обзор драйверов базы данных

Независимость данных

Независимость базы данных в Clarion-е достигается при помощи технологии встроенного драйвера, которая позволяет осуществлять доступ к данным практически любой файловой системы, используя один и тот же набор команд языка Clarion. В системе Clarion имеется достаточное количество драйверов и к ней добавляются новые драйверы.

Совет: **Перед использованием драйвера базы данных он должен быть зарегистрирован. Драйверы, входящие в комплект поставки, уже зарегистрированы. Для получения дополнительной информации о регистрации добавляемый драйверов базы данных см. Руководство Пользователя – Конфигурирование.**

Команды языка Clarion для доступа к данным из различных систем одинаковы; просто выберите правильный драйвер файла из выпадающего списка в вашем словаре данных и больше не беспокойтесь об этом. Драйвер файла переводит команды Clarion во внутренние команды выбранной файловой системы. Все драйверы файлов читают и пишут данные в формате файловой системы используемого драйвера без применения временных файлов или процедур импорта/экспорта.

Выбор правильного драйвера

Выбор файловой системы является важным решением, и мы рекомендуем вам собрать как можно больше информации из как можно большего числа источников для того, чтобы поддержать свое решение. Тем не менее, несмотря на всю важность выбора файловой системы, Clarion не делает этот процесс необратимым. Если выбранная вами файловая система вас почему-либо не устраивает, вы можете заменить ее. Например, некоторые разработчики используют драйвер TopSpeed для стадии разработки проекта как инструментальное средство, а затем заменяют его драйвером SQL, откладывая тем самым издержки на SQL-матобеспечение и SQL-сервер на более позднюю стадию проекта.

Общие характеристики драйверов

Импорт определения файла

Для существующих данных вы как правило имеете возможность импортировать определение файла в ваш Clarion - словарь данных. Мы настоятельно рекомендуем вам импортировать определение файла всегда, когда это только возможно, так как это не только уменьшает затраты вашего времени и усилий, но и гарантирует отсутствие ошибок в описании файлов.

Ключи, индексы и производительность

Несмотря на то что вы можете определять индексы в вашем словаре данных, которые не существуют внутри родной файловой системы, мы не рекомендуем делать этого, так как от этого будет страдать производительность вашего приложения. Вместо этого мы рекомендуем определять требуемые ключи или индексы собственными средствами выбранной вами файловой системы и затем импортировать определение файла вместе с ключами и индексами в ваш словарь данных.

Сортирующие последовательности

По умолчанию все драйверы баз данных TopSpeed упорядочивают данные с использованием сортирующих последовательностей. Добавление OEM атрибута - основание для использования ASCII сортирующей последовательности драйвером.

Отладка и трассировка файловых операций

Все драйверы баз данных TopSpeed могут создавать файл-протокол, содержащий команды ввода-вывода Clarion, соответствующие им команды работающей файловой системы и их коды завершения.

Вы можете генерировать либо общий протокол системы, либо заказной (условное протоколирование, основанное на логике вашей программы).

Общий протокол

Для создания общего протокола вы должны добавить в ваш файл WIN.INI следующие строки:

[CW*driver*]
Profile=[1|0]
Details=[1|0]
TraceFile=[Pathname],

где *driver* это имя драйвера базы данных (например [CWTopSpeed]). И имя

секции INI-файла [*CWdriver*] и имя точки входа внутри секции INI-файла не зависят от регистра.

Profile=1 сообщает драйверу о необходимости включать в файл протокола команды ввода-вывода Clarion, Profile=0 сообщает драйверу о том, что команды ввода-вывода Clarion не надо подключать в файл протокола.

Details=1 сообщает драйверу о необходимости включать в файл протокола содержимое буфера записи, однако, если файл зашифрован, для того, чтобы буфер записи появился в протоколе, вы должны добавить в INI-файл еще переключатель ALLOWDETAILS=1 (см. ALLOWDETAILS). Если Details=0 -содержимое буфера записи в протокол не включается. Для того, чтобы переключатель Details работал, необходимо, чтобы Profile=1.

TraceFile - имя файла, в который записывается протокол. Если TraceFile отсутствует - протокол записывается в текущий каталог, в файл с именем *driver*.log. *Pathname* - полный путь к файлу протокола. Если путь не указан, запись будет происходить в текущий каталог. Запись в протокол открывает файл протокола для исключительного использования. Если файл протокола к началу протоколирования уже существует, новые записи протокола дописываются в него.

Условное протоколирование

Для условного протоколирования вы можете использовать синтаксис свойств непосредственно из вашей программы, управляя процессом включения и выключения процесса записи в протокол. Протоколирование эффективно для отдельных файлов и для структур VIEW, в которых протоколируемый файл является первичным файлом.

```
file{PROP:Profile}=Pathname          !Включение протокола операций ввода/
                                     !вывода Clarion
file{PROP:Profile}=''                !Выключение протокола операций ввода/
                                     !вывода Clarion
PathName = file{PROP:Profile}        !Запрос имени файла протокола
file{PROP:Log}=string                !Запись строки в файл протокола
file{PROP:Details}=1                 !Включение необходимости
                                     !протоколирования буфера записи
fFile{PROP:Details}=0                !Выключение необходимости !
                                     протоколирования буфера записи
```

Pathname - Полный путь к файлу протокола. Если вы не указали путь, драйвер запишет файл протокола в текущем каталоге.

Вы можете также выполнить условное протоколирование при помощи команды SEND() и строки драйвера LOGFILE. Подробно См. LOGFILE

Проверка ошибок на уровне языка

Когда код ошибки, возвращаемый функцией ERRORCODE() равен 90, вы можете использовать функцию FILEERRORCODE() для захвата сообщения и кода возврата сервера или файловой системы. См. Описание языка для получения дополнительной информации об этой функции.

Сообщения об ошибках

Все TopSpeed-драйверы базы данных посылают сообщения об ошибках, доступ к которым осуществляется при помощи функций ERRORCODE(), ERROR(), FILEERRORCODE() и FILEERROR() (см *Описание языка (Language Reference)*). Драйвер посылает код завершения операции ввода/вывода немедленно после ее выполнения (OPEN, NEXT, GET, ADD, DELETE, и т.д.).

Совет: Функция ERRORFILE() возвращает имя файла, обработка которого вызвала посылку сообщения об ошибке.

Значения кодов ошибки, возвращаемые функцией ERRORCODE() и соответствующие этим кодам тексты сообщений, выдаваемые функцией ERROR(), приведены в Описании Языка. Вид текстов сообщений, возвращаемый функцией ERROR(), может быть отредактирован при помощи процедуры LOCALE и файла (CLAMSG). Для получения дополнительной информации см. Интернационализация (*Internationalization), CLAMSG,* и *LOCALE* в Описании Языка *(Language Reference).*

Кроме того, для драйверов Btrieve, xBase (Clipper, dBaseIII, dBaseIV, FoxPro), и ODBC может быть отконфигурирован текст сообщений,выдаваемый функцией FILEERROR(). См раздел Конфигурируемые сообщения об ошибках (*Configurable Error Messages*) для каждого драйвера, где приводятся списки значений кодов ошибок, возвращаемых функцией FILEERRORCODE() и соответствующие им тексты сообщений, возвращаемые функцией FILEERROR().

Кэширование диска и целостность данных

Для многофайловых систем кэширование диска может вступать в конфликт с обеспечением целостности данных. Под кешированием диска мы подразумеваем любые утилиты (например SMARTDRV), которые сообщают драйверу базы данных, что запись на диск состоялась, в то время как на самом деле этого еще не произошло.

Для улучшения производительности механизм кэширования обычно накапливает несколько записей в оперативной памяти и затем производит их запись на диск одной порцией. Однако в то время как кэширование увеличивает производительность системы, оно же может привести к разрушению файлов, если по каким-либо причинам система аварийно прекратит работу (аварийное отключение питания например) до того, как данные из кэша будут переписаны на диск. Надежная система питания может решительным образом уменьшить этот риск. Поэтому мы настоятельно рекомендуем не кэшировать диск. Если вы все таки это делаете, вы должны быть уверены, что используете надежный источник бесперебойного питания.

Общие строки драйверов

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. Различные, специфичные для каждого драйвера строки описаны в секции *Driver Strings (*в описании конкретных драйверов). Строки драйверов направляются тремя способами: оператором OPEN или оператором CREATE, процедурой SEND, и при помощи синтаксиса свойств.

DRIVER('Driver', '/DriverString = value')

Операторы OPEN(file) и CREATE(file) посылают драйверу любые строки драйвера, указанные в атрибуте DRIVER оператора описания файла. Оператор OPEN посылает строку непосредственно *перед* открытием файла. Вы можете специфицировать эти строки либо вручную, при объявлении файла (см. *DRIVER в Описании языка*), либо в словаре данных (поле Опции драйвера **(Driver Options)** в диалоге Свойства файла **(File Properties)**—см Редактор словаря-Свойства файла в Руководстве пользователя). В любом случае, строке драйвера должен предшествовать символ (/).

Например:

```
MyFile      FILE,DRIVER('TopSpeed','/LOGFILE=MyFile.Log')
            CODE
            OPEN(MyFile)                !Посылает драйверу имя файла протокола
```

[MyVar =] SEND(file, 'DriverString [= value]')

Функция SEND посылает или запрашивает строку драйвера в любое время выполнения программы и, в том числе, до момента открытия файла. SEND может одновременно установить новое значение строки драйвера и вернуть старое значение строки или может только установить новое значение, или только вернуть текущее. При использовании SEND, ISAM драйвер не требует лидирующего слэша в строке драйвера, а SQL драйвер требует его.

Например:

```
SEND(MyFile,'LOGFILE='&MyLogFile)                  !установка файла протокола
                                                   !для ISAM
MyLogFile=SEND(MySQLFile,'/LOGFILE')               !получение имени файла
                                                   !протокола для SQL
OldLogFile=SEND(MyFile,'LOGFILE='&NewLogFile)      !установка нового имени
                                                   !файла протокола и
                                                   !одновременное получение
                                                   !старого имени
```

file{PROP:DriverString}

Синтаксис свойств является альтернативой функции SEND. Используя синтаксис свойств, вы можете посылать драйверу строки драйвера в любое время, но только после того, как файл уже открыт. В рамках синтаксиса свойств строки драйвера не требуют лидирующего слэша.

Например:

```
MyLogFile = 'MyFile.Log'
MyFile{PROP:Profile}=MyLogFile          !Установка файла протокола
MyLogFile = MyFile{PROP:Profile}        !Получение имени файла протокола
```

Все драйверы TopSpeed поддерживают следующие строки драйвера:
ALLOWDETAILS

DRIVER('Driver', '/ALLOWDETAILS = TRUE | FALSE')

Строка драйвера ALLOWDETAILS разрешает драйверу включать содержимое буфера записи в файл-протокол для шифрованных файлов.

Строка драйвера ALLOWDETAILS работает с переключателем Details, описанным в разделе Отладка и трассировка файловых операций.

Общие свойства драйверов

Вы можете установить или затребовать свойства драйвера для управления его поведением или для получения информации о драйвере. Специфические свойства драйверов описаны в разделе «Свойства драйвера» для каждого из драйверов.

Вы получаете доступ к свойствам драйвера при помощи синтаксиса свойств.

Например:

```
MyFile{PROP:Profile}=MyLogFile          !установить файл протокола
MyLogFile = MyFile{PROP:Profile}        !Получить имя файла протокола
```

См также *VIEW и FILE Properties* в описании языка. Все TopSpeed - драйверы базы данных поддерживают следующие свойства драйверов (Driver Properties).

PROP:SQLDriver

PROP:SQLDriver возвращает значение, указывающее, поддерживает ли драйвер SQL. Значение единица (1 or True) указывает на то что драйвер поддерживает SQL; и значение ноль (0 or False) указывает на то что драйвер не поддерживает SQL.

Например:

```
SQLDriver# = Customer{PROP:SQLDriver}       !Запрос о возможности работать с
                                            !SQL
IF SQLDriver#
        SQLString»=Customer{PROP:SQL}       !Получить последний SQL оператор
                                            !END
```

Драйвер ASCII

Спецификации

Драйвер ASCII читает и пишет стандартные ASCII файлы без разграничителей полей. Это часто используется для импорта / экспорта данных с больших машин через ASCII двумерный файл. По умолчанию записи разграничиваются символом возврата каретки/перевода строки. Драйвер ASCII не поддерживает ключи.

Файл:	
C4ASCL.LIB	Статически вызываемая библиотека Windows (16-битовая)
C4ASCXL.LIB	Статически вызываемая библиотека Windows (32- битовая)
C4ASC.LIB	Библиотека экспорта Windows (16-битовая)
C4ASCX.LIB	Библиотека экспорта Windows (32-битовая)
C4ASC.DLL	Динамически вызываемая библиотека Windows (16-битовая)
C4ASCX.DLL	Динамически вызываемая библиотека Windows (32-битовая)

Совет: Из-за отсутствия реляционных характеристик и ненадежности (любой может посмотреть и изменить файл ASCII с помощью Notepad), вы вряд ли будете использовать драйвер ASCII для хранения больших файлов данных. Но он может помочь вам при управлении полем списка или при создании программы для просмотра текстовых файлов - используйте его, чтобы открыть файл и читать его в многострочном редакторе.

Поддерживаемые типы данных

STRING
GROUP

Характеристики файла/максимумы

Размер файла:	4,294,967,295 байт
Записей на файл:	4,294,967,295 байт
Размер записи:	65,520 байт
Размер поля:	65,520 байт
Полей на запись:	65,520
Ключей/Индексов на файл:	Нет
Размер ключа:	Нет

Полей мемо на файл:	Нет
Размер поля мемо:	Нет
Открытие файла данных:	Зависит от операционной системы

Строки драйвера

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. См. Общие характеристики драйверов - Строки драйвера.

Замечание: Некоторые строки символов драйвера не действуют после открытия файла, поэтому синтаксис функции SEND для модификации установки не перечислен. Однако синтаксис функции SEND для возврата величины переключения приводится в перечне для всех строк драйвера.

ASCII Драйвер поддерживает следующие строки драйвера:

CLIP

```
DRIVER('ASCII', '/CLIP = on | off' )
[ Clip» = ] SEND(file, 'CLIP [ = on | off ]' )
```

Драйвер автоматически удаляет пробелы в конце записи перед записыванием ее в файл. И наоборот, драйвер автоматически добавляет отсеченные пробелы в конец записи при чтении данных. Для отключения этой возможности установите CLIP = OFF. По умолчанию CLIP = ON. Функция SEND возвращает установленное значение CLIP(ON или OFF) как STRING(3).

CTRLZISEOF

```
DRIVER('ASCII', '/CTRLZISEOF = on | off' )
[ EOF» = ] SEND(file, 'CTRLZISEOF [ = on | off ]' )
```

По умолчанию (CTRLZISEOF=on) драйвер файла подразумевает, что любой символ Ctrl-Z внутри файла указывает на конец файла. Для отключения этого свойства установите CTRLZISEOF=off. Функция SEND возвращает установленное значение CTRLZISEOF как STRING(3).

ENDOFRECORD

```
DRIVER('ASCII', '/ENDOFRECORD = n [,m ]' )
[ EOR» = ] SEND(file, 'ENDOFRECORD [ = n [,m ]]' )
```

Определяет символ-ограничитель записей. *n* представляет число символов, которые образуют символ-ограничитель записи; *m* представляет ASCII код(ы) для символов конца записи, отделенные запятыми. По умолчанию 2,13,10, показывая тем самым, что 2 символа отмечают конец записи, возврат каретки (13) и перевод строки (10). Функция SEND возвращает символ-ограничитель записи.

Совет: Мэйнфреймы часто используют для разграничения записей только символ возврата каретки. Вы можете использовать ENDOFRECORD для чтения этих файлов.

FILEBUFFERS

```
DRIVER('ASCII', '/FILEBUFFERS = n' )
[ Buffers» = ] SEND(file, 'FILEBUFFERS [ = n ]' )
```

Устанавливает размер буфера, используемого для чтения и записи, размер буфера равен n*512. Используйте строку драйвера /FILEBUFFERS для увеличения размера буфера, если доступ к данным слишком медленный. Максимальный размер буфера равен 65,504 для 16-битовых приложений и 4,294,967,264 для 32-битовых. Функция SEND возвращает размер буфера в байтах.

Совет: По умолчанию размер буфера для файлов, открытых с запретом на запись для других пользователей больше 1024 или (2*размер записи) и больше 512 (1* размер записи) для всех других режимов открытия.

TAB

```
DRIVER('ASCII', '/TAB = n' )
[ Spaces» = ] SEND(file, 'TAB [ = n ]' )
```

Устанавливает или запрашивает TAB/SPACE расширение. Драйвер ASCII заменяет символы табуляции (ASCII символ 9) при чтении пробелами. Величина указывает число пробелов, которыми необходимо заменить символ табуляции в соответствии с указаниями, даваемыми ниже. Величина по умолчанию равна 8. Функция SEND возвращает количество пробелов, которое заменяет табуляционную метку.

Если n > 0, пробелы заменяют каждый символ табуляции, до тех пор, пока указатель символа движется до следующего, кратного числу *n*. Например, с величиной по умолчанию 8, если символ TAB является третьим символом в записи,

символ табуляции заменяется на 6 пробелов.

Если n=0, драйвер перемещает символ табуляции без замены.

Если n<0, драйвер перемещает символ табуляции с положительной величиной n пробелов. Например, "TAB=-4" вызывает 4 пробела для замены каждого символа табуляции, независимо от позиции символа в записи.

Если n=-100, символы табуляции остаются как есть; драйвер *не* заменяет их пробелами.

QUICKSCAN

DRIVER('ASCII', '/QUICKSCAN = on | off')
[QScan» =] SEND(file, 'QUICKSCAN [= on | off]')

Устанавливает способ буферизации файла. Драйвер ASCII файлов за одно обращение читает буфер (а не запись) и тем самым ускоряет доступ к данным. В многопользовательской среде такой способ буферизации не дает 100% уверенности в правильности данных при последующем к ним доступе, так как за время между отдельными обращении данные могли быть изменены другим пользователем. В качестве меры безопасности драйвер повторно читает данные в буфер перед доступом к каждой записи. Чтобы *отключить* повторное чтение, установите QUICKSCAN на ON. По умолчанию стоит ON для файлов, открытых с запретом доступа по записи для других пользователей, и OFF для всех других режимов открытия файлов. Функция SEND возвращает установку Quickscan (ON или OFF) в виде STRING(3).

Атрибуты файла	**Поддержка**
CREATE	ДА
DRIVER [,*строка драйвера*])	ДА
NAME	ДА
ENCRYPT	НЕТ
OWNER(*Пароль)*	НЕТ
RECLAIM	НЕТ
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	Y[4]
EXTERNAL(*член*)	ДА
DLL([*признак*])	ДА
OEM	ДА

Структуры файла	Поддержка
INDEX	НЕТ
KEY	НЕТ
MEMO	НЕТ
BLOB	НЕТ
RECORD	ДА

Атрибуты индекса, ключа, мемо	Поддержка
BINARY	НЕТ
DUP	НЕТ
NOCASE	НЕТ
OPT	НЕТ
PRIMARY	НЕТ
NAME	НЕТ
Сортировка по возрастанию	НЕТ
Сортировка по убыванию	НЕТ
Смешанная сортировка	НЕТ

Атрибуты поля	Поддержка
DIM	ДА
OVER	ДА
NAME	ДА

Файловые процедуры	Поддержка
BOF(*файл*)	НЕТ
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	НЕТ
BUILD(*ключ*)	НЕТ
BUILD(*индекс*)	НЕТ
BUILD(*индекс*, *компонент*)	НЕТ
BUILD(*индекс*, *компонент*, *фильтр*)	НЕТ
BYTES(*файл*)	ДА
CLOSE(*файл*)	ДА
COPY(*файл*, *новый файл*)	ДА
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	НЕТ
DUPLICATE(*ключ*)	НЕТ
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА

FLUSH(*файл*)	НЕТ
LOCK(*файл*)	ДА
NAME(*метка*)	ДА
OPEN(*файл, режим доступа*)	ДА
PACK(*файл*)	НЕТ
POINTER(*файл*)	Y[2]
POINTER(*ключ*)	НЕТ
POSITION(*файл*)	Y[3]
POSITION(*ключ*)	НЕТ
RECORDS(*файл*)	НЕТ
RECORDS(*ключ*)	НЕТ
REMOVE(*файл*)	ДА
RENAME(*файл, новый файл*)	ДА
SEND(*файл, Сообщение*)	ДА
SHARE(*файл, режим доступа*)	ДА
STATUS(*файл*)	ДА
STREAM(*файл*)	НЕТ
UNLOCK(*файл*)	ДА

Доступ к записям	Поддержка
ADD(*файл*)	ДА
ADD(*файл, длина*)	НЕТ
APPEND(*файл*)	ДА
APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	НЕТ
GET(*файл, ключ)*	НЕТ
GET(*файл, указатель файла*)	ДА
GET(*файл, указатель файла, длина*)	НЕТ
GET(*ключ, указатель ключа*)	НЕТ
HOLD(*файл*)	НЕТ
NEXT(*файл*)	ДА
NOMEMO(*файл*)	НЕТ
PREVIOUS(*файл*)	НЕТ
PUT(*файл*)	Y[1]
PUT(*файл, указатель файла*)	Y[1]
PUT(*файл, указатель файла, длина*)	НЕТ
RELEASE(*файл*)	НЕТ
REGET(*файл,строка*)	ДА
REGET(*ключ,строка*)	НЕТ

RESET(*файл,строка*)	ДА
RESET(*ключ,строка*)	НЕТ
SET(*файл*)	ДА
SET(*файл, ключ*)	НЕТ
SET(*файл, указатель файла*)	ДА
SET(*ключ*)	НЕТ
SET(*ключ, ключ*)	НЕТ
SET(*ключ, указатель ключа*)	НЕТ
SET(*ключ, ключ, указатель файла*)	НЕТ
SKIP(*файл, число*)	НЕТ
WATCH(*файл*)	НЕТ

Обработка транзакций	Поддержка
LOGOUT(*вр. ож., файл, ..., файл*)	НЕТ
COMMIT	НЕТ
ROLLBACK	НЕТ

Обработка пустых записей	Поддержка
NULL(*поле*)	НЕТ
SETNULL(*поле*)	НЕТ
SETNONNULL(*поле*)	НЕТ

Примечания.

1 Когда используется оператор PUT(), в файл должно быть возвращено такое же количество символов, какое было перед этим прочитано из файла. Если вы возвращаете оператором PUT большее количество символов, чем было прочитано, эти «ЭКСТРА»-символы будут перекрывать символы в последующих записях файла. Если вы возвращаете оператором PUT меньшее количество символов, чем было прочитано, замещено в файле будет только возвращаемое количество символов. Остальные же символы в записи исходного файла останутся такими, какими они были до выполнения оператора PUT.

2 Функция POINTER() возвращает относительную позицию байта внутри файла.

3 POSITION(файл) возвращает STRING(4).

4 THREAD атрибут файла требует дополнительного блока управления файлом для каждого процесса, имеющего доступ к данному файлу.

Драйвер Basic

Спецификации

Драйвер Basic читает и пишет разграниченные запятыми ASCII файлы. Кавычки окружают строки символов, запятая разграничивает поля, а возврат каретки / перевод строки разграничивает записи. Впервые этот формат файла был использован в языке программирования BASIC. Драйвер BASIC не поддерживает ключи и обработку в обратном направлении, и поэтому BASIC файлы не являются хорошим выбором для произвольного доступа к данным.

Совет: Формат файла Basic хорошо подходит для тех случаев, когда необходимо иметь общие данные с программами электронных таблиц. Обычное расширение файла, используемое для этих файлов - это *.CSV*., что означает величины, разделенные запятой.

Файл: C4BASL.LIB	Статически вызываемая библиотека Windows (16-битовая)
C4BASXL.LIB	Статически вызываемая библиотека Windows (32- битовая)
C4BAS.LIB	Библиотека экспорта Windows (16-битовая)
C4BASX.LIB	Библиотека экспорта Windows (32-битовая)
C4BAS.DLL	Динамически вызываемая библиотека Windows (16-битовая)
C4BASX.DLL	Динамически вызываемая библиотека Windows (32-битовая)

Поддерживаемые типы данных

BYTE	DECIMAL
SHORT	PDECIMAL
USHORT	STRING
LONG	CSTRING
ULONG	PSTRING
SREAL	DATE
REAL	TIME
BFLOAT4	GROUP
BFLOAT8	

Характеристики файла/максимумы

Размер файла:	4,294,967,295 байт
Записей на файл:	4,294,967,295 байт
Размер записи:	65,520 байт
Размер поля:	65,520 байт
Полей на запись:	65,520
Ключей/Индексов на файл:	Нет
Размер ключа:	Нет
Полей мемо на файл:	Нет
Размер поля мемо:	Нет
Открытие файла данных::	Зависит от операционной системы

Строки драйвера

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. См. Общие характеристики драйверов - Строки драйвера.

Замечание: Некоторые строки символов драйвера не действуют после открытия файла, поэтому синтаксис функции SEND для модификации установки не перечислен. Однако синтаксис функции SEND для возврата величины переключения приводится в перечне для всех строк драйвера.

BASIC Драйвер поддерживает следующие строки драйвера:

ALWAYSQUOTE

```
DRIVER('BASIC', '/ALWAYSQUOTE = on | off' )
[ QScan» = ] SEND(file, 'ALWAYSQUOTE [ = on | off ]' )
```

Для совместимости с файлами данных в формате Basic, созданными продуктами, которые *не* помещают строки символов в кавычки, поставьте ALWAYSQUOTE на OFF.

Когда содержимое поля строки символов включает запятую или кавычку, а

ALWAYSQUOTE выключено, драйвер Basic автоматически помещает кавычки вокруг строки символов при записи в файл. Это также относится к символам-разделителям, установленным с FIELDDELIMETER или COMMA. Например, при использовании по умолчанию и при ALWAYSQUOTE выключенном, поле STRING, содержащее величину *1313 Mockingbird Lane, Apt.33* автоматически сохраняется как: "1313 Mockingbird Lane, Apt.33"

Функция SEND возвращает установку ALWAYSQUOTE (ON или OFF) как STRING(3).

COMMA

DRIVER('BASIC', /COMMA = n')
[Comma» =] SEND(file, 'COMMA [= n]')

Устанавливает символ разделитель полей.

n представляет ASCII код для символа конца поля. По умолчанию 1,44, что эквивалентно "/FIELDDELIMITER=1,44".

Функция SEND возвращает ASCII код символа разделителя полей.

CTRLZISEOF

DRIVER('Basic', '/CTRLZISEOF = on | off')
[EOF» =] SEND(file, 'CTRLZISEOF [= on | off]')

По умолчанию (CTRLZISEOF=on) драйвер файла подразумевает, что любой символ Ctrl-Z внутри файла указывает на конец файла. Для отключения этого свойства установите CTRLZISEOF=off. Функция SEND возвращает установленное значение CTRLZISEOF как STRING(3).

ENDOFRECORD

DRIVER('Basic', '/ENDOFRECORD = n [,m]')
[EOR» =] SEND(file, 'ENDOFRECORD [= n [,m]]')

Определяет символ-ограничитель записей.

n представляет число символов, которые образуют символ-ограничитель записи;

m представляет ASCII код(ы) для символов конца записи, отделенные запятыми. По умолчанию 2,13,10, показывая тем самым, что 2 символа отмечают конец записи, возврат каретки (13) и перевод строки (10).Функция SEND возвращает символ-ограничитель записи.

Совет: Мэйнфреймы часто используют для разграничения записей только

символ возврата каретки. Вы можете использовать ENDOFRECORD для чтения этих файлов.

ENDOFRECORDINQUOTE

```
DRIVER('BASIC', '/ENDOFRECORDINQUOTE = on|off' )
[ EORQuote» = ] SEND(file, 'ENDOFRECORDINQUOTE [ = on|off ]' )
```

По умолчанию (ENDOFRECORDINQUOTE=ON) драйвер файла не распознает маркер конца записи, расположенный внутри строки. Для того, чтобы маркер конца записи всегда сигнализировал о конце записи, необходимо установить ENDOFRECORDINQUOTE=OFF.

Функция SEND возвращает текущее значение ENDOFRECORDINQUOTE (ON или OFF) в форме STRING(3).

FIELDDELIMITER

```
DRIVER('BASIC', '/FIELDDELIMITER = n [,m ]' )
[ Limiter» = ] SEND(file, 'FIELDDELIMITER [ = n [,m ]]' )
```

Определяет символ-разграничитель полей. Этот символ действует в дополнение к любой строке разграничителю определенной строкой драйвера /QUOTE.

n представляет число символов, образующих разграничитель полей;

m представляет разделенные запятыми ANSI код(ы) символов разграничителя полей. По умолчанию 1,44.

Функция SEND возвращает текущее значение символов разграничителя полей .

Замечание: Если заданы одновременно FIELDDELIMITER и COMMA, действует строка драйвера, заданная последней.

FILEBUFFERS

```
DRIVER('Basic', '/FILEBUFFERS = n' )
[ Buffers» = ] SEND(file, 'FILEBUFFERS [ = n ]' )
```

Устанавливает размер буфера, используемого для чтения и записи, размер буфера равен n*512. Используйте строку драйвера /FILEBUFFERS для увеличения размера буфера, если доступ к данным слишком медленный. Максимальный размер буфера равен 65,504 для 16-битовых приложений и 4,294,967,264 для 32-битовых. Функция SEND возвращает размер буфера в байтах.

Замечание: По умолчанию размер буфера для файлов открытых с запретом на

запись для других пользователей больше 1024 или (2*размер записи) и больше 512 или (1* размер записи) для всех других режимов открытия.

QUICKSCAN

```
DRIVER('BASIC', '/QUICKSCAN = on|off' )
[ QScan» = ] SEND(file, 'QUICKSCAN [ = on|off ]' )
```

Устанавливает способ буферизации файла. Драйвер BASIC файлов за одно обращение читает буфер (а не запись) и тем самым ускоряет доступ к данным. В многопользовательской среде такой способ буферизации не дает 100% уверенности в правильности данных при последующем к ним доступе, так как за время между отдельными обращениями данные могли быть изменены другим пользователем. В качестве меры безопасности драйвер повторно читает данные в буфер перед доступом к каждой записи. Чтобы *отключить* повторное чтение, установите QUICKSCAN на ON. По умолчанию стоит ON для файлов, открытых с запретом доступа по записи для других пользователей, и OFF для всех других режимов открытия файлов. Функция SEND возвращает установку Quickscan (ON или OFF) в виде STRING(3).

Замечание: Величины, разграниченные TAB, представляют обычный формат, совместимый с буфером обмена Windows. Использование строки символов драйвера файла BASIC /COMMA=9 позволит вам читать файлы буфера обмена Windows.

QUOTE

```
DRIVER('BASIC', '/QUOTE = n' )
[ Quote» = ] SEND(file, 'QUOTE [ = n ]' )
```

Определяет символ ограничитель строк.
n ANSI код ограничителя строки. По умолчанию 34.
Функция SEND возвращает текущее значение символа ограничителя строки.

Атрибуты файла	**Поддержка**
CREATE	ДА
DRIVER [*драйвер,строка драйвера*])	ДА
NAME	ДА
ENCRYPT	НЕТ
OWNER(*Пароль)*	НЕТ
RECLAIM	НЕТ

PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	Y[4]
EXTERNAL(*член*)	ДА
DLL([*признак*])	ДА
OEM	ДА

Структуры файла	Поддержка
INDEX	НЕТ
KEY	НЕТ
MEMO	НЕТ
BLOB	НЕТ
RECORD	ДА

Атрибуты индекса, ключа, мемо	Поддержка
BINARY	НЕТ
DUP	НЕТ
NOCASE	НЕТ
OPT	НЕТ
PRIMARY	НЕТ
NAME	НЕТ
Сортировка по возрастанию	НЕТ
Сортировка по убыванию	НЕТ
Смешанная сортировка	НЕТ

Атрибуты поля	Поддержка
DIM	ДА
OVER	ДА
NAME	ДА

Файловые процедуры	Поддержка
BOF(*файл*)	НЕТ
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	НЕТ
BUILD(*ключ*)	НЕТ
BUILD(*индекс*)	НЕТ
BUILD(*индекс*, *компонент*)	НЕТ
BUILD(*индекс*, *компонент*, *фильтр*)	НЕТ
BYTES(*файл*)	ДА

CLOSE(*файл*)	ДА
COPY(*файл, новый файл*)	ДА
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	НЕТ
DUPLICATE(*ключ*)	НЕТ
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА
FLUSH(*файл*)	НЕТ
LOCK(*файл*)	ДА
NAME(*метка*)	ДА
OPEN(*файл, режим доступа*)	ДА
PACK(*файл*)	НЕТ
POINTER(*файл*)	Y[2]
POINTER(*ключ*)	НЕТ
POSITION(*файл*)	Y[3]
POSITION(*ключ*)	НЕТ
RECORDS(*файл*)	НЕТ
RECORDS(*ключ*)	НЕТ
REMOVE(*файл*)	ДА
RENAME(*файл, новый файл*)	ДА
SEND(*файл, Сообщение*)	ДА
SHARE(*файл, режим доступа*)	ДА
STATUS(*файл*)	ДА
STREAM(*файл*)	НЕТ
UNLOCK(*файл*)	ДА

Доступ к записям	Поддержка
ADD(*файл*)	ДА
ADD(*файл, длина*)	НЕТ
APPEND(*файл*)	ДА
APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	НЕТ
GET(*файл, ключ*)	НЕТ
GET(*файл, указатель файла*)	ДА
GET(*файл, указатель файла, длина*)	НЕТ
GET(*ключ, указатель ключа*)	НЕТ
HOLD(*файл*)	НЕТ
NEXT(*файл*)	ДА
NOMEMO(*файл*)	НЕТ

PREVIOUS(*файл*)	НЕТ
PUT(*файл*)	Y[1]
PUT(*файл, указатель файла*)	Y[1]
PUT(*файл, указатель файла, длина*)	НЕТ
RELEASE(*файл*)	НЕТ
REGET(*файл,строка*)	ДА
REGET(*ключ,строка*)	НЕТ
RESET(*файл,строка*)	ДА
RESET(*ключ,строка*)	НЕТ
SET(*файл*)	ДА
SET(*файл, ключ*)	НЕТ
SET(*файл, указатель файла*)	ДА
SET(*ключ*)	НЕТ
SET(*ключ, ключ*)	НЕТ
SET(*ключ, указатель ключа*)	НЕТ
SET(*ключ, ключ, указатель файла*)	НЕТ
SKIP(*файл, число*)	НЕТ
WATCH(*файл*)	НЕТ

Обработка транзакций	Поддержка
LOGOUT(*вр. ож., файл, ..., файл*)	НЕТ
COMMIT	НЕТ
ROLLBACK	НЕТ

Обработка пустых записей	Поддержка
NULL(*поле*)	НЕТ
SETNULL(*поле*)	НЕТ
SETNONNULL(*поле*)	НЕТ

Примечания.

1 Когда используется оператор PUT(), в файл должно быть возвращено такое же количество символов, какое было перед этим прочитано из файла. Если вы возвращаете оператором PUT большее количество символов, чем было прочитано, эти «ЭКСТРА» - символы будут перекрывать символы в последующих записях файла. Если вы возвращаете оператором PUT меньшее количество символов, чем было прочитано, замещено в файле будет только возвращаемое количество символов. Остальные же символы в записи исходного файла останутся такими, какими они были до выполнения оператора PUT.

2 Функция POINTER() возвращает относительную позицию байта внутри файла.

3 POSITION(файл) возвращает STRING(4).

4 THREAD атрибут файла требует дополнительного блока управления файлом для каждого процесса, имеющего доступ к данному файлу.

Драйвер Btrieve

Спецификации

Этот драйвер файла читает и записывает файлы Btrieve, используя прямой низкоуровневый доступ.

В рамках Clarion for Windows драйвер файла Btrieve выполнен с использованием .DLL и .EXE, поставленного Pervasive Software (прежде Btrieve Technologies, Inc.). Для приложения, которое должно использовать драйвер файлов Btrieve, исполняемая программа должна сопровождаться следующими EXE и DLL файлами:

16-bit

WBTR32.EXE
WBTRLOCL.DLL
WBTRCALL.DLL
WBTRVRES.DLL

32-bit

Вы должны приобрести 32-bit Btrieve процессор. Заказать процессор Вы можете в TopSpeed по телефону 1-800-354-5444 или в Pervasive Software.

ЛИЦЕНЗИОННОЕ ПРЕДУПРЕЖДЕНИЕ: Зарегистрированный владелец Clarion не может распространять вышеупомянутые файлы за пределами своей организации без лицензии от Pervasive Software. Для того, чтобы получить лицензию, пожалуйста обратитесь в:

Pervasive Software
8834 Capital of Texas Highway North, Suite 300
Austin, Texas 78759
Phone: (800) 287-4383 or (512) 794-1719
Internet: www.pervasive.com

C4BTRL.LIB	Статически вызываемая библиотека Windows (16-битовая)
C4BTRXL.LIB	Статически вызываемая библиотека Windows (32- битовая)
C4BTR.LIB	Библиотека экспорта Windows (16-битовая)
C4BTRX.LIB	Библиотека экспорта Windows (32-битовая)
C4BTR.DLL	Динамически вызываемая библиотека Windows (16-битовая)
C4BTRX.DLL	Динамически вызываемая библиотека Windows (32-битовая)

Поддерживаемые типы данных

Типы данных Clarion	Типы данных Btrieve
BYTE	STRING (1 байт)
SHORT	INTEGER (2 байта)
LONG	INTEGER (4 байта)
SREAL	FLOAT (4 байта)
REAL	FLOAT (8 байтов)
BFLOAT4	BFLOAT (4 байта)
BFLOAT8	BFLOAT (8 байтов)
PDECIMAL	DECIMAL
STRING	STRING
CSTRING	ZSTRING
PSTRING	LSTRING
DATE	DATE
TIME	TIME
USHORT	UNSIGNED BINARY (2 байта)
ULONG	UNSIGNED BINARY (4 байта)
MEMO	STRING,LVAR или NOTE
BYTE,NAME('LOGICAL')	LOGICAL*
USHORT,NAME('LOGICAL')	LOGICAL*
PDECIMAL,NAME('MONEY')	MONEY*
STRING(@N0n-),NAME('STS')	SIGNED TRAILING SEPARATE*
DECIMAL*	

Замечания:

• Вы можете сохранить типы Clarion DECIMAL в файле Btrieve. Однако, вы не можете построить ключ или индекс, используя это поле. Это сделано для предоставления обратной совместимости со старыми Clarion программами, которые используют Btrieve LEM. Если вам необходимы стандартные Btrieve десятичные данные, которые совместимы с любыми другими Btrieve программами, вы должны использовать тип данных PDECIMAL и избегать данных типа DECIMAL

• Если вы хотите создать файл с типами полей LOGICAL или MONEY, вы должны установить LOGICAL или MONEY с атрибутом поля NAME, соответственно. Если вы осуществляете доступ к существующему файлу, атрибут NAME не требуется.

Поле LOGICAL может быть объявлено как BYTE или USHORT, в зависимости от того, является ли оно одно- или двухбайтным LOGICAL:

```
LogicalField1                BYTE          !Одно байтовый LOGICAL
LogicalField2                USHORT   !Двух байтовый LOGICAL
```

MONEY может быть объявлено как PDECIMAL(x,2), где x - это общее число разрядов для хранения:

```
MoneyField  PDECIMAL(7,2),NAME('MONEY')      !хранит до 99999,99
```

Поля Btrieve NUMERIC не полностью поддерживаются драйвером. Btrieve NUMERIC сохраняется, как строка символов с последним символом, содержащим разрядность и знак. Возможные величины для этого последнего символа:

```
                1 2 3 4 5 6 7 8 9 0
  Positive:     A B C D E F G H I (
  Negative:     J K L M N O P Q R )
```

Чтобы получить доступ к полю NUMERIC, вы должны определить STRING(@NOx), где x на единицу меньше, чем число разрядов в NUMERIC, и STRING (1) для индикатора знака. Драйвер Btrieve не управляет этим знаковым полем, приложение должно быть написано так, чтобы непосредственно обслуживать его.

Например, для того, чтобы получить доступ к NUMERIC(7):

```
NumericGroup GROUP                !хранит от -999999 до 999999
Number       STRING(@NO6)          !числа
Sign         STRING(1)             !символьный индикатор
             END
```

Характеристики файла/максимумы

Размер файла:	4,000,000,000 байт
Записей на файл:	Ограничено размерами файла
Размер записи:	
Клиент:	65,535 байт переменной длины
Сервер	57,000 байт переменной длины
Размер поля:	65,520 байт
Полей на запись:	65,520
Ключей/Индексов на файл:	24 с NLM5 256 с NLM6

	Client	Btrieve	v6.15
	Размер страницы	Макс. сегментов	ключей
	512	8	
	1,024	2	3
	1,536	2	4
	2,048	5	4
	4,096	119	
	Это общее число компонент. Если у вас многокомпонентный ключ, построенный из трех полей, это считается, как три индекса при счете числа разрешенных индексов.		
Размер ключа:	255 байт		
Полей мемо на файл:	Зависит от системы		
Размер поля мемо:	65520 байт		
Открытие файла данных::	Зависит от операционной системы		

Строки драйвера

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. См. Общие характеристики драйверов - Строки драйвера.

Драйвер Btrieve поддерживает следующие строки драйвера:

ACS

```
DRIVER('BTRIEVE', '/ACS = filename' )
[ SortSeq» = ] SEND(file, 'ACS [ = filename ]' )
```

Когда создается Btrieve файл, вы можете указать альтернативную последовательность сортировки для STRING ключей. Эти сортирующая последовательность обычно извлекается из последовательности, заданной в INI файле вашей программы. Однако, Btrieve поставляет файлы поддержки интенсивной сортировки. Для того, чтобы создать ваш файл с использованием этой сортирующей последовательности, вы должны указать имя сортирующего файла в строке драйвера.

