Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана» (МГТУ им. Н.Э.Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Написание дефрагментатора диска для Windows»

Студент ИУ9-52 _____ Атымханова М.
(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ Коновалов А.В.
(Подпись, дата) (И.О.Фамилия)

2018 г.

Оглавление

Введение. Постановка задачи.

При начальной установке операционной системы все нужные ей программы и файлы устанавливаются последовательно с самого начала диска, каждый новый каталог следует за предыдущим. За установленными файлами единым непрерывным участком следует свободное пространство. Но со временем по мере создания и удаления файлов диск обычно приобретает нежелательную фрагментацию, где повсеместно встречаются файлы и области свободного пространства. [1]

Это происходит при записи файла на диск, когда возникает ситуация, когда на диске нет участка памяти объема пригодного для размещения файла в непрерывную последовательность единиц памяти, таким образом файл записывается в свободные участки диска расположенные в разных местах, тем самым фрагментируясь(разбиваясь на фрагменты). Такие ситуации и таким образом размещенные файлы замедляют процесс считывания и записи с/на диск. Чем больше фрагментированных файлов располагается на диске, тем более сложно обращаться с ним. Такой диск подлежит дефрагментации.

Дефрагментация — это процесс перемещения на диске блоков памяти принадлежащих файлу в последовательную цепочку его сегментов.

Целью данного курсового проекта было написание приложения, реализующего процедуру дефрагментации жесткого диска.

Приложение состоит из двух компонентов: библиотеки динамической компоновки (DLL) для работы с низкоуровневыми функциями операционной системы и приложения графического пользовательского интерфейса.

Библиотека реализует функции обращения к памяти такие как считывание карты диска, считывания структур размещения файлов, их перемещения, процедуру дефрагментации. Языком реализации был выбран C++. Функции осуществляющие системные вызовы используют библиотеку <Windows.h> и Windows API дефрагментации.

Для реализации графического пользовательского интерфейса была выбрана графическая подсистема Windows Presentation Foundation в составе .NETFramework с реализацией на языке C#.

Чтобы код приложения написанного на C# мог вызывать функции из библиотеки DLL, написанной на C++, нужные функции библиотеки должны экспортироваться с использованием соответствующего соглашения о вызовах и без декорирования имен. Взаимодействие между компонентами .NET Framework и неуправляемым программным кодом было реализовано через функции пространства имен System.Runtime.InteropServices и сервиса вызова неуправляемых функций из управляемого кода(Platform Invocation Services).

Одна из основных функций приложения — процедура дефрагментации в соответствие с реализованным алгоритмом.

Работа над курсовым проектом была разделена на четыре этапа:

- Изучение API дефрагментации
- Реализация графического пользовательского интерфейса
- Реализация алгоритма дефрагментации
- Тестирование программы

1. Основные понятия файловой системы и API дефрагментации

1.1. Файловая система

Файловая система – способ организации данных в виде файлов на устройствах внешней памяти (жестких и оптических дисках, устройствах флеш-памяти и т. п.).

Файловая система должна обеспечивать:

- безопасное и надежное хранение данных
- программный интерфейс доступа к файлам;
- организацию файлов в виде иерархии каталогов.

Windows поддерживает несколько файловых систем для различных внешних устройств:

NTFS – основная файловая система семейства Windows NT;

FAT (File Allocation Table – таблица размещения файлов) –Существует три варианта FAT, отличающихся разрядностью идентификаторов, указывающих размещение файлов: FAT12, FAT16 и FAT32;

Файловая система оперирует понятиями диск, раздел, простые и составные тома, сектор, кластер.

Диск (disk) –устройство внешней памяти, например, жесткий диск или оптический диск.

Раздел (partition) – непрерывная часть жесткого диска. Диск может содержать несколько разделов.

Том (volume) или логический диск (logical disk) – область внешней памяти, с которой операционная система работает как с единым целым. Тома бывают простые и составные.

Понятия раздела и простого тома отличаются: понятие "раздел" связано с физическим устройством, а понятие "том" – с логическим представлением внешней памяти.

Сектор (sector) –блок данных фиксированного размера на диске; наименьшая единица информации для диска. Типичный размер сектора для жестких дисков равен 512 байтам.

Кластер (cluster) – логический блок данных на диске, включающий один или несколько секторов. Количество секторов, составляющих кластер, обычно кратно степеням двойки. Размер кластера задается операционной системой в процессе высокоуровневого форматирования, которое может осуществляться многократно.

При записи на диск файл всегда будет занимать целое число кластеров. Например, файл размером 100 байт в файловой системе с размером кластера 4 КБ будет занимать ровно 4 КБ.

Выбор размера кластера связан со следующими соображениями. Малые кластеры позволяют сократить размер фактически неиспользуемого дискового пространства, возникающего за счет размещения файла в целом числе кластеров. Но при этом общее количество кластеров на диске увеличивается и размер служебных структур файловой системы, в которых хранится информация о файлах, возрастает.

Структура NTFS:

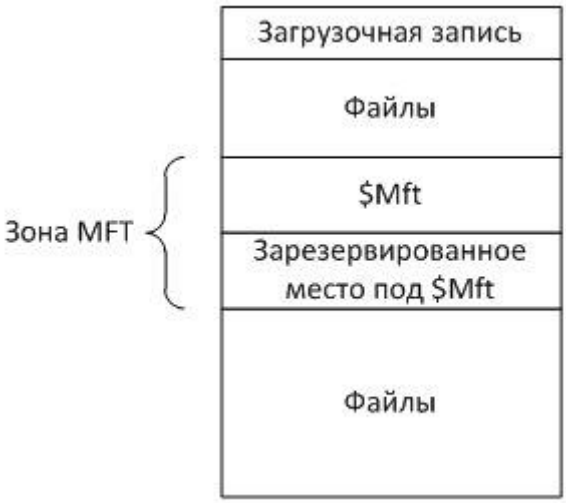


Рисунок 1 Структура NTFS

В начале тома находится загрузочная запись тома (VolumeBootRecord), в которой содержится код загрузки Windows, информация о томе (в частности, тип файловой системы), адреса системных файлов. Загрузочная запись занимает обычно 8 КБ (16 первых секторов).

В определенной области тома (адрес начала этой области указывается в загрузочной записи) расположена основная системная структура NTFS– главная таблица файлов (Master File Table, MFT). В записях этой таблицы содержится вся информация о расположении файлов на томе, а небольшие файлы хранятся прямо в записях MFT.

Поскольку *MFT* является важнейшей системной структурой, к которой при операциях с томом наиболее часто происходят обращения, выгодно хранить файл $Mft в непрерывной области логического диска, чтобы избежать его фрагментации (размещения в разных областях диска), и, следовательно, повысить скорость работы с ним. С этой целью при форматировании тома выделяется непрерывная область, называемая зоной MFT (MFTZone). По мере увеличения главной таблицы файлов, файл $Mft расширяется, занимая зарезервированное место в зоне.

Остальное место на томе NTFS отводится под файлы – системные и пользовательские.

Главная таблица файлов MFT состоит из множества записей о файлах (файловых записей), расположенных на томе. Размер одной записи – 1 КБ (2 сектора). Самая первая запись в MFT– это запись о самом файле $Mft. Во второй записи содержится информация о файле $MftMirr – зеркальной копии MFT. В этом файле дублируются первые 4 записи таблицы MFT, в том числе записьо $Mft. В случае возникновения сбоя, если MFT окажется недоступной, информация о системных файлах будет считываться из $MftMirr (в загрузочной записи имеется адрес $MftMirr).

Некоторые записи в таблице*MFT*:

$Volume – файл информации о томе, в котором содержатся имя тома (Volume label), версия NTFS и набор флагов состояния тома, например, флаг (т. н. грязный бит, dirty bit), установка которого означает, что том был поврежден и требует восстановления при помощи системной утилиты Chkdsk;

$BitMap – файл битовой карты (bitmap), каждый бит в этой карте соответствует кластеру на томе: если бит равен 1, кластер занят, иначе – свободен;

В данном курсовом проекте зона MFT не подвергается дефрагментации.

Файл NTFS:

Основная информация о файле содержится в файловой записи (FileRecord) размером 1 КБ таблицы MFT, а небольшие файлы целиком хранятся в файловой записи.

Рисунок 2 Файл NTFS

Для обозначения кластеров файла используются два типа номеров: LCN и VCN. При помощи первого типа, LCN (Logical Cluster Number – логический номер кластера), нумеруются все кластеры на диске, от первого до последнего. LCN применяется, чтобы найти начальный кластер группы. Номера VCN (Virtual Cluster Number – виртуальный номер кластера) обозначают порядковый номер кластера внутри группы.

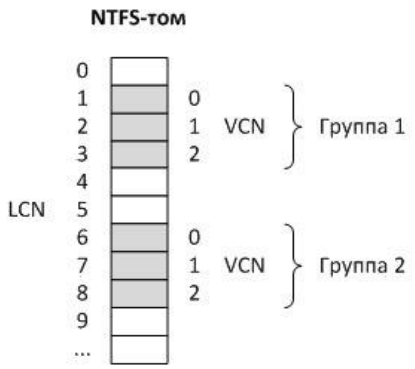


Рисунок 3 LCN, VCN кластеры файла

Процесс разбиения файла на небольшие фрагменты при записи на диск называется фрагментацией. Если на диске много фрагментированных файлов, скорость чтения носителя уменьшается, поскольку поиск кластеров, в которых хранятся файлы, требует времени.

