Содержание

1 Инструменты и режимы комплекса.

1.1 Главная форма задачи восстановления данных

Внешний вид основной формы задачи зависит от настроек, введенных при создании задачи, и может существенно меняться. Элементы этой формы можно подразделить на группы:

- Основное меню;
- Панель кнопок быстрого доступа;
- Информационная панель;
- Панель с закладками протокола, карты результатов копирования, просмотра сектора;
- Панель проводника.

Наличие панелей, состав элементов на них и отображаемая информация зависят от настроек задачи и конкретного режима.

1.1.1 Основное меню главной формы задачи

Состав элементов основного меню постоянен, но доступность его пунктов может зависеть от типа задачи и текущего выполняемого процесса.

1.1.1.1 Меню "Настройки"

Состав пунктов этого меню неизменен, может лишь меняться доступность пункта *Параметры задачи*. В некоторых случаях этот пункт недоступен.

1.1.1.1.1 Справочник регулярных выражений

Данный справочник включает в себя список регулярных выражений, используемый поисковой системой программы.

Вместе с программой поставляется базовый справочник регулярных выражений, включающий более 160 записей. При необходимости пользователь может добавить в этот справочник свои записи.

1.1.1.1.1.1 Понятие "Регулярного выражения"

Регулярное выражение - это способ описания шаблонов для поиска текста и проверки соответствия текста шаблону. Специальные метасимволы позволяют определять, например, что Вы ищете подстроку в начале входной строки или определенное число повторений подстроки.

Модель работы, реализованная в программе "PC 3000 Flash", упрощенная, однако ее функциональности вполне достаточно для успешной работы.

Регулярное выражение может быть создано как вручную, так и с помощью двоичного редактора и плагина *Add Grep* (подробнее см. раздел 1.3.7 *Режим просмотра и редактирования сектора*).

1.1.1.1.1.2 Синтаксис регулярных выражений

Регулярные выражения состоят из символов и операторов. Список операторов приведен в следующей таблице.

?	Любой символ
.	Любой символ
*	Любое количество любых символов
"xxx"	Строка символов
\000	Символ (если начинается с 0 – в восьмеричной кодировке)
\999	Символ (если начинается с 1..9 – в десятичной кодировке)
\xHH	Символ с шестнадцатеричным кодом HH

\n	Перевод строки
\r	Перевод каретки
\t	Табуляция
\f	Перевод формата
[xxxx]	Любой из множества символов. Можно задавать диапазоном "A-Z"
[-xxxx]	Любой не из множества символов
%x	Специальные наборы символов: W - любое количество пробелов w - одиночный пробел N - любое количество символов "0123456789" n - одиночная символ "0123456789" A - любое количество символов из "a-z" или "A-Z" a - одиночный символ из "a-z" или "A-Z" X - любое количество символов из "0-9" или "a-z" или "A-Z" x - одиночный символ из "0-9" или "a-z" или "A-Z"
^	С начала строки
@nnn	Начиная с nnn – позиции от начала
@-nnn	Начиная с nnn – позиции от конца
(xxx)	Подвыражение
(min:max)	Количество повторов следующего оператора
|	Логическое “ИЛИ” для двух подвыражений

Любой символ совпадает с самим собой, если он не относится к специальным метасимволам. Последовательность символов совпадает с такой же последовательностью во входной строке, так что шаблон "bluh" совпадет с подстрокой "bluh" во входной строке.

Если необходимо, чтобы метасимволы или escape-последовательности воспринимались как обычные символы, их нужно предварять символом "\", например, метасимвол "^" обычно совпадает с началом строк, однако, если записать его как "\^", то он будет совпадать с символом "^", "\\" совпадает с "\" и т.д.

Примеры:

Выражение	**Находит**
Foobar	'foobar'
\^FooBarPtr	'^FooBarPtr'
^GIF8[79]a	В начале строки GIF87a или GIF89a
@510\x55\xAA	Байты $55 $AA на позиции @510
(@3"MS") * (@510\x55\xAA)	Сочетание строки “MS” на 3 позиции и байтов $55 $AA на позиции 510
^\xF8{12}(\xFF)	С начала стоки байт $F8 и 12 последующих $FF
foo\x20bar	'foo bar' (обратите внимание на пробел посередине)
Foob[aeiou]r	'foobar', 'foober' и т.д. но не 'foobbr', 'foobcr' и т.д

1.1.1.1.1.3 Работа со справочником

Внешний вид справочника представлен на Рисунке 1.

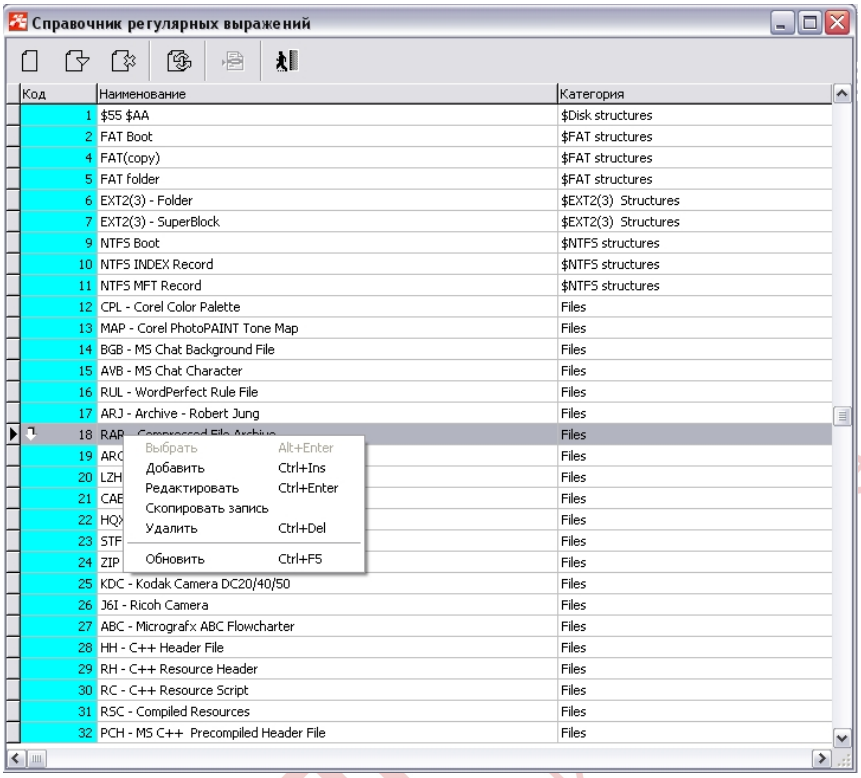


Рисунок 1 Справочник регулярных выражений

Работа со справочником осуществляется либо через контекстное меню списка, появляющееся при нажатии правой клавиши мышки, либо с использованием кнопок "быстрого доступа", находящихся на панели в верхней части окна, либо (что рекомендуется) с использованием комбинаций клавиш "быстрого доступа", которые можно увидеть в контекстном меню.

Имеется возможность сортировки данных по возрастанию или убыванию значений любой из колонок, а также возможность быстрого позиционирования на записи, начинающейся с определенного символа (для этого нужно только нажать на клавиатуре требуемый символ, учитывая его регистр).

Команда *Выбрать* становится доступной, когда справочник используется для выбора критериев поиска данных.

При выборе пунктов *Добавить* и *Редактировать* на экране появляется форма редактирования реквизитов записи (рисунок ниже).

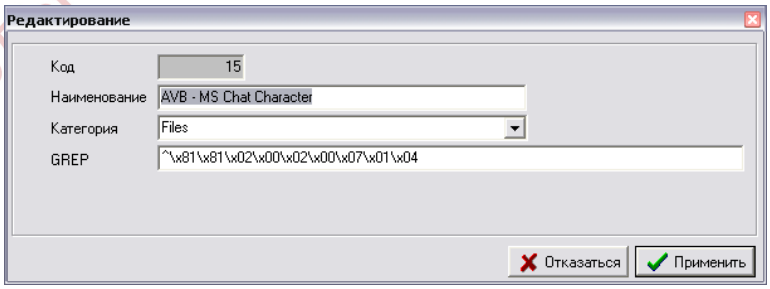


Рисунок 2 Форма редактирования записи регулярного выражения

При добавлении желательно обдуманно назначать категорию и имя, хотя для поиска имеет значение лишь поле *GREP*. Необходимо учитывать, что создание и использование сложных выражений существенно замедляет процесс поиска. Быстрее всего происходит поиск простых выражений, начинающихся с указания позиции: '^' или '@'.

1.1.1.1.2 Справочник "чернового" восстановления

Данный справочник используется в ходе работы метода чернового восстановления, который будет описан ниже.

Внешний вид справочника "чернового" восстановления приведен на рисунке ниже.

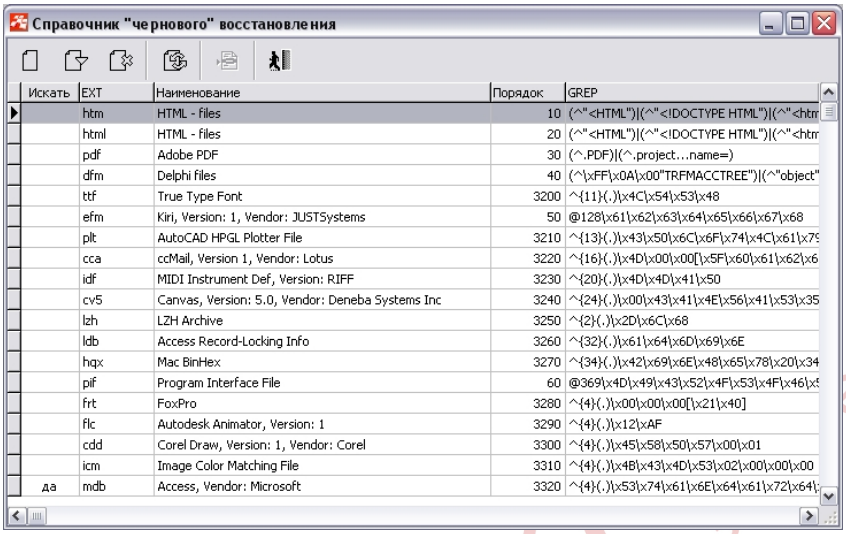


Рисунок 3 Справочник “чернового” восстановления

Работа с этим справочником очень похожа на работу с вышеописанным справочником GREP.

Существенным отличием является лишь набор полей, которые хорошо видны на форме редактирования:

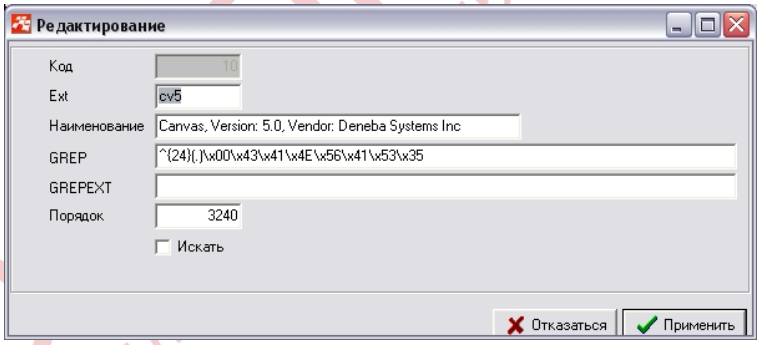


Рисунок 4 Редактирование записи справочника "чернового" восстановления

При осуществлении поиска с помощью справочника "чернового" восстановления используются только регулярные выражения отмеченные в столбце *Искать*. Для расширения списка используемых при поиске выражений необходимо либо пометить соответствующее существующее выражение из справочника, либо создать такое выражение и установить флажок в поле *Искать*.

При запуске "чернового" восстановления, пользователю предоставляется возможность выбора способа расчета размера найденных данных. Существует два варианта. В первом, при расчете размера используются только регулярные выражения отмеченные в столбце *Искать* справочника. Во втором, используются все регулярные выражения справочника (но только при расчете размера данных). Конкретный способ расчета размера зависит от искомых данных, например, для архивов *ZIP*, предпочтительнее первый способ, т.к. если размер архива будет больше его реального размера, то ряд программ для восстановления данных из повреждённых архивов (например ZipFix) позволит извлечь из него данные. В случае если архив меньше своего реального размера вам вряд ли удастся извлечь из него даже часть данных.

Помимо этого, значимыми для поиска и обработки результатов являются поля *Ext* (расширение), *GREP* (основной критерий), *GREPEXT* (уточняющий критерий) и поле *Ord* (порядок анализа).

Поле *GREPEXT* служит для уточнения типа найденного заголовка файла в случае, когда несколько типов файлов имеют одинаковый заголовок и различия дальше в теле файлов.

Особое значение имеет поле *Ord* – его наличие связано с возможностью того, что одно из анализируемых GREP является упрощенным подмножеством другого регулярного выражения. Чтобы получить верный результат, необходимо сначала искать более сложное и лишь после него менее сложное.

Существует возможность редактирования данного справочника - это достаточно сложное и ответственное дело. Однако при создании задачи восстановления данных создается копия справочника "чернового" восстановления, с которой и происходит работа в текущей задаче. Соответственно, в случае необходимости справочник можно восстановить. С другой стороны, если Вы хотите, чтобы внесенные изменения были сохранены, то следует редактировать справочник, относящийся не к текущей задаче, а к программе PC-3000 Flash.

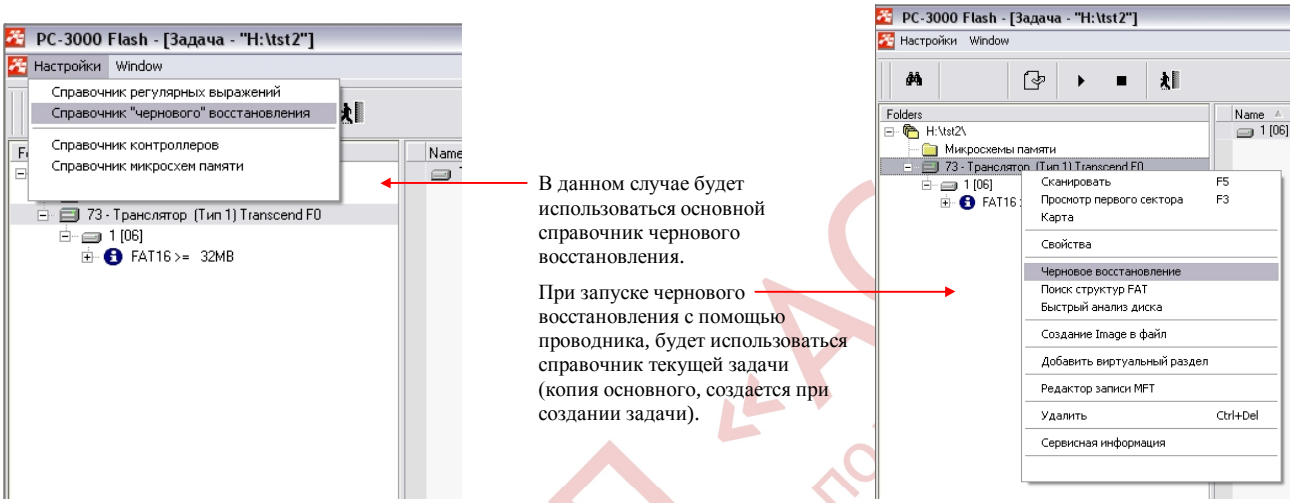
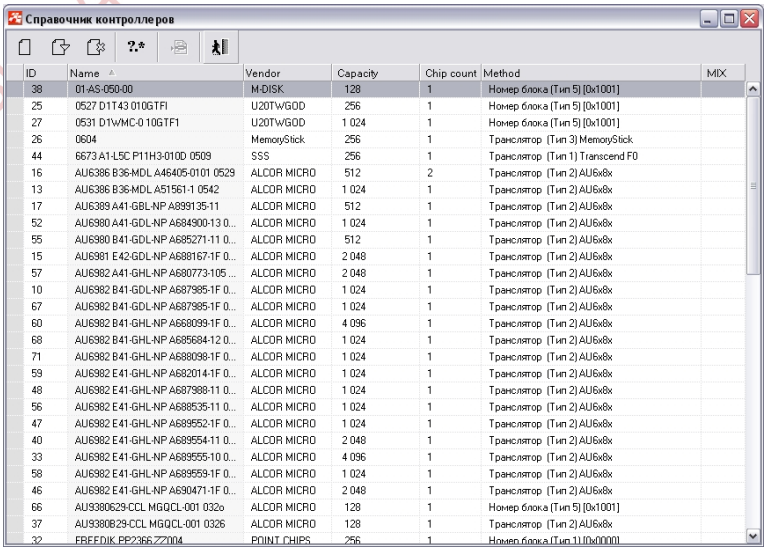


Рисунок 5 Справочник “чернового” восстановления и его копия в текущей задаче

Основной справочник "чернового" восстановления PC-3000 Flash используется, если запустить метод "чернового" восстановления из меню главной формы. Справочник текущей задачи используется в том случае, если метод "чернового" восстановления запущен с помощью панели быстрого доступа или любого доступного метода проводника.

1.1.1.2 Справочник контроллеров.

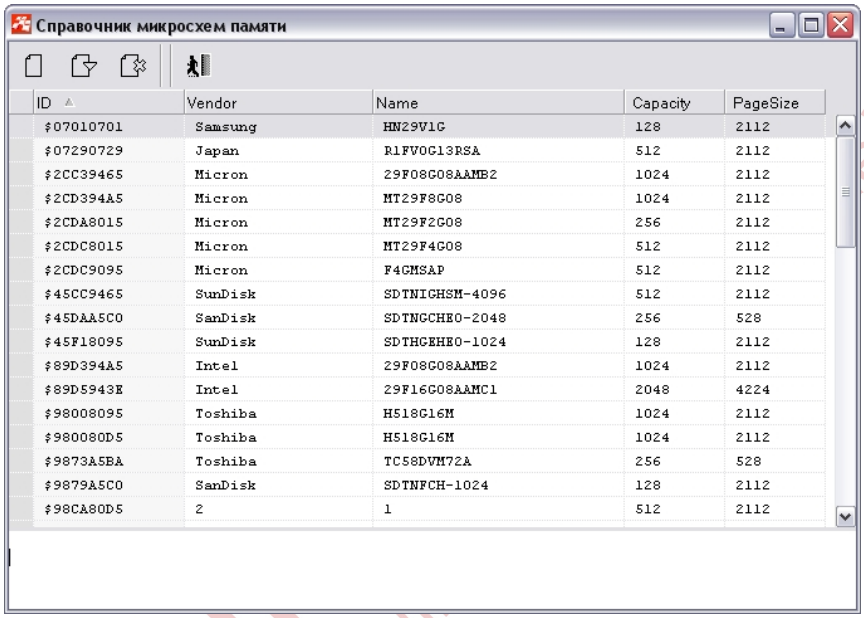
Программа содержит список поддерживаемых в автоматическом режиме контроллеров с ассоциированным им набором необходимых для восстановления данных действий, включая как методы предварительной подготовки, так и непосредственно метод анализа. Справочник доступен для ознакомления посредством данного пункта меню. Непосредственное применения справочника осуществляется через метод «Восстановление по контроллеру» (описан в краткой версии документации).



Справочник пополняется синхронно с выходом обновления программного обеспечения комплекса.

1.1.1.3 Справочник микросхем памяти.

В комплексе присутствует база микросхем памяти, в которой содержатся все необходимые для корректной работы параметры и особенности протокола и внутренней структуры отдельного типа микросхем памяти. Через данный пункт меню справочник доступен для ознакомления и редактирования параметров микросхем.



ID	Vendor	Name	Capacity	PageSize
$07010701	Samsung	HN29V1G	128	2112
$07290729	Japan	R1FV0G13RSA	512	2112
$2CC39465	Micron	29F08G08AAMB2	1024	2112
$2CD394A5	Micron	MT29F8G08	1024	2112
$2CDA8015	Micron	MT29F2G08	256	2112
$2CDC8015	Micron	MT29F4G08	512	2112
$2CDC9095	Micron	F4GMSAP	512	2112
$45CC9465	SunDisk	SDTNIGHSM-4096	512	2112
$45DAA5C0	SanDisk	SDTNGCHE0-2048	256	528
$45F18095	SunDisk	SDTHGEHE0-1024	128	2112
$89D394A5	Intel	29F08G08AAMB2	1024	2112
$89D5943E	Intel	29F16G08AAMC1	2048	4224
$98008095	Toshiba	H518G16M	1024	2112
$980080D5	Toshiba	H518G16M	1024	2112
$9873A5BA	Toshiba	TC58DVM72A	256	528
$9879A5C0	SanDisk	SDTNFCH-1024	128	2112
$98CA80D5	2	1	512	2112

1.1.2 Информационная панель

Внешний вид панели приведен на рисунке:

Рисунок 6 Информационная панель задачи

Панель *Устройство* – содержит информацию о подключенном накопителе (модель, серийный номер, версию прошивки, емкость накопителя и общее количество доступных секторов (LBA)).

Панель *Режим* – содержит наименование текущего режима работы.

Панель *Операция* – содержит дополнительную информацию во время выполнения режима (например, наименование текущей операции, ее статус, возможно номер текущего LBA, и прочее).

Данная панель присутствует практически во всех режимах за исключением режима “Проводник”, в котором ее наличие существенно уменьшило бы рабочую область, что явно нежелательно.

1.1.3 Панель протокола

Эта панель названа так условно. На самом деле она является "блокнотом" с несколькими закладками, состав которых зависит от задачи и режима и может меняться.

Панель присутствует практически во всех режимах и отличается только составом закладок и их настройкой.

1.1.3.1 Закладка "Протокол"

Закладка *Протокол* присутствует всегда. Протокол является общим для всех режимов работы программы.

Кроме собственно протокола, куда выводятся сообщения, включает в себя ползунковый индикатор хода выполнения текущего процесса (нижняя часть закладки) и органы управления выводом данных в протокол (ползунковый регулятор уровня детализации вывода отладочной информации и панель кнопок управления).

Внешний вид закладки приведен на рисунке:

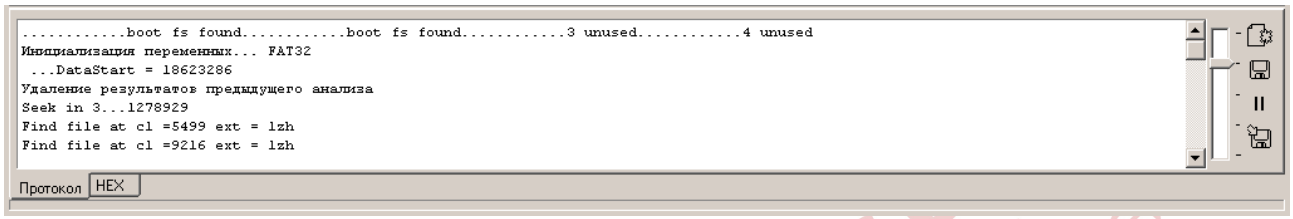


Рисунок 7 Закладка "Протокол"

Регулятор уровня детализации вывода отладочной информации позволяет управлять интенсивностью потока текстовых данных, выводимых в протокол. Если ползунок находится в верхнем положении – выводится полная информация о ходе процесса (вплоть до читаемого сектора), если в нижнем – выводится информация только о значимых ошибках. Чем выше положение регулятора, тем больше поток выводимой информации.

Объем протокола определяется настройками программы PC-3000 Flash и при его превышении часть протокола автоматически чистится.

Кнопка *Очистить протокол* служит для полной очистки протокола.

Кнопка *Сохранить журнал в файл* позволяет сохранить текущее состояние журнала в файл.

Кнопка-триггер *Пауза* позволяет, не прерывая вывод данных в протокол, остановить его автоматическую прокрутку в конец, что дает возможность без неудобств просматривать протокол во время интенсивного вывода в него данных. Кнопка может быть в нажатом и отжатом состоянии.

Кнопка-триггер *Автосохранение протокола в файл* позволяет одновременно с выводом информации в протокол сохранять ее в файл, и в нем сохранять полный Log задачи без потерь при очистке протокола.

1.1.3.2 Закладка "HEX"

Закладка служит для отображения первого сектора выбранного объекта и появляется в тех случаях, когда можно говорить об объектах данных, текущем объекте, его положении и первом секторе.

В данной реализации это режимы *Проводник* и *Карта объекта*.

Закладка HEX может выглядеть немного по-разному в зависимости от настроек задачи.

С помощью контекстного меню закладки можно изменить такие параметры как количество байт в строке, колонке, шрифт и кодировку.

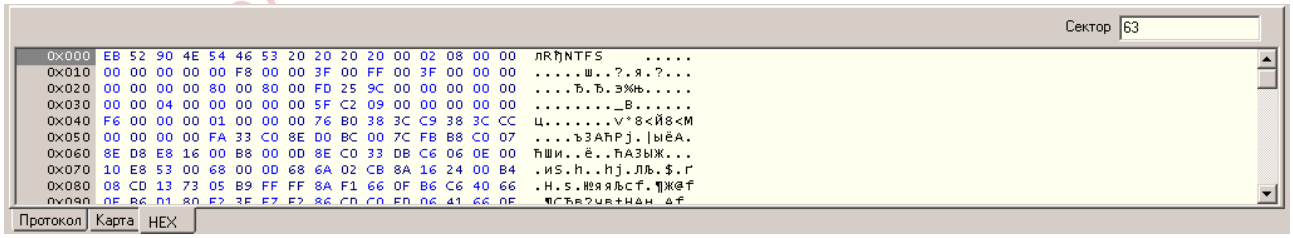


Рисунок 8 Закладка "HEX"

1.2 Основные режимы работы

1.2.1 Режим "Проводник"

Основное назначение данного режима – визуальное представление логической структуры данных, хранящихся на тестируемом накопителе, помощь пользователю в ее понимании, и, при необходимости, модификации.

Свое имя данный режим получил из-за внешнего сходства с *Проводником* Windows (слева – дерево, справа – список, вверху – панель инструментов и контекстное меню при нажатии на правую клавишу мыши).

Состав и структура выводимых данных в проводнике определяются в основном требованием максимально более точно передать принципы организации данных на диске (сущность, размещение, взаимосвязь и т.д.). Методы в свою очередь, также связаны с потребностями задач восстановления данных (Просмотр первого сектора объекта, получения связанного списка секторов объекта и т.д.).

1.2.1.1 Внешний вид, управление и навигация

Внешний вид режима *Проводник* приведен ниже:

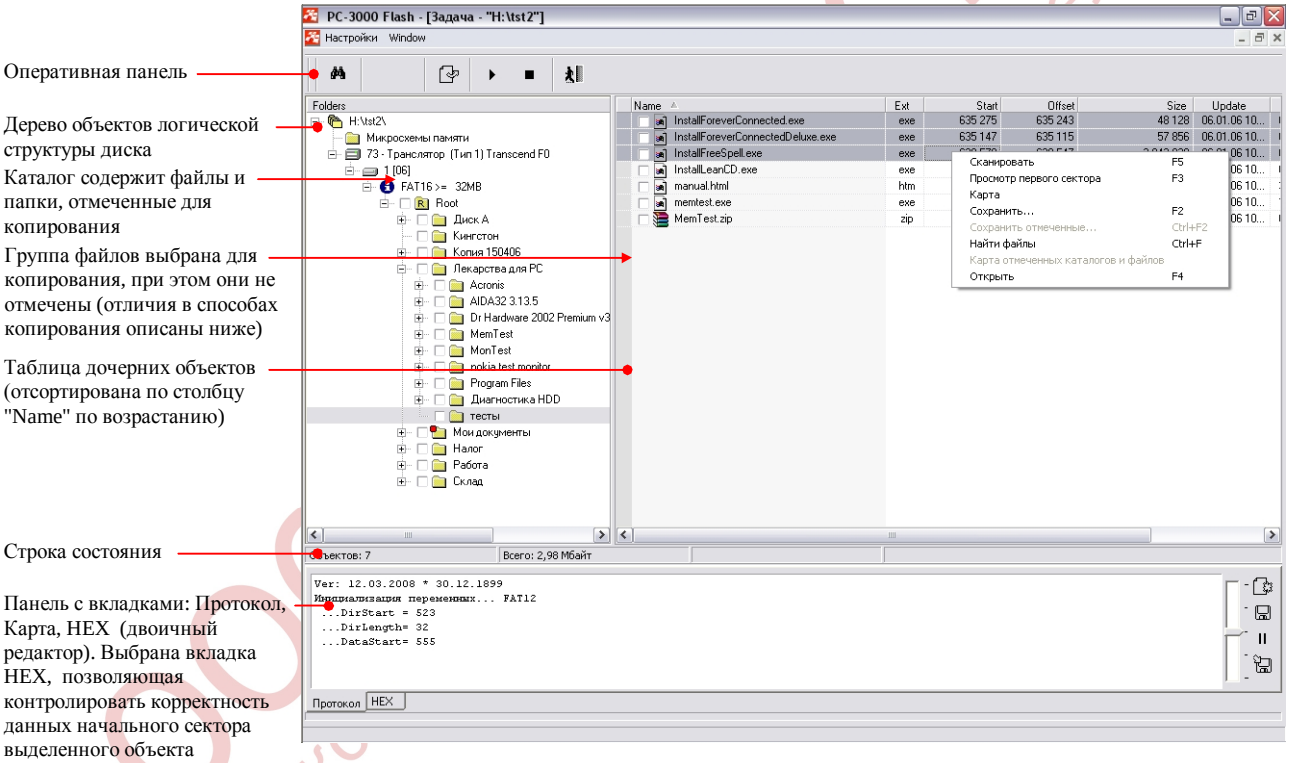


Рисунок 9 Внешний вид режима "Проводник"

В левой части экрана располагается дерево списка объектов логической структуры диска, справа от него располагается список дочерних объектов для текущего выбранного объекта дерева. Навигация по обоим спискам и между ними осуществляется с помощью клавиатуры или с помощью мыши. Все возможные действия над выбранным в данный момент элементом любого из списков доступны через контекстное меню (некоторые пункты контекстного меню дублируются набором "горячих" клавиш).

Таблица дочерних объектов по умолчанию отсортирована по возрастанию ("Δ") по столбцу *Name*. Пользователь может изменить способ (по возрастанию "Δ", по убыванию "∇") и столбец сортировки, щелкнув на заголовке требуемого столбца левой клавишей мыши. Помимо способа сортировки пользователь может изменить ширину столбцов. Измененные пользователем настройки по умолчанию сохраняются в параметрах текущей задачи. В таблице дочерних объектов для обозначения файлов используются иконки, ассоциированные в Вашей ОС с данным типом данных, для всех остальных файлов используется иконка -  .

Над списками расположена оперативная панель кнопок "быстрого" доступа. Оперативная панель кнопок "быстрого доступа" разбита на несколько групп (см. рисунок ниже):

- Дополнительные режимы (*Поиск регулярных выражений*);
- Управление выполняемым процессом (*Пропустить* и *Прервать процесс*);
- Выход из задачи.

Рисунок 10 Оперативная панель режима "Проводник"

Кнопка *Прервать* служит для прерывания любого из выполняемых в данный момент проводником процессов ("раскрутка" каталога, копирование и т.п.). Появление данной кнопки (при отсутствии кнопки *Начать*) и особое внимание к ней связано с тем, что в ходе работы с дисками с проблемами в логической структуре, может возникнуть нештатная ситуация (зацикливание, копирование файла с ошибочно-большим размером, чтение большой области разрушений), которая может потребовать прерывания.

Под списками расположена строка состояния, служащая для отображения дополнительной информации о выбранном элементе дерева (число вложенных объектов и их суммарный объем (только для каталогов)).

В нижней части окна находится панель с закладками протокола и двоичного редактора для просмотра текущего сектора. Состав закладок зависит от настроек задачи.

В ходе восстановления данных могут возникнуть непредвиденные сложности, связанные с логическими разрушениями данных, которые, в свою очередь, могут привести к проблемам при выполнении логического разбора с использованием проводника.

Такая непредсказуемость предполагает реализацию двух основных требований:

- возможность получения максимума информации о ходе выполняющегося в данный момент процесса (вплоть до информации о номере считываемого сектора);
- возможность в любой момент прервать исполнение любого процесса.

Для реализации перечисленных выше требований в режиме *Проводник* используются три основных элемента – панель для вывода протокола работы и служебной информации, кнопка *Прервать* и ползунковый регулятор уровня выводимой информации о ходе исполняющихся в данный момент процессов.

Окно для вывода протокола работы и служебной информации комментариев не требует.

Как уже упоминалось выше, кнопка *Прервать* служит для прерывания текущего исполняемого процесса. Следует знать, что в ходе работы проводника активно используется многопоточная модель работы Windows и нажатие кнопки должно привести к прерыванию исполняемого в данный момент активного процесса (например, "раскрутки" каталога, копирования файлов, чтения FAT и т.д.). В силу особенностей работы с неисправным накопителем некоторые процессы (чтение сектора, неважно, успешное или неуспешное) должны быть определенным образом завершены, иначе придется снимать питание с накопителя, подавать его снова и т.п. Поэтому иногда нажатие кнопки *Прервать* не приводит сразу к видимым результатам (появлению в окне информации соответствующего сообщения, прекращению работы с накопителем и т.д.). Нажмите кнопку *Прервать* несколько раз.

Регулятор уровня отладочной информации на вкладке *Протокол* позволяет регулировать объем выводимой информации о ходе процесса. Верхнее положение регулятора – полная информация, нижнее – ее отсутствие, промежуточные – соответственно, более или менее полная.

В случае, когда в ходе создания задачи была включена опция *восстановление транслятора*, справа от списка объектов появляется *Таблица сдвигов* о которой подробно будет рассказано в разделе *Режим восстановления транслятора*.

1.2.1.2 Объекты проводника

Объекты, которые отображаются на панелях списков проводника, являются элементами логической структуры данных накопителя. Их назначение приводится в таблице ниже.

Иконка	Наименование	Описание
	MBR	Главная загрузочная запись MBR (Master Boot Record, 0 сектор)
	Primary/Secondary Partition	Первичный/вторичный раздел Partition Entry (16-байтная структура)
	Extended Partition	Расширенный раздел Partition Entry (16-байтная структура)
	Boot	Загрузочный сектор раздела
	Root	Корневой каталог раздела
	Folder	Каталог файловой системы
	Lost&Found	Искусственный каталог, в который будут помещаться каталоги и файлы, найденные в ходе выполнения методов логического восстановления, для которых не найден родительский каталог
	File	Файлы раздела. + – для файла, имеющего подобный признак, содержимое первого сектора совпадает с ожидаемым для данного типа файлов, (аналогичное значение имеет зеленый индикатор, примененный к каталогу); + – для файла, имеющего подобный признак, содержимое первого сектора не совпадает с ожидаемым для данного типа файлов, (аналогичное значение имеет зеленый индикатор, примененный к каталогу). Косвенно может служить признаком поврежденного файла. + – удаленный файл или папка.

Каждому типу объектов, выводимых на панели проводника, соответствуют свои пиктограммы, текстовая информация и набор действий, доступных через контекстное меню объекта. Состав этих действий определяется типом объекта и типом файловой системы, к которой относится данный объект. Контекстное меню можно вызвать нажатием правой клавиши мыши или сочетанием клавиш "Alt+Down".

Общими почти для всех перечисленных объектов являются методы *Сканировать* и *Просмотр первого сектора*.

Сканировать – данный метод означает перечитать заново (возможно, в первый раз) всю информацию, связанную с выбранным объектом и обновить список дочерних объектов. Естественно, что для каждого типа объекта определено свое действие (для MBR – 0 сектор, для каталога – соответствующий и определяемый файловой системой список секторов и т.д.). Метод не определен для объектов типа *File*, т.к. вся информация о файлах содержится в объекте *Folder*.

Просмотр первого сектора - загружает первый сектор выбранного объекта в редактор двоичных данных для просмотра (и редактирования при необходимости). В некоторых случаях, данный метод может оказаться недоступным, хотя он определен для всех объектов проводника. Это возможно в нескольких случаях:

- объект создан искусственно;
- недостаточно данных об объекте (в случае NTFS разделов может возникнуть ситуация, когда информация о файле получена из каталога, а соответствующая объекту, запись MFT, отсутствует или разрушена, в результате в проводнике появляется файл с нулевым размером).

Помимо этого, существует одна неоднозначность, которая требует пояснений. В случае NTFS разделов, встает вопрос о том, что понимать под первым сектором – первый сектор соответствующей записи MFT или первый сектор потока данных. Эта неоднозначность решена исходя из местонахождения потока данных объекта. Для данных, размещенных резидентно (внутри MFT), под первый сектором подразумевается (и отображается) первый сектор записи MFT (хотя зачастую, поток данных находится во втором секторе записи). Для данных, размещенных не резидентно – это первый сектор потока данных размещенных вне MFT.

1.2.1.3 Главная загрузочная запись MBR

Для сектора MBR, кроме перечисленных выше, доступны следующие методы:

– *Карта* (для объекта MBR под картой понимается вся поверхность накопителя; в списке цепочек, отображаются доступные разделы диска);

– *Свойства* (на экране отображается окно таблицы разделов, данные таблицы доступны для редактирования, см. рисунок ниже);

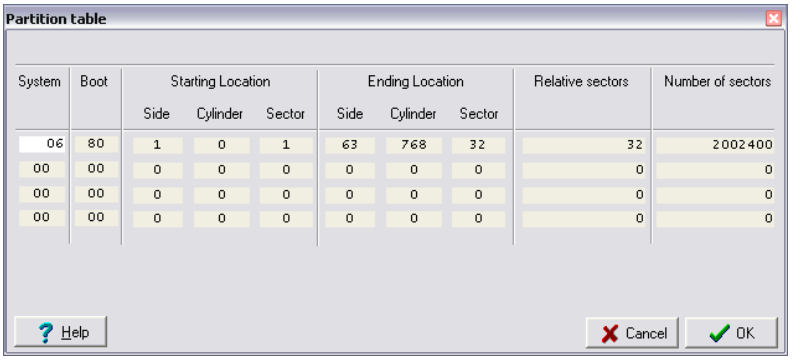


Рисунок 11 Окно описателей накопителя (MBR) вызываемое методом "Свойства"

– *Черновое восстановление* (черновое восстановление запускается на весь накопитель с интервалом равным единице, т.к. информация о размере кластера располагается в boot-секторе раздела и на этом этапе еще не прочитана, подробнее см. раздел *Черновое восстановление*);

– *Поиск структур NTFS* (автоматический режим, предназначенный для восстановления NTFS разделов, подробнее см. раздел *Автоматические методы восстановления NTFS разделов)*;

– *Быстрый анализ диска* (автоматический режим, предназначен для создания виртуальной таблицы разделов в случае, когда MBR не доступен);

– *Создание Image в файл* (данный режим создает копию данных на основании *partition table* в файл указанный пользователем для возможного анализа сторонними программными продуктами);

– *Редактор записи MFT* (режим предназначен для визуального представления логической структуры записи MFT разделов NTFS, подробнее см. подраздел 1.3.8 *Редактор записи MFT*);

Карта. Карта MBR или EPR может быть использована, например, для быстрого доступа к boot-сектору соответствующего раздела или копии boot-сектора, если она есть. Для разделов FAT32, копия boot-сектора обычно находится через шесть секторов от основного, а для NTFS разделов копия находится в конце раздела. При необходимости, найденные копии могут быть использованы для восстановления основного boot-сектора раздела (через буфер обмена или файл) или создания виртуального раздела.

Быстрый анализ диска. Данный метод автоматического поиска предназначен для создания виртуальной таблицы разделов в случае, если MBR исследуемого накопителя по каким-либо причинам недоступен или повреждён.

Виртуальная таблица разделов строится на основании найденных загрузочных секторов FAT и NTFS разделов и их копий. Сначала, поиск загрузочных секторов производится в небольших областях в начале и конце диска. В том случае, если загрузочные сектора найдены, то на основании их данных в виртуальную таблицу разделов заносится информация о границах разделов, которые они описывают. Далее поиск осуществляется рекурсивно, т.е. имея информацию о границах разделов, найденных на первом этапе, мы можем выполнить поиск загрузочных секторов на границах найденных на предыдущем этапе разделов. И т.д.

Метод имеет ограничения. Т.к. поиск загрузочных секторов осуществляется только в небольших областях в начале и конце исследуемого диска (на первом этапе), можно спрогнозировать ситуацию, в которой данный метод даст отрицательные результаты. Предположим, что диск имел два раздела – первый NTFS, у которого разрушен загрузочный сектор и второй раздел FAT. В этом случае, поиск в небольшой области в начале диска результатов не даст т.к. загрузочный сектор NTFS раздела разрушен. Поиск в конце диска результатов тоже не даст т.к. разделы FAT имеют загрузочный сектор и его копию в начале раздела (соответственно, если это второй раздел, то эти сектора будут находиться где-то в середине диска, если разделы приблизительно одинаковые по размеру). Соответственно, на первом этапе поиска мы не обнаружим загрузочных секторов (или их копий) на основании которых мы бы смогли продолжить поиск. В этом случае следовало бы воспользоваться поиском по всему разделу загрузочных секторов, но этот способ поиска очевидно нельзя назвать быстрым.

Данный режим является неким упрощенным (из-за этого и быстрым) и полностью автоматическим способом решения проблемы наиболее простых логических разрушений. Если данный метод не дает положительных результатов, то можно воспользоваться более сложными методами логического восстановления (для FAT и NTFS разделов) которые будут описаны ниже. Однако для этих методов желательно (а в большинстве случаев необходимо) иметь полную копию всех данных для проведения сложного и длительного анализа.

Добавить виртуальный раздел. Виртуальный раздел представляет собой искусственный раздел в режиме *Проводник*, полученный в результате создания boot-сектора раздела в базе данных задачи (без записи данных на исследуемый диск). В результате создания такого искусственного загрузочного сектора, в проводнике появляется новый раздел выбранного при создании типа, с параметрами задаваемыми этим boot-сектором. Для виртуального раздела созданного вручную, доступны все методы обычного раздела в отличие от виртуальных разделов создаваемых в результате работы каких-либо автоматических режимов восстановления информации (например, *логического восстановления*). При необходимости, можно сохранить виртуальный boot-сектор на диск.

При создании виртуального раздела пользователь должен задать тип создаваемого раздела, начальный и конечный LBA и размер кластера (рисунок ниже). После этого, пользователю предлагается окно для корректировки остальных параметров загрузочного сектора.

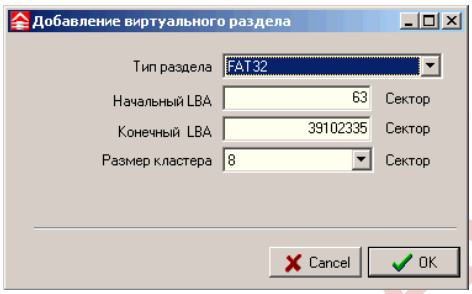


Рисунок 12 Окно задания параметров виртуального раздела

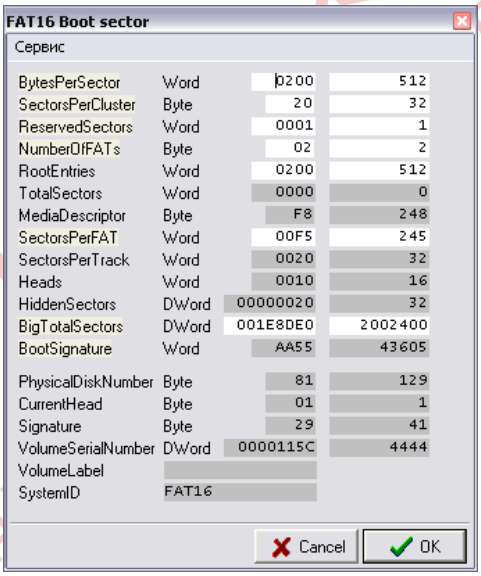


Рисунок 13 Окно редактирования boot-сектора виртуального раздела

Существует еще один, возможно более удобный в некоторых случаях, способ добавления виртуальных разделов. Данный способ доступен из формы просмотра для загрузочного сектора (*Сервис → Добавить виртуальный раздел* …). В свою очередь, форму просмотра можно вызвать из окна двоичного редактора (*Просмотр как…*).

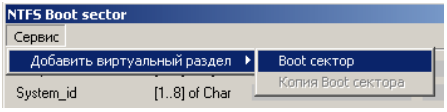


Рисунок 14 Добавление виртуального раздела через форму просмотра загрузочного сектора

В качестве примера, предположим, что у нас есть накопитель, у которого разрушен сектор *MBR*. Мы можем запустить поиск *GREEP* загрузочных секторов в небольшой начальной области накопителя (*FAT, NTFS*). Допустим, по сигнатуре был найден загрузочный сектор NTFS раздела. Открыв данный сектор в двоичном редакторе, воспользуемся методом *Просмотр как …→ Boot NTFS*, если данные сектора кажутся корректными, мы можем добавить этот сектор в качестве виртуального раздела (наиболее важные параметры загрузочного сектора в данном режиме проверяются автоматически и в том случае, если они выходят из допустимых границ –

подсвечиваются желтым фоном). При этом производится проверка – может ли быть данный сектор копией или это оригинал. Проверка включает в себя расчет положения раздела (не выходит ли он за границы накопителя), контроль положения другой копии загрузочного сектора и таблиц MFT и MFT Mirror для разделов NTFS и таблиц FAT для FAT разделов. В том случае, если произведенные проверки не дают однозначного ответа является ли данный сектор копией или это оригинал, в списке контекстного меню остаются доступными оба варианта (*Boot сектор* и *Копия Boot сектора*), иначе – в списке остается доступным только один возможный и корректный вариант.

1.2.1.4 Partition Entry

Для объекта *раздел* доступны следующие методы:

– *Карта*;

– *Черновое восстановление* (черновое восстановление запускается на раздел с интервалом равным единице, т.к. информация о размере кластера располагается в boot-секторе раздела и на этом этапе еще не прочитана, подробнее см. раздел *Черновое восстановление*).

– *Поиск структур NTFS* (автоматический режим, предназначенный для восстановления NTFS разделов, подробнее см. раздел *Автоматические методы восстановления NTFS разделов)*;

1.2.1.5 Загрузочный сектор раздела (boot)

Существенные различия в доступных методах имеют объекты, соответствующие загрузочным (*boot*) секторам разделов. У этих объектов есть методы, присущие всем типам разделов.

1.2.1.5.1 Общие методы присущие всем типам разделов

– *Свойства*;

– *Карта раздела*;

– *Черновое восстановление* (черновое восстановление запускается на раздел с интервалом равным размеру кластера, подробнее см. раздел *Черновое восстановление*).

Свойства. Данный метод позволяет просмотреть (отредактировать) структуру загрузочного сектора раздела. В качестве примера на рисунке ниже приводится окно загрузочного сектора раздела с FAT16.

Замечание! Значения в светлых окошках формы значимы для корректного восстановления информации данного раздела. При редактировании данных полей производится проверка корректности введенных данных. В случае если введенные данные не корректны, название параметра выделяется с помощью желтого фона.

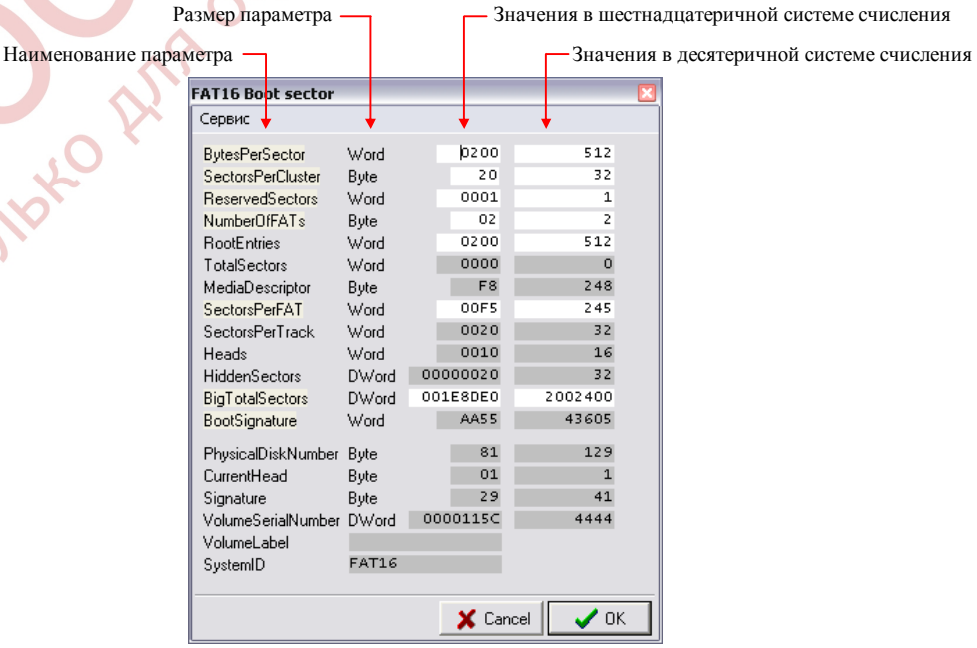


Рисунок 15 Просмотр загрузочного сектора раздела FAT16 с помощью метода "Свойства"

Карта раздела. Данный режим, может быть использован:

– для проверки и коррекции положения важных с точки зрения восстановления метаданных раздела при восстановлении транслятора (копий загрузочного сектора, копий таблицы FAT, начала таблицы MFT и т.д.)

– для проверки местоположения и целостности основных метаданных раздела в случае логических разрушений (быстрый доступ к копиям загрузочного сектора, к таблицам FAT)

– в случае работы с неисправным накопителем можно выбрать наиболее важные с точки зрения цепочки (например, таблицы FAT, boot-сектора) и вычитать их в копию с наиболее жёсткими параметрами (увеличенное количество попыток чтения), оценить качество вычитывания.

Подробнее об использовании данного режима см. раздел *Карта объекта*. Внешний вид режима *Карта раздела* приводится ниже.