ALLOWREAD

```
DRIVER('BTRIEVE', '/ALLOWREAD = ON|OFF' )
[ Read» = ] SEND(file, 'ALLOWREAD [ = ON|OFF ]' )
```

По умолчанию к файлу Btrieve, созданному с именем владельца, можно получить доступ *только* в режиме "только-для-чтения" если имя владельца неизвестно. Чтобы предупредить *любой* доступ к файлу без имени владельца, установите ALLOWREAD на OFF.

Функция SEND возвращает текущее значение ALLOWREAD (ON или OFF) в форме STRING(3).

APPENDBUFFER

```
DRIVER('BTRIEVE', '/APPENDBUFFER = size ' )
[ Buffer» = ] SEND(file, 'APPENDBUFFER [ = size ]' )
```

По умолчанию APPEND добавляет все записи к файлу за один раз. Чтобы получить лучшую эффективность в сети, вы можете сообщить драйверу о необходимости построить буфер записей, а затем отправить их в файл одной операцией. Это делается с помощью SEND(file,'APPENDBUFFER=size'), где size - число записей, которые вы хотите разместить в буфер. Функция SEND возвращает текущее значение объема буфера в количестве записей.

BALANCEKEYS

```
DRIVER('BTRIEVE', '/BALANCEKEYS = ON|OFF' )
[ Balance» = ] SEND(file, 'BALANCEKEYS [ = ON|OFF ]' )
```

При создании файла Btrieve вы можете использовать эту строку символов драйвера, чтобы сообщить Btrieve, что все ключи, связанные с файлом, должны быть сохранены в виде сбалансированного двоичного дерева. Это сэкономит пространство диска, но замедлит выполнение операций добавления, удаления и обновления там, где изменяются величины ключей. Функция SEND возвращает текущее значение BALANCEKEYS (ON или OFF) в форме STRING(3).

COMPRESS

```
DRIVER('BTRIEVE', '/COMPRESS = ON|OFF' )
[ Read» = ] SEND(file, 'COMPRESS [ = ON|OFF ]' )
```

Btrieve дает вам возможность сжать данные перед хранением. Это позволяет снизить требования к хранению, но одновременно уменьшает эффективность. Когда COMPRESS в позиции ON, CREATE создает сжатый файл Btrieve. Функция SEND

возвращает текущее значение COMPRESS (ON или OFF) в форме STRING(3).

FREESPACE

```
DRIVER('BTRIEVE', '/FREESPACE = 0|10|20|30' )
[ Read» = ] SEND(file, 'FREESPACE [ = 0|10|20|30 ]' )
```

Устанавливает процент свободного пространства для поддержания страниц переменной длины. Величина по умолчанию - нуль. Функция SEND возвращает процент свободного пространства для поддержания страниц переменной длины в форме STRING.

LACS

```
DRIVER('BTRIEVE', '/LACS [ =|country_id,codepage ]' )
[ Sequence» = ] SEND(file, 'LACS [ =|country_id,codepage ]' )
```

В версии Btrieve 6.15 Btrieve добавлена характеристика Local Alternate Collating Sequences (локальные переменные сортирующие последовательности). Это позволяет вашему ключу в виде строки символов сортировать на основе кода страны для машины, выполняющей вашу программу. Чтобы использовать эту характеристику, поместите '/LACS=' в строку символов вашего драйвера.

/LACS=country_ID,code_page

Вы можете также указать определяемые пользователем Alternate Collating Sequences (переменные сортирующие последовательности). Это позволяет вашему ключу в виде строки символов сортировать на основе DOS кода страны и кода страницы для конкретной страны. Чтобы использовать эту характеристику, поместите '/LACS=country_id,codepage' в строку символов вашего драйвера. Запомните, что не должно быть пробелов вокруг запятой. Функция SEND возвращает country_ID,code_page или строку символов ',' (если используется зависящее от машины LACS.

MEMO

```
DRIVER('BTRIEVE', '/MEMO = SINGLE|LVAR|NOTE [,delimiter]' )
[ Memo» = ] SEND(file, 'MEMO [ = SINGLE|LVAR|NOTE [,delimiter]]' )
```

/MEMO=SINGLE

Чтобы получить доступ к существующим файлам Btrieve, созданным с помощью Btrieve LEM из Clarion 2.1, или файлам с переменной длиной записей, установите MEMO на SINGLE.

/MEMO=LVAR

Чтобы получить доступ к файлу с переменной длиной записей, используйте MEMO в стиле SINGLE, чей размер равняется максимальному размеру компоненты записи переменной длины. Чтобы добавить/поместить записи к файлу этого стиля с бинарными данными, сохраненными в секции переменной длины, используйте ADD(файл,длина), APPEND(файл,длина) и PUT(файл,позиция,длина) функции. Драйвер игнорирует параметр *позиция* в функции PUT, но инициирует его в *0(нуль)* для будущей совместимости. Функции ADD, APPEND или PUT удалят все концевые пробелы для текстовых мемо и NULL символы для бинарных мемо перед сохранением записи.

/MEMO=NOTE,<delimiter>

Чтобы получить доступ к файлам данных Xtrieve, которые имеют тип данных Note или LVar.

При типе данных NOTE установите разграничитель конца поля. Установите величину ASCII для разграничителя. Мемо NOTE и LVAR не требуют использования размерных вариантов ADD, APPEND и PUT при сохранении записей. Маркер конца записи не обязателен для мемо стиля NOTE. Драйвер автоматически добавляет маркер конца записи перед сохранением записи и удаляет его перед помещением данных мемо в буфер мемо.

В качестве примера "/MEMO=NOTES,141" указывает файл с полем Xtrieve Notes, использующим CR/LF в качестве разграничителя. Чтобы получить более подробную информацию о типах данных Xtrieve, обратитесь к документации, поставляемой Novell.

SEND(file,'MEMO')

Возвращает установку MEMO: NORMAL, NOTE,LVAR или SINGLE.

PAGESIZE

```
DRIVER('BTRIEVE', '/PAGESIZE=SIZE' )
[ PSize» = ] SEND(file, 'PAGESIZE[=SIZE' )
```

Устанавливает размер страницы Btrieve Page во время создания файла. Размер должен быть всегда кратным 512 с максимумом 4096. Большой размер страницы обычно приводит к более эффективному хранению на диске. *Не прибавляйте пробелов перед или после* знака равенства.

Функция SEND возвращает текущее значение размера страницы в форме

STRING.

PREALLOCATE

```
DRIVER('BTRIEVE', '/PREALLOCATE = n' )
[ Read» = ] SEND(file, 'PREALLOCATE [ = n ]' )
```

При создании файла Btrieve вы можете заранее выделить *n* страниц пространства диска для размещения файла. Установка по умолчанию - нуль.

Функция SEND возвращает число страниц выделенного пространства диска в форме STRING.

TRUNCATE

```
DRIVER('BTRIEVE', '/TRUNCATE  = ON|OFF' )
[ Trunc» = ] SEND(file, 'TRUNCATE [ = ON|OFF ]' )
```

При создании файла Btrieve вы можете использовать эту строку символов драйвера, чтобы сообщить Btrieve о необходимости отсечения остаточных пробелов. Это заставляет Btrieve хранить запись, как хранятся записи переменной длины. Функция SEND возвращает текущее значение TRUNCATE(ON или OFF) в форме STRING(3).*

Атрибуты файла	**Поддержка**
CREATE	ДА
DRIVER [*драйвер,строка драйвера*])	ДА
NAME	ДА
ENCRYPT	ДА
OWNER(*Пароль)*	ДА[1]
RECLAIM	ДА
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	Y[15]
EXTERNAL(*член*)	ДА
DLL([*признак*])	ДА
OEM	ДА

Структуры файла	Поддержка
INDEX	ДА
KEY	ДА
MEMO	ДА[2]

BLOB	НЕТ
RECORD	ДА

Атрибуты индекса, ключа, мемо	Поддержка
BINARY	ДА[16]
DUP	ДА
NOCASE	ДА
OPT	ДА
PRIMARY	ДА
NAME	ДА[2]
Сортировка по возрастанию	ДА
Сортировка по убыванию	ДА
Смешанная сортировка	ДА

Атрибуты поля	Поддержка
DIM	ДА
OVER	ДА
NAME	ДА

Файловые процедуры	Поддержка
BOF(*файл*)	ДА[10]
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	ДА[3]
BUILD(*ключ*)	ДА[3]
BUILD(*индекс*)	ДА[3]
BUILD(*индекс*, *компонент*)	НЕТ
BUILD(*индекс*, *компонент*, *фильтр*)	НЕТ
BYTES(*файл*)	НЕТ
CLOSE(*файл*)	ДА
COPY(*файл*, *новый файл*)	ДА
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА[10]
FLUSH(*файл*)	ДА
LOCK(*файл*)	НЕТ[4]
NAME(*метка*)	ДА
OPEN(*файл*, *режим доступа*)	ДА

PACK(*файл*)	ДА
POINTER(*файл*)	ДА[11]
POINTER(*ключ*)	ДА[11]
POSITION(*файл*)	ДА[12]
POSITION(*ключ*)	ДА[12]
RECORDS(*файл*)	ДА
RECORDS(*ключ*)	ДА
REMOVE(*файл*)	ДА
RENAME(*файл*, *новый файл*)	ДА
SEND(*файл*, *Сообщение*)	ДА
SHARE(*файл*, *режим доступа*)	ДА
STATUS(*файл*)	ДА
STREAM(*файл*)	ДА
UNLOCK(*файл*)	НЕТ

Доступ к записям	Поддержка
ADD(*файл*)	ДА[5]
ADD(*файл*, *длина*)	ДА[5]
APPEND(*файл*)	ДА[6]
APPEND(*файл*, *длина*)	ДА[5,6]
DELETE(*файл*)	ДА[7]
GET(*файл*, *ключ*)	ДА
GET(*файл*, *указатель файла*)	ДА
GET(*файл*, *указатель файла*, *длина*)	НЕТ
GET(*ключ*, *указатель ключа*)	ДА
HOLD(*файл*)	ДА
NEXT(*файл*)	ДА
NOMEMO(*файл*)	ДА
PREVIOUS(*файл*)	ДА
PUT(*файл*)	ДА[5]
PUT(*файл*, *указатель файла*)	НЕТ
PUT(*файл*, *указатель файла*, *длина*)	ДА
RELEASE(*файл*)	ДА
REGET(*файл*,*строка*)	ДА
REGET(*ключ*,*строка*)	ДА
RESET(*файл*,*строка*)	ДА
RESET(*ключ*,*строка*)	ДА
SET(*файл*)	ДА
SET(*файл*, *ключ*)	ДА

SET(*файл, указатель файла*)	ДА[8]
SET(*ключ*)	ДА
SET(*ключ, ключ*)	ДА
SET(*ключ, указатель ключа*)	ДА[8]
SET(*ключ, ключ, указатель файла*)	ДА[9]
SKIP(*файл, число*)	ДА
WATCH(*файл*)	ДА

Обработка транзакций	Поддержка
LOGOUT(*вр. ож., файл, ..., файл*)	ДА[13,14]
COMMIT	ДА[14]
ROLLBACK	ДА

Обработка пустых записей	Поддержка
NULL(*поле*)	НЕТ
SETNULL(*поле*)	НЕТ
SETNONNULL(*поле*)	НЕТ

Примечания

1 Мы рекомендуем использовать длинные и содержащие специальные символы значения пароля, так как именно таким способом обеспечивается большая эффективность защиты. Например, пароль подобный «;%?? $$^» гораздо сложнее разгадать, чем пароль типа «SALARY.»

Совет: Для определения переменной, содержащей актуальное значение пароля в поле Owner Name в диалоге «Свойства файла» (File Properties dialog), перед именем переменной наберите восклицательный знак '!'. Например: !MyPassword.

2 Драйвер игнорирует любой атрибут NAME на поле MEMO. Поля MEMO могут находиться либо в отдельном файле, либо в файле данных, если строка драйвера MEMO установлена на SINGLE, LVAR или NOTE. Если строка драйвера MEMO не установлена, имя отдельного файла MEMO - "MEM", предваряемое первыми пятью символами метки файла, плюс расширение файла "DAT". Установка строки драйвера MEMO ограничивает вас одним мемо полем на файл.

3 Если используется после APPEND(), но прежде, чем файл закрыт, это добавляет ключи, опущенные APPEND(). Во всех других случаях BUILD() перестраивает файл и ключи. Если вы хотите только перестроить ключи, то выполнить BUILD(ключ) для каждого ключа быстрее, чем BUILD(файл).

4 Btrieve прямо не поддерживает захват файла, Если вам требуется захват файла, используйте LOGOUT.

5 При использовании мемо типа LVAR или NOTE, убедитесь, что мемо имеет подходящую структуру. Если структура неправильная, а драйвер рассчитывает длину, большую, чем максимальный размер мемо, определенный для данного файла, эти функции отказывают и поэтому код ошибки устанавливается на 57 - Недопустимый файл мемо.

6 Btrieve не поддерживает бесключевые изменения. Чтобы эмулировать поведение APPEND(), драйвер опускает все возможные индексы при первом вызове. Вызов BUILD() немедленно после дополнения записей перестраивает опущенные ключевые поля.

7 DELETE Btrieve разрушает информацию позиционирования при обработке в физическом порядке. Драйвер пытается изменить позицию к соответствующей записи. Это не всегда возможно и может потребовать от драйвера читать с начала файла. Использование ключевого порядка обработки позволяет избежать этого возможного замедления.

8 Если указатель файла или указатель ключа имеют величину нуль, драйвер игнорирует параметр указателя. Обработка устанавливается или на файловый, либо на ключевой порядок, указатель записи устанавливается на первый элемент.

9 Если указатель файла имеет величину нуль, обработка начинается на величине первого ключа, чья позиция больше, чем (или меньше, чем для PREVIOUS) указатель файла.

10 Эти функции поддерживаются, но они не рекомендуются. Они вызывают большее количество вводов/выводов с диска, чем ERRORCODE(). Btrieve возвращает *eof* при чтении после последней записи. Следовательно, драйвер должен читать следующую запись, а затем *следующую*, чтобы увидеть, конец ли это файла, а затем вернуться к записи, если вы хотите.

11 POINTER() возвращает относительную позицию внутри файла, но не номер записи.

12 POSITION(file) возвращает STRING(4). POSITION(key) возвращает

STRING длиной, равной размеру поля ключа + 4 байта.

13 Если система разрушается во время прохождения транзакции (LOGOUT—COMMIT), восстановление происходит автоматически драйвером Btrieve сразу, как только будет новое обращение к поврежденному файлу.

Когда вы выдаете вызов LOGOUT(), все файлы Btrieve, к которым был доступ во время транзакции, выводятся из системы. Это означает, что следующий фрагмент программы некорректен, так как вы не можете закрыть выведенный из системы файл:

```
LOGOUT(1,file1)
OPEN(file1)
CLOSE(file1)
```

14 См. также *PROP:Logout* в описании языка.

15 Файлы с атрибутом THREAD не требуют дополнительного блока управления файлом при доступе к нему из новых процессов.

16 OEM - преобразование неприменимо к BINARY MEMOs. Драйвер подразумевает, что BINARY MEMO заполнено нулями или пробелами.

Прочее

Клиент/Сервер

Для Btrieve на основе Клиент/Сервер Netware Btrieve является Серверо-ориентированной версией Btrieve, которая работает на сервере Novell.

Структура файла

Одиночный файл обычно хранит и данные и все ключи. Имена файлов данных по умолчанию получают расширение файла *.DAT. По умолчанию драйвер сохраняет мемо-поля в отдельном файле или в самом файле данных, если на это имеется указание в соответствующей строке символов драйвера.

Так как Btrieve является независимым от модели данных обработчиком индексированных записей, он не хранит описания полей данных внутри своих файлов. Приложение, обращающееся к этим файлам, определяет, как интерпретировать записи файлов на основе файла определения данных (.DDF). Отсутствие .DDF файла, описывающего Btrieve файл, создает очень большие проблемы для приложения, интерпретирующего данные из Btrieve файла.

Файл формата Btrieve сохраняет минимальную информацию о структуре информации в файле. Драйвер проверяет достоверность вашего описания относительно информации в файле. Тем не менее, возможно успешное открытие файла Btrieve, который имеет определения ключей, не совпадающих точно с вашим

определением. Вы должны быть уверены, что ваш файл и определения ключей точно соответствуют файлу Btrieve.

Ключи и индексы

Ключи KEYs - динамические и автоматически обновляются при изменении файла данных.

Индексы INDEXs сохраняются отдельно от файлов данных. Файлы INDEX получают временное имя файла и удаляются, когда программа заканчивается нормально. INDEXs -статичны - они не обновляются автоматически при изменении файлов данных. Оператор BUILD создает или обновляет индексные файлы.

Длина записей

Драйвер сохраняет записи менее 4К как записи фиксированной длины. Записи более 4К он сохраняет как записи переменной длины. Минимальная длина записи 4 байта. Одна запись может удерживаться в каждом открытом файле каждым пользователем.

Размер страницы

Чтобы определить физическую длину записи, прибавьте 8 байт для каждого KEY, который разрешает дублирование. Прибавьте 4 байта, если файл допускает переменную длину записи. Наконец, прибавьте 6 байт для заголовка на страницу.

Например: Если размер записи равен 300 байт, а файл имеет три ключа KEY, которые допускают дублирование, полный размер записи равен:

300	*размер записи*
x 24	*заголовок для трех KEY с атрибутом DUP*
= 324	*физическая длина записи*

Размер страницы 512 содержит только одну такую запись, а 182 байта на страницу останутся неиспользованными (512-6-324). Если бы размер страницы был 1024, на страницу пришлись бы три сохраненных записи и только 46 байт остались бы неиспользованными (1024-6-(324*3)).

Вы должны загрузить BTRIEVE.EXE с размером страницы, равным или большим, чем самый большой размер страницы любого файла, к которому вы получите доступ.

Разделение файлов

Btrieve позволяет вам открыть файл в пяти различных форматах: NORMAL, ACCELERATED, READ-ONLY, VERIFY или EXCLUSIVE. Эквивалентные

состояния Clarion OPEN:

Btrieve State	Clarion режим доступа OPEN/SHARE
ACCELERATED	Читать/записывать с FCB режимом совместимости (2H)
READ-ONLY	Только читать(0H,10H,20H,30H,40H)
VERIFY	Только записывать с FCB совместимостью (1H)
EXCLUSIVE	Только писать с любым флажком отрицания Deny (11H,21H,31H,41H); Читать/писать с Deny All, читать и писать (12H,22H,32H)
NORMAL	Читать/писать с Deny None (не отрицать никого)(42H)

Btrieve позволяет файлу иметь особого владельца. Смотрите строку символов драйвера /READONLY относительно деталей установки этого флажка. Файл может быть также зашифрован. Это устанавливается с помощью атрибута ENCRYPT. Файл может быть зашифрован только тогда, когда дано имя владельца.

Указатель записи

Btrieve использует беззнаковую длинную переменную для своего внутреннего указателя записи; отрицательные величины лишаются своего знака. Мы рекомендуем тип данных ULONG для вашего указателя записи.

Сортирующая последовательность

- Атрибут ключа: **NOCASE**

NLM 5 не поддерживает нечувствительное к регистру индексирование. При необходимости вы должны иметь переменную, сортирующую последовательность, которая нечувствительна к регистру сортировки.

Btrieve поддерживает альтернативную сортирующую последовательность. Однако NLM 6 не поддерживает *ни* NOCASE, *ни* переменную, сортирующую последовательность. Если вы установите и то и другое, преимущество имеет атрибут NOCASE. Функция SEND не возвращает никакой ошибки.

- Изменение сортирующей последовательности

Btrieve хранит сортирующую последовательность вне файла. Чтобы изменить сортирующую последовательность вы должны изменить .ENV файл, затем создать новый Btrieve файл на основе модифицированного .ENV файла и затем скопировать новый файл в старый.

Определение ключа

• При определении файла, определение ключа не требует точного соответствия ключа основному файлу. Например, вы можете иметь физический файл с единственной компонентой STRING (20). Вы можете определить это как ключ с двумя строчными компонентами общей длиной 20. Правило таково, что типы данных должны совпадать и должен совпадать общий размер. Однако, если Clarion определение не точно совпадает с глубинным файлом, драйвер не может оптимизировать операторы APPEND() или BUILD().

• Атрибут Ключа NAME может прибавить дополнительные функциональные возможности..

KEY,NAME('MODIFIABLE=true|false')

Btrieve дает вам возможность создать ключ, который не может быть изменен, будучи создан. Чтобы использовать эту характеристику, вы можете воспользоваться атрибутом имени при ключе для установки MODIFIABLE на FALSE. По умолчанию устанавливается на TRUE.

KEY,NAME('ANYNULL')

Btrieve позволяет вам создать ключ, который не будет включать запись, если один какой-то компонент нулевой. Чтобы создать такой ключ, установите ANYNULL в имени ключа.

Например, чтобы создать ключ, который неизменен и исключить ключи, если какая-либо компонента равна нулю:

```
Key1   KEY(+pre:field,-pre:field2),NAME('ANYNULL MODIFIABLE=FALSE')
```

KEY,NAME('AUTOINCREMENT')

Оператор Clarion CREATE создает Btrieve автонумеруемый ключ.

KEY,NAME('REPEATINGDUPLICATE')

По умолчанию Btrieve версии 6 сохраняет ссылку только для первой записи в серии повторяющихся записей в ключе. Другие появления дубликата величины ключа получены следованием за списком связи, сохраненном на записи. Чтобы создать индекс, где все дублирующие записи сохранены в ключе, используйте NAME('REPEATINGDUPLICATE'). Это дает более крупные ключи, но случайный доступ к дубликатным записям быстрее. (Эта характеристика имеется только для файлов версии 6).

Сообщения об ошибках

Все драйверы баз данных TopSpeed's посылают коды завершения и сообщения, доступ к которым осуществляется при помощи функций ERRORCODE(), ERROR(), FILEERRORCODE() и FILEERROR() (См. Описание языка). Драйвер посылает код немедленно после завершения каждой операции ввода/вывода (OPEN, NEXT, GET, ADD, DELETE, и т.д.).

Совет: Функция ERRORFILE() возвращает имя файла или таблицы, которая выдала ошибку.

См. Ошибки времени выполнения (*Run Time Errors)* в описании языка, где приведено описание кодов ошибок и их соответствие сообщениям, выдаваемым функцией ERROR(). Текст сообщения, выдаваемый функцией ERROR(), может быть изменен при помощи .ENV файла (секция CLAMSG) и процедуры LOCALE. Для получения дополнительной информации см. «Интернационализация» (*Internationalization), CLAMSG* и *LOCALE* в описании языка.

В дополнение, текст сообщений FILEERROR() Btrieve драйвера также может быть изменен при помощи .ENV файла (секция CLAMSG) и процедуры LOCALE. Однако, значение CLAMSG должно быть равно величине, возвращаемой функцией FILEERRORCODE() + 1000.

FILEERRORCODE	CLAMSG	FILEERROR
1	1001	Invalid Operation
2	1002	IO Error
3	1003	File Not Open
4	1004	Key value Not
5	1005	Duplicate Key Value
6	1006	Invalid Key Number
7	1007	Different Key Number
8	1008	Position Not Set
9	1009	End of File
10	1010	Modifiable Key Value Error
11	1011	Bad File Name
12	1012	File Not Found
13	1013	Extended File Error
14	1014	Pre-Image Open Error
15	1015	Pre-Image I/O Error
16	1016	Expansion Error
17	1017	Close Error

18	1018	Disk Full
19	1019	Unrecoverable Error
20	1020	Btrieve Record Manager Inactive
21	1021	Key Buffer Too Short
22	1022	Data Buffer Length
23	1023	Position Block Length
FILEERRORCODE	**CLAMSG**	**FILEERROR**
24	1024	Page Size Error
25	1025	Create IO Error
26	1026	Number Of Keys
27	1027	Invalid Key Position
28	1028	Invalid Record Length
29	1029	Invalid Key Length
30	1030	Not A Btrieve File
31	1031	File Already Extended
32	1032	Extend I/O Error
33	1033	Unknown Error Code
34	1034	Invalid Extension Name
35	1035	Directory Error
36	1036	Transaction Error
37	1037	Transaction Active
38	1038	Transaction Control File I/O Error
39	1039	End/Abort Transaction Error
40	1040	Transaction Max Files
41	1041	Operation Not Allowed
42	1042	Incomplete Accelerated Access
43	1043	Invalid Record Address
44	1044	Null Key Path
45	1045	Inconsistent Key Flags
46	1046	Access to File Denied
47	1047	Maximum Open Files
48	1048	Invalid Alternate Sequence Defintion
49	1049	Key Type Error
50	1050	Owner Already Set
51	1051	Invalid Owner
52	1052	Error Writing Cache
53	1053	Invalid Interface
54	1054	Variable Page Error
55	1055	Autoincrement Error

56	1056	Incomplete Index
57	1057	Expanded Memory Error
58	1058	Compression Buffer Too Short
59	1059	File Already Exists
60	1060	Reject Count Reached
61	1061	Small Ex Get Buffer Error
62	1062	Invalid Get Expression
63	1063	Invalid Ext Insert Buffer
64	1064	Optimise Limit Reached
65	1065	Invalid Extractor
66	1066	RI Too Many Databases
67	1067	RI DDF Cannot Open
68	1068	RI Cascade Too Deep
69	1069	RI Cascade Error
71	1071	RI Violation
72	1072	RI Reference File Cannot Open
FILEERRORCODE	**CLAMSG**	**FILEERROR**
73	1073	RI Out of Sync
74	1074	End Changed to Abort
76	1076	RI Conflict
77	1077	Cannot Loop in Server
78	1078	Dead Lock
79	1079	Programming Error
80	1080	Conflict
81	1081	Lock Error
82	1082	Lost Position
83	1083	Read Outside Transaction
84	1084	Record in Use
85	1085	File in Use
86	1086	File Table Full
87	1087	Handle Table Full
88	1088	Incompatible Mode Error
90	1090	Redirected Device Table Full
91	1091	Server Error
92	1092	Transaction Table Full
93	1093	Incompatible Lock Type
94	1094	Permission Error
95	1095	Session No Longer Valid
96	1096	Communications Environment

97	1097	Data Message Too Small
98	1098	Internal Transaction Error
100	1100	No Cache Memory Available
101	1101	No OS Memeory Available
102	1102	No Stack Available
103	1103	Chunk Offset Too Long
104	1104	Locale Error
105	1105	Cannot Create with Bat
106	1106	Chunk cannot Get Next
107	1107	Chunk incompatible File
1001	2001	Lock Parameter Out of Range
1002	2002	Memory Allocation Error
1003	2003	Mem Parameter Too Small
1004	2004	Page Size Parameter Out of Range
1005	2005	Invalid Preimage Parameter
1006	2006	Preimage Buff Param Out of Range
1007	2007	Files Parameter Out of Range
1008	2008	Invalid Init Parameter
1009	2009	Invalid Trans Parameter
1010	2010	Error Accessing Tran Control File
1011	2011	Compression Buf Parm Out of Range
1013	2013	Task List Full
1014	2014	Stop Warning
1016	2016	Already Iniialised
2001	3001	Insufficient Memory Allocated
2002	3002	Invalid Option

FILEERRORCODE CLAMSG FILEERROR

2003	3003	No Local Access Allowed
2004	3004	SPX Not Installed
2005	3005	Incorrect SPX Version
2006	3006	No Available SPX Connection
2007	3007	Invalid Pointer Parameter

Драйвер Clarion

Спецификации

Драйвер файла Clarion совместим с файловой системой, используемой Clarion for DOS 3.1 и Clarion Professional Developer 2.1.

Ключи и индексы существуют как файлы, отдельные от файла данных. Ключи динамичны - они автоматически обновляются при изменении файлов данных. Расширение файла ключа по умолчанию - *.K##. Индексы статичны - они не подвержены автоматическому обновлению и обновляются при помощи оператора BUILD.

Драйвер сохраняет записи как записи фиксированной длины. MEMO поля хранятся в отдельном файле. По умолчанию имя файла мемо - первые восемь символов метки файла и расширение .MEM.

Файлы

C4CLAL.LIB	Статически вызываемая библиотека Windows (16-битовая)
C4CLAXL.LIB	Статически вызываемая библиотека Windows (32- битовая)
C4CLA.LIB	Библиотека экспорта Windows (16-битовая)
C4CLAX.LIB	Библиотека экспорта Windows (32-битовая)
C4CLA.DLL	Динамически вызываемая библиотека Windows (16-битовая)
C4CLAX.DLL	Динамически вызываемая библиотека Windows (32-битовая)

Совет: За исключением только некоторых ASCII форматов некоторых популярных PC систем разработки приложений баз данных, формат файла Clarion обеспечивает наиболее надежные средства хранения данных.

Типы данных

BYTE	DECIMAL
SHORT	STRING (255 байт максимум)
LONG	MEMO
REAL	GROUP

Характеристики файла/максимумы

Размер файла:	Ограничено местом на диске
Записей на файл:	4,294,967,295
Размер записи:	65,520 байт
Размер поля:	65,520 байт
Полей на запись:	65,520
Ключей/Индексов на файл:	251
Размер ключа:	245 байт
Полей мемо на файл:	1
Размер поля мемо:	65520 байт
Открытие файла данных::	Зависит от операционной системы

Строки драйвера

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. См. Общие характеристики драйверов - Строки драйвера.

Замечание: Некоторые строки драйвера не действуют после того, как файл открыт, поэтому никакая функция SEND выполняется для этих строк. Однако в тоже время функция SEND возвращает значения переключателей для всех строк драйвера

Драйвер Clarion поддерживает следующие строки драйвера:

DELETED

```
SEND(file, 'DELETED')
```

Для использования только с командой SEND, когда включен (ON) / IGNORESTATUS. Сообщает состояние загруженной записи. Если удалена, возвращаемая строка символов -"ON", если нет - "OFF".

HELD

```
SEND(file, 'HELD')
```

Для использования только с командой SEND, когда включен (ON) / IGNORESTATUS. Сообщает состояние загруженной записи. Если удержана,

возвращенная строка символов -"ON", если нет - "OFF".

IGNORECORRUPTIONS

```
DRIVER('Clarion', '/IGNORECORRUPTIONS = on | off' )
[ Status» = ] SEND(file, 'IGNORECORRUPTIONS [ = on | off ]' )
```

Драйвер считает заголовок ключа файла разрушенным в том случае, когда количество удаленных записей в заголовке файла отлично от того, что указано в заголовке ключа. По умолчанию (IGNORECORRUPTIONS=OFF), когда предпринимается попытка открыть такой поврежденный файл, драйвер посылает сообщение об ошибке ERRORCODE()=46. Для того, чтобы драйвер игнорировал эти нарушения (иногда безопасные) в ключе установите IGNORECORRUPTIONS=ON.

Функция SEND возвращает текущее значение IGNORECORRUPTIONS(ON или OFF) в форме STRING(3).

IGNORESTATUS

```
DRIVER('Clarion', '/IGNORESTATUS = on | off' )
[ Status» = ] SEND(file, 'IGNORESTATUS [ = on | off ]' )
```

Когда IGNORESTATUS = ON, драйвер *не* пропускает удаленные записи при доступе к файлу с помощью GET(),NEXT() и PREVIOUS() в порядке файла. Эта позиция также позволяет поместить PUT() на удаленную или удержанную запись. /IGNORESTATUS требует открыть файл в эксклюзивном режиме.

Функция SEND возвращает текущее значение IGNORESTATUS(ON или OFF) в форме STRING(3).

MAINTAINHEADERTIME

```
DRIVER('Clarion', '/MAINTAINHEADERTIME = on | off' )
[ Status» = ] SEND(file, 'MAINTAINHEADERTIME [ = on | off ]' )
```

Когда MAINTAINHEADERTIME =ON, Драйвер поддерживает в заголовке файла отметки времени (последнего изменения) при любых обстоятельствах. Когда MAINTAINHEADERTIME =OFF (по умолчанию), драйвер для увеличения производительности иногда игнорирует эти отметки времени.

Функция SEND возвращает текущее значение MAINTAINHEADERTIME(ON или OFF) в форме STRING(3).

RECOVER

SEND(file, 'RECOVER = n')

Строка RECOVER (восстановление), когда *n* больше 0, разблокирует (UNLOCK) файл данных или освобождает (RELEASE) удерживаемую запись и откатывает незавершенную транзакция для восстановления файла в случае разрушения системы. См. также «Обработка транзакций для файлов Clarion».

n представляет число или секунды ожидания перед вызовом процесса восстановления. Когда *n* равно нулю, процесс восстановления отключен.

Существуют два способа использования RECOVER:
SEND(file,RECOVER=n)
OPEN(file)

Этим способом освобождается файл, заблокированный в момент аварийной остановки машины. Это также откатывает незавершенную на момент аварии транзакцию.
SEND(file,RECOVER=n)
GET or NEXT or PREVIOUS

Этим способом удаляется флаг блокировки записей заблокированных в момент аварийной остановки машины. Далее приводится фрагмент кода удаляющий флаги блокировки со всех записей файла:

```
OPEN(file)                !Убедитесь что никто больше не использует этот файл
SEND(file,'IGNORESTATUS=ON')
SET(file)
LOOP
        NEXT(file)
        IF ERRORCODE() THEN BREAK.
        IF SEND(file,'HELD') = 'ON' THEN
                                SEND(file,'RECOVER=1')
                                REGET(file,POSITION(file))
.       .
```

RECOVER не может быть использовано в качестве строки драйвера - вы можете его использовать только с функций SEND. Функция SEND возвращает

пустую строку.

Атрибуты файла	Поддержка
CREATE	ДА
DRIVER [*драйвер,строка драйвера*])	ДА
NAME	ДА
ENCRYPT	ДА
OWNER(*Пароль*)	ДА[1]
RECLAIM	ДА
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	ДА[8]
EXTERNAL(*член*)	ДА
DLL([*признак*])	ДА
OEM	ДА

Структуры файла	Поддержка
INDEX	ДА
KEY	ДА
MEMO	ДА
BLOB	НЕТ
RECORD	ДА

Атрибуты индекса, ключа, мемо	Поддержка
BINARY	ДА[9]
DUP	ДА
NOCASE	ДА
OPT	ДА
PRIMARY	ДА
NAME	ДА
Сортировка по возрастанию	ДА
Сортировка по убыванию	НЕТ
Смешанная сортировка	НЕТ

Атрибуты поля	Поддержка
DIM	ДА
OVER	ДА
NAME	ДА

Файловые процедуры	Поддержка
BOF(*файл*)	ДА[2]
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	ДА
BUILD(*ключ*)	ДА
BUILD(*индекс*)	ДА
BUILD(*индекс, компонент*)	ДА
BUILD(*индекс, комп-т, фильтр*)	НЕТ
BYTES(*файл*)	ДА
CLOSE(*файл*)	ДА
COPY(*файл, новый файл*)	ДА
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА[2]
FLUSH(*файл*)	ДА
LOCK(*файл*)	ДА
NAME(*метка*)	ДА
OPEN(*файл, режим доступа*)	ДА
PACK(*файл*)	ДА
POINTER(*файл*)	ДА[3]
POINTER(*ключ*)	ДА[3]
POSITION(*файл*)	ДА[3]
POSITION(*ключ*)	ДА[4]
RECORDS(*файл*)	ДА
RECORDS(*ключ*)	ДА
REMOVE(*файл*)	ДА
RENAME(*файл, новый файл*)	ДА
SEND(*файл, Сообщение*)	ДА
SHARE(*файл, режим доступа*)	ДА
STATUS(*файл*)	ДА
STREAM(*файл*)	ДА
UNLOCK(*файл*)	ДА

Доступ к записям	Поддержка
ADD(*файл*)	ДА[10]
ADD(*файл, длина*)	НЕТ
APPEND(*файл*)	ДА

APPEND(*файл, длина*)	НЕТ		
DELETE(*файл*)	ДА		
GET(*файл, ключ*)		ДА	
GET(*файл, указатель файла*)		ДА	
GET(*файл, указатель файла, длина*)		НЕТ	
GET(*ключ, указатель ключа*)		ДА	
HOLD(*файл*)		ДА	
NEXT(*файл*)		ДА	
NOMEMO(*файл*)	ДА		
PREVIOUS(*файл*)		ДА	
PUT(*файл*)		ДА[10]	
PUT(*файл, указатель файла*)		ДА	
PUT(*файл, указатель файла, длина*)		НЕТ	
RELEASE(*файл*)		ДА	
REGET(*файл,строка*)		ДА	
REGET(*ключ,строка*)		ДА	
RESET(*файл,строка*)			ДА
RESET(*ключ,строка*)			ДА
SET(*файл*)		ДА	
SET(*файл, ключ*)		ДА	
SET(*файл, указатель файла*)		ДА	
SET(*ключ*)		ДА	
SET(*ключ, ключ*)		ДА	
SET(*ключ, указатель ключа*)			ДА
SET(*ключ, ключ, указатель файла*)			ДА
SKIP(*файл, число*)		ДА	
WATCH(*файл*)		ДА	

Обработка транзакций	Поддержка[5]	
LOGOUT(*вр. ож., файл, ..., файл*)		ДА[6,7]
COMMIT	ДА	
ROLLBACK		ДА

Обработка пустых записей	Поддержка
NULL(*поле*)	НЕТ

SETNULL(*поле*)	НЕТ
SETNONNULL(*поле*)	НЕТ

Примечания.

1 Мы рекомендуем использовать длинные и содержащие специальные символы значения пароля, так как именно таким способом обеспечивается большая эффективность защиты. Например, пароль подобный «;%?? $$^» гораздо сложнее разгадать, чем пароль типа «SALARY.»

Совет: Для определения переменной, содержащей актуальное значение пароля в поле Owner Name в диалоге «Свойства файла» (File Properties dialog), перед именем переменной наберите восклицательный знак '!'. Например: !MyPassword.

2 Эти функции поддерживаются, но не рекомендуются из-за отсутствия поддержки в других файловых системах. NEXT и PREVIOUS посылают Error 33, если вы попытаетесь читать за концом файла.

3 Функция POINTER() возвращает номер записи

4 Функция POSITION(файл) возвращает STRING(4). POSITION(ключ) возвращает строку длиной, равной размеру поля ключа + 4 байта.

5 Для поддержки обработки транзакций переключатель RECOVER должен быть включен в начале работы вашей программы. См. «Строки драйвера».

6 LOGOUT блокирует файл. См. Prop:Logout в описании языка.

7 Невозможно включить в одну транзакцию одновременно некоторый файл и его алиас.

8 Файлы с атрибутом THREAD требуют дополнительного блока управления файлом для каждого процесса, который требует доступа к этому файлу.

9 OEM преобразование неприменимо к BINARY MEMOs. Драйвер подразумевает BINARY MEMO заполненным нулями или пробелами.

10 Ранние версии 16-битовых. программ Clarion некорректно выполняли операции записи при работе с системными буферами операционной системы. Эти проблемы решены, начиная с Clarion 2.003 и в более поздних

версиях, запись стала несколько медленнее, но зато надежнее. Для ускорения работы используйте STREAM и FLUSH.

Прочее

Обработка транзакций для файлов Clarion

Когда вы выполняете оператор LOGOUT, драйвер Clarion создает транзакционный файл с именем программы и расширением TR$. По умолчанию этот файл создается в том же каталоге, в котором расположена программа. Для создания транзакционного файла в другом месте добавьте в файл WIN.INI секцию CWC21:

```
[CWC21]
CLATMP=PATH..
```

Тут PATH это каталог, видимый для всех пользователей.

Оператор PUTINI('CWC21','CLATMP',path) создает правильную секцию INI файла.

В процессе выполнения транзакции для каждого файла данных, изменяющегося в данной транзакции, создается LOG-файл (файл - протокол). Эти файлы - протоколы всегда размещаются там же, где находятся файлы данных. Если во время выполнения транзакции происходит аварийное завершение работы системы (крах), никто из пользователей не сможет получить доступ к этим файлам до тех пор, пока не будет выполнена процедура восстановления файлов. См. команду SEND(file, 'RECOVER = n')

Метки полей

Драйвер Clarion поддерживает только полностью квалифицированные имена полей (метка+префикс), длина которых не превышает 16-ти символов. Если длина превышает предельное значение, драйвер Clarion усекает имя и оставляет первые 16 символов. Дублирование имен полей внутри заголовка файла может вызвать проблемы при работе с файлами DOS версий Clarion начиная с 2.1 и более ранних версий. Это может стать проблемой в том случае, если вы импортируете определение файла из Clarion файла (*.DAT) и затем компилируете Clarion приложение, основанное на этих импортированных определениях файлов содержащий неуникальные имена.

Вы можете избежать проблемы дублирования имен полей, используя NAME-атрибут (поле **External Name**(внешние имена**)** в диалоге **Field Properties**(свойства поля) словаря данных) назначения уникальных имен для тех полей, у которых первые шестнадцать символов совпадают. Благодаря уникальным именам в NAME - атрибуте ваше приложение может ссылаться к полям по меткам, и драйвер Clarion будет использовать уникальные NAME - атрибуты для разрешения конфликтов.

Драйвер Clipper

Спецификации

Драйвер файла Clipper совместим с Clipper Summer '87 и Clipper 5.0. Расширение файла данных по умолчанию *.DBF.

Ключи и индексы существуют как файлы, отдельные от файла данных. Ключи динамические - они автоматически обновляются при изменении файла данных. Индексы статические - они не обновляются автоматически, а вместо этого требуют для обновления выполнения оператора BUILD. Расширение индексного файла по умолчанию *.NTX.

Драйвер сохраняет записи, как записи фиксированной длины. Поля мемо хранятся в отдельном файле. Имя мемо файла - первые восемь символов метки файла с расширением .DBT.