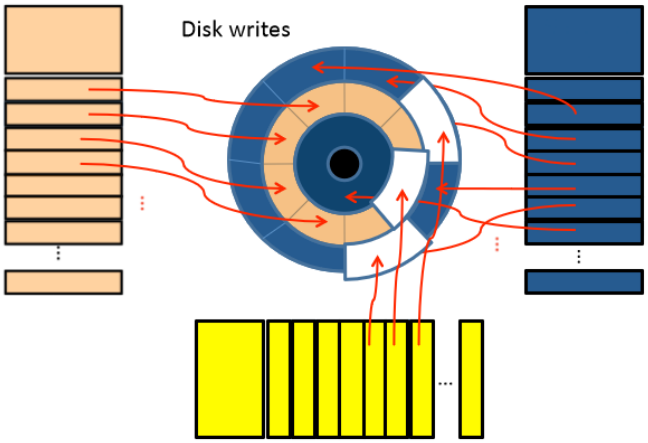


Рисунок 4 Иллюстрация фрагментации диска в процессе одновременной записи нескольких файлов

1.2. API дефрагментации

Дефрагментация— процесс обновления и оптимизации логической структуры раздела диска с целью обеспечить хранение файлов в непрерывной последовательности кластеров. Применяется в основном в отношении файловых систем FAT и NTFS. После дефрагментации ускоряется чтение и запись файлов, а, следовательно, и работа программ. Другое определение дефрагментации: перераспределение файлов на диске, при котором они располагаются в непрерывных областях.

Длинные файлы занимают несколько кластеров. Если запись производится на незаполненный диск, то кластеры, принадлежащие одному файлу, записываются подряд. Если диск переполнен, на нем может не быть цельной области, достаточной для размещения файла. Тем не менее, файл все-таки запишется, если на диске много мелких областей, суммарный размер которых достаточен для записи. В этом случае файл записывается в виде нескольких фрагментов.

Процедура дефрагментации:

Дефрагментатор диска взаимодействует с файловой системой для перемещения файлов и отображения свободного места, что приводит к улучшению производительности ввода-вывода. Когда фрагментированный файл запрашивается через вызов ввода-вывода, головка диска должна перемещаться на разные дорожки на диске, чтобы прочитать все фрагменты.

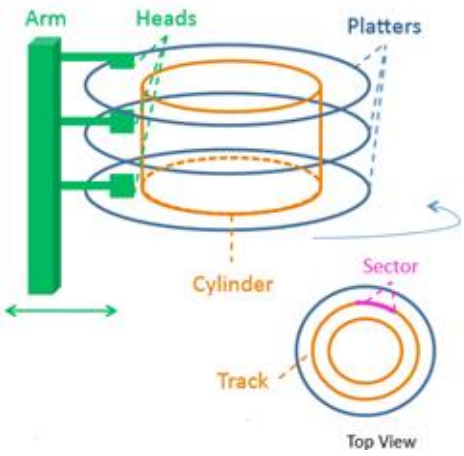


Рисунок 5 Физическая структура диска

Файл, который распространяется по тысячам фрагментов, может потребовать тысячи вызовов ввода-вывода для контроллера диска и перегрузку перемещения головки диска для завершения операции чтения, что приводит к более низкой производительности. Чтобы уменьшить как вызовы ввода-вывода, так и движение головки, необходимых для чтения файла, дефрагментатор диска перемещает фрагменты файла в непрерывную последовательность кластеров на диске.

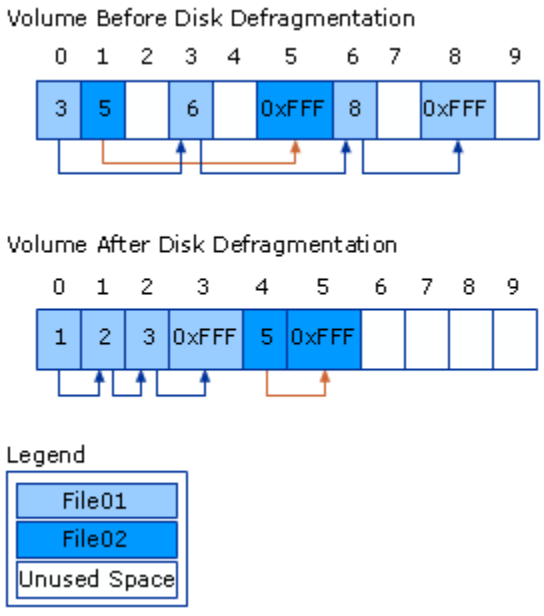


Рисунок 6 Файл до и после дефрагментации

Операционная система Windows имеет API, с помощью которого можно создать приложение дефрагментатор этой ОС. Главный API операционной системы — это множество системных вызовов.

3 управляющих кода для системного вызова DeviceIoControl, работа с которыми обеспечивает следующий функционал:

- поиск пустых кластеров
- определение местоположения на диске кластеров файла
- перемещение кластеров на диске

1.2.1. Управляющие коды

FSCTL_GET_VOLUME_BITMAP

Функция DeviceIoControl вызванная с данным управляющим кодом заполняет переданную по указателю в ее параметры структуру VOLUME_BITMAP_BUFFER, содержащую поля:

- стартовый логический номер кластера(LCN) (может быть округлен в меньшую сторону в зависимости от файловой системы)
- количество кластеров в томе начиная с LCN
- массив байтов, который представляет битовую карту кластеров тома, каждый бит которой, хранит 1 в случае занятости кластера файлом(или его резервированием), иначе 0.

Структура выполняет функционал отображения занятых и доступных кластеров на томе. Вместе с данным управляющим кодом в функцию DeviceIoControl передается дескриптор тома, для которого нужно составить битовую карту кластеров. Дескриптор тома является возвращаемым объектом функции CreateFile**,** вызванной с параметрами имени тома в формате "\\?\Volume{*GUID*}\", где GUID — глобальный уникальный идентификатор тома. Карта тома является актуальной не позднее привнесенных изменений в томе путем удаления/добавления файла, движением его фрагментированных частей. Обновление структуры VOLUME_BITMAP_BUFFER в алгоритме дефрагментации происходит после каждого изменения в томе.

FSCTL_GET_RETRIEVAL_POINTERS

Функция DeviceIoControl вызванная с данным управляющим кодом и дескриптором файла заполняет переданную по указателю в ее параметры структуру RETRIEVAL_POINTERS_BUFFER. Структура описывает размещение и расположение файла на диске, содержит поля:

- стартовый фактический номер кластера (Virtual Cluster Number — VCN) (может быть округлен в меньшую сторону в зависимости от файловой системы)

- массив структур "фрагмент" (Extent), содержащих поля из стартового номера следующего фактического фрагмента (NextVcn) и логического расположения текущего (Lcn).
- количество элементов в массиве Extents

FSCTL_MOVE_FILE

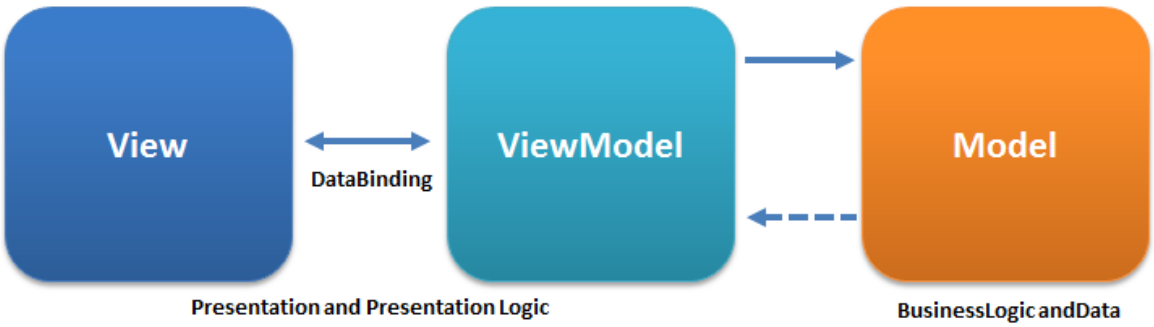
Перемещает один и более виртуальных(фактических) кластеров файла с одного логического кластера в другой внутри того же тома. Перемещение осуществляется путем вызова функции DeviceIoControl с параметром управляющего кода FSCTL_MOVE_FILE. В параметры входит также дескриптор тома внутри которого содержится файл, чьи кластеры подлежат перемещению, указатель на структуру MOVE_FILE_DATA, содержащую поля:

- дескриптор файла
- номер начального виртуального кластера файла с которого начинается перемещение
- номер логического кластера тома в который будет производится перемещение
- количество перемещаемых кластеров

2. Архитектура программы

Программа представляет собой приложение Windows Presentation Foundation(WPF) платформы .NET, которое выполняет функции графического интерфейса, и библиотеку динамической компоновки, которая предоставляет функции анализа тома диска, его дефрагментацию.

При реализации графического интерфейса в WPF применим шаблон Model-View-ViewModel(MVVM). Шаблон MVVM делится на три части:



Модель(*Model*), представляет собой бизнес логику и фундаментальные данные, необходимые для работы приложения.