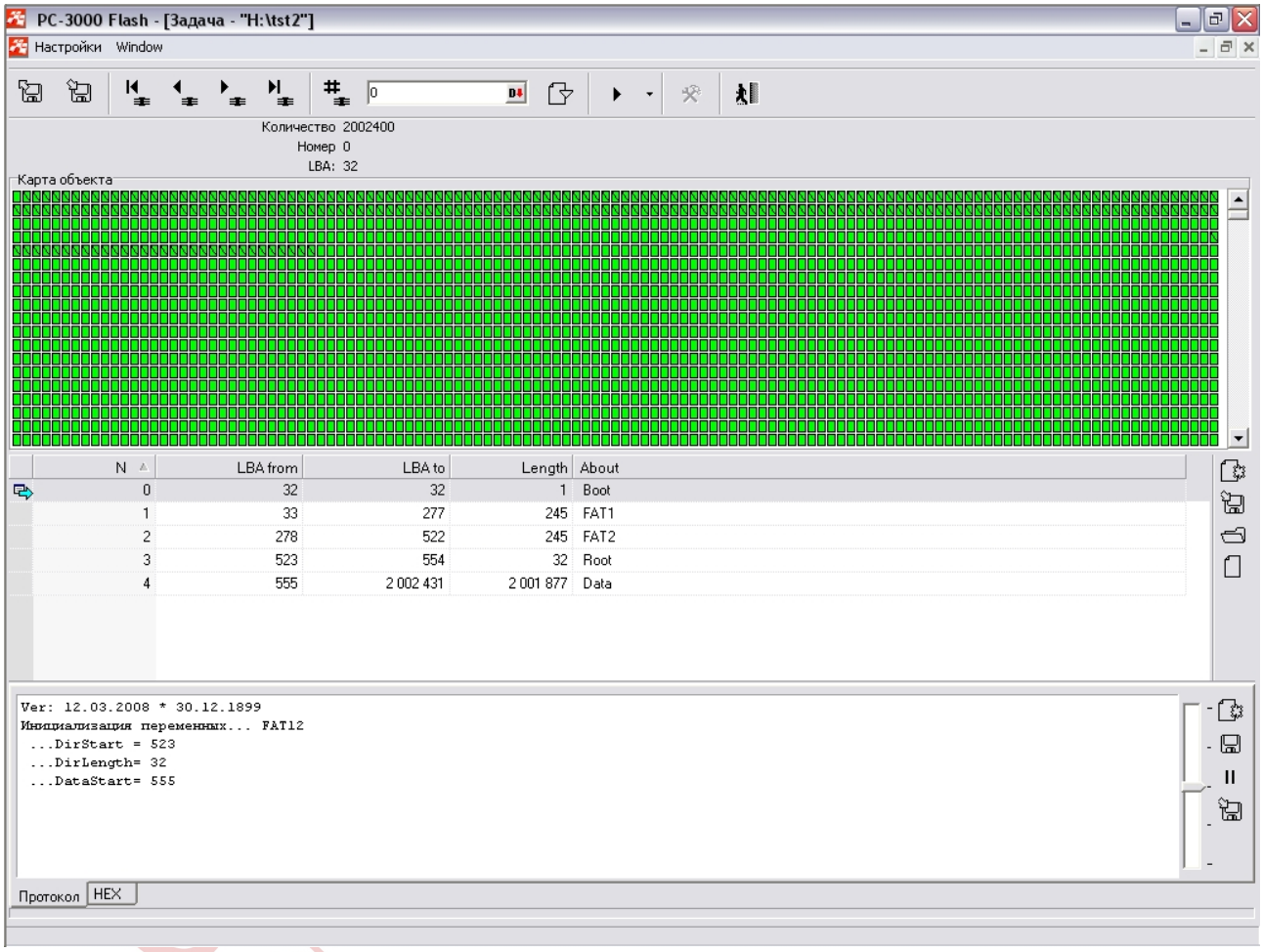


Рисунок 16 Режим "Карта раздела" для раздела FAT16

Кроме этого, в состав меню включены методы, присущие лишь конкретному типу раздела.

1.2.1.5.2 Дополнительные методы для разделов FAT

– *Карта занятых секторов*. Для разделов FAT строится карта занятого пространства раздела по активной копии FAT. Карта позволяет снизить временные затраты и нагрузку на накопитель.

Например, в том случае, если удается построить карту занятых секторов и таблицы FAT адресуют восстанавливаемые данные (т.е. раздел не переформатирован), то для сокращения времени и нагрузки на накопитель можно работать только с построенной картой для создания копии данных и не вычитывать сбойные сектора в не адресуемой области раздела.

– *Карта незанятых секторов*. Для разделов FAT строится карта незанятого пространства раздела по активной копии FAT.

В том случае, если восстанавливаемая информация находится в неадресуемой области раздела (один раздел установлен поверх другого и искомые данные были на первом разделе), то имеет смысл искать пользовательские данные, используя режимы *чернового восстановления* или *поиск GREP* по карте незанятого.

– *Карта цепочки секторов*. Позволяет получить полную карту объекта от начального сектора по активной копии FAT.

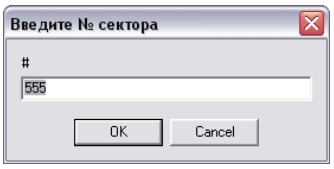


Рисунок 17 Форма для ввода начального сектора при построении карты цепочки секторов

Режим может быть использован для копирования файлов, для которых утеряна информация из слота родительского каталога. В этом случае, с помощью поиска регулярных выражений найдено начало файла и в активной копии FAT есть информация о размещении данного объекта, начинающегося с найденного сектора, то, построив карту цепочек секторов, сохраняем объект на штатный диск. Затем переименовываем его соответствующим образом (поиск регулярных выражений определяет предполагаемое расширение объекта) и получаем файл данных с размером, округленным до размера кластера. Некоторые типы файлов могут не открыться в этой ситуации, поэтому исходный размер файла придется корректировать вручную.

– *Показывать удаленные* – данный пункт определяет, будут ли показываться удаленные файлы и папки в окне проводника текущего FAT раздела.

При этом следует понимать, что данный метод проводника в общем случае не является полным аналогом режима *Анализа данных раздела* с выбранной опцией *Искать удаленные файлы* т.к. в данном случае будут найдены лишь удаленные объекты в подкаталогах раздела FAT, дерево которых строится рекурсивно. При использовании метода *Анализа данных раздела* с выбранной опцией *Искать удаленные файлы*, также могут быть найдены удаленные объекты расположенные в потерянных каталогах (т.е. каталогах, на которые нет ссылок из других папок).

Кроме этого, отображение удаленного объекта в проводнике не гарантирует, что он будет успешно скопирован.

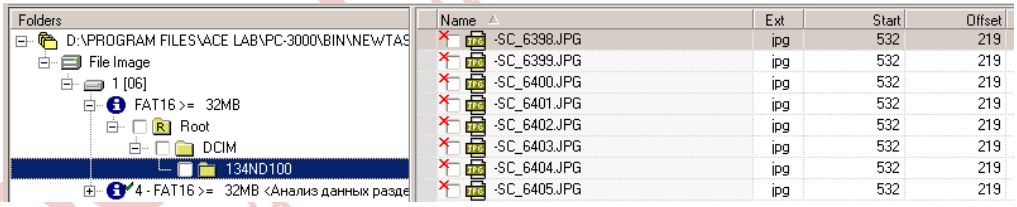


Рисунок 18 Окно проводника с удаленными объектами

– *Контроль целостности подкаталогов*. Как уже отмечалось выше, дерево объектов проводника строится рекурсивно, т.е. имеется корневой каталог, содержащий описатели подкаталогов и файлов, в свою очередь эти подкаталоги также могут содержать свои подкаталоги и файлы и т.д. Предположим, объект корневого каталога (или любого другого) содержит описатель вложенного подкаталога. Если отмечен данный пункт меню, то будет произведена проверка, имеется ли в указанном кластере подкаталог, описываемый соответствующим описателем родительского каталога, и если его там не оказалось, то этот объект не будет отображен на дереве объектов. Если контроль целостности подкаталогов не используется, то в дерево объектов выводятся все подкаталоги без проверки их фактического наличия.

Данный метод не должен использоваться в случае восстановления транслятора, потому что требуемый каталог может на начальном этапе создания виртуального транслятора оказаться не на требуемом месте из-за сдвигов.

– *Контроль целостности описания файлов*. При выборе данного пункта меню, для файлов будет производиться проверка корректности имени, размера и начального сектора (исключение составляет дата). Соответственно, если эти данные не корректные (например, начальный сектор выходит за рамки раздела), то соответствующий объект в проводнике не отображается.

– *Использовать копию Boot* (обеспечивает возможность работы с разделом FAT32 используя данные копии загрузочного сектора).

– *Копии FAT … → 1 копия (2 копия, Игнорировать FAT)* (обеспечивает возможность выбрать номер копии FAT, с которой будет выполняться дальнейшая работа или оказаться от учета FAT).

– *Копии FAT … → Подключить файл в качестве копии таблицы FAT.*

С помощью режима *Карта раздела*, можно сохранить копию FAT в файл, а с помощью данного метода, возможно подключить сохраненную копию (возможно исправленную вручную или с помощью сторонних программных средств) для адресации раздела. При подключении файла, в контекстном меню раздела появляется соответствующая запись (рисунок ниже).

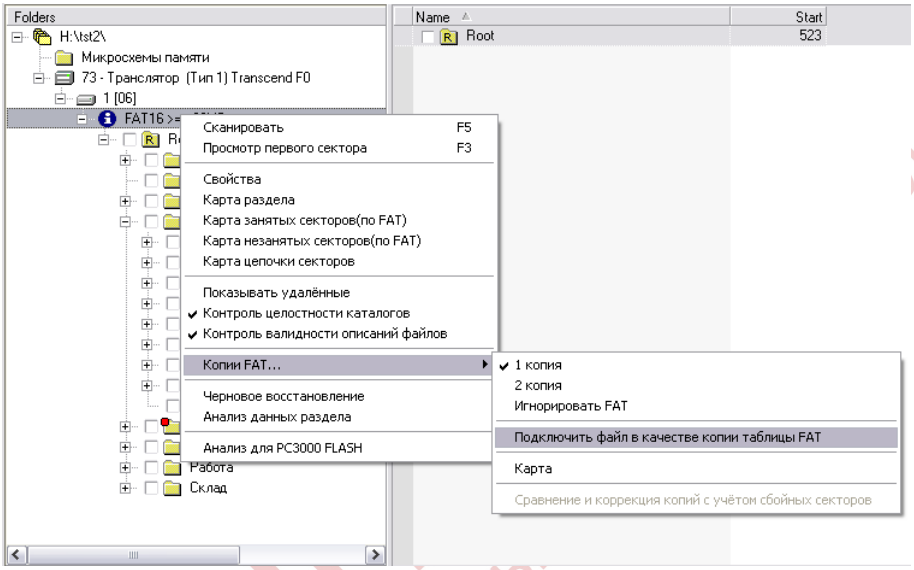


Рисунок 19 Подключение файла в качестве копии таблицы FAT

Следует иметь ввиду, что при выходе из задачи и в случае сканирования объектов "владельцев", данная настройка не сохраняется, о чем выводится предупреждающее сообщение при запуске метода (для объекта boot-сектор, такими объектами являются *Задача*, *MBR* и *слот MBR*).

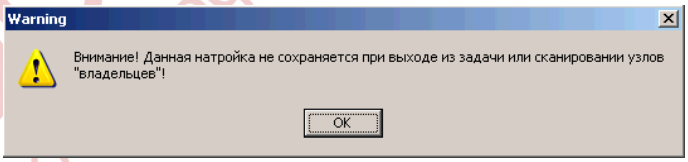


Рисунок 20 Подключение файла в качестве копии таблицы FAT

– *Копии FAT … → Карта* (позволяет просмотреть карту выбранной копии FAT).

– *Копии FAT … → Сравнение и коррекция копий с учетом сбойных секторов*. Данный метод доступен только в режиме создания копии. В результате выполнения, вычитываются обе копии FAT, затем в каждой копии сбойные сектора подменяются секторами из другой копии, если они прочитаны без ошибок.

– *Показывать удаленные* (определяет необходимость отображения удаленных каталогов и файлов).

– *Анализ данных радела* (позволяет выполнить автоматический разбор раздела FAT в случае серьезных логических разрушений, подробнее см. раздел *Анализ данных раздела*).

1.2.1.5.3 Дополнительные методы для разделов NTFS

– *Активный Boot*. У любого NTFS раздела существует два загрузочных сектора – основной, расположенный в начале раздела и копия, расположенная в конце раздела. В случаях если основной загрузочный сектор и его копия совпадают, или один из этих секторов не корректен (или его не удается вычитать, или он просто содержит мусор), мы однозначно можем выбрать загрузочный сектор, с которым

можно продолжить работу и, соответственно, пункт меню *активный boot* становится недоступным. В случае если основной загрузочный сектор и его копия различаются, но оба корректны, пользователю предоставляется возможность выбора одного из них (*Base* и *BackUp* соответственно).

– *Выбор дескриптора таблицы MFT* . Как и в случае дублирования загрузочных секторов, в ФС NTFS дублируются и первые 4 записи таблицы MFT. Для них осуществляется аналогичная проверка: PC-3000 Flash либо однозначно определяет с какими записями (для активного Boot) продолжить работу (при этом пункт меню становится недоступным), либо предоставляет пользователю возможность выбора если обе копии корректны но различаются между собой (*Base* и *Mirror* соответственно).

– *Карта занятых секторов*. Для разделов NTFS строится карта занятого пространства раздела по $Bitmap. Карта позволяет при работе с неисправным накопителем (с большим количеством bad-секторов) снизить временные затраты и нагрузку на накопитель.

Например, в том случае, если удается построить карту занятых секторов и таблица MFT адресует восстанавливаемые данные (т.е. раздел не переформатирован), то для сокращения времени и нагрузки на накопитель можно работать только с построенной картой для создания копии данных и не вычитывать сбойные сектора в не адресуемой области раздела.

– *Карта незанятых секторов*. Для разделов NTFS строится карта незанятого пространства раздела по $Bitmap.

В том случае, если восстанавливаемая информация находится в неадресуемой области раздела (один раздел установлен поверх другого и искомые данные были на первом разделе), то имеет смысл искать пользовательские данные, используя режимы *чернового восстановления* или *поиск GREP* по карте незанятого.

– *Карта таблицы MFT*. Таблица MFT является наиболее важной составной частью метаданных для NTFS разделов.

В случае накопителей с логическими разрушениями, факт построения карты MFT говорит о том, что мы имеем все метаданные для восстановления информации (если это требуемая таблица MFT, т.е. раздел не переформатирован). Режим позволяет оценить размер таблицы MFT и количество записей в ней. Например, восстанавливается раздел содержащий большое количество данных (несколько тысяч файлов), при этом построенная карта таблицы MFT мала (допустим, занимает 500 секторов, что приблизительно соответствует 250 адресуемым файлам). Это означает, что вероятнее всего, данный раздел был переформатирован и таблица MFT не может быть использована для восстановления требуемой информации т.к. относится к новому разделу.

В случае работы с неисправными накопителями (наличие сбойных секторов), построение карты MFT говорит о том, что мы имеем только информацию о размещении всех необходимых метаданных, которые затем необходимо вычитать с максимальной тщательностью (используя повторы и различные команды чтения). По количеству сбойных секторов можно качественно оценить разрушения метаданных. Чем успешнее удалось вычитать таблицу MFT, тем выше вероятность того, что мы получим доступ к требуемым данным. Успешно считанная таблица MFT дает информацию об атрибутах данных и их размещении. При этом в дальнейшем, нет гарантии в том, что имея информацию о размещении требуемого объекта мы сможем его скопировать. После вычитывания таблицы MFT рекомендуется запустить метод *Сканировать MFT* в результате в проводнике будет построено дерево виртуальной файловой системы, описываемой данной таблицей MFT.

Теоретически, *Карта таблицы MFT* может быть использована для установки точек сдвига для цепочек таблицы MFT при восстановлении транслятора в ручном режиме (более целесообразно воспользоваться автоматическим методом доступным из контекстного меню объекта *слот MBR*, однако возможны ситуации, когда данный метод не может быть использован, подробнее см. раздел *Ограничения использования*).

– *Свойства элемента MFT*. Элемент таблицы MFT можно выбрать либо по номеру записи, либо по номеру LBA. При этом в протокол выводятся краткая сводка указанного элемента таблицы MFT, а полное описание элемента выводится с помощью специализированного редактора (*Редактор записи MFT*, подробнее см. подраздел 1.3.8).

– *Анализ данных раздела* (автоматический режим, предназначенный для восстановления NTFS разделов, подробнее см. раздел *Автоматические методы восстановления NTFS разделов)*;

– *Сканировать незанятое пространство* (автоматический режим, предназначенный для восстановления удаленных NTFS разделов, подробнее см. раздел *Автоматические методы восстановления NTFS разделов)*;

– *Сканировать MFT* (выполняется сканирование записей таблицы MFT и на ее основе строится виртуальная файловая система выбранного раздела, подробнее см. раздел *Сканирование MFT*).

– *Очистить таблицу результатов*. При работе проводника с разделом NTFS в базе данных задачи сохраняются некие метаданные, которые условно называются таблицей результатов. Если таблица результатов была заполнена исходя из неверных предпосылок, то данный метод позволяет, изменив предпосылки, очистить таблицу с метаданными и повторно ее заполнить, используя данный метод.

Например, в случае работы с разделом с неисправным транслятором, в таблицу результатов могут попасть неверные данные (куски таблицы MFT смещены относительно положения, которое они должны занимать, в результате не удается правильно рассчитать номера записей таблицы). Соответственно, после того, как найдены все сдвиги таблицы MFT (сдвиги для всех ее кусочков) мы должны получить доступ к данным с помощью проводника. Для этого необходимо выполнить очистку таблицы результатов от недостоверных данных (которые были помещены в нее на том этапе работы проводника, когда не были найдены все сдвиги) и выполнить сканирование MFT (в таблицу результатов будут помещены достоверные данные с учетом найденных сдвигов).

– *Общий список нерезидентных файлов*. Для разделов NTFS может быть построена таблица всех нерезидентных файлов, в которой все объекты упорядочены в порядке возрастания их начального LBA. Данная таблица, в первую очередь, может использоваться для поиска точек сдвига при восстановлении транслятора (подробнее см. подраздел **Ошибка! Источник ссылки не найден.**). Для исходных разделов накопителя и виртуальных разделов созданных вручную, таблица объектов строится по мере раскрытия дерева раздела. Для виртуальных разделов, полученных в результате автоматических методов восстановления, список нерезидентных файлов заполнен полностью, т.к. в процессе их работы выполняется сканирование таблицы MFT или поиск всех записей.

1.2.1.6 Каталоги и файлы

Для объекта *каталог*, доступны следующие методы:

– *Сканировать;*
– *Просмотр первого сектора;*
– *Карта;*
– *Сохранить …(F2)* ;
– *Сохранить отмеченные … (Ctrl+F2)* ;
– *Найти файлы (Ctrl+F)*;
– *Карта каталога;*
– *Карта отмеченных каталогов и файлов;*
– *Отчет по отмеченным каталогам и файлам;*
– *Свойства элемента MFT* (только для NTFS разделов, в протокол выводятся краткая сводка элемента таблицы MFT соответствующая выделенному объекту, а полное описание элемента выводится с помощью специализированного *Редактора записи MFT* , подробнее см. подраздел 1.3.8).

Сохранить – данный метод позволяет сохранить выбранные объекты (файлы и/или каталоги с содержимым) в папку, указанную пользователем. Метод работает с файлами и папками, выделенными на дереве списка объектов или в таблице дочерних объектов (при этом для дерева списка не действует множественное выделение в отличие от таблицы дочерних объектов). Выделение объектов производится по аналогии со стандартным проводником Windows, за исключением выделения от текущего объекта до первого и/или последнего (соответственно, *Ctrl+Shift+Home* и *Ctrl+Shift+End*). Для любого каталога использование данного метода приводит к копированию всего содержимого этого каталога.

Например, на рисунке ниже приводится таблица дочерних объектов с тремя выделенными EXE файлами. При этом, в таблице "отмечены" другие три файла и каталог, но на действия метода *Сохранить* это не окажет никакого влияния.

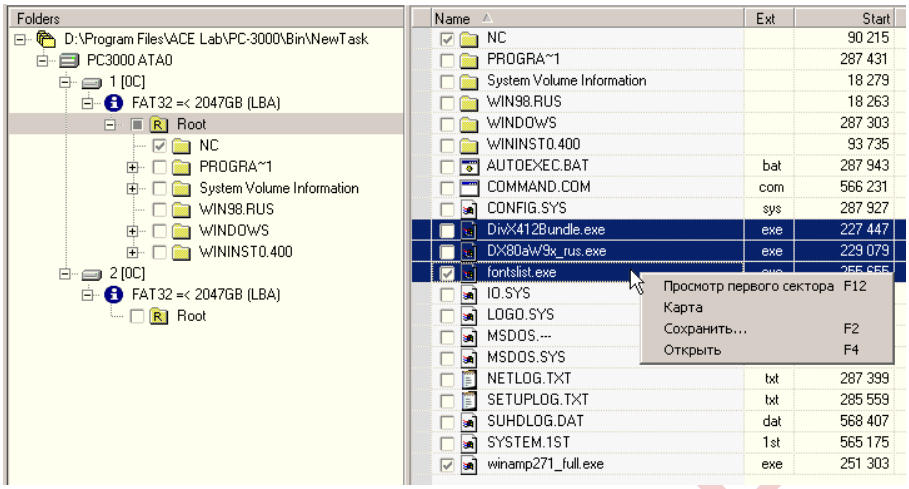


Рисунок 21 Множественное выделение в таблице дочерних объектов

Сохранить отмеченные – данный метод позволяет сохранить отмеченные в проводнике (находящиеся ниже по иерархии относительно объекта, для которого вызывается метод) файлы и папки на штатный накопитель.

Например, на рисунке ниже приводится дерево списка объектов с выделенным каталогом "Windows". В выделенном каталоге есть отмеченные файлы и папки (часть из них видна на рисунке, и об этом говорит значок слева от папки). При выборе метода *Сохранить*, будет выполнена попытка скопировать всю папку "Windows". При выборе метода *Сохранить отмеченные*, будут скопированы на штатный накопитель отмеченные объекты находящиеся в выбранной папке (при этом отмеченная папка "NC" не будет скопирована т.к. находится на одном уровне с папкой "Windows").

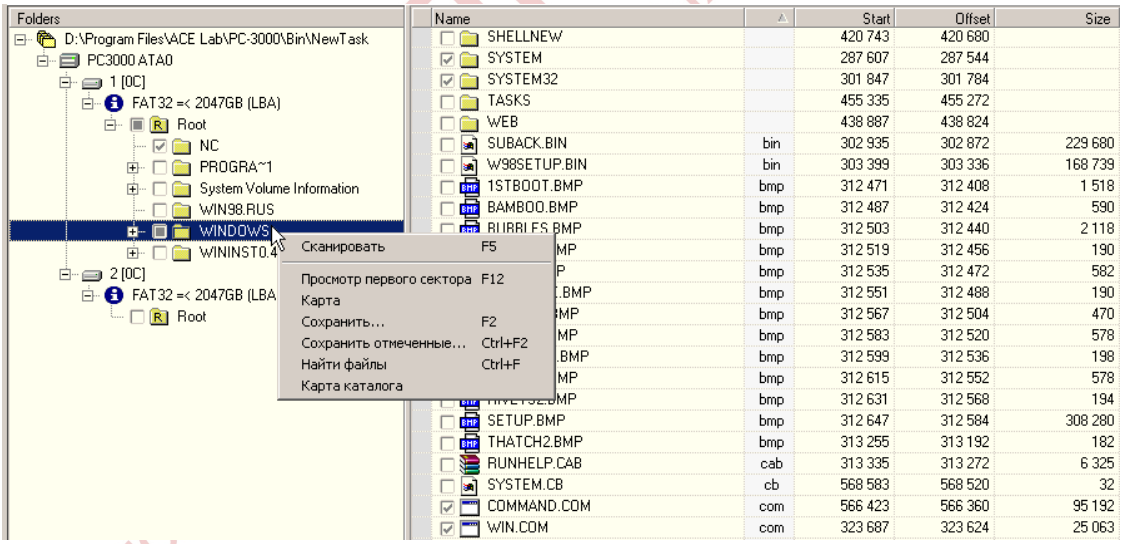


Рисунок 22 Сохранение отмеченных объектов на штатный накопитель

С помощью данного метода можно отметить требуемые для копирования данные, перейти на объект *root* и сохранить все отмеченные данные в папку задачи.

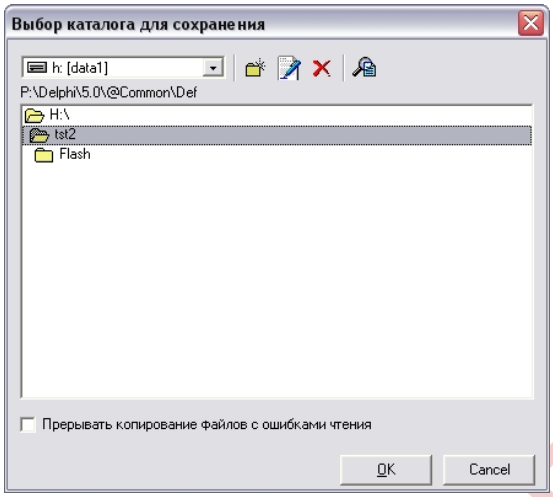


Рисунок 23 Окно выбора каталога для сохранения данных пользователя

Найти файлы – с помощью данного метода осуществляется поиск файлов и папок удовлетворяющих условиям заданных пользователем. Результатом каждого поиска является экземпляр проводника, содержащий найденные файлы/папки (при этом сохраняется иерархия данных, т.е. найденные файлы находятся в своих родительских каталогах). Соответственно, можно осуществлять поиск рекурсивно, например, сначала найти каталоги отвечающие требуемому условию, а затем запустить поиск файлов по найденному.

Следует иметь ввиду, что поиск осуществляется для всего родительского каталога на дереве объектов.

На рисунке ниже приводятся результаты поиска в корневом каталоге всех файлов с расширением "exe".

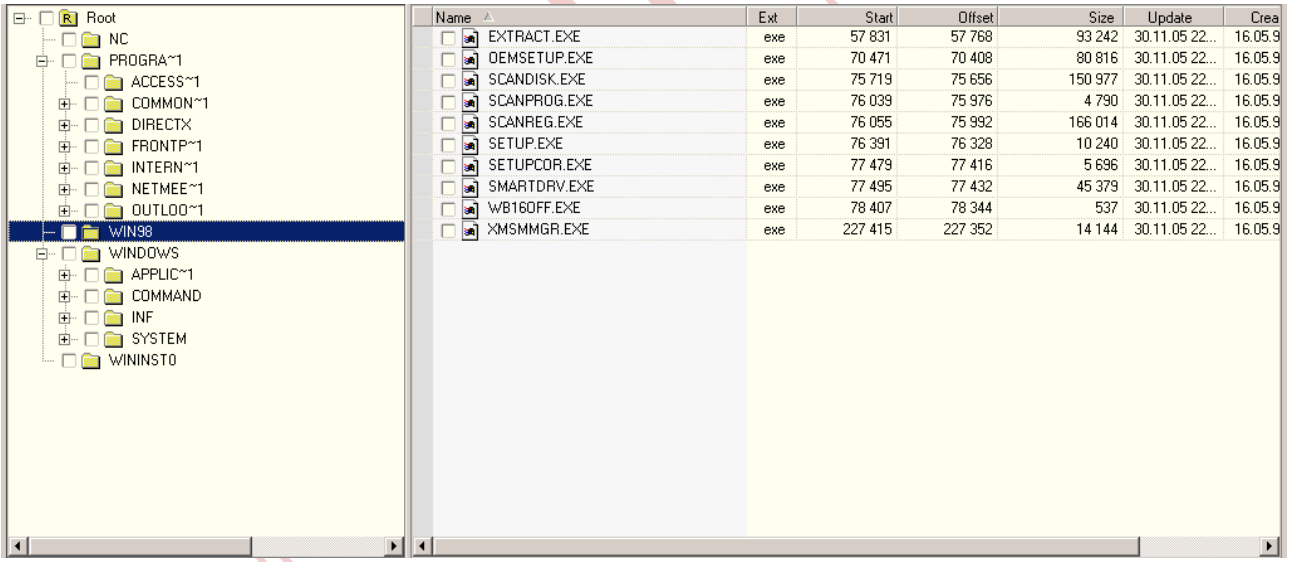


Рисунок 24 Результаты поиска "*.exe" в каталоге "root"

Для осуществления поиска, пользователю необходимо определить критерии поиска.

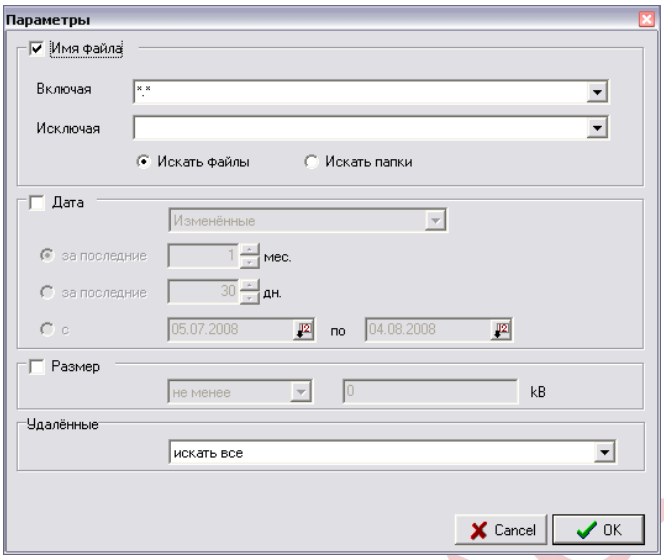


Рисунок 25 Определение критериев поиска файлов

Пользователю предоставляется возможность поиска *по имени*, *дате* и *размеру*. При поиске по имени разделяется поиск файлов и папок. Поиск по дате и размеру аналогичен общепринятым и не требует пояснений. При поиске по имени, пользователь может определить одновременно оба критерия, в этом случае в результирующий список будут выводиться файлы, удовлетворяющие одному из условий группы *Включая* и не удовлетворяющие ни одному из группы *Исключая*. Синтаксис написания фильтров аналогичен синтаксису поисковых режимов ОС Windows.

С помощью выпадающего списка *Удаленные,* возможно задать для поиска следующие варианты: *искать все*, *искать только не удаленные* и *искать только удаленные*.

Карта – данный режим позволяет визуально просмотреть карту размещения выбранного объекта (каталога или файла). Для каталога отображается список цепочек занимаемых данным каталогом (сканирование вложенных объектов не осуществляется).

Карта не строится для искусственно созданных папок и объектов размещенных резидентно (NTFS).

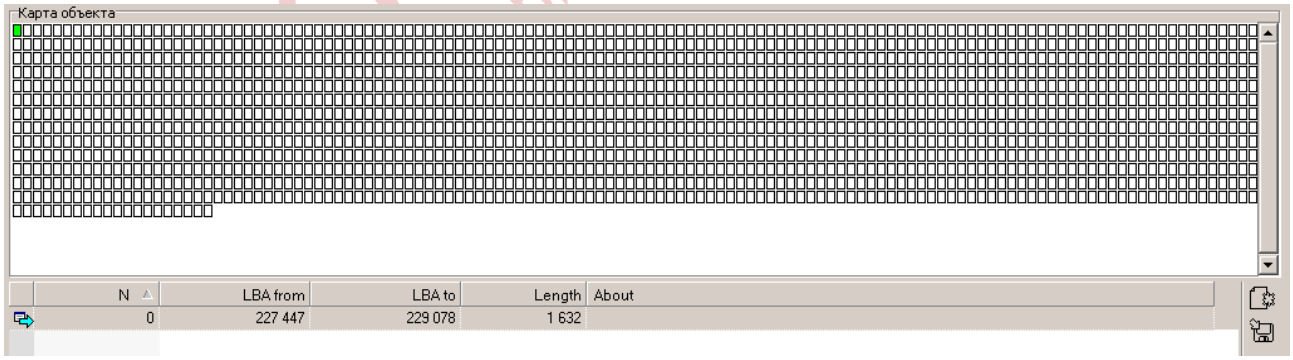


Рисунок 26 Режим "Карта"

Карта каталога – в результате выполнения данного режима строится суммарная карта всех объектов каталога без метаданных в случае NTFS разделов. Режим недоступен для объекта проводника *root*, т.к. по сути, карта для root-каталога это карта занятых секторов.

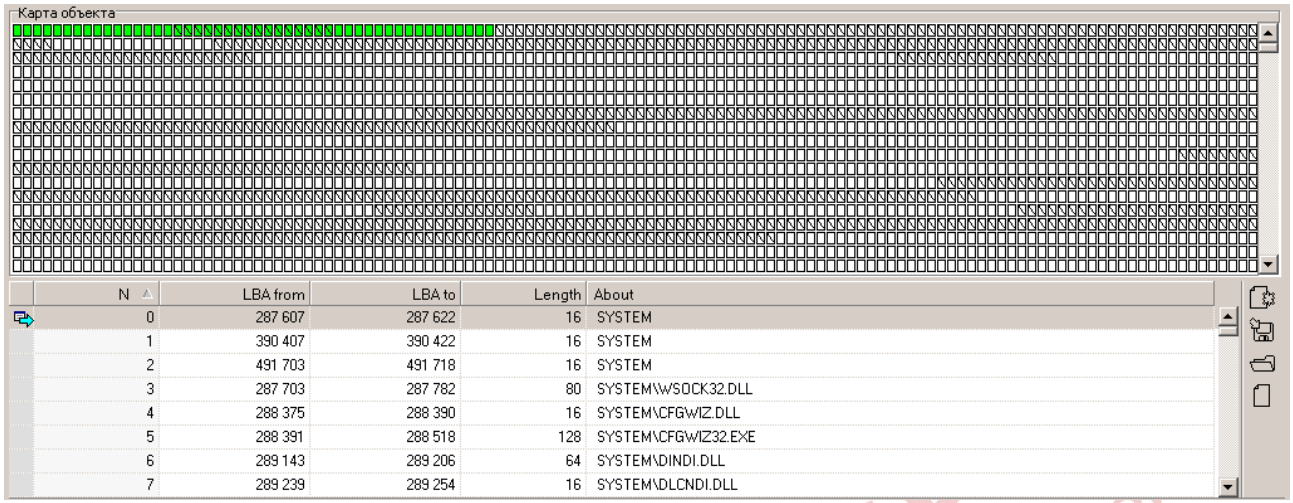


Рисунок 27 Режим "Карта каталога"

Более подробно о работе с картой см. в разделе *Карта объекта*.

Карта отмеченных каталогов и файлов – данный режим строит карту отмеченных каталогов и файлов с учетом иерархии объекта, для которого данный метод вызван. Например, если в корневом каталоге отмечены два каталога – *Downloads* и *MyDocuments*, и метод вызван из контекстного меню *root*-директории, то в карту попадут оба каталога. Если метод вызван из контекстного меню одной из отмеченных папок – в карту попадет соответствующий каталог. Если метод вызван из контекстного меню любого другого каталога или файла – в карту не попадет ни один объект.

Полученная карта может быть использована для выборочного копирования секторов, относящихся к требуемым каталогам и файлам.

Отчет по отмеченным каталогам и файлам – с помощью данного режима в стандартный текстовый редактор (*Блокнот*) выводится список тех отмеченных объектов, которые содержат непрочитанные, измененные или прочитанные с ошибкой сектора.

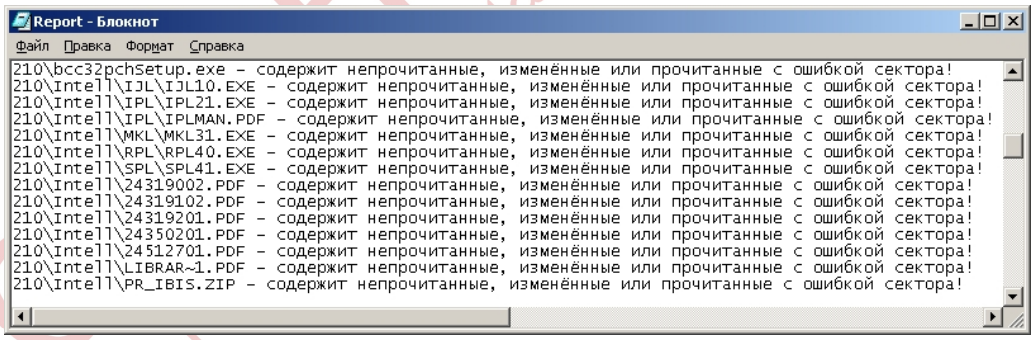


Рисунок 28 Отчет по отмеченным каталогам и файлам

В контекстном меню для файлов, отсутствует метод *Карта каталога* и добавлен метод *Открыть (F4)*.

Использование при восстановлении логической структуры данных исправного накопителя.

Использование режима для выполнения восстановления логической структуры данных основано на возможности корректировать данные на исследуемом диске и тут же проверять полученные результаты, используя режимы проводника.

К сожалению, все случаи применения описать трудно, поэтому приведем два примера:

Пример 1. Предположим, что мы имеем накопитель с двумя разделами, первый из которых FAT32 необходимо восстановить, и у него разрушены boot и boot-copy (слоты MBR целы). Существует два варианта действий: редактировать boot-сектор непосредственно на накопителе или создать виртуальный раздел с требуемыми параметрами (тип, размер кластера и т.д.) Если Вы уверены в своих действиях и у накопителя нет проблем с записью, то можно воспользоваться первым вариантом.

Однако более просто и безопасно создать виртуальный раздел. В данной ситуации нам придется вручную отредактировать boot-сектор созданного виртуального раздела.

Замечание. В том случае, если бы копия boot-сектора не была разрушена, можно было бы поступить двумя способами. Найти копию загрузочного сектора с помощью поиска регулярных выражений, открыть сектор в двоичном редакторе в режиме *Просмотр как…→ Boot FAT32* и с помощью опции *Сервис → Добавить виртуальный раздел* создать требуемый виртуальный раздел. Или запустить режим *Быстрый анализ диска*, доступный из контекстного меню *MBR* (в данном случае объем вычитанных и проанализированных данных будет незначительно больше и в результаты анализа попадут все найденные разделы).

Предположим, что мы пошли по второму пути и создаем виртуальный раздел. В окне *Добавление виртуального раздела* необходимо указать тип раздела (обычно он известен и в данном случае FAT32), начальный и конечный LBA и размер кластера. Начальный и конечный LBA берем из слота MBR. Значение размера кластера пока оставляем неизменным (значение по умолчанию 8) , его мы скорректируем позже.

После ввода всех параметров в окне *Добавление виртуального раздела*, нажимаем кнопку *Ok* и переходим к редактированию boot-сектора раздела. Нам необходимо корректно заполнить лишь значения значимые для восстановления информации, и эти значения, отмечены с помощью светлого фона. Опишем порядок заполнения этих значимых данных:

– *BytesPerSector* – количество байт в секторе, обычно 512 (200h).

– *SectorsPerCluster* – количество секторов в кластере, значение по умолчанию – 8. Вычислить значение данного поля можно следующим образом. Запустить режим *Поиск GREP* выбрав в качестве критерия поиска *FAT folder*. Прервать процесс можно в том случае, когда найдено несколько папок. Затем, с помощью найденных папок определить приблизительное положение root-директории и количество секторов в кластере. Для этого необходимо перейти в режим *Просмотр* первой из найденных папок, и в двоичном редакторе воспользоваться режимом *Просмотр как… → FAT Folder*. В появившемся окне *FAT Подкаталог* на панели *FAT – положение Root* вычислить ряд значений *RootSectors* для разных величин *SectorsPerCluster* (например, 4, 8, 16, 32). Повторить расчет для нескольких найденных каталогов. В результате, приблизительное положение root-директории будет определяться совпадающими (или мало отличающимися) значениями *RootSectors* при фиксированном *SectorsPerCluster*. Соответствующее значение *SectorsPerCluster* заносим в boot-сектор виртуального раздела.

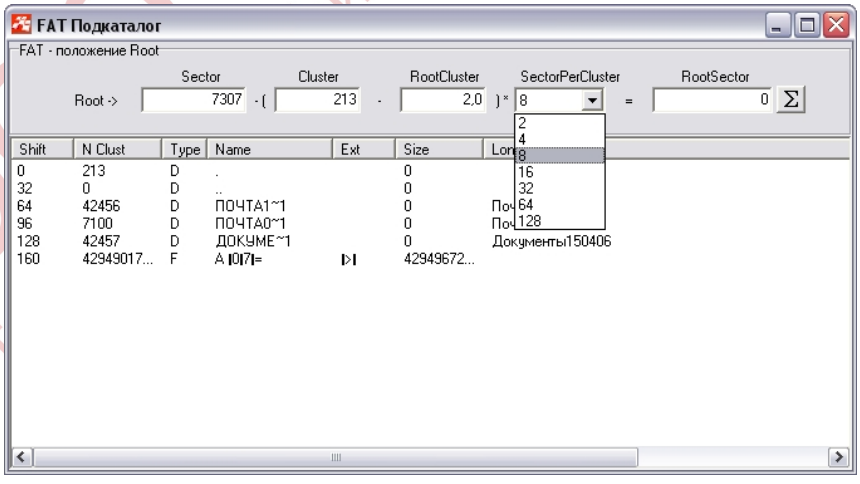


Рисунок 29 Окно "FAT подкаталог", вычисление положения root-директории и размера кластера

– ReservedSectors - количество секторов, занятых загрузчиком и зарезервированных, значение по умолчанию для FAT32 - 32. Для того чтобы проверить корректность данного параметра, необходимо перейти в режим Карта раздела объекта boot-сектор и с помощью таблицы цепочек, навигационной панели и панели HEX (двоичный редактор) проверить положение первой копии FAT (FAT начинается с дескриптора носителя, F8h – жесткий диск). Иногда возникает ситуация когда первая копия FAT уничтожена, но вторая осталась. Тогда, увеличивая значение параметра ReservedSectors так, чтобы он ссылался на вторую копию и меняя число копий FAT на одну, мы можем продолжить работу с

разделом. Точное положений копий FAT можно найти с помощью режима Поиск GREP выбрав в качестве критерия поиска FAT (Copy).

– BigTotalSectors - число секторов. Данный параметр вычисляется как разница между Конечным и Начальным LBA.

– BigSectorsPerFAT - число секторов в одной FAT. Данный параметр вычисляется следующим образом. При расчете параметра SectorsPerCluster мы вычислили приблизительное положение root-директории, и теперь необходимо его уточнить. Для этого необходимо перейти на найденный сектор с помощью двоичного редактора и проверить является ли он root-директорией. Проверка осуществляется следующим образом, во первых, сектор должен содержать элементы каталога (Просмотр как… → FAT Folder), во вторых, каталог не должен содержать ссылку на родительский каталог, и в третьих, обычно, если сдвинуться на один сектор влево, то он будет содержать нули (это конец FAT, и обычно он не заполнен). Уточнив таким образом положение root-директории, из него вычитаем начальный LBA и число зарезервированных секторов, в результате получаем число секторов занимаемых FAT. Если копия FAT одна, это и есть искомое значение, если же копий FAT две (что обычно и бывает), то полученное число мы делим пополам.

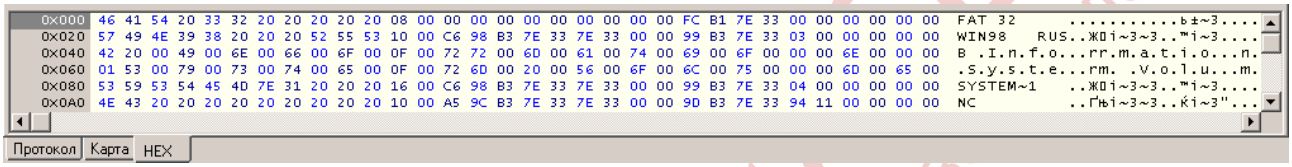


Рисунок 30 Окно редактора двоичных данных с первым сектором root-директории раздела FAT32

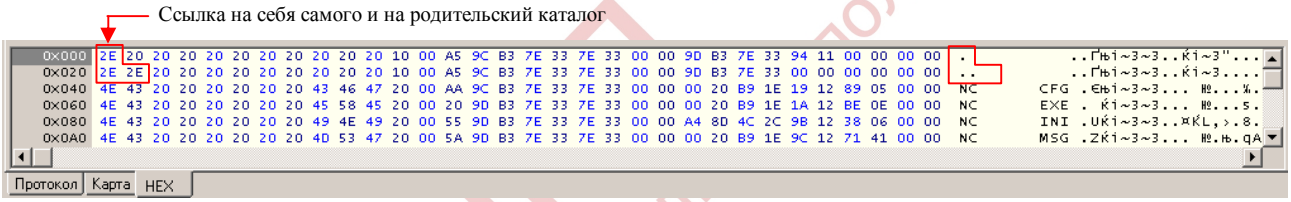


Рисунок 31 Окно редактора двоичных данных с первым сектором произвольной директории раздела FAT32

В результате корректно произведенного заполнения boot-сектора виртуального раздела, мы получим доступ к требуемым данным первого FAT32 раздела. При необходимости созданный загрузочный сектор можно сохранить на накопителе и работать с ним в дальнейшем штатно (это не относится к случаю, когда на накопитель нельзя производить запись).

Пример 2. На восстанавливаемом накопителе существуют два раздела (FAT или NTFS не важно), но разрушен сектор MBR. Сектор *MBR* можно восстановить в ручном режиме указав тип раздела и его относительное местоположение на накопителе из найденных с помощью *поиска GREEP* загрузочных секторов. Кроме этого, существует автоматический режим *Быстрый анализ диска*.

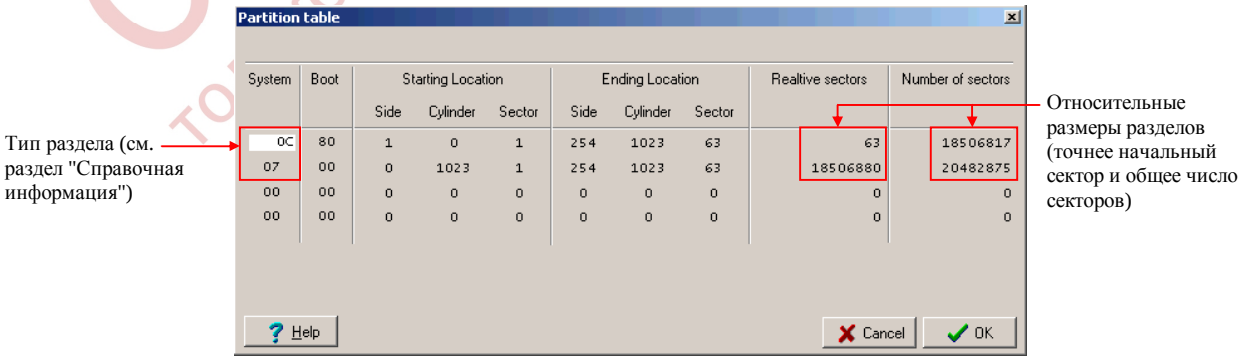


Рисунок 32 Окно описателей накопителя (MBR) вызываемое методом "Свойства"

Методом *Быстрый анализ диска* доступен из контекстного меню сектора *MBR*. В результате его работы будут созданы виртуальные слоты *MBR* на основании всех найденных загрузочных секторов. Воспользовавшись данным автоматическим режимом, мы не восстановим сектор MBR в

дословном понимании, хотя доступ к данным разделов будет получен с помощью виртуальных слотов MBR.

Рассмотрим ручной способ восстановления слотов MBR. Переходим в режим *Поиск GREP*, в качестве критериев поиска выбираем загрузочные сектора разделов FAT и NTFS и запускаем поиск. В результате поиска будет найдено несколько загрузочных секторов (т.е. искомые загрузочные сектора, возможно, их копии и ошибочно идентифицированные) из которых необходимо выбрать требуемые. Проверить корректность загрузочного сектора можно с помощью двоичного редактора и режима *Просмотр как…→ Boot XXX*. Отобрав таким образом требуемые два загрузочных сектора, необходимо стать в проводнике на сектор MBR и с помощью контекстного меню вызвать для него метод *Свойства*. В появившемся окне остается отредактировать тип разделов и их относительные координаты в соответствии с найденными загрузочными секторами.

1.3 Дополнительные режимы работы

1.3.1 Режим "Карта объекта"

Когда речь идет о карте объекта, пользователь должен понимать, что каждый объект логической структуры диска размещен на диске и имеет некую карту размещения – последовательность секторов.

Данный режим предназначен для визуальной работы с этой картой. Он облегчает понимание того, каким образом объект размещен на диске, как он фрагментирован и, в случае выполнения задачи копирования данных с поврежденного накопителя, как прошло копирование (какие сектора не прочитались, какие прочитались с ошибкой и т.д.).

Внешний вид режима *Карта объекта* представлен на рисунке ниже.