C4CPLL.LIB	Статически вызываемая библиотека Windows (16-битовая)
C4CPLXL.LIB	Статически вызываемая библиотека Windows (32- битовая)
C4CPL.LIB	Библиотека экспорта Windows (16-битовая)
C4CPLX.LIB	Библиотека экспорта Windows (32-битовая)
C4CPL.DLL	Динамически вызываемая библиотека Windows (16-битовая)
C4CPLX.DLL	Динамически вызываемая библиотека Windows (32-битовая)

Совет: Будучи популярной системой разработки приложений баз данных xBase, Clipper обеспечивает обычный формат файла для многих инсталлированных деловых приложений и их файлов данных. Используйте драйвер Clipper, чтобы получить доступ к этим файлам в их родном формате.

Поддерживаемые типы данных

Формат файла xBase сохраняет все данные в виде строк символов ASCII. Вы можете установить типы строк символов STRING с помощью символьных шаблонов для каждого поля, либо установить типы данных Clarion, которые драйвер преобразует автоматически.

Тип данных Clipper	*Тип данных Clarion*	*символьный шаблон*
Date	DATE	STRING(@D12)
*Numeric	REAL	STRING(@N-_p.d)
*Logical	BYTE	STRING(1)

Character	STRING	STRING
*Memo	MEMO	MEMO

Если ваше приложение читает и записывает в существующие файлы, достаточен будет символьный шаблон. Однако, если ваше приложение *создает* Clipper файл, вам может потребоваться дополнительная информация для этих типов данных Clipper:

• Чтобы создать цифровое поле в словаре данных, выберите тип данных REAL (действительный). В атрибуте Внешнее имя на закладке Атрибуты установите *'NumericFieldName*=N(*Precision,DecimalPlaces)'*, где *NumericFieldName* - имя поля, *Precision* - точность поля, а *DecimalPlaces* - число десятичных мест. С данными типа REAL вы не можете получить доступ к полям символа или мест в определении поля, вам следует назначить новые атрибуты с помощью выражения во Внешнем поле имени на закладке Атрибуты.

Например, если вы хотите создать поле, называемое Number (номер) с девятью значащими цифрами и двумя десятичными знаками, введите 'Number=N(9,2) во внешнем поле имени на закладе Атрибуты свойств поля в словаре данных. Если вы кодируете вручную присущие Clarion типы данных, добавьте атрибут NAME, используя тот же самый синтаксис. Если вы кодируете вручную символьный шаблон, STRING@N-_9.2),NAME(*'Number'),*где *Number* - имя поля.

• Чтобы создать логическое поле, используя словарь данных, выберите тип данных BYTE. Специальных шагов нет; однако, смотрите в разных разделах советы о чтении данных из поля.
Если вы вручную кодируете символьный шаблон, добавьте атрибут NAME: STRING(1),NAME('*LogFld*=L').

• Чтобы создать поле данных, используя словарь данных, выберите тип данных DATE, а не LONG, который вы обычно используете для форматов файлов TopSpeed или Clarion.

• Объявления поля MEMO требуют указателя поля в файловой структуре записи. Объявите указатель поля как STRING(10) или LONG. Это поле будет сохранено в файле .DBF, содержащем смещение мемо в файле .DBT. Объявление MEMO должно иметь атрибут NAME(), называющий указатель поля. Пример объявления поля:

File	FILE,DRIVER('Clipper')
Memo1	MEMO(200),NAME('Notes')
Memo2	MEMO(200),NAME('Text')
Rec	RECORD

```
Mem1Ptr                 LONG,NAME('Notes')
Mem2Ptr                 STRING(10),NAME('Text')
            END
END
```

Совет: Когда это возможно, пользуйтесь утилитой импорта файла (File Import Utility) в редакторе словаря Clarion for Windows для того, чтобы назначить свои файлы.

Характеристики файла/максимумы

Размер файла:	2,000,000,000 байт
Записей на файл:	1,000,000,000 байт
Размер записи:	4,000 байт (Clipper '87) 8,192 байт (Clipper 5.0)
Размер поля	
Символы:	254 байт (Clipper '87) 2048 байт (Clipper 5.0)
Данные:	8 байт
Логическое:	1 байт
Цифровое:	20 байт включая десятичную точку
Мемо:	65,520 байт (см. замечание)
Полей на запись:	1024
Ключей/индексов на файл:	неограниченно
Размер ключа	
Символ:	100 байт
Цифры, данные:	8 байт
Мемо полей на файл:	в зависимости от имеющейся памяти
Открывание файла:	в зависимости от операционной системы

Строки драйвера

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. См. Общие характеристики драйверов - Строки драйвера.

Замечание: Некоторые строки драйвера не действуют после того, как файл открыт, поэтому никакая функция SEND выполняется для этих строк. Однако в

тоже время функция SEND возвращает значения переключателей для всех строк драйвера

Драйвер Clipper поддерживает следующие строки драйвера:

BUFFERS

```
DRIVER('CLIPPER', '/BUFFERS = n' )
[ Status» = ] SEND(file, 'BUFFERS [ = n ]' )
```

Устанавливает размер буфера, используемого для чтения и записи в файл, где размер буфера равен (n*512). Используйте /BUFFERS строку драйвера для увеличения размера буфера, если доступ к файлу слишком медленен. Максимальный размер буфера равен 65,5054 байт для 16-битовых приложений и 4,294,967,264 для 32-битовых. Функция SEND возвращает размер буфера в байтах.

Совет: По умолчанию имеется три буфера по 1024 байта каждый. Увеличение числа буферов не будет увеличивать производительность программы, если файлы разделяются между несколькими пользователями.

RECOVER

```
DRIVER('CLIPPER', '/RECOVER' )
[ Status» = ] SEND(file, 'RECOVER' )
```

Эквивалентно команде Xbase RECALL, которая восстанавливает записи, маркированные для удаления. При использовании драйвера Clipper оператор DELETE отмечает флажком запись как "неактивную". Драйвер не удаляет запись до тех пор, пока не выполнены команды PACK (упаковать) или BUILD (построить).

/RECOVER оценивается каждый раз при открытии вами файла, если вы добавите строку символов драйвера к словарю данных. Когда драйвер восстанавливает записи, ранее отлакированные для удаления, вы должны вручную перестроить ключи и индексы с помощью оператора BUILD.

IGNORESTATUS

```
DRIVER('CLIPPER', '/IGNORESTATUS = on | off ' )
[ Status» = ] SEND(file, 'IGNORESTATUS [ on | off ]' )
```

Когда установка *ON*, драйвер *не* перескакивает через удаленные записи во время выполнения операторов GET, NEXT и PREVIOUS при доступе к файлу, открытому в физической последовательности. Это также делает доступным выполнение оператора PUT для удаленной или задержанной записи. /

IGNORESTATUS требует открытия файла в эксклюзивном режиме. Функция SEND возвращает текущее значение IGNORESTATUS(ON или OFF) в форме STRING(3).

DELETED

[Status» =] SEND(file, 'DELETED')

Для использования только с командой SEND, когда включен IGNORESTATUS. Сообщает состояние текущей записи. Если она удалена, возвращаемая строка символов - "ON", если нет - "OFF".

Атрибуты файла	**Поддержка**
CREATE	ДА[1]
DRIVER [*драйвер,строка драйвера*])	ДА
NAME	ДА
ENCRYPT	НЕТ
OWNER(*Пароль)*	НЕТ
RECLAIM	НЕТ[2]
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	Y[16]
EXTERNAL(*член*)	ДА
DLL([*признак*])	ДА
OEM	ДА

Структуры файла	Поддержка
INDEX	ДА
KEY	ДА
MEMO	ДА[3]
BLOB	НЕТ
RECORD	ДА

Атрибуты индекса, ключа, мемо	Поддержка
BINARY	НЕТ
DUP	ДА[4]
NOCASE	ДА
OPT	НЕТ
PRIMARY	ДА

NAME	ДА[3]
Сортировка по возрастанию	ДА
Сортировка по убыванию	ДА
Смешанная сортировка	ДА

Атрибуты поля	Поддержка
DIM	НЕТ
OVER	ДА
NAME	ДА[1]

Файловые процедуры	Поддержка
BOF(*файл*)	ДА[11]
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	ДА
BUILD(*ключ*)	ДА
BUILD(*индекс*)	ДА[3]
BUILD(*индекс*, *компонент*)	ДА[5]
BUILD(*индекс*, *компонент*, *фильтр*)	НЕТ
BYTES(*файл*)	НЕТ
CLOSE(*файл*)	ДА
COPY(*файл, новый файл*)	ДА[6]
CREATE(*файл*)	ДА1
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА[11]
FLUSH(*файл*)	ДА
LOCK(*файл*)	ДА
NAME(*метка*)	ДА
OPEN(*файл, режим доступа*)	ДА[7]
PACK(*файл*)	ДА
POINTER(*файл*)	ДА[12]
POINTER(*ключ*)	ДА[12]
POSITION(*файл*)	ДА[13]
POSITION(*ключ*)	ДА[13]
RECORDS(*файл*)	ДА[14]
RECORDS(*ключ*)	ДА[14]
REMOVE(*файл*)	ДА
RENAME(*файл, новый файл*)	ДА[8]

SEND(*файл, Сообщение*)	ДА
SHARE(*файл, режим доступа*)	ДА[7]
STATUS(*файл*)	ДА
STREAM(*файл*)	ДА
UNLOCK(*файл*)	ДА

Доступ к записям	Поддержка
ADD(*файл*)	ДА[9]
ADD(*файл, длина*)	НЕТ
APPEND(*файл*)	ДА[9]
APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	ДА[6]
GET(*файл, ключ)*	ДА
GET(*файл, указатель файла*)	ДА
GET(*файл, указатель файла, длина*)	НЕТ
GET(*ключ, указатель ключа*)	ДА
HOLD(*файл*)	ДА[10]
NEXT(*файл*)	ДА
NOMEMO(*файл*)	ДА
PREVIOUS(*файл*)	ДА
PUT(*файл*)	ДА
PUT(*файл, указатель файла*)	ДА
PUT(*файл, указатель файла, длина*)	НЕТ
RELEASE(*файл*)	ДА
REGET(*файл,строка*)	ДА
REGET(*ключ,строка*)	ДА
RESET(*файл,строка*)	ДА
RESET(*ключ,строка*)	ДА
SET(*файл*)	ДА
SET(*файл, ключ*)	ДА
SET(*файл, указатель файла*)	ДА[8]
SET(*ключ*)	ДА
SET(*ключ, ключ*)	ДА
SET(*ключ, указатель ключа*)	ДА
SET(*ключ, ключ, указатель файла*)	ДА
SKIP(*файл, число*)	ДА
WATCH(*файл*)	ДА

Обработка транзакций	Поддержка[15]
LOGOUT(*вр. ож., файл, ..., файл*)	НЕТ
COMMIT	НЕТ
ROLLBACK	НЕТ

Обработка пустых записей	Поддержка
NULL(*поле*)	НЕТ
SETNULL(*поле*)	НЕТ
SETNONNULL(*поле*)	НЕТ

1 Если ваше приложение создает файл Clipper, вам может понадобится дополнительная NAME информация для Clipper типов данных. Для Clipper числовых полей используются данные типа REAL. Затем в NAME-атрибуте (поле **External Name** (внешнее имя) на закладке **Attributes** в диалоге **Field Properties** (свойства поля) укажите

'*NumericFieldName*=N(*Precision*,*DecimalPlaces*)' где *NumericFieldName* - имя поля, *Precision* - число значащих цифр и *DecimalPlaces* - количество цифр после десятичной запятой. Для дополнительной информации см. выше ТИПЫ ДАННЫХ.

Для Clipper - полей, содержащих логические значения, используйте тип данных Clarion BYTE. Для дополнительной информации см. выше ТИПЫ ДАННЫХ. См. также раздел «Разное» для советов по чтению данных из поля.

Для Clipper - полей, содержащих календарные даты, используйте тип данных Clarion DATE. Для дополнительной информации см. выше ТИПЫ ДАННЫХ.

2 Когда драйвер удаляет запись из базы данных Clipper, запись физически не удаляется, а вместо этого драйвер маркирует ее, как неактивную. Поля мемо не удаляются физически из мемо файла, однако они не могут быть найдены, если они относятся к неактивной записи. Записи ключа удаляются из индексного файла. Чтобы удалить записи и мемо файлы навсегда, выполните оператор PACK(file).

Совет: Для программистов, знакомых с Clipper, этот драйвер обрабатывает удаленные записи в соответствии с тем, как Clipper обрабатывает их после того, как дана команда SET DELETED ON (установите удаленное). Записи, отмеченные для удаления, игнорируются исполняемыми программными операторами, но остаются в файле данных.

3 Объявление MEMO поля требует поля-указателя. Объявите поле-указатель

как STRING(10) или как LONG. Это поле в .DBF - файле будет хранить смещение MEMO в .DBT - файле. Объявление MEMO должно иметь NAME(), атрибут именующий поле-указатель. Для дополнительной информации см. выше ТИПЫ ДАННЫХ.

4 В Clipper нормальным является ввод многих записей с дубликатами уникальных ключевых компонент. Однако только первая из этих записей индексирована. Таким образом обработка в ключевом порядке показывает только эту первую запись. Если вы удаляете запись, затем вводите новую запись с той же самой величиной ключа, ключевой файл продолжает указывать на удаленную запись, а не на новую запись. В этой ситуации файловый драйвер Clipper изменяет ключевой файл, чтобы указать на активную запись, а не на удаленную запись. Это означает, что если вы используете программу Clipper для удаления уникальной записи, а затем вставите дубликат этой записи, новая запись невидима при обработке в ключевом порядке до тех пор, пока не выполнена упаковка. Если вы выполняете тот же самый процесс в программе Clarion, новая запись оказывается видимой при обработке в ключевом порядке.

5 При построении динамических индексов *компоненты* могут принять одну из двух следующих форм:

BUILD(DynNdx, '+Pre:FLD1, -Pre:FLD2')

Эта форма устанавливает имена полей, на которых строится индекс. Имена полей должны появиться в атрибуте поля NAME, если он используется, или это должны быть имена меток. Для совместимости с Clarion может быть использован префикс, но он игнорируется.

BUILD(DynNdx, 'T[Expression]')

Эта форма устанавливает тип и выражение, используемые для построения индекса, о чем смотрите ниже - в разделе "Разное" - *Назначение ключа*.

6 Команда COPY() копирует файлы данных и мемо файлы, используя *newfile (*новый файл*)*, при помощи которого вы может установить новое имя файла или каталога. Файлы ключей или индексов копируются, если *newfile* является характеристикой подкаталога. Для копирования индексного файла в новый файл используйте специальную форму команды копирования:

COPY(file, '<index>|< newfile >')

Если используется ссылка на недействительный индекс, этот оператор выдает

сообщение *File Not Found (файл не найден)*. Команда COPY предполагает по усмотрению расширение ".NTX" для имен файлов источника и целевого файла, если ни одно из них специально не определено. Если вы требуете имя файла без расширения, закончите имя точкой.

Дана структура файла:

```
Clar2  FILE,CREATE,DRIVER('Clipper')
NumKey  KEY(Num),DUP
StrKey          KEY(Str1)
StrKey2         KEY(Str2)
AMemo           MEMO(100),NAME('mem')
Record          RECORD
Num        STRING(@n-_9.2)
STR1       STRING(2)
STR2       STRING(2)
  Mem      STRING(10)
```

Следующие команды копируют это определение файла в A:

```
COPY(Clar2,'A:\CLAR2')
COPY(Clar2,'StrKey|A:\STRKEY')
COPY(Clar2,'StrKey2|A:\STRKEY2')
COPY(Clar2,'NumKey|A:\NUMKEY')
```

После этих вызовов на накопителе A будут существовать следующие файлы: CLAR2.DBE,CLAR2.DBT,STRKEY.NTX,STRLEY2.NTX и NUMKEY.NTX.

7 Вам не нужен SHARE (или VSHARE) в любой среде (NOVEL, например), которая поддерживает блокировку собственными средствами.

8 Команда RENAME копирует файлы данных и файлы мемо с помощью *newfile*, определяющем новое имя файла или путь. Файлы ключей и индексов должны быть переименованы с помощью того же синтаксиса, что и команда COPY (см. выше).

9 Оператор ADD проверяет на дублирование ключи перед модификацией файла данных или связанных с ним ключевых KEY файлов. В результате оператор ADD выполняется медленнее, чем APPEND, который не выполняет никаких проверок и не обновляет ключи KEY. При добавлении больших количеств данных к базе данных, используйте APPEND...BUILD вместо ADD.

10 Clipper выполняет блокирование записи путем блокирования всей записи в файле данных. Это предотвращает доступ для чтения для других процессов.

Следовательно, мы рекомендуем уменьшить до предела количество времени, на которое удерживается запись.

11 Хотя драйвер поддерживает эти функции, мы не рекомендуем их использовать. Эти функции для своего выполнения требуют физического доступа к файлам и значительных ресурсов. Вместо этого проверьте величину, возвращаемую функцией ERRORCODE() после каждого последовательного доступа. NEXT() или PREVIOUS() посылают код ошибки 33 (запись недоступна), если сделана попытка получить доступ к записи за концом или началом файла.

12 Нет различия между указателями файла и указателями ключа; и тот и другой содержат ту же самую величину для любой данной записи.

13 Функция POSITION(file) возвращает STRING(12). Функция POSITION(key) возвращает строку длиной, равной размеру ключа плюс 4 байта

14 В рамках Clipper функция RECORDS() сообщает одно и то же число записей для файла данных и его ключей и индексов. Обычно не бывает различия в числе записей, если INDEX не устарел. Так как оператор DELETE не удаляет записи физически, *число записей, о которых сообщает функция* RECORDS()*, включает неактивные записи*. Будьте осторожны при использовании этой функции.

15 Вы должны убрать отметку в поле **Enclose RI code in transaction frame** на закладке **File Control** tab в диалоге **Global Properties** когда используете Clipper драйвер для приложений, генерируемых с использованием шаблонов.

16 Файлы с атрибутом THREAD требуют дополнительного блока управления для каждого процесса, получающего доступ к данному файлу.

Разное

Булева оценка

¤ Clipper позволяет логическому полю принять одну из девяти возможных величин (y,Y,n,N,t,T,f,F или символ пробела). Символ пробела - это ни истина и ни ложь. При использовании логического поля из заранее существующей базы данных в логическом выражении учтите все эти возможности. Помните, что когда поле STRING используется как выражение, оно истинно, если содержит какие-либо данные, и оно ложно, если равно нулю или пусто. Следовательно, чтобы оценить

истинность логического поля, выражение должно быть истина, если поле содержит какую-либо из "истинных" характеристик (T,t,Y или y). Например, если логическое поле было использовано, чтобы установить продукт как налогооблагаемый или необлагаемый, выражение для оценки его истинности будет:
(*If Condition):*

```
Taxable='T' OR Taxable='t' OR Txable='Y' OR
Taxable='y'
```

Большие MEMO

¤ Clarion for Windows поддерживает поля MEMO размером до 64K. Если у вас существует файл, включающий поле мемо, намного большее чем 64K, вы можете использовать файл, но не модифицировать большие MEMO.

¤ Вы можете определить, когда ваше приложение встретит большое MEMO путем определения, что переменная указателя мемо не пуста, но поле мемо при этом выглядит пустым. Выдается код ошибки 47 (неправильное объявление записи). Если вы попытаетесь обновить такую запись, любая модификация MEMO поля будет игнорироваться.

Длинные имена поля

¤ Clipper поддерживает максимум 10 символов в имени поля. Если вам нужно больше, используйте External Name (внешнее имя) с 10 или менее символами.

Последовательность сортировки

¤ Драйвер Clipper Clarion for Windows поддерживает международные порядки сортировки, однако, для соблюдения совместимости с международным порядком сортировки Clipper, удалите строку CLADIGRAPH= из файла ..\CLARION4\BIN\CLARION4.ENV.

Определение ключа

¤ Clipper поддерживает использование выражений для определения ключей. В редакторе словаря вы можете поместить выражение во внешнее имя поля в диалоговом окне Key Properties(свойства ключа). Общий формат внешнего имени:

```
'FileName=T[Expression]'
```

Где *FileName* представляет имя индексного файла (который может содержать путь и расширение файла), а *T* представляет тип индекса. Действительными типами являются: C = символ, D = данные, N = цифра. Если типом является D или N, тогда *Expression (выражение)* может назвать только одно поле.

Выражения в виде строк символов могут использовать оператор "+" для конкатенации многих строк символов аргументов. Цифровые выражения

используют операторы '+' или '-' в их основном смысле. Максимальная длина Clipper выражения - 250 символов.

Выражение может относиться к множественным полям в записи и содержать функции xBase. Квадратные скобки должны ставиться вокруг выражения. Текущие поддерживаемые функции даются ниже. Если драйвер встречает неподдерживаемую xBase функцию в заранее существовавшем файле, он посылает код ошибки 76 "Недействительная строка символов индекса", когда файл открыт для ключей и статических индексов.

Поддерживаемые xBase функции определения ключа

ALLTRIN(string) Удаляет начальные и конечные пробелы.

CTOD(string) Преобразует ключ в виде строки символов в дату. Формат строки *string* mm/dd/yy; результат имеет форму 'yyyymmdd'. Элемент yyyy по умолчанию означает двадцатое столетие. Недействительные данные дают в результате пустой ключ.

DELETED() Возвращает TRUE, если запись удалена.

DESCEND(string|date|numeric) Инвертирует аргумент и создает нисходящие Clipper индексы.

DTOC(date) Преобразует ключ данных в строку символов формата 'mm/dd/yy'.

DTOS(date) Преобразует ключ данных в строку символов формата 'yyyymmdd'.

FIXED(float) Преобразует ключ из представления с плавающей точкой в представление с фиксированной точкой.

FLOAT(numeric) Преобразует ключ из представления с фиксированной точкой в представление с плавающей точкой.

IIF(bool, val1, val2) Возвращает величину val1 если первый параметр TRUE, иначе возвращает величину val2.

LEFT(string, n) Возвращает самые левые *n* символов строки символов ключа в виде строки символов длиной *n*.

LOWER (string, n) Преобразует строку символов ключа в нижний регистр.

LTRIM (string) Удаляет пробелы с левой стороны от строки символов.

RECNO( ) Возвращает номер текущей записи.

RIGHT(string, n) Возвращает самые правые *n* символов строки символов ключа в виде строки символов длиной *n*.

RTRIM(string) Удаляет пробелы с правой стороны строки символов.

STR(numeric [,length[, decimal places]]) Преобразует цифровые данные в строку

символов. Длина строки символов и число десятичных знаков после запятой - параметры необязательные. По умолчанию длина строки символов 10, а число знаков после десятичной точки 0.

SUBSTR(string,offset,n) Возвращает подстроку символов ключа в виде строки символов *string* , начиная с позиции *offset* и длиной в *n* символов.

TRIM (string) Удаляет пробелы с правой стороны от строки символов (идентично RTRIM).

UPPER(string) Преобразует строку символов ключа в верхний регистр.

VAL(string) Преобразует строку символов ключа в цифру.

Сообщения об ошибках

Все драйверы баз данных TopSpeed's посылают коды завершения и сообщения, доступ к которым осуществляется при помощи функций ERRORCODE(), ERROR(), FILEERRORCODE() и FILEERROR() (См. Описание языка). Драйвер посылает код немедленно после завершения каждой операции ввода/вывода (OPEN, NEXT, GET, ADD, DELETE, и т.д.).

Совет: Функция ERRORFILE() возвращает имя файла или таблицы, которая выдала ошибку.

См. Ошибки времени выполнения (*Run Time Errors)* в описании языка, где приведено описание кодов ошибок и их соответствие сообщениям, выдаваемым функцией ERROR(). Текст сообщения, выдаваемый функцией ERROR(), может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Для получения дополнительной информации см. «Интернационализация» (*Internationalization), CLAMSG* и *LOCALE* в описании языка.

В дополнение, текст сообщений FILEERROR() Clipper драйвера также может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Однако значение CLAMSG должно быть равно величине, возвращаемой функцией FILEERRORCODE() + 4000.

FILEERRORCODE	CLAMSG	FILEERROR
1	4001	file write failure(Ошибка чтения файла)

2	4002	file read failure(Ошибка записи в файл)
3	4003	memory allocation error(Ошибка распределения памяти)
4	4004	file pointer reposition failed(Ошибка позиционирования)
5	4005	file not found(файл не найден)
6	4006	file corrupted(файл разрушен)
7	4007	bad user specified key expression (ошибка спецификации ключа)
8	4008	no handles available
9	4009	no index pages loaded
10	4010	index page was not loaded
11	4011	file close failure (ошибка закрытия файла)
12	4012	invalid command (неправильная команда)
13	4013	invalid handle number
14	4014	invalid filename (неправильное имя файла)
15	4015	invalid date (неправильная дата)
16	4016	invalid time (неправильное время)
17	4017	file not in memo format(не MEMO файл)
18	4018	invalid Clipper version (неправильная версия Clipper)
19	4019	file header length error (неправильная д лина заголовка файла)
20	4020	last file change date in error (неправильная дата последней коррекции файла)
21	4021	parameter address NULL (адрес параметра пустой)
22	4022	invalid key type (неправильный тип ключа)
23	4023	invalid key length (неправильная длина ключа)
24	4024	item length incorrect (неправильная длина элемента)
25	4025	invalid root page (неправильная корневая страница)
26	4026	bad maximum number of keys per page (неправильное число ключей на странице)
27	4027	invalid number of fields (неправильное число полей)
28	4028	field name invalid (неправильное имя поля)
29	4029	bad field length (неправильная длина поля)
30	4030	decimal places parameter invalid (неправильная позиция десятичной точки)

31		4031	invalid field type (неправильный тип поля)
32		4032	invalid record length (неправильная длина записи)
33		4033	bad data (неправильные данные)
34		4034	memo soft line length invalid (Неправильная длина MEMO)
35		4035	MDX flag in DBF file invalid (Неправильный флаг MDX и DBX-файле)
36		4036	file open for reading only (файл открыт только для чтения)
37		4037	file locking violation (файл заблокирован)
38		4038	sharing buffer overflow (ошибка совместного использования буфера)
39		4039	path not found (путь не найден)
40		4040	access to file denied (доступ к файлу запрещен)
41		4041	invalid access code (неправильный код доступа)
42		4042	file must be locked first (файл должен быть заблокирован первым_
43		4043	diskette changed (дискета заменена)
44		4044	bad minimum number of keys per page (неправильное минимальное количество ключей на странице)
45		4045	some files remain open (некоторые файлы остались открытыми)
46		4046	could not open the file (невозможно открыть файл)
47		4047	flush to disk failure (сброс данных на диск не состоялся)
48		4048	invalid tag handle
49		4049	invalid page size in blocks (неправильный размер страницы)
50	4050		invalid tag name
51	4051		invalid page offset adder (неправильное смещение страницы)
52	4052		invalid max number of tag table elements
53	4053		bad tag table element length
54	4054		invalid tag count
55	4055		unknown key format switches
56	4056		unknown switch error
57	4057		tag in use
58	4058		Windows GlobalLock error
59	4059		Windows Task lookup Failed
60	4060		internal lock buffer overflow
61	4061		internal lock buffer underflow

Драйвер dBaseIII

Спецификации

Драйвер dBase 3 совместим с dBaseIII. Расширение файла по умолчанию *.DBF.".

Ключи и индексы существуют, как файлы, отдельные от файла данных. Ключи динамические - они автоматически обновляются при изменении файла данных. Индексы статические - они не обновляются автоматически, а вместо этого требуют для обновления выполнения оператора BUILD. Расширение индексного файла по умолчанию *.NTX.

Драйвер сохраняет записи, как записи фиксированной длины. Поля мемо хранятся в отдельном файле. Имя мемо файла - первые восемь символов метки файла с расширением .DBT.

C4DB3L.LIB	**Статически вызываемая библиотека Windows (16-битовая)**
C4DB3XL.LIB	**Статически вызываемая библиотека Windows (32- битовая)**
C4DB3.LIB	**Библиотека экспорта Windows (16-битовая)**
C4DB3X.LIB	**Библиотека экспорта Windows (32-битовая)**
C4DB3.DLL	**Динамически вызываемая библиотека Windows (16-битовая)**
C4DB3X.DLL	**Динамически вызываемая библиотека Windows (32-битовая)**

Совет: dBaseIII является, по-видимому, наиболее распространенным форматом файла для PC приложений баз данных. В настоящее время даже издательские программы могут импортировать dBaseIII совместимые .DBF файлы. Если главная задача вашего приложения - экспортировать файлы данных для других приложений, о которых вы не знаете ничего, вы должны рассмотреть прежде всего этот формат.

Поддерживаемые типы данных

Формат файла xBase сохраняет все данные в виде строк символов ASCII. Вы можете установить типы строк символов STRING с помощью символьных шаблонов для каждого поля, либо установить типы данных Clarion, которые драйвер преобразует автоматически.

Тип данных DBase III	*Тип данных Clarion*	*символьный шаблон*
Date	DATE	STRING(@D12)
*Numeric	REAL	STRING(@N-_p.d)
*Logical	BYTE	STRING(1)
Character	STRING	STRING
*Memo	MEMO	MEMO

Если ваше приложение читает и записывает в существующие файлы, достаточен будет символьный шаблон. Однако, если ваше приложение *создает* dBASE III файл, вам может потребоваться дополнительная информация для типов данных dBASE III:

¤ Чтобы создать цифровое поле в словаре данных, выберите тип данных REAL (действительный). В атрибуте Внешнее имя на закладке Атрибуты установите *'NumericFieldName*=N(*Precision,DecimalPlaces)'* где *NumericFieldName* - имя поля, *Precision* - точность поля, а *DecimalPlaces* - число десятичных мест. С данными типа REAL вы не можете получить доступ к полям символа или мест в определении поля, вам следует назначить новые атрибуты с помощью выражения во Внешнем поле имени на закладке Атрибуты.

Например, если вы хотите создать поле, называемое Number (номер) с девятью значащими цифрами и двумя десятичными знаками, введите 'Number=N(9,2) во внешнем поле имени на закладе Атрибуты свойств поля в словаре данных.

Если вы кодируете вручную присущие Clarion типы данных, добавьте атрибут NAME, используя тот же самый синтаксис.

Если вы кодируете вручную символьный шаблон, STRING@N-_9.2),NAME(*'Number')*,где *Number* - имя поля.

¤ Чтобы создать логическое поле, используя словарь данных, выберите тип данных BYTE. Специальных шагов нет; смотрите в разных разделах советы о чтении данных из поля.

Если вы вручную кодируете символьный шаблон, добавьте атрибут NAME: STRING(1),NAME('*LogFld*=L').

¤ Чтобы создать поле данных, используя словарь данных, выберите тип данных DATE, а не LONG, который вы обычно используете для форматов файлов TopSpeed или Clarion.

¤ Объявления поля MEMO требуют указателя поля в файловой структуре записи. Объявите указатель поля как STRING(10) или LONG. Это поле будет сохранено в файле .DBF, содержащем смещение мемо в файле .DBT. Объявление MEMO должно иметь атрибут NAME(), называющий указатель поля. Пример объявления поля:

```
File                FILE,DRIVER('DBase III')
Memo1               MEMO(200),NAME('Notes')
Memo2               MEMO(200),NAME('Text')
Rec                 RECORD
Mem1Ptr             LONG,NAME('Notes')
Mem2Ptr             STRING(10),NAME('Text')
END
END
```

Совет: Когда это возможно, пользуйтесь утилитой импорта файла (File Import Utility) в редакторе словаря Clarion для того, чтобы назначить свои файлы.

Характеристики файла/максимумы

Размер файла:	2,000,000,000 байт
Записей на файл:	1,000,000,000 байт
Размер записи:	4,000 байт
Размер поля	
Символы:	254 байт
Данные:	8 байт
Логическое:	1 байт
Цифровое:	20 байт включая десятичную точку
Мемо:	65,520 байт (см. замечание)
Полей на запись:	128
Ключей/индексов на файл:	неограниченно
Размер ключа	
Символ:	100 байт
Цифры, данные:	8 байт
Мемо полей на файл:	в зависимости от имеющейся памяти
Открывание файла:	в зависимости от операционной системы

Строки драйвера

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто

характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. См. Общие характеристики драйверов - Строки драйвера.

Замечание: Некоторые строки драйвера не действуют после того, как файл открыт, поэтому никакая функция SEND выполняется для этих строк. Однако в тоже время функция SEND возвращает значения переключателей для всех строк драйвера

Драйвер dBASE III поддерживает следующие строки драйвера:

BUFFERS

```
DRIVER('DBASE III', '/BUFFERS = n' )
[ Status» = ] SEND(file, 'BUFFERS [ = n ]' )
```

Устанавливает размер буфера, используемого для чтения и записи в файл, где размер буфера равен (n*512). Используйте /BUFFERS строку драйвера для увеличения размера буфера, если доступ к файлу слишком медленен. Максимальный размер буфера равен 65,5054 байт для 16-битовых приложений и 4,294,967,264 для 32-битовых. Функция SEND возвращает размер буфера в байтах.

Совет: По умолчанию имеется три буфера по 1024 байта каждый. Увеличение числа буферов не будет увеличивать производительность программы, если файлы разделяются между несколькими пользователями.

RECOVER

```
DRIVER('DBASE III', '/RECOVER' )
[ Status» = ] SEND(file, 'RECOVER' )
```

Эквивалентно команде Xbase RECALL, которая восстанавливает записи, маркированные для удаления. При использовании драйвера DBase IV оператор DELETE отмечает флажком запись как “неактивную”. Драйвер не удаляет запись до тех пор, пока не выполнена команды PACK (упаковать).

/RECOVER оценивается каждый раз при открывании вами файла, если вы добавите строку символов драйвера к словарю данных. Когда драйвер восстанавливает записи, ранее отлакированные для удаления, вы должны вручную перестроить ключи и индексы с помощью оператора BUILD.

IGNORESTATUS

```
DRIVER('DBASE III', '/IGNORESTATUS = on | off ' )
[ Status» = ] SEND(file, 'IGNORESTATUS [ on | off ]' )
```

Когда задана установка *ON*, драйвер *не* перескакивает через удаленные записи во время выполнения операторов GET, NEXT и PREVIOUS при доступе к файлу, открытому в физической последовательности. Это также делает доступным выполнение оператора PUT для удаленной или задержанной записи. / IGNORESTATUS требует открытия файла в эксклюзивном режиме. Функция SEND возвращает текущее значение IGNORESTATUS(ON или OFF) в форме STRING(3).

DELETED

```
[ Status» = ] SEND(file, 'DELETED' )
```

Для использования только с командой SEND, когда включен IGNORESTATUS. Сообщает состояние текущей записи. Если она удалена, возвращаемая строка символов - "ON", если нет - "OFF".

OMNIS

```
DRIVER('DBASE3', '/OMNIS' )
SEND(file, 'OMNIS' )
```

Назначает совместимость заголовка файла OMNIS и разделителя файла.

Атрибуты файла	Поддержка
CREATE	ДА
DRIVER [*драйвер,строка драйвера*])	ДА
NAME	ДА
ENCRYPT	НЕТ
OWNER(*Пароль)*	НЕТ
RECLAIM	НЕТ[1]
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	Y[12]
EXTERNAL(*член*)	ДА
DLL([*признак*])	ДА
OEM	ДА

Структуры файла	Поддержка
INDEX	ДА
KEY	ДА
MEMO	ДА
BLOB	НЕТ
RECORD	ДА

Атрибуты индекса, ключа, мемо	Поддержка
BINARY	НЕТ
DUP	ДА[2]
NOCASE	ДА
OPT	НЕТ
PRIMARY	ДА
NAME	ДА
Сортировка по возрастанию	ДА
Сортировка по убыванию	ДА
Смешанная сортировка	НЕТ

Атрибуты поля	Поддержка
DIM	НЕТ
OVER	ДА
NAME	ДА

Файловые процедуры	Поддержка
BOF(*файл*)	ДА[8]
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	ДА
BUILD(*ключ*)	ДА
BUILD(*индекс*)	ДА
BUILD(*индекс*, *компонент*)	ДА[3]
BUILD(*индекс*, *компонент*, *фильтр*)	НЕТ
BYTES(*файл*)	НЕТ
CLOSE(*файл*)	ДА
COPY(*файл*, *новый файл*)	ДА[4]
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА[8]

FLUSH(*файл*)	ДА
LOCK(*файл*)	НЕТ
NAME(*метка*)	ДА
OPEN(*файл, режим доступа*)	ДА[5]
PACK(*файл*)	ДА
POINTER(*файл*)	ДА[9]
POINTER(*ключ*)	ДА[9]
POSITION(*файл*)	ДА[10]
POSITION(*ключ*)	ДА[10]
RECORDS(*файл*)	ДА[11]
RECORDS(*ключ*)	ДА[11]
REMOVE(*файл*)	ДА
RENAME(*файл, новый файл*)	ДА[4]
SEND(*файл, Сообщение*)	ДА
SHARE(*файл, режим доступа*)	ДА[5]
STATUS(*файл*)	ДА
STREAM(*файл*)	ДА
UNLOCK(*файл*)	НЕТ

Доступ к записям	Поддержка
ADD(*файл*)	ДА[6]
ADD(*файл, длина*)	НЕТ
APPEND(*файл*)	ДА[6]
APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	ДА[1]
GET(*файл, ключ)*	ДА
GET(*файл, указатель файла*)	ДА
GET(*файл, указатель файла, длина*)	НЕТ
GET(*ключ, указатель ключа*)	ДА
HOLD(*файл*)	ДА[7]
NEXT(*файл*)	ДА
NOMEMO(*файл*)	ДА
PREVIOUS(*файл*)	ДА
PUT(*файл*)	ДА
PUT(*файл, указатель файла*)	ДА
PUT(*файл, указатель файла, длина*)	НЕТ
RELEASE(*файл*)	ДА
REGET(*файл,строка*)	ДА
REGET(*ключ,строка*)	ДА

RESET(*файл,строка*)	ДА
RESET(*ключ,строка*)	ДА
SET(*файл*)	ДА
SET(*файл, ключ*)	ДА
SET(*файл, указатель файла*)	ДА
SET(*ключ*)	ДА
SET(*ключ, ключ*)	ДА
SET(*ключ, указатель ключа*)	ДА
SET(*ключ, ключ, указатель файла*)	ДА
SKIP(*файл, число*)	ДА
WATCH(*файл*)	ДА

Обработка транзакций	Поддержка[13]
LOGOUT(*вр. ож., файл, ..., файл*)	НЕТ
COMMIT	НЕТ
ROLLBACK	НЕТ

Обработка пустых записей	Поддержка
NULL(*поле*)	НЕТ
SETNULL(*поле*)	НЕТ
SETNONNULL(*поле*)	НЕТ

1 Когда драйвер удаляет запись из базы данных DBase III, запись физически не удаляется, а вместо этого драйвер маркирует ее, как неактивную. Поля мемо не удаляются физически из мемо файла, однако они не могут быть найдены, если они относятся к неактивной записи. Записи ключа удаляются из индексного файла.

Чтобы удалить записи и мемо файлы навсегда, выполните оператор PACK(file).

Совет: Для программистов, знакомых с DBase III, этот драйвер обрабатывает удаленные записи в соответствии с тем, как DBase III обрабатывает их после того, как дана команда SET DELETED ON (установите удаленное). Записи, отмеченные для удаления, игнорируются исполняемыми программными операторами, но остаются в файле данных.

2 dBASE III не поддерживает никаких видов уникальных индексов. Поэтому все ключи должны иметь атрибут DUP.

3 При построении динамических индексов *компоненты* могут принять одну из двух следующих форм:

- BUILD(DynNdx, '+Pre:FLD1, -Pre:FLD2')

Эта форма устанавливает имена полей, на которых строится индекс. Имена полей должны появиться в атрибуте поля NAME, если он используется, или это должны быть имена меток. Для совместимости с Clarion может быть использован префикс, но он игнорируется.

- BUILD(DynNdx, 'T[Expression]')

Эта форма устанавливает тип и выражение, используемые для построения индекса, о чем смотрите ниже - в разделе "Разное" - *Назначение ключа*..

4 Команда COPY() копирует файлы данных и мемо файлы, используя *newfile (*новый файл*)*, при помощи которого вы может установить новое имя файла или каталога. Файлы ключей или индексов копируются, если *newfile* является характеристикой подкаталога. Для копирования индексного файла в новый файл используйте специальную форму команды копирования:

COPY(file, '<index>|< newfile >')

Если используется ссылка на недействительный индекс, этот оператор выдает сообщение *File Not Found (файл не найден)*. Команда COPY предполагает по усмотрению расширение ".NTX" для имен файлов источника и целевого файла, если ни одно из них специально не определено. Если вы требуете имя файла без расширения, закончите имя точкой.

Дана структура файла:

```
Clar2  FILE,CREATE,DRIVER('DBase III')
NumKey  KEY(Num),DUP
StrKey                    KEY(Str1)
StrKey2                   KEY(Str2)
AMemo                     MEMO(100),NAME('mem')
Record                    RECORD
Num                       STRING(@n-_9.2)
STR1                      STRING(2)
STR2                      STRING(2)
  Mem                     STRING(10)
```

Следующие команды копируют это определение файла в A:

```
COPY(Clar2,'A:\CLAR2')
COPY(Clar2,'StrKey|A:\STRKEY')
COPY(Clar2,'StrKey2|A:\STRKEY2')
COPY(Clar2,'NumKey|A:\NUMKEY')
```

После этих вызовов на накопителе A будут существовать следующие файлы: CLAR2.DBE,CLAR2.DBT,STRKEY.NTX,STRLEY2.NTX и NUMKEY.NTX.

5 Вам не нужен SHARE (или VSHARE) в любой среде (NOVEL, например), которая поддерживает блокировку собственными средствами.

6 Оператор ADD проверяет на дублирование ключи перед модификацией файла данных или связанных с ним ключевых KEY файлов. В результате оператор ADD выполняется медленнее, чем APPEND, который не выполняет никаких проверок и не обновляет ключи KEY. При добавлении больших количеств данных к базе данных, используйте APPEND...BUILD вместо ADD.