Представление(*View*)— это графический интерфейс, то есть окно, кнопки и т.п. Представление является подписчиком на событие изменения значений свойств или команд, предоставляемых Моделью Представления. В случае, если в Модели Представления изменилось какое-либо свойство, то она оповещает всех подписчиков об этом, и Представление, в свою очередь, запрашивает обновленное значение свойства из Модели Представления. В случае, если пользователь воздействует на какой-либо элемент интерфейса, Представление вызывает соответствующую команду, предоставленную Моделью Представления.

Модель Представления(*ViewModel*) является, с одной стороны, абстракцией Представления, а с другой, предоставляет обёртку данных из Модели, которые подлежат связыванию. То есть, она содержит Модель, которая преобразована к Представлению, а также содержит в себе команды, которыми может пользоваться Представление, чтобы влиять на Модель.

Представлению посвящен следующий раздел.

Модель представляет из себя комплекс из двух библиотек: библиотеки динамической компоновки DefragLibrary.dll и библиотеки классов PInvokeLibrary.dll.

DefragLibrary.dll оперирует системными вызовами и цельно выполняет блоки операций в ходе анализа/дефрагментации диска в одном пространстве, на языке C++, который для CLR является неуправляемым кодом и не транслируется в CIL. Такой способ реализации был выбран в целях улучшения быстродействия и предпочтения такой архитектуры библиотеки для оперирования системными вызовами и управляющими кодами.

PInvokeLibrary.dll обеспечивает взаимодействие между DefragLibrary.dll и WPF приложением графического интерфейса, которое выполняется средой CLR и окружение программы является управляемым кодом. PInvokeLibrary.dll написана на C#; транслирует функции и структуры из DefragLibrary.dll с применением PInvoke и маршалинга структур в управляемый код.

Платформа .NET Framework предоставляет функции вызова неуправляемого кода (PInvoke) с помощью атрибута Dllimport, что позволяет управляемым приложениям вызывать неуправляемые функции, поставляемые в библиотеках DLL.

Модель представления описывает вызовы необходимых функции и их обработку в зависимости от функционала кнопки/элемента выбора тома диска и последующую отрисовку в представлении.

2.1. Реализация графического интерфейса программы

Графический интерфейс программы реализован как приложение Windows Presentation Forms. WPF предназначается для создания динамических, управляемых данными систем представления данных.Каждая

часть системы предназначена для создания объектов с помощью наборов свойств, которые определяют их поведение. Привязка данных является основополагающей частью системы и интегрирована на каждом уровне.

Традиционные приложения создают отображение, а затем привязывают его к некоторым данным.В WPF все, что касается элемента управления, каждый аспект отображения, создается посредством некоторого типа привязки данных. Текст внутри кнопки отображается путем создания составного элемента управления внутри кнопки и путем привязки его отображения к свойству содержимого кнопки.

При углублении в архитектуру WPF обнаруживается, что можно создавать значительно более сложные приложения, которые фундаментально обрабатывают данные, как драйвер ядра приложения.

2.1.1. Архитектура WPF(Windows Presentation Foundation)

Основные компоненты WPF показаны на расположенном ниже рисунке. Красные разделы диаграммы (PresentationFramework, PresentationCore и milcore) представляют собой основные части кода WPF. Только один из этих компонентов является неуправляемым — milcore. Компонент milcore написан в неуправляемом коде, чтобы обеспечить тесную интеграцию с DirectX. Все отображения в WPF выполнены с помощью подсистемы DirectX, позволяющей эффективно использовать оборудование и программное обеспечение для визуализации. Для WPF также необходим тонкий контроль над памятью и выполнением. Механизм композиции в milcore крайне чувствителен к производительности и требует отказа от многих преимуществ CLR в пользу производительности.

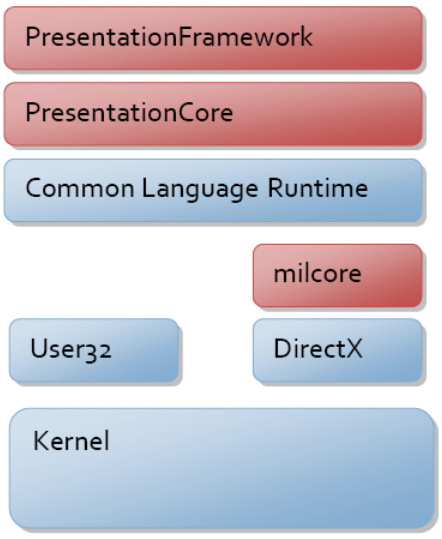


Рисунок 7 Архитектура WPF

Разработка приложения WPF включает:

- Определение разметки XAML для разработки приложения пользовательского интерфейса.
- Написание кода для построения модели поведения приложения.

- Создание управляющих определений приложения.
- Добавление элементов управления и макета, входящих в состав интерфейса приложения.
- Создание стилей, обеспечивающих унифицированный внешний вид компонентов приложения.
- Связывание интерфейса с данными для заполнения и синхронизация данных с пользовательским интерфейсом.

2.1.2. Классы графического интерфейса

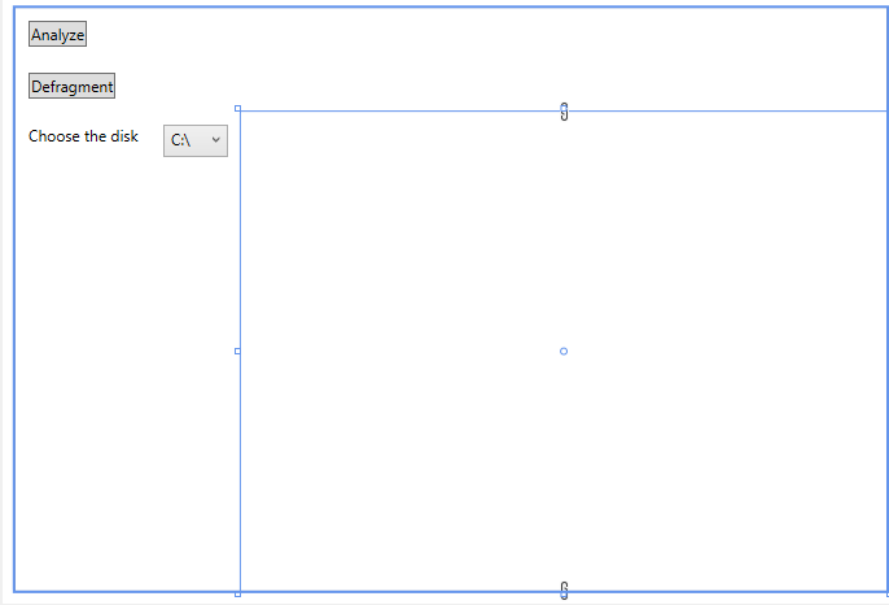


Рисунок 8 Шаблон представления

Представление(View) приложения дефрагментатора состоит из набора элементов

Grid : **System.Windows.Controls.Panel**

Элемент объекта **System.Windows.Controls**

Задание области с таблицей переменного размера, состоящей из столбцов и строк.

Пространство имен **System.Windows.Controls** предоставляет классы для создания элементов, известных как элементы управления, которые позволяют пользователю взаимодействовать с приложением. Классы элементов управления являются основой пользовательского взаимодействия с приложением, так как они позволяют пользователю просматривать, выбирать или вводить данные или другую информацию.

DockPanel : **System.Windows.Controls.Panel**

Элемент объекта**System.Windows.Controls**

Определяет область, в которой можно упорядочивать дочерние элементы горизонтально либо вертикально относительно друг друга.

ComboBox : **System.Windows.Controls.Primitives.Selector**

Элемент объекта**System.Windows.Controls**

Представляет элемент управления для выбора с раскрывающимся списком, который можно отображать и скрывать, щелкая стрелку в элементе управления.

В данном приложении выполняет функционал выбора раздела диска.

Button : **System.Windows.Controls.Primitives.ButtonBase**

Элемент объекта **System.Windows.Controls**

Представляет элемент управления "Кнопка Windows", который будет реагировать на событие System.Windows.Controls.Primitives.ButtonBase.Click.

В данном приложении три элемента: первый вызывает в модели функцию анализа тома, второй вызывает функцию дефрагментации применительно к тому выбранному в элементе ComboBox, третий осуществляет останов процесса дефрагментации.

Canvas : **System.Windows.Controls.Panel**

Элемент объекта**System.Windows.Controls**

Определяет область, внутри которой можно явным образом разместить дочерние элементы с помощью координат, относящихся к области System.Windows.Controls.Canvas

Отрисовка карты диска производится в контексте этого объекта.

Все элементы объекта **System.Windows.Controls** наследуют по восходящей от объектов в иерархии классов.