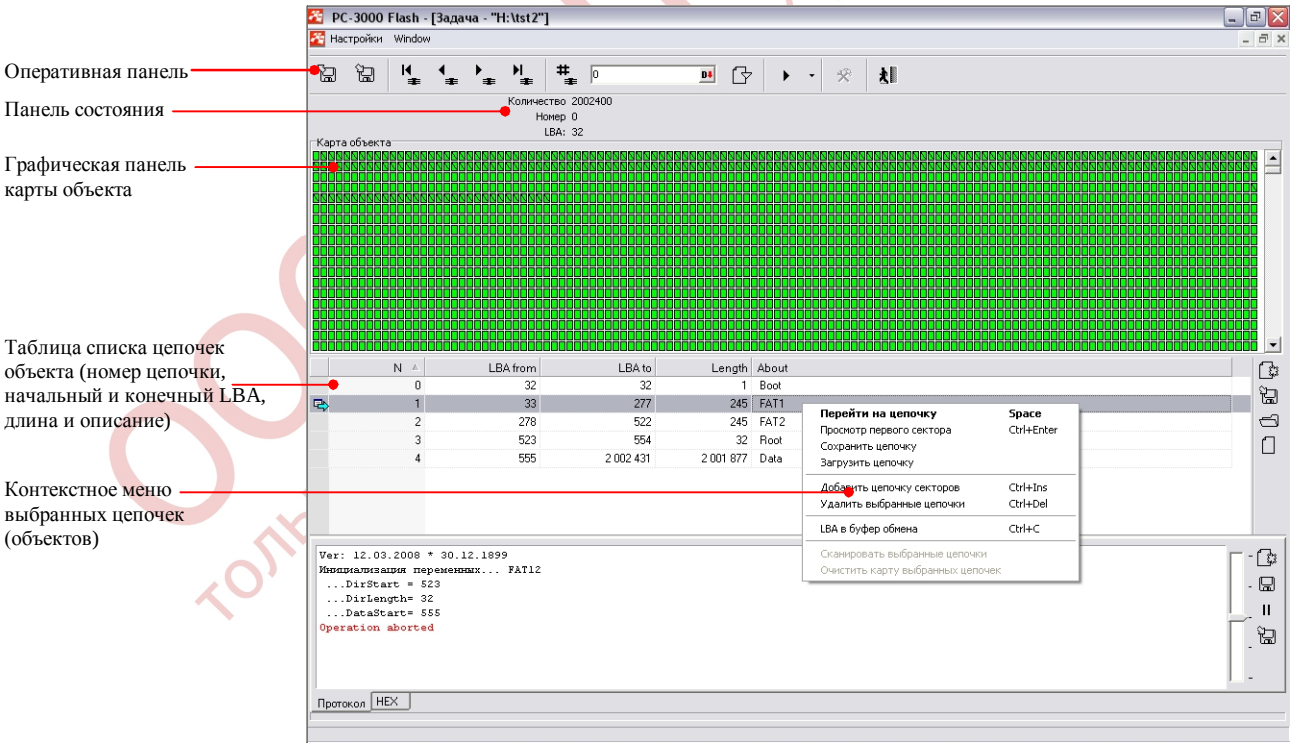


Рисунок 33 Внешний вид режима просмотра карты объекта

В верхней части окна расположены оперативная панель навигации для перемещения по карте и информационные строки, отображающие текущее состояние. Назначение клавиш оперативной панели приводится ниже.

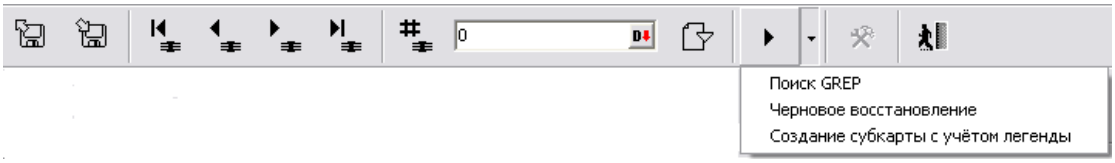


Рисунок 34 Оперативная панель режима "Карта объекта"

– Загрузить/Сохранить из/в файл (следует иметь ввиду, что данные методы работают со всем объектом, т.е. если Вы перешли в режим карта для какого-либо раздела, то загружаться/сохраняться будет весь раздел);

– Управление процессом позиционирования (*Первый*, *Предыдущий*, *Следующий*, *Последний*);

– *Перейти к сектору*, окно номера сектора на который будет осуществляться переход и *Редактировать*;

– *Поиск GREEP*;

– *Черновое восстановление*;

– *Создание карты с учетом легенды*;

– Параметры задачи;

– Выйти из задачи.

Создание карты с учетом легенды. На основании текущей карты можно построить субкарту которая будет содержать только сектора с заданными признаками. Например, имея карту каталога с частично вычитанными данными, можно с помощью данного метода построить субкарту содержащую только не прочитанные сектора с информацией о принадлежности к конкретным файлам.

Черновое восстановление и *Поиск по GREP*. Данные режимы работы доступны для всех объектов загружаемых в карту. Причем, данные режимы позволяют работать как с непрерывными объектами (например, раздел), так и цепочками секторов (карта незанятого пространства). При черновом восстановлении по карте производится корректировка размеров найденных объектов по области сканирования. Подробнее о данных режимах см. разделы *Черновое восстановление* и *Поиск по GREP*.

Когда речь идет о перемещении, необходимо помнить о двух аспектах: объект имеет относительные и абсолютные (с учетом реального размещения на диске) координаты. Если относительные координаты непрерывны и линейны (например, от 0 до 999 сектора при длине 1000 секторов), то абсолютные (LBA) напрямую зависят от степени фрагментации диска и могут быть какими угодно. Когда речь идет о перемещении, то имеется в виду относительное перемещение.

В средней части окна расположена собственно карта объекта. Если навести указатель мыши на квадратик сектора, то появится всплывающая подсказка с относительными и абсолютными координатами. Для того чтобы отличить цепочки секторов, применяется перемежающееся (или чередующееся) перечеркивание.

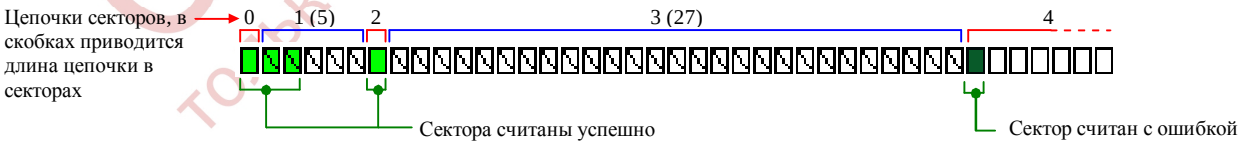


Рисунок 35 Отображение цепочек и считанных секторов

Текущее состояние сектора отображается на карте с помощью цвета. Легенду цветов можно посмотреть с помощью кнопки *Легенда* на вкладке *Карта*. В том случае, когда копия данных не создается, все сектора на карте отображаются как прочитанные, однако они могли и не читаться (предполагается, что если копия данных не создается, то у тестируемого накопителя нет проблем с чтением, и любой сектор может быть прочитан в нужный момент).

Под картой расположена таблица списка цепочек. По умолчанию, таблица отсортирована по номеру цепочки (о чем говорит значок "Δ" в заголовке и цвет столбца), однако пользователь может изменить и столбец, по которому производиться сортировка, и порядок (возрастание "Δ" или убывание "∇") с помощью щелчка левой клавиши мыши на соответствующем заголовке таблицы.

Способ сортировки, помимо удобства восприятия данных, влияет на очередность сканирования (и сохранения).

Все значения таблицы, кроме столбца *N*, можно редактировать (щелчок левой клавиши мыши на соответствующем значении).

Замечание! В случае изменения значений таблицы цепочек проверка корректности введенных данных не производится (т.е. возможно пересечение цепочек).

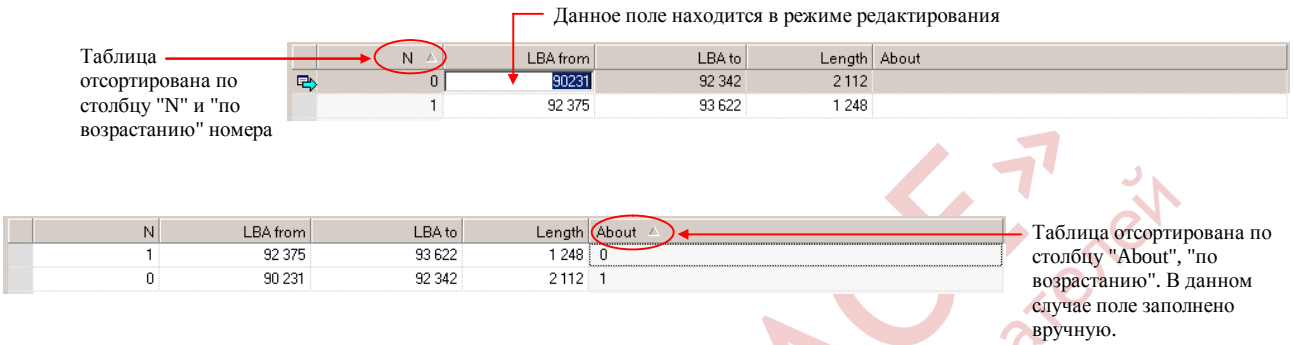


Рисунок 36 Таблица цепочек

В случае, когда просматривается карта объекта, цепочки которого имеют смысловое значение, заполнено поле *About* (например, Boot, FAT1, FAT2, Data и др.). Если же говорить о цепочках, не имеющих смысловых значений (например, цепочки файла), то редактирование поля *About* можно использовать для создания произвольного упорядочивания и затем сканирования или сохранения этих цепочек в выбранном порядке.

С помощью контекстного меню таблицы цепочек доступна возможность добавления и удаления цепочек, что позволяет создавать и использовать вручную созданную карту объекта (пункты *Добавить цепочку секторов* и *Удалить выбранные цепочки*). Это можно использовать при "ручной сборке" файла (начало, и возможно конец цепочки ищется с помощью поисковой системы).

Возможна индивидуальная работа с каждой цепочкой (можно перейти на ее начало, 2 раза щелкнув на ней левой клавишей мыши), сохранить ее на диск в виде файла или загрузить из файла. Методы доступны через контекстное меню списка. Помимо индивидуальной работы с цепочками доступно выделение и работа с набором цепочек (выделение нескольких цепочек осуществляется с помощью щелчка левой клавиши мыши и нажатой клавиши *Ctrl*).

Можно очистить выбранные цепочки секторов (т.е. заполнить их нулями) и карту (т.е. в случае создания копии, для выбранных цепочек устанавливается признак о том, что они не были прочитаны).

В случае выбора пункта контекстного меню, который может привести к потере данных, выводится предупреждающее сообщение, например, на рисунке ниже показано окно сообщения при попытке очистить выбранные цепочки.

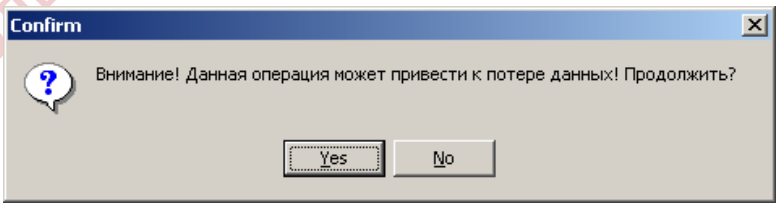


Рисунок 37 Окно сообщения при попытке очистить выбранные цепочки

В правой части таблицы списка цепочек расположены "горячие клавиши" для работы со списком. Данные кнопки позволяют:

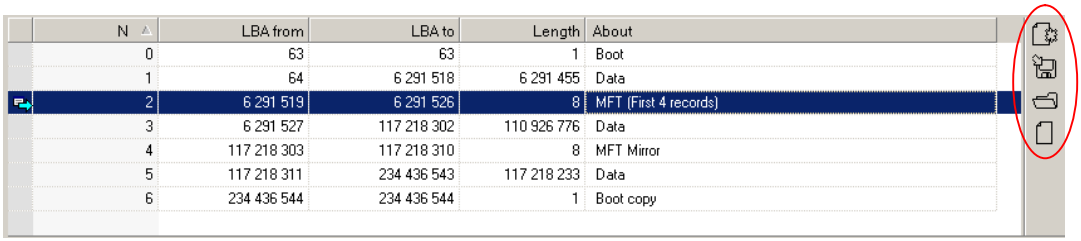
– *Очистить карту*. В результате будет полностью очищен список цепочек, соответственно исчезнет и их размещение.

– *Сохранить карту в файл*. В результате выполнения данного метода информация о списке цепочек построенной карты будет сохранена в файл. Этим методом можно воспользоваться, например, в том случае, если строится карта большого объекта и время затрачиваемое на ее построение достаточно длительное. Тогда

можно один раз построить эту карту, сохранить ее в файл, а затем не строить ее заново, а загружать из этого файла, что существенно быстрее.

– *Загрузить карту из файла*;

– *Добавить цепочку секторов*. Данный метод позволяет в ручном режиме создавать любую карту путем добавления требуемых цепочек секторов (возможно с учетом требуемого порядка их вычитывания).



N	LBA from	LBA to	Length	About
0	63	63	1	Boot
1	64	6 291 518	6 291 455	Data
2	6 291 519	6 291 526	8	MFT (First 4 records)
3	6 291 527	117 218 302	110 926 776	Data
4	117 218 303	117 218 310	8	MFT Mirror
5	117 218 311	234 436 543	117 218 233	Data
6	234 436 544	234 436 544	1	Boot copy

Рисунок 38 "Горячие клавиши" для работы со списком

Карта объекта очень удобна для задач выборочного копирования наиболее важных данных. В первую очередь это относится к объекту проводника boot-сектор раздела, для которого в режиме карта раскрывается структура. Например, для разделов FAT из данного режима можно произвести вычитывание копий FAT, а для NTFS разделов – таблицы MFT (первые 4 ее записи). В случае FAT разделов, сохраненную в файл копию FAT можно подключить для работы с требуемым разделом (подробнее см. режим *Проводник*).

В нижней части окна размещается панель с закладками протокола и просмотра сектора. Если активна закладка просмотра сектора – при переходе на выбранный сектор карты на закладке отображается содержимое данного сектора.

В случае, когда восстанавливается транслятор, карта раздела и использование закладки *HEX* позволяет установить на место важные с точки зрения дальнейшего восстановления цепочки данных (начало таблицы MFT и т.п.).

1.3.2 Режим поиска регулярных выражений

Данный режим является основным поисковым режимом программы. Он позволяет выполнять множественный поиск секторов, содержащих данные, удовлетворяющие условиям выбранных в качестве критериев поиска регулярных выражений (из соответствующего справочника).

Внешний вид режима приведен на рисунке ниже.

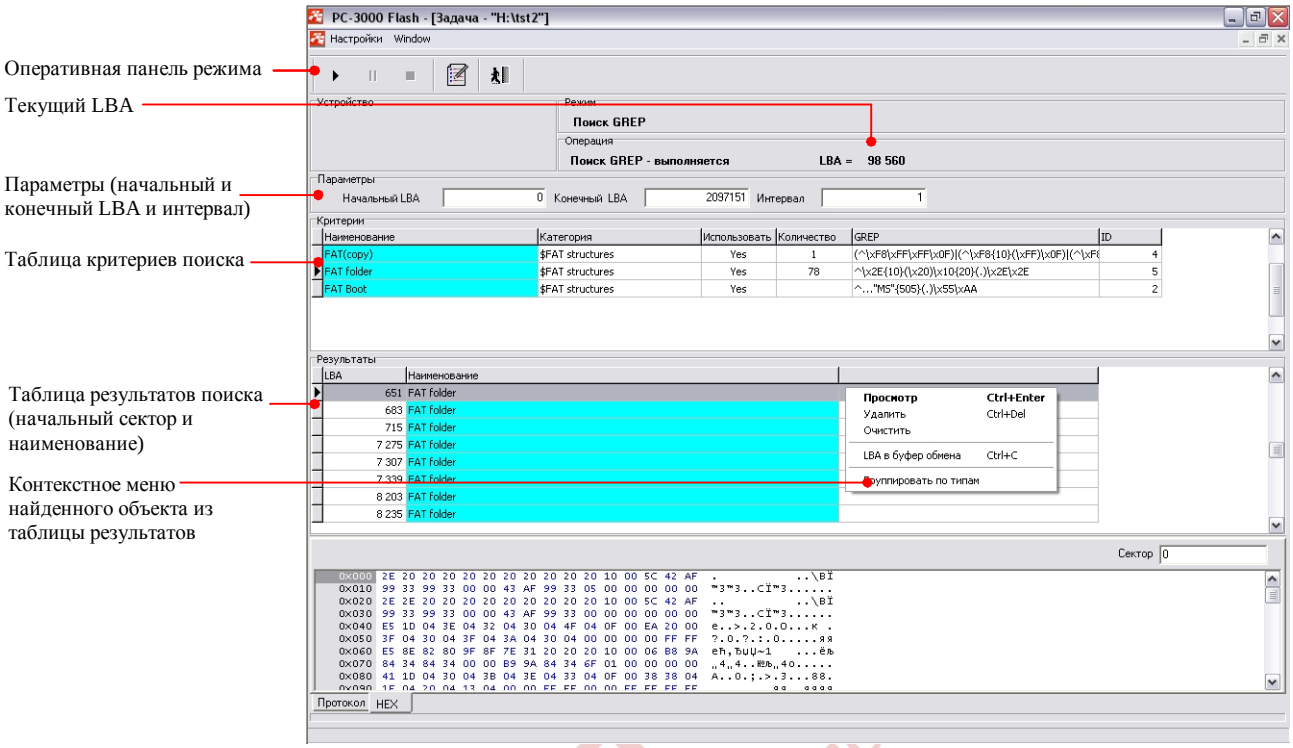


Рисунок 39 Внешний вид режима "Поиск GREP"

Оперативная панель режима проста и содержит всего пять кнопок – *Выполнить*, *Пауза*, *Прервать*, *Редактор сектора* и *Выйти*.

Для того чтобы выполнять поиск, необходимо определить диапазон поиска (не всегда), шаг поиска (если Вы ищете выровненные по началу кластеров данные – шаг должен быть размером с кластер, это позволяет ускорить процесс поиска), выбрать критерии поиска из справочника регулярных выражений и определить их используемость в запускаемом процессе поиска. Диапазон поиска не указывается в том случае, если он запускается для какой-либо карты проводника (в этом случае он определяется объектом карты). Величина интервала поиска должна (в идеале) совпадать с размером кластера, и если мы его знаем, его необходимо установить для сокращения времени и ошибок поиска. Если же размер кластера нам не известен, то остается единственный вариант – интервал равный единице.

Для того чтобы выбрать критерии, воспользуйтесь пунктом контекстного меню *Добавить* списка критериев. При этом на экране появится модальная форма списка регулярных выражений, работая с которой и пользуясь ее методом *Выбрать* (Alt+Enter), можно быстро определить список критериев.

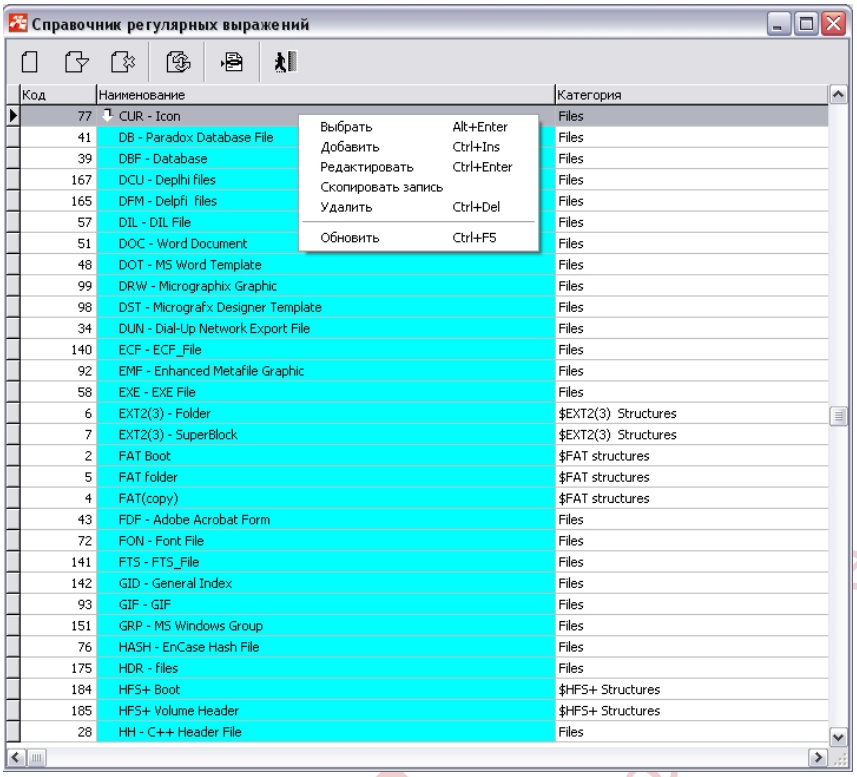


Рисунок 40 Модальная форма "Справочника регулярных выражений"

Указать, использовать или не использовать выбранный критерий в запускаемом процессе поиска, можно с помощью контекстного меню или двойного щелчка мыши по соответствующему критерию. Конечно, можно искать всегда все отобранные критерии, однако с целью экономии времени, желательно для конкретного диапазона оставлять активными только нужные критерии. Чем больше активных критериев и сложнее они, тем медленнее выполняется поиск (особенно для сложных, не имеющих прямое указание на положение искомых данных критерии).

Используя контекстное меню списка результатов можно настроить группировку списка по типам (критериям). Можно перейти к просмотру (редактированию) сектора.

1.3.3 Режим "Чернового восстановления"

Метод *чернового восстановления* является последним по применимости в ряду средств восстановления информации пользователя предоставляемых комплексом. Основная идея данного метода – восстановление файлов с известными заголовками при отсутствии информации об их размещении из предположения их непрерывности.

Иначе говоря, если найдены два заголовка файлов известного типа, то с большой долей вероятности можно предположить (если нет точной информации о расположении из таблицы FAT, записи MFT и т.п.), что файл имеет тип, определяемый первым заголовком и размер, определяемый разницей LBA первого и второго.

Это предположение не всегда верно, так как может быть ошибка в идентификации, файл может быть фрагментирован и т.п. Кроме того, в этом случае теряется информация об имени, дате создания, положении (каталоге).

Но, когда нет другого выхода, лучше восстановить хоть что-то, чем ничего. При этом нужно иметь в виду, что конкретный результат восстановления существенно зависит от конкретной ситуации, есть ли в используемом справочнике заголовков соответствующий, насколько фрагментирован диск, велики ли размеры искомых файлов. В различных случаях процент успешно восстановленных файлов может колебаться от 0 до 70%.

Этот метод нужно применять в случаях, когда логические разрушения очень велики или когда речь идет о файловой системе, для которой у Вас нет средств логического восстановления.

Режим внешне очень похож на режим поиска регулярных выражений (рисунок ниже).

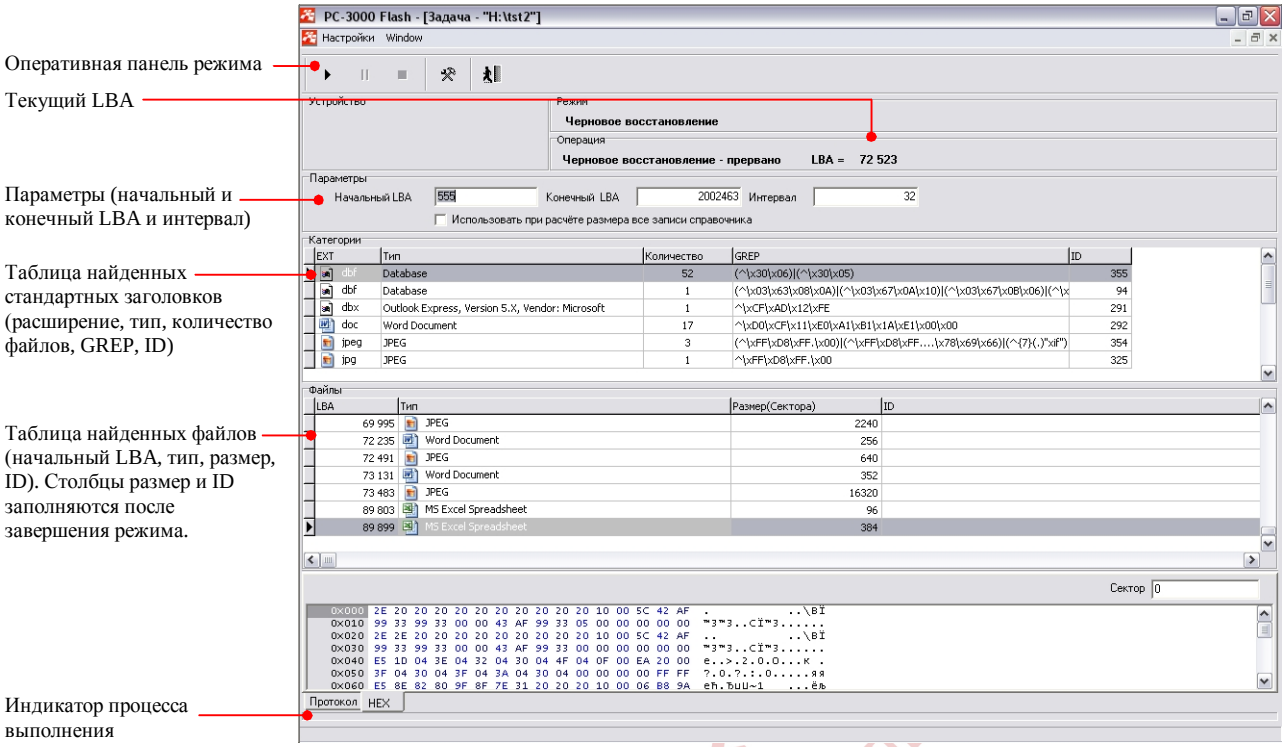


Рисунок 41 Режим чернового восстановления.

Основных отличий два:

– критерии поиска не выбираются, их список определяется *Справочником чернового восстановления* (при этом, критерии поиска могут быть изменены, подробнее см. ниже);

– результатом поиска являются предполагаемые файлы, которые можно скопировать. При этом имя файла - номер сектора начала, расширение определяется критерием. Если копируются все файлы определенного типа (контекстное меню списка критериев), создается подкаталог по имени расширения.

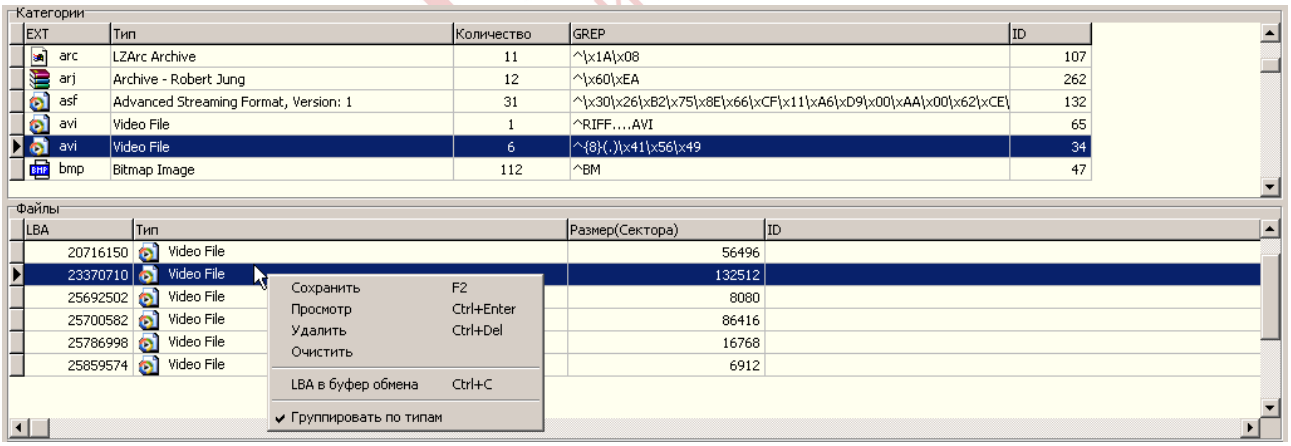


Рисунок 42 Результаты работы режима "чернового восстановления"

Справочник чернового восстановления. В данном справочнике собраны все регулярные выражения, которые могут быть использованы для идентификации файлов пользователя. Справочник чернового восстановления доступен из меню *Настройки* → *Справочник чернового восстановления* (см. рисунок ниже).

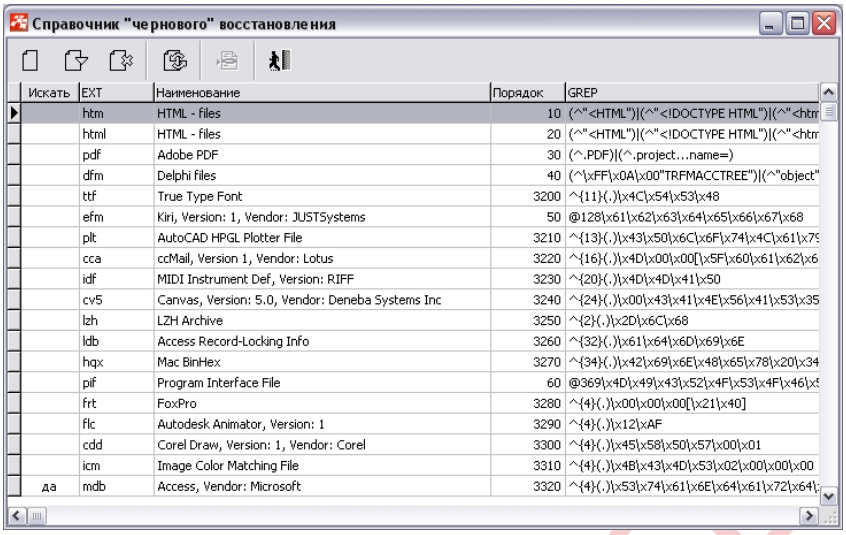


Рисунок 43 Справочник "чернового" восстановления

При выполнении “чернового восстановления”, для поиска предполагаемого начала файла используются только регулярные выражения отмеченные в столбце *Искать*. Для расширения списка используемых при поиске выражений необходимо либо пометить соответствующее существующее выражение из справочника, либо создать такое выражение и установить флажок в поле *Искать*.

При запуске "чернового" восстановления, пользователю предоставляется возможность выбора способа расчета размера найденных данных. Существует два варианта. В первом, при расчете размера используются только регулярные выражения отмеченные в столбце *Искать* справочника. Во втором, используются все регулярные выражения справочника (но только при расчете размера данных). Выбор конкретного способа расчета размера зависит от типа искомых данных, например, для архивов *ZIP*, предпочтительнее первый способ, т.к. если размер архива будет больше его реального размера, то ряд программ для восстановления данных из повреждённых архивов (например ZipFix) позволит извлечь из него данные. В случае если архив меньше своего реального размера вам вряд ли удастся извлечь из него даже часть данных.

Каждая строка справочника задает критерий поиска для файлов определенного типа с помощью поля *GREP*. В процессе чернового восстановления осуществляется поиск этих критериев. Данные поля *GREPEXT* используются для уточнения типа файла в случае наличия нескольких одинаковых GREP для разных типов файлов. Если *GREP* для нескольких типов файлов одинаковый и *GREPEXT* для них не существует, то расширение выбирается с помощью поля *порядок*, точнее по наименьшему порядку.

Дополнительные критерии, задаваемые полем *GREPEXT*, в справочнике не отображаются. Увидеть поле с дополнительными критериями, можно войдя в режим редактирования элемента списка.

Существует возможность редактирования данного справочника - это достаточно сложное и ответственное дело. Однако при создании задачи восстановления данных создается копия справочника "чернового" восстановления, с которой и происходит работа в текущей задаче. Соответственно, в случае необходимости справочник можно восстановить. С другой стороны, если Вы хотите, чтобы внесенные изменения были сохранены, то следует редактировать справочник, относящийся не к текущей задаче, а к программе PC-3000 Flash.

Основной справочник "чернового" восстановления PC-3000 Flash не используется при выполнении чернового восстановления. При создании задачи создаётся его копия. Редактировать основной справочник можно из меню главной формы. Справочник текущей задачи редактируется в режиме "чернового" восстановления.

Еще одной важной особенностью является то, что для данного режима ошибочная идентификация начала файла может привести к потере данных.

Это можно пояснить на примере - найдено много заголовков файлов определенного типа, имеющего простую и нехарактерную сигнатуру заголовка, а со слов клиента (или Вы сами убеждены) этого типа файлов не должно быть на исследуемом диске. Это означает, что найденные результаты – ошибочные, и они могут

привести к неверному расчету размеров других важных файлов, и их требуется удалить (при этом автоматически будут пересчитаны все размеры).

Правильное определение начала области сканирования и шага может существенно уменьшить количество ошибок и сэкономить время. Надо иметь в виду, что сканирование желательно начинать от известного начала кластеризации и с шагом, равным размеру кластера. Если эта информация отсутствует, то в качестве первого параметра подойдет любой, найденный при поиске с шагом 1, достоверный заголовок файла, а в качестве второго – минимальный размер кластера по умолчанию (например, для FAT - 8).

Возможные действия над элементами списков *Категории* и *Файлы* доступны через контекстные меню списков.

Режим "чернового" восстановления доступен через пункт меню *Сервис* → *Черновое восстановление*, либо через соответствующие пункты меню объектов проводника с той разницей, что для объектов проводника типа "слот таблицы разделов" установлены границы поиска (раздел), а для "Boot" еще и шаг (размер кластера).

1.3.4 Режим "Анализ данных раздела" (для ФС FAT)

Данный режим доступен через контекстное меню boot-сектора раздела FAT проводника.

Предназначен для поиска утерянных или недоступным по каким-либо причинам данных в режиме проводник FAT разделов. Для корректной работы данного метода необходим корректный boot-сектор раздела (начало кластеризации, размер кластера) или виртуальный boot-сектор.

В случае неисправных устройств (проблемные для чтения сектора) режим можно использовать в сочетании с опцией *создавать копию*.

Режим полностью автоматический, однако, на этапе определения параметров и целей сканирования требуется вмешательство пользователя.

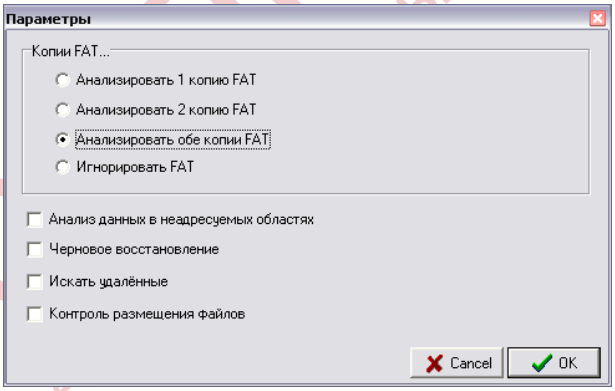


Рисунок 44 Определение параметров сканирования раздела

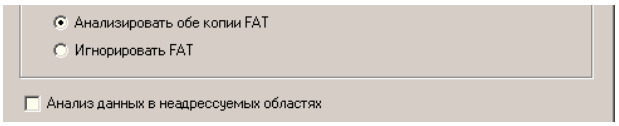
В зависимости от выбранных опций режима анализа объем вычитываемых данных меняется, но он всегда меньше полного объема раздела. Например, если выбрана опция использовать одну копию FAT, то вычитывается только соответствующая копия, сектора каталогов и начальные сектора файлов адресуемых выбранной копией FAT.

Копии FAT. Пользователю предоставляется возможность выбора между одной из копий FAT, совместного использования обеих копий или их игнорирование. Если для анализа выбрана одна из копий или обе копии FAT, то в базе данных задачи строится две таблицы – таблица объектов и таблица цепочек объектов. Первая содержит объекты, адресуемые FAT (т.е. папки и файлы), а вторая – цепочки секторов соответствующих объектов (содержит всю информацию о размещении конкретного объекта). Если анализируется одна из копий FAT, то данная копия вычитывается и таблицы заполняются на ее основе. Если для анализа используются обе копии, то они вычитываются и таблицы заполняются как пересечение копий (т.е. в таблицу попадают объекты одинаково адресуемые обеими копиями и объекты по-разному адресуемые копиями).

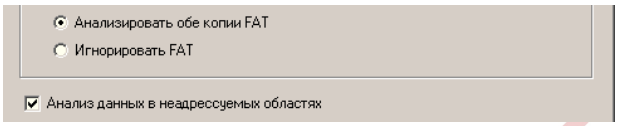
Игнорирование копий FAT в процессе восстановления данных подразумевает, что анализ будет производиться по всему разделу без учета копий (используется в случае, когда копии FAT содержат мусор).

Анализ данных в неадресуемых областях. Данный параметр, наряду с использованием/игнорированием копий FAT, определяет область анализируемых данных. Возможны несколько комбинаций.

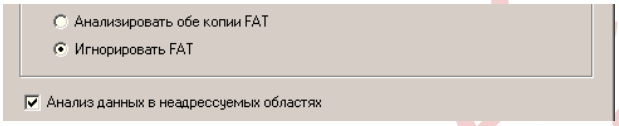
1) Анализируется область данных, адресуемая FAT (в данном случае подразумевается, что для анализа выбрана одна из копий FAT, или обе).



2) Анализируется вся область раздела, но при этом учитываются копии FAT



3) Анализируется вся область данных (Опция *Анализ данных в неадресуемых областях* включается автоматически если выбрать *Игнорировать FAT*), но при этом копии FAT не учитываются.



Выполнять "Черновое восстановление". Данная опция определяет, будет ли использоваться при анализе данных черновое восстановление. Черновое восстановление может использоваться как при анализе данных адресуемых FAT для предварительного уточнения типа данных, так и в обычном понимании, т.е. для восстановления непрерывных данных с помощью справочника регулярных выражений.

Искать удаленные файлы. Данный параметр определяет необходимость поиска удаленных файлов при сканировании по одной или обеим копиям FAT.

Контроль размещения удаленных файлов. При использовании данной опции осуществляется проверка пересечения удаленного файла и объектов адресуемых FAT (информация о начале удаленного файла берется из слота каталога, а размещение из предположения его непрерывности). Если происходит такое пересечение (и включен контроль), то размер удаленного файла уменьшается в соответствии с размещением более достоверного объекта.

Как говорилось выше, в ходе анализа формируются две таблицы – объектов и цепочек объектов. В этих таблицах для объектов вводится понятие достоверности. Например, если анализируются обе копии FAT, то объекты адресуемые одинаково обеими копиями наиболее достоверны, менее достоверны объекты адресуемые копиями по-разному. Что касается атрибутов файла (и в первую очередь это расширение), то информация, полученная в результате чернового восстановления, имеет низкую достоверность, а информация, полученная из слота каталога – более высокую. Т.е. если есть информация о типе файла на основании данных родительского каталога, то она является более достоверной по сравнению с расширением, полученным в результате поиска регулярных выражений. Однако, в том случае, если для файла не найден родительский каталог, то расширение полученное в результате чернового восстановления является хоть и мало достоверной, но единственной доступной. Разделяется достоверность каталогов адресуемых FAT и "потерянных" (например, из слота каталога адресуемого FAT имеем информацию о каком-то файле, при этом из слота "потерянного" каталога есть информация о том, что это другой файл с другими атрибутами, соответственно информация из адресуемого FAT каталога более достоверная и данные из "потерянного" каталога отбрасываем).

Замечание! Описанный принцип принятия решения действует для всех объектов восстанавливаемого раздела за исключением файлов с расширением ".chk". Данные файлы являются результатом работы программы проверки и ремонта структуры раздела "Check Disk" и обычно они расположены в каталогах типа "Folder.XXX" (других программных средств, создающих файлы с таким расширением нам не известно). Для файлов с таким расширением, намеренно снижена достоверность (т.е. если в процессе анализа будет найдена информация из любого другого каталога описывающая этот же объект, то она будет считаться более достоверной).

В качестве примера, на рисунке ниже приводится окно проводника с каталогом "Folder.000". В данном каталоге находится 139 объектов (см. строку статуса), при этом если запустить режим *Анализа данных раздела*,

то большинство файлов из этой папки будет перемещено в другие родительские каталоги (возможно и удаленные) с соответствующими новыми атрибутами.

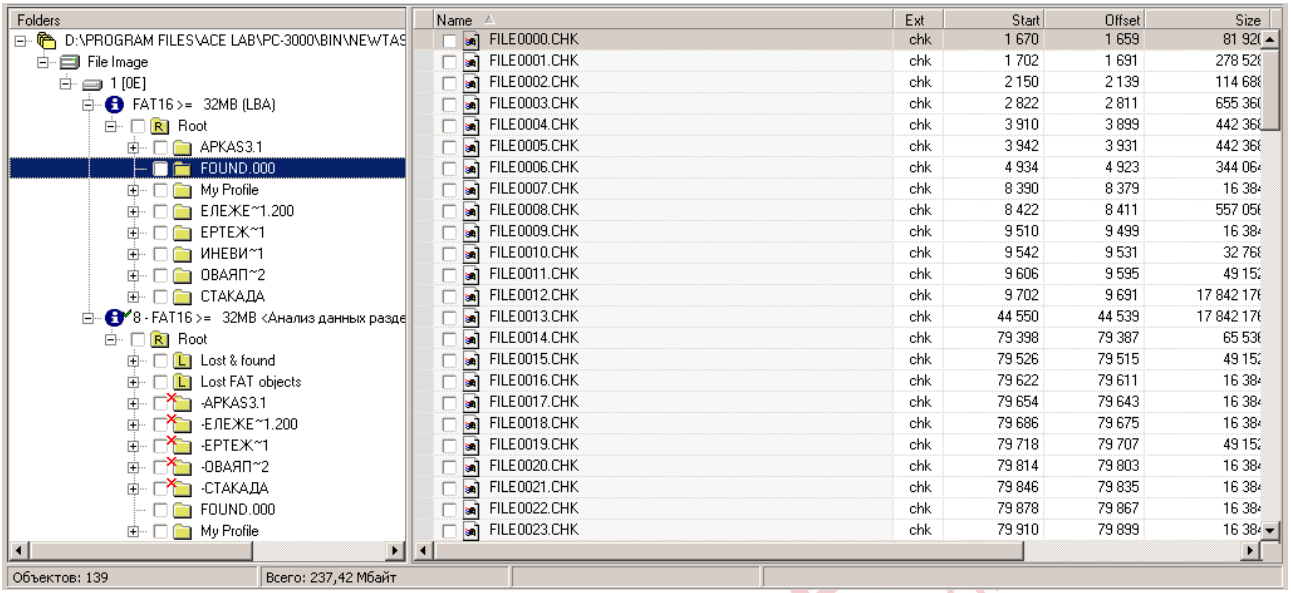


Рисунок 45 Использование режима "Анализа данных раздела"

Выбор параметров анализа в конкретной ситуации зависит от исходных данных, некоторые советы приводятся в разделе *Особенности использования режима* ниже по тексту.

После определения параметров выполняется сам режим, внешне похожий на режимы *Логического сканирования* и *Сканирование MFT* (форма с протоколом, картой и кнопками *Стоп* и *Выход*).

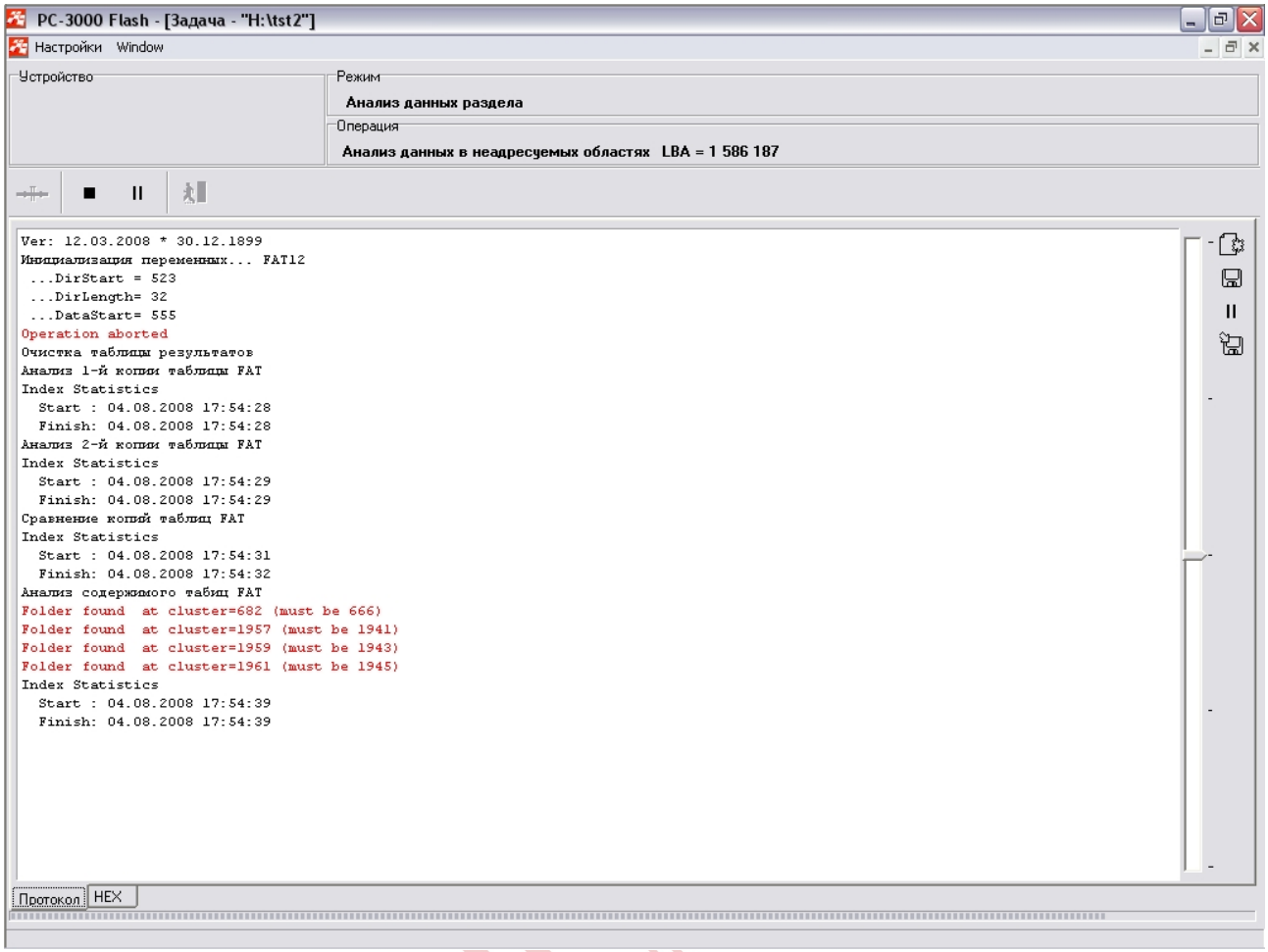


Рисунок 46 Режим "Анализа данных раздела"

В том случае, если для папки утеряна информация о родительском каталоге, то в виртуальной модели раздела создается искусственная папка *Lost&Found* в которую помещается этот объект и те объекты, для которых он является родительским.

Если существует информация о каком-то объекте FAT, т.е. в одной из копий FAT или в обеих есть связанный набор цепочек, на который нет ни одной ссылки из всех найденных каталогов, то можно предположить, что данный набор цепочек представляет собой файл (или обрывок файла). Такие объекты помещаются в искусственно созданную папку *Lost FAT objects*, если при этом выполняется черновое восстановление, то таким объектам будет уточнено расширение с помощью справочника регулярных выражений. При создании, каталогам присваиваются имена типа *FolderXXX* (XXX - номер начального сектора), а файлам - имена типа *FileXXX*.

Режим *анализа данных раздела* можно запускать несколько раз с разными опциями – все результаты сохраняются на дереве объектов проводника, их можно сравнивать и в зависимости от ситуации пользоваться одним или другим. Т.е. можно запустить анализ сначала по одной копии FAT, затем, при необходимости, по другой копии, затем по обеим, при необходимости – игнорировать FAT и т.д. все варианты в зависимости от времени выполнения анализа и полученных результатов.

Необходимо завершить выполнение режима анализа до конца, т.к. прервать исполнение на каком-то промежуточном этапе, то даже в случае если найдены все данные, они могут быть неправильно интерпретированы и не полностью построится дерево файловой системы.

После завершения обработки (и режима), в случае, когда будет найдено достаточно данных для их интерпретации в вариант виртуальной файловой системы, соответствующие объекты появятся в режиме *Проводник*.

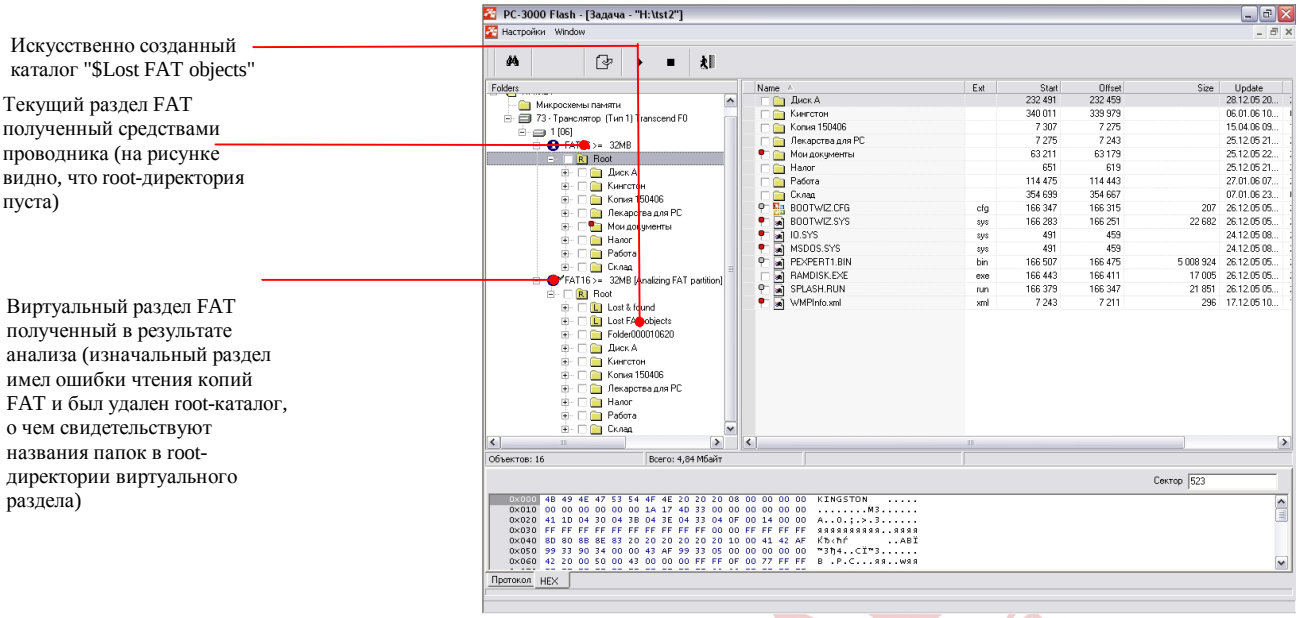


Рисунок 47 Режим проводника с результатами анализа раздела

1.3.4.1 *Особенности использования режима*

В зависимости от исходных данных для восстановления информации следует выбирать различные параметры при запуске данного режима. Принципиальные отличия носят две исходные постановки задачи:

- восстанавливаются данные адресуемые FAT;
- восстанавливаются данные не адресуемые FAT.