7 DBase III выполняет блокирование записи путем блокирования всей записи в файле данных. Это предотвращает доступ для чтения для других процессов. Следовательно, мы рекомендуем уменьшить до предела количество времени, на которое удерживается запись.

8 Хотя драйвер поддерживает эти функции, мы не рекомендуем их использовать. Эти функции для своего выполнения требуют физического доступа к файлам и значительных ресурсов. Вместо этого проверьте величину, возвращаемую функцией ERRORCODE() после каждого последовательного доступа. NEXT() или PREVIOUS() посылают код ошибки 33 (запись недоступна), если сделана попытка получить доступ к записи за концом или началом файла.

9 Нет различия между указателями файла и указателями ключа; и тот и другой содержат ту же самую величину для любой данной записи.

10 Функция POSITION(file) возвращает STRING(12). Функция POSITION(key) возвращает строку длиной, равной размеру ключа плюс 4 байта

11 В рамках Clipper функция RECORDS() возвращает одно и то же число записей для файла данных и его ключей и индексов. Обычно не бывает различия в числе записей, если INDEX не устарел. Так как оператор DELETE не удаляет записи физически, *число записей, о которых сообщает функция* RECORDS(), *включает неактивные записи*. Будьте осторожны при использовании этой функции.

12 Файлы с атрибутом THREAD требуют дополнительного блока управления для каждого процесса, получающего доступ к данному файлу.

13 Вы должны убрать отметку в поле **Enclose RI code in transaction frame** на закладке **File Control** tab в диалоге **Global Properties** когда используете DBase III драйвер для приложений, генерируемых с использованием шаблонов.

Разное

Булева оценка

¤ DBase III позволяет логическому полю принять одну из девяти возможных величин (y,Y,n,N,t,T,f,F или символ пробела). Символ пробела - это ни истина и ни ложь. При использовании логического поля из заранее существующей базы данных в логическом выражении учтите все эти возможности. Помните, что когда поле STRING используется как выражение, оно истинно, если содержит какие-либо данные, и оно ложно, если равно нулю или пусто. Следовательно, чтобы оценить истинность логического поля, выражение должно быть истина, если поле содержит какую-либо из "истинных" характеристик (T,t,Y или y). Например, если логическое поле было использовано, чтобы установить продукт как налогооблагаемый или необлагаемый, выражение для оценки его истинности будет:
(*If Condition):*

```
Taxable='T' OR Taxable='t' OR Txable='Y' OR
Taxable='y'
```

Большие MEMO

¤ Clarion for Windows поддерживает поля MEMO размером до 64K. Если у вас существует файл, включающий поле мемо, намного большее, чем 64K, вы можете использовать файл, но не модифицировать большие MEMO.

¤ Вы можете определить, когда ваше приложение встретит большое MEMO путем определения, что переменная указателя мемо не пуста, но поле мемо при этом выглядит пустым. Выдается код ошибки 47 (неправильное объявление записи). Если вы попытаетесь обновить такую запись, любая модификация MEMO поля будет игнорироваться.

Длинные имена поля

¤ DBase III поддерживает максимум 10 символов в имени поля. Если вам нужно больше, используйте External Name (внешнее имя) с 10 или менее символами.

Последовательность сортировки

¤ Драйвер DBase III Clarion for Windows поддерживает международные порядки сортировки, однако, для соблюдения совместимости с международным

порядком сортировки DBase III, удалите строку CLADIGRAPH= из файла ..\CLARION4\BIN\ CLARION4.ENV.

Определение ключа

¤ DBase III поддерживает использование выражений для определения ключей. В редакторе словаря вы можете поместить выражение во внешнее имя поля в диалоговом окне Key Properties(свойства ключа). Общий формат внешнего имени:

'FileName=T[Expression]'

Где *FileName* представляет имя индексного файла (который может содержать путь и расширение файла), а *T* представляет тип индекса. Действительными типами являются: C = символ, D = данные, N = цифра. Если типом является D или N, тогда *Expression (выражение)* может назвать только одно поле.

Выражения в виде строк символов могут использовать оператор "+" для конкатенации многих строк символов аргументов. Цифровые выражения используют операторы '+' или '-' в их основном смысле. Максимальная длина DBase III выражения - 250 символов.

Выражение может относиться к множественным полям в записи и содержать функции xBase. Квадратные скобки должны ставиться вокруг выражения. Текущие поддерживаемые функции даются ниже. Если драйвер встречает неподдерживаемую xBase функцию в ранее существовавшем файле, он посылает код ошибки 76 - "Недействительная строка символов индекса", когда файл открыт для ключей и статических индексов.

Поддерживаемые xBase функции определения ключа

ALLTRIN(string)	**Удаляет начальные и конечные пробелы.**
CTOD(string)	**Преобразует ключ в виде строки символов в дату. Формат строки *string* mm/dd/yy; результат имеет форму 'yyyymmdd'. Элемент yyyy по умолчанию означает двадцатое столетие. Недействительные данные дают в результате пустой ключ.**
DELETED()	**Возвращает TRUE, если запись удалена.**
DTOC(date)	**Преобразует ключ данных в строку символов формата 'mm/dd/yy'.**

DTOS(date) Преобразует ключ данных в строку символов формата 'yyyymmdd'.

FIXED(float) Преобразует ключ из представления с плавающей точкой в представление с фиксированной точкой.

FLOAT(numeric) Преобразует ключ из представления с фиксированной точкой в представление с плавающей точкой.

IIF(bool, val1, val2) Возвращает величину val1 если первый параметр TRUE, иначе возвращает величину val2.

LEFT(string, n) Возвращает самые левые *n* символов строки символов ключа в виде строки символов длиной *n*.

LOWER (string, n) Преобразует строку символов ключа в нижний регистр.

LTRIM (string) Удаляет пробелы с левой стороны от строки символов.

RECNO( ) Возвращает номер текущей записи.

RIGHT(string, n) Возвращает самые правые *n* символов строки символов ключа в виде строки символов длиной *n*.

RTRIM(string) Удаляет пробелы с правой стороны строки символов.

STR(numeric [,length[, decimal places]]) Преобразует цифровые данные в строку символов. Длина строки символов и число десятичных знаков после запятой - параметры необязательные. По умолчанию длина строки символов 10, а число знаков после десятичной точки 0.

SUBSTR(string,offset,n) Возвращает подстроку символов ключа в виде строки символов *string*, начиная с позиции *offset* и длиной в *n* символов.

TRIM (string) Удаляет пробелы с правой стороны от строки символов (идентично RTRIM).

UPPER(string) **Преобразует строку символов ключа в верхний регистр.**

VAL(string) **Преобразует строку символов ключа в цифру.**

Сообщения об ошибках

Все драйверы баз данных TopSpeed's посылают коды завершения и сообщения, доступ к которым осуществляется при помощи функций ERRORCODE(), ERROR(), FILEERRORCODE() и FILEERROR() (См. Описание языка). Драйвер посылает код немедленно после завершения каждой операции ввода/вывода (OPEN, NEXT, GET, ADD, DELETE, и т.д.).

Совет: Функция ERRORFILE() возвращает имя файла или таблицы которая выдала ошибку.

См. Ошибки времени выполнения (*Run Time Errors)* в описании языка, где приведено описание кодов ошибок и их соответствие сообщениям, выдаваемым функцией ERROR(). Текст сообщения, выдаваемый функцией ERROR(), может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Для получения дополнительной информации см. «Интернационализация» (*Internationalization), CLAMSG,* и *LOCALE* в описании языка.

В дополнение, текст сообщений FILEERROR() DBase III драйвера также может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Однако значение CLAMSG должно быть равно величине, возвращаемой функцией FILEERRORCODE() + 4000.

FILEERRORCODE	CLAMSG	FILEERROR
1	4001	file write failure(Ошибка чтения файла)
2	4002	file read failure(Ошибка записи в файл)
3	4003	memory allocation error(Ошибка распределения памяти)
4	4004	file pointer reposition failed(Ошибка позиционирования)
5	4005	file not found(файл не найден)

6		4006	file corrupted(файл разрушен)
7		4007	bad user specified key expression (ошибка спецификации ключа)
8		4008	no handles available
9		4009	no index pages loaded
10		4010	index page was not loaded
11		4011	file close failure (ошибка закрытия файла)
12		4012	invalid command (неправильная команда)
13		4013	invalid handle number
14		4014	invalid filename (неправильное имя файла)
15		4015	invalid date (неправильная дата)
16		4016	invalid time (неправильное время)
17		4017	file not in memo format(не MEMO файл)
18		4018	invalid DBase III version (неправильная версия DBase III)
19		4019	file header length error (неправильная длина заголовка файла)
20		4020	last file change date in error (неправильная дата последней коррекции файла)
21		4021	parameter address NULL (адрес параметра пустой)
22		4022	invalid key type (неправильный тип ключа)
23		4023	invalid key length (неправильная длина ключа)
24		4024	item length incorrect (неправильная длина элемента)
25		4025	invalid root page (неправильная корневая страница)
26		4026	bad maximum number of keys per page (неправильное число ключей на странице)
27	4027		invalid number of fields (неправильное число полей)
28	4028		field name invalid (неправильное имя поля)
29	4029		bad field length (неправильная длина поля)
30	4030		decimal places parameter invalid (неправильная позиция десятичной точки)
31	4031		invalid field type (неправильный тип поля)
32	4032		invalid record length (неправильная длина записи)
33	4033		bad data (неправильные данные)

34	4034	memo soft line length invalid (Неправильная длина MEMO)
35	4035	MDX flag in DBF file invalid (Неправильный флаг MDX и DBX-файле)
36	4036	file open for reading only (файл открыт только для чтения)
37	4037	file locking violation (файл заблокирован)
38	4038	sharing buffer overflow (ошибка совместного использования буфера)
39	4039	path not found (путь не найден)
40	4040	access to file denied (доступ к файлу запрещен)
41	4041	invalid access code (неправильный код доступа)
42	4042	file must be locked first (файл должен быть заблокирован первым_
43	4043	diskette changed (дискета заменена)
44	4044	bad minimum number of keys per page (неправильное минимальное количество ключей на странице)
45	4045	some files remain open (некоторые файлы остались открытыми)
46	4046	could not open the file (невозможно открыть файл)
47	4047	flush to disk failure (сброс данных на диск не состоялся)
48	4048	invalid tag handle
49	4049	invalid page size in blocks (неправильный размер страницы)
50	4050	invalid tag name
51	4051	invalid page offset adder (неправильное смещение страницы)
52	4052	invalid max number of tag table elements
53	4053	bad tag table element length
54	4054	invalid tag count
55	4055	unknown key format switches
56	4056	unknown switch error
57	4057	tag in use
58	4058	Windows GlobalLock error
59	4059	Windows Task lookup Failed
60	4060	internal lock buffer overflow
61	4061	internal lock buffer underflow

Драйвер dBaseIV

Спецификации

Драйвер dBase 4 совместим с dBaseIV. Расширение файла по умолчанию *.DBF.

Ключи и индексы существуют, как файлы, отдельные от файла данных. Ключи динамические - они автоматически обновляются при изменении файла данных. Индексы статические - они не обновляются автоматически, а вместо этого требуют для обновления выполнения оператора BUILD. Расширение индексного файла по умолчанию *.NDX .

Драйвер сохраняет записи, как записи фиксированной длины. Поля мемо хранятся в отдельном файле. Имя мемо файла - первые восемь символов метки файла с расширением .DBT.

C4DB4L.LIB	Статически вызываемая библиотека Windows (16-битовая)
C4DB4XL.LIB	Статически вызываемая библиотека Windows (32- битовая)
C4DB4.LIB	Библиотека экспорта Windows (16-битовая)
C4DB4X.LIB	Библиотека экспорта Windows (32-битовая)
C4DB4.DLL	Динамически вызываемая библиотека Windows (16-битовая)
C4DB4X.DLL	Динамически вызываемая библиотека Windows (32-битовая)

Совет: dBaseIV никогда не была так широко распространена, как dBase III. Выбирайте этот драйвер только тогда, когда вы должны обмениваться данными с другим конечным пользователем, использующим dBase IV.

Типы данных

Формат файла xBase сохраняет все данные в виде строк символов ASCII. Вы можете установить типы строк символов STRING с помощью символьных шаблонов для каждого поля, либо установить типы данных Clarion, которые драйвер преобразует автоматически.

Тип данных DBase IV	*Тип данных Clarion*	*символьный шаблон*
Date	DATE	STRING(@D12)
*Numeric	REAL	STRING(@N-_p.d)
*Logical	BYTE	STRING(1)
Character	STRING	STRING
*Memo	MEMO	MEMO

Если ваше приложение читает и записывает в существующие файлы, достаточен будет символьный шаблон. Однако, если ваше приложение *создает* dBASE IV файл, вам может потребоваться дополнительная информация для этих типов данных dBASE IV:

¤ Чтобы создать цифровое поле в словаре данных, выберите тип данных REAL (действительный). В атрибуте Внешнее имя на закладке Атрибуты установите *'NumericFieldName*=N(*Precision,DecimalPlaces)'*, где *NumericFieldName* - имя поля, *Precision* - точность поля, а *DecimalPlaces* - число знаков после запятой. С данными типа REAL вы не можете получить доступ к полям, определяющим число значащих символов и положение десятичной точки, и поэтому вам следует назначить новые атрибуты с помощью выражения во Внешнем поле имени на закладке Атрибуты.

Например, если вы хотите создать поле, называемое Number (номер) с девятью значащими цифрами и двумя десятичными знаками, введите 'Number=N(9,2) во внешнем поле имени на закладе Атрибуты свойств поля в словаре данных.

Если вы кодируете вручную присущие Clarion типы данных, добавьте атрибут NAME, используя тот же самый синтаксис.

Если вы кодируете вручную символьный шаблон, STRING@N-_9.2),NAME(*'Number'),* где *Number* - имя поля.

¤ Чтобы создать логическое поле, используя словарь данных, выберите тип данных BYTE. Специальных шагов нет; однако, смотрите в разных разделах советы о чтении данных из поля.
Если вы вручную кодируете символьный шаблон, добавьте атрибут NAME: STRING(1),NAME('*LogFld*=L').

¤ Чтобы создать поле данных, используя словарь данных, выберите тип данных DATE, а не LONG, который вы обычно используете для форматов файлов TopSpeed или Clarion.

¤ Объявления поля MEMO требуют указателя поля в файловой структуре записи. Объявите указатель поля как STRING(10) или LONG. Это поле будет сохранено в файле .DBF, содержащем смещение мемо в файле .DBT. Объявление MEMO должно иметь атрибут NAME(), называющий указатель поля. Пример объявления поля:

```
File                FILE,DRIVER('DBase IV')
Memo1               MEMO(200),NAME('Notes')
Memo2               MEMO(200),NAME('Text')
Rec                    RECORD
Mem1Ptr             LONG,NAME('Notes')
Mem2Ptr             STRING(10),NAME('Text')
END
END
```

Совет: Когда это возможно, для определения ваших собственных файлов пользуйтесь утилитой импорта файла (File Import Utility) в редакторе словаря Clarion.

Характеристики файла/максимумы

Размер файла:	2,000,000,000 байт
Записей на файл:	1,000,000,000 байт
Размер записи:	4,000 байт
Размер поля	
Символы:	254 байт
Данные:	8 байт
Логическое:	1 байт
Цифровое:	20 байт включая десятичную точку
С плавающей точкой:	20 байт включая десятичную точку
Мемо:	65,520 байт (см. замечание)
Полей на запись:	512
Ключей/индексов на файл:	
.NDX	неограниченно
.MDX	47 на .MDX файл
Размер ключа	
Символ:	100 байт
Цифры, данные:	8 байт
Мемо полей на файл:	в зависимости от имеющейся памяти
Открытых файлов:	в зависимости от операционной системы

Строки драйвера

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. См. Общие характеристики драйверов - Строки драйвера.

Замечание: Некоторые строки драйвера не действуют после того, как файл открыт, поэтому никакая функция SEND выполняется для этих строк. Однако в тоже время функция SEND возвращает значения переключателей для всех строк драйвера

Драйвер dBASE IV поддерживает следующие строки драйвера:

BUFFERS

```
DRIVER('DBASE IV', '/BUFFERS = n' )
[ Status» = ] SEND(file, 'BUFFERS [ = n ]' )
```

Устанавливает размер буфера, используемого для чтения и записи в файл, где размер буфера равен (n*512). Используйте /BUFFERS строку драйвера для увеличения размера буфера, если доступ к файлу слишком медленен. Максимальный размер буфера равен 65,5054 байт для 16-битовых приложений и 4,294,967,264 для 32-битовых. Функция SEND возвращает размер буфера в байтах.

Совет: По умолчанию имеется три буфера по 1024 байта каждый. Увеличение числа буферов не будет увеличивать производительность программы, если файлы разделяются между несколькими пользователями.

RECOVER

```
DRIVER('DBASE4, '/RECOVER' )
[ Status» = ] SEND(file, 'RECOVER' )
```

Эквивалентно команде Xbase RECALL, которая восстанавливает записи, маркированные для удаления. При использовании драйвера DBase IV оператор DELETE отмечает флажком запись как “неактивную”. Драйвер не удаляет запись до тех пор, пока не выполнена команды PACK (упаковать).

RECOVER оценивается каждый раз при открытии вами файла, если вы добавите

строку символов драйвера к словарю данных. Когда драйвер восстанавливает записи, ранее отлакированные для удаления, вы должны вручную перестроить ключи и индексы с помощью оператора BUILD.

IGNORESTATUS

DRIVER('DBASE4', '/IGNORESTATUS = on | off ')
[Status» =] SEND(file, 'IGNORESTATUS [on | off]')

Когда установка *ON*, драйвер *не* перескакивает через удаленные записи во время выполнения операторов GET, NEXT и PREVIOUS при доступе к файлу, открытому в физической последовательности. Это также делает доступным выполнение оператора PUT для удаленной или задержанной записи. /IGNORESTATUS требует открытия файла в эксклюзивном режиме. Функция SEND возвращает текущее значение IGNORESTATUS(ON или OFF) в форме STRING(3).

DELETED

[Status» =] SEND(file, 'DELETED')

Для использования только с командой SEND, когда включен IGNORESTATUS. Сообщает состояние текущей записи. Если она удалена, возвращаемая строка символов - "ON", если нет - "OFF".

Атрибуты файла	**Поддержка**
CREATE	ДА
DRIVER [*драйвер,строка драйвера*])	ДА
NAME	ДА
ENCRYPT	НЕТ
OWNER(*Пароль)*	НЕТ
RECLAIM	НЕТ[1]
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	Y[12]
EXTERNAL(*член*)	ДА
DLL([*признак*])	ДА
OEM	**ДА**

Структуры файла	**Поддержка**
INDEX	ДА
KEY	ДА
MEMO	ДА

BLOB	НЕТ
RECORD	**ДА**

Атрибуты индекса, ключа, мемо	**Поддержка**
BINARY	НЕТ
DUP	ДА[2]
NOCASE	ДА
OPT	НЕТ
PRIMARY	ДА
NAME	ДА
Сортировка по возрастанию	ДА
Сортировка по убыванию	ДА
Смешанная сортировка	НЕТ

Атрибуты поля	**Поддержка**
DIM	НЕТ
OVER	ДА
NAME	**ДА**

Файловые процедуры	**Поддержка**
BOF(*файл*)	ДА[8]
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	ДА
BUILD(*ключ*)	ДА
BUILD(*индекс*)	ДА
BUILD(*индекс*, *компонент*)	ДА[3]
BUILD(*индекс*, *компонент*, *фильтр*)	НЕТ
BYTES(*файл*)	НЕТ
CLOSE(*файл*)	ДА
COPY(*файл*, *новый файл*)	ДА[4]
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА[8]
FLUSH(*файл*)	ДА
LOCK(*файл*)	НЕТ
NAME(*метка*)	ДА
OPEN(*файл*, *режим доступа*)	ДА[5]

PACK(*файл*)	ДА
POINTER(*файл*)	ДА[9]
POINTER(*ключ*)	ДА[9]
POSITION(*файл*)	ДА[10]
POSITION(*ключ*)	ДА[10]
RECORDS(*файл*)	ДА[11]
RECORDS(*ключ*)	ДА[11]
REMOVE(*файл*)	ДА
RENAME(*файл, новый файл*)	ДА[4]
SEND(*файл, Сообщение*)	ДА
SHARE(*файл, режим доступа*)	ДА[5]
STATUS(*файл*)	ДА
STREAM(*файл*)	ДА
UNLOCK(*файл*)	НЕТ

Доступ к записям	**Поддержка**
ADD(*файл*)	ДА[6]
ADD(*файл, длина*)	НЕТ
APPEND(*файл*)	ДА[6]
APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	ДА[1]
GET(*файл, ключ)*	ДА
GET(*файл, указатель файла*)	ДА
GET(*файл, указатель файла, длина*)	НЕТ
GET(*ключ, указатель ключа*)	ДА
HOLD(*файл*)	ДА[7]
NEXT(*файл*)	ДА
NOMEMO(*файл*)	ДА
PREVIOUS(*файл*)	ДА
PUT(*файл*)	ДА
PUT(*файл, указатель файла*)	ДА
PUT(*файл, указатель файла, длина*)	НЕТ
RELEASE(*файл*)	ДА
REGET(*файл,строка*)	ДА
REGET(*ключ,строка*)	ДА
RESET(*файл,строка*)	ДА
RESET(*ключ,строка*)	ДА
SET(*файл*)	ДА
SET(*файл, ключ*)	ДА

SET(*файл, указатель файла*)	ДА
SET(*ключ*)	ДА
SET(*ключ, ключ*)	ДА
SET(*ключ, указатель ключа*)	ДА
SET(*ключ, ключ, указатель файла*)	ДА
SKIP(*файл, число*)	ДА
WATCH(*файл*)	ДА

Обработка транзакций	**Поддержка**[13]
LOGOUT(*вр. ож., файл, ..., файл*)	НЕТ
COMMIT	НЕТ
ROLLBACK	НЕТ

Обработка пустых записей	**Поддержка**
NULL(*поле*)	НЕТ
SETNULL(*поле*)	НЕТ
SETNONNULL(*поле*)	НЕТ

1 Когда драйвер удаляет запись из базы данных DBase IV, запись физически не удаляется, а вместо этого драйвер маркирует ее, как неактивную. Поля мемо не удаляются физически из мемо файла, однако они не могут быть отысканы, если они относятся к неактивной записи. Записи ключа удаляются из индексного файла. Чтобы удалить записи и мемо файлы навсегда, выполните оператор PACK(file).

Совет: Для программистов, знакомых с DBase IV, этот драйвер обрабатывает удаленные записи в соответствии с тем, как DBase IV обрабатывает их после того, как дана команда SET DELETED ON. Записи, отмеченные для удаления, игнорируются исполняемыми программными операторами, но остаются в файле данных.

2 В DBase IV нормальным является ввод многих записей с дубликатами уникальных ключевых компонент. Однако только первая из этих записей индексирована. Таким образом обработка в ключевом порядке показывает только эту первую запись. Если вы удаляете запись, затем вводите новую запись с той же самой величиной ключа, ключевой файл продолжает указывать на удаленную запись, а не на новую запись. В этой ситуации файловый драйвер DBase IV изменяет ключевой файл, чтобы указать на активную запись, а не на удаленную запись. Это означает, что если вы используете программу DBase IV для удаления уникальной записи, а затем вставите дубликат этой записи, новая запись будет невидима при обработке в ключевом порядке до тех пор, пока не выполнена упаковка. Если вы

выполняете тот же самый процесс в программе Clarion, новая запись оказывается видимой при обработке в ключевом порядке.

3 При построении динамических индексов *компоненты* могут принять одну из двух следующих форм:

- BUILD(DynNdx, '+Pre:FLD1, -Pre:FLD2')

Эта форма устанавливает имена полей, на которых строится индекс. Имена полей должны появиться в атрибуте поля NAME, если он используется, или это должны быть имена меток. Для совместимости с Clarion может быть использован префикс, но он игнорируется.

- BUILD(DynNdx, 'T[Expression]')

Эта форма устанавливает тип и выражение, используемые для построения индекса, о чем смотрите ниже, - в разделе "Разное" - *Назначение ключа*.

4 Команда COPY() копирует файлы данных и мемо файлы, используя *newfile (*новый файл*)*, при помощи которого вы может установить новое имя файла или каталога. Файлы ключей или индексов копируются, если *newfile* является характеристикой подкаталога. Для копирования индексного файла в новый файл используйте специальную форму команды копирования:

COPY(file, '<index>|< newfile >')

Если используется ссылка на недействительный индекс, этот оператор выдает сообщение *File Not Found (файл не найден)*. Команда COPY предполагает по усмотрению расширение ".NTX" для имен файлов источника и целевого файла, если ни одно из них специально не определено. Если вы требуете имя файла без расширения, закончите имя точкой.

Дана структура файла:

```
Clar2  FILE,CREATE,DRIVER('DBase IV')
NumKey  KEY(Num),DUP
StrKey                    KEY(Str1)
StrKey2                   KEY(Str2)
AMemo                     MEMO(100),NAME('mem')
Record                    RECORD
Num                       STRING(@n-_9.2)
STR1                      STRING(2)
STR2                      STRING(2)
  Mem                     STRING(10)
```

Следующие команды копируют это определение файла в A:

```
COPY(Clar2,'A:\CLAR2')
COPY(Clar2,'StrKey|A:\STRKEY')
COPY(Clar2,'StrKey2|A:\STRKEY2')
COPY(Clar2,'NumKey|A:\NUMKEY')
```

После этих вызовов на накопителе A будут существовать следующие файлы: CLAR2.DBE,CLAR2.DBT,STRKEY.NTX,STRLEY2.NTX и NUMKEY.NTX.

5 Вам не нужен SHARE (или VSHARE) в любой среде (NOVEL, например), которая поддерживает блокировку собственными средствами.

6 Оператор ADD проверяет на дублирование ключи перед модификацией файла данных или связанных с ним ключевых KEY файлов. В результате оператор ADD выполняется медленнее, чем APPEND, который не выполняет никаких проверок и не обновляет ключи KEY. При добавлении больших количеств данных к базе данных, используйте APPEND...BUILD вместо ADD.

7 DBase IV выполняет блокирование записи (путем блокирования всей записи) в файле данных. Это предотвращает доступ для чтения для других процессов. Следовательно, мы рекомендуем уменьшить до предела количество времени, на которое удерживается запись.

8 Хотя драйвер поддерживает эти функции, мы не рекомендуем их использовать. Эти функции для своего выполнения требуют физического доступа к файлам и значительных ресурсов. Вместо этого проверьте величину, возвращаемую функцией ERRORCODE() после каждого последовательного доступа. NEXT() или PREVIOUS() посылают код ошибки 33 (запись недоступна), если сделана попытка получить доступ к записи за концом или началом файла.

9 Нет различия между указателями файла и указателями ключа; и тот и другой содержат ту же самую величину для любой данной записи.

10 Функция POSITION(file) возвращает STRING(12). Функция POSITION(key) возвращает строку длиной, равной размеру ключа, плюс 4 байта.

11 В рамках Clipper функция RECORDS() возвращает одно и то же число записей для файла данных и его ключей и индексов. Обычно не бывает различия в числе записей, если INDEX не устарел. Так как оператор DELETE не удаляет записи физически, *число записей, о которых сообщает функция* RECORDS(), *включает*

неактивные записи. Будьте осторожны при использовании этой функции.

12 Файлы с атрибутом THREAD требуют дополнительного блока управления для каждого процесса, получающего доступ к данному файлу.

13 Вы должны убрать отметку в поле Enclose RI code in transaction frame **на закладке** File Control **tab в диалоге** Global Properties **когда используете DBase IV драйвер для приложений, генерируемых с использованием шаблонов.**

Разное

Международная последовательность сортировки

Драйвер dBase IV сортирует так, как будто бы в поле нет диакритических знаков, поэтому А сортируется так же, как Д. Если два слова идентичны за исключением диакритических знаков, то эти слова сортируются так, как будто диакритический символ больше, чем нормальный символ. Например, ?a < Ab < ?b , в то время как CLADIGRAPH ?AE будет сортировать как Ab <?a< ?b. Решение - если тот же самый файл использован в Clarion for Windows и в dBase IV, выполните оператор BUILD, чтобы перестроить ключи перед обновлением файла (чтение файла не вызывает проблем).

Булева оценка

DBase IV позволяет логическому полю принять одну из 11 возможных величин (1,0,y,Y,n,N,t,T,f,F или символ пробела). Символ пробела - это ни истина и ни ложь. При использовании логического поля из заранее существующей базы данных в логическом выражении учтите все эти возможности. Помните, что когда поле STRING используется как выражение, оно истинно, если содержит какие-либо данные, и оно ложно, если равно нулю или пусто. Следовательно, чтобы оценить истинность логического поля, выражение должно быть истина, если поле содержит какую-либо из “истинных” характеристик (T,t,Y или y). Например, если логическое поле было использовано, чтобы установить продукт как налогооблагаемый или необлагаемый, выражение для оценки его истинности будет:

(*If Condition):*

```
Taxable='1' OR Taxable='T' OR Taxable='t' OR Taxable='Y' OR Taxable='y'
```

Большие MEMO

¤ Clarion for Windows поддерживает поля MEMO размером до 64K. Если у вас существует файл, включающий поле мемо, намного большее чем 64K, вы можете использовать файл, но не модифицировать большие MEMO.

¤ Вы можете определить, когда ваше приложение встретит большое MEMO путем определения, что переменная указателя мемо не пуста, но поле мемо при этом

выглядит пустым. Выдается код ошибки 47 (неправильное объявление записи). Если вы попытаетесь обновить такую запись, любая модификация MEMO поля будет игнорироваться.

Длинные имена поля

¤ DBase IV поддерживает максимум 10 символов в имени поля. Если вам нужно больше, используйте External Name (внешнее имя) с 10 или менее символами.

Определение ключа

¤ DBase IV поддерживает использование выражений для определения ключей. В редакторе словаря вы можете поместить выражение во внешнее имя поля в диалоговом окне Key Properties(свойства ключа). Общий формат внешнего имени:

```
'FileName=T[Expression]',
```

где *FileName* представляет имя индексного файла (который может содержать путь и расширение файла), а *T* представляет тип индекса. Действительными типами являются: C = символ, D = данные, N = цифра. Если типом является D или N, тогда *Expression (выражение)* может назвать только одно поле.

¤ Множественный индекс (.MDX) требует для атрибутов KEY или INDEX атрибута NAME() для определения типа памяти для ключа и любого выражения, генерирующего значение ключа. Общий формат NAME() атрибута для KEY или INDEX:

```
NAME('TagName|FileName[PageSize]=T[Expression],FOR[Expression]')
```

Далее описываются параметры NAME() атрибута

TagName	**Имя метки индекса внутри множественного индекса. Если пропущена, то драйвер создает файл в стиле .NDX dBASE IV с именем, указанном в FileName**
FileName	**Имя индексного файла, которое может содержать путь и расширение.**
PageSize	**Указывает, что когда создается .MDX файл (если TagName указано), число в диапазоне от 2 до 32 специфицирующее количество 512-байтовых блоков в каждой индексной странице. Эта величина используется только в момент создания файла. Если вы для каждой метки индекса в одном и том же .MDX файле указываете различные размеры страниц, то выбрано будет максимальное значение. По умолчанию 2.**
T	**Указывает тип индекса. Допустимыми типами являются: C = символьный, D - дата, N - числа. Если тип D или N тогда**

Expression	**Expression(выражение) может именовать только одно поле. Указывает выражение для генерации индекса. Это может быть ссылка к нескольким полям и вызов функций xBASE. Поддерживаемые функции приведены ниже. Выражение должно быть заключено в квадратные скобки.**

Элементы NAME() атрибута могут опускаться (справа - налево). Когда вы задаете выражение, вы должны также указать тип и имя. Если выражение пропущено, драйвер будет определять выражение из поля ключа при создании файла, или из индекса при открытии файла.

Если опущен тип, драйвер определит тип индекса по первой компоненте ключа при создании файла или из индекса при открытии файла.

Если атрибут NAME() пропущен целиком, имя индексного файла будет определяться из метки ключа. По умолчанию путь будет таким же, как и .DBF.

Длина имени метки не должна превышать 9 символов. Если имя слишком длинное, оно будет автоматически усечено.

Специфицируйте все имена полей в NAME() атрибуте без префиксов.

¤ dBase IV дополнительно поддерживает использование оператора xBASE FOR в выражениях для определения ключа. Эти выражения должны быть простыми условиями в форме :

expression comparison_op expression

comparison_op may be <, <=, =<, <>, =, =>, >= or >.

Выражения могут ссылаться на несколько полей в записи и содержать функции xBASE. Поддерживаемые функции приведены ниже. Если в момент открытия файла драйвер обнаружит неподдерживаемую xBASE функцию, будет послано сообщение ERROR76 ‘INVALID INDEX STRING’.

Строковые выражения могут использовать оператор ‘+’ для конкатенации отдельных строковых аргументов. Численные выражения используют операторы ‘+’ и ‘-‘ в их обычном смысле. Максимальная длина dBase IV выражения равно 250 символам.

.

Выражения в виде строк символов могут использовать оператор "+" для конкатенации многих строк символов аргументов. Цифровые выражения используют операторы '+' или '-' в их основном смысле. Максимальная длина DBase IV выражения - 250 символов.

Выражение может относиться к множественным полям в записи и содержать функции xBase. Квадратные скобки должны ставиться вокруг выражения. Текущие поддерживаемые функции даются ниже. Если драйвер встречает неподдерживаемую xBase функцию в заранее существовавшем файле, он посылает код ошибки 76 "Недействительная строка символов индекса", когда файл открыт для ключей и статических индексов.

Поддерживаемые xBase функции определения ключа

ALLTRIN(string)	Удаляет начальные и конечные пробелы.
CTOD(string)	Преобразует ключ в виде строки символов в дату. Формат строки *string* mm/dd/yy; результат имеет форму 'yyyymmdd'. Элемент yyyy по умолчанию означает двадцатое столетие. Недействительные данные дают в результате пустой ключ.
DELETED()	Возвращает TRUE, если запись удалена.
DTOC(date)	Преобразует ключ данных в строку символов формата 'mm/dd/yy'.
DTOS(date)	Преобразует ключ данных в строку символов формата 'yyyymmdd'.
FIXED(float)	Преобразует ключ из представления с плавающей точкой в представление с фиксированной точкой.
FLOAT(numeric)	Преобразует ключ из представления с фиксированной точкой в представление с плавающей точкой.
IIF(bool, val1, val2)	Возвращает величину val1, если первый параметр TRUE, иначе возвращает величину val2.
LEFT(string, n)	Возвращает самые левые *n* символов строки символов ключа в виде строки символов длиной *n*.
LOWER (string, n)	Преобразует строку символов ключа в нижний регистр.
LTRIM (string)	Удаляет пробелы с левой стороны от строки символов.

RECNO( ) Возвращает номер текущей записи.

RIGHT(string, n) Возвращает самые правые *n* символов строки символов ключа в виде строки символов длиной *n*.

RTRIM(string) Удаляет пробелы с правой стороны строки символов.

STR(numeric [,length[, decimal places]]) Преобразует цифровые данные в строку символов. Длина строки символов и число десятичных знаков после запятой - параметры необязательные. По умолчанию длина строки символов 10, а число знаков после десятичной точки 0.

SUBSTR(string,offset,n) Возвращает подстроку символов ключа в виде строки символов *string*, начиная с позиции *offset* и длиной в *n* символов.

TRIM (string) Удаляет пробелы с правой стороны от строки символов (идентично RTRIM).

UPPER(string) Преобразует строку символов ключа в верхний регистр.

VAL(string) Преобразует строку символов ключа в цифру.

Сообщения об ошибках

Все драйверы баз данных TopSpeed's посылают коды завершения и сообщения, доступ к которым осуществляется при помощи функций ERRORCODE(), ERROR(), FILEERRORCODE() и FILEERROR() (См. Описание языка). Драйвер посылает код немедленно после завершения каждой операции ввода/вывода (OPEN, NEXT, GET, ADD, DELETE, и т.д.).

Совет: Функция ERRORFILE() возвращает имя файла или таблицы, которая выдала ошибку.

См. Ошибки времени выполнения (*Run Time Errors)* в описании языка, где приведено описание кодов ошибок и их соответствие сообщениям, выдаваемым функцией ERROR(). Текст сообщения, выдаваемый функцией ERROR(), может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Для получения дополнительной информации см. «Интернационализация» (*Internationalization), CLAMSG* и *LOCALE* в описании языка.

В дополнение, текст сообщений FILEERROR() DBase IV драйвера также может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Однако значение CLAMSG должно быть равно величине, возвращаемой функцией FILEERRORCODE() + 4000.

FILEERRORCODE	CLAMSG	FILEERROR
1	4001	file write failure (Ошибка чтения файла)
2	4002	file read failure (Ошибка записи в файл)
3	4003	memory allocation error (Ошибка распределения памяти)
4	4004	file pointer reposition failed (Ошибка позиционирования)
5	4005	file not found (файл не найден)
6	4006	file corrupted (файл разрушен)
7	4007	bad user specified key expression (ошибка спецификации ключа)
8	4008	no handles available
9	4009	no index pages loaded
10	4010	index page was not loaded
11	4011	file close failure (ошибка закрытия файла)
12	4012	invalid command (неправильная команда)
13	4013	invalid handle number
14	4014	invalid filename (неправильное имя файла)
15	4015	invalid date (неправильная дата)
16	4016	invalid time (неправильное время)
17	4017	file not in memo format (не MEMO файл)
18	4018	invalid DBase IV version

(неправильная версия DBase IV)

19 4019 file header length error
(неправильная длина заголовка файла)

20 4020 last file change date in error
(неправильная дата последней коррекции файла)

21 4021 parameter address NULL
(адрес параметра пустой)

22 4022 invalid key type
(неправильный тип ключа)

23 4023 invalid key length
(неправильная длина ключа)

24 4024 item length incorrect
(неправильная длина элемента)

25 4025 invalid root page
(неправильная корневая страница)

26 4026 bad maximum number of keys per page
(неправильное число ключей на странице)

27 4027 invalid number of fields
(неправильное число полей)

28 4028 field name invalid
(неправильное имя поля)

29 4029 bad field length
(неправильная длина поля)

30 4030 decimal places parameter invalid
(неправильная позиция десятичной точки)

31 4031 invalid field type
(неправильный тип поля)

32 4032 invalid record length
(неправильная длина записи)

33 4033 bad data
(неправильные данные)

34 4034 memo soft line length invalid
(Неправильная длина MEMO)

35 4035 MDX flag in DBF file invalid
(Неправильный флаг MDX и DBX-файле)

36 4036 file open for reading only
(файл открыт только для чтения)

37 4037 file locking violation
(файл заблокирован)

38	4038	sharing buffer overflow (ошибка совместного использования буфера)
39	4039	path not found (путь не найден)
40	4040	access to file denied (доступ к файлу запрещен)
41	4041	invalid access code (неправильный код доступа)
42	4042	file must be locked first (файл должен быть заблокирован первым)
43	4043	diskette changed (дискета заменена)
44	4044	bad minimum number of keys per page (неправильное минимальное количество ключей на странице)
45	4045	some files remain open (некоторые файлы остались открытыми)
46	4046	could not open the file (невозможно открыть файл)
47	4047	flush to disk failure (сброс данных на диск не состоялся)
48	4048	invalid tag handle
49	4049	invalid page size in blocks (неправильный размер страницы)
50	4050	invalid tag name
51	4051	invalid page offset adder (неправильное смещение страницы)
52	4052	invalid max number of tag table elements
53	4053	bad tag table element length
54	4054	invalid tag count
55	4055	unknown key format switches
56	4056	unknown switch error
57	4057	tag in use
58	4058	Windows GlobalLock error
59	4059	Windows Task lookup Failed
60	4060	internal lock buffer overflow
61	4061	internal lock buffer underflow

Драйвер DOS

Спецификации

Драйвер файла DOS читает и записывает любые бинарные, байт-адресуемые файлы. Ни поля, ни записи не имеют разграничителей. При чтении записи драйвер читает число байт, определенных в структуре файла RECORD, если параметр длины не установлен в операторе GET.

Драйвер DOS поддерживает параметр длины для операторов ADD, APPEND, GET и PUT; это позволяет вести в файле DOS записи переменной длины.

Функция POINTER (указатель) возвращает относительную байтовую позицию в пределах файла для начала последней записи, к которой имели доступ операторы ADD, APPEND, GET или NEXT.

Этот драйвер файла выполняет только последовательную обработку вперед. Не поддерживаются никакие ключи или функции обработки транзакции, не поддерживается и оператор PREVIOUS.

Совет: В связи с его ограничениями, главная функция данного драйвера - это функция дискового редактора для бинарных файлов.

Файл:

C4DOSL.LIB	Статически вызываемая библиотека Windows (16-битовая)
C4DOSXL.LIB	Статически вызываемая библиотека Windows (32- битовая)
C4DOS.LIB `	Библиотека экспорта Windows (16-битовая)
C4DOSX.LIB	Библиотека экспорта Windows (32-битовая)
C4DOS.DLL	Динамически вызываемая библиотека Windows (16-битовая)
C4DOSX.DLL	Динамически вызываемая библиотека Windows (32-битовая)

Типы данных

BYTE	DECIMAL
SHORT	PDECIMAL
USHORT	STRING
LONG	CSTRING
ULONG	PSTRING
SREAL	DATE

REAL	TIME
BFLOAT4	GROUP
BFLOAT4	

Характеристики файла/максимумы

Размер файла:	4,294,967,295 байт
Записей на файл:	4,294,967,295 байт
Размер записи:	65,520 байт
Размер поля:	65,520 байт
Полей на запись:	65,520
Ключей/Индексов на файл:	Нет
Размер ключа:	Нет
Полей мемо на файл:	Нет
Размер поля мемо:	Нет
Открывание файла данных::	Зависит от операционной системы

Строки драйвера

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. См. Общие характеристики драйверов - Строки драйвера.