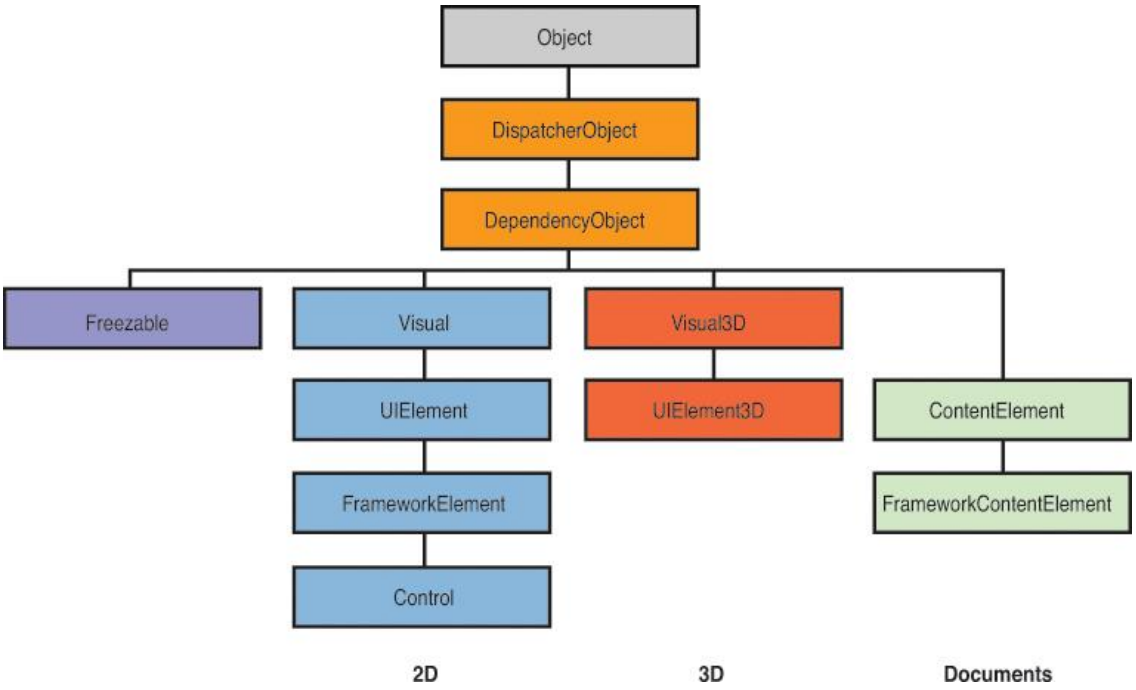


Рисунок 9 Иерархия классов WPF

А именно наследование производится от:

System.Threading.DispatcherObject

Большинство объектов в WPF произошли от DispatcherObject, который предоставляет базовые конструкции для работы с параллелизмом и поточностью.

System.Windows.DependencyObject

WPF предоставляет обширную систему свойств, полученную из DependencyObject. Система свойств является системой свойств "зависимостей" в том смысле, что она отслеживает зависимости между выражениями свойств и автоматически проверяет значения свойства при изменении зависимости.

System.Windows.Media.Visual

После определения системы свойств следующим шагом является рисование точек на экране. Класс Visual предоставляет средства для построения дерева визуальных объектов, которые дополнительно включают инструкции по рисованию и метаданные о способе визуализации этих инструкций (обрезка, преобразование и другие).

Класс Visual является реальной точкой входа в систему композиции WPF. Класс Visual является точкой соединения между двумя подсистемами, управляемым API и неуправляемым компонентом milcore.

WPF отображает данные, проходя по неуправляемым структурам данных под управлением milcore. Эти структуры, называемые узлами композиции, представляют собой иерархическое дерево отображения с инструкциями по визуализации в каждом узле. Это дерево, показанное в правой части расположенного ниже рисунка, доступно только через протокол обмена сообщениями.

При программировании WPF создаются элементы Visual и производные типы, которые осуществляют внутреннее взаимодействие с деревом композиции через этот протокол обмена сообщениями. Каждый элемент Visual в WPF может создать один, ни одного или несколько узлов композиции.

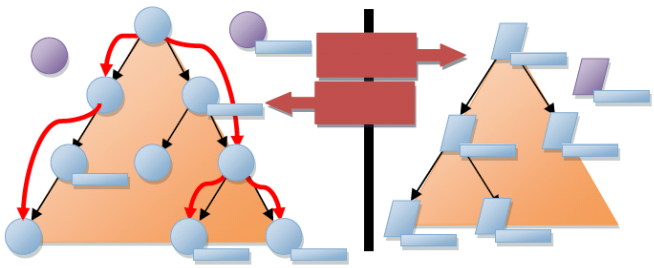


Рисунок 7 Визуальное дерево WPF. Пример

System.Windows.UIElement

UIElement определяет основные подсистемы, включая "Макет", "Ввод данных" и "События".

Макет представляет собой основное понятие в WPF.

Каждое событие ввода преобразуется, по крайней мере, в два события — событие "предварительного просмотра" и фактическое событие.

В плане архитектуры элемент UIElement может рассматриваться как приблизительный эквивалент дескриптора окна в программировании Win32 или как элемент в программировании Динамический HTML (DHTML). Элемент UIElementявляется базовым элементом уровня ядра WPF.

System.Windows.FrameworkElement

Предоставляет набор свойств, событий и методов для элементов Windows Presentation Foundation (WPF) на уровне среды WPF. Данный класс представляет указанную реализацию уровня среды WPF, построенную на основе API-интерфейсы уровня ядра WPF, определенного элементом UIElement.

FrameworkElement является точкой соединения между классами элементов уровня среды WPF и набора служб презентации UIElement уровня ядра WPF.

2.1.3. Отрисовка карты диска

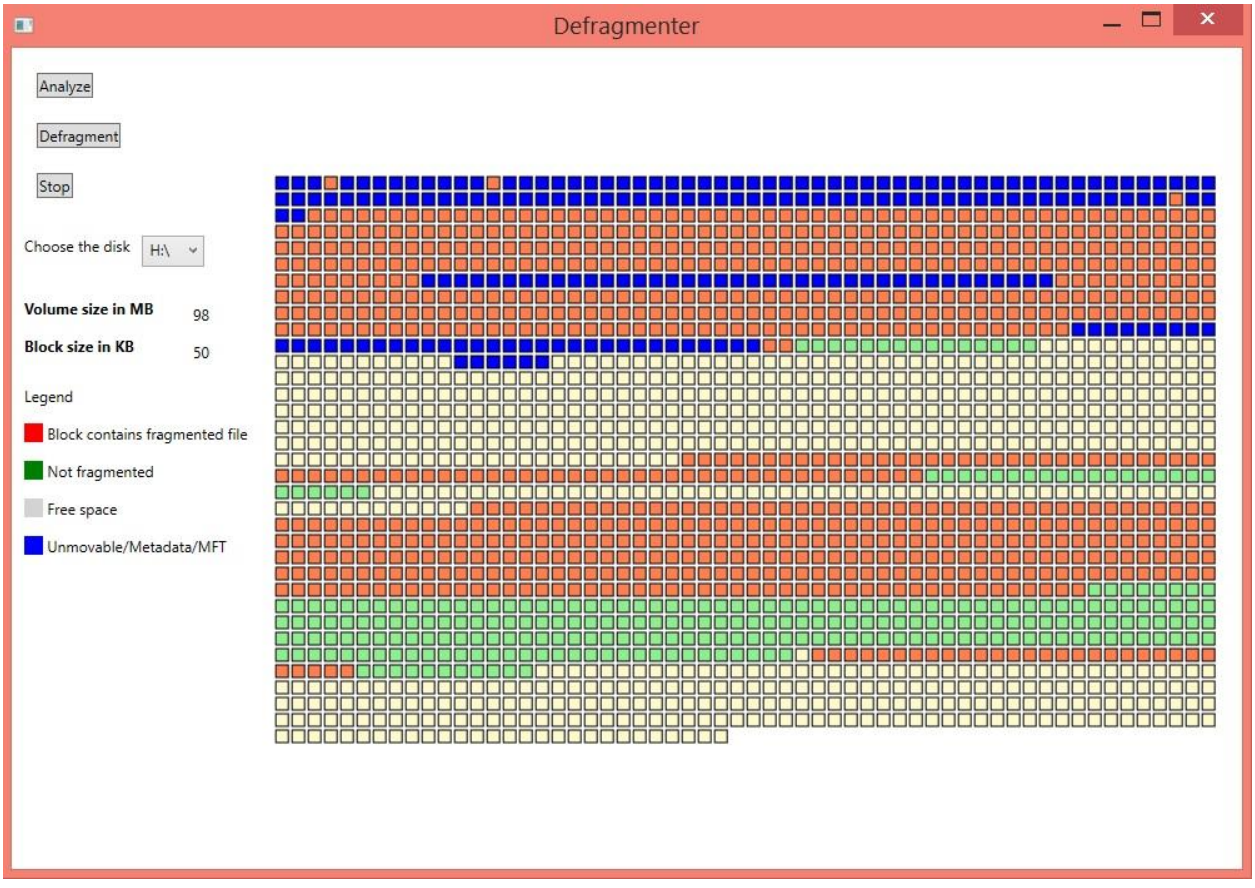


Рисунок 8 Карта фрагментированного раздела диска

Содержимым элемента Canvas на панели приложения становится коллекция прямоугольников, представленных объектами RectangleGeometry как разновидность объекта Geometry, переданных в контекст класса Path, который позволяет рисовать фигуры различного рода.

Множество прямоугольников отображает карту тома.