1) Восстанавливаются данные адресуемые FAT. Т.е. на восстанавливаемом разделе сохранилась одна или две корректных копии FAT, возможно частично разрушенных, но они относятся к восстанавливаемому разделу (т.е. копии адресуют восстанавливаемые данные). Возможна ситуация когда обе копии целы, а восстанавливаются удаленные файлы или файлы и папки для которых потеряны родительские каталоги (в результате они не видны с помощью проводника). Это не относится к ситуации, когда раздел был переформатирован, т.к. в этом случае, хотя копии FAT и существуют, но они адресуют совершенно другие данные (данные нового раздела).

В большинстве случаев наиболее универсальным можно считать выбор режима *Использовать 1 и 2 копию FAT*. В этом случае, производится чтение обеих копий FAT и строятся две таблицы – объектов и цепочек объектов как пересечение этих копий (т.е. объекты одинаково адресуемые обеими копиями и объекты адресуемы по разному). Режим работы с конкретной копией FAT может быть использован в том случае, если одна из копий сильно разрушена или полностью уничтожена (например, вирусом). Т.к. режим анализа можно запускать несколько раз с различными параметрами, то первоначально, можно запустить анализ по одной копии FAT, затем по другой, потом по обеим и сравнить полученные результаты.

Желательно включить опцию *"Черновое восстановление"*, т.к. в этом случае на этапе построения таблицы цепочек объектов будет производиться предварительное уточнение типа данных. В том случае, если в дальнейшем для данных не будет найден родительский каталог, то они помещаются в искусственно созданный каталог *Lost FAT objects* с расширением, полученным в результате чернового восстановления. Если же черновое восстановление не производилось, то тип данных не будет определен для таких объектов.

При восстановлении удаленных данных необходимо воспользоваться опциями *Искать удаленные файлы* и *Контроль размещения удаленных файлов*.

2) Восстанавливаются данные не адресуемые FAT. В большинстве случаев, эта ситуация возникает после переформатирования раздела (возможно нескольких). В результате форматирования появляются новые копии FAT, которые адресуют данные нового созданного раздела. При этом восстанавливаемые данные будут находиться в областях не адресуемых новыми копиями FAT. Несмотря на то, что копии FAT адресуют данные которые нам не нужны, ими можно воспользоваться в сочетании с опцией *Анализ данных в неадресуемых*

областях исходя из того, что восстанавливаемые данные не должны пересекаться с новыми. Игнорировать копии FAT имеет смысл только в том случае, когда они забиты "мусором".

Использование чернового восстановления является обязательным, т.к. при отсутствии информации из FAT только этот метод восстанавливает карту размещения непрерывных данных пользователя на основании справочника регулярных выражений и карты не занятого пространства. Тип данных, полученный в результате чернового восстановления может, быть уточнен с помощью информации из "потерянных" каталогов, оставшихся от восстанавливаемого раздела (при этом могут быть уточнены и другие атрибуты файлов, в первую очередь имя).

1.3.5 Режим "Сканирования MFT"

Для того чтобы понимать, о чем идет речь, требуются некоторые комментарии к организации структуры данных для разделов типа NTFS.

Данные в NTFS разделах организованы значительно сложнее, чем в случае FAT, и полностью описывать эту организацию в данном разделе мы не будем (подробное описание файловых систем приводится в разделе справочной информации настоящего описания). Пользователь должен лишь знать и понимать несколько вещей:

1) Данные в случае NTFS восстанавливать проще, чем в случае FAT.

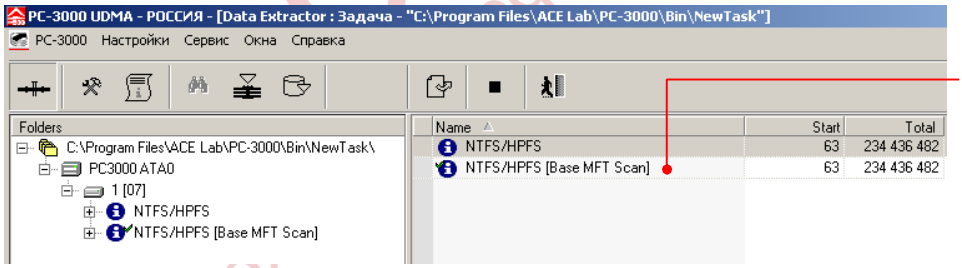
2) В любом разделе NTFS присутствует базовая таблица размещения каталогов и файлов (MFT- Master File Table). В этой таблице содержится полная информация обо всем, что храниться на диске, в том числе и о самой себе (две копии – основная и Mirror).

3) Конкретная запись MFT полностью описывает конкретный объект файловой системы или даже в некоторых случаях хранит данные этого объекта резидентно.

4) Если Вы, открыв с помощью проводника раздел или каталог NTFS, не увидели ничего, то это может лишь означать, что Вам не удалось прочитать что-то из первых 15 служебных записей MFT (вероятней всего корневой каталог) или индексную запись выбранного каталога. Скорее всего, многое из ненайденного (если не все), можно восстановить, используя метод *Сканировать MFT*.

5) Если конкретная запись MFT разрушена, то данные, связанные с ней, не восстановимы.

Метод *Сканировать MFT* позволяет просканировать всю таблицу MFT и построить на ее основании дерево виртуальной файловой системы.



Виртуальный раздел NTFS полученный в результате сканирования таблицы MFT (в скобках указывается с помощью какого дескриптора осуществлялось сканирование, в данном случае - основного)

Рисунок 48 Виртуальный раздел полученный в результате сканирования таблицы MFT

Сканирование таблицы MFT осуществляется на основании информации полученной из текущего дескриптора таблицы MFT (основного или копии). Под дескриптором таблицы MFT понимаются первые 4 записи таблицы. У любого NTFS раздела существует два дескриптора – основной и копия. Если основной дескриптор и копия совпадают, сканирование осуществляется с помощью основного. Если один из дескрипторов некорректен (его не удается вычитать или он содержит мусор), PC-3000 Flash переключается на корректный дескриптор и сканирование осуществляется с его помощью. Если оба дескриптора корректны но различаются, пользователь может выбрать дескриптор с помощью которого будет продолжаться работа и, соответственно, сканирование таблицы MFT (выбор осуществляется с помощью пункта *Выбор дескриптора таблицы MFT*, доступного из контекстного меню загрузочного сектора раздела).

Результатом выполнения режима является построение виртуальной файловой системы выбранного раздела. Файлы и каталоги, для которых не найден "родительский" каталог, помещаются в искусственный каталог *Lost&Found*.

1.3.6 Автоматический метод восстановления NTFS разделов

Автоматический метод восстановления NTFS разделов включают в себя следующие режимы:

1) *Поиск структур NTFS* (режим доступен из контекстного меню объектов проводника "диск" и слот MBR).

2) *Анализ данных раздела* (режим доступен из контекстного меню загрузочного сектора NTFS раздела).

3) *Сканировать незанятое пространство* (режим доступен из контекстного меню загрузочного сектора NTFS раздела).

Метод восстановления NTFS разделов один, однако для него существует три режима отличающиеся друг от друга исходными данными для анализа.

Метод предназначен для полного сканирования области исследуемого устройства на предмет наличия структур NTFS, анализа найденных структур и последующего создания виртуальных разделов NTFS для доступа к пользовательским данным.

Использование данного метода рекомендуется только в случае очень серьезных логических разрушений на полностью исправных устройствах (диски с логическими проблемами или копии дисков с физическими проблемами). В случае неисправных устройств (проблемные для чтения сектора) можно использовать в сочетании с опцией *создавать копию*. Однако более предпочтительно сначала создать полную копию диска (раздела), а затем запускать метод восстановления.

Под серьезными логическими разрушениями подразумеваются разрушения, которые не позволяют извлечь данные с помощью других более простых методов, предлагаемых программой (в первую очередь подразумевается режим *Сканирование MFT*, и исключая режим *Черновое восстановление* – это самый последний в ряду применимых методов).

Существенным недостатком метода является необходимость полного чтения сканируемой области данных и долгий сложный анализ результатов. Для полностью исправного устройства чтение и поиск метаданных может занять несколько часов (в зависимости от объема диска, скорости его чтения, количества искомых и найденных данных).

Существенным достоинством является его высокая эффективность в случае логических разрушений (даже когда поверх одной файловой системы установлена одна или несколько новых).

Режимы полностью автоматические и вмешательство пользователя требуется лишь на первоначальном этапе. Связано это в первую очередь с самой постановкой задачи данного режима – помочь в случае серьезных логических разрушений, которые пользователь не может полностью понять, охарактеризовать и, соответственно, предпринять конкретные действия по их исправлению или просто поиску данных клиента с использованием других средств программы.

Выше упоминалось, что метод восстановления один, однако для него существует несколько вариантов отличающихся между собой разными исходными данными. Поясним это более подробно.

Режим *Поиск структур NTFS* может быть запущен из контекстного меню объекта проводника "диск". Форма задания исходных данных показана на рисунке ниже.

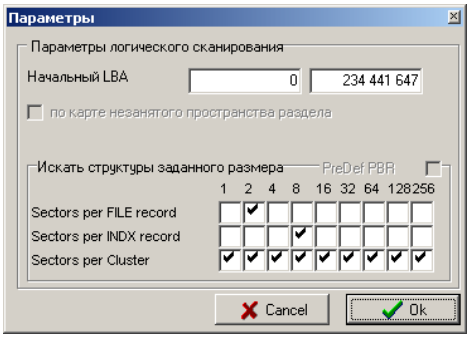


Рисунок 49 Форма задания исходных параметров для режима *Поиск структур NTFS*

В качестве области сканирования будет выбран весь диск и анализироваться будут все возможные значения размера кластера. Это самый трудоемкий вариант восстановления.

Данный режим рекомендуется использовать в двух случаях:

1) Сектор MBR и загрузочные сектора разделов не доступны (или они не вычитываются, или содержат мусор).

Причем, это не относится к случаю, когда загрузочные сектора разделов или их копии существуют и корректны. В этом случае, сектор MBR вероятнее всего может быть восстановлен с помощью режима *Быстрый анализ диска* или в ручном режиме.

2) Восстанавливается NTFS раздел удаленный в результате переформатирования с изменением размеров разделов или их параметров.

Режим *Поиск структур NTFS* также может быть запущен из контекстного меню объекта проводника *слот MBR*. В этом случае, в качестве области сканирования будет выбран уже текущий раздел, но анализироваться будут также все возможные значения размера кластера.

Данный режим может быть использован в следующих случаях:

1) Загрузочный сектор восстанавливаемого раздела и его копия не доступны (или они не вычитываются, или содержат мусор).

2) Восстанавливается NTFS раздел удаленный в результате переформатирования текущего раздела (размер раздела не изменялся).

Режим *Анализ данных раздела* может быть запущен из контекстного меню загрузочного сектора NTFS раздела. Причем, в случае если и загрузочный сектор и его копия не доступны, режим полностью аналогичен предыдущему режиму.

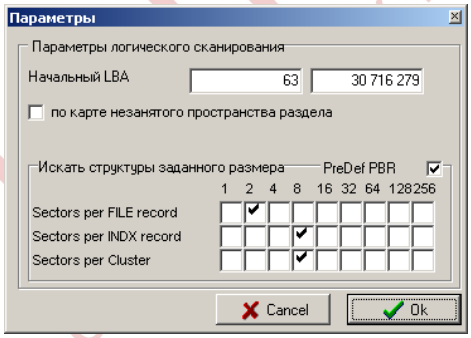


Рисунок 50 Форма задания исходных параметров для режима *Анализ данных раздела*

В качестве области сканирования в параметрах указывается текущий раздел. Из загрузочного сектора (или его копии) по умолчанию подставляется размер кластера, что существенно снижает временные затраты на анализ и уменьшается вероятность принятия ошибочного решения о организации раздела. Информация о размерах структур берется из загрузочного сектора (или его копии) в том случае если установлен переключатель *PreDef PBR*.

Режим может быть использован в случае серьезных логических разрушений раздела, у которого остался корректным загрузочный сектор (или его копия) и он не был переформатирован (для этого случая используется следующий режим).

В параметрах режима пользователь может выбрать опцию *по карте незанятого пространства раздела*, в этом случае будет анализироваться только незанятое пространство раздела (в этом случае могут быть найдены удаленные данные текущего раздела). В том случае, если Вам необходимо осуществить поиск данных возможно оставшихся от предыдущего раздела, то следует либо запустить метод *Сканировать незанятое пространство*, либо настроить текущий метод: установить флажок *по карте незанятого* и пометить все возможные значения *Sectors per Cluster* (т.к. размер кластера у предыдущего размера мог быть совершенно другим).

Режим *Сканировать незанятое пространство* может быть запущен из контекстного меню загрузочного сектора NTFS раздела только при условии корректности *boot* (или его копии) и карты по *Bitmap* (иначе он не доступен).

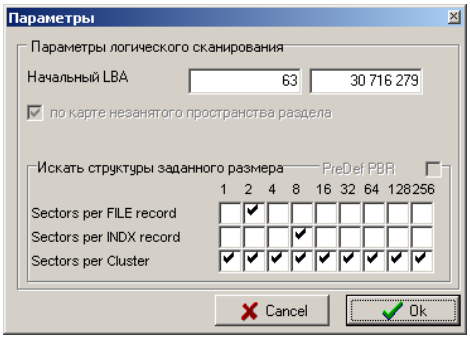


Рисунок 51 Форма задания исходных параметров для режима *Сканировать незанятое пространство*

Как следует из названия режима, анализироваться будет незанятое пространство раздела исходя из *Bitmap* на предмет поиска данных оставшихся от предыдущих NTFS разделов. Т.е. исходная ситуация такова: был раздел NTFS, затем его переформатировали не изменяя размеров, записали какие-то данные и вспомнили, что на предыдущем разделе остались нужные данные. Т.к. текущий раздел не может являться источником информации о размере кластера для предыдущих разделов, то в параметрах анализа устанавливаются все возможные варианты *Sectors per Cluster*.

После определения параметров выполняется сам режим, внешне очень похожий на режимы *Анализ данных раздела* для разделов FAT и *Сканирование MFT* (форма с протоколом и кнопками *Стоп* и *Выход*). Во время выполнения логического восстановления в информационную панель *Операция* выводятся текущие производимые операции.

Состав выполняемых данным режимом действий можно разбить на две важные части – поиск данных и их обработка.

Сначала выполняется поиск в выбранном диапазоне секторов. Причем, если вы запускали данный режим ранее, и был выполнен поиск в областях, входящих в выбранный регион, то эти области будут в процессе поиска пропущены, это не должно вас удивлять (во время поиска в информационную панель *Операция* выводиться номер текущего сектора, например *LBA = 5 167*).

После того как поиск завершиться, выполняется обработка найденных данных. Если поиск будет прерван, программа задаст вопрос о необходимости выполнить обработку.

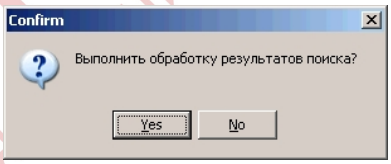


Рисунок 52 Окно сообщения при прерывании процесса поиска

Это связано с тем, что конечный результат обработки может существенно зависеть от всего состава найденных данных (иногда без одного прочитанного сектора можно сделать совершенно другие и иногда неверные выводы об организации данных), и Вы должны сами принять необходимое решение.

С этой же проблемой связано то, что результаты предыдущей обработки данных перед каждой новой обработкой данных удаляются, для того чтобы построить новую модель данных на основе всей совокупности найденной информации.

После завершения обработки (и режима), в случае, когда будет найдено достаточно данных для их интерпретации в вариант виртуальной файловой системы, соответствующие объекты появятся в режиме *Проводник*.

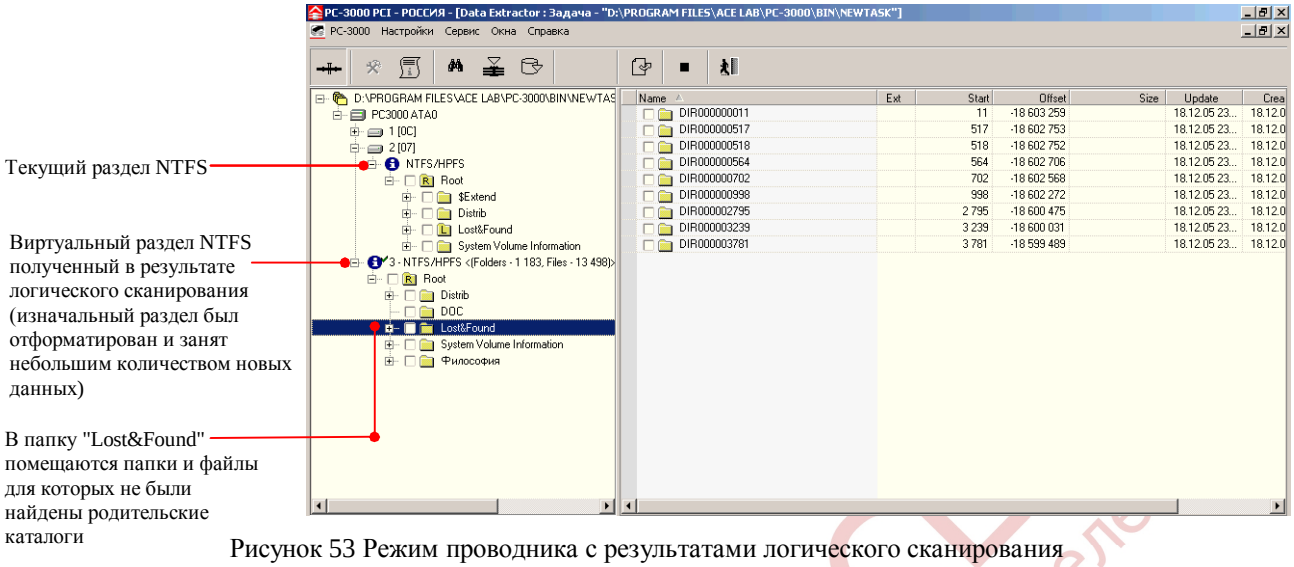


Рисунок 53 Режим проводника с результатами логического сканирования

1.3.7 Режим просмотра и редактирования сектора

Этот режим предназначен для просмотра и редактирования данных конкретного прочитанного сектора и доступен в следующих случаях:

– При работе с картой копирования или объекта. Для того чтобы перейти в него, нужно два раза щелкнуть по квадратику соответствующего сектора или выполнить пункт контекстного меню карты *Редактировать* или пункт контекстного меню списка цепочек карты объекта *Просмотр первого сектора*.

– При работе в режимах *Поиск регулярных выражений* и *Черновое восстановление* - это пункт контекстного меню элемента списка результатов *Просмотр сектора*.

– В ходе работы в режиме *Проводник* - это пункт контекстного меню объекта *Просмотр первого сектора*.

После вызова появится модальное окно режима редактирования:

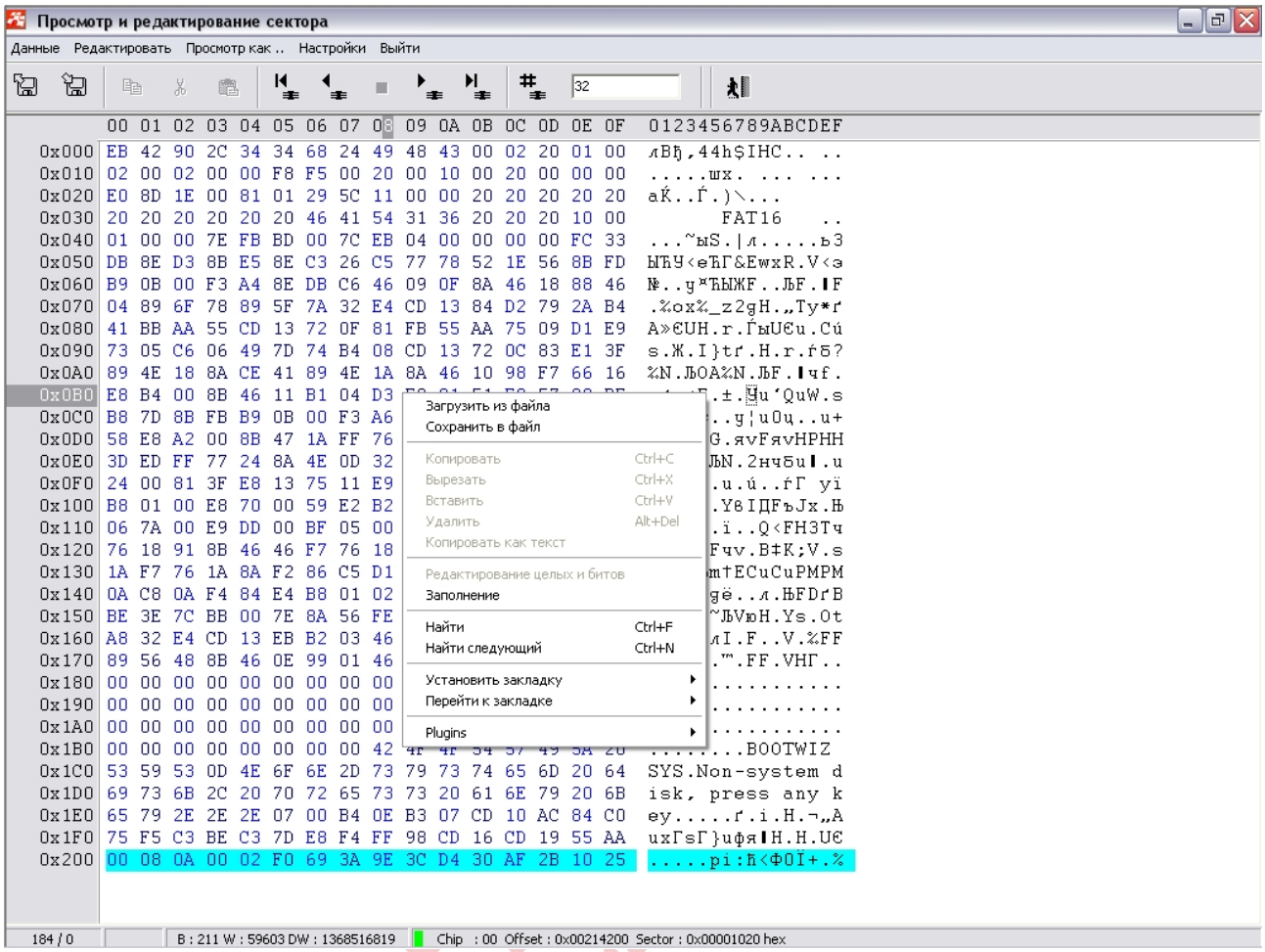


Рисунок 54 Внешний вид режима просмотра и редактирования сектора

В верхней части окна располагается основное меню формы и панель кнопок быстрого доступа.

В центральной части окна находится область для просмотра и редактирования данных. Эта область разделена на две части. В левой части – шестнадцатеричное представление данных, в правой – символьное.

В нижней части окна расположена панель статуса и дополнительной информации.

1.3.7.1 Пункт меню "Данные"

Пункт меню *Данные* позволяет Вам:

- *Загрузить* (*Ctrl+R*) или прочитать данные;
- *Сохранить* (*F2*) данные в соответствующий сектор;
- *Сохранить в файл*;
- *Загрузить из файла*.

Метод *Загрузить* позволяет Вам повторно прочитать данные текущего сектора.

Метод *Сохранить* позволяет Вам сохранить отредактированные или загруженные из файла данные в сектор.

Метод *Сохранить в файл* предназначен для сохранения данных прочитанного сектора в файл.

Метод *Загрузить из файла* позволяет загрузить в редактор данные из файла.

1.3.7.2 Пункт меню "Редактирование"

Объединяет группу методов поиска и редактирование данных сектора:

- *Найти;*

- *Найти следующий;*
- *Заполнение;*
- *Редактирование целых и битов;*
- *Выделить все.*

Метод *Найти* позволяет искать по маске позицию искомых данных в пределах загруженного сектора.

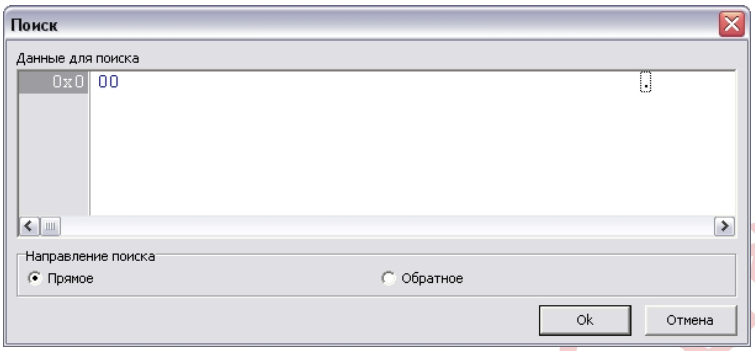


Рисунок 55 Окно поиска

Метод *Найти следующий* позволяет, не меняя маски, найти следующее вхождение искомой структуры.

Метод *Заполнение* позволяет выполнить заполнение выделенной последовательности байтов данных байтом-заполнителем. При этом пользователь может, как просто скопировать байт-заполнитель в данные файла (оператор "="), так и заменить исходные данные результатом побитовой операции каждого байта исходных данных и байта-заполнителя (операторы "AND", "OR", "XOR").

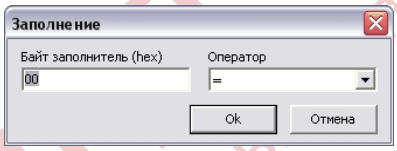


Рисунок 56 Окно установки параметров заполнения

Двоичный редактор позволяет редактировать фрагменты редактируемого документа как одно, двух или четырех байтовые целые числа. Для того чтобы начать редактирование необходимо выделить в редакторе фрагмент данных длинной 1, 2 или 4 байта и выбрать в меню *Правка* пункт *Редактирование целых и битов* (Ctrl+B). По этой команде отображается форма редактирования. Вы можете редактировать значение в виде десятичного, шестнадцатеричного или двоичного представлений.

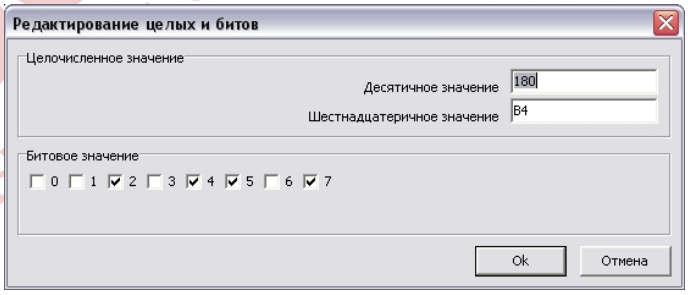


Рисунок 57 Редактирование целых и битов

Метод *Выделить все* предназначен для выделения без мышки и клавиатуры всей области данных. С учетом того, что многие методы режима работают с выделенной областью (в том числе работа с буфером обмена), это бывает полезно.

1.3.7.3 Пункт меню "Просмотр как…"

Объединяет группу методов для интерпретации и просмотра данных прочитанного сектора в виде конкретной структуры:

- *Partition table* (таблицы разделов);

– *Boot FAT16* (загрузочный сектор раздела типа FAT16 и FAT12);
– *Boot FAT32* (загрузочный сектор раздела типа FAT32);
– *Boot NTFS* (загрузочный сектор раздела типа NTFS);
– *FAT folder* (каталог раздела FAT);
– *Ext2(3) superblock* (суперблок, аналог *boot*, раздела LINUX с файловой системой EXT2 или EXT3);
– *Ext2(3) GroupDescriptors* (сектор таблицы дескрипторов групп раздела LINUX с файловой системой EXT2 или EXT3);
– *HFS+ Volume Header*.

Все представления (за исключением каталога FAT) позволяют редактировать данные полей структур и могут быть использованы при восстановлении данных. Данные методы доступны не только из двоичного редактора, но и с помощью пункта *Свойства* контекстных меню соответствующих объектов режима *Проводник*.

Внешний вид некоторых представлений приведен на рисунках ниже:

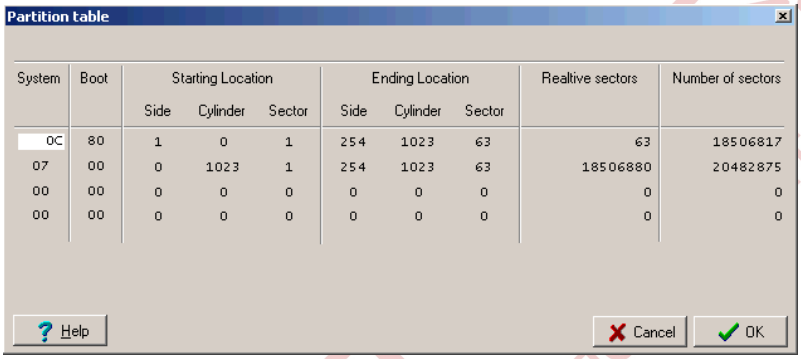


Рисунок 58 Просмотр описателей накопителя в режиме "Partition table"

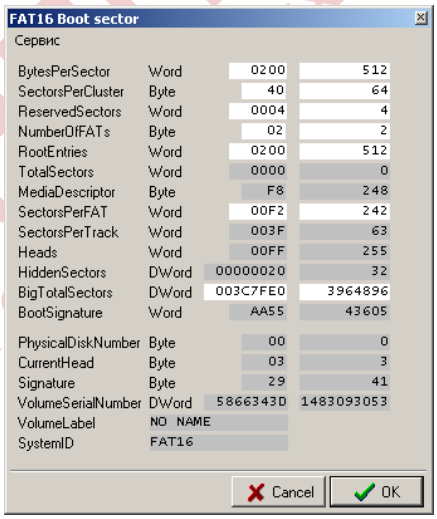


Рисунок 59 Просмотр загрузочного сектора в режиме "Boot FAT16"

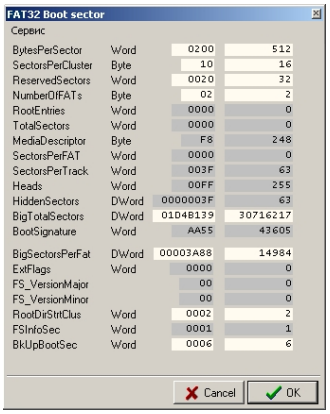


Рисунок 60 Просмотр загрузочного сектора в режиме "Boot FAT32"

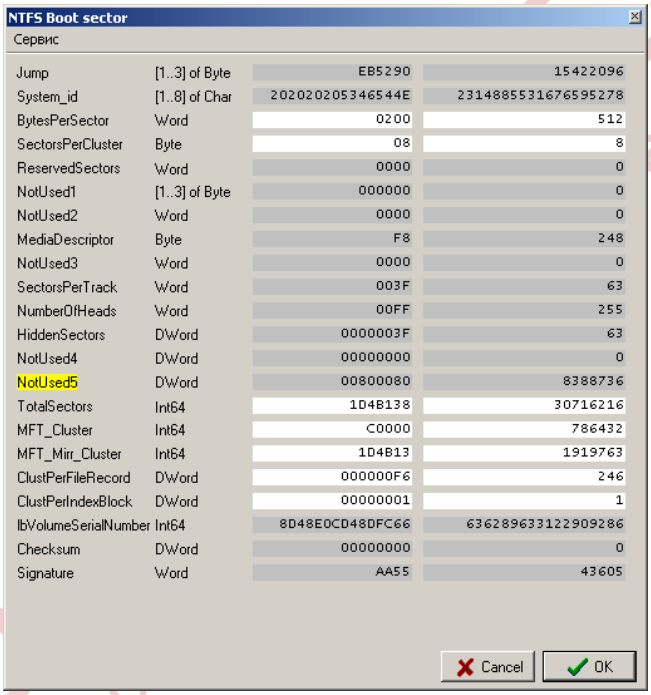


Рисунок 61 Просмотр загрузочного сектора в режиме "Boot NTFS"

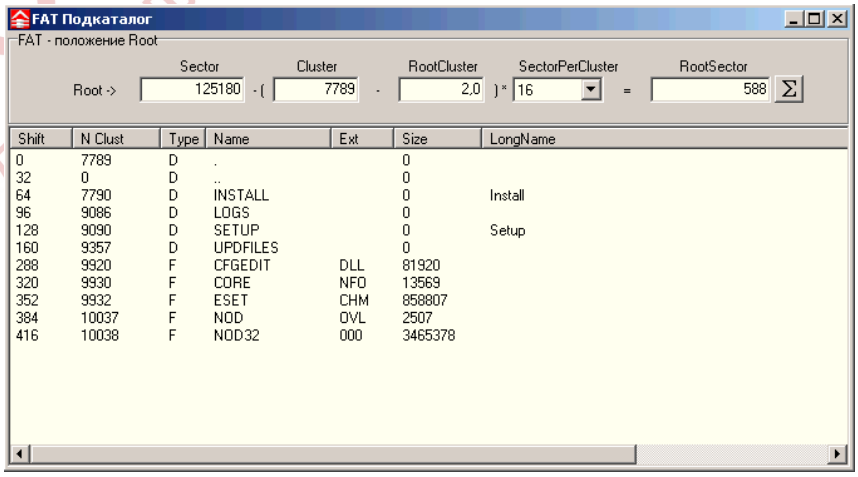


Рисунок 62 Просмотр сектора с элементами каталога в режиме "FAT folder"

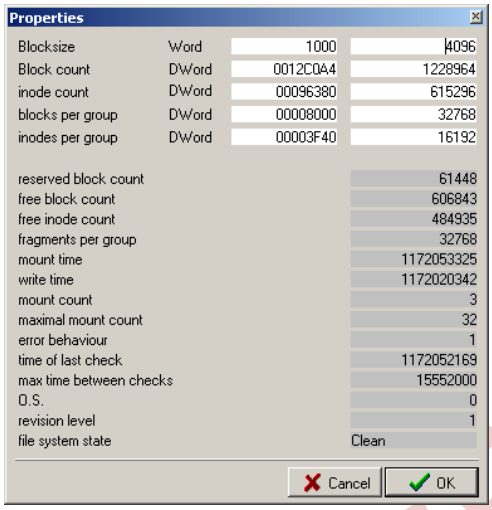


Рисунок 63 Просмотр суперблока Ext2(3)

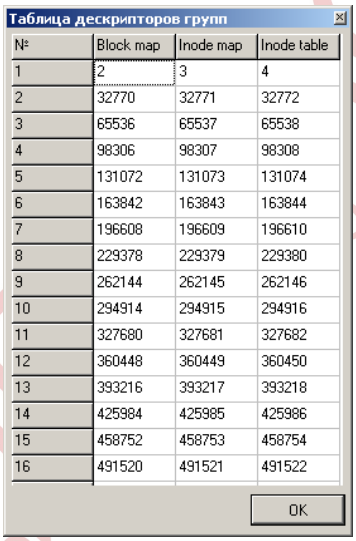


Рисунок 64 Просмотр таблицы дескрипторов групп Ext2(3)

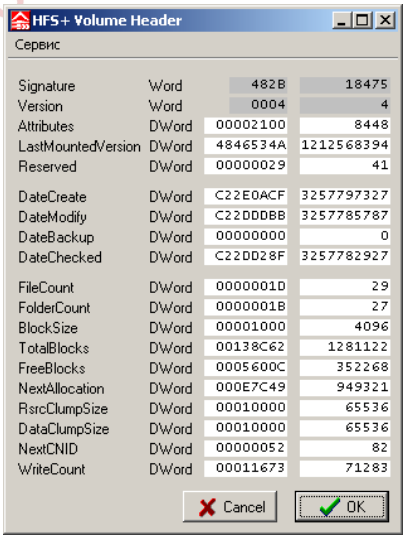


Рисунок 65 Просмотр HFS+ *Volume Header*

1.3.7.4 *Пункт меню "Настройки"*

Данный пункт меню позволяет изменить форму представления данных на панели данных:
- установить количество байт в строке;
- установить количество байт в колонке;
- поменять шрифт и его размер;
- поменять трансляцию текстового представления;
- поменять цвета.

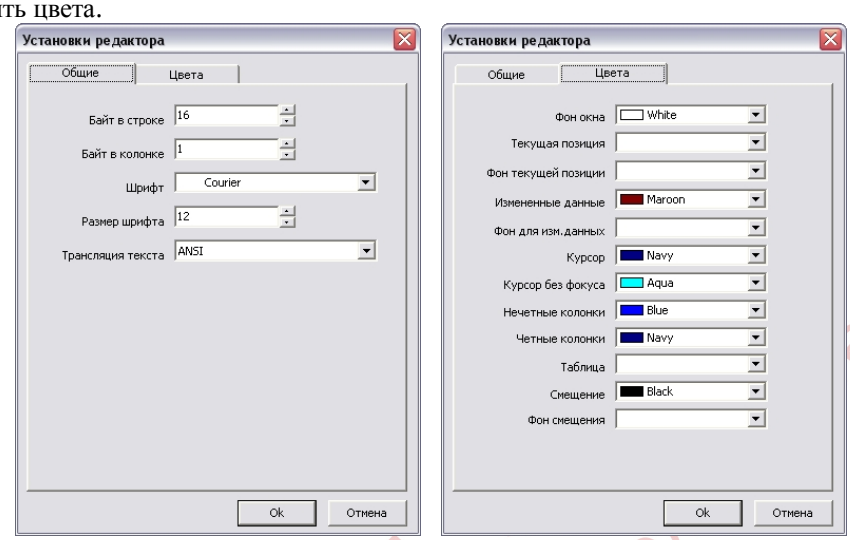


Рисунок 66 Окно установок редактора двоичных данных

1.3.7.5 *Панель кнопок быстрого доступа*

В верхней части окна расположены оперативная панель навигации для перемещения и информационные строки, отображающие текущее состояние. Назначение клавиш оперативной панели приводится ниже.

Рисунок 67 Оперативная панель двоичного редактора

- Загрузить/Сохранить данные;
- Редактирование (*Копировать*, *Вырезать*, *Вставить*);
- Управление процессом позиционирования (*Первый*, *Предыдущий*, *Следующий*, *Последний*);
- *Перейти к сектору* и окно номера сектора на который будет осуществляться переход;
- Выйти из задачи.

1.3.7.6 *Панель просмотра и редактирования, закладки*

Эта панель основная и служит для просмотра и редактирования загруженных данных.

В левой части панели – шестнадцатеричное представление данных, в правой – текстовое.

Методы, применимые к этой панели, доступны через контекстное меню, появляющееся при нажатии правой кнопки манипулятора мышь:

- Загрузить из файла;
- Сохранить в файл;
- Копировать (Ctrl+C);
- Вырезать (Ctrl+X);
- Вставить (Ctrl+V);
- Удалить (Alt+Del);
- Копировать как текст;
- Редактирование целых и битов;
- Заполнение;

– Найти (Ctrl+F);
– Найти следующий (Ctrl+N);
– Установить закладку;
– Перейти к закладке;
– Plugins.

Часть этих методов дублирует методы основного меню и панели кнопок "быстрого доступа", часть доступна лишь через контекстное меню (например, работа с закладками и копирование данных как текст).

Для установки закладки используется подменю *Установить закладку* контекстного меню редактора или "горячие" клавиши *Shift+Ctrl+N* (где N – номер закладки, всего можно установить 10 закладок). Выбор пункта подменю устанавливает закладку, который отображается в редакторе слева от смещения строки, а выбранный пункт меню становится "отмеченным". Повторный выбор пункта подменю *Установить закладку* удаляет закладку и снимает с пункта меню признак "отметки".

Для того, чтобы перейти к установленной закладке следует выбрать требуемый пункт подменю *Перейти к закладке* выпадающего контекстного меню редактора или воспользоваться "горячими" клавишами *Ctrl+N* (где N – номер закладки). Пункты подменю соответствующие установленным закладкам отображаются как "отмеченные".

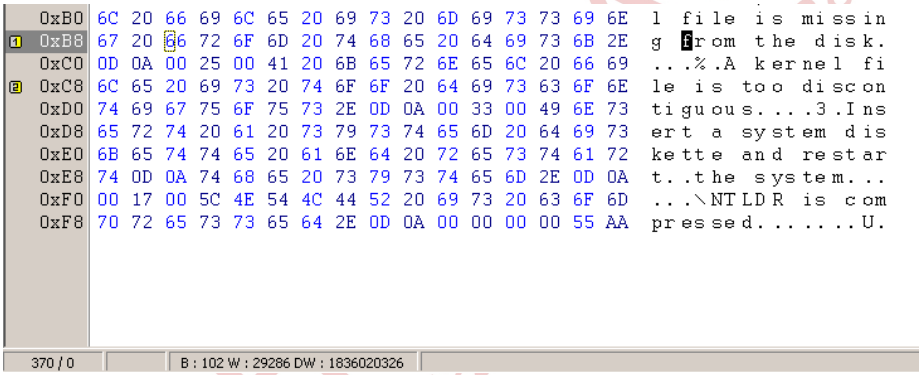


Рисунок 68 Окно двоичного редактора с установленными закладками

Для того чтобы скопировать данные как текст, необходимо выделить фрагмент редактируемых данных, и выбрать пункт *Копировать как текст* контекстного меню. Поскольку копируются не сами данные, а их текстовое представление, то на экран выводится форма ввода параметров представления данных.

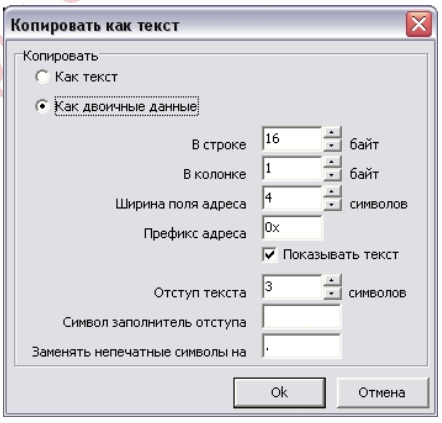


Рисунок 69 Окно копирования данных как текста

Текст/двоичные данные. Если выбран режим *Текст*, то редактируемые данные считаются текстом, и помещаются в буфер обмена без преобразования. Не следует использовать этот способ копирования, если данные содержат непечатные символы. Если выбран режим *Двоичные данные*, то в буфер обмена помещается текстовый отчет, внешне повторяющий вид редактора двоичных данных. Остальные вводимые параметры влияют на его формирование и доступны только в режиме *Двоичные данные*.

Байт в строке - количество байт в каждой строке представления.

Байт в колонке - количество байт в каждой колонке представления.

Ширина адреса - устанавливает размер поля адреса (смешения). Значения дополняются до указанной ширины нулями слева.

Префикс адреса - устанавливает текст, который выводится слева от значения адреса (например "0x" или "$").

Показывать текст - признак наличия / отсутствия текстового представления двоичных данных.

Отступ текста - количество символов разделяющих область отображение текстовых и бинарных данных.

Символ отступа - символ разделитель текстовых и бинарных данных.

Заменять непечатные символы - символ на который будут заменяться непечатные символы в области текстовых данных.

Plugins. В данный раздел контекстного меню предполагается включать расширения двоичного редактора. На данном этапе в это подменю включен один метод - *Add Grep*. Данный метод позволяет добавить из двоичного редактора в справочник регулярных выражений требуемый критерий поиска. Для этого необходимо в двоичном редакторе выделить требуемую подстроку данных (которая будет служить критерием поиска) и с помощью контекстного меню вызвать метод *Plugins → Add Grep*. При этом на экран выводится форма для редактирования регулярного выражения в которой можно задать наименование критерия (по умолчанию – *New Grep*), категорию и при необходимости незначительно модифицировать критерий поиска.

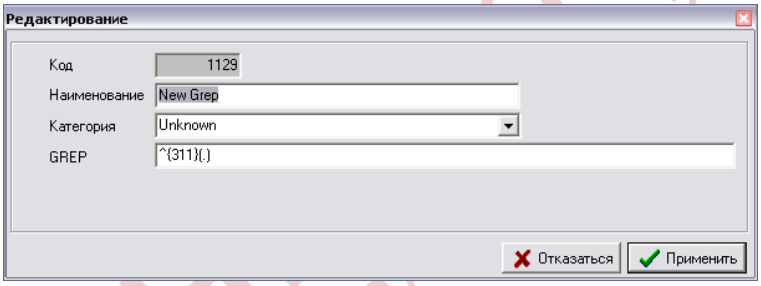


Рисунок 70 Форма редактирования критерия поиска

1.3.7.7 Панель статуса и дополнительной информации

Строка состояния отражает текущее состояние редактирования. Панель разбита на четыре части:

1) смещение от начала и длина выделения (dec);

2) признак наличия не сохраненных изменений (при этом измененные данные в окне просмотра и редактирования выделяются с помощью цвета фона);

3) информация о значении данных с текущей позиции курсора (Byte,Word, DoubleWord);

4) информация о статусе прочитанного сектора (в соответствии с легендой в режиме создания копии).

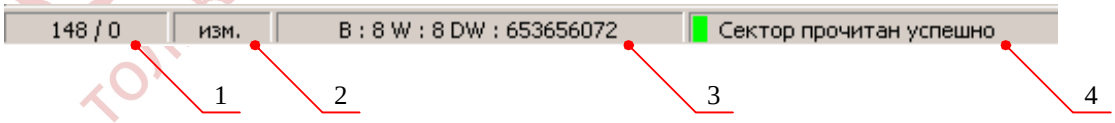


Рисунок 71 Строка состояния двоичного редактора

1.3.8 Редактор записи MFT

1.3.8.1 Назначение

Основное назначение *Редактора записи MFT* – визуальное представление логической структуры записи MFT, помощь пользователю в ее понимании, и, при необходимости, модификации.

1.3.8.2 Внешний вид, управление

Внешний вид *Редактора записи MFT* приведен ниже.

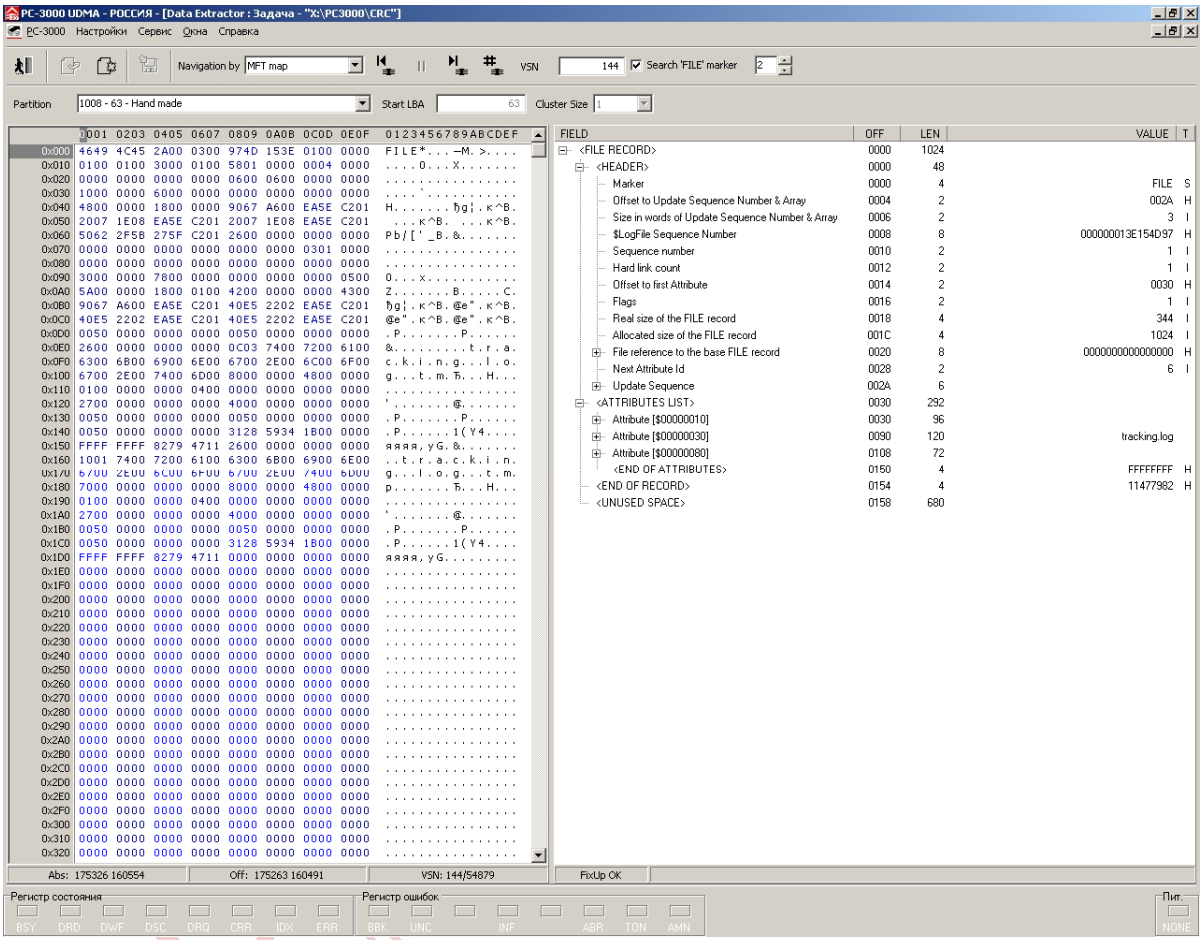


Рисунок 72 Внешний вид Редактора записи MFT

В левой части экрана располагается панель HEX-редактора, справа – панель древовидной структуры записи.

Навигация по обеим панелям и между ними осуществляется с помощью клавиатуры или с помощью мыши. При навигации по любой из панелей, на другой панели подсветкой выделяется соответствующее поле записи.

Над панелями расположена оперативная панель кнопок "быстрого" доступа и органов управления

1.3.8.3 Панель инструментов

Оперативная панель инструментов показана на рисунке ниже.

Рисунок 73 Панель инструментов редактора записи MFT

Оперативная панель разбита на несколько групп:

1) Выход из редактора;

2) Контроль целостности содержимого записи: восстановление и подготовка к записи;

При загрузке MFT записи в буфер HEX-редактора производится контроль целостности содержимого записи и ее восстановление (см. п.6.3.4). Результат этой операции отображается в строке состояния древовидной структуры, и может принимать следующие значения:

– *FixUp OK* – запись успешно восстановлена;

– *FixUp OK (1st)* – успешно восстановлен первый сектор записи, а реальный размер записи *Real size of the FILE Record* меньше или равный 512 байт позволяет игнорировать ошибку восстановления второго сектора MFT записи;

– FixUp Error – не удалось восстановить запись, для дальнейшего редактирования записи требуется правка полей *Update Sequence Number & Array*.