Замечание: Некоторые строки символов драйвера не действуют после открытия файла, поэтому синтаксис функции SEND для модификации установки не перечислен. Однако синтаксис функции SEND для возврата величины переключения приводится в перечне для всех строк драйвера.

DOS Драйвер поддерживает следующие строки драйвера:

FILEBUFFERS

```
DRIVER('DOS', '/FILEBUFFERS = n' )
[ Buffers» = ] SEND(file, 'FILEBUFFERS [ = n ]' )
```

Устанавливает размер буфера, используемого для чтения и записи файла. Размер буфера равен n*512 байт. Используйте строку драйвера /FILEBUFFERS для увеличения размера буфера, если скорость доступа к данным слишком мала. Максимальный размер буфера равен 65504 для 16-битовых приложений и

4,294,967,264 для 32-битовых. Функция SEND возвращает размер буфера в байтах.

Совет: По умолчанию размер буфера для файлов, открытых с запретом на запись для других пользователей, больше 1024 или (2*размер записи) и больше 512 или (1*размер записи) - для всех других режимов открытия.

FILEBUFFERS

```
DRIVER('ASCII', '/FILEBUFFERS = n' )
[ Buffers» = ] SEND(file, 'FILEBUFFERS [ = n ]' )
```

Устанавливает размер буфера, используемого для чтения и записи, размер буфера равен n*512. Используйте строку драйвера /FILEBUFFERS для увеличения размера буфера, если доступ к данным слишком медленный. Максимальный размер буфера равен 65,504 для 16-битовых приложений и 4,294,967,264 для 32-битовых. Функция SEND возвращает размер буфера в байтах.

Совет: По умолчанию размер буфера для файлов, открытых с запретом на запись для других пользователей, больше 1024 или (2*размер записи) и больше 512 или (1*размер записи) - для всех других режимов открытия.

QUICKSCAN

```
DRIVER('ASCII', '/QUICKSCAN = on | off' )
[ QScan» = ] SEND(file, 'QUICKSCAN [ = on | off ]' )
```

Устанавливает способ буферизации файла. Драйвер DOS файлов за одно обращение читает буфер (а не запись) и тем самым ускоряет доступ к данным. В многопользовательской среде такой способ буферизации не дает 100% уверенности в правильности данных при последующем к ним доступе, так как за время между отдельными обращении данные могли быть изменены другим пользователем. В качестве меры безопасности драйвер повторно читает данные в буфер перед доступом к каждой записи. Чтобы *отключить* повторное чтение, установите QUICKSCAN на ON. По умолчанию стоит ON для файлов, открытых с запретом доступа по записи для других пользователей, и OFF для всех других режимов открытия файлов. Функция SEND возвращает установку Quickscan (ON или OFF) в виде STRING(3).

Атрибуты файла	**Поддержка**
CREATE	ДА
DRIVER [,*строка драйвера*])	ДА
NAME	ДА

ENCRYPT	НЕТ
OWNER(*Пароль*)	НЕТ
RECLAIM	НЕТ
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	Y[4]
EXTERNAL(*член*)	ДА
DLL([*признак*])	ДА
OEM	**ДА**

Структуры файла	**Поддержка**
INDEX	НЕТ
KEY	НЕТ
MEMO	НЕТ
BLOB	НЕТ
RECORD	**ДА**

Атрибуты индекса, ключа, мемо	**Поддержка**
BINARY	НЕТ
DUP	НЕТ
NOCASE	НЕТ
OPT	НЕТ
PRIMARY	НЕТ
NAME	НЕТ
Сортировка по возрастанию	НЕТ
Сортировка по убыванию	НЕТ
Смешанная сортировка	НЕТ

Атрибуты поля	**Поддержка**
DIM	ДА
OVER	ДА
NAME	**ДА**

Файловые процедуры	**Поддержка**
BOF(*файл*)	НЕТ
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	НЕТ
BUILD(*ключ*)	НЕТ
BUILD(*индекс*)	НЕТ

BUILD(*индекс*, *компонент*)	НЕТ
BUILD(*индекс*, *компонент*, *фильтр*)	НЕТ
BYTES(*файл*)	ДА
CLOSE(*файл*)	ДА
COPY(*файл*, *новый файл*)	ДА
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	НЕТ
DUPLICATE(*ключ*)	НЕТ
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА
FLUSH(*файл*)	НЕТ
LOCK(*файл*)	ДА
NAME(*метка*)	ДА
OPEN(*файл*, *режим доступа*)	ДА
PACK(*файл*)	НЕТ
POINTER(*файл*)	ДА[2]
POINTER(*ключ*)	НЕТ
POSITION(*файл*)	ДА[3]
POSITION(*ключ*)	НЕТ
RECORDS(*файл*)	Да
RECORDS(*ключ*)	НЕТ
REMOVE(*файл*)	ДА
RENAME(*файл*, *новый файл*)	ДА
SEND(*файл*, *Сообщение*)	ДА
SHARE(*файл*, *режим доступа*)	ДА
STATUS(*файл*)	ДА
STREAM(*файл*)	НЕТ
UNLOCK(*файл*)	ДА

Доступ к записям	**Поддержка**
ADD(*файл*)	ДА
ADD(*файл*, *длина*)	ДА
APPEND(*файл*)	ДА
APPEND(*файл*, *длина*)	ДА
DELETE(*файл*)	НЕТ
GET(*файл*, *ключ)*	НЕТ
GET(*файл*, *указатель файла*)	ДА
GET(*файл*, *указатель файла*, *длина*)	ДА
GET(*ключ*, *указатель ключа*)	НЕТ

HOLD(*файл*)	НЕТ
NEXT(*файл*)	ДА
NOMEMO(*файл*)	НЕТ
PREVIOUS(*файл*)	ДА
PUT(*файл*)	ДА
PUT(*файл, указатель файла*)	ДА[1]
PUT(*файл, указатель файла, длина*)	ДА[1]
RELEASE(*файл*)	НЕТ
REGET(*файл,строка*)	ДА
REGET(*ключ,строка*)	НЕТ
RESET(*файл,строка*)	ДА
RESET(*ключ,строка*)	НЕТ
SET(*файл*)	ДА
SET(*файл, ключ*)	НЕТ
SET(*файл, указатель файла*)	ДА
SET(*ключ*)	НЕТ
SET(*ключ, ключ*)	НЕТ
SET(*ключ, указатель ключа*)	НЕТ
SET(*ключ, ключ, указатель файла*)	НЕТ
SKIP(*файл, число*)	НЕТ
WATCH(*файл*)	НЕТ

Обработка транзакций	**Поддержка**
LOGOUT(*вр. ож., файл, ..., файл*)	НЕТ
COMMIT	НЕТ
ROLLBACK	НЕТ

Обработка пустых записей	**Поддержка**
NULL(*поле*)	НЕТ
SETNULL(*поле*)	НЕТ
SETNONNULL(*поле*)	НЕТ

Примечания.

1 Когда используется оператор PUT(), в файл должно быть возвращено такое же количество символов, какое было перед этим прочитано из файла. Если вы возвращаете оператором PUT большее количество символов, чем было прочитано, эти «ЭКСТРА»-символы будут перекрывать символы в последующих записях файла. Если вы возвращаете оператором PUT меньшее количество символов, чем было прочитано, замещено в файле будет только возвращаемое количество символов, остальные же символы в записи исходного файла останутся такими, какими они

были до выполнения оператора PUT.

2 Функция POINTER() возвращает относительную позицию байта внутри файла.

3 POSITION(файл) возвращает STRING(4).

4 THREAD атрибут файла требует дополнительного блока управления файлом для каждого процесса, имеющего доступ к данному файлу.

Драйвер FoxPro / FoxBase

Спецификации

Драйвер FOXPRO совместим с FoxPro / FoxBase. Расширение файла по умолчанию - *.DBF.".

Расширение индексного файла по умолчанию - *.IDX. Расширение по умолчанию файла Мемо - .FBT. FoxPro также поддерживает файлы множественных индексов (расширение по умолчанию - *.CDX). В различных разделах описываются процедуры использования файлов .CDX.

C4FOXL.LIB	Статически вызываемая библиотека Windows (16-битовая)
C4FOXXL.LIB	Статически вызываемая библиотека Windows (32- битовая)
C4FOX.LIB	Библиотека экспорта Windows (16-битовая)
C4FOXX.LIB	Библиотека экспорта Windows (32-битовая)
C4FOX.DLL	Динамически вызываемая библиотека Win dows (16-битовая)
C4FOXX.DLL	Динамически вызываемая библиотека Win dows (32-битовая)

Совет: Формат индексного файла FoxPro является становым хребтом его превозносимой технологии «Rushmore». Старая поговорка «There's no free lunch» применима в данном случае. Добавление и присоединение записей к большой базе данных является более медленным процессом, чем в других форматах xBase, что связано со временем, необходимым для обновления индексного файла.

Типы данных

Формат файла xBase сохраняет все данные в виде строк символов ASCII. Вы можете установить типы строк символов STRING с помощью символьных шаблонов для каждого поля, либо установить типы данных Clarion, которые драйвер преобразует автоматически.

Тип данных FOXPRO / FOXBASE	*Тип данных Clarion*	*символьный шаблон*
Date	DATE	STRING(@D12)
*Numeric	REAL	STRING(@N-_p.d)
*Logical	BYTE	STRING(1)

Character	STRING	STRING
*Memo	MEMO	MEMO

Если ваше приложение читает и записывает в существующие файлы, достаточен будет символьный шаблон. Однако, если ваше приложение *создает* FOXPRO / FOXBASE файл, вам может потребоваться дополнительная информация для этих типов данных FOXPRO / FOXBASE:

• Чтобы создать цифровое поле в словаре данных, выберите тип данных REAL (действительный). В атрибуте Внешнее имя на закладке Атрибуты установите *'NumericFieldName*=N(*Precision,DecimalPlaces)'*, где *NumericFieldName* - имя поля, *Precision* - точность поля, а *DecimalPlaces* - число знаков после запятой. С данными типа REAL вы не можете получить доступ к полям, определяющим число значащих символов и положение десятичной точки, и поэтому вам следует назначить новые атрибуты с помощью выражения во Внешнем поле имени на закладке Атрибуты.

Например, если вы хотите создать поле, называемое Number (номер) с девятью значащими цифрами и двумя десятичными знаками, введите 'Number=N(9,2) во внешнем поле имени на закладе Атрибуты свойств поля в словаре данных.

Если вы кодируете вручную присущие Clarion типы данных, добавьте атрибут NAME, используя тот же самый синтаксис.

Если вы кодируете вручную символьный шаблон, STRING@N-_9.2),NAME(*'Number')*,где *Number* - имя поля.

· Чтобы создать логическое поле, используя словарь данных, выберите тип данных BYTE. Специальных шагов нет; однако, смотрите в разных разделах советы о чтении данных из поля.

Если вы вручную кодируете символьный шаблон, добавьте атрибут NAME: STRING(1),NAME('*LogFld*=L').

• Чтобы создать поле данных, используя словарь данных, выберите тип данных DATE, а не LONG, который вы обычно используете для форматов файлов TopSpeed или Clarion.

• Объявления поля MEMO требуют указателя поля в файловой структуре записи. Объявите указатель поля как STRING(10) или LONG. Это поле будет сохранено в файле .DBF, содержащем смещение мемо в файле .DBT. Объявление MEMO должно иметь атрибут NAME(), называющий указатель поля. Пример

объявления поля:

```
File                    FILE,DRIVER('FOXPRO / FOXBASE')
Memo1                   MEMO(200),NAME('Notes')
Memo2                   MEMO(200),NAME('Text')
Rec                         RECORD
Mem1Ptr                     LONG,NAME('Notes')
Mem2Ptr                     STRING(10),NAME('Text')
END
END
```

Характеристики файла/максимумы

Размер файла:	2,000,000,000 байт
Записей на файл:	1,000,000,000 байт
Размер записи:	4,000 байт
Размер поля	
Символы:	254 байт
Данные:	8 байт
Логическое:	1 байт
Цифровое:	20 байт включая десятичную точку
С плавающей точкой:	20 байт включая десятичную точку
Мемо:	65,520 байт (см. замечание)
Полей на запись:	512
Ключей/индексов на файл:	неограниченно
Размер ключа	
Символ:	100 байт (.IDX) 254 байт (.CDX)
Цифры, данные:	8 байт
Мемо полей на файл:	в зависимости от имеющейся памяти
Открытых файлов:	в зависимости от операционной системы

Строки драйвера

Имеются переключатели или «строки драйверов», при помощи который вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. См. Общие характеристики драйверов - Строки драйвера.

Замечание: Некоторые строки драйвера не действуют после того, как файл открыт, поэтому никакая функция SEND не выполняется для этих строк. Однако в тоже время функция SEND возвращает значения переключателей для всех строк драйвера

Драйвер FOXPRO / FOXBASE поддерживает следующие строки драйвера:

BUFFERS

```
DRIVER('FOXPRO', '/BUFFERS = n' )
[ Status» = ] SEND(file, 'BUFFERS [ = n ]' )
```

Устанавливает размер буфера, используемого для чтения и записи в файл, где размер буфера равен (n*512). Используйте /BUFFERS строку драйвера для увеличения размера буфера, если доступ к файлу слишком медленен. Максимальный размер буфера равен 65,5054 байт для 16-битовых приложений и 4,294,967,264 для 32-битовых. Функция SEND возвращает размер буфера в байтах.

Совет: По умолчанию имеется три буфера по 1024 байта каждый. Увеличение числа буферов не будет увеличивать производительность программы, если файлы разделяются между несколькими пользователями.

RECOVER

```
DRIVER('FOXPRO', '/RECOVER' )
[ Status» = ] SEND(file, 'RECOVER' )
```

Эквивалентно команде Xbase RECALL, которая восстанавливает записи, маркированные для удаления. При использовании драйвера FOXPRO / FOXBASE оператор DELETE отмечает флажком запись как "неактивную". Драйвер не удаляет запись до тех пор, пока не выполнена команды PACK (упаковать).

RECOVER оценивается каждый раз при открытии вами файла, если вы добавите строку символов драйвера к словарю данных. Когда драйвер восстанавливает записи, ранее отлакированные для удаления, вы должны вручную перестроить ключи и индексы с помощью оператора BUILD.

IGNORESTATUS

```
DRIVER('FOXPRO', '/IGNORESTATUS = on | off ' )
[ Status» = ] SEND(file, 'IGNORESTATUS [ on | off ]' )
```

Когда приведена установка *ON*, драйвер *не* перескакивает через удаленные записи во время выполнения операторов GET, NEXT и PREVIOUS при доступе к файлу, открытому в физической последовательности. Это также делает доступным

выполнение оператора PUT для удаленной или задержанной записи. / IGNORESTATUS требует открытия файла в эксклюзивном режиме. Функция SEND возвращает текущее значение IGNORESTATUS(ON или OFF) в форме STRING(3).

DELETED

[Status» =] SEND(file, 'DELETED')

Для использования только с командой SEND, когда включен IGNORESTATUS. Сообщает состояние текущей записи. Если она удалена, возвращаемая строка символов - "ON", если нет - "OFF".

Атрибуты файла	**Поддержка**
CREATE	ДА
DRIVER [*драйвер,строка драйвера*])	ДА
NAME	ДА
ENCRYPT	НЕТ
OWNER(*Пароль)*	НЕТ
RECLAIM	НЕТ[1]
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	Y[13]
EXTERNAL(*член*)	ДА
DLL([*признак*])	ДА
OEM	**НЕТ[2]**

Структуры файла	**Поддержка**
INDEX	ДА
KEY	ДА
MEMO	ДА
BLOB	НЕТ
RECORD	**ДА**

Атрибуты индекса, ключа, мемо	**Поддержка**
BINARY	НЕТ[14]
DUP	ДА[3]
NOCASE	ДА
OPT	НЕТ
PRIMARY	ДА

NAME	ДА
Сортировка по возрастанию	ДА
Сортировка по убыванию	ДА
Смешанная сортировка	НЕТ

Атрибуты поля	**Поддержка**
DIM	НЕТ
OVER	ДА
NAME	**ДА**

Файловые процедуры	**Поддержка**
BOF(*файл*)	ДА[9]
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	ДА
BUILD(*ключ*)	ДА
BUILD(*индекс*)	ДА
BUILD(*индекс*, *компонент*)	ДА[4]
BUILD(*индекс*, *компонент*, *фильтр*)	НЕТ
BYTES(*файл*)	НЕТ
CLOSE(*файл*)	ДА
COPY(*файл, новый файл*)	ДА[5]
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА[9]
FLUSH(*файл*)	ДА
LOCK(*файл*)	НЕТ
NAME(*метка*)	ДА
OPEN(*файл, режим доступа*)	ДА[6]
PACK(*файл*)	ДА
POINTER(*файл*)	ДА[10]
POINTER(*ключ*)	ДА[10]
POSITION(*файл*)	ДА[11]
POSITION(*ключ*)	ДА[11]
RECORDS(*файл*)	ДА[12]
RECORDS(*ключ*)	ДА[12]
REMOVE(*файл*)	ДА
RENAME(*файл, новый файл*)	ДА[5]

SEND(*файл, Сообщение*)	ДА
SHARE(*файл, режим доступа*)	ДА[6]
STATUS(*файл*)	ДА
STREAM(*файл*)	ДА
UNLOCK(*файл*)	НЕТ

Доступ к записям	**Поддержка**
ADD(*файл*)	ДА[7]
ADD(*файл, длина*)	НЕТ
APPEND(*файл*)	ДА[7]
APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	ДА[1]
GET(*файл, ключ)*	ДА
GET(*файл, указатель файла*)	ДА
GET(*файл, указатель файла, длина*)	НЕТ
GET(*ключ, указатель ключа*)	ДА
HOLD(*файл*)	ДА[8]
NEXT(*файл*)	ДА
NOMEMO(*файл*)	ДА
PREVIOUS(*файл*)	ДА
PUT(*файл*)	ДА
PUT(*файл, указатель файла*)	ДА
PUT(*файл, указатель файла, длина*)	НЕТ
RELEASE(*файл*)	ДА
REGET(*файл,строка*)	ДА
REGET(*ключ,строка*)	ДА
RESET(*файл,строка*)	ДА
RESET(*ключ,строка*)	ДА
SET(*файл*)	ДА
SET(*файл, ключ*)	ДА
SET(*файл, указатель файла*)	ДА
SET(*ключ*)	ДА
SET(*ключ, ключ*)	ДА
SET(*ключ, указатель ключа*)	ДА
SET(*ключ, ключ, указатель файла*)	ДА
SKIP(*файл, число*)	ДА
WATCH(*файл*)	ДА

Обработка транзакций	**Поддержка**[15]

LOGOUT(*вр. ож., файл, ..., файл*)	НЕТ
COMMIT	НЕТ
ROLLBACK	НЕТ

Обработка пустых записей	**Поддержка**
NULL(*поле*)	НЕТ
SETNULL(*поле*)	НЕТ
SETNONNULL(*поле*)	НЕТ

1 Когда драйвер удаляет запись из базы данных FOXPRO, запись физически не удаляется, а вместо этого драйвер маркирует ее, как неактивную. Поля мемо не удаляются физически из мемо файла, однако они не могут быть отысканы, если они относятся к неактивной записи. Записи ключа удаляются из индексного файла. Чтобы удалить записи и мемо файлы навсегда, выполните оператор PACK(file).

Совет: Для программистов, знакомых с FOXPRO, этот драйвер обрабатывает удаленные записи в соответствии с тем, как FOXPRO обрабатывает их после того, как дана команда SET DELETED ON. Записи, отмеченные для удаления, игнорируются исполняемыми программными операторами, но остаются в файле данных.

2 Если вам необходим доступ к данным FoxPro, содержащим символы из альтернативной части кодовой таблицы и записанным не английской версией FoxPro, вы должны будете использовать ODBC. Однако, если у вас нет строковых ключей, вы можете использовать драйвер FoxPro, вызывая ConvertOEMToANSI и ConvertANSIToOEM после чтения записи и перед ее обновлением, соответственно.

3 В FOXPRO нормальным является ввод многих записей с дубликатами уникальных ключевых компонент. Однако, только первая из этих записей индексирована. Таким образом, обработка в ключевом порядке показывает только эту первую запись. Если вы удаляете запись, затем вводите новую запись с той же самой величиной ключа, ключевой файл продолжает указывать на удаленную запись, а не на новую запись. В этой ситуации файловый драйвер FOXPRO изменяет ключевой файл, чтобы указать на активную запись, а не на удаленную запись. Это означает, что если вы используете программу FOXPRO для удаления уникальной записи, а затем вставите дубликат этой записи, новая запись будет невидима при обработке в ключевом порядке до тех пор, пока не выполнена упаковка. Если вы выполняете тот же самый процесс в программе Clarion, новая запись оказывается видимой при обработке в ключевом порядке.

4 При построении динамических индексов *компоненты* могут принять одну из двух следующих форм:

```
BUILD(DynNdx, '+Pre:FLD1, -Pre:FLD2' )
```

Эта форма устанавливает имена полей, на которых строится индекс. Имена полей должны появиться в атрибуте поля NAME, если он используется, или это должны быть имена меток. Для совместимости с Clarion может быть использован префикс, но он игнорируется.

```
BUILD(DynNdx, 'T[Expression]')
```

Эта форма устанавливает тип и выражение, используемые для построения индекса, о чем смотрите ниже - в разделе "Разное" - *Назначение ключа*..

5 Команда COPY() копирует файлы данных и мемо файлы, используя *newfile (*новый файл*)*, при помощи которого вы может установить новое имя файла или каталога. Файлы ключей или индексов копируются, если *newfile* является характеристикой подкаталога. Для копирования индексного файла в новый файл используйте специальную форму команды копирования:

```
COPY(file, '<index>|< newfile >')
```

Если используется ссылка на недействительный индекс, этот оператор выдает сообщение *File Not Found (файл не найден)*. Команда COPY предполагает по усмотрению расширение ".NTX" для имен файлов источника и целевого файла, если ни одно из них специально не определено. Если вы требуете имя файла без расширения, закончите имя точкой.

Дана структура файла:

```
Clar2  FILE,CREATE,DRIVER('FOXPRO ')
NumKey  KEY(Num),DUP
StrKey                    KEY(Str1)
StrKey2                   KEY(Str2)
AMemo                     MEMO(100),NAME('mem')
Record                    RECORD
Num                       STRING(@n-_9.2)
STR1                      STRING(2)
STR2                      STRING(2)
  Mem                     STRING(10)
```

Следующие команды копируют это определение файла в A:

```
COPY(Clar2,'A:\CLAR2')
```

```
COPY(Clar2,'StrKey|A:\STRKEY')
COPY(Clar2,'StrKey2|A:\STRKEY2')
COPY(Clar2,'NumKey|A:\NUMKEY')
```

После этих вызовов на накопителе A будут существовать следующие файлы: CLAR2.DBE,CLAR2.DBT,STRKEY.NTX,STRLEY2.NTX и NUMKEY.NTX.

6 Вам не нужен SHARE (или VSHARE) в любой среде (NOVEL, например), которая поддерживает блокировку собственными средствами.

7 Оператор ADD проверяет на дублирование ключи перед модификацией файла данных или связанных с ним ключевых KEY файлов. В результате оператор ADD выполняется медленнее, чем APPEND, который не выполняет никаких проверок и не обновляет ключи KEY. При добавлении больших количеств данных к базе данных, используйте APPEND...BUILD вместо ADD.

8 FOXPRO выполняет блокирование записи (путем блокирования всей записи) в файле данных. Это предотвращает доступ для чтения для других процессов. Следовательно, мы рекомендуем уменьшить до предела количество времени, на которое удерживается запись.

9 Хотя драйвер поддерживает эти функции, мы не рекомендуем их использовать. Эти функции для своего выполнения требуют физического доступа к файлам и значительных ресурсов. Вместо этого проверьте величину, возвращаемую функцией ERRORCODE() после каждого последовательного доступа. NEXT() или PREVIOUS() посылают код ошибки 33 (запись недоступна), если сделана попытка получить доступ к записи за концом или началом файла.

10 Нет различия между указателями файла и указателями ключа; и тот и другой содержат ту же самую величину для любой данной записи.

11 Функция POSITION(file) возвращает STRING(12). Функция POSITION(key) возвращает строку длиной, равной размеру ключа плюс 4 байта.

12 В рамках Clipper функция RECORDS() возвращает одно и то же число записей для файла данных, его ключей и индексов. Обычно не бывает различия в числе записей, если INDEX не устарел. Так как оператор DELETE не удаляет записи физически, *число записей, о которых сообщает функция* RECORDS(), *включает неактивные записи*. Будьте осторожны при использовании этой функции.

13 Файлы с атрибутом THREAD требуют дополнительного блока управления

для каждого процесса, получающего доступ к данному файлу.

14 OEM преобразование не применимо к полям BINARY MEMO. С точки зрения драйвера BINARY MEMO заполнены нулями или пробелами.

15 Вы должны убрать отметку в поле Enclose RI code in transaction frame **на закладке** File Control **tab в диалоге** Global Properties, **когда используете FOXPRO драйвер для приложений, генерируемых с использованием шаблонов.**

Разное

Булева оценка

¤ FOXPRO / FOXBASE позволяет логическому полю принять одну из 11 возможных величин (1,0,y,Y,n,N,t,T,f,F или символ пробела). Символ пробела - это ни истина и ни ложь. При использовании логического поля из заранее существующей базы данных в логическом выражении учтите все эти возможности. Помните, что когда поле STRING используется как выражение, оно истинно, если содержит какие-либо данные, и оно ложно, если равно нулю или пусто. Следовательно, чтобы оценить истинность логического поля, выражение должно быть истина, если поле содержит какую-либо из "истинных" характеристик (T,t,Y или y). Например, если логическое поле было использовано, чтобы установить продукт как налогооблагаемый или необлагаемый, выражение для оценки его истинности будет:

(*If Condition):*
Taxable='1' OR Taxable='T' OR Taxable='t' OR Taxable='Y' OR Taxable='y'

Большие MEMO

¤ Clarion for Windows поддерживает поля MEMO размером до 64K. Если у вас существует файл, включающий поле мемо, намного большее, чем 64K, вы можете использовать файл, но не модифицировать большие MEMO.

¤ Вы можете определить, когда ваше приложение встретит большое MEMO путем определения, что переменная указателя мемо не пуста, но поле мемо при этом выглядит пустым. Выдается код ошибки 47 (неправильное объявление записи). Если вы попытаетесь обновить такую запись, любая модификация MEMO поля будет игнорироваться.

Длинные имена поля

¤ FOXPRO / FOXBASE поддерживает максимум 10 символов в имени поля. Если вам нужно больше, используйте External Name (внешнее имя) с 10 или менее символами.

Определение ключа

¤ FOXPRO / FOXBASE поддерживает использование выражений для определения ключей. В редакторе словаря вы можете поместить выражение во внешнее имя поля в диалоговом окне Key Properties(свойства ключа). Общий формат внешнего имени:

'FileName=T[Expression]',

где *FileName* представляет имя индексного файла (который может содержать путь и расширение файла), а *T* представляет тип индекса. Действительными типами являются: C = символ, D = данные, N = цифра. Если типом является D или N, тогда *Expression (выражение)* может назвать только одно поле.

¤ Множественный индекс (.CDX) требует для атрибутов KEY или INDEX атрибута NAME() для определения типа памяти для ключа и любого выражения, генерирующего значение ключа. Общий формат NAME() атрибута для KEY или INDEX:

NAME('TagName|FileName[PageSize]=T[Expression] ,COMPRESSED')

Далее описываются параметры NAME() атрибута:

TagName **Имя метки индекса внутри множественного индекса. Если пропущена, тогда драйвер создает файл в стиле .NDX FOXPRO / FOXBASE с именем, указанном в FileName**

FileName Имя индексного файла, которое может содержать путь и расширение.

PageSize Указывает, что когда создается .MDX файл (если TagName указано), число в диапазоне от 2 до 32, специфицирующее количество 512-байтовых блоков в каждой индексной странице. Эта величина используется только в момент создания файла. Если вы для каждой метки индекса в одном и том же .MDX файле указываете различные размеры страниц, то выбрано будет максимальное значение. По умолчанию 2.

TУказывает тип индекса. Допустимыми типами являются: C = символьный, D - дата, N - числа. Если тип D или N тогда Expression(выражение) может именовать

только одно поле.

Expression Указывает выражение для генерации индекса. Это может быть ссылка к нескольким полям и вызов функций xBASE. Поддерживаемые функции приведены ниже. Выражение должно быть заключено в квадратные скобки.

Compressed Когда указано, драйвер FoxPro создает совместимый с FoxPro2 сжатый .IDX файл

Элементы NAME() атрибута могут опускаться (справа - налево). Когда вы задаете выражение, вы должны также указать тип и имя. Если выражение пропущено, драйвер будет определять выражение из поля ключа (при создании файла), или из индекса (при открытии файла).

Если опущен тип, драйвер определит тип индекса по первой компоненте ключа при создании файла или из индекса при открытии файла.

Если атрибут NAME() пропущен целиком, имя индексного файла будет определяться из метки ключа. По умолчанию путь будет таким же, как и .DBF.

Длина имени метки не должна превышать 10 символов. Если имя слишком длинное, оно будет автоматически усечено.

Специфицируйте все имена полей в NAME() атрибуте без префиксов.

· FOXPRO / FOXBASE дополнительно поддерживает использование оператора xBASE FOR в выражениях для определения ключа. Эти выражения должны быть простыми условиями в форме :

expression comparison_op expression

comparison_op may be <, <=, =<, <>, =, =>, >= or >.

Выражения могут ссылаться на несколько полей в записи и содержать функции xBASE. Поддерживаемые функции приведены ниже. Если в момент открытия файла драйвер обнаружит неподдерживаемую xBASE функцию, будет послано сообщение ERROR76 'INVALID INDEX STRING'.

Строковые выражения могут использовать оператор '+' для конкатенации отдельных строковых аргументов. Численные выражения используют операторы '+' и '-' в их обычном смысле. Максимальная длина FOXPRO / FOXBASE выражения равно 250 символам.

Поддерживаемые xBase функции определения ключа

ALLTRIN(string) Удаляет начальные и конечные пробелы.

CTOD(string) Преобразует ключ в виде строки символов в дату. Формат строки *string* mm/dd/yy; результат имеет форму 'yyyymmdd'. Элемент yyyy по умолчанию означает двадцатое столетие.
Недействительные данные дают в результате пустой ключ.

DELETED() Возвращает TRUE, если запись удалена.

DTOC(date) Преобразует ключ данных в строку символов формата 'mm/dd/yy'.

DTOS(date) Преобразует ключ данных в строку символов формата 'yyyymmdd'.

FIXED(float) Преобразует ключ из представления с плавающей точкой в представление с фиксированной точкой.

FLOAT(numeric) Преобразует ключ из представления с фиксированной точкой в представление с плавающей точкой.

IIF(bool, val1, val2) Возвращает величину val1, если первый параметр TRUE, иначе возвращает величину val2.

LEFT(string, n) Возвращает самые левые *n* символов строки символов ключа в виде строки символов длиной *n*.

LOWER (string, n) Преобразует строку символов ключа в нижний регистр.

LTRIM (string) Удаляет пробелы с левой стороны от строки символов.

RECNO( ) Возвращает номер текущей записи.

RIGHT(string, n) Возвращает самые правые *n* символов строки символов ключа в виде строки символов длиной *n*.

RTRIM(string) Удаляет пробелы с правой стороны строки символов.

STR(numeric [,length[, decimal places]]) Преобразует цифровые данные в строку символов. Длина строки символов и число десятичных знаков после запятой - параметры необязательные. По умолчанию длина строки символов 10, а число знаков после десятичной точки 0.

SUBSTR(string,offset,n) Возвращает подстроку символов ключа в виде строки символов *string*, начиная с позиции *offset* и длиной в *n* символов.

TRIM (string)	Удаляет пробелы с правой стороны от строки символов (идентично RTRIM).
UPPER(string)	Преобразует строку символов ключа в верхний регистр.
VAL(string)	Преобразует строку символов ключа в цифру.

Сообщения об ошибках

Все драйверы баз данных TopSpeed's посылают коды завершения и сообщения, доступ к которым осуществляется при помощи функций ERRORCODE(), ERROR(), FILEERRORCODE() и FILEERROR() (См. Описание языка). Драйвер посылает код немедленно после завершения каждой операции ввода/вывода (OPEN, NEXT, GET, ADD, DELETE, и т.д.).

Совет: Функция ERRORFILE() возвращает имя файла или таблицы, которая выдала ошибку.

См. Ошибки времени выполнения (*Run Time Errors)* в описании языка, где приведено описание кодов ошибок и их соответствие сообщениям, выдаваемым функцией ERROR(). Текст сообщения, выдаваемый функцией ERROR(), может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Для получения дополнительной информации см. «Интернационализация» (*Internationalization), CLAMSG,* и *LOCALE* в описании языка.

В дополнение, текст сообщений FILEERROR() FOXPRO / FOXBASE драйвера также может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Однако значение CLAMSG должно быть равно величине, возвращаемой функцией FILEERRORCODE() + 4000.

FILEERRORCODE	CLAMSG	FILEERROR
1	4001	file write failure(Ошибка чтения файла)
2	4002	file read failure(Ошибка записи в файл)
3	4003	memory allocation error(Ошибка распределения памяти)
4	4004	file pointer reposition failed(Ошибка позиционирования)
5	4005	file not found(файл не найден)
6	4006	file corrupted(файл разрушен)

7	4007	bad user specified key expression (ошибка спецификации ключа)
8	4008	no handles available
9	4009	no index pages loaded
10	4010	index page was not loaded
11	4011	file close failure (ошибка закрытия файла)
12	4012	invalid command (неправильная команда)
13	4013	invalid handle number
14	4014	invalid filename (неправильное имя файла)
15	4015	invalid date (неправильная дата)
16	4016	invalid time (неправильное время)
17	4017	file not in memo format(не MEMO файл)
18	4018	invalid FOXPRO / FOXBASE version (неправильная версия FOXPRO / FOXBASE)
19	4019	file header length error (неправильная длина заголовка файла)
20	4020	last file change date in error (неправильная дата последней коррекции файла)
21	4021	parameter address NULL (адрес параметра пустой)
22	4022	invalid key type (неправильный тип ключа)
23	4023	invalid key length (неправильная длина ключа)
24	4024	item length incorrect (неправильная длина элемента)
25	4025	invalid root page (неправильная корневая страница)
26	4026	bad maximum number of keys per page (неправильное число ключей на странице)
27	4027	invalid number of fields (неправильное число полей)
28	4028	field name invalid (неправильное имя поля)
29	4029	bad field length (неправильная длина поля)
30	4030	decimal places parameter invalid (неправильная позиция десятичной точки)
31	4031	invalid field type

(неправильный тип поля)
32 4032 invalid record length
(неправильная длина записи)
33 4033 bad data
(неправильные данные)
34 4034 memo soft line length invalid
(Неправильная длина MEMO)
35 4035 MDX flag in DBF file invalid
(Неправильный флаг MDX и DBX-файле)
36 4036 file open for reading only
(файл открыт только для чтения)
37 4037 file locking violation
(файл заблокирован)
38 4038 sharing buffer overflow
(ошибка совместного использования буфера)
39 4039 path not found
(путь не найден)
40 4040 access to file denied
(доступ к файлу запрещен)
41 4041 invalid access code
(неправильный код доступа)
42 4042 file must be locked first
(файл должен быть заблокирован первым)
43 4043 diskette changed
(дискета заменена)
44 4044 bad minimum number of keys per page
(неправильное минимальное количество ключей на странице)
45 4045 some files remain open
(некоторые файлы остались открытыми)
46 4046 could not open the file
(невозможно открыть файл)
47 4047 flush to disk failure
(сброс данных на диск не состоялся)
48 4048 invalid tag handle
49 4049 invalid page size in blocks
(неправильный размер страницы)
50 4050 invalid tag name
51 4051 invalid page offset adder
(неправильное смещение страницы)

52	4052	invalid max number of tag table elements
53	4053	bad tag table element length
54	4054	invalid tag count
55	4055	unknown key format switches
56	4056	unknown switch error
57	4057	tag in use
58	4058	Windows GlobalLock error
59	4059	Windows Task lookup Failed
60	4060	internal lock buffer overflow
61	4061	internal lock buffer underflow

Обзор

Драйверы TopSpeed SQL

Драйверы TopSpeed SQL включают

- AS400
- Informix
- MSSQL
- ODBC
- Oracle
- Oracle
- SQL Anywhere

Драйверы SQL преобразуют стандартные Clarion операторы ввода/вывода и вызовы функций в оптимизированные операторы SQL, которые посылаются к SQL-серверу для выполнения. Таким образом вы можете использовать обычный Clarion код для доступа как к базе данных SQL, так и к обычной файловой системе, подобной TopSpeed. Это также означает, что вы можете использовать Clarion-шаблоны для генерации кода для вашей SQL-базы.

В дополнение к автоматической генерации SQL операторов SQL-драйвер отправляет SQL-серверу любые сформулированные вами SQL-операторы. SQL-драйвер интерпретирует результат выполнения вашего запроса к серверу и делает его доступным вашему приложению через операторы Clarion NEXT и PREVIOUS.

Все общие черты поведения SQL-драйвера описаны в данной главе. Частное поведение драйверов описано в отдельных главах.

Уникальные ключи

SQL-драйвер обычно используется с таблицами, имеющими уникальные ключи. Драйвер будет работать и на файлах без уникальных ключей, но только с существенно ограниченными возможностями. Без уникальных ключей команды RESET и REGET будут возвращать ошибку, и, кроме того, драйвер не сможет обновлять данные в SQL-базе.

Большинство шаблонов CLARION также требуют, чтобы вы определили первичный ключ для каждой таблицы.

Использование SQL-таблицы в вашем CLARION-приложении.

Регистрация SQL-драйвера

Перед тем, как ваше приложение сможет использовать конкретный драйвер, драйвер должен быть зарегистрирован средой CLARION. Драйверы, входящие в стандартный комплект поставки CLARION, уже зарегистрированы. Вы должны регистрировать любые добавляемые вами драйверы. См. *Clarion's Development Environment - Database Driver Registry (среда CLARION - регистрация драйверов) в Руководстве пользователя* для получения дополнительной информации о процедуре регистрации драйверов базы данных.

Импорт определения таблиц.

Как правило вы добавляете поддержку SQL в ваше приложение, импортируя SQL-таблицу, представление или определение синонимов в ваш CLARION-словарь данных. См. *Редактор словаря - Импорт определения файла (The Dictionary Editor—Importing File Definitions)* в *Руководстве пользователя* для получения дополнительной информации об импорте таблиц файлов и определений представлений. В данном разделе описывается импорт в целом. Специфические особенности импорта для каждого конкретного драйвера описаны в руководстве для этих драйверов.

Несмотря на то, что вы можете добавить определение вашего SQL файла в словарь вручную (или даже написать вручную объявление файла), мы настоятельно рекомендуем импортировать определение файла. Импорт таблицы уменьшает вероятность ошибки в словаре и гарантирует корректные спецификации типов данных, ключей, структур данных и т.д.

Импортированием достигается полное соответствие ваших SQL-таблиц базе данных SQL. В том случае, когда вы разрабатываете новую базу данных SQL, вы можете, конечно, планировать определение таблиц в словаре базы данных CLARION. Однако мы рекомендуем такой способ только для прототипа базы данных с минимальной сложности и минимумом эксплутационных требований. В большинстве случаев правильно обустроенная SQL-база данных требует определения гораздо большего количества всякого рода элементов, чем это может предоставить словарь данных CLARION - например, хранимые процедуры, триггеры, права доступа и распределение памяти.

После того как ваши таблицы определены в словаре данных, вы можете разрабатывать приложения для вашей SQL-базы точно также, как вы это делаете для других приложений.

Замечание: Специфические особенности импорта описаны в руководствах для каждого драйвера.

Мастер импорта SQL - диалог регистрации (Login Dialog)

Когда вы выбираете драйвер SQL из выпадающего списка, мастер импорта открывает диалоговое окна Login/Connection. **Диалог** Login/Connection **собирает информацию, необходимую для связи с SQL-базой данных.**

Замечание: Для того, чтобы вы могли связаться с SQL-базой данных и импортировать определение таблицы, база данных должна быть запущена и должна быть доступна для вашего компьютера.

Заполните поля в диалоговом окне Login/Connection
Next > **нажмите эту кнопку для открытия списка импорта.**

Мастер импорта SQL - диалог списка импорта (Import List Dialog)

Когда вы нажимаете кнопку NEXT, **мастер импорта открывает окно, в котором появляются элементы базы данных, которые можно импортировать.**

Отметьте таблицы, представления или синонимы, которые необходимо импортировать и затем нажмите кнопку FINISH **для запуска импорта. Мастер импорта добавит определения в ваш словарь данных и затем откроет диалоговое окно** Свойства файла **(**File Properties), **в котором вы сможете модифицировать, если это необходимо, определения, принятые по умолчанию.**

Импорт дополнительных таблиц, представлений и синонимов выполняется точно также. После того, как импорт всех элементов завершен, возвратитесь в словарь данных для того, чтобы определить отношения и удалить те колонки в таблицах, которые ваше CLARION-приложение не использует. См. *Advanced Techniques—Define Only the Fields You Use.*

Connect-информация и конфигурация драйвера - Свойства файла.