Каждый прямоугольник представляет некоторое количество расположенных подряд кластеров (их количество зависит от количества отображаемых прямоугольников и размера выбранного для анализа и отображения тома). Если эта область памяти отображаемая прямоугольником содержит в себе кластеры принадлежащие фрагментированному файлу, то он красного цвета; если в ней содержатся файлы метаданных/MFT(MasterFileTable) /память зарезервированная под MFT/неперемещаемые файлы, то синего цвета; если в нем содержатся только пустые кластеры, то он серого цвета, иначе если он содержит только кластеры нефрагментированных файлов, он отображается зеленым.

Элемент ComboBox позволяет выбрать том для дальнейших действий.

Три элемента Button представляют соответственно функцию анализа, дефрагментации и останова дефрагментации.

Отображаются размер выбранного тома и размер прямоугольника(Block).

2.2. Создание DLL

DLL представляет собой библиотеку функций (ресурсов), которыми может пользоваться любой процесс, загрузивший эту библиотеку.

Загрузившему DLL процессу доступны не все ее функции, а лишь явно предоставляемые самой DLL для "внешнего мира" — т.н.***экспортируемые***. Также существуют функции, предназначенные для "внутреннего" пользования. Чем больше функций экспортирует DLL — тем медленнее она загружается. Проектирование ***интерфейса*** — способ взаимодействия DLL с вызывающим кодом — описывается в следующих главах .

2.2.1. Архитектура

Динамически загружаемая библиотека(DLL) предполагает наличие секций импорта и экспорта. Секция экспорта указывает те идентификаторы объектов(функций, классов, переменных), доступ к которым предоставляет данная DLL. В общем случае именно секция экспорта представляет особый интерес для разработчиков.

Исполняемый код в DLL не предполагает автономного использования. Перед тем как можно будет приступить к использованию, необходимо загрузить DLL в область памяти вызывающего процесса. Это явление носит название«проецирование DLL на адресное пространство процесса». В конечном коде .exe файла, который генерирует компилятор, не будет инструкций процессора, соответствующих коду данной функции. Вместо

этого будет сгенерирована инструкция вызова соответствующей функции. Так как DLL отображена на адресное пространство процесса, то код DLL будет легко доступен по вызову. Т.о., DLL — особым образом оформленный программный компонент, доступ к исполняемому коду которого приложение получает в момент старта (DLL неявной загрузки).[3]

Основные направления использования DLL:

- всевозможные модули расширения функциональности приложений — так называемые plug-in ;
- локализация приложения;
- разделение объектов абстракции (функций, классов и проч.) между приложениями;
- независимость модификации кода — DLL может быть в любой момент переписана с сохранением экспортируемых интерфейсов;
- реализация определенных действий, которые можно совершить только при помощи DLL;
- хранилище ресурсов с возможностью независимого изменения этих ресурсов. [2]

В данной работе библиотека создавалась с целью организации API для вызова функций считывания разделов диска, анализа раздела, его дефрагментации, которые стали идентификаторами экспорта данной DLL. DLL реализована на C++ в среде Microsoft VisualStudio "VisualC++".

Так как DLL разрабатывалась на языке отличном от того, который используется при ее загрузке в GUI приложение, необходимо было задуматься о преодолении границ адресных пространств при передаче данных (параметров функций и т.д.) — так называемый marshalling. Этому посвящен раздел 2.3.2. Функционалу библиотеки посвящен следующий раздел.

2.2.2. Структуры и классы

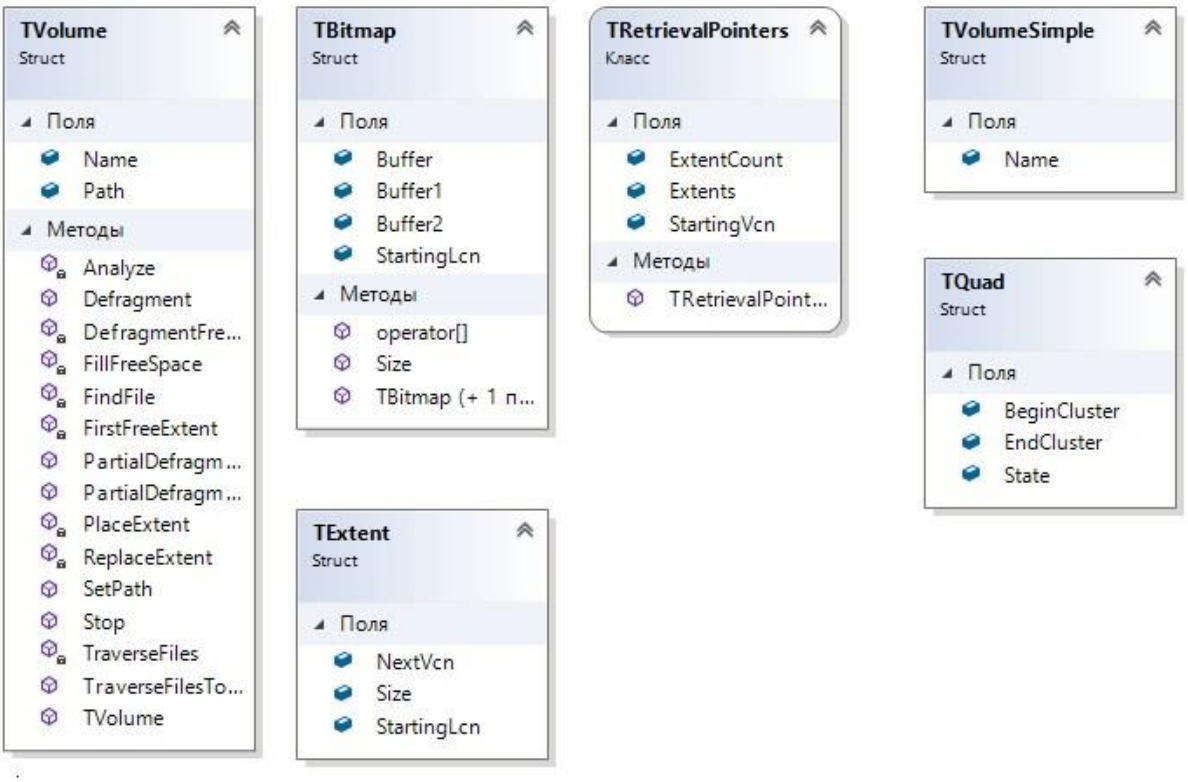


Рисунок 9 Классы и структурыDefragLibrary.dll

`struct TVolume`

Центральная структура проекта. Содержит поля описания имени тома, его GUID. Методы анализа тома, дефрагментации.

Функция `GetPathNames(constTVolume &Volume)` получает в параметры ссылку на данную структуру и возвращает путь к тому для инициализации этого поля.

Функция `GetVolumeHandle(TVolume Volume)` позволяет получить дескриптор тома и оперировать с ним.

`struct TVolumeSimple`

Массив упрощенных структур обозначения разделов на диске передается приложению для их отображения, выбора одного, анализа и дальнейших действий.

`struct TBitmap`

Структура схожая со структурой API дефрагментации `PVOLUME_BITMAP_BUFFER` выполненная в собственной реализации для удобства работы со структурой карты тома. Имеет конструктор с параметром `PVOLUME_BITMAP_BUFFER`.

Такую структуру возвращает функция `GetVolumeBitmap(HANDLEHandle)` от дескриптора структуры `TVolume`.

`struct TRetrievalPointers`

местный аналог структуры `PRETRIEVAL_POINTERS_BUFFER` (имеется конструктор копирования с такой структуры)

`struct TExtent`

Описывает непрерывный фрагмент файла, его стартовый логический номер кластера и фактический номер следующего фрагмента. Входит в состав `struct TRetrievalPointers:std::vector<TExtent> Extents` организует для файла обращение ко всем его фрагментам.

`struct TQuad`

Необходима по ту сторону .dll в количестве целого массива для обозначения кластеров начиная с `BeginClusterLcn` по `EndClusterLcn` в виде квадрата цвета определяемого полем `State`, которое принимает одно значение из множества: {кластеры пусты, кластеры занимаемы недефрагментированными файлами, хотя бы один кластер занимаем дефрагментированным файлом, кластеры принадлежат метаданным/месту зарезервированному под рост MFT/ MFT}.

Помимо классов DLL содержит и экспортирует следующие функции(для экспортирования функции из DLL — перед ее описанием следует указать ключевое слово`__declspec(dllexport)`).

```
extern "C" __declspec(dllexport)
TQuad *GetBlocks(TVolumeSimple VolumeName, uint32_t size);
```

Функция в спецификации вызова C с соответствующей сигнатурой, возвращает указатель на структуру `TQuad`.

Ключевое слово `extern` объявляет переменную или функцию и указывает, что она имеет внешние компоновки (ее имя будет видимым не только в пределах файла, в котором она определена, но и в других файлах).

Экспорт данных, функций, классов или классов функций-членов из библиотеки DLL можно осуществлять с помощью ключевого слова `__declspec(dllexport).` Ключевое слово `__declspec(dllexport)` добавляет директивы экспорта в объектный файл.

Атрибут `dllexport` функции предоставляет функцию с ее декорированным именем. Для функций C++ сюда входит видоизменение имени. Для функций C или функций, объявленных с модификатором `extern "C"`, сюда входит зависящее от платформы декорирование, основанное на соглашении о вызовах. К экспортированным функциям C и функциям `extern "C"` C++, использующим соглашение о вызова `__Cdecl`, не применяется декорирование имен.