3) Сохранение изменений.

Позволяет сохранить результаты редактирования записи на диск задачи. Перед сохранением MFT запись должна быть восстановлена и не должна иметь проблем с подготовкой к записи, т.е. состояние записи должно быть *FixUp OK*. Производится операция *подготовка к записи*, MFT запись сохраняется на диск задачи и остается в невосстановленном состоянии.

4) Навигация: *Режим*, (*Вперед*, *Назад*, *Перейти*) или (*Поиск вперед*, *Прервать*, *Поиск назад*, *Перейти*), окно ввода LBA/VSN/REC, включение режима поиска и стоп-условие поиска;

В зависимости от места старта *Редактора MFT записи* и состояния задачи доступны от одного до трех Режимов Навигации.

Навигация по *Drive* доступна всегда, независимо от места старта *Редактора MFT записи* и состояния задачи.

Навигация по *Corrupted MFT records* доступна в случае, когда были выполнены *Scan MFT* и/или *Partition analysis* раздела диска задачи, и/или *Search NTFS structures* на диске задачи. В этом режиме навигация осуществляется по списку предварительно найденных и помеченных как *corrupted* записей MFT. Порядковый номер загруженной записи и общее количество *corrupted* записей MFT отображается в третьей секции строки состояния HEX-редактора.

Навигация по *MFT map* доступна только при запуске *Редактора MFT записи* с раздела NTFS диска задачи и наличия «живой» карты MFT. Навигация осуществляется по виртуальным секторам карты MFT, порядковый номер виртуального сектора и общее количество виртуальных секторов карты MFT отображается в третьей секции строки состояния HEX-редактора.

Кнопки быстрого доступа *Вперед* и *Назад* позволяют последовательно перемещаться в выбранном направлении по записям/секторам. Кнопка *Перейти* позволяет осуществить непосредственный переход к выбранному сектору/записи, введя номер сектора/записи непосредственно в окно ввода.

При навигации по *Drive* или *MFT map* и включенном режиме поиска (элементы управления *Включение режима поиска* и *стоп-условие поиска*) кнопки *Поиск вперед* и *Поиск назад* позволяют осуществлять поиск маркера MFT записи «FILE», расположенного в первых четырех байтах записи, в заданном направлении. *Стоп-условие поиска* задает количество совпавших символов в маркере, например: при *Стоп-условие поиска* равном 3, редактор MFT записи остановится на записи, у которой в первых четырех байтах находится любая из комбинаций : xILE, FxLE, FIxE, FILx, где x – любой символ. Для принудительного прерывания поиска служит кнопка *Прервать*, остановка также произойдет при достижении граничных значений номера сектора - 0 или MAX.

5) Раздел диска *Partition*, начальный сектор раздела *Start LBA*, размер кластера *Cluster Size*.

Конечным результатом редактирования MFT записи является сохранение пользовательских данных. Сохранение пользовательских данных, хранящихся в нерезидентных атрибутах, требует знания начального сектора и размера кластера предполагаемого или реального раздела NTFS. Для этих целей и служит эта секция панели инструментов. Карта нерезидентного атрибута формируется исходя из значений *Start LBA* и *Cluster Size*. У пользователя есть два способа задать значения полей:

– выбрать раздел NTFS в выпадающем списке. Значения *Start LBA* и *Cluster Size* будут подставлены из параметров выбранного раздела, ручной ввод значений полей *Start LBA* и *Cluster Size* – запрещен.

– выбрать *Partition is not selected* в выпадающем списке и ввести значения *Start LBA* и *Cluster Size* вручную.

Состояние *Partition* влияет на доступность метода *Parse record, link to partition* панели HEX-редактора. При значении *Partition* равным *Partition is not selected* метод не доступен.

Значение окна ввода *Cluster Size* влияет на доступность методов *Карта* и *Сохранение атрибута в файл* панели древовидной структуры. При значении *Cluster Size* равном нулю, методы не доступны.

1.3.8.4 Панель HEX-редактора

HEX-редактор – стандартный, с двумя представлениями буфера - шестнадцатеричным и символьным.

Если поле записи является смещением, то в HEX-редакторе подсвечивается (красный шрифт на черном фоне) и позиция по значению поля.

Оба представления позволяют редактировать запись. HEX-редактор не имеет ограничений на редактирование записи. Под панелью HEX-редактора расположена *строка состояния*, разделенная на три секции. Первая секция отображает загруженные сектора относительно нулевого сектора диска – абсолютные LBA. Вторая секция отображает загруженные сектора относительно нулевого сектора выбранного раздела диска, или относительно поля *Start LBA*. Третья секция отображает *Номер Записи* / *Всего Записей* в режиме навигации по *Corrupted MFT records* или *VirtualSectorNumber* / *MaxVSN* в режиме навигации по *MFT map*.

Рисунок 74 Строка состояния HEX-редактора при навигации по *Drive*

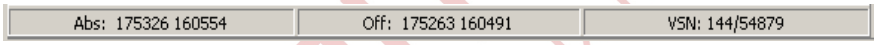


Рисунок 75 Строка состояния HEX-редактора при навигации по *MFT map*

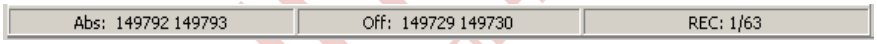


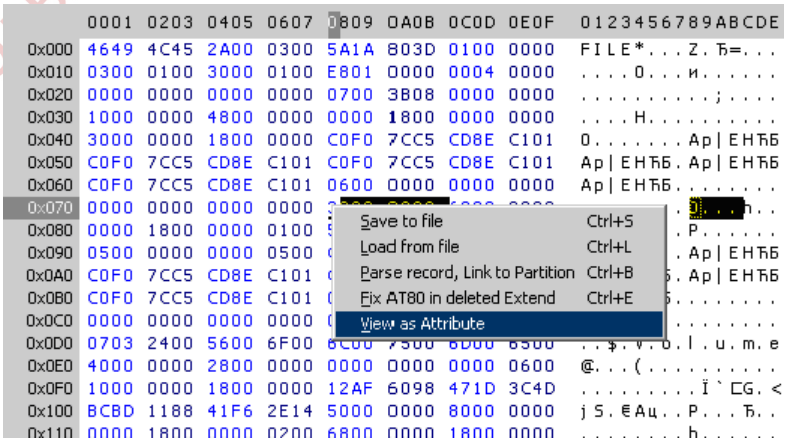
Рисунок 76 Строка состояния HEX-редактора при навигации по *Corrupted MFT records*

1.3.8.4.1 Доступные методы панели HEX-редактора

1) Методы *Save to file* и *Load from file*. Методы доступны постоянно. Позволяют сохранить в файл или, соответственно, загрузить из файла запись MFT.

2) Метод *View as Attribute* и *View as MFT Record*. В зависимости от положения курсора (смещения) в HEX-редакторе доступен метод *View as Attribute* (смещение больше 0), или *View as MFT Record* (смещение равно 0).

Панель древовидной структуры переходит в режим отображения соответственно атрибута (части MFT записи) или всей MFT записи.



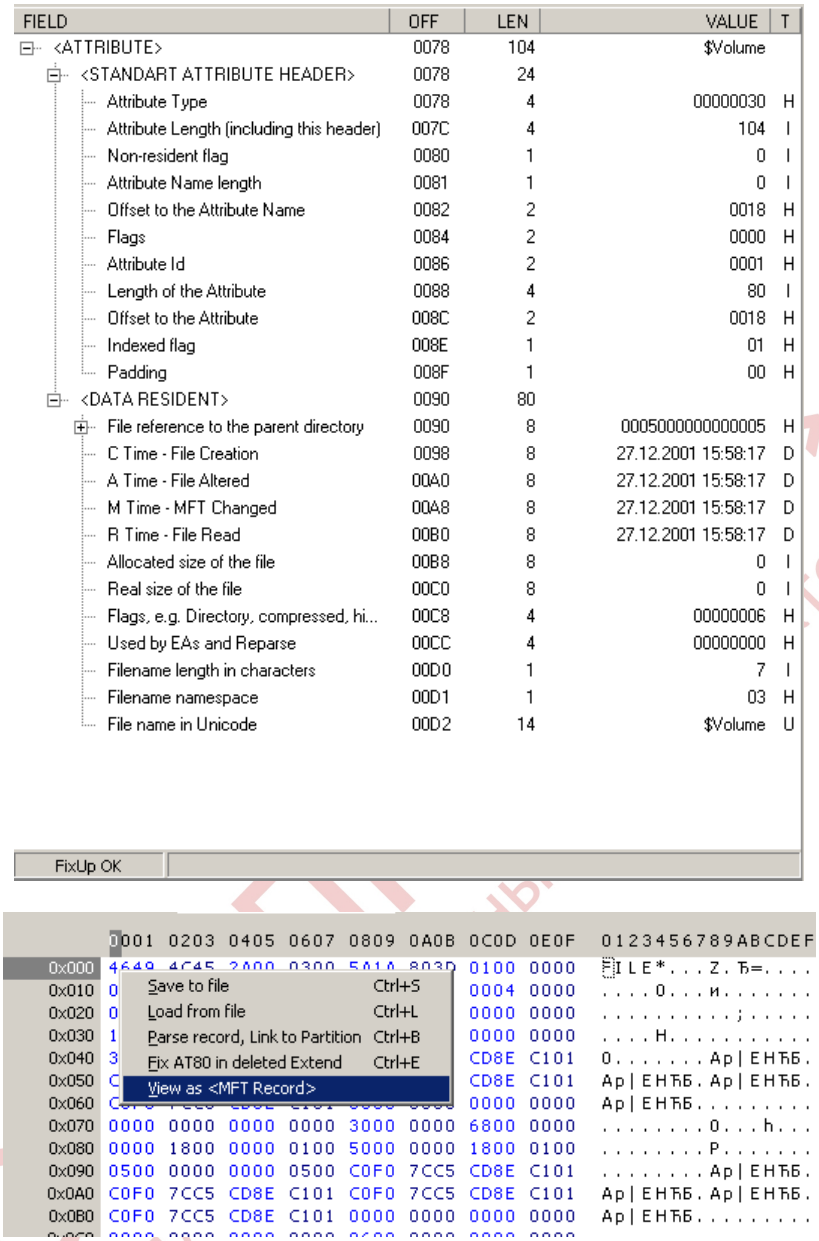


Рисунок 77 Метод *View as Attribute* и *View as MFT Record*.

3) Метод *Parse record, link to partition*. Метод предназначен для привязки MFT записи, не принадлежащей ни к одному разделу NTFS, к выбранному разделу NTFS, и обычно завершает процедуру ремонта записи.

Метод доступен если предварительно был выбран раздел NTFS на панели инструментов. В случае успешного завершения метода файл, описываемый MFT записью, будет помещен в каталог *Lost&Found* раздела NTFS.

4) Метод *Fix AT80 in deleted Extend*. Метод предназначен для восстановления атрибута $80 в удаленной EXTEND MFT записи и позволяет восстановить полностью (если основная запись была расширена одной дополнительной) или частично карту пользовательских данных.

Метод доступен при соблюдении следующих условий: список атрибутов пуст и *File reference to the FILE record* отличен от нуля.

Метод основан на особенности удаления подобных записей и не дает 100% гарантии восстановления.

1.3.8.5 Панель древовидной структуры

Панель предназначена для логического разбора, визуализации структуры записи MFT и ее редактирования. Каждое поле древовидной структуры описано структурой:

FIELD – имя поля;

OFF – абсолютное смещение поля в записи;

LEN – размер поля в байтах;

VALUE – значение;

T – представление значения (*S* – строковое, *I* – целое, *H* – шестнадцатеричное, *U* - строковое Unicode, *D* – дата время, ‘ ’ – нет)

Редактирование полей записи на панели древовидной структуры ограничено типами *S, I, H*. Переход в режим редактирования записи на панели древовидной структуры – клавишей *ENTER*.

Если поле является структурой, колонка *VALUE* иногда используется для вывода дополнительной информации, например, для *Attribute*, имеющего тип $30, дублируется поле *File name* резидентных данных.

Все ошибки логического разбора записи MFT выделяются подсветкой – красным шрифтом.

Под панелью древовидной структуры расположена *строка состояния*, разделенная на две секции. Первая секция отображает состояние контроля целостности содержимого записи. Вторая секция предназначена для отображения краткого описания ошибки выбранного поля записи MFT.

Из-за большого количества вариантов разрушений не все ошибки в настоящий момент имеют свои описания и рекомендации по исправлению. В процессе развития *Редактора MFT записи* их количество и качество будет возрастать.

Рисунок 78 Строка состояния панели древовидной структуры

1.3.8.5.1 Доступные методы панели древовидной структуры

1) Представление целых.

Метод доступен только для целочисленных полей структуры записи. С помощью этого метода можно изменить представление поля с целого на шестнадцатеричное и обратно.

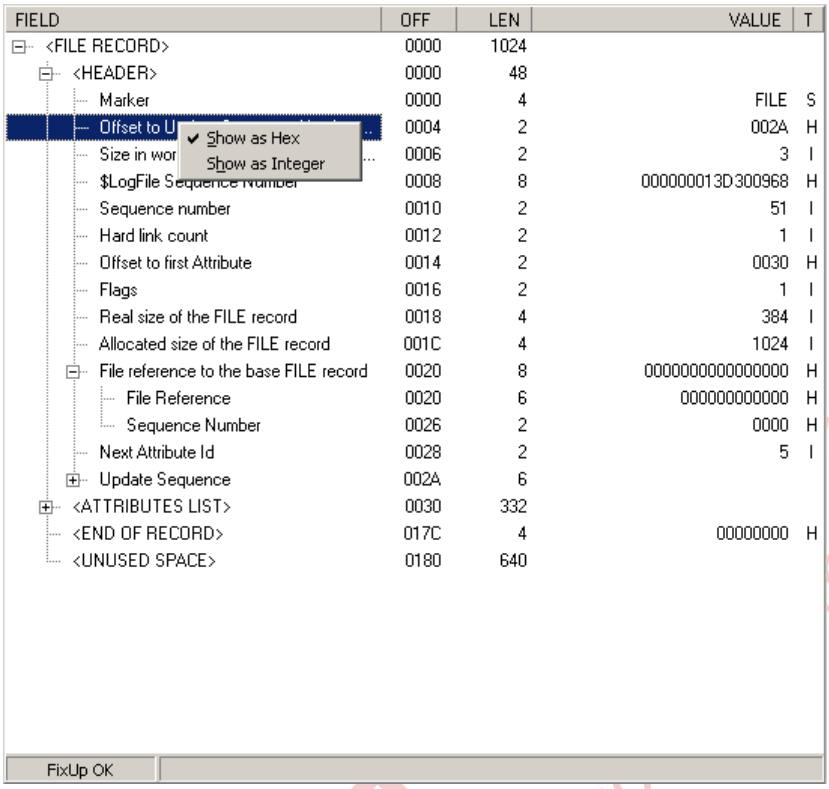


Рисунок 79 Представление целых

2) Карта.

Метод доступен для поля *DATA RUNS*, при условии: заданы начало раздела (нулевой сектор раздела), размер кластера раздела и поле не имеет ошибок формирования цепочек.

Метод открывает стандартную форму PC-3000 Flash для просмотра карты.

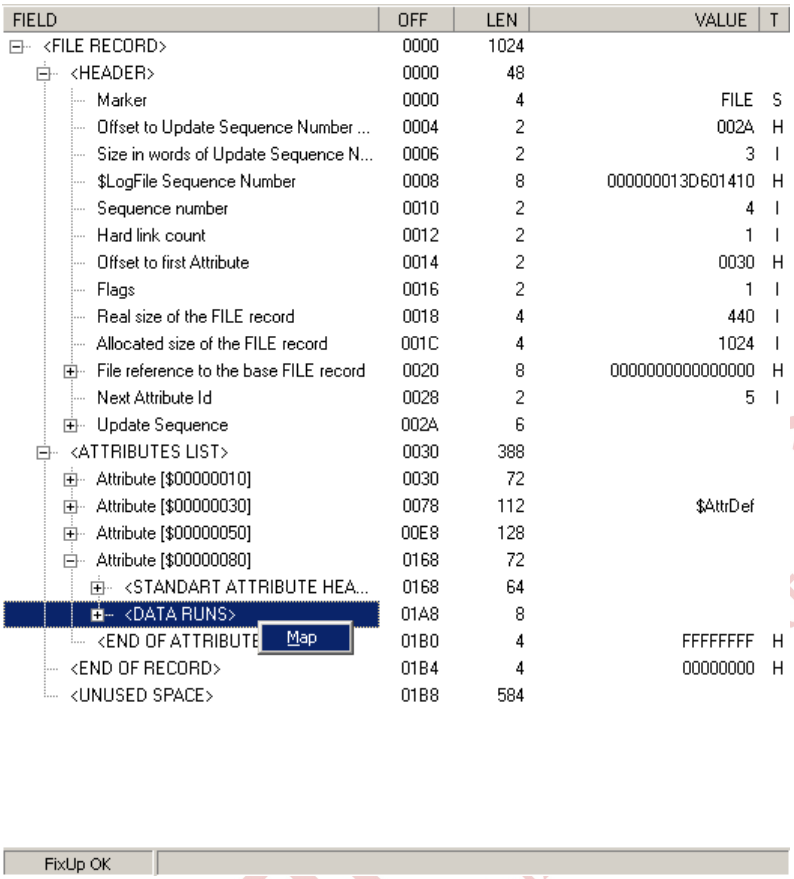


Рисунок 80 Карта

3) Сохранение атрибута в файл.

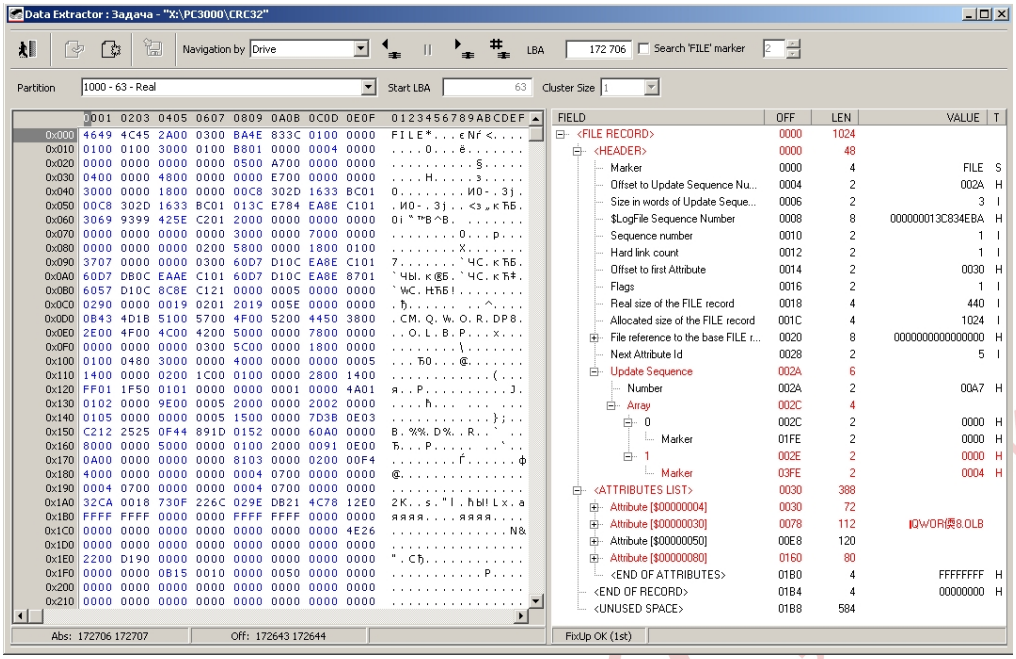
Метод доступен для любого резидентного или нерезидентного атрибута записи, если атрибут не имеет ошибок логического разбора записи и заданы начало раздела (нулевой сектор раздела) и размер кластера раздела.

При сохранении атрибута автоматически формируется имя файла LBA(XXXXXXX)_$XXXXXXXX, например: имя файла LBA(149790)_$00000050 говорит о том, что был сохранен атрибут $00000050 записи, расположенной по LBA = 149790.

Если атрибут именован, то имя атрибута добавляется к имени файла LBA(149790)_$00000050_<имя_атрибута>

Если сохраняется неименованный атрибут $80 и запись MFT содержит корректный атрибут $30, то имя файла подставляется из атрибута $30.

1.3.8.6 Пример восстановления. MFT запись с множественными случайными разрушениями



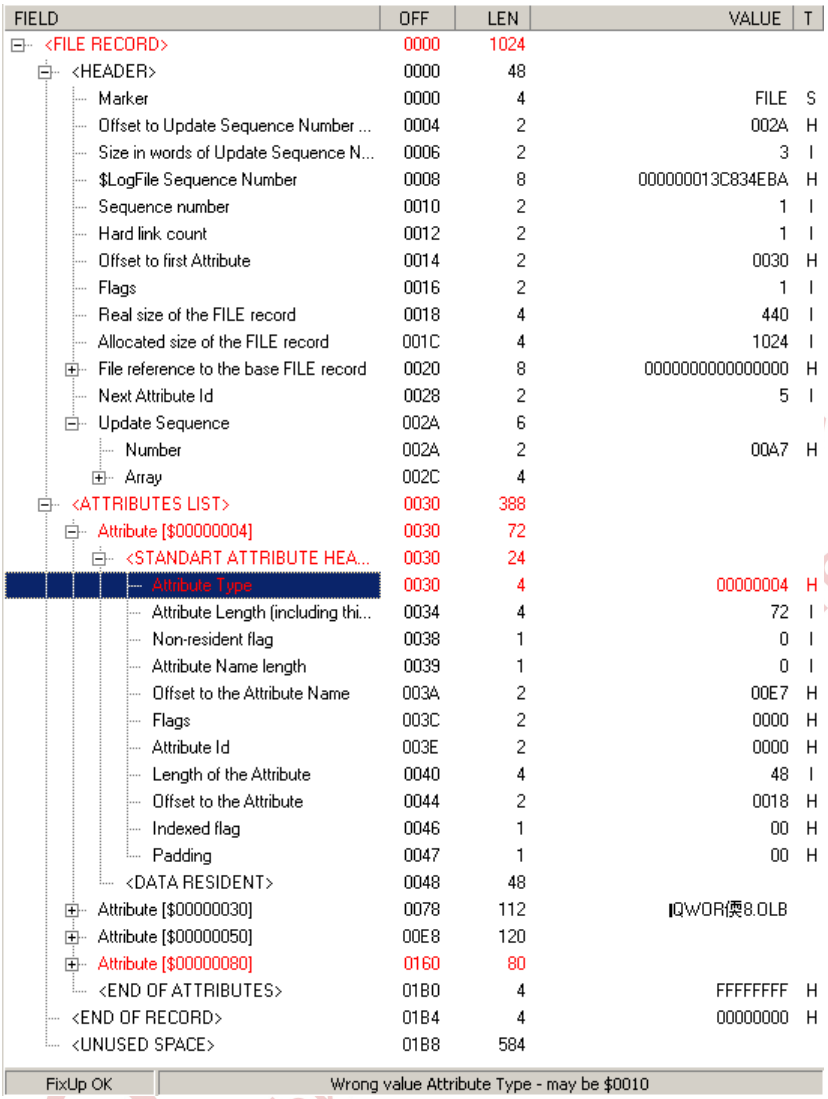
Запись восстанавливаем, последовательно восстанавливая ее поля, так как разрушенное поле часто искажает и следующие за ним.

Редактируем *<HEADER>*. Запись частично прошла контроль целостности, о чем свидетельствует статус записи *FixUp OK (1st).*

Необходимо откорректировать значение *Update Sequence → Array → 1 → Marker*. Правильное значение берем из *Update Sequence → Number* = $00A7.

Сначала кнопкой *UnFix* переводим запись в состояние готовности к записи. Исправляем *Update Sequence → Array → 1 → Marker* и проверяем контроль целостности кнопкой *FixUp*.

Статус записи изменился на *FixUp OK* и *<HEADER>* не имеет ошибок.



FIELD	OFF	LEN	VALUE	T
<FILE RECORD>	0000	1024		
<HEADER>	0000	48		
Marker	0000	4	FILE	S
Offset to Update Sequence Number ...	0004	2	002A	H
Size in words of Update Sequence N...	0006	2	3	I
$LogFile Sequence Number	0008	8	000000013C834EBA	H
Sequence number	0010	2	1	I
Hard link count	0012	2	1	I
Offset to first Attribute	0014	2	0030	H
Flags	0016	2	1	I
Real size of the FILE record	0018	4	440	I
Allocated size of the FILE record	001C	4	1024	I
File reference to the base FILE record	0020	8	0000000000000000	H
Next Attribute Id	0028	2	5	I
Update Sequence	002A	6		
Number	002A	2	00A7	H
Array	002C	4		
<ATTRIBUTES LIST>	0030	388		
Attribute [$00000004]	0030	72		
<STANDART ATTRIBUTE HEA...	0030	24		
Attribute Type	0030	4	00000004	H
Attribute Length (including thi...	0034	4	72	I
Non-resident flag	0038	1	0	I
Attribute Name length	0039	1	0	I
Offset to the Attribute Name	003A	2	00E7	H
Flags	003C	2	0000	H
Attribute Id	003E	2	0000	H
Length of the Attribute	0040	4	48	I
Offset to the Attribute	0044	2	0018	H
Indexed flag	0046	1	00	H
Padding	0047	1	00	H
<DATA RESIDENT>	0048	48		
Attribute [$00000030]	0078	112	IQWOR偰8.OLB	
Attribute [$00000050]	00E8	120		
Attribute [$00000080]	0160	80		
<END OF ATTRIBUTES>	01B0	4	FFFFFFFF	H
<END OF RECORD>	01B4	4	00000000	H
<UNUSED SPACE>	01B8	584		

FixUp OK | Wrong value Attribute Type - may be $0010

Переходим к следующей по порядку ошибке (недопустимый тип атрибута), это первый атрибут из списка атрибутов.

Во-первых, тип атрибута должен быть в списке $AttrDef:

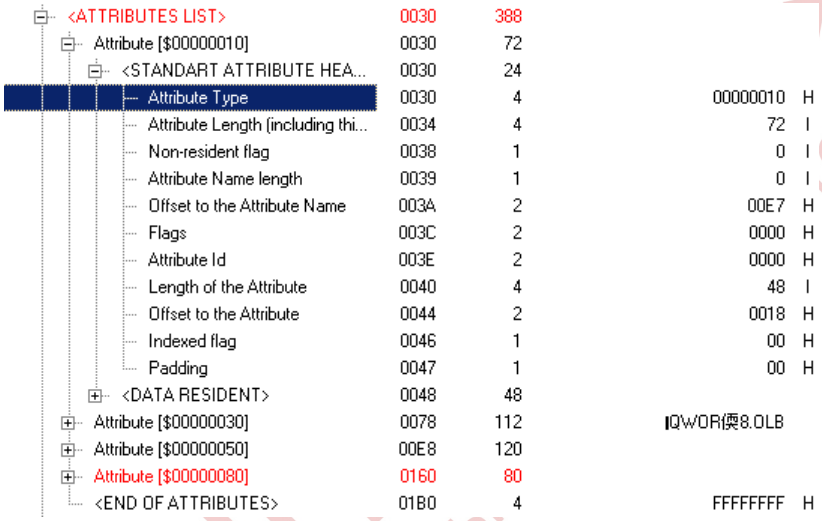
Тип	ОС	Имя
0x10		$STANDARD_INFORMATION
0x20		$ATTRIBUTE_LIST
0x30		$FILE_NAME
0x40	NT	$VOLUME_VERSION
0x40	2K	$OBJECT_ID
0x50		$SECURITY_DESCRIPTOR
0x60		$VOLUME_NAME
0x70		$VOLUME_INFORMATION
0x80		$DATA
0x90		$INDEX_ROOT
0xA0		$INDEX_ALLOCATION
0xB0		$BITMAP
0xC0	NT	$SYMBOLIC_LINK
0xC0	2K	$REPARSE_POINT

0xD0		$EA_INFORMATION
0xE0		$EA
0xF0	NT	$PROPERTY_SET
0x100	2K	$LOGGED_UTILITY_STREAM

Во-вторых, если *<HEADER> → File reference to the base FILE record* равен нулю (это – базовая MFT запись), то тип первого атрибут в списке однозначно должен быть $00000010.

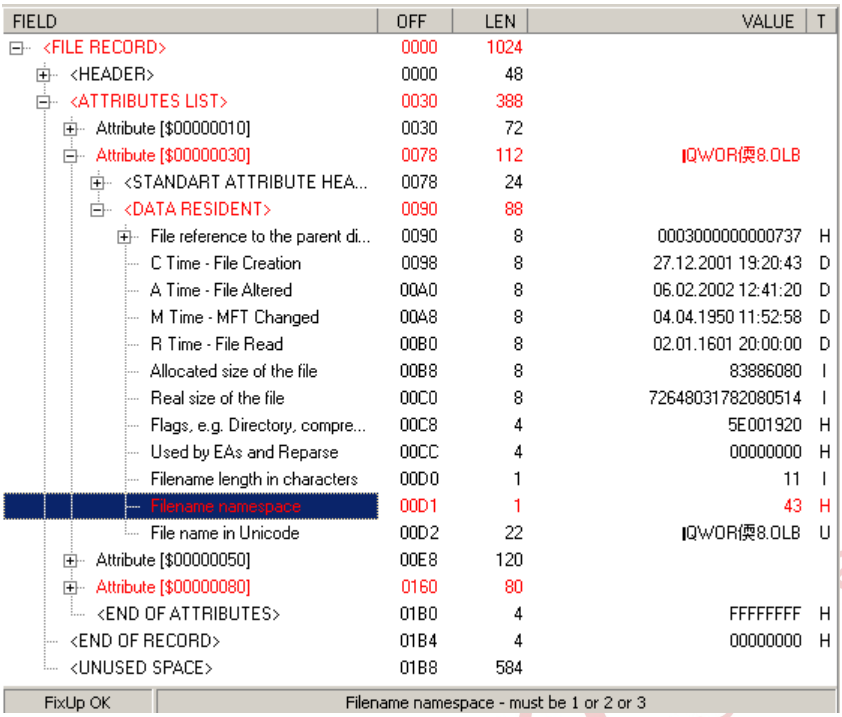
Конечно, в случае двойной ошибки, как в *<HEADER> → File reference to the base FILE record* так и в *Attribute → <STANDART ATTRIBUTE HEADER> → Attribute Type*, ситуация осложняется.

Редактируем *Attribute Type* и убеждаемся в отсутствии ошибок.



Далее - *Attribute[$00000030] → <DATA RESIDENT> → Filename namespace*. В этом поле используется два первых бита. Установленный бит 0 означает, что имя файла является длинным именем, установленный бит 1 - имя файла является коротким именем. Длина имени файла *Attribute[$00000030] → <DATA RESIDENT> → Filename length in characters* = 11, длина расширения - OLB – три символа. На основании этого делаем вывод, что *Attribute[$00000030] → <DATA RESIDENT> → Filename* namespace может принимать любое - 1,2,3 – значение.

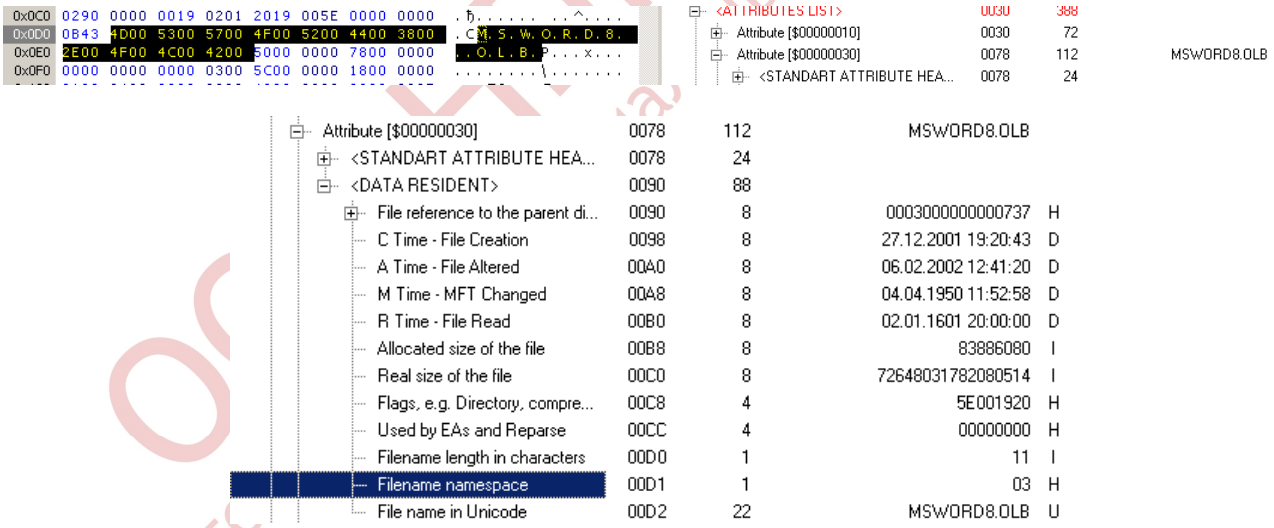
Редактируем *Filename namespace* и убеждаемся в отсутствии ошибок.



FIELD	OFF	LEN	VALUE	T
<FILE RECORD>	0000	1024		
<HEADER>	0000	48		
<ATTRIBUTES LIST>	0030	388		
Attribute [$00000010]	0030	72		
Attribute [$00000030]	0078	112	IQWOR俾8.OLB	
<STANDART ATTRIBUTE HEA...	0078	24		
<DATA RESIDENT>	0090	88		
File reference to the parent di...	0090	8	0003000000000737	H
C Time - File Creation	0098	8	27.12.2001 19:20:43	D
A Time - File Altered	00A0	8	06.02.2002 12:41:20	D
M Time - MFT Changed	00A8	8	04.04.1950 11:52:58	D
R Time - File Read	00B0	8	02.01.1601 20:00:00	D
Allocated size of the file	00B8	8	83886080	I
Real size of the file	00C0	8	72648031782080514	I
Flags, e.g. Directory, compre...	00C8	4	5E001920	H
Used by EAs and Reparse	00CC	4	00000000	H
Filename length in characters	00D0	1	11	I
Filename namespace	00D1	1	43	H
File name in Unicode	00D2	22	IQWOR俾8.OLB	U
Attribute [$00000050]	00E8	120		
Attribute [$00000080]	0160	80		
<END OF ATTRIBUTES>	01B0	4	FFFFFFFF	H
<END OF RECORD>	01B4	4	00000000	H
<UNUSED SPACE>	01B8	584		

FixUp OK | Filename namespace - must be 1 or 2 or 3

Переходим к полю *Attribute[$00000030]* → *<DATA RESIDENT>* → *File Name*. Это поле содержит символы в различной кодировке (различных кодовых таблиц). Редактируем поле, используя HEX-редактор. Стоит отметить, что имя файла не влияет на корректность пользовательских данных записи MFT, основное требование – имя файла должно быть корректным с точки зрения Вашей операционной системы.



Attribute [$00000030]	0078	112	MSWORD8.OLB	
<STANDART ATTRIBUTE HEA...	0078	24		
<DATA RESIDENT>	0090	88		
File reference to the parent di...	0090	8	0003000000000737	H
C Time - File Creation	0098	8	27.12.2001 19:20:43	D
A Time - File Altered	00A0	8	06.02.2002 12:41:20	D
M Time - MFT Changed	00A8	8	04.04.1950 11:52:58	D
R Time - File Read	00B0	8	02.01.1601 20:00:00	D
Allocated size of the file	00B8	8	83886080	I
Real size of the file	00C0	8	72648031782080514	I
Flags, e.g. Directory, compre...	00C8	4	5E001920	H
Used by EAs and Reparse	00CC	4	00000000	H
Filename length in characters	00D0	1	11	I
Filename namespace	00D1	1	03	H
File name in Unicode	00D2	22	MSWORD8.OLB	U

Далее *Attribute[$00000080]*. Это нерезидентный атрибут, имеет *<DATA RUNS>*.

В качестве подсказки каждый *Run* в колонке *VALUE* прокомментирован тремя полями:

1) NNNNN [SSSSS - LLLLL],

где NNNNN – количество кластеров в цепочке (хранится в каждой цепочке);

2) SSSSS – *Starting VCN* цепочки (вычисляется но основании *Last VCN* предыдущей цепочки),

3) LLLLL – *Last VCN* цепочки = SSSSS + NNNNN – 1.

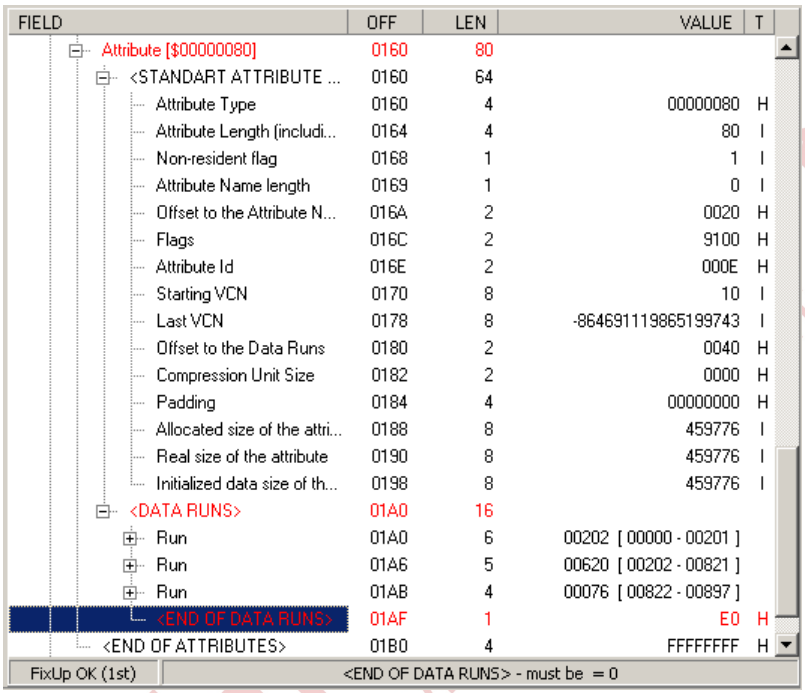
Соответственно, *Last VCN* последней цепочки есть *Last VCN* данных атрибута с точки зрения *<DATA RUNS>*.

Корректность нерезидентного атрибута можно контролировать по взаимосвязи определенных полей *<STANDART ATTRIBUTE HEADER>* и *<DATA RUNS>*.

Этими полями являются в *<STANDART ATTRIBUTE HEADER>* - *Starting VCN* и *Last VCN*, в *<DATA RUNS>* - *LastVCN* последней цепочки.

У корректного атрибута значения *LastVCN* *<STANDART ATTRIBUTE HEADER>* и *<DATA RUNS>* совпадают.

Попытаемся определить причину несовпадения.



FIELD	OFF	LEN	VALUE	T
Attribute [$00000080]	0160	80		
<STANDART ATTRIBUTE ...	0160	64		
Attribute Type	0160	4	00000080	H
Attribute Length (includi...	0164	4	80	I
Non-resident flag	0168	1	1	I
Attribute Name length	0169	1	0	I
Offset to the Attribute N...	016A	2	0020	H
Flags	016C	2	9100	H
Attribute Id	016E	2	000E	H
Starting VCN	0170	8	10	I
Last VCN	0178	8	-8646911119865199743	I
Offset to the Data Runs	0180	2	0040	H
Compression Unit Size	0182	2	0000	H
Padding	0184	4	00000000	H
Allocated size of the attri...	0188	8	459776	I
Real size of the attribute	0190	8	459776	I
Initialized data size of th...	0198	8	459776	I
<DATA RUNS>	01A0	16		
Run	01A0	6	00202 [00000 - 00201]	
Run	01A6	5	00620 [00202 - 00821]	
Run	01AB	4	00076 [00822 - 00897]	
<END OF DATA RUNS>	01AF	1	E0	H
<END OF ATTRIBUTES>	01B0	4	FFFFFFFF	H

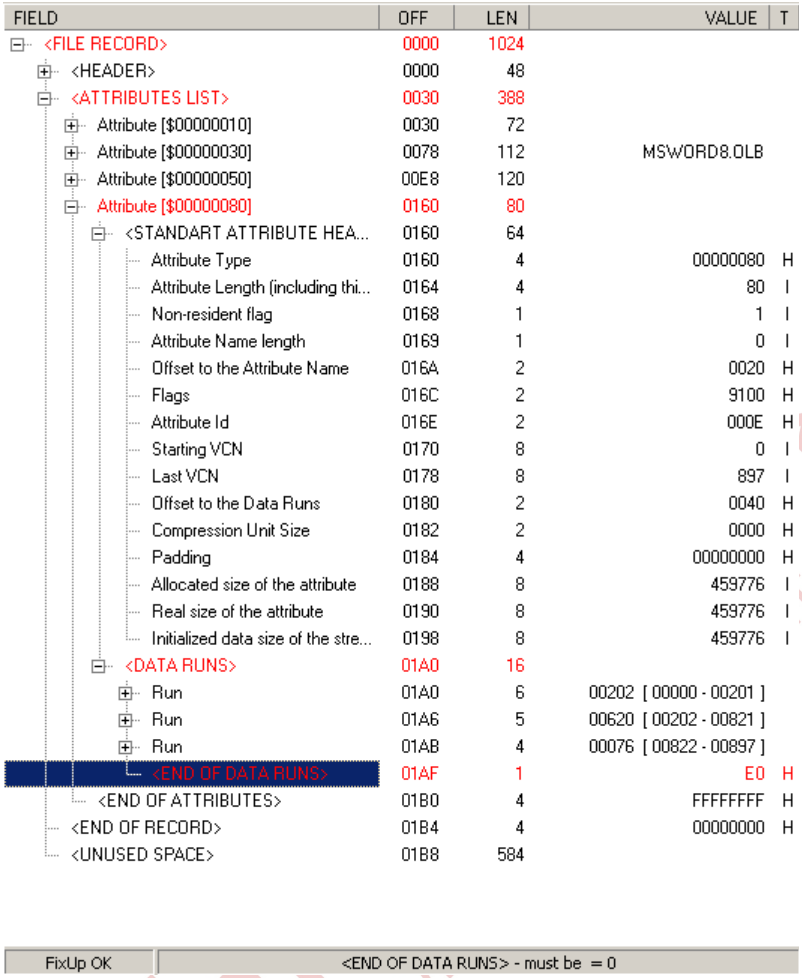
FixUp OK (1st) | <END OF DATA RUNS> - must be = 0

<STANDART ATTRIBUTE HEADER> → *Last VCN* - значение явно ошибочно. Во-первых – отрицательно, во-вторых – слишком огромное расхождение с *<DATA RUNS>* → *Last VCN*. Это позволяет сделать предположение, что запорчены старшие байты 8-ми байтового поля. Поэтому редактировать поле будем после перевода его в шестнадцатеричное представление, используя соответствующий метод.

Starting VCN	0170	8	10	I
Last VCN	0178	8	F400000200000381	H
Offset to the Data Runs	0180	2	0040	H

Исправив пятый и восьмой байт на $0, убеждаемся в правоте – значения *Last VCN* у *<STANDART ATTRIBUTE HEADER>* и *<DATA RUNS>* совпадают. Это также говорит о том, что поля, отвечающие за хранение количества кластеров в каждой цепочке *<DATA RUNS>* → *Run* → *Clusters length*, имеют корректные данные – маловероятно, чтобы ошибки в двух и более цепочках скомпенсировали друг друга, и итоговая сумма кластеров осталась неизменной.

<STANDART ATTRIBUTE HEADER> → *Starting VCN* должно быть равно 0 - если запись базовая. Редактируем поле.



FIELD	OFF	LEN	VALUE	T
<FILE RECORD>	0000	1024		
<HEADER>	0000	48		
<ATTRIBUTES LIST>	0030	388		
Attribute [$00000010]	0030	72		
Attribute [$00000030]	0078	112	MSWORD8.OLB	
Attribute [$00000050]	00E8	120		
Attribute [$00000080]	0160	80		
<STANDART ATTRIBUTE HEA...	0160	64		
Attribute Type	0160	4	00000080	H
Attribute Length (including thi...	0164	4	80	I
Non-resident flag	0168	1	1	I
Attribute Name length	0169	1	0	I
Offset to the Attribute Name	016A	2	0020	H
Flags	016C	2	9100	H
Attribute Id	016E	2	000E	H
Starting VCN	0170	8	0	I
Last VCN	0178	8	897	I
Offset to the Data Runs	0180	2	0040	H
Compression Unit Size	0182	2	0000	H
Padding	0184	4	00000000	H
Allocated size of the attribute	0188	8	459776	I
Real size of the attribute	0190	8	459776	I
Initialized data size of the stre...	0198	8	459776	I
<DATA RUNS>	01A0	16		
Run	01A0	6	00202 [00000 - 00201]	
Run	01A6	5	00620 [00202 - 00821]	
Run	01AB	4	00076 [00822 - 00897]	
<END OF DATA RUNS>	01AF	1	E0	H
<END OF ATTRIBUTES>	01B0	4	FFFFFFFF	H
<END OF RECORD>	01B4	4	00000000	H
<UNUSED SPACE>	01B8	584		

FixUp OK	<END OF DATA RUNS> - must be = 0

По определению цепочки <*DATA RUNS*> должны закончиться маркером конца данных - байтом со значением 0.

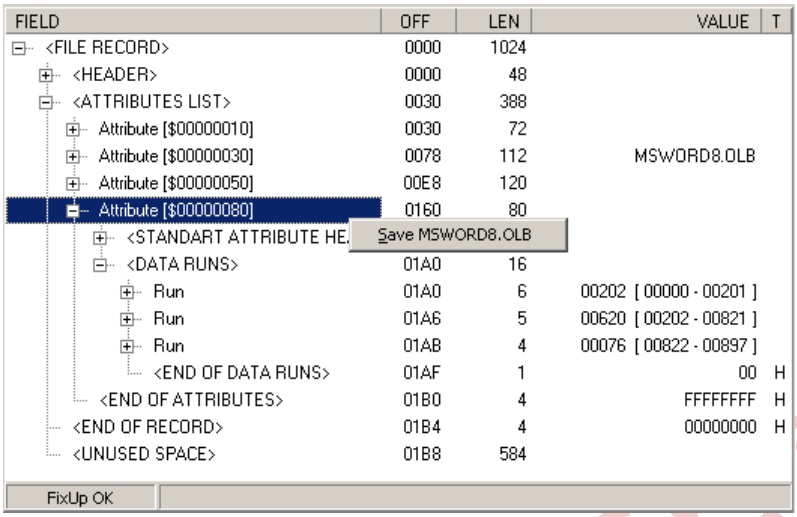
Редактируем поле <*END OF DATA RUNS*>, исправив значение на 0.

Сохраняем результат нашей работы на диск задачи, воспользовавшись соответствующей кнопкой на панели инструментов.

Дополнительно можно вычислить размер кластера, разделив *Allocated size of the attribute* на *LastVCN*+1. В нашем случае получаем 459776 / 898 = 512, *Cluster Size* = 1 sector. Размер кластера в секторах должен быть 2^N - [1,2,4,8,16,32,64,128].

Теперь можно открыть карту <*DATA RUNS*> атрибута $80 для контроля пользовательских данных. И если поле <*DATA RUNS*> → *Run* → *Clusters offset* первой цепочки <*DATA RUNS*> не было повреждено, то с большой вероятностью мы увидим реальные данные этой MFT записи.

Теперь можно сохранить пользовательские данные MFT записи, воспользовавшись методом сохранения атрибута в файл.



2 Справочная информация

2.1 Главная загрузочная запись MBR и таблица разделов

Информация о структуре диска - *таблица разделов* (partition table) – хранится в *главной загрузочной записи MBR* (Master Boot Record). *MBR* находится по адресу – цилиндр 0, головка 0, сектор 1 (в режиме LBA адресации это нулевой сектор). В начале этого сектора расположена программа *главного загрузчика* (*master boot*), а за ней находится таблица разделов, содержащая четыре описателя разделов. Каждый описатель задает границы разделов в двух системах: CHS и LBA. Помимо границ разделов, описатель задает и атрибуты раздела – системный код и флаг активности. *Флаг активности* указывает главному загрузчику, какой раздел ему следует загружать. Флаг активности может быть установлен только для одного раздела диска (или ни у одного). *Системный код* определяет тип раздела; операционная система для своей файловой системы может использовать разделы только известных ей типов.

Структура *MBR* следующая:

– байты *000h – 1BDh* – код загрузки *Boot*-сектора активного раздела;

– байты *1BEh-1CDh, 1CEh-1DDh, 1DEh-1EDh, 1EEh-1FDh* – описатели разделов (*Partition Entry*), 16-байтные структуры (см. таблицу 1);

– байты *1FE = 55h, 1FF = AAh* – сигнатура *MBR* (слово *AA55h*).

Таблица 1 Структура описателя раздела

Смещение	Длина, байт	Назначение
00h	1	*Boot Flag* – флаг активности раздела: *80h* – активный, *00h* – нет
01h	1	*Begin Head* – номер начальной головки
02h	2	*Begin SelCyl* – номер начального сектора и цилиндра
04h	1	*System Code* – системный код (см. таблицу 3)
05h	1	*Ending Head* – номер конечной головки
06h	2	*Ending SelCyl* – номер конечного сектора и цилиндра
08h	4	*Starting Sector (Relative Sector)* – номер начального сектора раздела
0Ch	4	*Num Sectors* – количество секторов в разделе

Таблица разделов может заполняться как с начала, так и с конца. Если разделов меньше четырех, то свободные описатели обнулены. Свободные описатели, как и используемые, могут располагаться в любом месте таблицы.

Номера сектора и цилиндра хранятся в двух байтах (*Begin SelCyl* и *Ending SelCyl*). Биты 0-7 номера цилиндра сохранены во втором байте, в то время как биты 8-9 сохранены в старших разрядах первого байта. Значение сектора сохранено в битах 0-5 первого байта.