Как правило вы добавляете поддержку SQL в ваше приложение, импортируя SQL или ODBC таблицу, представление и синонимы в ваш CLARION-словарь данных. Мастер импорта автоматически заполняет диалоговое окно «Свойства

файла» **некоторыми значениями, основанными на свойствах импортируемых элементов. Однако в диалоге «Свойства файла» имеется несколько полей, которые позволяют вам конфигурировать способ, которым SQL-драйвер будет получать доступ к базе данных. Эти поля описаны ниже.**

Опции драйвера

Обычно Мастер импорта ничего не размещает в поле «Опции драйвера»**. Однако вы можете добавить в это поле строку драйвера для управления способом, каким драйвер получает доступ к SQL-данным. Например, вы можете запустить журнал событий драйвера или определить, как драйвер должен обрабатывать пустые поля базы данных. См главу SQL строки драйвера для получения дополнительной информации.**

Имя владельца (Owner Name)

Обычно Мастер импорта в поле «Имя владельца» **размещает информацию для связи в SQL-базой данных: имя базы, имя пользователь, пароль и т.д.**

Для безопасности и переносимости вы можете захотеть специфицировать эту информацию, используя переменную вместо жесткого кодирования в вашем словаре. Для этого в поле Имя владельца (Owner Name) **вы должны написать имя переменной, которая будет хранить эту информацию. Имени переменной должен предшествовать восклицательный знак, !LoginString, например. После этого вы можете перед подключением к SQL-базе присваивать этой переменной любое значение, определяющее метод доступа к базе.**

Некоторые SQL-драйверы допускают дополнительную информацию в поле Имя владельца (Owner Name). **Эта информация описывается в документации по конкретным драйверам.**

Поведение SQL-Драйвера

Диалог автоматического подключения. (Automatic Login Dialog)

При получении доступа к SQL-таблице SQL драйвер автоматически просматривает «Имя пользователя» и «Пароль». Если «Имя пользователя» и «Пароль» к этому моменту уже зарегистрированы, драйвер использует эти значения. Если нет, драйвер запускает диалог автоматического подключения к базе данных и запрашивает «Имя пользователя» и «Пароль»

Замечание: Исключение составляет ODBC-драйвер для 16-ти разрядных

приложений, использующих библиотеку времени исполнения, вы должны для использования автоматического диалога подключения к базе данных добавить в ваш файл проекта файл \LIBSRC\driver.RSC. Для получения дополнительной информации см. Система проекта в Руководстве пользователя.

Мы рекомендуем открывать таблицы при старте вашей программы так, чтобы время, требуемое на регистрацию в базе данных, объединялось с временем запуска программы. Для SQL-драйверов Мастер приложения автоматически генерирует код, выполняющий это. Однако, если вы не используете Мастер приложения, вы можете достигнуть этого эффекта просто добавив SQL-таблицу в схему файлов для вашей главной процедуры. Это автоматически сгенерирует код для открытия базы.

Исключение составляет драйвер ODBC, для которого автоматический диалог подключения к базе позволяет пользователю вводить имя пользователя, пароль и имя базы данных.

В выпадающем списке баз данных выберите выбранный раннее главный компьютер. Если список баз данных пуст, вы можете ввести имя базы данных.

Обработка SET/NEXT и SET/PREVIOUS (SELECT/ORDER BY)

Оператор SET с последующим оператором NEXT в цикле LOOP является наиболее распространенным методом последовательной обработки в CLARION. Когда SQL драйвер идентифицирует комбинацию SET / NEXT, он генерирует оператор SELECT с предложением ORDER BY, основанным на ключе. Ключевое поле определяется из оператора SET. Например, драйвер SQL транслирует этот CLARION код:

```
Ord
FILE,PRE(Ord),DRIVER('SQLDriver'),NAME('ord')
NameDate KEY(+Ord:Name,+Ord:Date),NAME('DateKey')
Record                                  RECORD
Name                                    STRING(12),NAME('NameId')
Date                                    DATE,NAME('OrderDate')
Type                                    STRING(1),NAME('OrderType')
Details                                 STRING(20),NAME('OrderDetails')
                                                .       .
        CODE
        Ord:Name = 'SMITH'
        SET(Ord:NameDate,Ord:NameDate)
        LOOP
```

```
NEXT(Ord)
!... some processing
        END
```

в оператор SELECT

```
SELECT NameId,OrderDate,OrderType,OrderDetails FROM Ord
                    WHERE (NameID >= 'SMITH')
                    ORDER BY NameID, OrderDate
```

Совет: Оператор SET(file) (для обработки в физической последовательности, не в ключевой) поддерживает только NEXT оператор. Любые попытки выполнить оператор PREVIOUS при обработке в физической последовательности приводят к ошибке с ERRORCODE 80 (Функция не поддерживается)

NULL поля

Когда вы читаете строки SQL-таблицы с нулевыми (NULL) значениями полей, буфер записи CLARION будет содержать пустые строки для строковых полей и 0 для числовых полей, и функция ISNULL(поле) будет возвращать для этих полей TRUE. Если значение полей впоследствии изменится на ненулевые для числовых и непустые для строковых значений, функция ISNULL(поле) будет возвращать для этих полей FALSE.

Если вы хотите изменить NULL-поле на не NULL, но оставить его значение равным нулю или пустым, вы должны использовать процедуру SETNONULL(поле).

Если вы хотите сделать поле, которое до этого момента было не NULL, NULL-полем, воспользуйтесь процедурой SETNULL(поле) или SETNULL(запись). SETNULL() очищает поле или запись и устанавливает для них NULL-флаг.

Когда добавляется новая запись в файл по умолчанию, все пустые поля добавляются как нулевые или пустые строки, не NULL. Если вы хотите принудительно сделать поля NULL, вы должны использовать или SETNULL(поле) или SETNULL(запись).

Совет: ABC-шаблоны предоставляют точки вставки PrimeRecord и ValidateField, где вы можете вручную установить NULL для приложения в целом.

Соображения о производительности

Обычно среда разработки CLARION (Словарь данных, Мастер импорта, Драйверы базы данных и шаблоны) производит оптимизированные, высокопроизводительные SQL-приложения.

В данном разделе описываются некоторые вопросы, касающиеся разработки таких оптимизированных приложений. Вы должны помнить об этих вопросах для того, чтобы иметь возможность поддерживать высокий уровень ваших приложений в тех случаях, когда вы увеличиваете ваш контроль над процессом разработки.

Определение только используемых полей.

Для SQL-драйвера вам необходимо определить в словаре данных CLARION только актуальные для вас поля данных файла. Это уменьшает взаимное влияние друг на друга ваших CLARION - приложений и загрузку сети.

Например, если ваша SQL-таблица содержит 200 колонок, но для данной конкретной программы необходимы только три из них, затребуйте только эти три поля и в результате объем пересылаемых через сеть данных кардинально уменьшится. Если каждая колонка содержит 20 байт, то использование этих трех колонок потребуют передачи только 60 байт против передачи 200 колонок общим размером 4000 байт.

После того, как вы импортировали определение таблицы в словарь данных CLARION, используйте диалог редактор словаря Field / Key Definition **для тех колонок\полей, которые вам не понадобятся.**

Соответствие ключей CLARION SQL ограничениям и индексам

Обычно определение ключей CLARION не нуждается в точном соответствии с индексами базы данных SQL. Ключ CLARION просто превращается в предложение ORDER BY генерируемого драйвером оператора SELECT.

Однако, если ключ CLARION не соответствует ключу или индексу SQL, SQL-сервер должен будет строить временное логическое представление каждый раз, когда вы будете обращаться к таблице, используя этот несоответствующий ключ. Это может существенно замедлить работу с большими файлами.

Наилучшим способом, гарантирующим соответствие ключей CLARION и индексов SQL является импорт таблиц представления или синонимов в словарь данных CLARION . См. *Импорт определения таблиц.*(*Import the Table Definitions).*

ABC шаблоны и ABC библиотека оптимизации.

ABC шаблоны предоставляют некоторые возможности оптимизации выполнения просмотров для SQL-баз данных. Для получения дополнительной информации см. *Обзор шаблонов - —Classes Tab Options—Global—конфигурация BrowseClass Просмотр. См. также BrowseClass Properties—свойства ActiveInvisible, AllowUnfilled, и RetainRow.*

Библиотека ABC также предоставляет некоторые свойства для оптимизации просмотров для SQL-баз данных. Эти свойства управляют автоматической буферизацией для незапланированных представлений. Библиотека ABC автоматически выполняет буферизацию для основанных на SQL полей просмотра. Для получения дополнительной информации см. *ViewManager Properties—свойства PagesAhead, PagesBehind, PageSize,и TimeOut.*

Фильтрующий (сжимающий) локатор

Использование фильтрующего локатора в вашем поле просмотра вместо инкрементного или пошагового локатора может уменьшить объем данных, передаваемых между клиентом и сервером. Для получения дополнительной информации см. Шаблоны поля просмотра.(*BrowseBox Control Template).*

Приблизительный подсчет записей

По умолчанию шаблоны CLARION генерируют код для подсчета общего количества записей, которые должны быть обработаны в отчете. Этот подсчет позволяет точно сформировать индикатор состояния генерации отчета. Однако для больших таблиц выполнение оператора SELECT COUNT(*) может быть очень медленным.

Поэтому для больших отчетов мы рекомендуем для подавления оператора SELECT COUNT(*) приблизительный подсчет количества записей:

1. В диалоге «Дерево приложения**» (**Application Tree**) щелкните правой кнопкой мыши процедуру REPORT и затем из выпадающего списка выберите** Properties

Этим самым вы откроете диалог Свойства процедуры (Procedure Properties).

2. Нажмите кнопку Свойства отчета (Report Properties**) для того, чтобы открыть диалоговое окно** Report Properties

3. В поле Фильтр записей (Record Filter) **введите 1 для того чтобы стал**

возможным приблизительный подсчет записей.

Другой фильтр вы можете использовать почти во всех случаях когда *PrimaryKey* > 0. В некоторых случаях это может существенно увеличить скорость вашего отчета при доступе к данным по первичному ключу.

В поле Approx. Record Count **(Приблизительное количество записей) введите приблизительное количество записей в вашем отчете, например 5000.**

Нажмите кнопку ОК для того, чтобы закрыть диалоговое окно Report Properties.

Нажмите кнопку ОК еще раз для того, чтобы вернутся в диалоговое окно Application Tree

Нажмите кнопку SAVE **для сохранения вашей работы.**

Фиксированный и подвижный ползунок.

По умолчанию Мастер генерирует CLARION - код полей просмотра SQL - таблиц с фиксированным ползунком, потому что подвижный ползунок может стать причиной существенного уменьшения скорости работы для больших таблиц в CLARION/SQL приложениях.

1. В диалоговом окне Application Tree **щелкните правой кнопкой мыши на BROWSE процедуре для выбора** Extensions **из выпадающего меню.**

Этим самым вы откроете диалоговое окно Extension and Control Templates **(Шаблоны расширения и элементов управления)**

2. Списке выберите *Browse on …*, и затем нажмите кнопку Scroll Bar Behavior **(Поведение поля прокрутки).**

Откроется диалоговое окно Scroll Bar Behavior **(Поведение поля прокрутки).**

3. В списке Scroll Bar Type **(тип поля прокрутки) выберите *Fixed Thumb* (фиксированный ползунок), и затем нажмите кнопку** OK**.**

4. Нажмите кнопку ОК еще раз для того, чтобы вернутся в диалоговое окно Application Tree

5. Нажмите кнопку SAVE **для сохранения вашей работы.**

Пакетная обработка транзакций.

Большинство SQL-баз данных работает в режиме автоматического подтверждения конца транзакций. Это означает, что любые операции, которые изменяют таблицы (ADD, PUT или DELETE) выполняются с неявным подтверждением (COMMIT). Это может очень замедлять массированные обновления.

Для оптимизации пакетной обработки заключите пакетный процесс в транзакционную рамку (LOGOUT и COMMIT). Оператор LOGOUT отключит все неявные COMMIT до окончания пакетной транзакции явным оператором COMMIT или ROLLBACK. Например:

```
LOGOUT(.1,OrderDetail)                    !Начало транзакции
DO ErrHandler                             !всегда проверять на ошибки
LOOP X# = 1 TO RECORDS(DetailQue)!обработка записей
GET(DetailQue,X#)                         !чтение очередной из очереди
DO ErrHandler                             !проверка ошибок
Det:Record = DetailQue                    !присвоение буфера записи
ADD(OrderDetail)                          !и добавление записи в файл
DO ErrHandler                             ! проверка ошибок
END
COMMIT                                    !Успешное завершение транзакции
ErrHandler ROUTINE                        !Обработка ошибок
IF NOT ERRORCODE() THEN EXIT.             !Ошибок нет
ROLLBACK                                  !Откат неудачной транзакции
MESSAGE('Transaction Error - ' & ERROR())!Запись ошибки в протокол
RETURN                                    !выход
```

Вам может потребоваться вызвать промежуточный оператор COMMIT и LOGOUT для сохранения данных через заданные интервалы времени. Для получения дополнительной информации см. *Описание языка.*

Использование встроенного SQL.

Вы можете использовать синтаксис свойств Clarion (PROP:SQL) для того, чтобы послать SQL-оператор SQL-серверу, во время выполнения вашей обычной программы. Для обратной совместимости вы можете также использовать для посылки SQL оператора функцию SEND, однако мы рекомендуем использовать синтаксис свойств.

PROP:SQL

Вы можете использовать синтаксис свойств Clarion (PROP:SQL) для посылки SQL-оператора SQL-серверу во время выполнения вашей обычной программы. Вы можете послать любой SQL-оператор, поддерживаемый SQL-сервером.

Эта возможность позволяет вашей программе создавать внутренний SQL-интерфейс, независимый от SQL-драйвера. Например, множественные изменения записей могут часто выполнятся более эффективно простым SQL-оператором, нежели процедурой, сгенерированной шаблоном, которая изменяет записи по одной за раз. В случае, подобном этому, может иметь смысл взять управление на себя и послать серверу собственный SQL-оператор, и PROP:SQL позволит вам сделать это.

Функции FILEERRORCODE() и FILEERROR() возвращают коды ошибок и соответствующие им сообщения, установленные на SQL-сервере.

Вы можете также затребовать содержимое PROP:SQL для получения последнего SQL-оператора, выданного SQL-сервером.

Пример:

```
SQLFile{PROP:SQL}='SELECT field1,field2 FROM table1' |
& 'WHERE field1 > (SELECT max(field1)'                |
& 'FROM table2'                                      !Возвращает множество записей
SQLFile{PROP:SQL}='CALL GetRowsBetween(2,8)'         !Вызов хранимой процедуры
SQLFile{PROP:SQL}='CREATE INDEX ON table1(field1 DESC)'   !Создание индекса.
                                                     Выходного множества нет
SQLFile{PROP:SQL}='GRANT SELECT ON mytable TO fred'
                                                     !Задача администратора базы данных
SQLString=SQLFile{PROP:SQL}                          !Получение последнего SQL-оператора
SEND
```

Для посылки серверу SQL-команд вы можете использовать функцию SEND. Это актуально для совместимости по интерфейсу с более ранними версиями Clarion. Мы рекомендуем для посылки SQL-операторов SQL-серверу использовать синтаксис свойств.

Пример

```
SEND(SQLFile,'SELECT field1,field2 FROM table1'          |
```

```
& 'WHERE field1 > (SELECT max(field1)'    |
& 'FROM table2')
SEND(SQLFile,'CALL GetRowsBetween(2,8)')      !Вызов хранимой процедуры
SEND(SQLFile,'CREATE INDEX ON table1(field1 DESC)') !Выходного множества нет
```

Использование встроенного SQL для пакетной обработки

SQL выполняет большую работу в таких процедурах пакетной обработки, как печать отчетов, вывод экранов, заполненных строками таблиц или при массированном обновление таблиц.

Драйверы SQL наиболее полезны при просмотре таблиц или при печати. Однако в шаблоне пакетной обработки (Process template) или в коде, содержащим цикл, в котором записи обновляются одна за другой, возможности SQL используются не самым лучшим образом. Поэтому при создании пакетного обновления таблиц вместо использования шаблона Process лучше использовать встроенный SQL.

Например, Увеличить зарплату всем продавцам на 10% вы могли бы следующим способом:

```
        SQLFile
FILE,DRIVER('Oracle'),NAME(SalaryFile)
Record
        RECORD
        SalaryAmount                    PDECIMAL(5,2),NAME('JOB')
                                        .       .
                                    CODE
        SqlFile{PROP:SQL} = 'UPDATE SalaryFile SET '&|
         'SALARY=SALARY * 1.1 WHERE JOB=''S'''
```

Имена, используемые в SQL-операторе, являются именами SQL-таблиц, но не именами Clarion-полей.

PROP:SQLFilter

Вы можете использовать PROP:SQLFILTER для фильтрации вашего представления. PROP:SQLFILTER использует код собственно SQL, который выполняется быстрее, чем код Clarion.

Когда вы используете PROP:SQLFILTER, процесс фильтрации происходит непосредственно на сервере. В этом случае фильтр не должен содержать переменные или функции, неизвестные серверу, выражение фильтра должно полностью

соответствовать SQL-синтаксису и содержать правильные имена SQL-колонок.

Например:

```
View{PROP:SQLFilter} = 'Date = TO_DATE(''01-MAY-1996'',''DD-MON-YYYY'')'
или
View{PROP:SQLFilter} = 'StrField LIKE ''AD%'''
```

Комбинирование VIEW фильтра и SQL фильтра

Когда вы используете PROP:SQLFILTER, SQL-фильтр может замещать любой фильтр, определенный для VIEW или он может добавляться к фильтру, определенному для VIEW. Знак (+) перед SQL-фильтром добавляет SQL-фильтр к фильтру, определенному для VIEW.

Например:

```
View{PROP:SQLFilter} = '+ StrField LIKE ''AD%'''
```

Когда вы добавляете SQL-фильтр со знаком (+), фильтрация происходит таким образом, что отбираются записи, удовлетворяющие выражению (VIEW- фильтр) AND (SQL-фильтр).

Если знак (+) отсутствует, то SQL-фильтр замещает Clarion-фильтр. Когда вы замещаете Clarion-фильтр на SQL-фильтр, записи отбираются таким образом, как будто имеется только один SQL-фильтр.

Вызов хранимых процедур.

CALL

Для вызова хранимых процедур вы должны использовать в задании свойств стандартный SQL-синтаксис 'CALL «имя_хранимой_процедуры»'.

Например:

```
file{PROP:SQL} = 'CALL SelectRecordsProcedure'
```

NORESULTCALL

SQL-драйверы допускают также синтаксис 'NORESULTCALL «имя_хранимой_процедуры»' для вызова процедуры, которая не возвращает результирующего подмножества.

Для SQLAnywhere **NORESULTCALL является более эффективным, чем простой CALL. Для** ODBC **NORESULTCALL также является более эффективным, чем простой CALL и может понадобится для некоторых баз данных. Для** Oracle **NORESULTCALL это может понадобится для хранимых процедур, которые не возвращают результирующего подмножества.**

Например:

```
file{PROP:SQL} = 'NORESULTCALL GrantAccessProcedure'
```

Отладка вашего SQL-приложения.

Все SQL-драйверы TopSpeed могут создавать файлы, протоколирующие выполнение операций ввода/вывода Clarion, соответствующие операторам SQL и соответствующие коды возврата SQL.

Вы можете генерировать системный протокол и заказной протокол (условный протокол, основанный на логике вашей программы)

Системный протокол

Для ведения системного протокола вы должны добавить в WIN.INI следующие строки:

```
[CWdriver]

Profile=[1|0]

Details=[1|0]

Trace=[1|0]

TraceFile=[Pathname]
```

driver **имя драйвера базы данных ([CWORACLE], например). Ни имя секции INI, ни имя позиции INI не зависят от регистра……**

Profile=1 **сообщает драйверу о необходимости включать Clarion-операции ввода/вывода в файл - протокол. Profile=0 сообщает драйверу о том, что Clarion-операции ввода/вывода в файл протокола включать не следует. Для того, чтобы прочие переключатели имели эффект, Profile-переключатель должен быть установлен равным 1.**

Details=1 **сообщает драйверу о необходимости включать в протокол содержимое буфера записи, однако, если файл защищен кодированием, вы должны включить переключатель Details и переключатель ALLOWDETAILS на протоколирование содержимого буфера записи (см. *ALLOWDETAILS*). Details=0 сообщает драйверу о том, что протоколирование содержимого буфера записи не требуется. Для того, чтобы переключатель Details действовал, переключатель Profile должен быть включен.**

Trace=1 **сообщает драйверу о необходимости включать в протокол все обращения к файловой системе сервера, включая SQL-операторы и их коды возврата. Trace=0 предписывает пропускать эти вызовы. Переключатель Trace обычно протоколирует информацию, которая используется TopSpeed и практически бесполезна для разработчика.**

TraceFile задает имя файла протокола. Если TraceFile пропущен, драйвер записывает протокол в файл *driver*.log текущий каталог. Параметр *Pathname*- это полное имя пути или имя файла протокола. Если путь не указан, драйвер пишет протокол в текущий каталог.

Процедура протоколирования открывает файл-протокол в эксклюзивном доступе. Если файл к моменту начала протоколирования уже существует, новый записи протокола дописываются в этот файл.

Заказное протоколирование.

Для организации заказного протоколирования вы можете использовать синтаксис свойств в вашей программе для условного переключения различных уровней протоколирования. Для протоколирования будут доступны, собственно,таблицы базы данных и любые представления, для которых эти таблицы являются первичными.

```
file{PROP:Profile}=Pathname
        !Включение протоколирования операций ввода/вывода Clarion
file{PROP:Profile}=''
        ! Выключение протоколирования операций ввода/вывода Clarion
PathName = file{PROP:Profile}
        !Запрос имени файла-протокола
file{PROP:Log}=string
        !Запись строки в протокол
```

file{PROP:Details}=1!Включение возможности помещения в протокол буфера записи
File{PROP:Details}=0 Отключение возможности помещения в протокол буфера записи

Где *Pathname* полное имя пути или файла протокола. Если вы не указали путь, драйвер будет писать протокол в текущем каталоге.

Вы можете также управлять заказным протоколированием, используя команду SEND() и строку драйвера *LOGFILE*. Для дополнительной информации см. *LOGFILE*.

Обработка ошибок на уровне языка.

Вы можете использовать функции FILEERROR() и FILEERRORCODE() для перехвата сообщений и кодов возврата сервера SQL-драйверу. Для получения подробной информации см. Описание языка (*Language Reference)*.

Строки драйвера SQL

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением.

Замечание: Всем строкам драйвера SQL должен предшествовать символ '/'. Слэш позволяет драйверу различать строки драйвера и SQL-операторы, посылаемые функцией SEND().

Драйверы SQL поддерживают следующие строки драйвера.

ALLOWDETAILS

```
DRIVER('SQLDriver', '/ALLOWDETAILS = TRUE | FALSE' )
```

Строка драйвера ALLOWDETAILS разрешает драйверу включать буфер записи. Строка драйвера ALLOWDETAILS выполняется совместно с переключателем DETAILS, описанным в секции *Отладка вашего SQL-приложения*.

APPENDBUFFER

```
DRIVER('SQLDriver', '/APPENDBUFFER = size ' )
[ Buffer» = ] SEND(file, 'APPENDBUFFER [ = size ]' )
```

По умолчанию APPEND добавляет записи по одной за один раз. Для достижения лучшей производительности в сети, вы можете указать драйверу о

необходимости построения буфера записей для накопления и отсылки всех записей серверу за один прием. SIZE определяет количество записей, которое вы хотите разместить в этом буфере. Функция SEND() возвращает количество записей, которое будет размещено в буфере.

LOGFILE

```
DRIVER( 'SQLDriver', '/LOGFILE [= Pathname] [[message]]' )
[ LogFile» = ] SEND(file, '/LOGFILE [= Pathname] [[message]]' )
```

Строка драйвера *LOGFILE* включает или выключает процесс протоколирования и по требованию записывает сообщение в файл-протокол. При включении *LOGFILE* происходит запись операций ввода\вывода Clarion в указанный файл-протокол. Строка драйвера *LOGFILE* эквивалентна переключателю *Profile,* описанному в секции *Отладка вашего SQL-приложения.*

Pathname полное имя пути или файла протокола. Если вы не указали путь, драйвер будет писать протокол в текущем каталоге. Если *Pathname* пропущен, драйвер будет писать протокол в файл SQLDriver.log в текущем каталоге.

Если файл-протокол уже существует, драйвер будет писать в него, в противном случае драйвер создаст новый файл-протокол.

Параметр *message* необязательный, однако, если он указан, он должен быть заключен в квадратные скобки и открывающей скобке должен предшествовать пробел.

Замечание: /LOGFILE должен быть последней строкой драйвера специфицирующей DRIVER атрибут.

USEPRIMARY

```
DRIVER('SQLDriver', '/USEPRIMARY = TRUE | FALSE ' )
[ UsePrime» = ] SEND(file, '/USEPRIMARY [= TRUE | FALSE ]' )
```

Установкой *USEPRIMARY,* равным TRUE, вы можете принудить структуру VIEW использовать первичный ключ для вторичной сортировки. Вам это может понадобиться, если вы используете VIEW сортированным по ключу, отличному от первичного ключа или если ключ, определенный в словаре данных Clarion, отличается от индекса в базе данных на сервере.

По умолчанию (*USEPRIMARY=FALSE)* драйвер использует для упорядочивания записей только **специфицированные ключи. Следовательно, записи с некоторыми (повторяющимися) значениями ключа могут не всегда**

появляться в том же порядке. Хорошим примером этому является таблица просмотра файла *Customer (заказчик) no last name (второе имя)*, в то время когда первичным ключом является *customer number (номер заказчика)*. В этом примере два заказчика с одинаковыми вторыми именами могут не всегда появляться в той же последовательности, John Smith может иногда появляться перед Mary Smith, а иногда после Mary Smith. Для того, чтобы принудить эти записи всегда появляться в одной и той же последовательности, установите *USEPRIMARY=TRUE*.

Драйверы SQL предполагают, что любые ключи, определенные в словаре данных Clarion, соответствуют физическому индексу на сервере, и что сервер использует индекс, когда записи вызываются в ключевой последовательности. Если вы обнаруживаете, что записи в таблице просмотра появляются сдвоенными, эти предположения в данном случае оказываются ложными, и вам необходимо установить *USEPRIMARY=TRUE*.

Замечание: установка *USEPRIMARY=TRUE* может неблагоприятно сказаться на производительности.

WHERE

[Where» =] SEND (file, '/WHERE [where-clause]')

В том случае, когда ваша Clarion программа содержит оператор SET, и следующий за ним оператор NEXT или PREVIOUS SQL- драйверы автоматически строят SQL-предложение WHERE. Вы можете заставить драйвер генерировать предложение WHERE, используя строку драйвера WHERE.

Процедура SEND должна быть выполнена после оператора SET и до операторов NEXT или PREVIOUS.

Замечание: Оператор SET очищает любые предложения WHERE, установленные процедурой SEND.

Так как сгенерированный SQL-драйвером оператор SELECT не компилируется до операторов NEXT или PREVIOUS, функция SEND не посылает кода ошибки и не возвращает результата.

Например:

```
Ord       FILE,PRE(Ord),DRIVER('ODBC'),NAME('ord')
NameDate  KEY(+Ord:NameId,-Ord:Date)
Record    RECORD
Name      STRING(12),NAME('NameId')
```

```
Date        DATE,NAME('OrderDate')
Type        STRING(1),NAME('OrderType')
Details     STRING(20),NAME('OrderDetails')
    . .
 CODE
Ord:Name = 'SMITH'
SET(Ord:NameDate,Ord:NameDate)
SEND(Orders, 'WHERE OrderType = ''M'»)
LOOP
NEXT(Ord)
  !...some processing
            END
```

Генерируемый при этом код эквивалентен оператору SELECT:

```
SELECT NameId,OrderDate,OrderType,OrderDetails FROM Ord
 WHERE (NameID >= 'SMITH') AND (OrderType = 'M')
```

Свойства (property) SQL-драйвера

Вы можете использовать синтаксис свойств (property syntax) Clarion запроса и установки некоторых свойств (property)SQL-драйвера. Эти свойства описаны ниже.

PROP:Alias

PROP:Alias устанавливает или возвращает синоним SQL-драйвера, используемого когда генерируется оператор SELECT для представления содержащего целевой файл. PROP:Alias только возвращает значение предыдущей установки использовавшей PROP:Alias

```
Customer{PROP:Alias} = 'C'                    !set new table alias
OldAlias» = Customer{PROP:Alias} = ''          !use no alias
PROP:AppendBuffer
```

См. APPENDBUFFER

PROP:ConnectString

PROP\:ConnectString возвращает информацию о подключении к SQL-базе.

```
AFileOwner                          STRING(256)
AFile
FILE,DRIVER('ODBC'),OWNER(AFileOwner)
```

```
            CODE
            AFileOwner='DataSource'
            OPEN(Afile)
            IF NOT ERRORCODE()
              AFileOwner=AFile{PROP:ConnectString}
 END
PROP:Details
```

См. переключатель Details в секции *Отладка вашего SQL-приложения*

PROP:Disconnect

PROP:Disconnect закрывает все открытые файлы в базе данных на сервере и затем отключает приложение от базы данных.

PROP:Inner

PROP:Inner полезно для проверки строки драйвера ODBC. Для получения дополнительной информации см. PROP:Inner *Описании языка*.

PROP:Log

PROP:Log записывает строку в файл - протокол. Например:

```
AFile     FILE,DRIVER('ODBC'),OWNER('DataSource')

            CODE
            OPEN(Afile)
            IF NOT ERRORCODE()
            AFile{PROP:Log}='AFile opened:'&CLOCK()
            END
```

PROP:LogFile

Как PROP:Profile

PROP:OrderAllTables

Установка PROP:OrderAllTables = TRUE заставляет SQL-драйвер использовать связанные поля и поля, входящие в ключ первичного файла в посылаемом серверу предложении ORDER BY. Вам может понадобится этот переключатель, если вы используете Clarion VIEW, объединяющем несколько таблиц. По умолчанию (View{PROP:OrderAllTables}=FALSE) SQL-драйвер включает в посылаемом серверу предложении ORDER BY только компоненты первичного ключа. Например:

```
BRW1::View:Browse                    VIEW(Customer)
```

```
            PROJECT(CUST:CustNo)
            PROJECT(CUST:Name)
            PROJECT(CUST:Zip)
            PROJECT(CUST:CustNo)
            JOIN(ORD:ByCustomer,CUST:CustNo)
              PROJECT(ORD:OrderNo)
              PROJECT(ORD:OrderDate)
            END
          END
           CODE
?BRW1::View:Browse{PROP:OrderAllTables} = TRUE
```

В результате этого генерируется некий эквивалент следующего оператора SELECT

```
SELECT CustNo,Name,Zip,OrderNo,OrderDate FROM Customer,Ord
 WHERE (Customer.CustNo = Ord.CustNo)
ORDER BY CustNo,OrderNo
```

PROP:Profile

Установка PROP:Profile=TRUE указывает драйверу о необходимости включать операции ввода/вывода Clarion в файл протокол. См. описание переключателя Profile в секции *Отладка вашего SQL приложения*.

Profile=1 указывает драйверу о необходимости включать операции ввода/вывода Clarion в файл протокол; Profile=0 указывает драйверу о необходимости пропускать операции ввода/вывода. Для того, чтобы действовал переключатель Details, переключатель Profile должен быть равным 1.

Details=1 сообщает драйверу о необходимости включать в протокол содержимое буфера записи, однако, если файл защищен кодированием, вы должны включить переключатель Details и переключатель ALLOWDETAILS на протоколирование содержимого буфера записи (см. *ALLOWDETAILS*). Details=0 сообщает драйверу о том, что протоколирование содержимого буфера записи не требуется. Для того, чтобы переключатель Details действовал, переключатель Profile должен быть включен.

Замечание: /ALLOWDETAILS допускается только как параметр атрибута DRIVER (поле *Driver Options* в диалоговом окне свойства файла *(File Properties*). ALLOWDETAILS не допускается как параметр функции SEND/

SQL-Драйверы: поддерживаемые команды и функции.

Атрибуты файла	**Поддержка**
CREATE	ДА
DRIVER(*тип файла* [,*строка драйвера*])	ДА
NAME	ДА
ENCRYPT	НЕТ
OWNER(*пароль*)	ДА[1]
RECLAIM	НЕТ
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	ДА
EXTERNAL(*член*)	ДА
DLL([*флаг*])	ДА
OEM	НЕТ

Структура файла	Поддержка
INDEX	ДА
KEY	ДА
MEMO	НЕТ
BLOB	НЕТ
RECORD	ДА

Индексы, ключи, MEMO-атрибуты	Поддержка
BINARY	НЕТ[3]
DUP	ДА
NOCASE	ДА
OPT	НЕТ
PRIMARY	ДА
NAME	ДА
Ascending Components	ДА
Descending Components	ДА
Mixed Components	ДА

Атрибуты полей	Поддержка
DIM	НЕТ
OVER	ДА
NAME	ДА

Процедуры файла	Поддержка
BOF(*файл*)	НЕТ
BUFFER(*файл*)	ДА
BUILD(*файл*)	ДА
BUILD(*ключ*)	ДА
BUILD(*индекс*)	ДА[3]
BUILD(*индекс*, *компоненты*)	ДА[3]
BUILD(*index*, *components*, *filter*)	НЕТ
BYTES(*файл*)	ДА
CLOSE(*файл*)	ДА
COPY(*файл*, *новый файл*)	НЕТ
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	НЕТ
FLUSH(*файл*)	НЕТ
LOCK(*файл*)	НЕТ
NAME(*метка*)	ДА
OPEN(*файл*, *режим доступа*)	ДА
PACK(*файл*)	НЕТ
POINTER(*файл*)	НЕТ
POINTER(*ключ*)	НЕТ
POSITION(*файл*)	НЕТ
POSITION(*ключ*)	ДА
RECORDS(*файл*)	ДА
RECORDS(*ключ*)	ДА
REMOVE(*файл*)	ДА
RENAME(*файл*, *новый файл*)	НЕТ
SEND(*файл*, *сообщение*)	ДА
SHARE(*файл*, *режим доступа*)	ДА
STATUS(*файл*)	ДА
STREAM(*файл*)	НЕТ
UNLOCK(*файл*)	НЕТ

Доступ к записям	Поддержка
ADD(*файл*)	ДА
ADD(*файл*, *длина*)	НЕТ
APPEND(*файл*)	ДА

APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	ДА
GET(*файл, ключ)*	ДА
GET(*файл, указатель файла*)	НЕТ
GET(*файл, указатель файла, длина*)	НЕТ
GET(*key, указатель ключа*)	НЕТ
HOLD(*файл*)	НЕТ
NEXT(*файл*)	ДА
NOMEMO(*файл*)	НЕТ
PREVIOUS(*файл*)	ДА
PUT(*файл*)	ДА
PUT(*файл, указатель файла*)	НЕТ
PUT(*файл, указатель файла, длина*)	НЕТ
RELEASE(*файл*)	НЕТ
REGET(*файл, строка*)	НЕТ
REGET(*ключ, строка*)	ДА
RESET(*файл, строка*)	НЕТ
RESET(*ключ, строка*)	ДА
SET(*файл*)	ДА
SET(*файл, ключ*)	НЕТ
SET(*файл, указатель файла*)	НЕТ
SET(*ключ*)	ДА
SET(*ключ, ключ*)	ДА
SET(*ключ, указатель ключа*)	НЕТ
SET(*ключ, ключ, указатель файла*)	НЕТ
SKIP(*файл, число*)	ДА
WATCH(*файл*)	ДА

Обработка транзакций	Поддержка[2]
LOGOUT(*время ожидания, файл, ..., файл*)	ДА[4]
COMMIT	ДА
ROLLBACK	ДА

Обработка NULL-полей	Поддержка
NULL(*поле*)	ДА
SETNULL(*поле*)	ДА
SETNONNULL(*поле*)	ДА

Примечания

1 Мы рекомендуем использовать длинные и содержащие специальные символы значения пароля таким способом, каким обеспечивается большая эффективность защиты. Например, пароль подобный «;%?? $$^» гораздо сложнее разгадать, чем пароль типа «SALARY.»

Совет: Для определения переменной, содержащей актуальное значение пароля в поле Owner Name в диалоге «Свойства файла» (File Properties dialog), перед именем переменной наберите восклицательный знак '!'. Например: !MyPassword.

2 См также *PROP:Logout* в *Описании языка*

BUILD(index) устанавливает внутренние флаги драйвера для того, чтобы гарантировать, что драйвер сгенерировал правильное предложение ORDER BY. Драйвер при этом не обращается к серверу базы данных.

Будет ли LOGOUT блокировать таблицу, зависит от того, как настроена обработка транзакций на сервере. См. документацию вашего сервера.

ODBC - Акселератор

Обзор

ODBC акселератор - это один из нескольких SQL-драйверов TopSpeed. Эти SQL-драйверы объединяет общий код и многие общие, присущие драйверам TopSpeed уникальные достоинства: высокое быстродействие, технология буферизации, общие строки драйверов, SQL-совместимость. Для получения полной информации по этим общим характеристикам см. *Драйвер SQL*.

Драйвер ODBC преобразует стандартные операторы ввода/вывода и вызовы функций Clarion в оптимизированный SQL-оператор, который посылается на сервер для выполнения. Это означает, что вы можете использовать один и тот же код Clarion как для доступа к таблицам SQL, так и для обращения к другим файловым системам, файлам TopSpeed, например. Это также означает, что вы можете использовать код, сгенерированный по шаблонам Clarion для работы с SQL базой данных.

Драйвер ODBC лишь слегка отличается от других SQL-драйверов и, в целом, является обобщенным SQL-драйвером. Это не является обязательным для конкретного сервера, но в действительности справедливо для любого сервера, который поддерживает стандарты ODBC. В том числе для таких SQL-систем, как AS400, Informix, MSSQL, Oracle, Scalable SQL, SQL Anywhere, Sybase и многих не SQL-систем (dBase, Excel, FoxPro и т.д.). Эта глава описывает специальные вызовы и соглашения, актуальные при использовании ODBC для обращения к данным.

Все общие характеристики для всех SQL-драйверов документированы в главе *SQL-драйверы*. Индивидуальные особенности поведения ODBC-драйвера описаны в данной главе.

Замечание: Вы должны иметь Miscrosoft ODBC 2.1 или выше для 32-разрядного доступа к данным через Clarion's Database Manager или через 16-битовые приложения. Кроме того, если 32-битовые данные это данные Miscrosoft, вы должны работать под Windows NT для доступа к ним через Clarion's Database Manager или через 16-битовые приложения.

Вы можете переписать Miscrosoft ODBC 2.1 ftp.sunet.se/ftp/pub/vendor/Microsoft/developr/ODBC/public/ODBC21.exe.

Что такое ODBC

ODBC (Открытый интерфейс связи баз данных) является «стратегическим интерфейсом» Windows, предназначенным для доступа к данным из разнообразных систем управления базами данных, среди разнообразных сетей и платформ.

Стандарт ODBC был разработан и поддерживается Microsoft, который публикует ODBC Software Development Kit (SDK), и приспособлен для использования с Visual C++ продуктом. ODBC - это еще один способ, которым Clarion for Windows обеспечивает для вас расширяемую платформу для создания ваших приложений.

ODBC, за и против

Использование ODBC дает следующие преимущества:

· ODBC является прекрасным выбором в среде клиент - сервер, особенно, если сервер сродни Структурированному языку запросов (SQL) DBMS. Это дает вам возможность добавить поддержку архитектуры клиент - сервер к вашему приложению без необходимости делать что-либо помимо выбора драйвера файла. ODBC был специально спроектирован для создания непродаваемого специфического метода связи приложений с сервером. Посредством ODBC сервер может выполнить большую часть работы, особенно для операций SQL JOIN и PROJECT, ускоряя тем самым работу вашей программы.

· Существующие драйверы ODBC учитывают многие типы баз данных. Однако также существуют драйверы ODBC для баз данных, для которых Clarion может не иметь своего драйвера - например, для файлов Microsoft Excel и Lotus Notes.

· ODBC уже получил широкое распространение. Большие прикладные пакеты, такие, как Microsoft Office, например, инсталлируют драйверы ODBC для таких форматов файлов, как dBase и Microsoft Access. Помните, что многие внутренние ODBC драйверы усовершенствованы, и вам следует получить самые последние выпуски.

· ODBC не зависит от платформы. Одной из первейших целей Microsoft при установлении ODBC было поддержать более легкий доступ к системам наследства или корпоративным средам, где данные находятся на различных платформах или многочисленных DBMS. Когда имеются драйвер ODBC, не имеет значения,

используете ли вы продукты Microsoft NetBEUL, SPX/IPX, DECNet или иные; ваше приложение может связаться с DBMS и получить доступ к данным.

Если имеется много драйверов, а стандарт был разработан компанией, которая разработала Windows, вы можете рассмотреть использование ODBC, как выбранного драйвера для всех ваших приложений Windows. Тем не менее, при выборе использования драйвера ODBC вместо собственного драйвера базы данных Clarion for Windows, вы должны также учесть возможные невыгодные моменты:

· ODBC добавляет свой собственный дополнительный слой - диспетчер драйвера ODBC - между вашим приложением и базой данных. Когда вы получаете доступ к файлам на локальном жестком диске, это обычно приводит к более медленной работе. Диспетчер драйвера должен транслировать обращение ODBC API приложения к оператору SQL прежде, чем осуществить доступ к данным.

· ODBC использует SQL для связи с внутренней базой данных. Хотя это может быть очень эффективно при связи с процессорами базы данных Клиент/сервер, это обычно менее эффективно, чем прямой доступ к записи при использовании файловой системы, сконструированной для доступа к одиночной записи, такой, как xBase или Btrieve.

· Администратор ODBC, манипулируя или регистром Windows для 32-разрядных приложений или ODBC.ini для 16-разрядных, может собирать системы, которые работают как под 16, так и под 32-разрядными операционными системами.