Декорированное имя — это закодированная строка, созданная компилятором во время компиляции объекта, данных или определения функции. Оно содержит соглашения о вызовах, типы, параметры функции и другие сведения, а также имя. Такое дополнение имен, также известное как

искажение имени, помогает компоновщику найти правильные функции и объекты при компоновке исполняемого файла.

Если к функциям в библиотеках DLL, исходный код которых написан на языке C++, требуется доступ из модуля, написанного на другом языке(и наоборот), то эти функции следует объявить с помощью компоновки C, а не C++.

```
extern "C" __declspec(dllexport) TQuad *GetColouredBlocks(TValueString VolumeName, uint32_t size);
extern "C" __declspec(dllexport) uint64_t GetAllVolumesArray(int len, TVolumeSimple array[]);
extern "C" __declspec(dllexport) uint64_t GetAllVolumesArrayGUID(int len, TVolumeSimple array[]);
extern "C" __declspec(dllexport) uint64_t GetVolumeSizeInBytes(TValueString VolumeName);
extern "C" __declspec(dllexport) uint64_t DefragmentVolume(TVolumeSimple volume);
extern "C" __declspec(dllexport) uint64_t PartialDefr1(TValueString VolumeName, uint64_t rate);
extern "C" __declspec(dllexport) uint64_t PartialDefr2(TValueString VolumeName, uint64_t rate);
extern "C" __declspec(dllexport) void InterruptDefrag();
```

Рисунок 10 Объявление экспортируемых функций

2.3. Взаимодействие между компонентами

2.3.1. Platform Invocation Services (PInvoke)

Среда CLR предоставляет службы вызова неуправляемого кода (PInvoke), которые позволяют управляемому коду вызывать функции в формате C в библиотеках динамической компоновки (DLL), содержащих машинный код. Служба обнаруживает и вызывает экспортированную функцию и при необходимости маршалирует ее аргументы (целые числа, строки, массивы, структуры и так далее) через границы взаимодействия.

Маршалинг взаимодействия определяет, как данные передаются в аргументах и возвращаемых значениях методов между управляемой и неуправляемой памятью во время вызовов. Маршалинг взаимодействия — это процесс времени выполнения, выполняемый службой маршалинга среды CLR.

Рисунок 11 Вызов неуправляемой функции DLL с помощью вызова неуправляемого кода

Когда управляемый код вызывает неуправляемую функцию, то он выполняет следующую последовательность действий:

- Определяет библиотеку DLL, содержащую функцию.
- Загружает библиотеку DLL в память.
- Находит адрес функции в памяти и помещает ее аргументы в стек, выполнив по необходимости маршалинг данных.
- Передает управление неуправляемой функции.

```
public static class PInvoke
{
    [DllImport("DefragLibrary.dll", EntryPoint = "GetAllVolumesArray",
    CallingConvention = CallingConvention.StdCall)]
    private static extern int GetAllVolumesArrayInternal(
    int len, [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 0)] [In, Out] TVolume[] array);
    public static TVolume[] GetAllVolumes()...

    [DllImport("DefragLibrary.dll", EntryPoint = "GetColouredBlocks",
    CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern IntPtr GetColouredBlocksInternal([In] TVolume volume, UInt32 size);
    public static TQuad[] GetColouredBlocks(TVolume volume, uint size)...

    [DllImport("DefragLibrary.dll", EntryPoint = "PartialDefr1",
    CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    public static extern UInt64 DefragmentFirstStage([In] TVolume volume, UInt64 size);

    [DllImport("DefragLibrary.dll", EntryPoint = "InterruptDefrag",
     CallingConvention = CallingConvention.Cdecl)]
    public static extern void InterruptDefrag();
```

Рисунок 12 Пример импортируемых функций из DefragLibrary.dll

Вызов PInvoke выполняется с помощью атрибута `DllImport`. Этот атрибут принимает значение имени библиотеки DLL в качестве своего первого аргумента и помещается перед объявлением функции для каждой используемой в библиотеке DLL точки входа. Подпись функции должна соответствовать имени функции, экспортируемой из библиотеки DLL.

2.3.2. Маршалинг данных при вызове неуправляемого кода

Для вызова функций, экспортированных из неуправляемой библиотеки, приложению .NET Framework требуется прототип функции в управляемом коде, представляющий неуправляемую функцию. Чтобы создать прототип, который допускает вызов неуправляемого кода для правильного маршалинга данных, необходимо выполнить указанные ниже действия.

- Применить атрибут `DllImport` к статической функции или методу в управляемом коде.
- Заменить неуправляемые типы данных на управляемые.

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
public struct TVolume
{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 261)]
    public string Name;
}
[StructLayout(LayoutKind.Sequential)]
public struct TQuad
{
    public UInt64 State;
    public ulong BeginCluster, EndCluster;

    public override string ToString() => $"{BeginCluster} -- {EndCluster} [{State}]";
}
```

Рисунок 13 Прототипы управляемых структур для передачи аргументов и получения значений от функций, экспортируемых из неуправляемой библиотеки

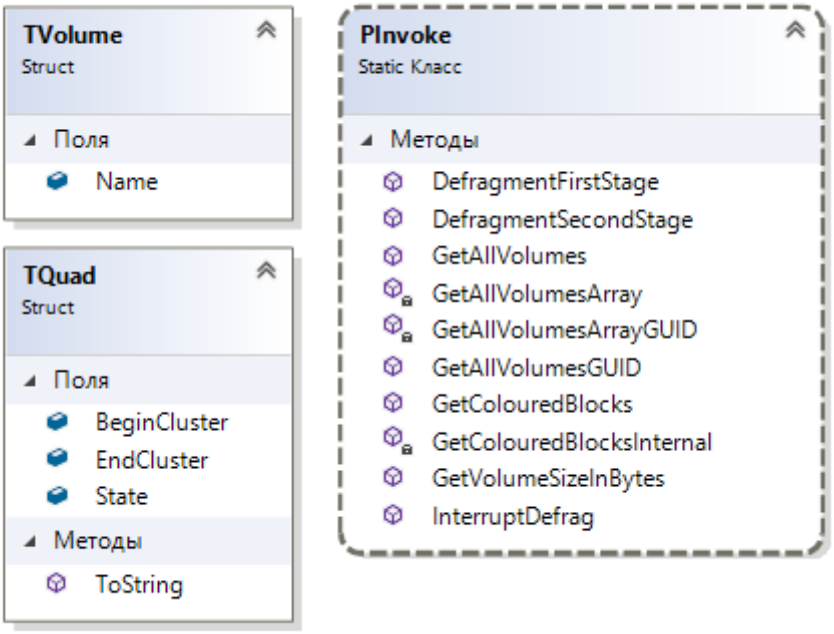


Рисунок 14 Структуры и класс PInvokeDefragLibrary.dll

3. Алгоритм дефрагментации

Реализованный в методе для структуры тома `TVolume::Defragment()` алгоритм состоит из двух функций: дефрагментации свободного места *(в размере половины всего свободного места раздела) к началу участка дефрагментируемой области и функции заполнения очищенного пространства размещением последовательно файлов/ фрагментов файлов/ продолжением незаконченного файла. Последовательное выполнение этих двух функций в цикле со смещением границы начала работы процедур на размер половины свободного места всего раздела до достижения последнего объема кластеров, покрывающего размер оставшихся для дефрагментации

файлов или их фрагментов на котором производится последний вызов процедур.

Вспомогательными для метода дефрагментации свободного места в начало задаваемое номером кластера и до правой границы

```
TVolume::DefragmentFreeSpace(int leftboundary, int rightboundary)
```

стали методы

```
TBitmapTVolume::Analyze()
```

(метод осуществляет вызов функции DeviceIoControl с управляющим кодом FSCTL_GET_VOLUME_BITMAP) возвращающий битмап, где каждый кластер помечен как непустой/ пустой/ метаданные и у непустого кластера содержащего файл помечено его имя (путь к нему), по которому функция

```
HANDLE GetFileHandle(const std::experimental::filesystem::path Path)
```

открывает дескриптор файла для получения структуры файла

```
TRetrievalPointers GetRetrievalPointersBuffer(HANDLE Handle)
```

(метод осуществляет вызов функции DeviceIoControl с управляющим кодом FSCTL_GET_RETRIEVAL_POINTERS). Так как возможность получить дескриптор и доступ можно только для ординарного файла, то как следствие алгоритм не оперирует ссылками/файлами других типов, что делает алгоритм неприменимым для дефрагментации системного раздела диска. Дальнейшая работа со структурой TRetrievalPointers осуществляется итерацией по экстентам и перемещением тех из них, которые находятся в области текущей дефрагментации свободного места в цепочку первых свободных участков памяти следующих за правой границей области. Это осуществляет метод

```
intTVolume::ReplaceExtent(HANDLEHandle, TExtentextent,
intboundary),
```

где Handle — дескриптор файла; метод осуществляет вызов функции DeviceIoControl с управляющим кодом FSCTL_MOVE_FILE.

Метод