Таблица 2 Бинарная карта *Begin SelCyl* (*Ending SelCyl*)

2-й байт								**1-й байт**							
F	**E**	**D**	**C**	**B**	**A**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Номер цилиндра										Номер сектора					
7	6	5	4	3	2	1	0	9	8	5	4	3	2	1	0

Когда диск работает в режиме LBA (логической адресации блоков), значения CHS (цилиндр-головка-сектор) для начала и конца размещения игнорируются. Отображение диска выполняется с абсолютным номером сектора, а не в терминах CHS (cylinder-head-sector). Таким образом, относительное значение сектора и длина раздела используются для идентификации объема раздела на диске. CHS значения вообще недопустимы для дисков объемом более чем 8.4GB.

Структура раздела зависит от его типа. Часть системных кодов приведена в таблице 3.

Таблица 3 Коды и типы разделов жесткого диска

Код	Раздел; ОС, с которой введен; Файловая система; Объем
00	Неизвестный (неформатированный) раздел;
01	DOS FAT12; MS-DOS 2.0; до 15 МБайт;
02	XENIX root
03	XENIX usr
04	DOS FAT16; MS-DOS 3.0; до 32 МБайт;
05	(DOS) Extended; MS-DOS 3.3; до 2 ГБайт;
06	DOS FAT16 (Big DOS); MS-DOS 4.0; до 2 ГБайт;
07	OS/2 HPFS или Windows NT NTFS;
08	AIX;
09	AIX bootable;
0A	OS/2 Boot Manage
0B	Win95 FAT32; Windows 95 OSR2; 512 МБайт – 2 ТБайт;
0C	Win95 FAT32 (LBA); Windows 95 OSR2; 512 МБайт – 2 ТБайт;
0E	Win95 FAT16 (LBA); Windows 95 OSR2; 32 МБайт – 2 ГБайт;
0F	(Win95) Extended (LBA); Windows 95 OSR2;
11	Hidden DOS FAT12
14	Hidden DOS FAT16
16	Hidden DOS FAT16
40	Venix 80286
52	Microport
63	GNU HURD
64	Novell Netware 2
65	Novell Netware 3
75	PC/IX
80	Old MINIX
81	Linux/MINIX
82	Linux Swap
83	Linux
85	Linux extended
93	Amoeba
94	Amoeba BBT
A5	BSD/386
B7	BSDI fs
B8	BSDI swap
C7	Syrinx
DB	CP/M
E1	DOS access
E3	DOS R/O
F2	DOS secondary
FF	BBT

Разделы с кодами (01, 04, 06, 0B, 0C, 0E) являются *первичными разделами* (primary partition) DOS/Windows. Первичных разделов может быть несколько (хотя старые утилиты FDISK из MS DOS и Windows 9x позволяет создавать не более одного первичного раздела). Первичный раздел содержит один *логический диск* (logical drive). В первом секторе логического диска (boot sector) находится загрузчик и описатель типа файловой системы и структура диска.

На рисунке ниже показан пример главной загрузочной записи MBR в двоичном редакторе для накопителя с двумя первичными разделами первый из которых является активным. Описатели третьего и четвертого раздела обнулены.

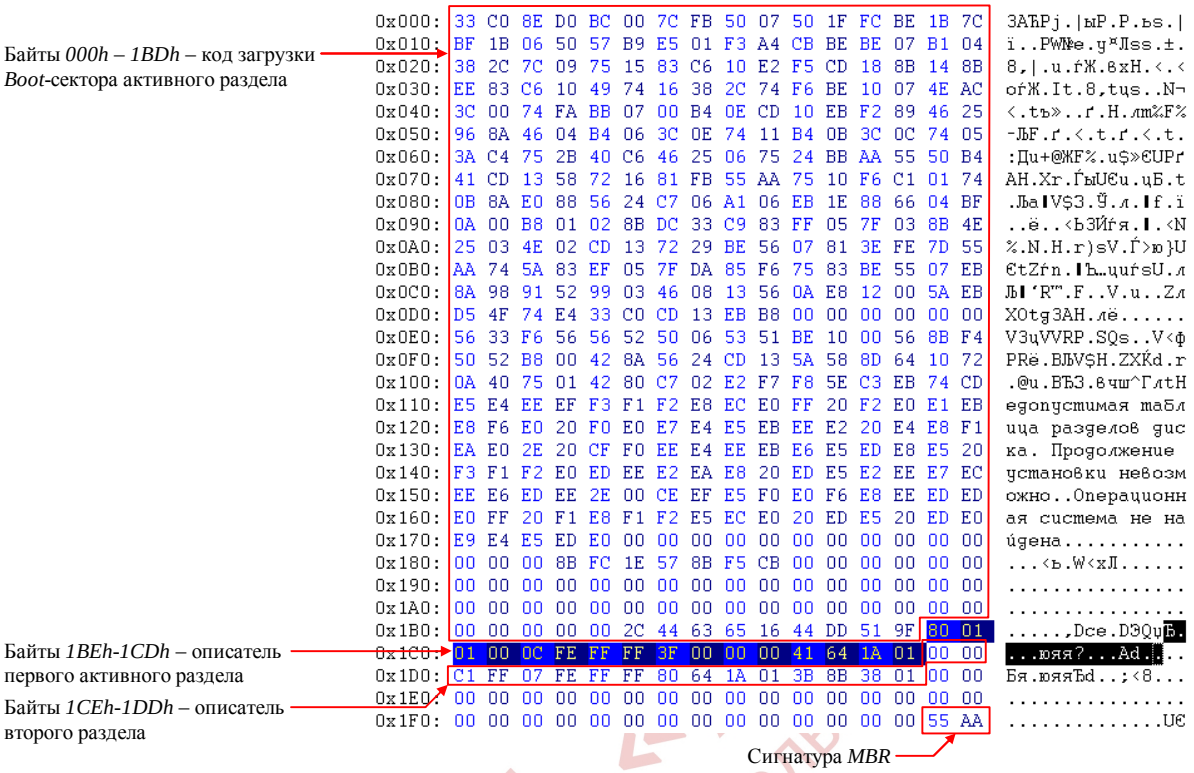


Рисунок 81 Пример главной загрузочной записи MBR

Ниже приводится расшифровка описателя активного раздела.

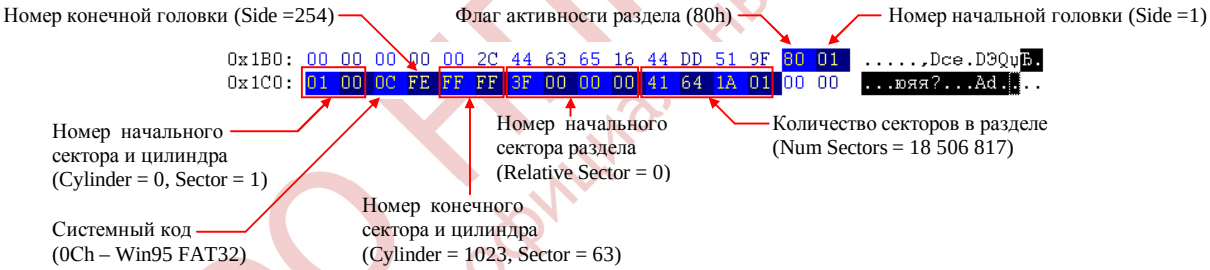
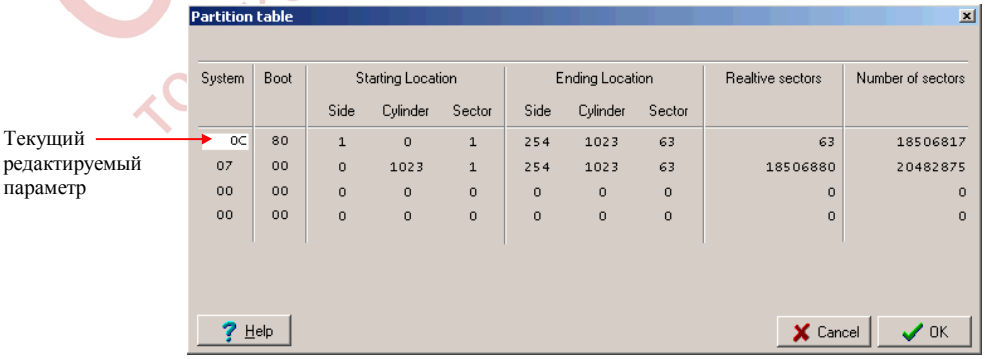


Рисунок 82 Пример описателя активного раздела

Примечание (использование "PC-3000 Flash"). Для удобства просмотра и редактирования таблицы разделов в программе "PC-3000 Flash" существует форма, показанная на рисунке ниже.



System	Boot	Starting Location			Ending Location			Realtive sectors	Number of sectors
		Side	Cylinder	Sector	Side	Cylinder	Sector		
0C	80	1	0	1	254	1023	63	63	18506817
07	00	0	1023	1	254	1023	63	18506880	20482875
00	00	0	0	0	0	0	0	0	0
00	00	0	0	0	0	0	0	0	0

Рисунок 83 Просмотр описателей накопителя в режиме "Partition table"

Данная форма может быть вызвана из контекстного меню объекта MBR в режиме *Проводник* (пункт *Свойства*) или с помощью двоичного редактора и режима *Просмотр как…→ Partition table*. Все значения таблицы могут редактироваться, текущий редактируемый параметр выделяется с помощью

светлого фона. Значения в таблице разделов приводятся в десятеричной системе счисления за исключением столбцов *System* и *Boot*, значения которых шестнадцатеричные.

Расширенный раздел (extended partition, код *05* или *0F*) служит для организации произвольного количества логических дисков. Первый сектор расширенного раздела аналогичен *MBR* (но загрузчик отсутствует) и содержит *расширенную таблицу разделов EPR* (Extended Partition Record), той же структуры, но с некоторыми оговорками. Первый описатель задает *вторичный* (secondary) раздел, отведенный под очередной логический диск. В нем указывается код раздела с файловой системой, координаты начала и конца раздела (трехмерные и линейные). Если этот логический диск занимает не весь объем расширенного раздела, то второй описатель тоже имеет код 05 или 0F и указывает на положение сектора со следующей расширенной таблицей разделов. Остальные описатели не используются (их коды нулевые). Если свободного места в разделе уже нет, то и второй описатель не используется. В следующей расширенной таблице разделов действуют те же правила. Эта цепочка заканчивается на расширенной таблице, у которой во втором описателе стоит нулевой код раздела. Второй описатель в расширенных таблицах может только указывать на положение следующей расширенной таблицы. Часть пространства расширенного раздела может оставаться не распределенной. Цепочка расширенных таблиц разделов должна быть непрерывной, неветвящейся (используются только два описателя, и только второй может указывать на следующую таблицу) и незацикленной (второй описатель не должен ссылаться на ту же таблицу или предыдущую в цепочке).

По расположению на физическом диске расширенные разделы являются вложенными друг в друга; все они располагаются в области, описанной в главной таблице разделов как расширенный раздел. В главной таблице может быть описан лишь один расширенный режим.

Если расширенные разделы имеют код 0Fh, то линейные адреса всех элементов таблиц будут указываться относительно начала физического диска.

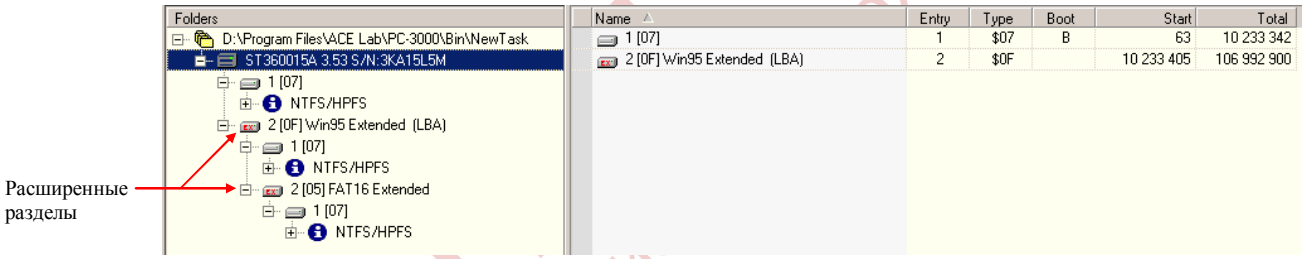


Рисунок 84 Отображение расширенных разделов в режиме "Проводник"

После четырех описателей в *MBR*, расположена сигнатура (два байта) *AA55h*, которая метит сектор как системный сектор. Это значение существует в каждом секторе структуры разделов (включая загрузочные секторы) и его отсутствие может свидетельствовать о вероятном повреждении структуры разделов.

Для главного загрузчика загрузить активный раздел означает найти положение первого (загрузочного) сектора этого раздела, загрузить сектор в память и передать управление на его начало. Дальнейшие действия выполняются загрузчиком активного раздела. Перед загрузкой активного раздела главный загрузчик проверяет, является ли найденный раздел единственным активным, и если нет, то останавливается с сообщением *Invalid partition table*. Если активный раздел не найден, главный загрузчик останавливается с сообщением *Missing operating System*. При загрузке сектора с *MBR* вызывается функция чтения одного сектора и если считать сектор без ошибок не удается, главный загрузчик останавливается с сообщением *Error loading operating system*.

2.2 Логический диск с системой FAT

2.2.1 Логический диск с системой FAT12/16

Файловая система FAT (File Allocation Table) была разработана Биллом Гейтсом и Марком МакДональдом в 1977 году и первоначально использовалась в операционной системе 86-DOS. Чтобы добиться переносимости программ из операционной системы CP/M в 86-DOS, в ней были сохранены ранее принятые ограничения на имена файлов. В дальнейшем 86-DOS была приобретена Microsoft и стала основой для ОС MS-DOS 1.0, выпущенной в августе 1981 года. FAT была предназначена для работы с гибкими дисками размером менее 1 Мб, и вначале не предусматривала поддержки жестких дисков. В настоящее время FAT поддерживает файлы и разделы размеров до 2 Гб.

Структура раздела FAT12/16 представлена в таблице ниже.

Таблица 4 Структура раздела FAT12/16

Системная область				Область данных			
Загрузочный сектор (boot sector)	FAT	FAT (копия)	Корневой каталог (root)	Кластер 2	Кластер 3	…	Кластер N

Примечание (использование программы "PC-3000 Flash"). Структуру любого раздела можно просмотреть с помощью режима *Карта раздела*, доступного из контекстного меню выбранного раздела в режиме *Проводник*. Данный режим, может быть использован:

– для проверки и коррекции положения важных с точки зрения восстановления метаданных раздела при восстановлении транслятора (копий загрузочного сектора, копий таблицы FAT, начала таблицы MFT и т.д.);

– для проверки местоположения и целостности основных метаданных раздела в случае логических разрушений (быстрый доступ к копиям загрузочного сектора, к таблицам FAT);

– в случае работы с неисправным накопителем можно выбрать наиболее важные с точки зрения цепочки (например, таблицы FAT, boot-сектора) и вычитать их в копию с наиболее жёсткими параметрами (увеличенное количество попыток чтения), оценить качество вычитывания.

Внешний вид режима *Карта раздела* приводится ниже.

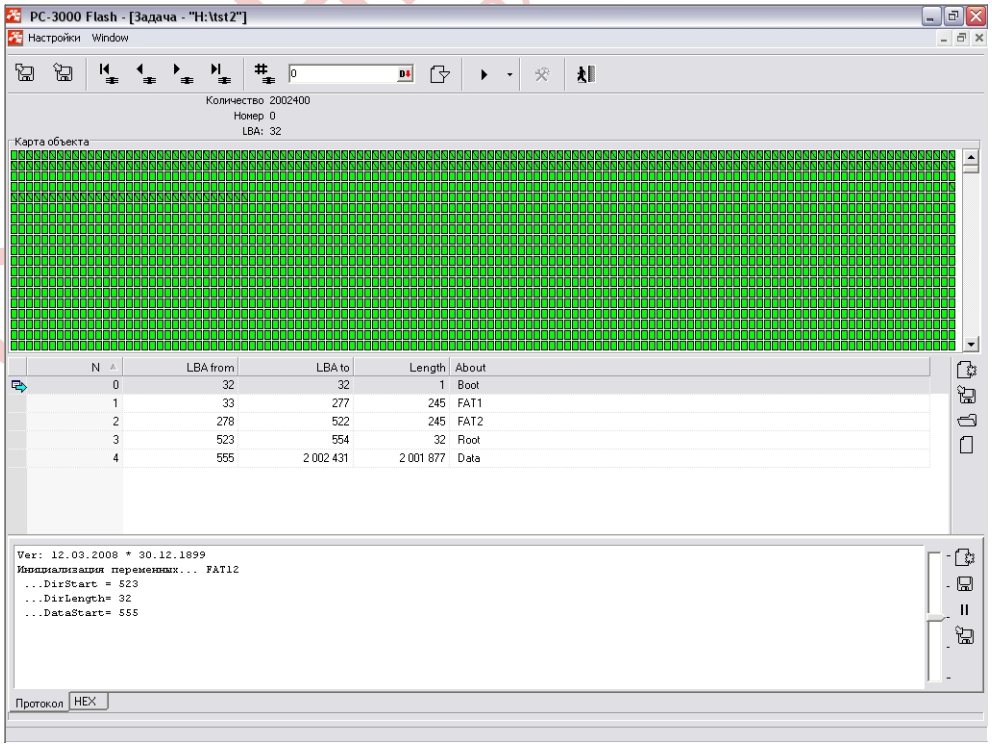


Рисунок 85 Режим "Карта раздела" для раздела FAT16

Сектор 0, называемый также загрузочным, содержит таблицу параметров диска и начальный загрузчик операционной системы. Первые 3 байта сектора 0 содержат команду перехода JMP на начало загрузчика: либо

байт 0E9h и 1 байт короткого смещения, за которым следует команда NOP (код 90h), либо байт 0EBh и два байта длинного смещения. Длинное смещение используется, если загрузчик располагается в зарезервированных секторах. *Загрузчик* является короткой программой, загружающей операционную систему или только ее ядро; также он может являться и средством для выбора загружаемой ОС (boot manager). В отличие от главного загрузчика, этот загрузчик привязан к своей ОС и записывается на диск при форматировании данного диска средствами этой ОС.

Далее расположено поле из 8 бит, в которое при форматировании заносится идентификатор версии ОС. Это текстовая строка, имеющая значение MSWIN4.1 для ОС Windows.

Далее располагается таблица BPB (BIOS parameter block), описывающая физические характеристики диска и позволяющая вычислить правильный физический адрес на диске по данному номеру логического сектора. За таблицей располагаются дополнительные данные.

За сектором 0 могут быть расположены дополнительные зарезервированные (для загрузчика) секторы. Общее число зарезервированных секторов, включая сектор 0, указывается в BPB.

За зарезервированными секторами (или непосредственно за сектором 0) располагается таблица размещения файлов (FAT), после которой могут располагаться дополнительные копии FAT. Обычно используется 2 копии. Число копий указано в BPB.

В конце системной области расположен корневой каталог, который имеет фиксированный размер. Размер корневого каталога в каталожных записях также указан в таблице BPB.

Таблица 5 – Структура загрузочного сектора

Смещение	Длина	Назначение
0h	3	Команда JMP на начало кода загрузчика
3h	8	Название ОС, например "MSDOS6.0"
0Bh	2	*BytesPerSector* – количество байтов в секторе, обычно 512 (200h)
0Dh	1	*SectorsPerCluster* – количество секторов в кластере
0Eh	2	*ReservedSectors* – количество секторов, занятых загрузчиком и зарезервированных
10h	1	*NumberOfFATs* – количество копий FAT
11h	2	*RootEntries* – максимальное число 32-байтных элементов корневого каталога
13h	2	*TotalSectors* – общее число секторов на томе. 0000 означает, что диск больше 32 Мбайт и число задается двойным словом BigTotSects (смещение 20h)
15h	1	*MediaDescriptor* - дескриптор носителя (то же, что и в первом байте FAT)
16h	2	*SectorsPerFAT* – число секторов в одной FAT
18h	2	*SectorsPerTrack* – число секторов на треке
1Ah	2	*Heads* – число головок
1Ch	4	*HiddenSectors* – число скрытых секторов
20h	4	*BigTotalSectors* – число секторов (для разделов > 32 Мбайт)
24h	1	*PhysicalDiskNumber* – логический номер устройства, номер присваивается в процессе форматирования (80h – первый жесткий диск)
25h	1	*CurrentHead* - зарезервирован
26h	1	*Signature* – сигнатура расширенного загрузчика (29h)
27h	4	*VolumeSerialNumber* – серийный номер тома (устанавливается при форматировании)
2Bh	11	*VolumeLabel* – метка тома (строка символов)
36h	8	*SystemID* – символьный идентификатор файловой системы (например, "FAT16")
3Eh	448	Область кода загрузчика
1FEh	2	*BootSignature* - Сигнатура 55AAh (конец загрузочного сектора)

Замечание! Наименование параметров на английском языке в данной таблице соответствует обозначениям, используемым в окне просмотра загрузочного сектора в режиме "FAT16 Boot sector" двоичного редактора.

Примечание (использование "PC-3000 Flash"). Для удобства просмотра и редактирования загрузочного сектора раздела с файловой системой FAT12 и FAT16 в программе "PC-3000 Flash" присутствует форма, показанная на рисунке ниже.

Значения в светлых окошках формы значимы для корректного восстановления информации данного раздела. При редактировании данных полей производится проверка корректности введенных

данных. В случае если введенные данные не корректны, название параметра выделяется с помощью желтого фона.

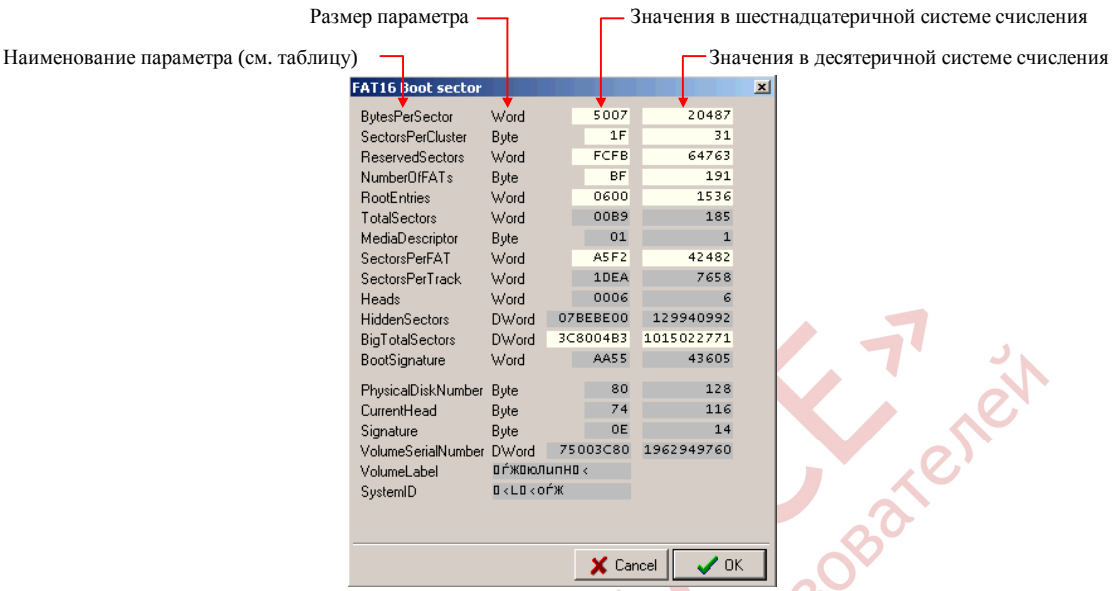


Рисунок 86 Просмотр загрузочного сектора в режиме "FAT16 Boot sector"

Данная форма может быть вызвана из контекстного меню объекта boot-сектор в режиме *Проводник* (метод *Свойства*) или с помощью двоичного редактора и режима *Просмотр как…→ FAT16 Boot sector*.

Файловая система FAT не может контролировать отдельно каждый сектор, поэтому она объединяет смежные сектора области данных в *кластеры* (clusters). Таким образом, уменьшается общее количество единиц хранения, за которыми должна следить файловая система. Каждый кластер имеет свой номер; размер кластера (число секторов) выбирается кратным степени 2, в зависимости от объема диска и размера FAT (см. таблицу 6). Файл на диске занимает целое число кластеров. Это приводит к тому, что часть пространства диска расходуется впустую. Если файл занимает более одного кластера, то все кластеры, занимаемые файлом, организуются в *цепочку кластеров* (cluster chain). Количество файлов на диске не может превышать количества кластеров (элементов FAT).

Таблица 6 Размер кластера FAT

Размер раздела, МБайт	**FAT16 Размер кластера, Кбайт (число секторов)**	**Размер FAT, Кбайт**	**FAT32 Размер кластера по умолчанию, Кбайт (число секторов)**	**Размер FAT, Кбайт**
0 – 15	4 (8), FAT12	0,5 – 6	-	-
16 – 27	2 (4)	16 – 28	-	-
128 – 255	4 (8)	64 – 128	-	-
256 – 511	8 (16)	64 – 128	-	-
512 – 1 023	16 (32)	64 – 128	4 (8)	512 – 1 023
1 024 – 2 145	32 (64)	64 – 128	4 (8)	1 024 – 2 145
2 146 – 8 191	-	-	4 (8)	2 146 – 8 191
8 192 – 16 383	-	-	8 (16)	4 096 – 8 191
16 384 – 32 767	-	-	16 (32)	4 096 – 8 191
32 768 и более	-	-	32 (64)	4 096 и более

Свое название FAT получила от одноименной таблицы размещения файлов. В таблице размещения файлов хранится информация о кластерах логического диска. Каждому кластеру в FAT соответствует отдельная запись – *элемент FAT*, который показывает, свободен ли данный кластер, занят ли данными файла, или помечен как сбойный (испорченный). Если кластер занят под файл, то в соответствующей записи в таблице размещения файлов указывается адрес кластера, содержащего следующую часть файла или код говорящий о

том, что данный кластер последний в цепочке. Из-за этого FAT называют файловой системой со связанными списками. Оригинальная версия FAT, разработанная для DOS 1.00, использовала 12-битную таблицу размещения файлов (т.е один элемент FAT имел размер 1,5 байта) и поддерживала разделы объемом до 16 Мб (в DOS можно создать не более двух разделов FAT). Для поддержки жестких дисков размером более 32 Мб разрядность FAT была увеличена до 16 бит (один элемент FAT - 2 байта), а размер кластера - до 64 секторов (32 Кб). Так как каждому кластеру может быть присвоен уникальный 16-разрядный номер, то FAT поддерживает максимально 2^{16}, или 65536 кластеров на одном томе.

Поскольку загрузочная запись слишком мала для хранения алгоритма поиска системных файлов на диске, то системные файлы должны находиться в определенном месте, чтобы загрузочная запись могла их найти. Фиксированное положение системных файлов в начале области данных накладывает жесткое ограничение на размеры корневого каталога и таблицы размещения файлов. Вследствие этого общее число файлов и подкаталогов в корневом каталоге на диске FAT ограничено 512.

Рассмотрим организацию таблицы размещения файлов на примере FAT16 (размер элемента FAT равен 2 байта). Элемент FAT содержит число (код), которое для FAT16 может иметь одно из следующих значений:

- 0000h – свободный кластер;
- 0002h-FFEFh – номер следующего элемента в цепочке;
- FFF7h – дефектный;
- FFF8h-FFFFh – последний в цепочке.

На рисунке ниже показано начало таблицы FAT жесткого диска. Элементы FAT с номерами 0 и 1 не используются, поскольку FAT начинается с байта *дескриптора носителя* (media descriptor), за которым следует три байта заполнителя (FFh). Дескриптор носителя определяет его тип: F8h – жесткий диск, F0h – дискета 1,44 Мбайт, F9-FFh – дискеты разных форматов. В FAT12 используются только три младшие тетрады (000-FFFh) вышеприведенных кодов, а заполнитель состоит из двух байтов.

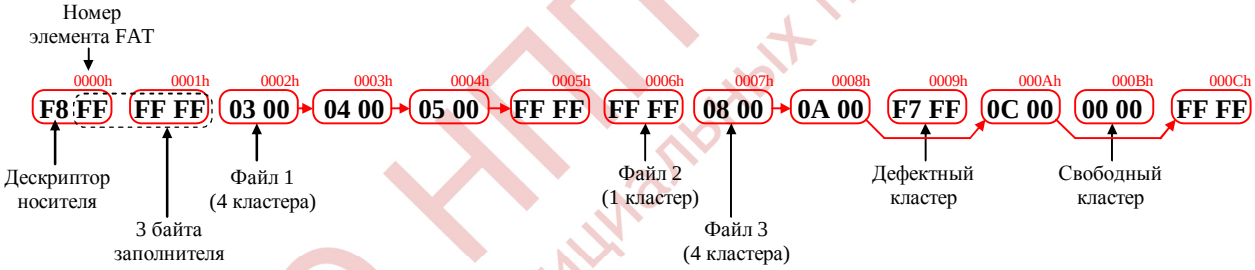


Рисунок 87 Организация цепочек кластеров

Информация о каждом файле располагается в элементе каталога, в который данный файл входит. *Элемент каталога* содержит размер файла (в байтах), имя файла, ссылку на первый кластер цепочки (номер кластера) и некоторую дополнительную информацию (атрибуты). Для того, чтобы прочитать файл целиком, необходимо найти его элемент в каталоге и считать первый кластер, на который ссылается этот элемент, - это будет начало файла. Далее следует прочитать соответствующий ему элемент FAT и определить, является ли он последним. Если он не последний, то он показывает номер следующего кластера цепочки, который также должен быть считан в "хвост" к ранее считанным кластерам. Когда очередь дойдет до последнего кластера в цепочке, следует отсечь лишние данные, если файл кончается не по границе кластера. Отсечение выполняется по длине файла, указанной в элементе каталога.

Кроме свободных, занятых и плохих кластеров на дисках могут образовываться *потерянные кластеры* (lost clusters). Это отдельные кластеры или цепочки, помеченные как занятые, но не принадлежащие ни одному файлу (на них нет ссылки ни из одного элемента каталога).

В FAT применяются следующие соглашения по именам файлов:

– имя должно начинаться с буквы или цифры и может содержать любой символ ASCII, за исключением пробела и символов *"/\[]:;|=,^*?*;

– длина имени не превышает 8 символов, за ним следует точка и необязательное расширение длиной до 3 символов;

– регистр символов в именах файлов не различается и не сохраняется.

Система *каталогов* или *папок* (directory, folder), в общем виде выглядит следующим образом. Сразу после последней копии FAT на диске выделяется фиксированная область *корневого каталога* (root directory), в которую может входить ограниченное число *элементов каталога* (directory entry). Каждый элемент каталога

имеет длину 32 байта и фиксированную структуру (см. таблицу 7). В нем указывается имя файла, номер начального кластера, длина, дата создания и некоторые атрибуты файла и служебные признаки. В каталог могут входить и элементы, указывающие на вложенные каталоги.

Таблица 7 Структура элемента каталога файловой системы FAT

Смещение, байт	Длина, байт	Назначение
00h	8	Имя файла, дополненное справа пробелами (20h) до 8 символов
08h	3	Тип (расширение) файла, дополненное пробелами до 3 символов
0Bh	1	Атрибуты файла: Бит 0 – Read Only (R/O), только чтение Бит 1 – Hidden (H), скрытый Бит 2 – System (Sys), системный Бит 3 – метка тома (Volume Label) Бит 4 – признак каталога (Directory Entry) Бит 5 – Archive (A)
0Ch-15h	10	Резерв
16h	2	Время последнего изменения (создания): Биты 0-4 – пары секунд (0-29) Биты 5-10 –минуты (0-59) Биты 11-15 – минуты (0-23)
18h	2	Дата последнего изменения (создания): Биты 0-4 – день (0-31) Биты 5-10 – месяц (0-59) Биты 0-3 – год, считая с 1980 (0-119)
1Ah	2	Номер начального кластера
1Ch	4	Размер файла в байтах (предельный размер – 4 Гбайт)

Первый символ имени (смещение 0) может иметь специальное значение:
- 00h – элемент каталога никогда не использовался;
- 05h – первый символ имени имеет код E5h;
- 2Eh – псевдоним каталога (точка);
- E5h – элемент удален.

Вложенный каталог для FAT выглядит почти как обычный файл, описываемый соответствующим элементом родительского каталога. В ссылке на каталог (помеченной специальным признаком) поле длины не используется (оно нулевое), а конец каталога определяется по концу цепочки кластеров. Вложенный каталог состоит из набора таких же элементов каталога. Самым первым элементом вложенного каталога является ссылка на себя самого под псевдонимом "." (из нее берется номер начального кластера данного каталога). За ним идет элемент со ссылкой на родительский каталог по псевдониму "..", она позволяет найти начало родительского каталога (нулевой номер кластера указывает на корневой каталог). В то время как корневой каталог имеет фиксированное количество вхождений, на вложенные каталоги такого ограничения не накладывается и число вхождений ограничивается только свободным местом на накопителе.

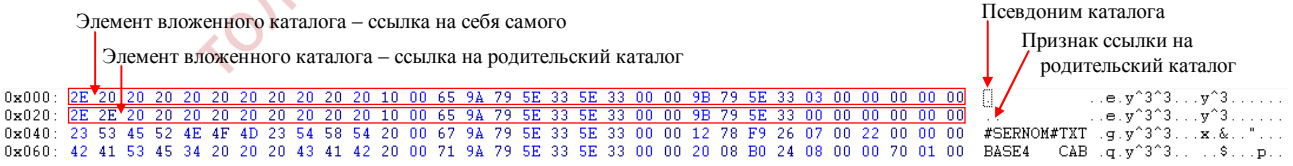


Рисунок 88 Пример вложенного каталога

В ОС Windows 95 ввели поддержку "длинных" имен файлов и каталогов. Эти имена допускают наличие символов разных регистров, наличие точек внутри имени, пробелов и некоторых других специальных символов. В каталоге для каждого файла (ссылки на каталог) отводиться несколько смежных 32-байтных блоков (элементов каталога). В первых нескольких элементах (см. таблицу 8) размещается "длинное" имя с учетом регистра, число блоков зависит от длины имени. Эти элементы имеют специфическое значение байта атрибутов 0Fh (метка тома, системный, скрытый и только чтение), благодаря которому старые ОС их не отображают и не воспринимают при поиске. За ними идет элемент с обычной структурой, в котором записано короткое имя в формате 8.3. Данный элемент каталога содержит номер начального кластера, дату создания и атрибуты файла

(каталога). Длинное имя не существует без короткого. Блоки с частями длинного имени нумеруются по порядку следования символов. Блок с номером 01h расположен непосредственно перед блоком с коротким именем; если его недостаточно, то перед ним будет блок с номером 02h и т.д. У последнего блока номер записывается увеличенным на 40h (если длинное имя размещается всего в одном блоке, то его номер будет 41h).

Таблица 8 Структура элемента каталога с длинным именем

Смещение, байт	Длина, байт	Назначение
00h	1	Порядковый номер
01h	5x2=10	Имя (двухбайтные символы Unicode)
0Bh	1	Атрибуты (0Fh)
0Ch	1	Тип (00h)
0Dh	1	Контрольный код (вычисляется из короткого имени)
0Eh	6x2=12	Имя (продолжение)
1Ah	2	0000h
1Ch	2x2=4	Имя (продолжение)

Ниже приведен пример элемента каталога с длинным именем (System Volume Information), каталог занимает три смежных 32-байтных блока (в скобках указана длина в байтах).

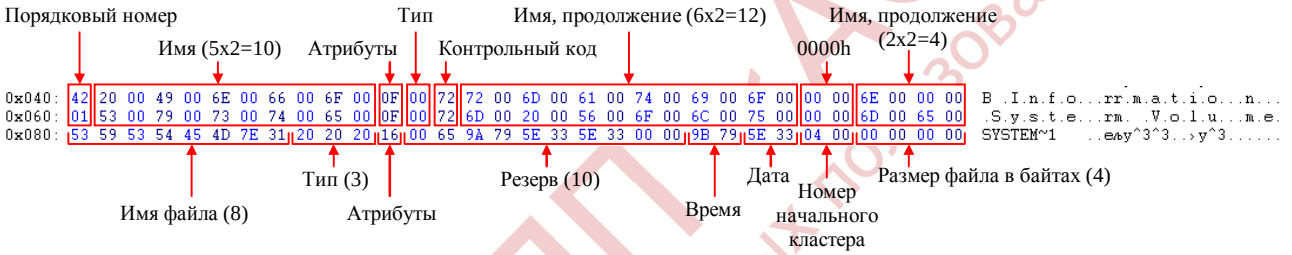


Рисунок 89 Пример каталога с длинным именем

2.2.2 Логический диск с системой FAT32

FAT32 - файловая система производная системы FAT (сначала FAT была 12-разрядной и позволяла работать с дискетами и логическими дисками объемом не более 16 Мбайт; в MS-DOS версии 3.0 таблица FAT стала 16-разрядной для поддержки дисков большей емкости, а для дисков объемом до 2 047 Гбайт используется 32-разрядная таблица FAT). FAT32 поддерживает меньшие размеры кластеров, что позволяет более эффективно использовать дисковое пространство по сравнению с FAT. Эта файловая система используется в операционных системах на DOS основе - Windows 95 OSR2, Windows 98 и Windows Me.

Структура раздела FAT32 представлена в таблице:

Таблица 9 Структура раздела FAT32

Системная область										Область данных			
Загрузчик (3 сектора)				Копия загрузчика									
BPB, BF_BPB, загрузчик (начало)	FSInfo	загрузчик (продолжение)	…	BPB, BF_BPB, загрузчик (начало)	FSInfo	загрузчик (продолжение)	…	FAT	FAT (копия)	Кластер 2	Кластер 3	…	Кластер N

FAT32 имеет расширенный загрузчик. Загрузчик теперь занимает не один, а три физических сектора, причем имеется резервная копия, которая размещается через некоторое количество секторов после основного загрузчика (номер резервной копии загрузочного сектора указывается в BF_BPB и обычно равен 6). Нулевой сектор загрузчика содержит блок параметров BIOS, расширенный блок параметров Big FAT BIOS Parameter Block (BF_BPB) и начало загрузочного кода. Сектор 1 содержит структуру FSInfo, предназначенную для управления FAT. В третьем секторе находится продолжение загрузочного кода. Блок параметров BIOS в FAT32 занимает больше места, поэтому к стандартному BPB блоку добавлен блок, называемый BF_BPB. Он содержит

те же структуры, что и стандартный BPB, но включает несколько дополнительных полей, которые нужны для FAT32. Изменения, внесенные в BPB для поддержки FAT32, описаны в таблице ниже.

Корневой каталог не имеет фиксированного положения и размера, как в FAT16. Вместо этого он является файлом специального вида, как и все другие каталоги. На практике корневой каталог обычно начинается с первого кластера области файлов (кластер 2).

Примечание (использование программы "PC-3000 Flash"). Структуру любого раздела можно просмотреть с помощью режима *"Карта раздела"*, доступного из контекстного меню выбранного раздела в режиме *"Проводник"*. Данный режим, может быть использован:

– для проверки и коррекции положения важных с точки зрения восстановления метаданных раздела при восстановлении транслятора (копий загрузочного сектора, копий таблицы FAT, начала таблицы MFT и т.д.)

– для проверки местоположения и целостности основных метаданных раздела в случае логических разрушений (быстрый доступ к копиям загрузочного сектора, к таблицам FAT)

– в случае работы с неисправным накопителем можно выбрать наиболее важные с точки зрения цепочки (например, таблицы FAT, boot-сектора) и вычитать их в копию с наиболее жёсткими параметрами (увеличенное количество попыток чтения), оценить качество вычитывания.

Подробнее об использовании данного режима см. раздел *"Карта раздела"*. Внешний вид режима *Карта раздела* приводится ниже.

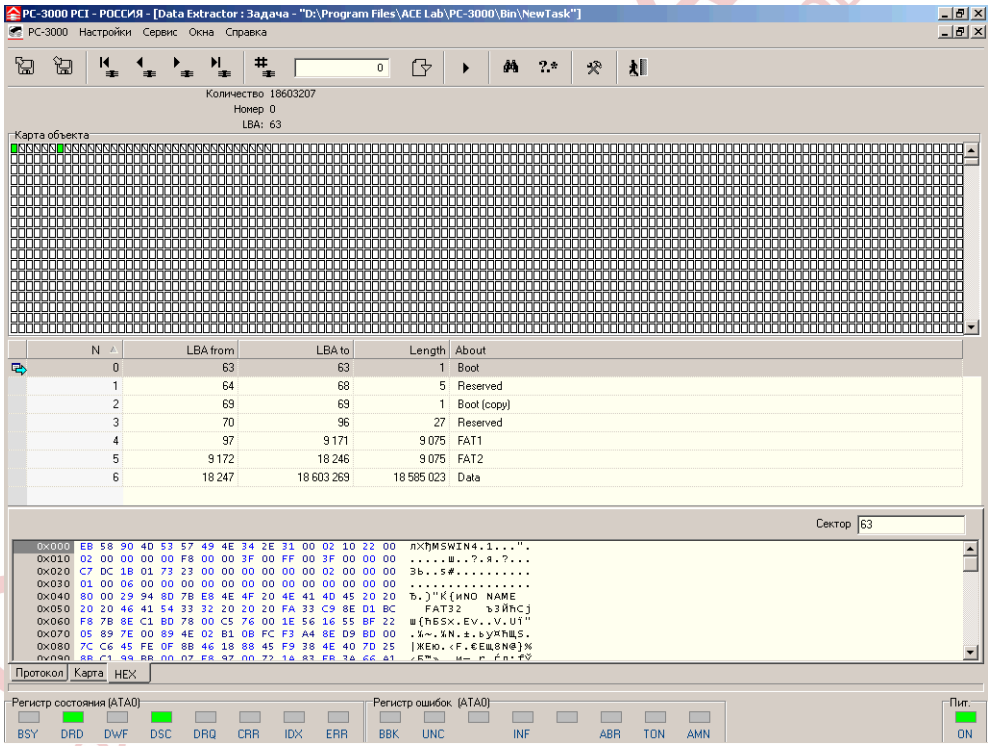


Рисунок 90 Режим "Карта раздела" для раздела FAT32

Примечание (использование "PC-3000 Flash"). Структуру сектора с элементами каталога раздела FAT можно просмотреть, загрузив соответствующий сектор в двоичный редактор и выбрав режим *Просмотр как…→ FAT подкаталог*. В окне просмотра для каждого элемента каталога приводятся: смещение, номер кластера, тип, имя, расширение, размер, длинное имя.

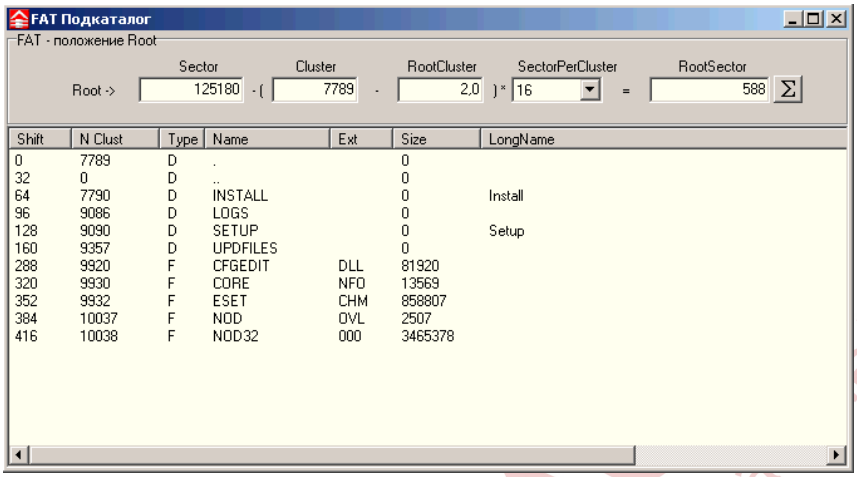


Рисунок 91 Просмотр сектора с элементами каталога в режиме "FAT подкаталог"

В окне просмотра присутствует панель (*FAT – положение Root*) для приблизительного расчета положения корневого (root) каталога. В расчете участвуют следующие значения, полученные из сектора с элементами каталога:

- *Sector* – начальный сектор текущего каталога;
- *Cluster* – начальный кластер текущего каталога;
- *RootCluster* – начальный кластер корневого каталога (для FAT12/16 – 0 или 1,5, для FAT32 – 2);
- *SectorPerCluster* – количество секторов на один кластер;
- *RootSector* – рассчитанный номер сектора корневого каталога.

Данная панель может также использоваться для определения размера кластера: для этого необходимо два сектора с элементами разных каталогов; подставляя различные значения *SectorPerCluster* в расчет, найти совпадающие или мало отличающиеся значения *RootSector*; соответствующее значение *SectorPerCluster* и будет искомым размером кластера для данного радела.

FAT32 использует 32-разрядные идентификаторы кластеров, но при этом резервирует старшие 4 бита, так что эффективный размер идентификатора кластера составляет 28 бит. Поскольку максимальный размер кластеров FAT32 равен 32 Кбайт, теоретически FAT32 может работать с 8-терабайтными томами. Windows 2000 ограничивает размер новых томов FAT32 до 32 Гбайт, хотя поддерживает существующие тома FAT32 большего размера (созданные в других операционных системах).

Первые две ячейки таблицы FAT в системе FAT32 используются следующим образом:

- ячейка 0 содержит байт-описатель носителя, дополненный слева двоичными единицами;
- ячейка 1 содержит код признака конца цепочки кластеров файла (обычно 0FFFFFFFh).

Бит 27 ячейки 1 является признаком завершения работы с диском: 1 - работа завершена нормальным образом, 0 - нет. Бит 26 ячейки 1 является аналогичным признаком нормального завершения операции ввода-вывода.

Таблица 10 – Структура загрузочного сектора раздела с FAT32

Смещение	Длина, байт	Назначение
0h	3	Команда JMP на начало кода загрузчика
3h	8	Название ОС, например "MSDOS6.0"
		BPB
0Bh	2	*BytesPerSector* – количество байт в секторе, обычно 512 (200h)
0Dh	1	*SectorsPerCluster* – количество секторов в кластере
0Eh	2	*ReservedSectors* – количество секторов, занятых загрузчиком и зарезервированных
10h	1	*NumberOfFATs* – количество копий FAT
11h	2	*RootEntries* – максимальное число 32-байтных элементов корневого каталога
13h	2	*TotalSectors* – общее число секторов на томе. 0000 означает, что диск больше 32 Мбайт и число задается двойным словом BigTotSects (смещение 20h)
15h	1	*MediaDescriptor* - дескриптор носителя (то же, что и в первом байте FAT)
16h	2	*SectorsPerFAT* – число секторов в одной FAT (зарезервировано)
18h	2	*SectorsPerTrack* – число секторов на треке
1Ah	2	*Heads* – число головок
1Ch	4	*HiddenSectors* – число скрытых секторов
20h	4	*BigTotalSectors* – число секторов (для разделов > 32 Мбайт)
		BF_BPB
24h	4	*BigSectorsPerFAT* – число секторов в одной FAT (для разделов > 32 Мбайт)
28h	2	*ExtFlags* – номер активной FAT
2Ah	1	*FS_VersionMajor* – номер версии
2Bh	1	*FS_VersionMinor* – номер ревизии
2Ch	4	*RootDirStrtClus* - номер первого кластера, занимаемого корневым каталогом на FAT32-диске
30h	2	*FSInfoSec* – номер сектора структуры FSInfo
32h	2	*BkUpBootSec* – номер резервной копии загрузочного сектора (относительно основного, номер основного – ноль, обычно равен 6)
34h	12	Зарезервировано
40h	1	*PhysicalDiskNumber* – логический номер устройства, номер присваивается в процессе форматирования (80h – первый жесткий диск)
41h	1	Зарезервировано
42h	1	*Signature* – сигнатура расширенного загрузчика (29h)
43h	4	*VolumeSerialNumber* – серийный номер тома (устанавливается при форматировании)
47h	11	*VolumeLabel* – метка тома (строка символов)
52h	8	*SystemID* – символьный идентификатор файловой системы (например, "FAT32")
5Ah	420	**Область кода загрузчика**
1FEh	2	*BootSignature* - Сигнатура 55AAh (конец загрузочного сектора)

Замечание! Наименование параметров на английском языке в данной таблице соответствует обозначениям, используемым в окне просмотра загрузочного сектора в режиме *FAT32 Boot sector* двоичного редактора.

Примечание (использование "PC-3000 Flash"). Для удобства просмотра и редактирования загрузочного сектора раздела с файловой системой FAT32 в программе "PC-3000 Flash" присутствует форма, показанная на рисунке ниже.

Значения в светлых окошках формы значимы для корректного восстановления информации данного раздела. При редактировании данных полей производится проверка корректности введенных данных. В случае если введенные данные не корректны, название параметра выделяется с помощью желтого фона.

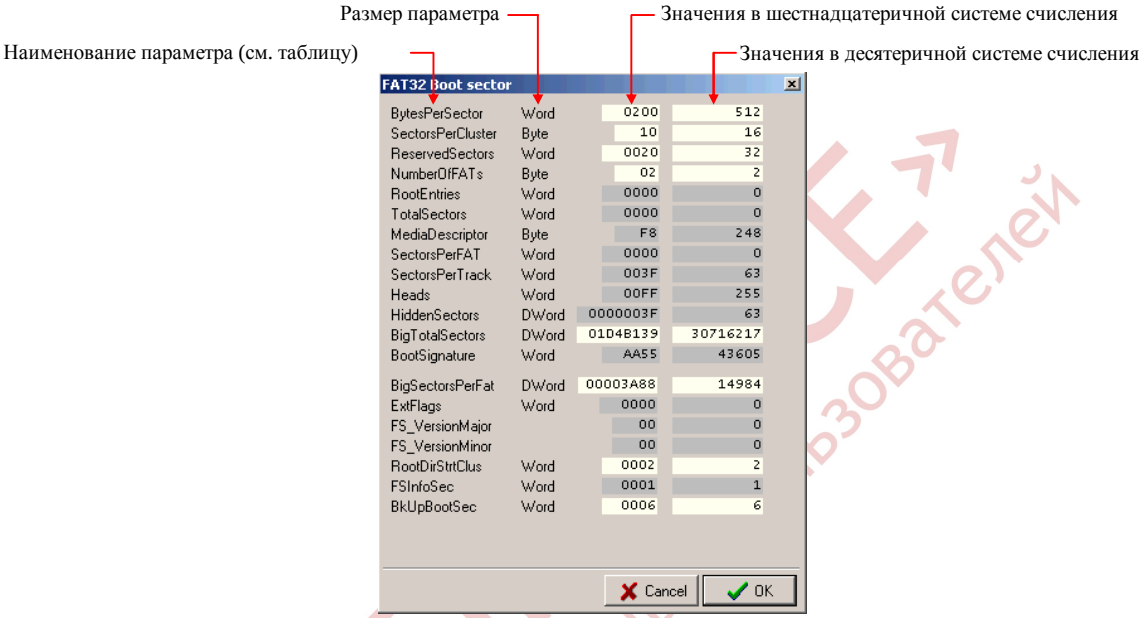


Рисунок 92 Просмотр загрузочного сектора в режиме "FAT32 Boot sector"

Данная форма может быть вызвана из контекстного меню объекта boot-сектор в режиме *Проводник* (пункт *Свойства*) или с помощью двоичного редактора и режима *Просмотр как…→ FAT32 Boot sector*.