· Информация, которая требуется диспетчеру базы данных ODBC для связывания с источником данных, изменяется при переходе от одного драйвера ODBC к другому. В отличие от выбора драйверов файла Clarion, где операции фактически прозрачны, вам может потребоваться выполнить некоторую работу, чтобы собрать информацию, требуемую для использования конкретного драйвера ODBC. В этой главе приводится ряд советов, которые позволят сделать это проще. Многие драйверы ODBC приходят с файлом .HLP, который документирует специальные установки (обычно хранящиеся в ODBC.INI); но на вас возлагается определенная ответственность при решении ваших проблем с помощью драйверов ODBC, поставляемых третьими сторонами.

· Драйверы ODBC не включены в Windows. При распространении вашего приложения вам нужно инсталлировать в систему конечного пользователя драйверы ODBC и диспетчер драйвера ODBC, если, конечно, он их еще не имеет.

Для этого нужно получить от Microsoft ODBC SDK. В некоторых случаях на сервере уже может быть установлен комплект распределения, который инсталлирует драйвер ODBC на рабочую станцию.

· Нормальная программа запуска Microsoft, которая инсталлирует диспетчер драйвера ODBC, добавляет applet к окну панели управления конечного пользователя для управления ODBC. Конечному пользователю очень удобно использовать этот инструмент для изменения установок в файле ODBC.INI.(только для 16-разрядов). Конечный пользователь может непреднамеренно удалить конечный драйвер ODBC, что сделает невозможным для вашего приложения связаться с файлом данных.

Учитывая все “за” и “против”, мы рекомендуем использование собственных драйверов файлов Clarion for Windows в тех случаях, когда для одного и того же формата файла существуют как собственный драйвер, так и драйвер ODBC.

Как работает ODBC

Когда вы используете ODBC, чтобы получить доступ к данным, для успешной работы требуется взаимодействие четырех компонент:

· Ваше приложение вызывает диспетчер драйвера ODBC и посылает ему соответствующие запросы данных через ODBC.API.

Clarion for Windows делает это для вас открыто, используя либо CW2ODBC16.DLL (16-битовая) либо CW2ODBC32.DLL расширения приложения. При ручном кодировании не забудьте включить эту библиотеку в проект. При распределении вашего приложения не забудьте включить этот файл вместе с вашим файлом .EXE (если вы не используете полностью укомплектованный .EXE).

· Диспетчер драйвера ODBC получает вызов API, проверяет ODBC.INI относительно информации об источнике данных, затем загружает «внутренний драйвер» драйвер ODBC.

Реальным «интерфейсом» для диспетчера драйвера является файл, называемый ODBCADM.EXE, который программа установки Microsoft помещает в каталог \Windows\System. Это администратор ODBC, который затем загружает другие библиотеки для работы.

· «Выходной» драйвер ODBC - это другая библиотека (.DLL), которая содержит исполняемый код, осуществляющий доступ к данным.

«Внутренние» драйверы поставляются различными источниками. Например,

корпорация Lotus Development поставляет драйвер ODBC для Lotus Notes. Microsoft Office распространяет ODBC SDK, содержащий драйверы для большинства их продуктов базы данных.

· Источник данных - это файл данных (обычно ODBC используется для доступа к локальным данным), или удаленная DBMS, такая как база данных Oracle 7.

Источник данных имеет описательное имя; например, «Microsoft Access Databases». Это имя служит именем раздела в файле ODBC.INI.

Диспетчер драйвера ODBC должен знать точное имя источника данных, чтобы он мог загрузить правильный драйвер для доступа к данным. Следовательно, жизненно важно, чтобы вы знали точное имя драйвера.

Типы данных ODBC

		Clarion Data Types											
ODBC Data Types		STRING	CSTRING	BYTE	SHORT	LONG	ULONG	SREAL	REAL	DECIMAL	PDECIMAL	DATE	TIME
M	CHAR	C	•	•	•	•	•	•	•	•	•		
I	VARCHAR	•	C	•	•	•	•	•	•	•	•		
N	LONG VARCHAR	•	•										
	DECIMAL		•	•1	•1	•1		•4	•4		•		
	NUMERIC		•	•1	•1	•1		•4	•4	•			
C	SMALLINT		•	•3	C5	•3	•3						
O	INTEGER		•	•3	•3	C5	•3						
R	REAL		•					C	•3	•4	•4		
E	FLOAT		•					•3	•	•4	•4		
	DOUBLE PRECISION		•					•3	C	•4	•4		
E	BIT		•	•	•3	•3	•3						
X	TINYINT		•	C5	•3	•3	•3						
T	BIGINT		•		•3	•3	•3	•3	•3	•	•		
E	BINARY	•											
N	VARBINARY	•											
D	LONG VARBINARY	•											
E	DATE											C	
D	TIME												C
	TIMESTAMP	•2											

Замечания

C **Тип данных Clarion может быть использован для управления типом данных ODBC. CREATE создает тип данных ODBC.**

· Тип данных Clarion может быть использован для управления типом данных ODBC, однако CREATE НЕ создает тип данных ODBC.

1 LONG, SHORT и BYTE из Clarion'а могут быть использованы с типами данных ODBC DECIMAL и NUMERIC, если поле ODBC не имеет десятичных мест.

2 Полями ODBC TIMESTAMP можно управлять с помощью STRING(8), с последующей GROUP поверх этого, которая содержит только поле DATE в поле TIME.

Пример:

```
TimeStampField          STRING(8),NAME('TimeStampField')
TimeStampGroup          GROUP,OVER(TimeStampField)
TimeStampDate             DATE
TimeStampTime             TIME
                        END
```

CREATE создает поле TIMESTAM, если вы используете подобную структуру.

3 Может возникнуть некоторая потеря точности.

4 Могут произойти ошибки округления.

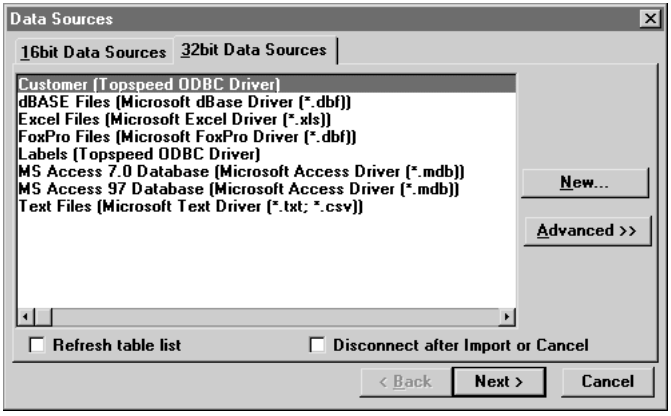
5 CREATE пытается создать TINYINT для BYTE. Если база данных не поддерживает TINYINT, CREATE обрабатывает BYTE как SHORT. CREATE пытается создать SMALLINT для SHORT. Если база данных не поддерживает SMALLINT, CREATE обрабатывает как LONG. CREATE пытается создать IN TEGER для LONG. Если база данных не поддерживает INTEGER, CREATE создает десятичное поле.

Замечание: Ваша внутренняя база данных может содержать типы данных, которые не перечислены здесь. Эти типы данных преобразуются к данным типа ODBC внутренней базой данных. Просмотрите вашу документацию по внутренней базе данных для того, чтобы определить, какой тип данных ODBC используется.

Импорт из ODBC.

Мастер импорта Редактора словаря данных Clarion позволяет вам импортировать определения таблиц в ваш словарь данных.

Когда вы выбираете ODBC-драйвер из выпадающего списка драйверов, Мастер импорта открывает диалоговое окно Data Sources. **Выберите источник данных и затем нажмите кнопку** NEXT **для импорта определения этих данных.**



Если источник данных не определен, вы можете добавить его, нажав кнопку New **и затем, следуя инструкциям ODBC, предоставляемым файловой системой, вы получите доступ к базе.**

После того, как вы выбрали источник данных, нажмите кнопку NEXT **для импорта определения этих данных. Нажатие кнопки NEXT импортирует определение таблицы и открывает диалоговое окно** File Properties, **позволяющее вам при необходимости модифицировать атрибуты файла.**

Информация для подсоединения к базе данных и конфигурация драйвера - Свойства файла

Обычно вы добавляете SQL поддержку к вашему приложению, импортируя определения SQL или ODBC таблиц в ваш Словарь данных. Мастер импорта автоматически заполняет диалоговое окно File Properties **некоторыми значениями, принимаемыми по умолчанию и основанными на свойствах импортируемых таблиц. Однако в диалоговом окне** File Properties **имеется несколько полей, которые вы можете использовать для дальнейшего конфигурирования способа, которым Scalable SQL драйвер будет связываться с базой данных. Эти поля и их использование описаны ниже.**

Имя владельца

Обычно Мастер импорта размещает информацию, необходимую для подключения к серверу (имя сервера, имя пользователя и т.д.) в поле Owner Name **(имя владельца).**

Некоторые базы данных могут требовать дополнительной информации, которую вы можете предоставить через поле Owner Name **(имя владельца). Эта информация должна следовать за паролем, быть разделена точками с запятой и удовлетворять синтаксису:** ***keyword=value;keyword=value*****.**

Например, для доступа к базе данных SYBASE через ODBC-драйвер:

```
DataSource,UserID,PassWord,DATABASE=DataBaseName;APP=APPName
```

Обращайтесь к вашей документации по SQL-серверу за информацией по этим ключевым словам, их использованию и действию.

Конфигурирование ключа—Свойства ключа

Обычно вы добавляете SQL поддержку к вашему приложению, импортируя определения SQL или ODBC таблиц в ваш Словарь данных. Мастер импорта автоматически заполняет диалоговое окно Key Properties **некоторыми значениями, принимаемыми по умолчанию и основанными на свойствах импортируемых таблиц. Однако в диалоговом окне** Key Properties **имеется несколько полей, которые вы можете использовать для дальнейшего конфигурирования способа, которым ODBC-драйвер будет связываться с базой данных. Эти поля и их использование описаны ниже.**

Внешнее имя

READONLY

Добавляя переключатель READONLY к External Name **(Внешнее имя), вы указываете ODBC-драйверу, что не надо вставлять поля при добавлении записей . Это необходимо для некоторых серверов (например Watcom), которые не допускают автонумеруемых ключей, установленных в ноль. Некоторые серверы не допускают нулевых автонумеруемых ключей, но они пропускают уведомление ODBC-драйвера об этом. Переключатель READONLY позволяет вам вручную сделать это.**

Конфигурация колонок - Свойства файла

Обычно вы добавляете SQL поддержку к вашему приложению, импортируя определения SQL или ODBC таблиц в ваш Словарь данных. Мастер импорта автоматически заполняет диалоговое окно Field Properties **некоторыми значениями, принимаемыми по умолчанию и основанными на свойствах импортируемых таблиц. Однако в диалоговом окне** Field Properties **имеется**

несколько полей, которые вы можете использовать для дальнейшего конфигурирования способа, которым ODBC-драйвер будет связываться с базой данных. Эти поля и их использование описаны ниже.

Внешнее имя

NOWHERE

Добавляя переключатель NOWHERE к External Name **(Внешнее имя), вы указываете ODBC-драйверу о необходимости исключать поле из любого предложения WHERE ,которое посылается серверу. Это необходимо для ряда серверов, когда включен режим WATCH. Некоторые серверы не допускают определенных типов данных в предложении WHERE, но они не уведомляют драйвер ODBC об этом. Переключатель NOWHERE позволяет вам вручную уведомить драйвер об этих ограничениях.**

Отладка вашего ODBC-приложения

Когда вы используете ODBC-драйвер, ODBC-администратор может создать файл-протокол, документирующий все вызовы, сделанные ODBC-драйвером. К ним относятся SQL-операторы, обращающиеся через ODBC-драйвер к данным и сообщения о любых ошибках. Так как такое протоколирование значительно замедляет выполнение вашей программы, его следует выполнять только во время тестирования. Кроме того, так как файл-протокол может чрезвычайно быстро расти в размерах, вам необходимо отключать протоколирование для удаления протокола.

Кроме «сования носа» в действующие операторы SQL, генерируемые драйвером, вы можете установить на нуль любые ошибки. Если приложение не могло связаться с внешней базой, вы можете открыть файл-протокол. Прокрутите его до тех пор, пока вы не обнаружите слово «SQLError» (ошибка).

Для получения дополнительной информации по администрированию протокола ODBC см. Microsoft документацию по ODBC.

Строки драйвера ODBC

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть

посланы драйверу файла во время работы для управления его поведением. Для получения обзорной информации об этих переключателях и параметрах времени исполнения см. *Common Driver Features—Driver Strings.*

Замечание: Всем строкам драйвера SQL должен предшествовать символ '/'. Символ '/' позволяет драйверу различать строки драйвера и SQL-операторы, посылаемые функцией SEND().

В дополнение к стандартным строкам драйвера, SQL ODBC-драйвер поддерживает следующие строки драйвера.

CLIPSTRINGS

```
DRIVER('ODBC', '/CLIPSTRINGS = TRUE | FALSE ' )
[ Clipped» = ] SEND(file, '/CLIPSTRINGS [ = TRUE | FALSE ]' )
```

По умолчанию (CLIPSTRINGS=TRUE), ODBC-драйвер отрезает хвостовые пробелы в строках, посылаемых серверу. (см. функцию *CLIP* в описании языка). Для посылки полных строк установите CLIPSTRINGS=FALSE.

FORCEUPPERCASE

```
DRIVER('ODBC', '/FORCEUPPERCASE = TRUE | FALSE ' )
 [ Uppered» = ] SEND(file, '/FORCEUPPERCASE [ = TRUE | FALSE ]' )
```

По умолчанию (FORCEUPPERCASE=FALSE), ODBC-драйвер передает имена таблиц функции SQLColumns для проверки на существование в базе данных в смешанном регистре. Однако некоторые серверы требуют имена таблиц в верхнем регистре. Для передачи имен таблиц в верхнем регистре установите FORCEUPPERCASE=FALSE. См. также *VERIFYVIASELECT*

JOINTYPE

```
        DRIVER('ODBC', '/JOINTYPE = Watcom | DB2 | Microsoft | FirstSQL | Inner
| None' )
        [ Join» = ] SEND(file, '/JOINTYPE [ = Watcom | DB2 | Microsoft | FirstSQL |
Inner | None ]' )
```

Стандарт ODBC 2.1 не поддерживает объединение более одного дочернего (или родительского) файла. Большинство продавцов не принимают эти ограничения расширяют стандарт. Однако они это делают различными способами. ODBC-драйвер пытается определить тип объединения, используя сервер, но если это не дает результата, вы должны использовать строку драйвера JOINTYPE для первичного файла в представлении. Заметим, что *Inner* обычно медленнее, чем

Watcom, DB2, Microsoft или *FirstSQL* и *None* медленнее, чем *Inner,* но будет работать со всеми внутренними базами, так как объединение должно быть сделано клиентом.

Стандарт ODBC 3.0 поддерживает множественное объединение и, поэтому, более гибкий ODBC 3.0 не требует этого переключателя.

NESTING

```
DRIVER('ODBC', '/NESTING = TRUE | FALSE ' )
[ Nest» = ] SEND(file, '/NESTING [ = TRUE | FALSE ]' )
```

Некоторые ODBC-драйверы не поддерживают представления типа «родитель>ребенок>внук». Драйвер ODBC пытается определить, возможно ли такое представление. Если это ему не удается и внутренняя база данных не поддерживает такого объединения, вам необходимо установить NESTING=FALSE. В этом случае объединение должно быть сделано клиентом.

USEINNERJOIN

```
DRIVER('ODBC', '/USEINNERJOIN= TRUE | FALSE' )
[ Join» = ] SEND(file, '/USEINNERJOIN [ = TRUE | FALSE ]' )
```

По умолчанию (USEINNERJOIN = True) драйвер ODBC генерирует следующее внутреннее объединение (ANSI стандарт SQL)

```
SELECT ... FROM table1 INNER JOIN table2 ON table1.field=table2.field
```

Однако ANSI SQL предоставляет альтернативный синтаксис для внутренних объединений. Для генерации в альтернативном синтаксисе установите USEINNERJOIN=FALSE

```
SELECT ... FROM table1, table2 WHERE table1.field=table2.field
```

VERIFYVIASELECT

```
DRIVER('ODBC', '/VERIFYVIASELECT = TRUE | FALSE' )
[ Verify» = ] SEND(file, '/VERIFYVIASELECT [ = TRUE | FALSE ]' )
```

VERIFYVIASELECT позволяет ODBC-драйверу использовать альтернативный, иногда более быстрый, метод для проверки полей при открытии таблиц. По умолчанию (VERIFYVIASELECT=FALSE), драйвер ODBC использует функцию SQLColumns для проверки полей. Однако некоторые серверы (в частности SQL Anywhere) могут проверять поля быстрее, используя оператор SELECT. Для проверки полей с использованием оператора SELECT установите VERIFYVIASELECT = TRUE.

Для сервера SQL Anywhere VERIFYVIASELECT = TRUE принято по умолчанию.

ZEROISNULL

```
DRIVER('ODBC', '/ZEROISNULL = TRUE | FALSE' )
[ Nulls» = ] SEND(file, '/ZEROISNULL [ = TRUE | FALSE ]' )
```

ZEROISNULL позволяет драйверу ODBC устанавливать поля DATE и TIME в ноль (0) быстрее, чем в NULL. По умолчанию (ZEROISNULL=TRUE), драйвер ODBC подразумевает поля DATE и TIME с нулевыми (0) значениями должны иметь во внутренней базе данных значение NULL, и преобразует их к NULL при записи в базу данных. Для того, чтобы разрешить драйверу устанавливать поля DATE и TIME в (0) прежде, установите ZEROISNULL = FALSE.

Свойства драйвера

Вы можете использовать синтаксис свойств Clarion для запросов и установки некоторых свойств ODBC-драйвера. В дополнение к стандартным свойствам SQL-драйвера, драйвер ODBC поддерживает следующие свойства.

PROP:hdbc

PROP:hdbc возвращает текущий hdbc используемый драйвером ODBC. Таким образом, ?MyFile{PROP:hdbc} может быть использован для вызова ODBC API требуемого hdbc.

PROP:henv

PROP:henv возвращает текущий henv, используемый драйвером ODBC. Таким образом, ?MyFile{PROP: henv } может быть использован для вызова ODBC API, требуемого henv. Например, функция SQLDescribeCol:

```
rc# = SQLDataSources(Myfile{PROP:henv},SQL_FETCH_NEXT,ODBC:driver, |
drvLen,drvlen,ODBC:Description,desclen,desclen)
```

PROP:hstmt

PROP:hstmt возвращает текущий hstmt, используемый драйвером ODBC. Таким образом, ?MyFile{PROP: hstmt } может быть использован для вызова ODBC API, требуемого hstmt. Например, функция SQLDescribeCol:

```
Myfile{PROP:SQL} = 'Select * from ATable'
rc# = SQLDescribeCol(Myfile{PROP:hstmt},Num,Name,Max,NameL, |

Type,Def,Scale,Null)
```

PROP:LoginTimeout

PROP:LoginTimeout устанавливает предел времени в секундах для окна подключения к базе данных SQL. Если пользователь не отвечает в отведенное время, подсоединение к базе данных отвергается и процедура подключения заканчивается аварийно. По умолчанию это время ожидания ввода пользователя. Некоторые серверы не поддерживают эту возможность и могут игнорировать данную инструкцию.

Например:

```
AFile    FILE,DRIVER('ODBC'),OWNER('DataSource')
         CODE
         OPEN(Afile)
         IF NOT ERRORCODE()
         AFile{PROP:LoginTimeOut}=60 !allow 1 minute for login
         END
```

PROP:LogonScreen

PROP:LogonScreen устанавливает и возвращает значение переключателя, определяющего будет ли драйвер автоматически запрашивать информацию, необходимую для подключения к серверу. По умолчанию (PROP:LogonScreen=1), драйвер открывает это окно. Для подавления автоматического открытия этого окна установите PROP:LogonScreen=0. Если строка соединения не поставляется, оператор OPEN не выполняется и FILEERRORCODE() возвращает '08001' или '28000.'

Например:

```
AFile    FILE,DRIVER('ODBC'),OWNER('DataSource')
         CODE
         AFile{PROP:LogonScreen}=0
         OPEN(Afile)
```

PROP:QuoteString

PROP:QuoteString устанавливает или возвращает ограничитель имени колонки (обычно это запятая), используемое ODBC-драйвером для окружения имен в генерируемых операторах SQL. Различные серверы требуют различных символов ограничителей.

Вы можете использовать PROP:QuoteString для построения ваших собственных динамических выражений SQL. Заметим, что вы должны заключать любые имена колонок, которые также являются зарезервированными словами SQL в принятые символы-ограничители.

Некоторые серверы возвращают символы-ограничители некорректно. Для этих серверов вы должны устанавливать значение PROP:QuoteString до их использования.

Команды файлов и функции

Атрибуты файла	Поддержка
CREATE	ДА
DRIVER(*тип файла[,строка драйвера*])	ДА
NAME	ДА
ENCRYPT	НЕТ
OWNER(*пароль*)	Y[2]
RECLAIM	НЕТ
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	Y[6]
EXTERNAL(*член*)	ДА
DLL([*флаг*])	ДА
OEM	НЕТ[3]

Структура файла	Поддержка
ИНДЕКС	Y[3]
КЛЮЧ	ДА[3]
MEMO	НЕТ
BLOB	НЕТ
RECORD	ДА

Атрибуты Индекс, Ключ, Memo	Поддержка
BINARY	N[7]
DUP	ДА
NOCASE	ДА
OPT	НЕТ
PRIMARY	ДА
NAME	ДА
Ascending Components	ДА
Descending Components	ДА
Mixed Components	ДА

Атрибуты полей	Поддержка
DIM	НЕТ
OVER	ДА
NAME	ДА

Процедуры файла	Поддержка
BOF(*файл*)	НЕТ
BUFFER(*файл*)	ДА
BUILD(*файл*)	ДА
BUILD(*ключ*)	ДА
BUILD(*индекс*)	ДА[8]
BUILD(*индекс*, *компоненты*)	ДА[8]
BUILD(*индекс*, *компоненты*, *фильтр*)	НЕТ
BYTES(*файл*)	ДА
CLOSE(*файл*)	ДА
COPY(*файл*, *новый файл*)	НЕТ
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	НЕТ
FLUSH(*файл*)	НЕТ
LOCK(*файл*)	НЕТ
NAME(*метка*)	ДА
OPEN(*файл*, *режим доступа*)	ДА
PACK(*файл*)	НЕТ
POINTER(*файл*)	НЕТ
POINTER(*ключ*)	НЕТ
POSITION(*файл*)	НЕТ
POSITION(*ключ*)	ДА
RECORDS(*файл*)	ДА
RECORDS(*ключ*)	ДА
REMOVE(*файл*)	ДА
RENAME(*файл*, *новый файл*)	НЕТ
SEND(*файл*, *message*)	ДА
SHARE(*файл*, *режим доступа*)	ДА
STATUS(*файл*)	ДА
STREAM(*файл*)	НЕТ
UNLOCK(*файл*)	НЕТ

Доступ к записям	**Поддержка**
ADD(*файл*)	ДА
ADD(*файл, длина*)	НЕТ
APPEND(*файл*)	ДА
APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	ДА
GET(*файл,ключ)*	ДА
GET(*файл, указатель файла*)	НЕТ
GET(*файл, указатель файла, длина*)	НЕТ
GET(*ключ, указатель ключа*)	НЕТ
HOLD(*файл*)	НЕТ
NEXT(*файл*)	ДА
NOMEMO(*файл*)	НЕТ
PREVIOUS(*файл*)	Y[4]
PUT(*файл*)	ДА
PUT(*файл, указатель файла*)	НЕТ
PUT(*файл, указатель файла, длина*)	НЕТ
RELEASE(*файл*)	НЕТ
REGET(*файл,строка*)	НЕТ
REGET(*ключ,string*)	ДА
RESET(*файл,строка*)	НЕТ
RESET(*ключ,строка*)	ДА
SET(*файл*)	Y[4]
SET(*файл, ключ*)	НЕТ
SET(*файл, указатель файла*)	НЕТ
SET(*ключ*)	ДА
SET(*ключ, ключ*)	ДА
SET(*ключ, указатель ключа*)	НЕТ
SET(*ключ, ключ, указатель файла*)	НЕТ
SKIP(*файл, count*)	ДА
WATCH(*файл*)	ДА

Обработка транзакций	**Поддержка**[5]
LOGOUT(*время ожидания, файл, ..., файл*)	Y[9]
COMMIT	ДА
ROLLBACK	ДА

Обработка Null полей	Поддержка
NULL(*поле*)	ДА
SETNULL(*поле*)	ДА
SETNONNULL(*поле*)	ДА

Примечания

1 ODBC-драйвер Clarion поддерживает список элементов, однако незапланированная файловая система может не поддерживать все эти элементы.

2 Мы рекомендуем использовать длинные и содержащие специальные символы значения пароля так, как чтобы обеспечивалась наибольшая эффективность защиты. Например, пароль, подобный «;%?? $$^» гораздо сложнее разгадать, чем пароль типа «SALARY.»

Совет: Для определения переменной, содержащей актуальное значение пароля в поле Owner Name в диалоге «Свойства файла» (File Properties dialog) перед именем переменной наберите восклицательный знак '!'. Например: !MyPassword.

Предполагается, что международная сортировка выполняется на основе, заложенной в файловой системе. Атрибут OEM и файл .ENV игнорируются.

4 PREVIOUS не поддерживается в физическом порядке

5 См. также *PROP:Logout* в *Описании языка*

6. Файлы с атрибутом THREAD не требуют дополнительных блоков управления для каждого процесса, получающего доступ к этим файлам.

8 BUILD(index) устанавливает внутренние флаги драйвера для того, чтобы гарантировать, что драйвер сгенерировал правильное предложение ORDER BY. Драйвер при этом не обращается к серверу базы данных.

9 Будет ли LOGOUT блокировать таблицу зависит от того, как настроена обработка транзакций на сервере. См. документацию вашего сервера.

Microsoft Access и ODBC

ODBC-драйвер который поставляется совместно с Access 2.0

ODBC-драйвер, который поставляется совместно с Access 2.0, работает только с другими апплетами Microsoft Office. Чтобы получить драйвер общего назначения, работающий с Clarion for Windows, вам нужно приобрести у Microsoft драйвер ODBC Kit v2.

Доступ к 32-разрядным данным ODBC из 16-разрядный приложений.

Чтобы получить доступ к 32-битовым ODBC драйверам, таким, как драйвер Access 7.0 из 16-битового приложения, вам следует использовать ODBC.DLL, версию 2.10 или более позднюю.

Замечание: Вы должны иметь Miscrosoft ODBC 2.1 или выше для 32-разрядного доступа к данным через Clarion's Database Manager или через 16-битовые приложения. Кроме того, если 32-битовые данные это данные Miscrosoft, вы должны работать под Windows NT для доступа к ним через Clarion's Database Manager или через 16-битовые приложения.

Вы можете переписать Miscrosoft ODBC 2.1 ftp.sunet.se/ftp/pub/vendor/Microsoft/developr/ODBC/public/ODBC21.exe.

Конфигурируемые сообщения об ошибках.

Все драйверы баз данных TopSpeed's посылают коды завершения и сообщения, доступ к которым осуществляется при помощи функций ERRORCODE(), ERROR(), FILEERRORCODE() и FILEERROR() (См. Описание языка). Драйвер посылает код немедленно после завершения каждой операции ввода/вывода (OPEN, NEXT, GET, ADD, DELETE, и т.д.).

Совет: Функция ERRORFILE() возвращает имя файла или таблицы, которая выдала ошибку.

См. Ошибки времени выполнения (*Run Time Errors)* в описании языка, где приведено описание кодов ошибок и их соответствие сообщениям, выдаваемым функцией ERROR(). Текст сообщения, выдаваемый функцией ERROR(), может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Для получения дополнительной информации см. «Интернационализация» (*Internationalization), CLAMSG,* и *LOCALE* в описании языка.

В дополнение, текст сообщений FILEERROR() DBase IV драйвера также может быть изменен при помощи .ENV файла секция (CLAMSG) и процедуры LOCALE. Однако значение CLAMSG должно быть равно величине, возвращаемой функцией FILEERRORCODE() + 4000.

```
100   4100    ODBC.DLL Could Not Be Loaded
```

Scalable SQL Accelerator

Сервер Scalable SQL

Для получения исчерпывающей информации по СУБД Scalable SQL обратитесь, пожалуйста, к документации Pervasive Software

Драйвер Scalable SQL

Драйвер Scalable SQL один из нескольких TopSpeed SQL-драйверов. Эти SQL-драйверы объединяет общий код и многие общие, присущие драйверам TopSpeed, уникальные достоинства: высокое быстродействие, технология буферизации, общие строки драйверов, SQL-совместимость. Для получения полной информации по этим общим характеристикам см. *Драйвер SQL*.

Драйвер Scalable SQL преобразует стандартные операторы ввода/вывода и вызовы функций Clarion в оптимизированный SQL-оператор, который посылается на сервер для выполнения. Это означает, что вы можете использовать один и тот же код Clarion для доступа к таблицам SQL или для обращения к другим файловым системам, файлам TopSpeed например. Это также означает, что вы можете использовать код, сгенерированный по шаблонам Clarion для работы с SQL базой данных.

Все общие характеристики для всех SQL-драйверов документированы в главе *SQL-драйверы*. Индивидуальные особенности поведения Scalable SQL -драйвера описаны в данной главе.

Мастер импорта SQL - диалог регистрации (Login Dialog)

Мастер импорта Clarion-редактора словаря данных позволяет вам импортировать таблицы Scalable SQL в ваш Clarion словарь данных. Когда вы выбираете драйвер SQL из выпадающего списка, мастер импорта открывает диалоговое окно Login/Connection. **Диалог** Login/Connection **собирает информацию, необходимую для связи с Scalable SQL-базой данных.**

Замечание: Для того, чтобы вы могли связаться с SQL-базой данных и импортировать определение таблицы, база данных должна быть запущена и должна быть доступна для вашего компьютера.

Заполните поля в диалоговом окне Login/Connection:

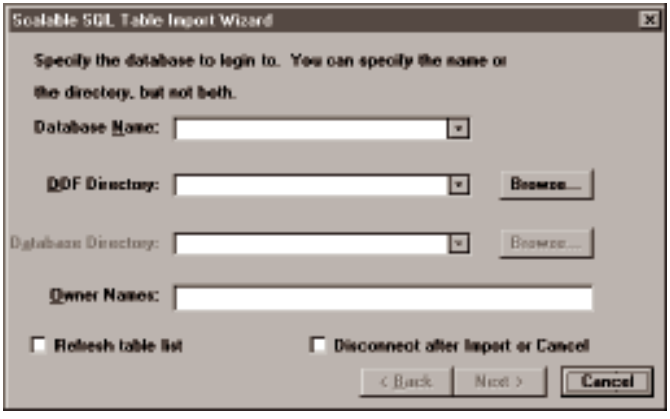
Database Name Выберите Scalable SQL базу данных, которая содержит таблицы, предназначенные для импорта. Если список **Data base Name** пуст, вы можете ввести имя с клавиатуры. **Информацию о том, как база данных специфицирована на сервере вы можете найти в документации по серверу. Спецификации базы данных могут зависеть от расположения сервера (удаленный или локальный) и от типа сетевого протокола (TCP/IP, IPX, и т.д.), используемого для доступа к данным.**

DDF Каталог Нажмите кнопку **Browse** для выбора пути или каталога, содержащего DDF-файл базы данных.

Database Directory

Нажмите кнопку **Browse** для выбора пути или каталога, содержащего базу данных.

Замечание: Определяйте Database Name или DDF каталог, но не и то и другое.



Owner Names (Имя владельца)

Если необходимо, введите разделенный запятыми список имен, используемый Scalable SQL драйвером при открытии зашифрованных файлов. Если имена содержат запятую или пробел они должны быть заключены в одиночные кавычки.

Refresh table list (Обновление списка таблиц)

Отметьте это поле для обновления списка таблиц, импортируемых при нажатии кнопки **NEXT**. Уберите эту

отметку для увеличения производительности, если вы уверены, что состав базы данных не изменился с момента последнего импорта.

Disconnect after Import or Cancel
(Отключение от базы после импорта или Cancel)
Отметьте это поле для отключения от сервера после импорта. Обычно вы должны очистить это поле, когда импортируете несколько таблиц для того, чтобы поддержать ваше соединение с сервером между импортами.

NEXT Нажмите эту кнопку для открытия диалогового окна Мастера импорта **Import List**.

SQL Мастер импорта - диалоговое окно Import List

Когда вы нажимаете кнопку NEXT >, **Мастер импорта открывает диалоговое окно** Import List. **Диалоговое окно** Import List **содержит список импортируемых таблиц.**

Отметьте в списке таблицы предназначаемые для импорта и затем нажмите кнопку Finish**. Мастер импорта добавит определение файла в ваш словарь данных и затем откроет окно** File Properties **для того, чтобы вы могли изменить установки, сделанные по умолчанию.**

Импорт дополнительных таблиц повторяет шаги, описанные выше. После того, как все необходимые таблицы импортированы, вернитесь в редактор словаря, где вы сможете установить отношения между таблицами и удалить некоторые колонки, не используемые вашим Clarion-приложением. См. *SQL-драйвер - Определение только необходимых полей*.

Информация для подсоединения к базе данных и конфигурация драйвера - Свойства файла

Обычно вы добавляете Scalable SQL поддержку к вашему приложению, импортируя определения таблиц в ваш Словарь данных. Мастер импорта автоматически заполняет диалоговое окно File Properties **некоторыми значениями, принимаемыми по умолчанию и основанными на свойствах импортируемых таблиц. Однако вы можете использовать поле** Owner Name **в окне** File Properties **для дальнейшего конфигурирования способа, которым Scalable SQL драйвер будет**

связываться с базой данных.

Scalable SQL допускает некоторую дополнительную информацию для идентификации базы данных в поле Owner Name**. Эта информация появляется альтернативно как:**

```
Database[,Owners;Switches]
        или
DDF=DDFPath[|Datapath][,Owners;Switches]
```

Где *Database* - имя базы данных Scalable SQL. *DDFPath* - путь к набору DDF (словарь данных Btrieve) файлов. *Datapath* - путь к соответствующим файлам данных. По умолчанию *Datapath= DDFPath*. *Owners* - разделенный запятыми список имен, по которому открываются зашифрованные Btrieve файлы. Если имя содержит запятую или пробел оно должно быть заключено в одиночные кавычки.

```
CREATEDDF=[0|1|2]
```

Где 0 создает новый DDF файл, 1 замещает существующий DDF файл и 2 удаляет существующий DDF файл.

Замечание: Переключатель CREATEDDF предоставляется для начальной установки для того, чтобы создать новый DDF файл. Вы никогда не должны использовать его при работе с существующей базой данных.

Свойства драйвера

Вы можете использовать синтаксис свойств Clarion для запроса и установки определенных свойств драйвера Scalable SQL. В дополнение к стандартным свойствам SQL-драйверов, драйвер Scalable SQL поддерживает следующие свойства.

PROP:LogonScreen

PROP:LogonScreen **устанавливает или возвращает значение переключатель, который определяет, будет ли драйвер автоматически выдавать окно для ввода информации, необходимой для подключения к серверу. По умолчанию (PROP:LogonScreen=1) драйвер показывает это окно, если это необходимо (если строка соединения с сервером - имя сервера, имя пользователя и пароль - не поставляется). Для подавления автоматического вывода этого окна установите PROP:LogonScreen=0. Если строка соединения не поставляется, оператор OPEN**

не выполняется и FILEERRORCODE() возвращает '08001' или '28000.'

Например:

```
AFile    FILE,DRIVER('SSQL')
         CODE
         AFile{PROP:LogonScreen}=0
         OPEN(Afile)
```

Scalable SQL Поддерживаемые команды и функции

Атрибуты файла	Поддержка
CREATE	ДА
DRIVER(*тип файла* [,*строка драйвера*])	ДА
NAME	ДА
ENCRYPT	НЕТ
OWNER(*пароль*)	ДА[1]
RECLAIM	НЕТ
PRE(*префикс*)	ДА
BINDABLE	ДА
THREAD	ДА
EXTERNAL(*член*)	ДА
DLL([*флаг*])	ДА
OEM	НЕТ

Структура файла	Поддержка
INDEX	ДА
KEY	ДА
MEMO	НЕТ
BLOB	НЕТ
RECORD	ДА

Атрибуты Index, Key, Memo	Поддержка
BINARY	НЕТ[3]
DUP	ДА
NOCASE	ДА
OPT	НЕТ
PRIMARY	ДА
NAME	ДА
В порядке увеличения	ДА
В порядке уменьшения	ДА
Смешанный порядок	ДА

Атрибуты полей	Поддержка
DIM	НЕТ
OVER	ДА
NAME	ДА

Процедуры файла	Поддержка
BOF(*файл*)	НЕТ
BUFFER(*файл*)	ДА
BUILD(*файл*)	ДА
BUILD(*ключ*)	ДА
BUILD(*индекс*)	ДА[3]
BUILD(*индекс*, *компонента*)	ДА[3]
BUILD(*индекс*, *компонента*, *фильтр*)	НЕТ
BYTES(*файл*)	ДА
CLOSE(*файл*)	ДА
COPY(*файл*, *новый файл*)	НЕТ
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	НЕТ
FLUSH(*файл*)	НЕТ
LOCK(*файл*)	НЕТ
NAME(*label*)	ДА
OPEN(*файл*, *режим доступа*)	ДА
PACK(*файл*)	НЕТ
POINTER(*файл*)	НЕТ
POINTER(*ключ*)	НЕТ
POSITION(*файл*)	НЕТ
POSITION(*ключ*)	ДА
RECORDS(*файл*)	ДА
RECORDS(*ключ*)	ДА
REMOVE(*файл*)	ДА
RENAME(*файл*, *новый файл*)	НЕТ
SEND(*файл*, *сообщение*)	ДА
SHARE(*файл*, *режим доступа*)	ДА
STATUS(*файл*)	ДА
STREAM(*файл*)	НЕТ

UNLOCK(*файл*)	НЕТ

Доступ к записям	Поддержка
ADD(*файл*)	ДА
ADD(*файл, длина*)	НЕТ
APPEND(*файл*)	ДА
APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	ДА
GET(*файл,ключ)*	ДА
GET(*файл, указатель файла*)	НЕТ
GET(*файл, указатель файла,длина*)	НЕТ
GET(*ключ, указатель ключа*)	НЕТ
HOLD(*файл*)	НЕТ
NEXT(*файл*)	ДА
NOMEMO(*файл*)	НЕТ
PREVIOUS(*файл*)	ДА
PUT(*файл*)	ДА
PUT(*файл, указатель файла*)	НЕТ
PUT(*файл, указатель файла, length*)	НЕТ
RELEASE(*файл*)	НЕТ
REGET(*файл,строка*)	НЕТ
REGET(*ключ,строка*)	ДА
RESET(*файл,строка*)	НЕТ
RESET(*ключ,строка*)	ДА
SET(*файл*)	ДА
SET(*файл, ключ*)	НЕТ
SET(*файл, указатель файла*)	НЕТ
SET(*ключ*)	ДА
SET(*ключ, ключ*)	ДА
SET(*ключ, указатель ключа*)	НЕТ
SET(*ключ, ключ, указатель файла*)	НЕТ
SKIP(*файл, число*)	ДА
WATCH(*файл*)	ДА

Обработка транзакций	Поддержка[2]
LOGOUT(*время ожидания, файл, ..., файл*)	ДА[4]
COMMIT	ДА
ROLLBACK	ДА

Обработка Null полей	Поддержка
NULL(*поле*)	ДА
SETNULL(*поле*)	ДА
SETNONNULL(*поле*)	ДА

Примечания

1 Мы рекомендуем использовать длинные и содержащие специальные символы значения пароля, так как именно таким способом обеспечивается большая эффективность защиты. Например, пароль подобный «;%?? $$^» гораздо сложнее разгадать, чем пароль типа «SALARY.»

Совет: Для определения переменной содержащей актуальное значение пароля в поле Owner Name в диалоге «Свойства файла» (File Properties dialog) перед именем переменной наберите восклицательный знак '!'. Например: !MyPassword.

2. См также *PROP:Logout* в *Описании языка*

3. BUILD(index) устанавливает внутренние флаги драйвера для того, чтобы гарантировать, что драйвер сгенерировал правильное предложение ORDER BY. Драйвер при этом не обращается к серверу базы данных.

4. Будет ли LOGOUT блокировать таблицу - зависит от того, как настроена обработка транзакций на сервере. См. документацию вашего сервера.

Драйвер TopSpeed

Обзор

Файловая система базы данных TopSpeed является высокоэффективным, высоконадежным, запатентованным драйвером файла для Clarion-инструментов разработки. Она несовместима по файлу с данными драйвера файлов Clarion.

Таблицы данных, ключи, индексы и все мемо могут быть сохранены вместе в одном DOS файле. Расширение файла по умолчанию *.TPS. Отдельный «Файл контроля транзакции» принимает расширение *.TCF.

Драйвер TopSpeed может, если это необходимо, сохранять многочисленные таблицы в одном файле DOS. Это дает вам возможность открыть столько файлов данных, ключей и индексов, сколько необходимо при использовании одного дескриптора файла DOS. Эта характеристика может быть особенно полезна, когда есть большое число малых таблиц или когда группа связанных файлов используется совместно. Все ключи, индексы и мемо сохраняются внутри.

Замечание: Когда несколько таблиц разделяют один дескриптор файла, режим первого открытия файла распространяется на все таблицы внутри него.

Кроме того, файловая система TopSpeed поддерживает тип данных BLOB (Binary Large Object - бинарный большой объект), поле полностью переменной длины и может быть размером более 64 К (как в 16-битовых, так и в 32-битовых приложениях). BLOB должен быть объявлен перед структурой RECORD. Память для BLOB выделяется и освобождается динамически по мере необходимости. Более полную информацию можно найти в разделе BLOB в Справочнике языка.