```
TVolume::FillFreeSpace(intleftboundary,intrightboundary)
```

собирает фрагменты файлов в непрерывную последовательность в область задаваемую границами и предварительно освобожденную от кластеров других файлов. Изначально метод определяет закончена ли целиком запись предыдущего файла или же стоит продолжать перемещать фрагменты незаконченного на предыдущей итерации файла. В первом случае вызывается функция поиска файла все кластеры которого размещаются после правой границы свободной области, так как слева от нее располагаются уже целиком дефрагментированные файлы. После того как файл найден перемещаются все его экстенты суммарный размер которых подходит для выделенной области; последний частично вместимый экстент может быть фрагментирован на два участка и первый из них будет записан в область, а второй будет добавлен к нему в ходе следующей итерации; при наличии

свободного места производится поиск следующего файла. В заданном диапазоне кластеров могут размещаться фрагменты неперемещаемых файлов, что учитывается в рассматриваемом методе. Алгоритм допускает фрагментацию файла только прерыванием размещения его кластеров кластерами неперемещаемых файлов.

В результате работы алгоритма все файлы записаны последовательно, некоторые из них не фрагментированы, остальные фрагментированы участками неперемещаемых файлов. Все свободное место дефрагментировано в конец тома.

4. Тестирование

В этом разделе приводятся два примера работы процедуры дефрагментации.