Структура FSInfo предназначена для ускорения выполнения операций с FAT. В ней содержится количество свободных секторов и номер первого свободного кластера. Формат структуры приведен в следующей таблице:

Таблица 11 – Структура сектора FSInfo

Смещение	Длина, байт	Назначение
000h	4	Признак структуры 41615252h
004h	480	Зарезервировано
1E4h	4	Признак структуры 61417272h
1E8h	4	Текущее число свободных кластеров
1ECh	4	Номер первого свободного кластера
1F0h	12	Зарезервировано
1FCh	4	Признак структуры AA550000h

Замечание! Если текущее число свободных кластеров содержит значение FFFFFFFFh, то это означает, что оно неизвестно и его следует вычислять. Если такое же число находится в поле первого свободного кластера, это означает, что искать свободный кластер необходимо с кластера 2. Другое значение номера первого свободного кластера указывает не на первый свободный кластер, а на кластер, с которого нужно начинать поиск свободных кластеров.

2.3 Логический диск с системой NTFS

2.3.1 Загрузочный сектор

Первый сектор логического диска носит название *загрузочного* (boot). Он содержит код загрузки (bootstrap code) и сведения о геометрии диска. Структура boot-сектора определяется архитектурными особенностями конкретной файловой системы.

Таблица 12 Структура boot-сектора логического диска с NTFS (сокращенная)

Смещение	Размер, байт	Описание
00h	3	Инструкция перехода
03h	8	OEM ID – идентификатор
0Bh	25	BPB
24h	48	Extended BPB
54h	426	Bootstrap Code
01FEh	2	Сигнатура boot-сектора (55AAh)

Примечание (использование программы "PC-3000 Flash"). Структуру любого раздела можно просмотреть с помощью режима *Карта раздела*, доступного из контекстного меню выбранного раздела в режиме *Проводник*. Данный режим, может быть использован:

– для проверки и коррекции положения важных с точки зрения восстановления метаданных раздела при восстановлении транслятора (копий загрузочного сектора, копий таблицы FAT, начала таблицы MFT и т.д.);

– для проверки местоположения и целостности основных метаданных раздела в случае логических разрушений (быстрый доступ к копиям загрузочного сектора, к таблицам FAT);

– в случае работы с неисправным накопителем можно выбрать наиболее важные с точки зрения цепочки (например, таблицы FAT, boot-сектора) и вычитать их в копию с наиболее жёсткими параметрами (увеличенное количество попыток чтения), оценить качество вычитывания.

Внешний вид режима *Карта раздела* приводится ниже.

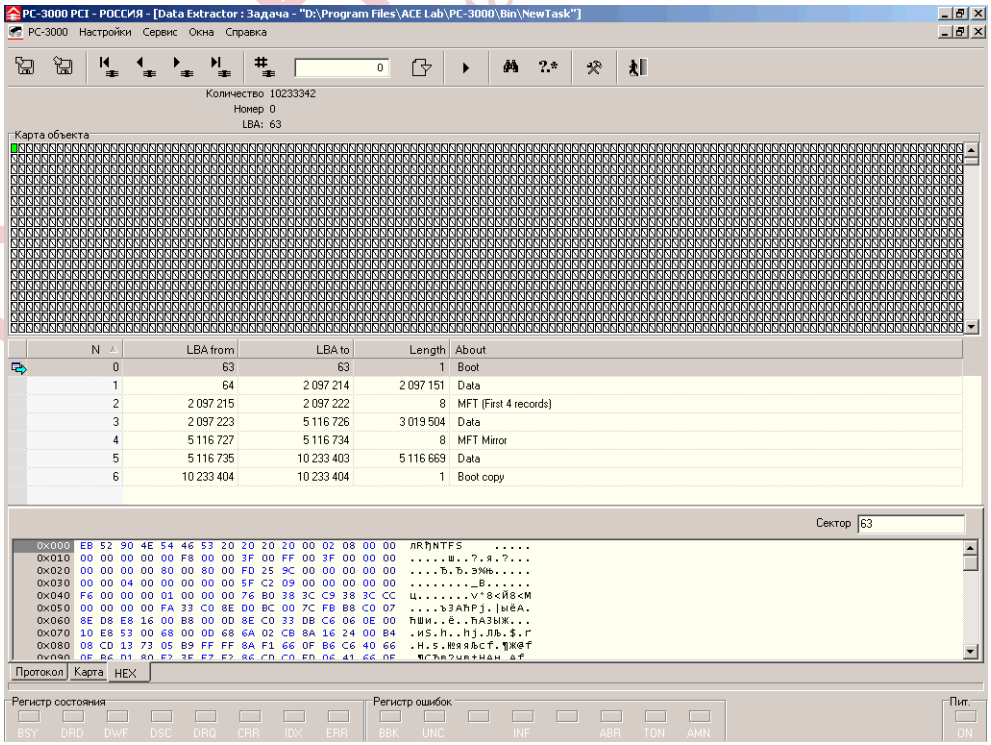


Рисунок 93 Режим "Карта раздела" для раздела NTFS

В начале boot-сектора расположена трехбайтовая машинная команда перехода на bootstrap code. С третьего по одиннадцатый байты (считая от нуля) хранится идентификатор производителя, определяющий тип и версию используемой файловой системы (например, "MSDOS5.0" для FAT16, "MSWIN4.0"/"MSWIN4.1" для FAT32 и "NTFS" для NTFS).

За идентификатором расположен 25-байтовый *блок параметров BIOS* (BIOS Parameter Block или сокращенно BPB), хранящий сведения о геометрии диска (число цилиндров, головок, секторов, размер сектора, кол-во секторов в кластере и т. д.).

NTFS распределяет пространство накопителя кластерами и использует для их нумерации 64 разряда, что дает возможность иметь 2^{64} кластеров, каждый размером до 64 Кбайт. Как и в FAT размер кластера может меняться, но необязательно возрастает пропорционально размеру диска. Размеры кластеров, устанавливаемые по умолчанию при форматировании раздела, приведены в таблице.

Таблица 13 Размер кластера по умолчанию

Размер накопителя	**Размер кластера, кбайт (секторов)**
менее 512 Мбайт	512 байт (1)
от 512 до 1023 Мбайт	1 (2)
от 1 до 2 Гбайт	2 (4)
от 2 до 4 Гбайт	4 (8)
от 4 до 8 Гбайт	8 (16)
от 8 до 16 Гбайт	16 (32)
от 16 до 32 Гбайт	32 (64)
свыше 32 Гбайт	64 (128)

За блоком параметров BIOS следует его продолжение – *расширенный блок параметров BIOS* (extended BPB), хранящий номер первого кластера MFT, его размер в кластерах, номер кластера с зеркалом MFT и некоторую другую информацию. В отличие от FAT16/32, MFT может располагаться в любом месте диска.

Следом за extend BPB идет Bootstrap Code, который ищет на диске операционный загрузчик (у Windows NT это ntldr), загружает его в память и передает ему управление. Если Bootstrap Code отсутствует, загрузка операционной системы становится невозможной, однако, при подключении восстанавливаемого диска вторым, раздел должен быть прекрасно виден.

Завершает boot-сектор сигнатура 55AAh.

Таблица 14 Структура boot-сектора накопителя с NTFS

Смещение	Размер, байт	Описание
00h	3	*Jump* - Инструкция перехода
03h	8	*System_id* - OEM ID
0Bh	2	**Блок параметров BIOS (BPB)** *BytesPerSector* - Количество байт на сектор (для жестких дисков всегда 512)
0Dh	1	*SectorsPerCluster* - Число секторов на кластер
0Eh	2	*ReservedSectors* - Количество зарезервированных секторов
10h	3	*NotUsed1* - Не используется NTFS (должно быть равно 0)
13h	2	*NotUsed2* - Не используется NTFS (должно быть равно 0)
15h	1	*MediaDescriptor* - Дескриптор носителя (для жестких дисков F8h)
16h	2	*NotUsed3* - Не используется NTFS (должно быть равно 0)
18h	2	*SectorsPerTrack* - Количество секторов в треке
1Ah	2	*NumberOfHeads* - Количество головок
1Ch	4	*HiddenSectors* - Количество скрытых секторов
20h	4	*NotUsed4* - Не используется NTFS (должно быть равно 0)
24h	4	**Расширенный блок параметров BIOS (extended BPB)** *NotUsed5* - Не используется NTFS (должно быть равно 0)
28h	8	*TotalSectors* - Общее количество секторов
30h	8	*MFT_Cluster* - Логический номер кластера, с которого начинается MFT
38h	8	*MFT_Mirr_Cluster* - Логический номер кластера, с которого начинается зеркало MTF
40h	4	*ClustPerFileRecord* - Количество кластеров на файловую запись (см. замечание)
44h	4	*ClustPerIndexBlock* - Количество кластеров на блок индексов
48h	8	*lbVolumeSerialNumber* - Серийный номер тома
50h	4	*Checksum* - Контрольная сумма (0 – не подсчитывать).
54h	426	**Bootstrap Code**
01FEh	2	*Signature* - Сигнатура boot-сектора (55AAh)

Замечание! Наименование параметров на английском языке в данной таблице соответствует обозначениям, используемым в окне просмотра загрузочного сектора в режиме *NTFS Boot sector* двоичного редактора.

Замечание! Значение параметра *ClustPerFileRecord* может быть рассчитано следующим образом. В некоторой документации указывается, что значение параметра равное F6h означает, что размер файловой записи равен ¼ от размера кластера (при этом, как рассчитать другие значения не объясняется). Мы предлагаем следующий расчет для данного параметра:

$$N, байт = 2^{(128+\text{ClustPerFileRecord})},$$

где N – размер файловой записи в байтах;

ClustPerIndexBlock – число в прямом коде размером в один байт (например, число в двоичном коде 1111 0110 – означает минус 118 в десятичном коде, соответственно размер файловой записи равен 2^{10} = 1024 байт). Значения параметра всегда отрицательные, а диапазон 0 … 31 недопустим.

Примечание (использование "PC-3000 Flash"). Для удобства просмотра и редактирования загрузочного сектора раздела с файловой системой NTFS в программе "PC-3000 Flash" присутствует форма, показанная на рисунке ниже.

Значения в светлых окошках формы значимы для корректного восстановления информации данного раздела. При редактировании данных полей производится проверка корректности введенных данных. В случае если введенные данные не корректны, название параметра выделяется с помощью желтого фона.

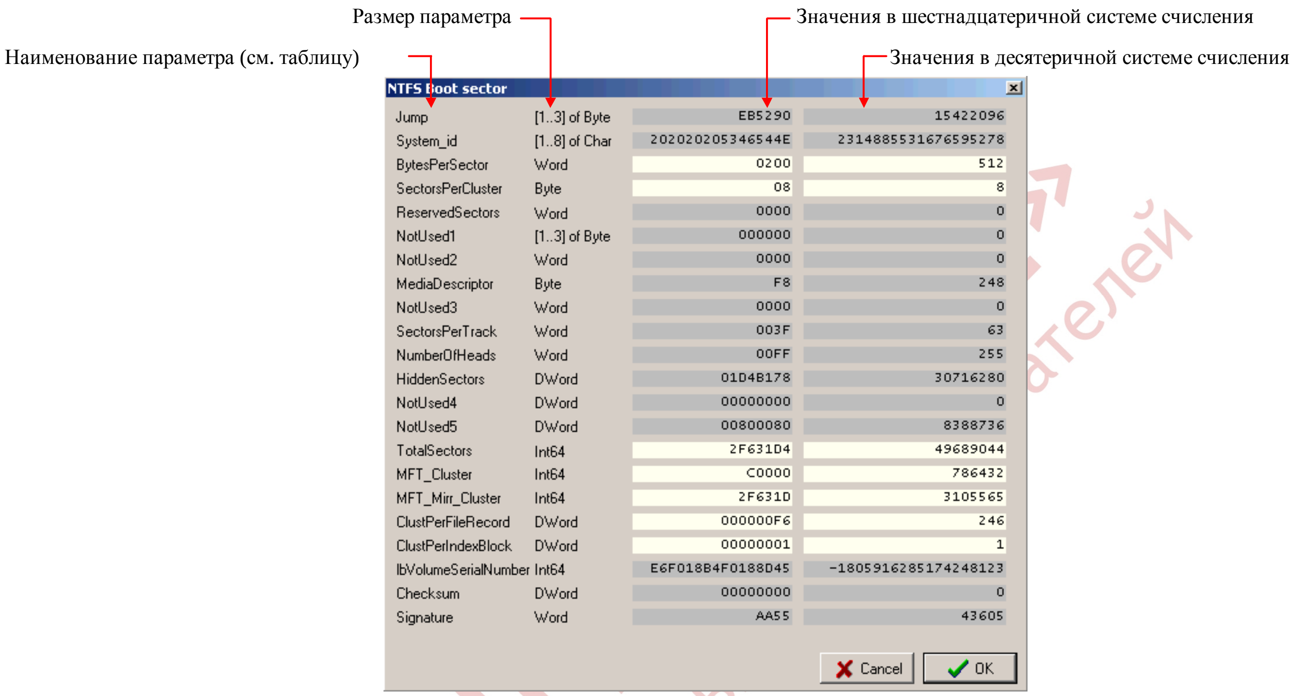


Рисунок 94 Просмотр загрузочного сектора в режиме "NTFS Boot sector"

Данная форма может быть вызвана из контекстного меню объекта boot-сектор в режиме *Проводник* (пункт *Свойства*) или с помощью двоичного редактора и режима *Просмотр как…→ NTFS Boot sector*.

2.3.2 *Общая таблица файлов MFT*

Каждый элемент файловой системы NTFS представляет собой файл - даже служебная информация. Самый главный файл NTFS называется *общей (или главной) таблицей файлов MFT* (Master File Table). Именно он размещается в MFT зоне и представляет собой централизованный каталог всех файлов диска, в том числе и себя самого. *MFT зона* (MFT-zone) создается в процессе форматирования логического раздела и по умолчанию занимает 12,5 % от емкости тома (в зависимости от значения параметра NtfsMftZoneReservation она может составлять 25 %, 37 % или 50 %).

В этой области расположен файл $MFT, изначально занимающий порядка 64 секторов и растущий от начала MFT-зоны к ее концу по мере создания новых пользовательских файлов/подкаталогов. Приблизительный размер MFT-файла можно оценить как произведение размера элемента записи таблицы (обычно 1 Кбайт) на полное количество файлов/подканалов раздела, включая недавно удаленные.

Для предотвращения фрагментации $MFT-файла, MFT-зона удержится зарезервированной вплоть до полного исчерпания свободного пространства тома, затем незадействованный "хвост" MFT-зоны усекается в два раза, освобождая место для пользовательских файлов. Этот процесс может повторяться многократно, вплоть до полной отдачи всего зарезервированного пространства.

При необходимости, $MFT-файл может быть перемещен в любую часть диска и тогда в начале тома его уже не окажется. Стартовый адрес $MFT-файла хранится в Boot-секторе по смещению 30h байт от его начала.

Рисунок 95 Структура дискового тома с файловой системой NTFS

$MFT-файл представляет собой набор записей типа *FILE Record*, каждая из которых описывает соответствующий ей файл или подкаталог. В большинстве случаев один файл/подкаталог полностью описывается одной записью FILE Record, хотя теоретически, этих записей может потребоваться и несколько.

2.3.3 Файловые записи (File Record)

Файловая запись состоит из *заголовка* (*header*) и одного или нескольких *атрибутов* (*attribute*) произвольной длины, завершаемых *маркером конца* (*end marker*) – 32-разрядным значением FF FF FF FFh. Несмотря на то, что количество и длина атрибутов меняется от одной файловой записи к другой, размер самой структуры файловой записи фиксирован и в больше случаев равен 1 Кбайту (это значение храниться в $Boot-файле). Первый байт файловой записи всегда совпадает с началом сектора.

Заголовок, Атрибут 1, Атрибут 2, … , Атрибут N, Маркер конца.

Рисунок 96 Структура файловой записи

Если реальная длина атрибутов меньше размеров файловой записи, ее хвост просто не используется. Если же атрибуты не вмещаются в отведенное им пространство, создается *дополнительная файловая запись* (*extra File Record*), ссылающаяся на свою предшественницу.

Таблица 15 Структура файловой записи

Смещение	Размер, байт	ОС	Описание
00h	4	любая	Сигнатура файловой записи (magic number) "FILE"
04h	2	любая	Смещение порядкового номера обновления (update sequence number), условно (S)
06h	2	любая	Размер в словах порядкового номера обновления и массива обновления (Update Sequence Number & Array)
08h	8	любая	Порядковый номер файла транзакций ($LogFile Sequence Number или сокращенно LSN)
10h	2	любая	Порядковый номер (sequence number)
12h	2	любая	Счетчик жестких ссылок (hard link)
14h	2	любая	Смещение первого атрибута
16h	2	любая	Флаги (flags): 00h – файловая запись не используется; 01h – файловая запись используется и описывает файл (file); 02h – файловая запись используется и описывает каталог (directory); 04h – нет информации; 08h – нет информации;
18h	4	любая	Реальный размер (real size) файловой записи
1Ch	4	любая	Выделенный размер (allocated size) файловой записи
20h	8	любая	Ссылка на (file reference) базовую файловую запись (base File Record) (0, если данная файловая запись базовая)
28h	2	любая	Идентификатор следующего атрибута (next attribute ID)
2Ah	2	XP	Для выравнивания
2Ch	4	XP	Индекс данной файловой записи (number of this MFT record)
	2	любая	Номер последовательности обновления (update sequence number)
	2S-2	любая	Массив последовательности обновления (update sequence array)

Сигнатура файловой записи. Первые четыре байта заголовка имеют значение 46 49 4C 45h ("FILE") и сигнализируют о том, что мы имеем дело с файловой записью MFT типа File Record.

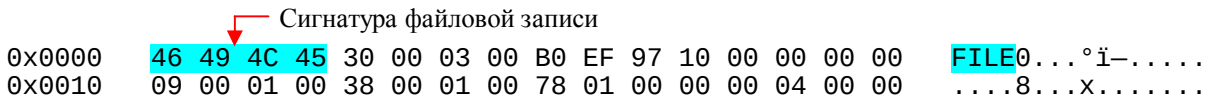


Рисунок 97 Сигнатура файловой записи

Смещение *порядкового номера обновления* (update sequence number). Следом за сигнатурой идет 16-разрядное число, содержащее смещение порядкового номера обновления (см. раздел *Последовательность обновления*).

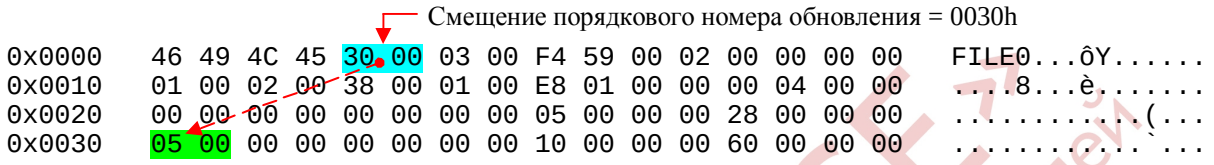


Рисунок 98 Смещение порядкового номера обновления

Размер заголовка (варьирующийся от одной операционной системы к другой) в явном виде не указан, вместо этого в заголовке присутствует 16-разрядное число расположенное по смещению 14h байт от начала сектора, содержащее *смещение первого атрибута* в байтах относительно начала файловой записи. Смещения последующих атрибутов (если они есть) определяются путем сложения размеров всех предыдущих атрибутов (размер каждого из атрибутов содержится в его заголовке) со смещением первого атрибута (подробнее см. раздел *Атрибуты*). В конце последнего атрибута находится 32-разрядный маркер конца – FF FF FF FFh.

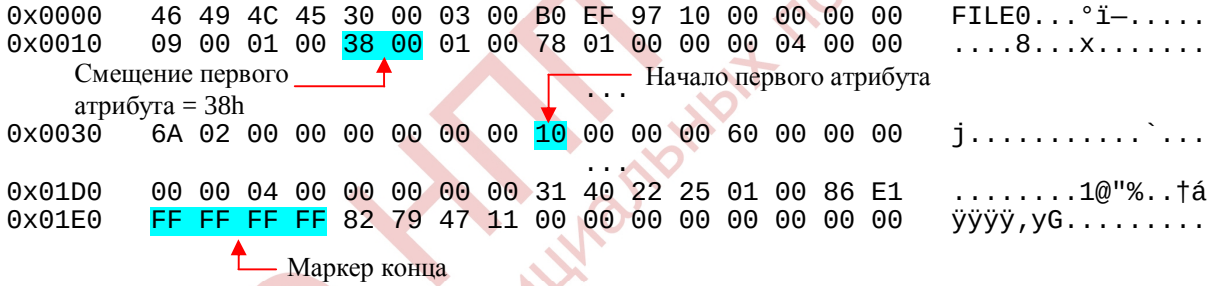


Рисунок 99 Указатель на первый элемент, маркер конца

Длина файловой записи хранится в двух полях 32-разрядных полях. Поле *реального размера* (*real size*), находящееся по смещению 18h байт от начала сектора, содержит совокупный размер заголовка, всех его атрибутов и маркера конца, округленный по 8-ми байтной границе. Поле *выделенного размера* (*allocated size*), находящееся по смещению 1Ch байт от начала сектора, содержит количество байт выделенных на файловую запись, округленный по размеру сектора.

Для 16-разрядного поля флагов, находящегося по смещению 16h байт от начала сектора определены три следующих значения: 00h – данная файловая запись не используется или ассоциированный с ней файл/каталог удален, 01h – файловая запись используется и описывает файл, 02h – файловая запись используется и описывает каталог. Очевидно, что значение данного поля получается в результате сложения определенных для него признаков, т.е. файловая запись, описывающая каталог, имеет значение 01h + 02h = 03h. Неописанные признаки 04h и 08h, возможно, несут информацию о версии файловой системы (NT, 2K) и используются применительно к таким системным файлам как $Secure, $Quota, $ObjID, $Reparse и $UsnJrnl.

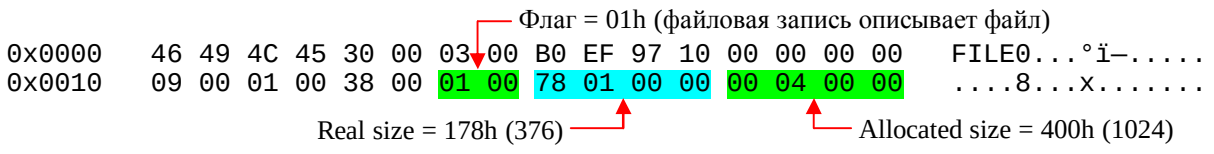


Рисунок 100 Флаг, поля real size и allocated size

64-разрядное поле, находящееся по смещению 20h байт от начала сектора содержит индекс базовой файловой записи. Для первой файловой записи это поле всегда равно нулю, а для всех последующих, расширенных (extra) записей – индексу первой файловой записи. Расширенные файловые записи могут

находится в любых частях MFT, не обязательно рядом с основной записью. Для обеспечения быстрого поиска расширенных файловых записей ведется список атрибутов $ATTRIBUTE_LIST. Список атрибутов представляет собой специальный атрибут, добавляемый к первой файловой записи и содержащий индексы расширенных записей. Формат списка атрибутов приведен в разделе *Типы атрибутов*.

```
0x0010   09 00 01 00 38 00 01 00 78 01 00 00 00 04 00 00   ....8...x.......
0x0020   00 00 00 00 00 00 00 00 05 00 00 00 1B 00 00 00   ................
```

Индекс базовой файловой записи (данная запись базовая)

Индекс файловой записи = 1Bh (27)

Рисунок 101 Индексы базовой и текущей файловой записи

2.3.4 *Последовательность обновления (Update Sequence)*

Файловые записи нуждаются в механизме контроля целостности своего содержимого. В NTFS для этой цели используется *последовательность обновления* (update sequence).

В конец каждого из секторов, образующих файловую запись записывается специальный 16-разрядный *порядковый номер обновления* (update sequence number), дублируемый в заголовке файловой записи. При каждой операции чтения два последних байта сектора сверяется с соответствующим полем заголовка и, если NTFS-драйвер обнаруживает расхождение, данная файловая запись считается недействительной.

Основное назначение последовательности обновления – защита от "обрыва записи". Если в процессе записи сектора на диск, исчезнет питающее напряжение, может случиться так, что половина файловой записи будет успешно записана, а половина – сохранит прежнее содержимое (файловая запись, обычно состоит из двух секторов). При каждой перезаписи сектора порядковый номер увеличивается на единицу, поэтому если произошел обрыв записи, значение порядкового номера, находящегося в заголовке файловой записи не будет совпадать с номером, расположенным в конце сектора.

Данные, на место которых были записаны порядковые номера последовательности обновления, хранятся в специальном *массиве обновления* (update sequence array), расположенном в заголовке файловой записи непосредственно за порядковым номером обновления.

Для проверки целостности файловой записи, необходимо извлечь из заголовка указатель на порядковый номер (он хранится по смещению 04h байт от начала заголовка), и сверить лежащее по этому адресу 16-разрядное значение с последним словом каждого из секторов, слагающих файловую запись. Если они не совпадут, то соответствующая структура данных повреждена, и использовать ее следует с большой осторожностью.

Указатель на порядковый номер = 30h

Число секторов в файловой записи = 03h – 1 = 2 сектора

Начало файловой записи

```
0x0000   46 49 4C 45 30 00 03 00 25 49 74 2A 00 00 00 00   FILE0...%It*....
0x0010   01 00 01 00 38 00 03 00 D8 01 00 00 00 04 00 00   ....8...Ø.......
0x0020   00 00 00 00 00 00 00 00 06 00 00 00 1E 00 00 00   ................
0x0030   0E 14 72 00 00 00 00 00 10 00 00 00 60 00 00 00   ..r.........`...
```

Порядковый номер обновления (проверочное значение)

```
                                    ...
0x01F0   08 03 73 00 6F 00 66 00 74 00 77 00 61 00 0E 14   ..s.o.f.t.w.a...
```

Конец первого сектора файловой записи

Порядковый номер обновления (проверяемое значение)

```
                                    ...
0x0000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 0E 14   ................
```

Конец второго сектора файловой записи

Рисунок 102 Проверка целостности файловой записи

После успешной проверки целостности файловой записи, необходимо восстановить ее исходный вид (предполагается изменение копии сектора в памяти). Для этого по смещению 06h от начала сектора находится 16-разрядное поле, хранящее совокупный размер порядкового номера обновления вместе с массивом обновления (SizeOf(update sequence number) + SizeOf(update sequence array)), выраженный в словах (два байта). Размер порядкового номера последовательности обновления всегда равен одному слову, соответственно,

значение 16-разрядного поля по смещению 06h минус один, определяет число секторов в файловой записи. Первое слово массива обновления соответствует последнему слову первого сектора файловой записи. Второе – последнему слову второго сектора, и т. д.

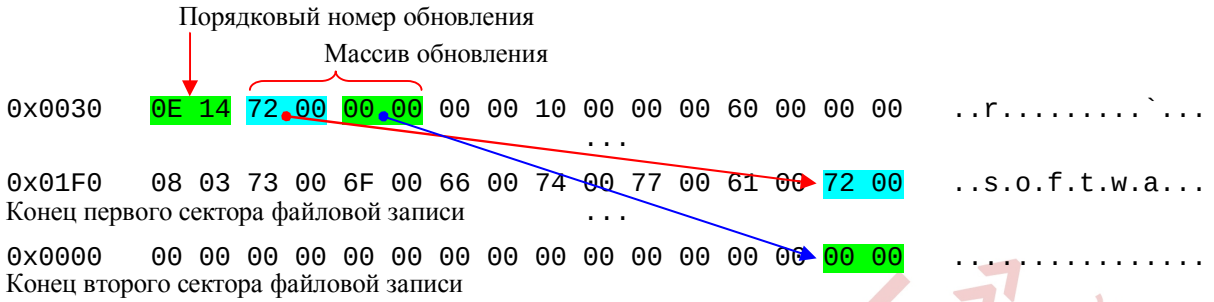


Рисунок 103 Восстановление копии файловой записи в памяти

2.3.5 *Атрибуты (Attribute)*

Любой атрибут состоит из *заголовка* (header) и *тела* (body). Заголовок атрибута и тело резидентного атрибута хранятся в файловой записи, расположенной внутри MFT (см. раздел *Файловые записи*). Тело нерезидентного атрибута хранится вне MFT, в одном или нескольких кластерах, перечисленных в заголовке данного атрибута в специальном списке (см. раздел *Списки отрезков*). Если 8-разрядное поле, расположенное по смещению 08h байт от начала атрибутного заголовка, равно нулю – атрибут считается резидентным, а если единице – то нет. Любые другие значения недопустимы.

Первые четыре байта атрибутного заголовка определяют его тип. Тип атрибута в свою очередь определяет формат представления тела атрибута.

Таблица 16 Типы атрибутов

Значение	ОС	Условное обозначение	Описание
010h	любая	$STANDARD_INFORMATION	Стандартная информация о файле (время, права доступа)
020h	любая	$ATTRIBUTE_LIST	Список атрибутов
030h	любая	$FILE_NAME	Имя файла
040h	NT	$VOLUME_VERSION	Версия тома
040h	2K	$OBJECT_ID	Уникальный GUID и прочие ID
050h	любая	$SECURITY_DESCRIPTOR	Дескриптор безопасности и списки прав доступа (ACL)
060h	любая	$VOLUME_NAME	Имя тома
070h	любая	$VOLUME_INFORMATION	Информация о томе
080h	любая	$DATA	Основные данные файла
090h	любая	$INDEX_ROOT	Корень индексов
0A0h	любая	$INDEX_ALLOCATION	Ветви (sub-nodes) индекса
0B0h	любая	$BITMAP	Карта свободного пространства
0C0h	NT	$SYMBOLIC_LINK	Символическая связь
0C0h	2K	$REPARSE_POINT	для сторонних производителей
0D0h	любая	$EA_INFORMATION	Расширенные атрибуты для HPFS
0E0h	любая	$EA	Расширенные атрибуты для HPFS
0F0h	NT	$PROPERTY_SET	Устарело и ныне не используется
100h	2K	$LOGGED_UTILITY_STREAM	Используется шифрованной файловой системой (EFS)

Следующие четыре байта заголовка содержат длину атрибута, выражаемую в байтах. Длина нерезидентного атрибута равна сумме длин его тела и заголовка, а длина резидентного атрибута равна длине его заголовка. Т.е., если к смещению атрибута добавить его длину, мы получим указатель на следующий атрибут (или маркер конца, если текущий атрибут – последний в цепочке).

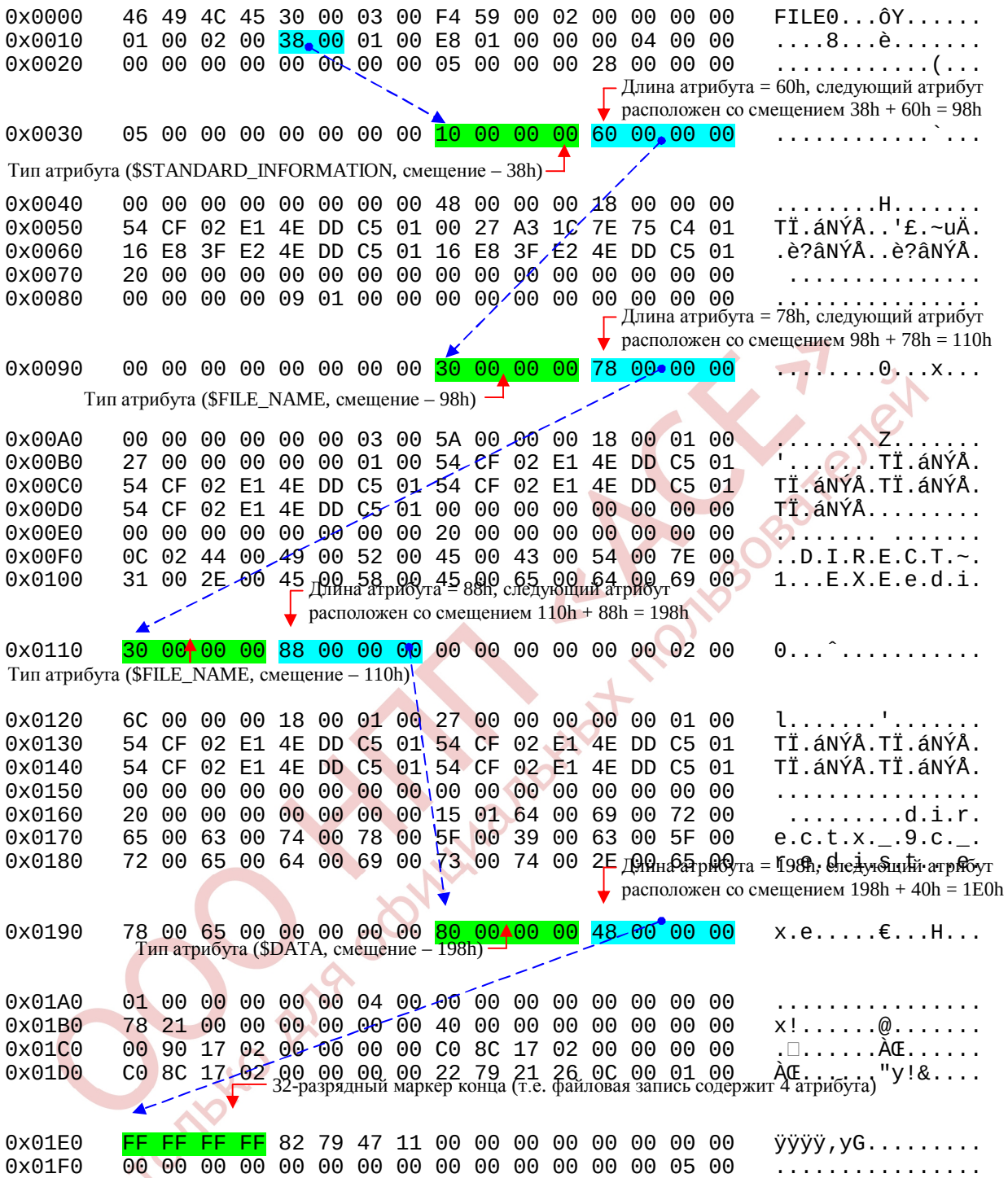


Рисунок 104 Определение местоположения атрибутов в файловой записи

Длина тела резидентных атрибутов, выраженная в байтах, хранится в 32-разрядном поле, расположенном по смещению 10h байт от начала атрибутного заголовка. 16-разрядное поле, следующее за ним, хранит смещение резидентного тела, отсчитываемое от начала атрибутного заголовка.

У нерезидентных атрибутов для хранения длины их тела используется множество полей. *Реальный размер тела атрибута* (real size of attribute), выраженный в байтах, хранится в 64-разрядном поле, находящемся по смещению 30h байт от начала атрибутного заголовка. Следующее за ним 64-разрядное поле хранит *инициализированный размер потока* (initialized data size of the stream), выраженный в байтах и, судя по всему, всегда равный реальному размеру тела атрибута. 64-разрядное поле, расположенное со смещением 28h байт от начала атрибутного заголовка, хранит *выделенный размер* (allocated size of attribute), выраженный в байтах и равный реальному размеру тела атрибута округленному до размера кластера (в большую сторону).

Два 64-разрядных поля, расположенные со смещением 10h и 18h байт от начала атрибутного заголовка задают первый (starting VCN) и последний (last VCN) номер виртуального кластера, принадлежащего телу нерезидентного атрибута. Виртуальные кластеры представляют собой логические номера кластеров, не зависящие от своего физического расположения на диске. В подавляющем большинстве случав, номер первого кластера тела нерезидентного атрибута равен нулю, а последний – количеству кластеров занятых телом атрибута, уменьшенном на единицу. 16-разрядное поле, расположенное со смещением 20h от начала атрибутного заголовка, содержит указатель на массив *Data Runs*, расположенный внутри этого заголовка и описывающий логический порядок размещения нерезидентного тела атрибута на диске (подробнее см. *Списки отрезков*).

Каждый атрибут имеет свой собственный *идентификатор* (attribute ID), уникальный для данной файловой записи и хранящийся в 16-разрядном поле, расположенном со смещением 0Eh от начала атрибутного заголовка.

Если атрибут имеет *имя* (attribute Name), то 16-разрядное поле, расположенное со смещением 0Ah байт от атрибутного заголовка, содержит указатель на него. Для безымянных атрибутов оно равно нулю (большинство атрибутов не имеют имени). Имя атрибута хранится в атрибутном заголовке в формате UNICODE, а его длина определяется 8-разрядным полем, расположенным со смещением 09h байт от начала атрибутного заголовка.

Если тело атрибута сжато, зашифровано или разряжено, 16-разрядное поле флагов, расположенное по смещению 0Ch байт от начала атрибутного заголовка не равно нулю.

Таблица 17 Структура резидентного атрибута

Смещение	Размер, байт	Значение	Описание
00h	4		Тип (type) атрибута (например,. 10h, 60h, B0h)
04h	4		Длина атрибута, включая этот заголовок
08h	1	00h	Нерезидентный флаг (non-resident flag)
09h	1	N	Длина имени атрибута (0, если атрибут безымянный)
0Ah	2	18h	Смещение имени (0, если атрибут безымянный)
0Ch	2	0001h 4000h 8000h	Флаги: - сжатый атрибут (compressed) - зашифрованный атрибут (encrypted) - разряженный атрибут (sparse)
0Eh	2		Идентификатор атрибута (attribute ID)
10h	4	L	Длина тела атрибута, без заголовка
14h	2	2N+18h	Смещение тела атрибута
16h	1		Индексный флаг
17h	1	00h	Для выравнивания
18h	2N	UNICODE	Имя атрибута (если есть)
2N+18h	L		Тело атрибута

Таблица 18 Структура нерезидентного атрибута

Смещение	Размер, байт	Значение	Описание
00h	4		Тип (type) атрибута (например,. 0x20, 0x80)
04h	4		Длина атрибута, включая этот заголовок
08h	1	01h	Нерезидентный флаг (non-resident flag)
09h	1	N	Длина имени атрибута (ноль если атрибут безымянный)
0Ah	2	40h	Смещение имени (ноль если атрибут безымянный)
0Ch	2	0001h 4000h 8000h	Флаги: - сжатый атрибут (compressed) - зашифрованный атрибут (encrypted) - разряженный атрибут (sparse)
0Eh	2		Идентификатор атрибута (attribute ID)
10h	8		Начальный виртуальный кластер (starting VCN)
18h	8		Конечный виртуальный кластер (last VCN)
20h	2	2N+40h	Смещение списка отрезков (data runs)
22h	2		Размер блока сжатия (compression unit size), округленный до 4 байт вверх
24h	4	00h	Для выравнивания
28h	8		Выделенный размер (allocated size), округленный до размера кластера
30h	8		Реальный размер (real size)
38h	8		Инициализированный размер потока (initialized data size of the stream)
40h	2N	UNICODE	Имя атрибута если есть
2N+40h	..		Список отрезков (data runs)

2.3.5.1 Атрибут стандартной информации ($STANDARD_INFORMATION, 10h)

Атрибут стандартной информации описывает время создания/изменения/последнего доступа к файлу, права доступа, а так же некоторую другую вспомогательную информацию (например, квоты).

Таблица 19 Структура атрибута стандартной информации (10h)

Смещение	Размер, байт	ОС	Описание
~ ~		любая	Стандартный атрибутный заголовок (standard attribute header)
00h	8	любая	C - время создания (creation) файла
08h	8	любая	A - время изменения (altered) файла
10h	8	любая	M - время изменения файловой записи (MFT changed)
18h	8	любая	R - время последнего чтения (read) файла
20h	4	любая	Права доступа MS-DOS (MS-DOS file permissions)
24h	4	любая	Старшее двойное слово номера версии (maximum number of versions)
28h	4	любая	Младшее двойное слово номера версии (version number)
2Ch	4	любая	Идентификатор класса (class ID)
30h	4	2K	Идентификатор владельца (owner ID)
34h	4	2K	Идентификатор безопасности (security ID)
38h	8	2K	Количество квотируемых байт (quota charged)
40h	8	2K	Номер последней последовательности обновления (update sequence number USN)

Для прав доступа MS-DOS (MS-DOS file permissions) определены следующие значения:

- 0001h - только на чтение (read-only);
- 0002h - скрытый (hidden);
- 0004h - системный (system);
- 0020h - архивный (archive);
- 0040h - устройство (device);
- 0080h - обычный (normal);

- 0100h - временный (temporary);
- 0200h - разряженный (sparse) файл;
- 0400h - reparse point;
- 0800h - сжатый (compressed);
- 1000h - оффлайновый (offline);
- 2000h - не идексируемый (not content indexed);
- 4000h - зашифрованный (encrypted).

2.3.5.2 Список атрибутов ($ATTRIBUTE_LIST, 20h)

Список атрибутов используется в тех случаях когда, все атрибуты файла не умещаются в базовой файловой записи и файловая система вынуждена располагать их в расширенных. Индексы расширенных файловых записей содержатся в атрибуте $ATTRIBUTE_LIST (список атрибутов), помещаемом в базовую файловую запись.

Атрибуты не умещаются в одной файловой записи в следующих случаях:
- файл содержит много альтернативных имен или жестких ссылок;
- файл сильно фрагментирован;
- файл содержит очень сложный дескриптор безопасности;
- файл имеет очень много потоков данных (т. е. атрибутов типа $DATA).

Таблица 20 Структура атрибута $ATTRIBUTE_LIST (20h)

Смещение	Размер, байт	Описание
~ ~		Стандартный атрибутный заголовок (standard attribute header)
00h	4	Тип (type) атрибута (см. таблицу *Типы атрибутов*)
04h	2	Длина записи (record length)
06h	1	Длина имени (name length), или ноль, если нет, условно – N
07h	1	Смещение имени (offset to name), или ноль если нет
08h	8	Начальный виртуальный кластер (starting VCN), или ноль, если атрибут резидентный
10h	8	Ссылка на базовую/расширенную файловую запись
18h	2	Идентификатор атрибута (attribute ID)
1Ah	2N	Имя в формате UNICODE (если N > 0)

2.3.5.3 Атрибут имени файла ($FILE_NAME, 30h)

Атрибут $FILE_NAME хранит имя файла в соответствующем пространстве имен (максимальная длина имени файла равна 255 символов в формате UNICODE). Таких атрибутов у файла можно быть и несколько (например, win32-имя и MS-DOS имя). Здесь же хранятся и жесткие ссылки (hard link), если они есть.

Таблица 21 Структура атрибута $FILE_NAME (30h)

Смещение	Размер, байт	Описание
~ ~		Стандартный атрибутный заголовок (standard attribute header)
00h	8	Ссылка (file reference) на материнский каталог
08h	8	C – время создания (creation) файла
10h	8	A – время последнего изменения (altered) файла
18h	8	M – время последнего изменения файловой записи (MFT changed)
20h	8	R – время последнего чтения (read) файла
28h	8	Выделенный размер (allocated size) файла
30h	8	Реальный размер (real size) файла
38h	4	Флаг прав доступа (см. описание атрибута стандартной информации)
3Ch	4	Используется HPFS
40h	1	Длина имени в символах – L
41h	1	Пространство имен файла (filename namespace)
42h	2L	Имя файла в формате UNICODE без завершающего нуля

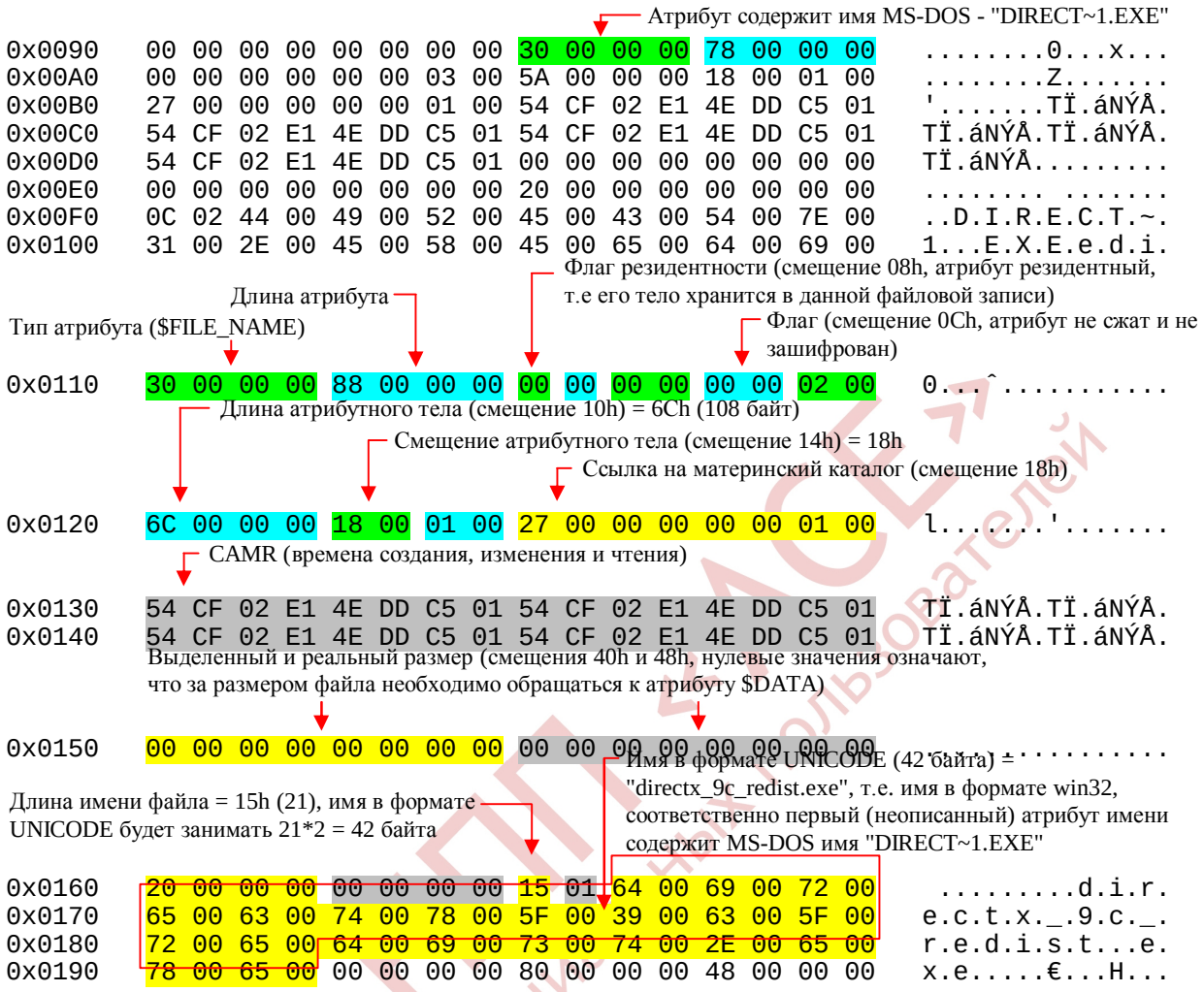


Рисунок 105 Пример атрибута $FILE_NAME

2.3.6 *Списки отрезков (Data Runs)*

Тела нерезидентных атрибутов хранятся на диске в одной или нескольких кластерных цепочках, называемых *отрезками* (runs). Отрезком называется последовательность смежных кластеров, характеризующаяся номером начального кластера и длинной. Совокупность отрезков называется списком (run-list).

Для экономии места длина отрезка и номер начального кластера хранятся в полях переменной длины. То есть, если размер отрезка умещается в байте (т. е. его значение не превышает 255), он и хранится в байте. Соответственно, если размер отрезка требует для своего представления двойного слова, он и хранится в двойном слове.

Сами же поля размеров хранятся в 32-разрядных полях, называемых *полубайтами* (nibble). Шестнадцатеричная система исчисления позволяет легко переводить байты в полубайты и наоборот. Младший полубайт соответствует младшему шестнадцатеричному разряду байта, а старший – старшему. Например, 69h состоит из двух полубайт – младший равен 9h, а старший – 6h.

Список отрезков представляет собой массив структур, каждая из которых описывает характеристики "своего" отрезка, а в конце списка 16-разрядный маркер конца равный 00h. Первый байт структуры состоит из двух полубайт: младший задает длину поля начального кластера отрезка (условно обозначаемого буквой F), старший – количество кластеров в отрезке (L). Поле длины отрезка идет следом. В зависимости от значения L оно может занимать от одного до восьми байт (более длинные поля недопустимы). Первый байт поля стартового кластера файла расположен со смещением 1 + L байт от начала структуры (что соответствует 2+2*L полубайтам).