Files:	C4TPSL.LIB	**Статическая библиотека Windows (16-bit)**
	C4TPSXL.LIB	**Статическая библиотека Windows (32-bit)**
	C4TPS.LIB	**Экспортная библиотека Windows(16-bit)**
	C4TPSX.LIB	**Экспортная библиотека Windows(16-bit) (32-bit)**
	C4TPS.DLL	**Динамически вызываемая библиотека Windows(16-bit)**
	C4TPSX.DLL	**Динамически вызываемая библиотека Windows (32-bit)**

Совет: Этот новый драйвер предлагает скорость, безопасность и требует меньших ресурсов от системы конечного пользователя.

Спецификации

Типы данных

BYTE	**DECIMAL**
SHORT	**STRING**
USHORT	**CSTRING**
LONG	**PSTRING**
ULONG	**MEMO**
SREAL	**GROUP**
REAL	**BLOB**
DATE	**TIME**

Характеристики файла/максимумы

Размер файла:	**Ограничен только местом на диске**
Записей на файл:	4,294,967,295
Размер записи:	15K
Размер поля:	15K
Полей на запись:	15K
Ключей/индексов на файл:	240
Размер ключа:	15K
Мемо полей на файл:	255
Размер поля мемо:	64K
Полей BLOB на файл:	255
Размер BLOB:	зависит от компьютера
Открывание файла:	в зависимости от операционной системы
Имя таблицы	**1000 байт**

Количество конкурирующих пользователей на файл 1024

Строки драйвера.

Имеются переключатели или «строки драйверов», при помощи которых вы можете управлять способом, которым ваше приложение создает, читает и пишет

файлы выбранной вами файловой системы. Строки драйвера - это просто характерные для данного драйвера сообщения или параметры, которые могут быть посланы драйверу файла во время работы для управления его поведением. Для получения обзорной информации об этих переключателях и параметрах времени исполнения см. *Common Driver Features—Driver Strings.*

Замечание: Некоторые строки драйвера не действуют после того как файл открыт, поэтому никакая функция SEND не выполняется для этих строк. В тоже время функция SEND возвращает значения переключателей для всех строк драйвера

Драйвер TopSpeed поддерживает следующие строки драйвера:

FLAGS

[Flags» =] SEND(file, 'FLAGS [= bitmap]')

Посылает и возвращает DOS-атрибуты файла. Используйте следующие, объявленные в EQUATES.CLW EQUATE для управления поведением целевого файла TopSpeed:

```
!TopSpeed флаги файла
TPSREADONLY   EQUATE(1)
```

Например, следующий код устанавливает файл «только для чтения» для доступа ODBC маскируя все другие флаги.

```
TpsFlags = SEND(MyFile, 'FLAGS')
SEND(MyFile, 'FLAGS ='&BOR(TpsFlags,TPSREADONLY)
```

FULLBUILD

[State» =] SEND(file, 'FULLBUILD [= on | off]')
[State» =] file{PROP:FULLBUILD} [= on | off]')

Драйвер TopSpeed использует механизм оптимизации добавления оператором APPEND большого количества записей в существующую таблицу. Перестроение ключей в соответствии только с добавляемой ключевой информацией делает процесс очень быстрым. Это поведение по умолчанию. Использование строки драйвера FULLBUILD модифицирует это поведение

FULLBUILD=ON сообщает следующему оператору BUILD о необходимости полной перестройки ключей. FULLBUILD=OFF восстанавливает оптимизирующий

режим BUILD. Обе версии команды SEND возвращают текущее состояние BUILD в виде строки 'ON' или 'OFF'. Вызов SEND(file,'FULLBUILD') возвращает текущее состояние BUILD без его изменения.

PNM=

TName» = SEND(*file*, 'PNM=[starting point]')

Возвращает следующее имя таблицы в суперфайле TopSpeed, после указанной *стартовой точки*. Если после указанной *стартовой точки* таблиц нет SEND возвращает пустую строку. Если *стартовая точка* пропущена или содержит пустую строку SEND возвращает имя первой таблицы в файле. PNM= допустимо только в команде SEND. Между знаком равенства и операндами не должно быть пробелов. Целевой *file* это метка любой таблицы внутри суперфайла TopSpeed.

Например пусть в некотором файле TopSpeed содержится таблица SUPP, следующий код выводит в алфавитном порядке список всех таблиц в этом файле:

```
CODE
name = ''
LOOP
        name = SEND(Supp,'PNM=' & name)
        If name
        MESSAGE(name)
        ELSE
        BREAK
        END
END
```

TCF

[TCFLocation» =] SEND(file, 'TCF [= filename]')

Устанавливает файл контроля транзакции, иной, чем принятый по умолчанию \TOPSPEED.TCF. Файл сохраняет все многофайловые законченные транзакции до окончания программы или до исполнения SEND(TCF=file name).

Другими словами, установка TCF влияет на все TopSpeed-файлы, доступные из программы. Для получения дополнительной информации см. *Обработка транзакций - файл TCF.*

Замечание: мы рекомендуем использовать один файл контроля транзакций на всю систему. Использование нескольких файлов с различными правами доступа может привести к частичному завершению транзакций - некоторые файлы внутри транзакции могут быть изменены, в то время как другие останутся неизмененными.

Поддерживаемые команды файлов и функции

Атрибуты файла	Поддержка
CREATE	ДА
DRIVER(*filetype* [,*driver строка*])	ДА
NAME	ДА
ENCRYPT	ДА
OWNER(*password*)	ДА[1]
RECLAIM	НЕТ[2]
PRE(*prefix*)	ДА
BINDABLE	ДА
THREAD	ДА[12]
EXTERNAL(*member*)	ДА
DLL([*flag*])	ДА
OEM	**ДА**

Структуры файла	Поддержка
ИНДЕКС	ДА
КЛЮЧ	ДА
MEMO	ДА[3]
BLOB	ДА
RECORD	**ДА**

Index, Key, Memo Атрибуты	Поддержка
BINARY	ДА[13]
DUP	ДА
NOCASE	ДА
OPT	ДА
PRIMARY	ДА
NAME	Y[4]
Ascending Components	ДА
Descending Components	ДА
Mixed Components	ДА

Атрибуты полей	Поддержка
DIM	ДА
OVER	ДА
NAME	**ДА**

Процедуры файла	Поддержка
BOF(*файл*)	ДА
BUFFER(*файл*)	НЕТ
BUILD(*файл*)	ДА
BUILD(*ключ*)	ДА
BUILD(*индекс*)	ДА
BUILD(*индекс*, *Компоненты*)	ДА
BUILD(*индекс*, *Компоненты*, *фильтр*)	ДА
BYTES(*файл*)	ДА
CLOSE(*файл*)	ДА
COPY(*файл*, *новый файл*)	ДА
CREATE(*файл*)	ДА
DUPLICATE(*файл*)	ДА
DUPLICATE(*ключ*)	ДА
EMPTY(*файл*)	ДА
EOF(*файл*)	ДА
FLUSH(*файл*)	ДА
LOCK(*файл*)	ДА[5]
NAME(*метка*)	ДА
OPEN(*файл*, *режим доступа*)	ДА
PACK(*файл*)	ДА[6]
POINTER(*файл*)	ДА[8]
POINTER(*ключ*)	ДА[8]
POSITION(*файл*)	ДА[9]
POSITION(*ключ*)	ДА[9]
RECORDS(*файл*)	ДА
RECORDS(*ключ*)	ДА
REMOVE(*файл*)	ДА
RENAME(*файл*, *новый файл*)	ДА
SEND(*файл*, *сообщение*)	ДА
SHARE(*файл*, *режим доступа*)	ДА
STATUS(*файл*)	ДА
STREAM(*файл*)	ДА[7]
UNLOCK(*файл*)	ДА

Доступ к записям	Поддержка
ADD(*файл*)	ДА
ADD(*файл*, *длина*)	НЕТ
APPEND(*файл*)	ДА

APPEND(*файл, длина*)	НЕТ
DELETE(*файл*)	Y[2]
GET(*файл,ключ)*	ДА
GET(*файл, указатель файла*)	ДА[8]
GET(*файл, указатель файла, length*)	НЕТ
GET(*ключ, указатель ключа*)	ДА
HOLD(*файл*)	ДА
NEXT(*файл*)	ДА
NOMEMO(*файл*)	ДА
PREVIOUS(*файл*)	ДА
PUT(*файл*)	ДА
PUT(*файл, указатель файла*)	ДА
PUT(*файл, указатель файла, длина*)	НЕТ
RELEASE(*файл*)	ДА
REGET(*файл,строка*)	ДА
REGET(*ключ,строка*)	ДА
RESET(*файл,строка*)	ДА
RESET(*ключ,строка*)	ДА
SET(*файл*)	ДА
SET(*файл, ключ*)	ДА
SET(*файл, указатель файла*)	ДА
SET(*ключ*)	ДА
SET(*ключ, ключ*)	ДА
SET(*ключ, указатель ключа*)	ДА
SET(*ключ, ключ, указатель файла*)	ДА
SKIP(*файл, число*)	ДА
WATCH(*файл*)	ДА

Обработка транзакций	**Поддержка**
LOGOUT(*время ожидания, файл, ..., файл*)	Y[11]
COMMIT	ДА
ROLLBACK	ДА

Обработка пустых полей	**Поддержка**
NULL(*поле*)	ДА
SETNULL(*поле*)	ДА
SETNONNULL(*поле*)	ДА

Примечания:

1 Мы рекомендуем использовать длинные и содержащие специальные символы значения пароля, так как именно таким способом обеспечивается большая эффективность защиты. Например, пароль подобный «;%?? $$^» гораздо сложнее разгадать, чем пароль типа «SALARY.»

Совет: Для определения переменной, содержащей актуальное значение пароля в поле Owner Name в диалоге «Свойства файла» (File Properties dialog) перед именем переменной наберите восклицательный знак '!'. Например: !MyPassword.

2 Драйвер TopSpeed автоматически использует пространство, освобожденное удаленными записями и ключами.

3 Файловая система TopSpeed использует одинаковые алгоритмы сжатия для структур RECORD и MEMO. Для данных размером меньше 255 байт, MEMO не имеет выигрыша в пространстве на диске по сравнению со STRING. Однако STRING всегда занимает оперативную память внутри буфера записи, в то время как под MEMO память распределяется только в момент открытия файла. MEMO BINARY предпочтительнее MEMO NONBINARY, и, кроме того, MEMO может быть пропущен во всех процессах с оператором NOMEMO.

4 Драйвер TopSpeed не поддерживает внешние имена для ключей, так как все ключи хранятся внутри.

5 LOCK() только влияет на другие LOCK() вызовы. Единственным результатом успешного обращения к LOCK() является то, что другие процессы получат ошибку FLALLK, когда они вызовут LOCK().

5 PACK выполняет BUILD и усекает файл до его минимального размера

7 STREAM блокирует файл на запись, но не на чтение.

8 GET(file, filepointer) требует указателя, возвращаемого функцией POINTER. POINTER возвращает физический адрес записи (не номер записи). Следовательно, вы не можете использовать GET(file,l) для поиска первой записи в файле TopSpeed, так как 1 - это не указатель.

9 POSITION(file) возвращает STRING(4). POSITION(key) возвращает STRING размером, равным размеру ключевого поля + 4.

10 Драйвер TopSpeed выполняет транзакции очень быстро (примерно в 100 раз быстрее чем драйвер Clarion). Внутри транзакционной рамки механизм драйвера TopSpeed посылает все транзакции в память. Оператор ROLLBACK просто освобождает память, а COMMIT потоком переписывает все изменения в базу.

Если система разрушается в процессе выполнения транзакции, драйвер TopSpeed автоматически произведет все возможные восстановления при следующем обращении к этим файлам

11 LOGOUT блокирует файл на запись, но не на чтение. См. также *PROP:Logout* в *Описании языка*.

12 Файлы с атрибутом THREAD не требуют дополнительных дескриптроров файла для каждого процесса, получающего доступ к этим файлам.

13 Атрибут OEM не применим к BINARY MEMO и BLOB.

Драйвер TopSpeed нечувствителен к регистру при преобразовании строк в нижний регистр. Это может привести к непредсказуемым результатам для символов, расположенных между большими и маленькими буквами (такими как ^(94) и _(95) как для ANSI так и для ASCII кодировок)

Разное

Разделение файлов

¤ SHARE и режимы открытия файла:

Поддерживаются следующие режимы открытия	Требуется Share
34 (12h) Читать/записывать, запрет записи (умолчание для OPEN(открыто)	**Да**
66 (42h) Читать/записывать, запретов никаких (умолчание для SHARE(общий)	**Да**
64 (40h) Только читать, запретов никаких	**Да**
18 (12h) Читать/записывать, запрет на все	**Нет**
16 (10h) Только читать, запрет на все	**Нет**
32 (20h) Только читать, запрет записывать	**Нет**

Для указанных режимов SHARE.EXE должно быть загружено в AUTOEXEC.BAT или CONFIG.SYS. Следующий пример загружает SHARE в AUTOEXEC.BAT, обеспечивая максимум 500 файлов заблокированными и по умолчанию 2048 байт для зоны хранения.

```
C:\DOS\SHARE.EXE /L:500
```

Если SHARE требуется, но не загружено, программа генерирует ошибку времени исполнения.

Совет: Вам не понадобится SHARE.EXE (или VSHARE) в любой среде (например, Novell, Win95 или Win NT), которая поставляет блокирование файлов как часть операционной системы.

Если никакая форма SHARE не представлена (SHARE или VSHARE), тогда для первого доступа к файлу драйвер открывает файл в режиме исключительного доступа. После этого последующие попытки открыть файл будут игнорироваться.

ERRORCODE 90

Драйвер TopSpeed посылает ERRORCODE=90 для нераспознаваемых ошибок времени исполнения. Одновременно с этим драйвер посылает FILEERRORCODE, который помогает нам диагностировать возникшую проблему. Эти сообщения об ошибках дают вам дополнительную возможность контролировать ошибки времени исполнения и предоставляют нам дополнительную информацию. Таким образом ваша программа может отлавливать ERRORCODE=90 и соответствующим образом реагировать на них.

Если вы получили ERRORCODE=90 от драйвера TopSpeed, мы хотим знать об этом. Пожалуйста пришлите нам копию файла и соответствующее значение FILEERRORCODE.

Большие ключи (или маленькая RAM)

Если общий размер ключей превышает объем доступной оперативной памяти или если имеется более, чем один ключ или в тех случаях, когда добавляется большое количество записей вместо оператора ADD(), рекомендуется использовать оператор APPEND(). Размером ключа в этом случае является количество элементов (сумма ключевых полей плюс 10 байт). Если добавляемые записи уже находятся в приблизительном порядке ключей, тогда вы можете учесть этот ключ для целей вышеупомянутого расчета.

В качестве примера, если файл имеет два 40-байтных ключа и 2 Мегабайта памяти, ADD() становится (относительно) медленным, когда размер базы данных превышает примерно 2,000,000/(40+10+40+10) = 20,000 записей.

Инкрементное Key/Index Build (построение ключей/индексов)

Драйвер TopSpeed выполняет инкрементное построение; это означает, что при построении ключа читаются только записи, начиная с первой записи, добавленной после того, как ключ был построен. Драйвер сливает новые ключи с существующим ключом. Таким образом, построение большого ключа, где только несколько недавно добавленных записей было модифицировано, должно быть быстрым. Смотрите выше строку драйвера FULLBUILD.

Построение индекса аналогично, но должно начинаться при минимальной физической записи, чья позиция в индексе изменилась с тех пор, как был построен последний индекс.

Динамические индексы не сохраняются, таким образом не могут быть построены инкрементно.

Выполнение пакетной обработки

При чтении большого числа записей используйте STREAM() или откройте файл в режиме запрета записи, например OPEN(file), а не SHARE(file). После того, как записи были прочитаны, вызовите FLUSH(), чтобы разрешить доступ другим пользователям.

Очень важно использовать STREAM() при добавлении/присоединении/помещении большого числа записей. STREAM() обычно делает обработку в 20 раз быстрее. Например, добавление 1000 записей может потребовать около 2 минут без STREAM() и только 5 секунд при использовании STREAM().

Нет необходимости использовать STREAM() или FLUSH() на выведенном из системы файле (работа на выведенных файлах всегда успешна).

STREAM не действует для заблокированных (LOCK) файлов

Указатели (POINTERS) и удаленные записи

Величина, возвращенная POINTER(key), соответствует физическому порядку записей данных. Поэтому, когда запись DELETED (удалена), указатель становится недействительным. Любой последующий доступ с помощью указателя терпит

неудачу. Если вам нужно точное совпадение, по которому возвращается ближайшая запись, используйте функцию POSITION().

Сжатие данных - STRING и MEMO

Драйвер TopSpeed сжимает буфер записи целиком (не отдельные поля внутри записи) поэтому компрессия может быть улучшена за счет размещения отдельных полей в объявлении файла “подогнанными” друг к другу.

Файловая система TopSpeed использует для полей STRING и MEMO один и тот же механизм сжатия, однако MEMO сжимается лучше, нежели STRING. В результате с точки зрения использования дискового пространства MEMO имеет преимущество по сравнению с большими STRING(больше 500 байт), а маленькие строки несколько производительнее по сравнению с MEMO. Чем больше строка тем больше выигрыш в производительности.

MEMO BINARY предпочтительнее MEMO NONBINARY, и, кроме того, MEMO может быть пропущен во всех процессах с оператором NOMEMO.

Однако STRING всегда занимает оперативную память внутри буфера записи, в то время как под MEMO память распределяется только в момент открытия файла. Также MEMO не может быть компонентой ключа.

Оценка размеров файла

Драйвер TopSpeed сжимает данные и ключи, и поэтому окончательный размер файла зависит от «сжимаемости» данных и ключей. В тяжелом случае (данные и ключи не могут быть сжаты из за отсутствия повторяющейся информации) размер файла может быть оценен как: (размер записи + размер всех ключевых компонент)*число записей + фиксированный заголовок.

(размер записи + размер всех ключевых компонент)*число записей + фиксированный заголовок.

В более реальном случае (данные и ключи сжимаемы) размер файла может быть оценен как:

((размер всех строковых полей)/(фактор сжатия) + размер всех двоичных полей + размер всех двоичных компонент ключа +(4*количество строковых компонент ключа))*количество записей + фиксированный заголовок

Заметим, что фиксированный заголовок зависит от определения файла.

Фиксированный заголовок включает около 800 байт для драйвера, плюс информация, описывающая поля и ключи. Больше полей и ключей и длиннее имена - больше заголовок. Приблизительно размер фиксированного заголовка можно вычислить как: 800 байт + 40 байт на каждое поле и ключ. Например:

Описание файла	Оценка фиксированного заголовка
1 поле, нет ключей	1KB
20 полей, 10 ключей	2KB
200 полей, 10 ключей	9KB

Предельное количество конкурирующих пользователей

Драйвер TopSpeed допускает одновременную работу с одним файлом файл не более чем 1024 пользователей; дополнительные пользователи ожидают момента, когда они попадут в это число. Практически, очень маловероятно, чтобы драйвер достиг этого предела, так как редко какая сеть или сервер будут поддерживать такое количество конкурирующих пользователей. Обычно мы рекомендуем для работы более, чем 30 пользователей файловые системы клиент/сервер.

Обработка транзакций - TCF-файл

Быстрая обработка транзакции и автоматическое восстановление

Драйвер TopSpeed выполняет транзакции очень быстро (примерно в 100 раз быстрее чем драйвер Clarion). Внутри транзакционной рамки механизм драйвера TopSpeed посылает все транзакции во временную область хранения на диск. Оператор ROLLBACK просто отмечает транзакцию как невыполненную, а COMMIT делает эту область частью файла TopSpeed.

Файл контроля транзакции (.TCF)используется для обеспечения уверенности, что изменения более, чем одного DOS - файла, которые группируются внутри транзакционной рамки либо завершены удачно либо не были произведены вовсе. По умолчанию файл контроля транзакций (TCF) имеет имя «\TOPSPEED.TCF.» Строка драйвера TCF позволяет вам изменить это. Для получения дополнительной информации см. *TCF.*

Когда рабочая станция обнаруживает, что файл, который находится в состоянии COMMIT, и который был включен в многофайловую транзакцию, необходимо обращение к файлу TCF для выяснения того, что необходимо делать в данной ситуации. Файл TCF предоставляет информацию (boolean) о том, утверждена многофайловая транзакция или нет.

Заметим что файл TCF содержит очень мало информации; она просто служит для координации подтверждений многофайловых транзакций. Актуальные 'rollback/commit' данные хранятся в TCF файле

Как TopSpeed обрабатывает транзакции

Оператор LOGOUT() присваивает каждой транзакции уникальный идентификатор, который запоминается в файле TCF. LOGOUT также запоминает имя файла TCF и идентификатор транзакции в каждом изменяемом файле TPS , таким образом, при следующем после разрушения системы открытии файла драйвер TopSpeed может найти файл TCF и сделать необходимое восстановление. Оператор COMMIT удаляет из файла TCF уникальный идентификатор транзакции.

Для того, чтобы выполнять свои функции, файл TCF должен быть доступен, когда осуществляется доступ к контролируемым им файлам. Поэтому вы обычно не должны удалять или перемещать его. Если транзакция изменяет сетевые файлы, вы должны определить файл контроля транзакций в сети.

Не является необходимым использовать тот же TCF файл для всех транзакций; однако, мы настоятельно рекомендуем это. Существование нескольких файлов TCF с разными уровнями доступа может привести к тому, что некоторые из файлов внутри транзакции могут быть изменены, в то время как другие останутся без изменения

Хранение нескольких таблиц (файлов данных) в одном файле .TPS

Используя специальную управляющую последовательность '\!' в атрибуте NAME() декларации файла TopSpeed, вы можете установить, что один .TPS файл будет хранить более одной таблицы. Например, чтобы объявить один TPS файл 's&p.tps',который должен содержать 3 логических таблицы, вызываем supp, part, ship:.

```
Supp    FILE,DRIVER('TopSpeed'),PRE(Supp),CREATE,NAME('S&P\!Supp')
  ...
Part    FILE,DRIVER('TopSpeed'),PRE(Part),CREATE,NAME('S&P\!Part')
  ...
Ship    FILE,DRIVER('TopSpeed'),PRE(Ship),CREATE,NAME('S&P\!Ship')
```

Файлы данных имеют общий дескриптор одного DOS файла, открытый, когда открыта первая таблица, и закрытый, когда закрыта последняя таблица. Режим первого открытия определяет режим открытия для всех других. Если первый режим - это режим «только для чтения», тогда все таблицы этого файла тоже будут в состоянии «только для чтения» и обновление для них будет запрещено.

Если одна таблица файла включена в оператор LOGOUT, это будет касаться и всех остальных. Если для одной таблицы выполнен оператор FLUSH(), то же будет выполнено и для других таблиц.

Эта возможность особенно полезна тогда, когда имеется большое число малых таблиц, или когда приложение должно иметь одновременный доступ к группе связанных файлов.

Команда SEND() позволяет программисту определить имена файлов в группе. Чтобы найти имя, выдайте:

```
SEND(file,'PNM=').
```

Это возвращает имя первой таблицы. Чтобы найти второе имя, выдайте:

```
SEND(file,'PNM=FirstTableName').
```

Это возвращает имя второй таблицы, и т.д.

Таблицы могут быть переименованы внутри группы; например, для вышеприведенных объявлений следующая команда переименует файл, названный Supp в Old_Supp:

```
RENAME(Supp,'S&P\!Old_Supp')
```

Если вы используете атрибут владельца OWNER на множественных таблицах в файле базы данных TopSpeed, все таблицы должны иметь тот же самый атрибут OWNER.

Если не установлена никакая управляющая последовательность, тогда дается имя таблицы по умолчанию 'unnamed' (без названия), так что все следующее эквивалентно:

```
foo   FILE,DRIVER('TopSpeed')
foo   FILE,DRIVER('TopSpeed'),NAME('foo')
foo   FILE,DRIVER('TopSpeed'),NAME('foo\!unnamed')
```

Сортирующие последовательности

Изменение сортирующей последовательности

Изменение сортирующей последовательности в Clarion 2.003 или в более ранних

версиях Clarion (при изменении файла .ENV или атрибута OEM) приводило к разрушению файла данных.

В Clarion4 это не так, сортирующая последовательность теперь хранится внутри файла данных. Эти изменение полностью сохраняет совместимость с более ранними версиями Clarion. Старые файлы продолжают работать как и раньше, и новые файлы доступны старым программам.

Для добавления сортирующей последовательности к существующему файлу необходимо просто полностью перестроить ключи.

```
SEND(file,'FULLBUILD=on')
BUILD(file)
```

Сортирующая последовательность для файлов TopSpeed устанавливается в момент создания таблицы или при выполнении полной перестройки ключей. Поэтому атрибут OEM только обозначает создание файла или полную перестройку ключей.

Любые приложения, которые используют неправильную сортирующую последовательность (из-за несовместимого файла .ENV) при доступе к файлу, могут привести к непредсказуемым результатам, но не разрушат файла.

Доступ к файлам TopSpeed через Access Jet и ODBC

Иногда Access Jet возвращает «#deleted» для каждого затребованного поля. Это известная ошибка Microsoft Access. По умолчанию запрос, используемый Access, выполняет внутреннее сравнение множества записей для определения, были ли записи в базе данных удалены или изменены. Известно, что данный механизм работает плохо для некоторых типов данных, например DECIMAL.

Microsoft рекомендует использовать сквозной SQL запрос как способ обойти эту проблему. Для создания такого запроса:

1 Выберите SQL Specific, Pass Through из меню Query в режиме создания запросов. Для Access 7 выберите Query и нажмите кнопку New.
2 Примите принятый по умолчанию Design View.
3 Закройте окно Show Table
4 Выберите SQL Specific из меню Query и выберите Pass-Through.
5 Введите оператор SQL

Запрос может быть сохранен для будущего использования. Этот метод будет корректировать фактически все проблемы, не результирующая сетка недоступна для изменений. Изменения должны выполнятся при помощи запроса UPDATE, когда используется SQL Passthrough.

Утилита восстановления базы данных TopSpeed

Система файлов TopSpeed предназначена для автоматического исправления большинства ошибок. Однако, если файл данных физически поврежден во время неправильной работы системы, Утилита восстановления базы данных TopSpeed может восстановить поврежденные части ваших данных.

Внимание: Утилита восстановления базы данных TopSpeed является аварийным инструментом ремонта и не может использоваться в обычных условиях. Пользуйтесь ею только тогда, когда файл был поврежден.

Утилита восстановления базы данных TopSpeed читает поврежденное поле и пишет восстановленные записи в новый файл. Она использует информацию, хранящуюся в заголовке файла или сканирует файл в поисках неповрежденных частей.

По желанию, если оригинальный заголовок разрушен, вы можете использовать файл образец, содержащий информацию заголовка файла. Файл образец - это любой файл с описанием файла идентичным описанию разрушенного файла. Вы можете создать файл образец при помощи оператора CREATE(file) и затем сохранить полученный пустой файл под новым именем.

Утилита восстановления базы данных TopSpeed является свободно распределяемой утилитой, предназначенной для того, чтобы дать конечному пользователю возможность восстановить поврежденные файлы.

Совет: Лицензионное соглашение Clarion for Windows применимо к TPSFIX.EXE; распределение ограничено. Вы можете распределять утилиту своим пользователям, но они этого делать не имеют права.

Утилита восстановления предназначена для работы интерактивно или явно через параметры командной строки. Интерактивно вы обеспечиваете параметры через два диалоговых окна мастера. Вы также можете выполнять TPSFIX не интерактивно из командной строки при помощи оператора Clarion RUN() или через вызов Windows API или через ярлык Windows 95 или через Program Manager

ERRORCODE 90 и разрушенные файлы

Драйвер TopSpeed посылает ERRORCODE=90 для нераспознаваемых ошибок времени исполнения. Одновременно с этим драйвер посылает FILEERRORCODE, который помогает нам диагностировать возникшую проблему.

ERRORCODE 90 обычно обозначает, что файл TopSpeed разрушен. В большинстве случаев разрушение файла это следствие сбоя оборудования. Например, один из заказчиков наблюдал в сети, состоящей из 50 машин почти ежедневное разрушения файла из-за 2 плохих сетевых карт из 50. После замены этих плохих карт разрушения файла исчезли.

Если вы получили ERRORCODE=90 от драйвера TopSpeed, мы хотим знать об этом. Перед тем, как вы восстановите файл, пожалуйста, сделайте копию разрушенного файла и пришлите его нам вместе с соответствующим значением FILEERRORCODE.

Использование утилиты восстановления в интерактивном режиме

1. Запустите утилиту, щелкнув для этого дважды клавишей мыши на Пиктограмме утилиты восстановления базы данных TopSpeed в Программной группе Clarion for Windows .

Появляется диалоговое окно TopSpeed Database Recovery Utility **(утилита восстановления базы данных TopSpeed). Утилита содержит два диалоговых окна.**

2. В разделе Source **(файл, который нужно восстановить) установите имя файла или нажмите кнопку** Browse **(просмотр) для выбора ее из стандартного диалогового окна открытия файла.**

3. Если у файла есть пароль, наберите его в поле ввода Password **(пароль).**

Если файл базы данных содержит много таблиц (файлы данных), каждая таблица должна иметь тот же самый пароль.

4. По желанию в разделе Destination **(назначение - файл результата) установите имя файла для целевого файла или нажмите кнопку** Browse **(просмотр) для выбора его в стандартном диалоговом окне открытия файла.**

По умолчанию расширение .TPR добавляется к имени исходного файла. Это параметр, выбираемый по желанию. Если он опущен, первоначальный (исходный) файл переписывается и создается резервный файл. Исходный файл переименовывается в *filename*.TPX (имя файла), где x автоматически меняется от 1 до 9 каждый раз, как создается новый файл. Если использованы все девять номеров, любой последующий созданный файл получает расширение .TP$ и переписывается.

5. Если файл результата должен иметь другой пароль, наберите его в поле

ввода Password **(пароль). Если он опущен, пароль удаляется.**

6. Нажмите кнопку Next **(следующий).**

Появится второе мастер-диалоговое окно для Утилиты восстановления базы данных TopSpeed.

7. По желанию установите имя файла Example File **(образцовый файл) или нажмите кнопку** Browse **(просмотр) для выбора его из стандартного диалогового окна открытия файла.**

Утилита использует файл образец для определения плана таблицы и определения ключей в том случае, если эти области исходного файла повреждены. Расширение. По умолчанию - .TPE, но если вы будете выбирать, вы можете использовать расширение DOS.

Совет: Мы рекомендуем отгрузить образцовый файл, когда вы будете развертывать ваше приложение. Это улучшает восстановление данных из поврежденного файла.

8. Если у образцового файла есть пароль, наберите его в поле ввода Password **(пароль).**

9. Если вы хотите, чтобы утилита перестроила Ключи, отметьте поле флажков Build Keys **(построить ключи).**

Если вы это не делаете, ключи перестраиваются первоначальным приложением, когда оно пытается открыть.

10. Если вы хотите использовать информацию заголовка в исходном файле, отметь поле Use Header **(использовать заголовок).**

Использование информации заголовка оптимизирует работу утилиты, но оно не должно применяться, если заголовок файла испорчен. Если это опущено, утилита просматривает весь файл данных и восстанавливает все неповрежденные страницы.

11. Если приложение использует Локальный (Locate .ENV) файл для альтернативной сортирующей последовательности, установите файл .ENV или нажмите кнопку Browse **(просмотр) для выбора его из стандартного диалогового окна открывания файла.**

12. Если файл использует атрибут OEM для управления сортирующей последовательностью, отметьте поле Use OEM **(используйте OEM).**

Это сделает возможным преобразование OEMTOANSI и ANSITOOEM.

13. Нажмите кнопку Start **(пуск) для того, чтобы начать процесс восстановления.**

Если утилита не находит никаких ошибок, появляется сообщение, информирующее вас о том, что «No Errors Detected in <filename.ext>» (в ... никаких ошибок не найдено) и спрашивающее, хотите ли вы продолжать восстановление.

Параметры командной строки

Утилита может также принять параметры командной строки, что дает вам возможность исполнить ее из приложения или пиктограммы Менеджера программ или ярлыка в Windows 95.

Здесь приведен синтаксис для неинтерактивного выполнения TPSFIX с параметрами командной строки.

TPSFIX *sourcepath[?password] [destpath[?password]]*
[/E:examplepath[?password]] [/L:Localepath] [/H] [/K] [/P] [/O]

TPSFIX	Исполняемый (TPSFIX.EXE)
sourcepath	**Имя файла и путь к исходному (поврежденному) файлу базы данных.**
?password	**Пароль файла.**
Destpath	**Имя файла и путь восстановленного файла базы данных. Если пропущен destpath такой же как и *sourcepath* и в этом случае требуется файл образец**
/E:examplepath	**Имя файла и путь образцового файла базы данных. Этот параметр требуется для всех «фиксировать и поместить» операций (когда *sourcepath* = *destpath*)**
/L:localepath	**Имя и путь Locate (.ENV) файла, используемого для установки альтернативной сортировочной последовательности.**
/H	**Если установлено, утилита использует информацию заголовка в исходном файле.**
/K	**Если установлено, утилита перестраивает все ключи для базы данных.**
/P	**Если установлено, пользователь получает приглашение для каждого параметра, даже если они имеются на командной строке.**
/O	**Если установлено, файл использует OEMTOANS и ANSITOOEM для определения сортировочной последовательности. Смотрите *Интернационализация в Справочнике языка*.**

Использование утилиты в вашем приложении

Есть несколько моментов, которые нужно учесть, прежде чем использовать утилиту TPSFIX. Ввиду следующего мы не рекомендуем выполнять TPSFIX из

вашего приложения. Гораздо лучше инструктировать вашего конечного пользователя закрыть прикладную программу перед выполнением TPSFIX.

- Файл базы данных НЕ должен быть открытым при работе утилиты. Перед стартом TPSFIX убедитесь, что файл закрыт.
- Чтобы предотвратить доступ во время процесса восстановления до его завершения, TPSFIX автоматически блокирует файл.
- Будет более эффективно и безопасно, если ваше приложение перестроит ключи KEYs (опуская параметр /K при восстановлении). Это также хороший способ проверить результат восстановления.

Автоматическое «Fix-in-Place» восстановление

При пропущенном параметре *destpath* и при использовании файла образца, вы можете переписать на месте разрушенный файл. Это и есть «Fix-in-Place» - восстановление. Утилита TPSFIX создает промежуточный файл, вам это не доставляет никаких хлопот. Например:

```
TPSFIX.EXE Datafile.TPS /E:Example.TPE /H
```

или через точку вставки

```
RUN('TPSFIX.EXE Datafile.TPS /E:Example.TPE /H')
```

Это восстанавливает файл «datafile.TPS» с помощью файла «Example.TPE» как образца для планов таблицы и ключа, не перестраивает ключи и использует информацию заголовка в исходном файле. TPSFIX автоматически сохраняет исходный файл (с расширениями от TP1 до TP9) для отката. При каждом новом выполнении утилиты номер файла автоматически увеличивается.

Раздельное восстановление.

Этот метод требует двух строк кода вставки, но дает вам контроль над процессом переименования. Вы вставляете текст в точку вставки из позиции меню или кнопкой. Например:

```
COPY(datafilelabel, 'Datafile.OLD')          ! копирование исходного файла
                                             ! в файл Datafile.OLD
RUN(TPSFIX Datafile.OLD Datafile.tps /H)     ! выполнение утилиты с использованием
                                             !переименованного файла в качестве
                                             !источника и исходного файла в качестве
                                             !мишени
```

Это копирует файл datafilelabel в DATAFILE.OLD, восстанавливает файл и пишет его в DATAFAIL.TPS , используя для этого информацию заголовка в исходном файле.

Утилита копирования базы данных TopSpeed

Утилита копирования базы данных TopSpeed дает вам возможность копировать в файлы (и из файлов) базы данных TopSpeed, содержащие много таблиц (файлы данных) в одном файле DOS.

Совет: Лицензионное соглашение Clarion for Windows применимо к TPSFIX.EXE; распределение утилиты ограничено. Вы можете распределять своим пользователям, но они этого делать не имеют права.

Пример использования

Используя данную утилиту, вы можете:

√ Извлечь файлы из мультитабличной базы данных TopSpeed. Это дает вам возможность извлечь файл, использовать программу преобразования, а затем вставить файл обратно.

√ Вставить файлы в мультитабличную базу данных TopSpeed. Это дает вам возможность создать каждый файл по отдельности, а затем вставить их все в один .TPS файл.

√ Заменить файлы из мультитабличной базы данных TopSpeed.

√ Удалить файлы из мультитабличной базы данных TopSpeed.

√ Добавить новую таблицу к мультитабличной базе данных TopSpeed.

√ Добавить новый ключ к существующему файлу в мультитабличной базе данных TopSpeed.

√ Добавить новое поле или модифицировать существующее поле в таблице внутри мультитабличной базы данных TopSpeed.

√ Заменить или модифицировать поле или ключ в мультитабличной базе данных TopSpeed.

√ Комбинировать однотабличные файлы базы данных TopSpeed в один мультитабличный файл базы данных TopSpeed.

√ Копировать файл данных в файле мультитабличной базы данных TopSpeed в другую базу данных.

Рассмотрение совместного использования файлов

Внимание: Перед тем, как применять эту или любую другую утилиту на «живых» файлах данных, вы должны создать резервные файлы.

Файл базы данных НЕ следует открывать во время использования утилиты. Прежде, чем запускать утилиту, убедитесь, что файл закрыт. Если исключительный доступ не разрешается, утилита показывает предупреждение «Отказ доступа» и дезактивирует кнопку Copy**.**

Чтобы предотвратить доступ до момента, когда процесс копирования будет завершен, утилита автоматически блокирует файл.

Интерфейс утилиты копирования

Source (copy from) Источник (откуда копировать)

Database Name (имя базы данных)
Имя файла DOS для файла-источника базы данных TopSpeed, из которого копируются таблицы. Наберите имя файла, или же отберите существующий файл путем нажатия кнопки **Browse** (просмотр).

Password (пароль) Пароль (атрибут OWNER (владелец)) для файла-источника. Если файл имеет пароль, он должен быть представлен. Если база данных источника не имеет пароля, это поле ввода дезактивируется.

File to Copy (файл, который нужно копировать).
Таблица в базе данных TopSpeed, из которой нужно копировать. Отберите из ниспадающего списка. Если база данных содержит только одну таблицу, она обычно называется UNNAMED (без имени).

Remove File from Source Database (удалить файл из базы данных)
Отметьте это поле, если вы хотите удалить таблицу из базы данных источника после ее копирования. Если это поле не отмечено, таблица остается в базе данных источника. При использовании этой опции, когда последняя таблица перемещена из базы данных источника, вы имеете опцию удаления физического файла с диска.

Rebuild Keys (построить ключи заново)
Выполняет полное построение ключей базы данных источника.

Destination (copy to) (назначение (куда копировать))

Database Name (имя базы данных)
Имя файла DOS для файла-мишени базы данных TopSpeed, в который

копируются таблицы. Наберите имя файла, или же отберите существующий файл путем нажатия кнопки **Browse** (просмотр)..

Password (пароль) Пароль (атрибут OWNER (владелец)) для файла-мишени. Если файл места назначения имеет пароль, он должен быть представлен. Если база данных источника не имеет пароля, а пароль представлен, он добавляется. Если создается новая база данных, введите здесь новый пароль.

New File (новый файл)
Имя таблицы в базе данных TopSpeed, в которую нужно копировать файл- источник. По умолчанию это дает имя таблицы из базы данных источника. Если вы хотите создать базу данных с одной таблицей, новый файл должен быть назван UNNAMED (без имени).

Copy (копировать)
Нажмите эту кнопку для того, чтобы скопировать таблицу источника в место ее назначения.

Exit (выход) Нажмите эту кнопку, чтобы закрыть утилиту.

Параметры командной строки утилиты копирования

TPSCOPY ***sourcefile[?password] [!tablename destinationfile[?password] ! tablename [/R] [/D]***

TPSCOPY Запускает исполняемую программу(TPSCOPY.EXE)

sourcefile **(файл- источник)**
Имя файла DOS для файла-источника базы данных TopSpeed, из которого копируются таблицы.

?*password* **(пароль)**
Пароль файла (атрибут OWNER (владелец)). Если у файла имеется пароль, он должен быть представлен. Если у базы данных нет пароля, это можно опустить.

!*tablename* **(имя таблицы)**
Имя конкретной таблицы в базе данных TopSpeed. Если база данных содержит только одну таблицу, она называется UN NAMED (без имени).

destinationfile **(файл места назначения)**
Имя файла DOS для файла-мишени базы данных TopSpeed, в который копируется таблица. Наберите имя файла (включая расширение) для того, чтобы создать новый файл. в

/R **Переместить данную таблицу из базы данных источника после того, как она скопирована. у в файле) из OREDRS.TPS (который не имеет пароля)**

/D **Удалить базу данных источника, не имеющую оставшихся таблиц (удалить физический файл с диска).**

/K **Выполняет полную постройку ключей базы данных источника перед копированием.**

Примеры:

```
TPSCOPY orders.tps?acme!CUSTOMER customer.tps?acme96!UNNAMEDNAMED
```

Копирует таблицу заказчика из OREDRS.TPS (используя пароль acme) в однотабличный файл CUSTOMER.TPS, который имеет пароль acme96.

```
TPSCOPY orders.tps?fred!CUSTOMER customer.tps?barney!UNNAMED
```

Копирует таблицу заказчика из OREDRS.TPS (используя пароль fred) в однотабличный файл CUSTOMER.TPS, который имеет пароль barney.

```
TPSCOPY orders.tps!UNNAMED orders.tps?acme96!UN
```

Копирует таблицу UNNAMED (единственную таблицу в файле) из OREDRS.TPS (который не имеет пароля) в таблицу CUSTOMER.TPS в файле ORDERS.TPS, который имеет пароль acme96.