Таблица 22 Структура отрезка

Смещение, полубайт	Размер, полубайт	Описание
0	1	Размер поля длины (L)
1	1	Размер поля начального кластера (S)
2	2*L	Количество кластеров в отрезке
2+2*L	2*S	Номер начального кластера отрезка

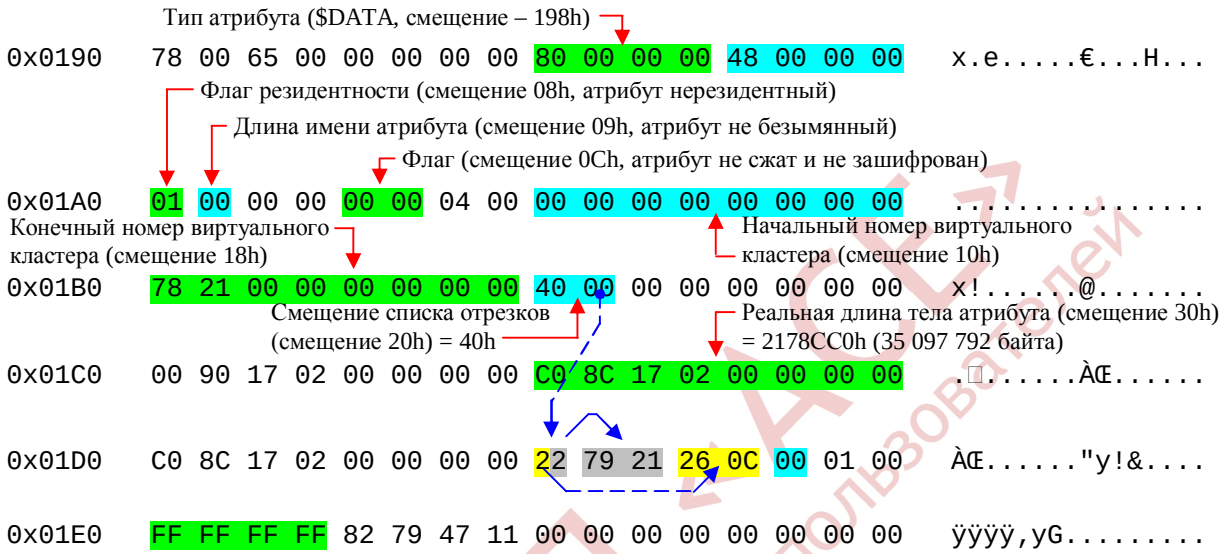


Рисунок 106 Пример атрибута $DATA

Рассмотрим, список отрезков, соответствующий нормальному не фрагментированному файлу показанному на рисунке выше: "22 79 21 26 0C 00".

Первый байт – 22h. Младший полубайт (x2h) описывает размер поля длины отрезка, старший (2xh) – размер поля начального кластера. Следующие несколько байт представляют поле длины отрезка, размер которого в данном случае равен двум байтам – 79h 21h. Два других байта (26h 0Ch) задают номер начального кластера отрезка. Нулевой байт на конце (00h) сигнализирует о том, что это последний отрезок в файле. В результате, наш файл состоит из одного отрезка, начинающегося с кластера 0C26h (обратный порядок байт) и заканчивающегося кластером 0C26h + 2179h = 2D9Fh.

В том случае, если бы файл был фрагментирован, то список отрезков будет выглядеть, например, таким образом: "31 38 73 25 34 32 14 01 E5 11 02 31 42 AA 00 03 00". Первый отрезок (run 1) начинается с кластера 342573h и продолжается вплоть до кластера 342573h + 38h = 3425ABh. Второй отрезок (run 2) начинается с кластера 0211E5h и продолжается вплоть до кластера 0211E5h + 114h = 212F9h. Третий отрезок (run 3) начинается с кластера 0300AAh и продолжается вплоть до кластера 0300AAh + 42h = 300ECh. Нулевой байт на конце (00h) сигнализирует о том, что это последний отрезок в файле. Таким образом, файл состоит из трех отрезков, разбросанных по диску в следующем порядке: 342573h – 3425ABh; 0211E5h – 212F9h; 0300AAh – 300ECh.

Начиная с версии 3.0 NTFS поддерживает разряженные (sparse) атрибуты, т. е. такие, которые не записывают на диск кластеры, содержащие одни нули (если атрибут разряжен, то его флаг равен 8000h). При этом поле номера начального кластера отрезка может быть равным нулю, что означает, что данную отрезку не выделен никакой кластер. Поле длины содержит количество кластеров, заполненных нулями. Их не нужно считывать с диска. Вы должны самостоятельно изготовить их в памяти.

2.3.7 Метафайлы

Первые 12 записей в MFT всегда занимают служебные метафайлы: $MFT (собственно, сам $MFT), $MFTMirr (зеркало $MFT), $LogFile (файл транзакций), $Volume (сведения о дисковом томе), $AttrDef (определенные атрибуты), '.' (корневой каталог), $Bitmap (карта свободного пространства), $Boot (системный загрузчик), $BadClus (перечень плохих кластеров) и т. д. Каждый из них отвечает за какой-либо аспект работы системы. В следующей таблице приведены используемые в данный момент метафайлы и их назначение.

Первые четыре записи настолько важны, что продублированы в специальном $MFTMirr-файле, находящимся приблизительно посередине диска (точное расположение хранится в boot-секторе по смещению 38h байт от его начала). Вопреки своему названию, $MFTMirr это отнюдь не зеркало всего $MFT-файла, это всего лишь копия первых четырех элементов.

Записи с 12 по 15 помечены как используемые, но в действительности же они пусты. Записи с 16 по 23 не задействованы и помечены как неиспользуемые.

Начиная с 24 записи располагаются пользовательские файлы и каталоги. Четыре метафайла, появившихся в W2K – $ObjId, $Quota, $Reparse и $UsnJrnl – могут располагаться в любой записи, номер которой равен 24 или больше (номера файловых записей отсчитываются, начиная с нуля).

Таблица 23 Служебные файлы NTFS (метафайлы)

№ записи в MFT	Имя файла	ОС	Описание
0	$MFT	любая	Общая файловая таблица (Master File Table, MFT)
1	$MFTMirr	любая	Резервная копия первых четырех элементов MFT
2	$LogFile	любая	Журнал транзакций (transactional logging file)
3	$Volume	любая	Серийный номер, время создания, dirty flag (флаг не сброшенного кэша) тома
4	$AttrDef	любая	Определение атрибутов
5	. (точка)	любая	Корневой каталог (root directory) тома
6	$Bitmap	любая	Карта свободного/занятого пространства
7	$Boot	любая	Загрузочная записи (boot record) тома
8	$BadClus	любая	Список плохих кластеров (bad clusters) тома
9	$Quota	NT	Информация о квотах (quota information)
9	$Secure	2K	Использованные дескрипторы безопасности (security descriptors)
10	$UpCase	любая	Таблица заглавных символов (uppercase characters) для трансляции имен
11	$Extend	2K	Каталоги: $ObjId, $Quota, $Reparse, $UsnJrnl
12-15	не используется	любая	Помечены как используемые, но в действительности пустые
16-23	не используется	любая	Помечены как неиспользуемые
≥ 24	ObjId	2K	Уникальные идентификаторы каждого файла
≥ 24	$Quota	2K	Информация о квотах (quota information)
≥ 24	$Reparse	2K	Информация типа reparse point
≥ 24	$UsnJrnl	2K	Журнал шифрованной файловой системы (journaling of encryption)
≥ 24	файл	любая	Обычные файлы
≥ 24	каталог	любая	Обычные каталоги

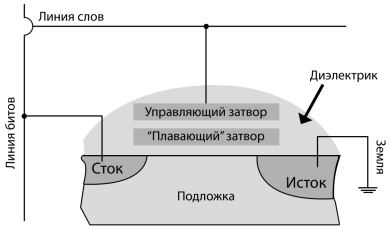
3 Устройство микросхем NAND флэш-памяти.

3.1 Flash (полное историческое название Flash Erase EEPROM):

Во флэш-памяти используется несколько отличный от EEPROM тип ячейки-транзистора. Технологически флэш-память родственна как EPROM, так и EEPROM. Основное отличие флэш-памяти от EEPROM заключается в том, что стирание содержимого ячеек выполняется для определённого блока (кластера, кадра или страницы). Среду существующих микросхем, величина блока является достаточно широко варьируемой величиной. Таким образом, в общем случае, для того, чтобы изменить один байт, сначала в буфер считывается весь блок, где содержится подлежащий изменению байт, стирается содержимое блока, изменяется значение байта в буфере, после чего производится запись измененного в буфере блока. Такая схема существенно снижает скорость записи небольших объёмов данных в произвольные области памяти, однако значительно увеличивает быстродействие при последовательной записи данных большими порциями.

3.2 Организация flash-памяти

Ячейки флэш-памяти бывают как на одном, так и на двух транзисторах. В простейшем случае каждая ячейка хранит один бит информации и состоит из одного полевого транзистора со специальной электрически изолированной областью ("плавающим" затвором - floating gate), способной хранить заряд многие годы. Наличие или отсутствие заряда кодирует один бит информации. Также в ячейке имеются «сток» и «исток», при программировании между ними, вследствие воздействия положительного поля на управляющем затворе, создается канал – поток электронов.

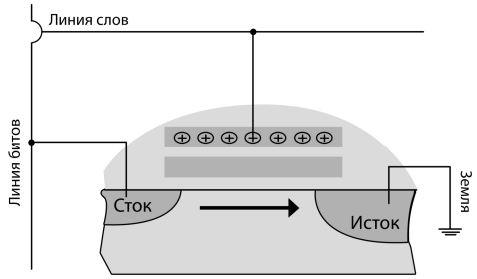


При записи заряд помещается на плавающий затвор одним из двух способов (зависит от типа ячейки): методом инжекции "горячих" электронов или методом туннелирования электронов. Стирание содержимого ячейки (снятие заряда с "плавающего" затвора) производится методом туннеллирования.

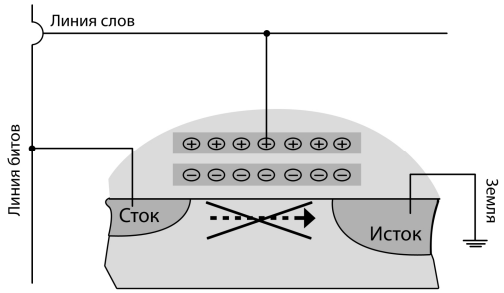
Как правило, наличие заряда на транзисторе понимается как логический "0", а его отсутствие - как логическая "1". Современная флэш-память обычно изготавливается по 0,13- и 0,18-микронному техпроцессу.

3.3 Общий принцип работы ячейки флэш-памяти.

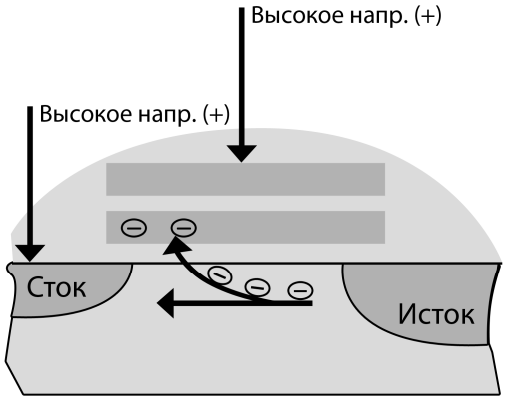
Рассмотрим простейшую ячейку флэш-памяти на одном n-p-n транзисторе. Ячейки подобного типа чаще всего применялись во flash-памяти с NOR архитектурой, а также в микросхемах EPROM. Поведение транзистора зависит от количества электронов на "плавающем" затворе. "Плавающий" затвор играет ту же роль, что и конденсатор в DRAM, т. е. хранит запрограммированное значение. Помещение заряда на "плавающий" затвор в такой ячейке производится методом инжекции "горячих" электронов (CHE - channel hot electrons или HEI -hot-electron injection) , а снятие заряда осуществляется методом квантомеханического туннелирования Фаулера-Нордхейма (Fowler-Nordheim [FN] tunneling).



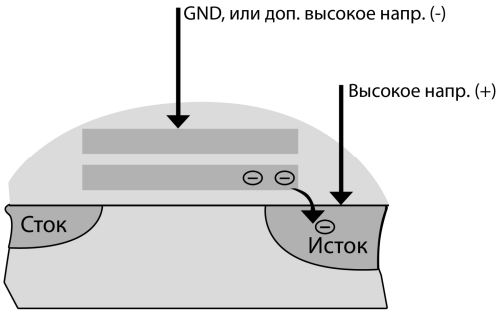
При чтении, в отсутствие заряда на "плавающем" затворе, под воздействием положительного поля на управляющем затворе, образуется n-канал в подложке между истоком и стоком, и возникает ток.



Наличие заряда на "плавающем" затворе меняет вольт-амперные характеристики транзистора таким образом, что при обычном для чтения напряжении канал не появляется, и тока между истоком и стоком не возникает.



При программировании на сток и управляющий затвор подаётся высокое напряжение (причём на управляющий затвор напряжение подаётся приблизительно в два раза выше). "Горячие" электроны из канала инжектируются на плавающий затвор, накапливаются, и изменяют вольт-амперные характеристики транзистора. Такие электроны называют "горячими" за то, что обладают высокой энергией, достаточной для преодоления потенциального барьера, создаваемого тонкой плёнкой диэлектрика.
Такой механизм переноса заряда используется в достаточно большом количестве микросхем флэш памяти, но параллельно с ним стал применятся эффект туннелирования Фаулера-Норхейма, ранее применявшийся только для стирания.



При стирании высокое напряжение подаётся на исток. На управляющий затвор (опционально) подаётся высокое отрицательное напряжение. Электроны туннелируют на исток.

Инжекция «горячих электронов» - процесс переноса заряда через энергетический барьер, образованный тонким слоем диэлектрика, за счет увеличения кинетической энергии электронов в канале между истоком и стоком ячейки.

Эффект туннелирования - один из эффектов, использующих волновые свойства электрона. Сам эффект заключается в преодолении электроном потенциального барьера малой "толщины". Для наглядности представим себе структуру, состоящую из двух проводящих областей, разделенных тонким слоем диэлектрика (обеднённая область). Преодолеть этот слой обычным способом электрон не может - не хватает энергии. Но при создании определённых условий (соответствующее напряжение и т.п.) электрон проскакивает слой диэлектрика (туннелирует сквозь него), создавая ток.

Важно отметить, что при туннелировании электрон оказывается "по другую сторону", не проходя через диэлектрик.

Эффект тунелирования Фаулера-Нордхейма – переход электронов в плавающий затвор при смещении потенциального барьера электрическим полем. Поле возникает при приложении разности потенциалов между управляющим затвором (-) и истоком (+).

Различия методов тунеллирования Фаулера-Нордхейма (FN) и метода инжекции "горячих" электронов:

Channel FN tunneling - не требует большого напряжения. Ячейки, использующие FN, могут быть меньше ячеек, использующих CHE.

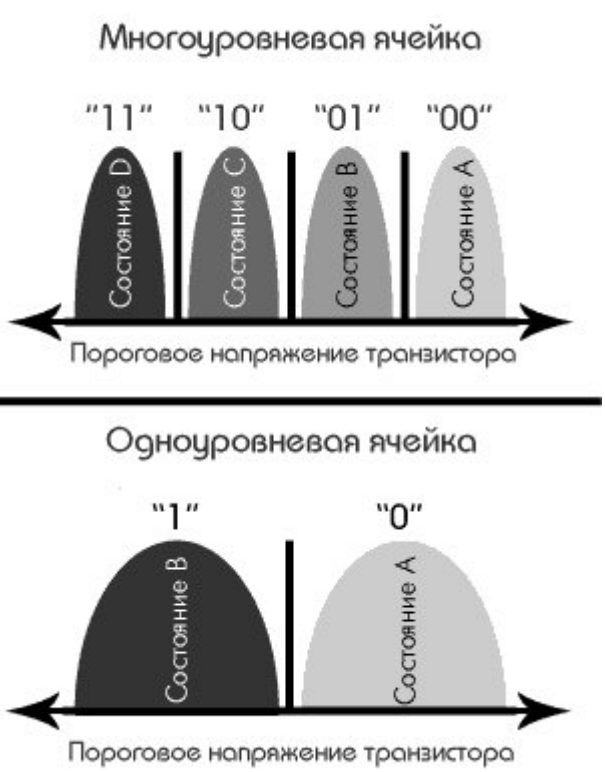
CHE injection (CHEI) - требует более высокого напряжения, по сравнению с FN. Таким образом, для работы памяти требуется поддержка двойного питания.

Программирование методом CHE осуществляется быстрее, чем методом FN.

Следует заметить, что, кроме FN и CHE, существуют другие методы программирования и стирания ячейки, которые успешно используются на практике, однако два описанных нами применяются чаще всего.

Процедуры стирания и записи сильно изнашивают ячейку флэш-памяти (неравномерности в структуре диэлектрика приводят к возникновению локальных областей, со значительной неоднородностью поля, что вызывает появление «перегруженных» участков, а соответственно ускорению процессов из износа – нарушение структуры и изменения сопротивления), поэтому в новейших микросхемах некоторых производителей применяются специальные алгоритмы, оптимизирующие процесс стирания-записи, а также алгоритмы, обеспечивающие равномерное использование всех ячеек в процессе функционирования. При возникновении неоднородности в слое диэлектрика повышенный ток в них приводит к появлению областей с локальным перегревом, что является причиной большей деградации, а также порождает «ток утечки», препятствуя свободному перемещению зарядов.

3.4 Многоуровневые ячейки (MLC - Multi Level Cell).



В последнее время многие компании начали выпуск микросхем флэш-памяти, в которых одна ячейка хранит два бита. Технология хранения двух и более бит в одной ячейке получила название MLC (multilevel cell - многоуровневая ячейка). Существуют типы микросем памяти, хранящих 4 бита в одной ячейке. В настоящее время многие компании находятся в поисках предельного числа бит, которое способна хранить многоуровневая ячейка.

В технологии MLC используется аналоговая природа ячейки памяти. Как известно, обычная однобитная ячейка памяти может принимать два состояния - "0" или "1". Во флэш-памяти эти два состояния различаются по величине заряда, помещённого на "плавающий" затвор транзистора. В отличие от "обычной" флэш-памяти, MLC способна различать более двух величин зарядов, помещённых на "плавающий" затвор, и, соответственно, большее число состояний. При этом каждому состоянию в соответствие ставится определенная комбинация значений бит.

Во время записи на "плавающий" затвор помещается количество заряда, соответствующее необходимому состоянию. От величины заряда на "плавающем" затворе зависит пороговое напряжение транзистора. Пороговое напряжение транзистора можно измерить при чтении и определить по нему записанное состояние, а значит и записанную последовательность бит.

Теоретически возможно прочитать и записать в ячейку более 4 бит, но на практике это сопряжено с проблемами устранения шумов, а также постепенной утечкой электронов при продолжительном хранении.

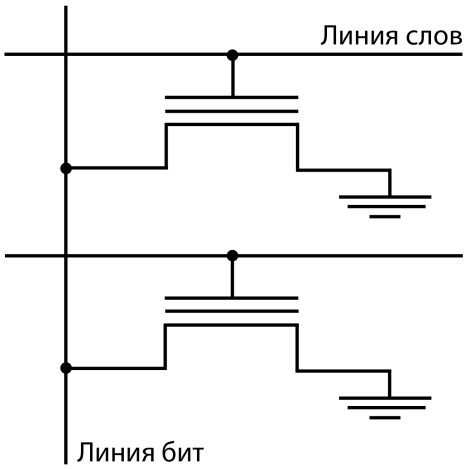
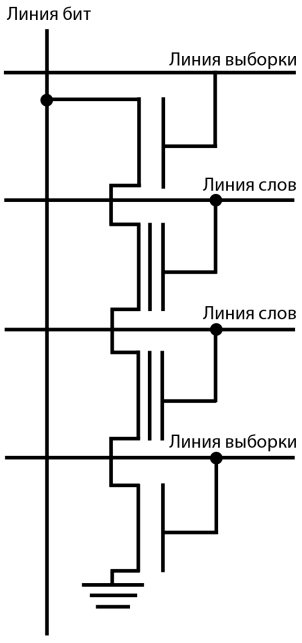
Технология многоуровневых ячеек от Intel (для NOR-памяти) носит название StrtaFlash, аналогичная от AMD (для NAND) – MirrorBit

Параметр	SLC	MLC	Комментарий
Логическая емкость	малая	Большая	
Скорость	Высокая	Низкая	
ECC	1 бит на 528 байт	4 бита на 528 байтов	Требования к кодам исправляющим ошибки(ECC) выше, т.к. в большей степени подвержены возникновению ошибок
Количество циклов записи/стирания	~100 000	~10 000	
Стоимость памяти в	Высокая	Низкая	

пересчете на 1 бит			

3.5 Архитектура флэш-памяти.

Существует несколько типов архитектур (организаций соединений между ячейками) флэш-памяти. Наиболее распространёнными в настоящее время являются микросхемы с организацией NOR и NAND.

NOR (NOT OR, ИЛИ-НЕ)	
	Ячейки работают сходным с EPROM способом. Интерфейс параллельный. Произвольное чтение и запись. Преимущества: быстрый произвольный доступ, возможность побайтной записи. Недостатки: относительно медленная запись и стирание. Из перечисленных здесь типов имеет наибольший размер ячейки, а потому плохо масштабируется. Единственный тип памяти, работающий на двух разных напряжениях. Идеально подходит для хранения кода программ (PC BIOS, сотовые телефоны), идеальная замена обычному EEPROM.
	Основные производители: AMD, Intel, Sharp, Micron, Ti, Toshiba, Fujitsu, Mitsubishi, SGS-Thomson, STMicroelectronics, SST, Samsung, Winbond, Macronix, NEC, UMC.
	Программирование: методом инжекции "горячих" электронов Стирание: туннеллированием FN
NAND (NOT AND, И-НЕ)	
	Доступ произвольный, но небольшими блоками (наподобие кластеров жёсткого диска). Последовательный интерфейс. Не так хорошо, как AND память подходит для задач, требующих произвольного доступа. Преимущества: быстрая запись и стирание, небольшой размер блока. Недостатки: относительно медленный произвольный доступ, невозможность побайтной записи. Наиболее подходящий тип памяти для приложений, ориентированных на блочный обмен: MP3 плееров, цифровых камер и в качестве заменителя жёстких дисков.
	Основные производители: Toshiba, AMD/Fujitsu, Samsung, National
	Программирование: туннеллированием FN Стирание: туннеллированием FN

3.6 Доступ к флэш-памяти

Существует три основных типа доступа:

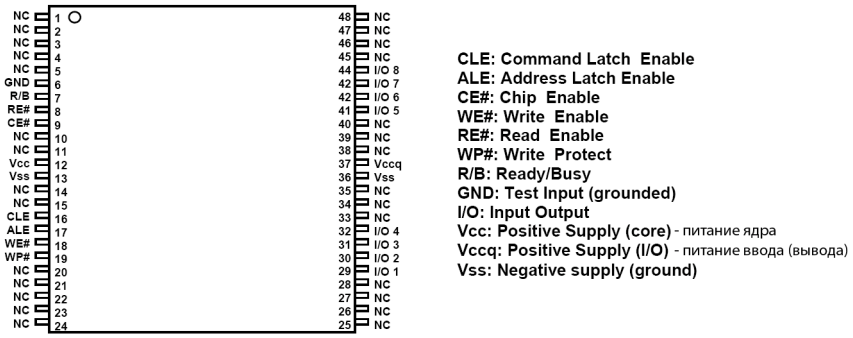
- обычный (Conventional): произвольный асинхронный доступ к ячейкам памяти.

- пакетный (Burst): синхронный, данные читаются параллельно, блоками по 16 или 32 слова. Считанные данные передаются последовательно, передача синхронизируется. Преимущество перед обычным типом доступа - быстрое последовательное чтение данных. Недостаток - медленный произвольный доступ.

- страничный (Page): асинхронный, блоками по 4 или 8 слов. Преимущества: очень быстрый произвольный доступ в пределах текущей страницы. Недостаток: относительно медленное переключение между страницами.

3.7 Интерфейс микросхем NAND флэш-памяти

Назначение выводов стандартной NAND флэш микросхемы памяти в корпусе TSOP-48 показано на рисунке ниже.



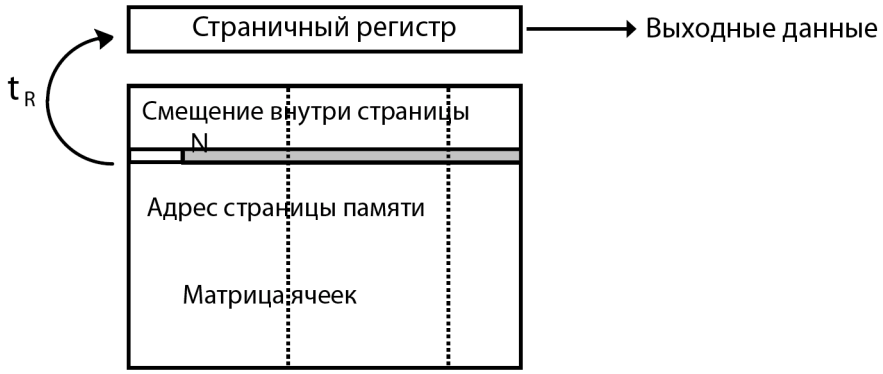
Суть интерфейса достаточно проста. Когда на выводе #CE присутствует логический «0», флэш записывает байт с линий данных при наличии логического «0» на выводе #WE (Write Enable), или выдает байт данных на шину при наличии логического «0» на выводе #RE (Read Enable). Когда на #CE присутствует высокий уровень, флэш игнорирует любые состояния выводов # RE и #WE, при этом выводы шины адрес-данные находятся в высокоимпедансном («третьем») состоянии. Сигналы CLE(Command Latch Enable) и ALE (Address Latch Enable) предназначены для управления внутренним мультиплексором, определяя какой из внутренних регистров будет подключен к внешним линиям. Для этих сигналов существует всего 3 корректные комбинации:

ALE	CLE	Выбранный регистр
0	0	Регистр данных
0	1	Регистр команды
1	0	Регистр адреса
1	1	Неопределенная комбинация

Все операции чтения/записи с флэш основаны на таком понятии, как страница, минимальная единица информации при обмене данными с флеш-микросхемой. Оправданность и рациональность применения страничной идеологии следует из сути NAND технологий, давшей название данному типу памяти. Наиболее часто встречаются размеры страницы размером 528 и 2112 байт. Суть размера страницы – это разрядность внутреннего регистра данных флэш-микросхемы. Операции стирания во флэш-памяти основана на блочной основе. Блок – совокупность страниц, число которых достаточно сильно различается между производителями флэш-памяти, но всегда является степенью 2-ки. Таким образом, для NAND флэш–памяти определены только 3 базовые операции: чтение страницы, программирование страницы, стирание блока.

3.7.1 Чтение страницы.

При операции чтения страницы микросхемы флэш-памяти данные страницы переносятся из памяти в регистр данных для выдачи на внешние линии. При этом осуществляется следующая последовательность действий:



3.7.1.1 Фаза команды:

На линии управление подается следующая комбинация : CLE = 1, ALE =0. При этом на линии данных выставляется значение «00» (наиболее часто используемый код команды). После этого на линию #WE, которая в неактивном состоянии равна «1» подается отрицательный импульс. Эта последовательность действий приведет к записи в командный регистр команды «read mode 1».

3.7.1.2 Фаза адреса:

На линии управления подается комбинация: CLE = 0, ALE =1. При этом на линии данных выставляется первый байт адреса требуемой страницы памяти. После этого на линию #WE подается отрицательный импульс. При этом будет выполнена запись первого адресного байта в внутренний регистр адреса. Данный байт определяет номер столбца (N column - Ncol), то есть сдвиг от начала страницы, начиная с которого будут выдаваться данные.

В зависимости от объема флэш-микросхемы, разрядность номера столбца может различаться. Обычно значение номера столбца равно «0» для того, чтобы чтение выполнялось от начала страницы. Но также возможно установить это значение в диапазоне от 0 до размера страницы. Если протокол флэш-микросхемы предусматривает только 1 байт для определения смещения от начала страницы, т.е. в диапазоне от 0 до 255. То, так как размер страницы минимум 528 байт, для данной микросхемы будет существовать дополнительные команда чтения со смещения 256 в странице (для микросхем Toshiba код команды «0x01») и команда чтение со смещение 512 (для микросхем Toshiba код оманды «0x50»). Следует учитывать, что при любой команде страница считывается из памяти в регистр полностью. Применение не нулевого значения номера колонки совместно с любой из команд чтения просто сдвигает указатель на данные внутри самого регистра. Адресные байты, следующие за номером колонки называются «номером строки» (N row), определяют положение страницы внутри блока и адресуют блок внутри микросхемы памяти. В зависимости от объема флэш-микросхемы, число байт, определяющих адрес строки, может различаться.

3.7.1.3 Фаза передачи данных.

На линии управления CLE и ALE подаются логические «0». При этом микросхема памяти находится в состоянии подготовки данных «BUSY». В течение этого периода сигнал на линии «BUSY» соответствует логическому «0». Среднее время подготовки дынных порядка 25 микросекунд. За это время данные будут прочитаны из массива ячеек памяти и помещены в регистр данных. В течении этого периода времени у некоторых микросхем необходимо удерживать сигнал на линии #CE в состоянии логического «0», но для большинства микросхем состояние сигнала #CE в течении данного промежутка времени не имеет решающей роли.

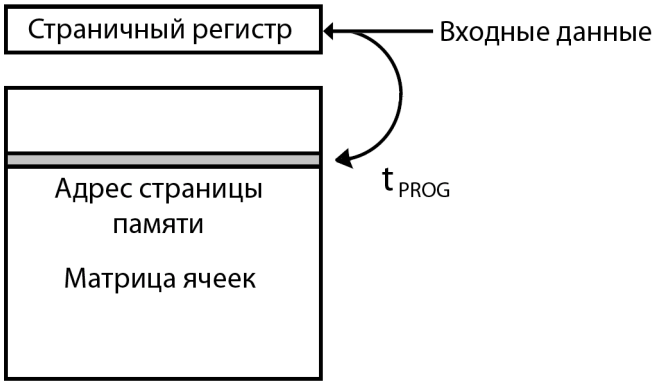
3.7.1.4 Фаза вычитывания данных.

Переход линии #R/B в состояние «1» является признаком наличия в регистре данных информации, доступной для считывания. Первым байтом при считывании будет байт, расположенный по смещению, определенным номером столбца при подаче команды. Каждый импульс сигнала #RE приводит к появлению на

внешней шине данных следующего байта в регистре. После чтения последнего байта страницы, микросхемы (если поддержка подобной операции описана в спецификации данной микросхемы памяти) автоматически переходит в состояние «BUSY» (занято), для считывания из массива ячеек следующей страницы. При этом дополнительной команды подавать не нужно. С документации подобный алгоритм чтения называется «последовательное чтение» (sequential read).

3.7.2 Программирование страницы.

При выполнении операции программирования страницы данные, равные размеру страницы, переносятся во внутренний регистр данных, после чего программируются в массив ячеек памяти.



3.7.2.1 Фаза команды:

На линии управления подается следующая комбинация: CLE =1 ,ALE =0. При этом на линии данных выставляется код команды записи. После этого на линию #WE, которая в неактивном состоянии равна «1» подается отрицательный импульс. Эта последовательность действий приведет к записи в командный регистр команды «serial data input».По этой команде также сбрасывается внутренний регистр дынных в состояние «все 1», т.е. заполняется 0xFF.

3.7.2.2 Фаза адреса.

На линии управления подается комбинация: CLE = 0, ALE =1. На линии данных выставляется первый байт адреса требуемой страницы памяти. После этого на линию #WE подается отрицательный импульс. При этом будет выполнена запись первого адресного байта в внутренний регистр адреса. Данный байт определяет номер столбца (N column - Ncol), то есть сдвиг от начала страницы, начиная с которого будут размещаться данные.

В зависимости от объема флэш-микросхемы разрядность номера столбца может различаться. Обычно, значение номера столбца равно «0» для того, чтобы запись выполнялось от начала страницы. Но также возможно установить это значение в диапазоне от 0 до размера страницы. Следует учитывать, что при любой команде страница переписывается из регистра в ячейки памяти полностью. Применение не нулевого значения номера колонки совместно с любой из команд чтения просто сдвигает указатель начального смещения данных внутри самого регистра. Тем не менее, при появлении на шине данных команды программирование «0x10» память будет переписана содержимым регистра данных, при этом так как неиспользованные байта регистра были сброшены в состояние «1», то запись их в ячейки памяти не приведет к изменению состояния ячеек.

Как и в операции чтения, адресные байты, следующие за номером колонки, называются номером строки (N row) и определяют положение страницы внутри блока и адресуют блок внутри микросхемы памяти. В зависимости от объема флэш-микросхемы, число байт, определяющих адрес строки, может различаться.

3.7.2.3 Фаза передачи данных.

На управляющие линии подается комбинация : CLE = 0, ALE =0. После этого данные побайтно записываются во внутренний регистр. При попытке записи в регистр данных, объемом превышающим размер страницы (а соответственно и внутреннего регистра), последний байт регистра будет содержать последний записанный байт.

3.7.2.4 Фаза программирования.

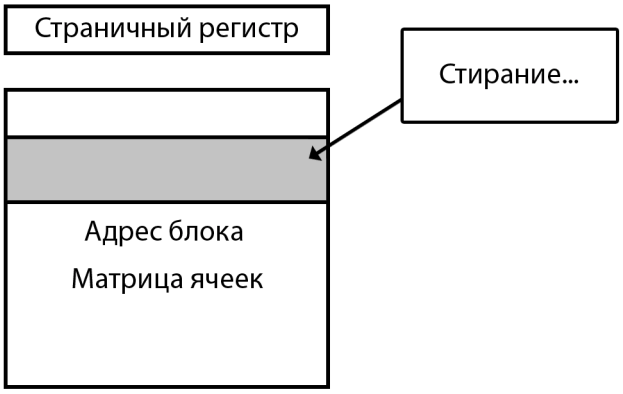
На управляющие линии подается комбинация: CLE =1, ALE =0. При этом на линии данных выставляется код команда «auto programm» (0x10), которая переноситься в командный регистр при появлении на линии #WE отрицательного импульса. Устройство переходит в состояние «BUSY» (занято) на время порядка 250 мкс. В течение этого от периода, даже если сигнал на линии #CE перейдет в состояние «1», операция программирования будет завершена.

3.7.2.5 Фаза проверки статуса (TimeOut Check Phase)

На данной фазе выполняется контроль корректности предыдущей операции записи. При чтении статуса в определенном бите байта статуса (в зависимости от типа флэш-микросхемы) может быть возвращена информация о возможной защите устройства от записи и о результате верификации страницы после записи. Если подобная ошибка возникает, блок, которому принадлежит данная страница, должен быть помечен как «BAD».

3.7.3 Стирание блока.

При выполнении операции «стирание блока» группа последовательных страниц (размер группы зависит от собственных особенностей флеш – памяти, таких как объем микросхемы, тип ячеек памяти и др.) стирается за одну операцию. В противоположность операции программирования, которая переводит биты в ячейках из состояния «1» в состояние «0», операция стирания осуществляет обратный перевод –из состояния «0» в состояние «1». В микросхемах при первом использовании все корректные блоки находятся в состоянии «стерт».



3.7.3.1 Фаза команды.

На линии управления подается следующая комбинация: CLE =1 ,ALE =0. При этом на линии данных выставляется код команды стирания. После этого на линию #WE, которая в неактивном состоянии равна «1» подается отрицательный импульс. Эта последовательность действий приведет к записи в командный регистр команды «auto block erase».

3.7.3.2 Фаза адреса.

При наличие на линиях CLE и ALE состоянии «0», в адресный регистр необходимо записать адрес блока, который необходимо стереть.

3.7.3.3 Фаза стирания.

На линии управления подается комбинация: CLE =1, ALE =0. При этом на линии данных подается код команды «auto block erase confirm», которая переноситься в командный регистр при появлении на линии #WE отрицательного импульса. Устройство переходит в состояние «BUSY» (занято) на время порядка 2 мс на блок. В течение этого от периода, даже если сигнал на линии #CE перейдет в состояние «1», операция стирания будет завершена.

3.7.3.4 Фаза проверки статуса.

Аналогично команде программирования, после выполнения операции стирания рекомендуется выполнить проверку статуса, чтобы убедиться, что в процессе не возникло состояния внутреннего тайм аута. Если при проверке статуса, микросхема возвращает признак ошибки верификации, то данный блок необходимо пометить как «BAD» (сбойный) , т.к. до того, как случился внутренний тайм аут, внутренний алгоритм флэш-микросхемы, попробовал выполнить операцию несколько раз.

3.8 Эффект износа ячеек в микросхемах флэш-памяти

3.8.1 Износ ячеек (Wear Leveling)

NAND флэш-память, в отличие от магнитных носителей, имеет ограниченное число циклов записи/стирания, и со временем подвержена явлению «износа», в результате чего не может быть больше использована. По той причине, что большинство файловых систем проектировалось для использования на магнитных носителях или носителях с аналогичными характеристиками, в них присутствуют элементы, характеризующиеся определенным физическим расположением, которые могут быть многократно перезаписаны. Например, в файловой системе FAT, область таблиц FAT и каталогов модифицируется значительное число при перезаписи или добавлении файла. Когда подобная файловая система содержит значительное число файлов, в области носителя, содержащей таблицы FAT и каталоги, операции стирания/записи выполняются намного чаще, чем в любой другой области.

Когда флэш-память используется как эмулятор магнитного диска, физические области микросхемы, содержащие таблицы FAT и директории, будет «изнашиваться» в первую очередь, приводя к ранним ошибкам в файловой системе, реализованной на флэш-микросхеме. Контроллер флэш-накопителя, реализуя алгоритм контроля равномерности «износа» ячеек памяти (программно или аппаратно), осуществляет запись блоков данных по всему доступному объему флэш-памяти, выполняя трансляцию одного логического адреса в различные физические адреса для каждой транзакции записи на диск. В общем случае, подобная таблица трансляции логического адреса в физический существует и модифицируется в ОЗУ контроллера, и инициализируется при подаче питания по уникальному для каждого отдельного контроллера алгоритму, путем сканирования физических блоков в микросхеме NAND флэш, на предмет определения присутствия каждого в

логической структуре образа диска. Данная таблица может периодически сохраняться на флэш-микросхеме либо в специализированной области, недоступной при трансляции обращения к диску, либо в произвольном месте в пределах всего доступного пространства флэш микросхемы. В последнем случае месторасположение такой таблицы или элементов структуры таблицы маркируется уникальным для каждого конкретного контроллера способом.

В отличие от накопителя на магнитных носителях, скорость выборки чтения блока не зависит от их физического расположения в микросхеме, что позволяет реализовывать достаточно сложные алгоритмы трансляции логического адреса в физический. В большей части накопителей данная функция распределения данных реализуется встроенным программным обеспечением контроллера (firmware), а в картах памяти типа SM (Smart Media) и xD, возлагается на программное обеспечение устройства, в котором данная карты применяется.

Общим недостатком всех алгоритмов трансляции является уязвимость самих таблиц трансляции, т.к. в процессе модификации данных в них при работе алгоритма контроля равномерности износа, таблицы должны постоянно обновляться. Но в отличие от остальных областей памяти, данные таблицы привязаны к месту расположения, т.к. они должны быть постоянно доступны контроллеру. Таким образом, блоки, содержащие таблицы трансляции, подвергаются более интенсивному износу, чем блоки, содержащие пользовательскую информацию. Для уменьшения износа в данном случае применяются специальные методы, но их вариации значительно ограничены по отношению к числу возможных реализаций алгоритма для пользовательских данных. Тем не менее, повреждение или потеря доступа к данным таблицам, в зависимости от принципов функционирования контроллера, ведет либо к искажению доступа к пользовательским данным, либо к полной потере доступа. В общем случае, применения специализированных утилит, можно восстановить работоспособность контроллера, если он не поврежден, но это вероятнее всего приведет к форматированию микросхемы памяти и уничтожению данных пользователя.

Таким образом, в случае возникновения ошибок, вызванных износом ячеек памяти в области пользовательской информации, это может привести к повреждению файлов или к логическому повреждению файловой системы, в зависимости от места возникновения ошибки. При таком типе повреждений накопитель определяется в системе и для его восстановления пригодны любые методы логического восстановления диска. В случае же сбоя в служебной области страницы, будет полностью утерян доступ к пользовательским данным (эквивалентно повреждению контроллера), и применение стандартных способов восстановление логической структуры будет не возможно.

В идеальном случае контроль равномерности износа должен быть реализован посредством применения в накопителях, основанных на флэш-памяти, специализированных файловых систем. Существует некоторое количество новых файловых систем, которые формируют запись новых последовательностей данных даже при модификации одной фиксированной области. Данные файловые системы применяют технику известную как journaling. Их таких для флэш-памяти: JFFS2 (Journaling Flash File System 2) и YAFFS (Yet Another Flash File System), которые автоматически контролируют равномерность износа, распределяя запись блоков данных последовательно в свободные области флэш-памяти.

3.8.2 Ошибки флэш-памяти

3.8.2.1 Идентификация Ошибочных блоков (Bad Block).

Спецификация NAND флэш допускает существование определенного процента (~2 %) Bad блоков уже на этапе производства. Список Bad блоков (или таблица Bad блоков) должны формироваться устройством, использующим данную флэш-микросхему, самостоятельно. Подобная таблица может храниться в одном из корректных блоков данной микросхемы, в другой микросхеме или ОЗУ устройства. Таблицу Bad блоков необходимо вести также и по той причине, что NAND флэш-микросхемы, в отличие от магнитных носителей имеют ограниченное число циклов стирания/записи. Поэтому все флэш-микросхемы со временем подвержены износу и не могут после этого использоваться. В соответствии с этим таблицы Bad блоков должны поддерживать возможность добавления блоков, в которых ошибки возникают в процессе использования.

Допущение существования Bad блоков позволяет снизить требования к качеству производства NAND флэш-памяти и, соответственно, снизить её себестоимость. Наличие Bad блоков не оказывает никакого эффекта на остальные блоки, так как каждый блок является независимой индивидуальной единицей, изолированной от

битовой линии транзисторным ключом выбора блока. Но, учитывая высокую вероятности появления bad блоков, во флэш-микросхеме предусмотрен дополнительный объем.

При производстве микросхем флэш-памяти обнаруженные в результате выходного контроля Bad блоки могут маркироваться для поддержки автоматического формирования таблиц Bad блоков электронными устройствами. Например, в микросхемах Toshiba, байт по смещению 0x205 (517) равен 0x00 на каждой странице в составе Bad блока. Аналогичный формат маркирования имеют микросхемы SmartMedia.

Производитель определяет ошибочный блок, путем выполнения серии тестов с различный паттерном (шаблоном) по всему объему флэш-микросхемы при жестких температурных условиях и предельных напряжениях.

Причины появления ошибки в блоке могут быть разными (ошибка декодирования, ошибка линии слов, собственные ошибки ячеек памяти), поэтому при появлении ошибок блок должен быть исключен из работы.

Появление Bad блоков происходит в процессе работы флэш-памяти. Микросхемы флэш-памяти имеют конечное время жизни и подвержены износу. Так как каждый блок является независимым, то он может быть отдельно стерт, причем данная операция не повлияет на время жизни соседних блоков. Для NAND памяти каждый хороший блок может быть стерт и перепрограммирован обычно более чем от 100 000 до 1 000 000 раз до появления ошибок. В настоящее время данная величина постоянно повышается. Более точные данные необходимо искать в документации на конкретную микросхему.

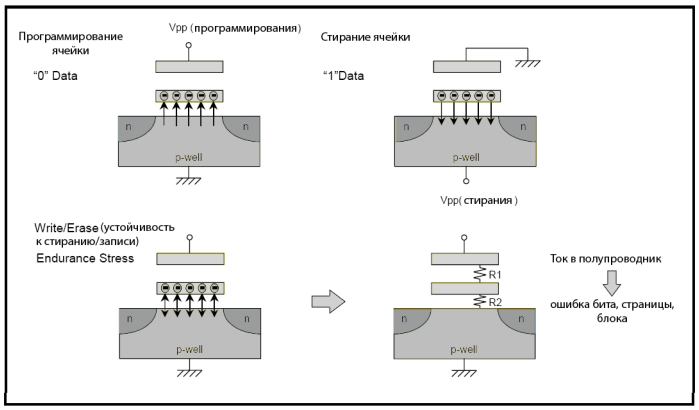
Одной из основных причин явления износа ячеек является накопление избыточного заряда в слое полупроводника, что приводит к возрастанию времени стирания ячейки, до момента превышения предельного значения. Время программирования при повышении износа ячеек увеличивается мене значительно, поэтому вероятность появления первой ошибки программирования очень мала. Обычно, только при очень серьезных внутренних повреждениях или отклонениях показателей от нормы возможно появление ошибок программирования.

3.8.2.2 Типы «периодических, постоянных», «допустимых» ошибок. Механизм. Симптомы.

Если в процессе работы микросхемы памяти происходят ошибки бита на страницу, то это не является причиной идентифицировать блок как Bad блок. Блок считается сбойным при возникновении в нем ошибки стирания или программирования. Можно отметить следующие типы ошибок.

«Сопротивляемость»(устойчивость) к стиранию/ программированию. (Write/Erase Cycle Endurance)

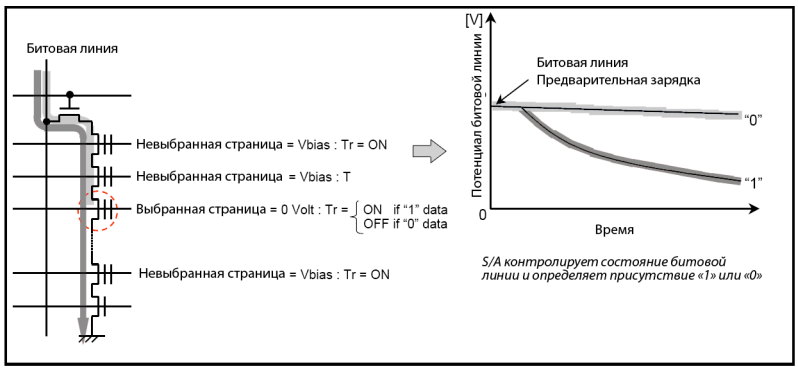
Подобная ошибка может проявляться как ошибка ячейки, страницы, блока при чтении статуса после соответствующей операции.



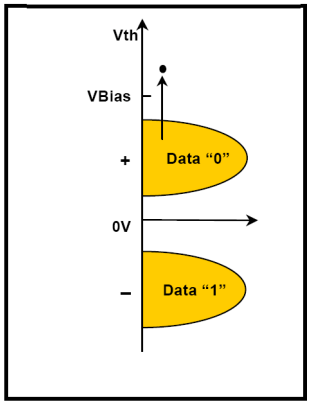
Суть ошибки в том, что в слое полупроводника присутствуют свободные электроны, которые могут оседать на плавающем затворе стертой ячейки, приводя к неоднозначным результатам чтения ячейки, либо самопроизвольному «стеканию» заряда с затвора (аналог тока в полупроводнике), что также приводит к ошибкам верификации.

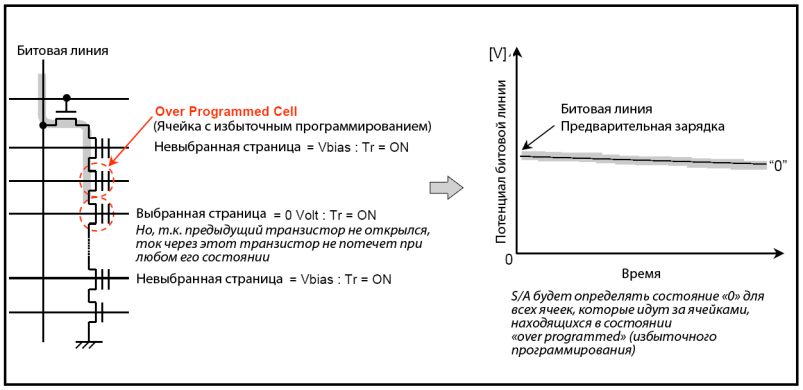
Ошибки перепрограммирования (Over Programming).

Данная ошибка возникает, когда пороговое напряжение транзистора, соответствующее состоянию «0» ячейке памяти начинаем значительно превышать номинальное, в результате превышения тока программирования. В нормальном состоянии все пороговые напряжения ячеек ниже некоторого напряжения смещения, (V bias), поэтому подача напряжения V bias на невыбранные страницы в текущей транзакции чтения приводит к открытию соответствующих транзисторов.



Если пороговое напряжение одной из ячеек в цепочке начинает превышать напряжение V bias, то подачи номинального напряжения смещения будет недостаточно для открытия транзистора, поэтому ячейка никогда не «откроется».





Несмотря на то, что ошибки происходят при выполнении операции программирования, обнаружить их можно только посредством операции чтения. Симптомом данной ошибки является чтение «0» на всех ячейках одной битовой линии (или, что эквивалентно, «0» в одном и том же бите каждой страницы внутри блока), при условии, что на данной линии предполагается наличие «1». Т.е. эквивалентно возникновению однобитовой ошибки на каждой странице в пределах блока. Сбойная ячейка восстанавливает свое состояние после стирания блока

Сбой программирования (Program Disturb).

Данная ошибка характеризуется ситуацией, когда бит данных самопроизвольно перепрограммируется из состояния «1» в состояние «0» в процессе программирования страницы. Подобная ошибка может происходить как при программировании текущей страницы, так и при программировании последующих страниц блока. Колебания напряжения смещение в блоке при программированiь страницы становятся причиной появления тока в ячейках памяти. Многократные попытки программирования при возникновении ошибки приводят к ухудшению ситуации. Состояние ошибки, возникшее в ячейке памяти, может быть сброшено при стирании блока, содержащего данную ячейку. Вероятность ошибки повышается при операции произвольно программирования страниц в блоке. Поэтому в документации на микросхемы NAND флешь памяти существует требование последовательного программирования страниц блока (От младшего адреса к старшему).

3.8.3 Код коррекции ошибок ECC (Error correcting Code).

Применение в NAND флэш технологии ECC диктуется требованиями необходимости контроля четности накапливаемых данных. Программные ошибки (особенно возникающие в процессе программирования) происходят с частотой примерно 10^{-10}, что эквивалентно 1 биту на 10 billion запрограммированных бит. Достаточным для NAND флэш считается Hamming code, корректирующий однобитовые ошибки и детектирующий 2 битовые, а также алгоритм Рида-Соломона, который стал стандартным методом кодирования как для NOR так и для NAND флэш памяти.

Собственно выявлением bad секторов и занимается алгоритм ECC – при записи он сравнивает записываемую информацию с реально записанной. В случае возникновения ошибки, записываемые данные перемещаются с резервную область, а место, где была зафиксирована ошибка помечается как bad.