Первый пример демонстрирует работоспособность алгоритма дефрагментации за приемлемое время раздела диска объемом 98MB с малым количеством свободного места.

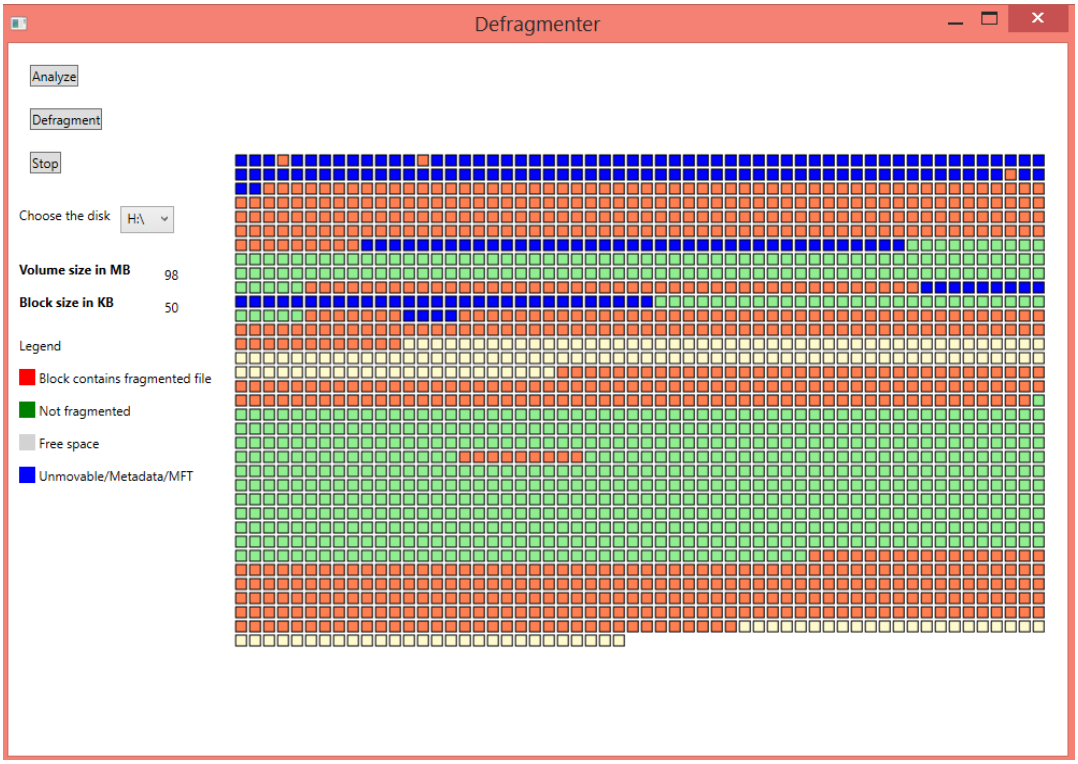


Рисунок 15 Карта раздела диска до дефрагментации

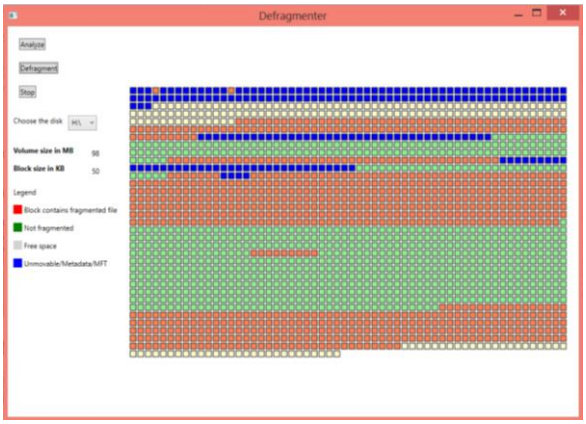


Рисунок 16 Карта раздела в процессе дефрагментации

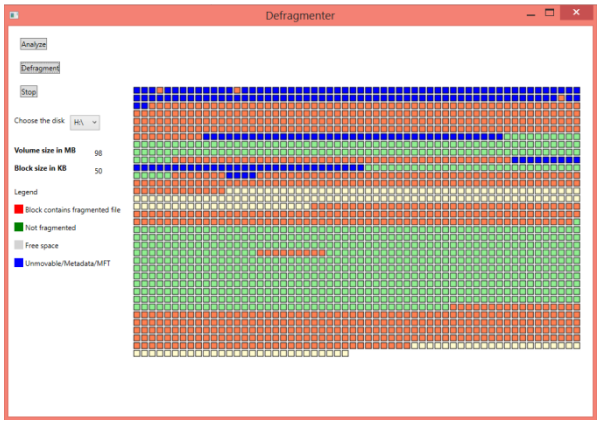


Рисунок 17 Карта раздела в процессе дефрагментации

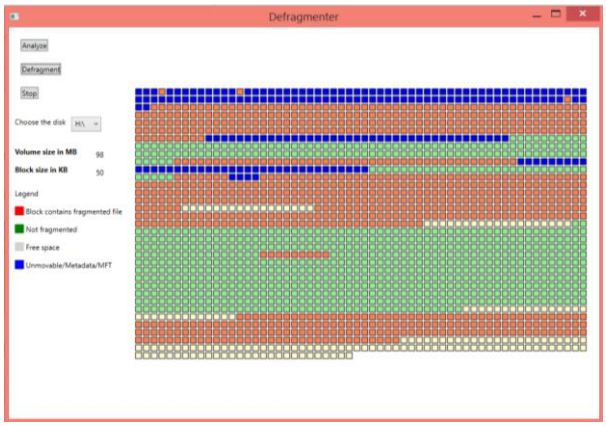


Рисунок 18 Карта раздела в процессе дефрагментации

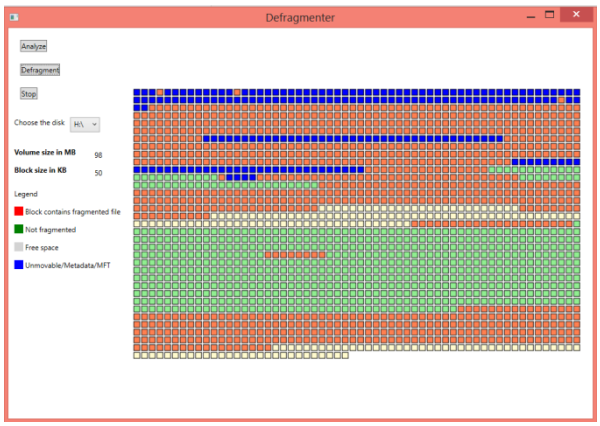


Рисунок 19 Карта раздела в процессе дефрагментации

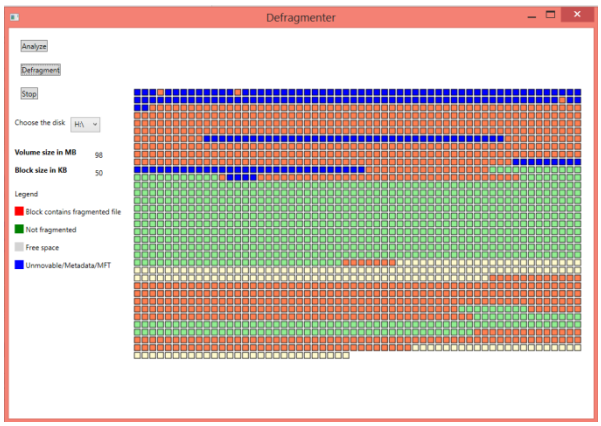


Рисунок 20 Карта раздела в процессе дефрагментации

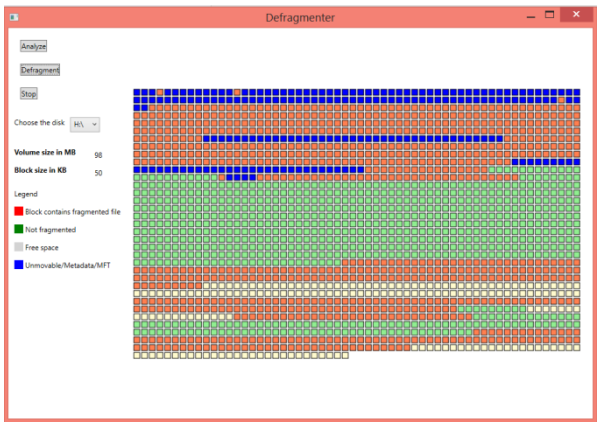


Рисунок 21 Карта раздела в процессе дефрагментации

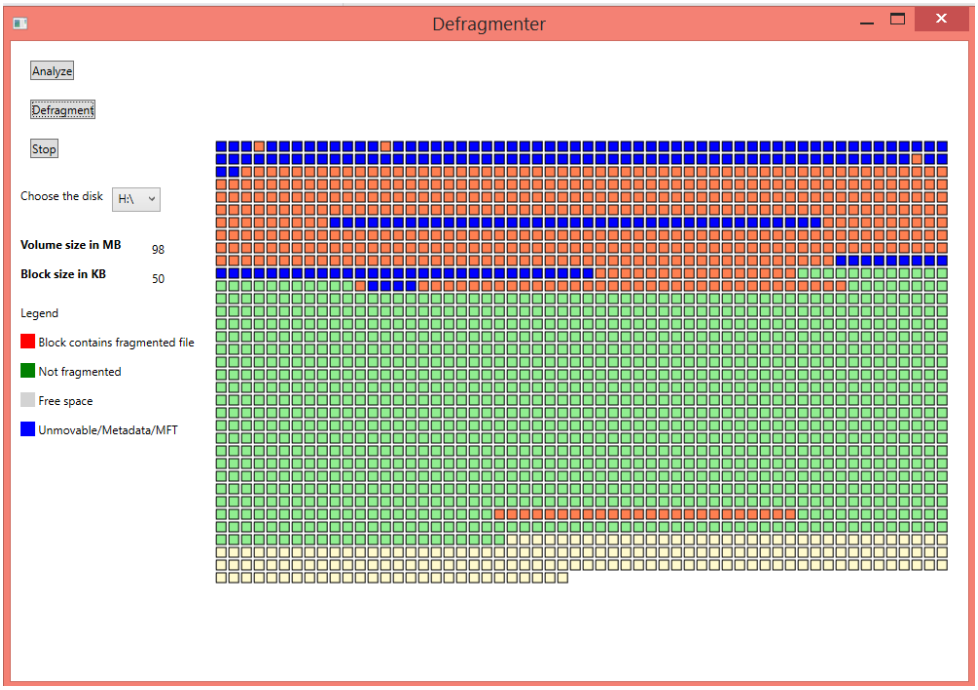


Рисунок 22 Карта раздела после дефрагментации

Второй пример иллюстрирует процесс дефрагментации на другой карте тома с достаточно большим количеством свободного места, что значительно ускоряет процесс.

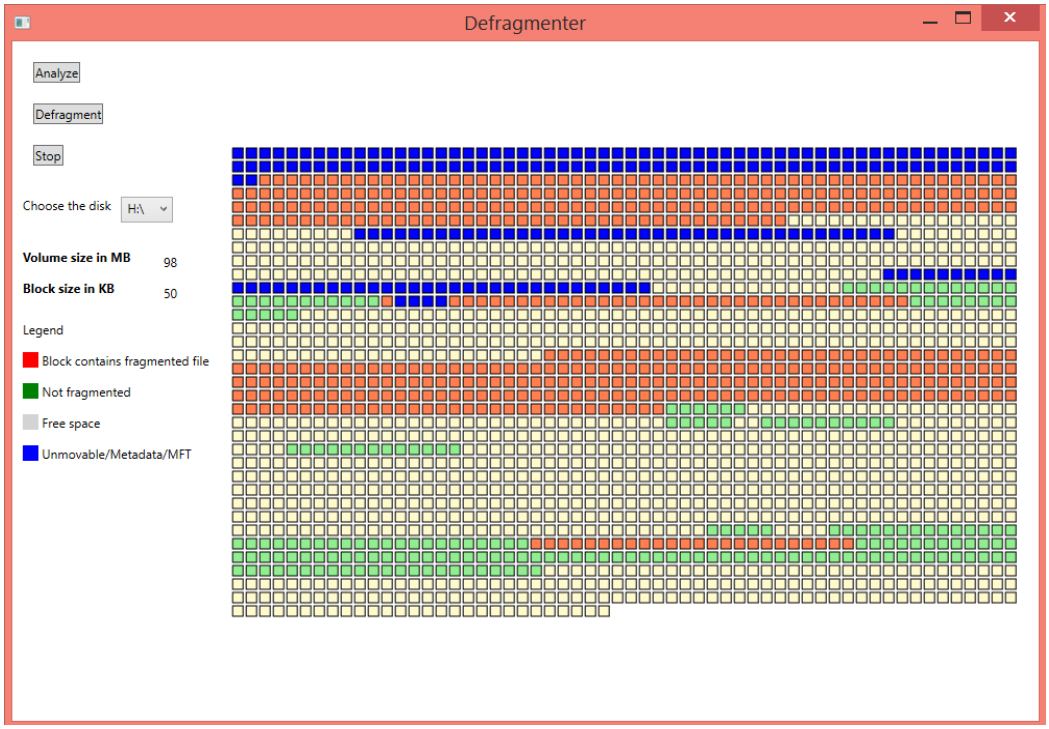


Рисунок 23 Карта раздела до дефрагментации

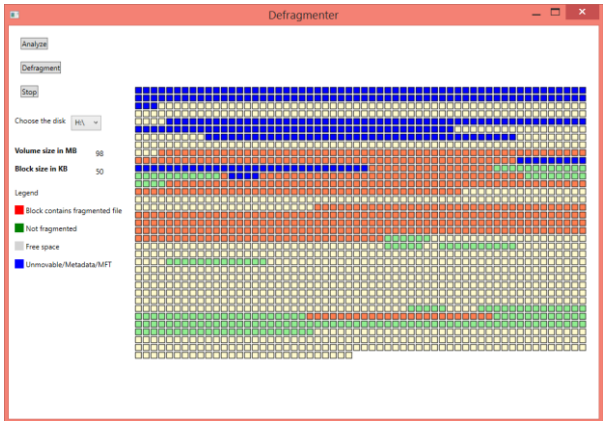


Рисунок 24 Карта раздела в процессе дефрагментации

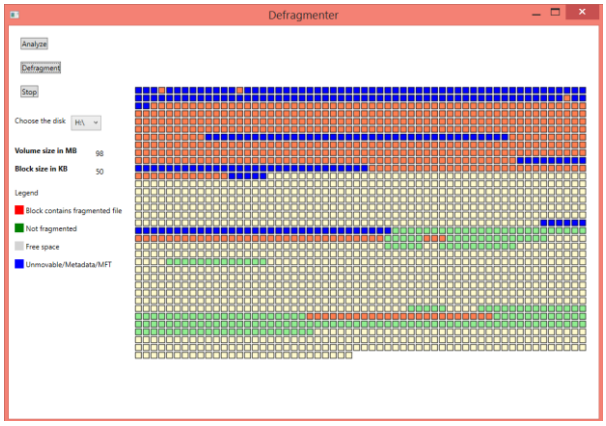


Рисунок 25 Карта раздела в процессе дефрагментации

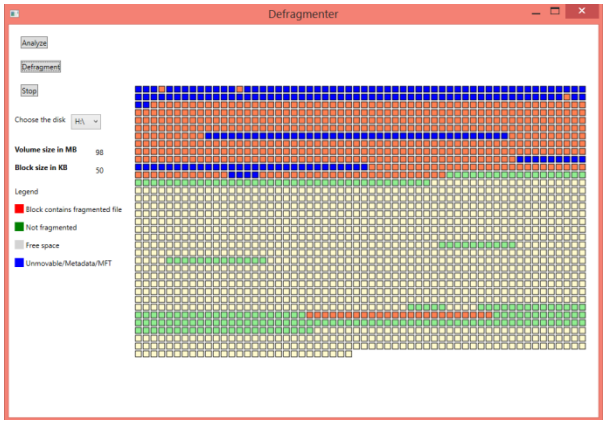


Рисунок 26 Карта раздела в процессе дефрагментации

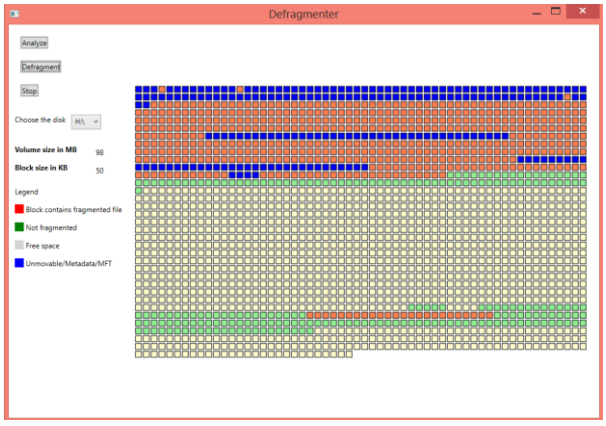


Рисунок 27 Карта раздела в процессе дефрагментации

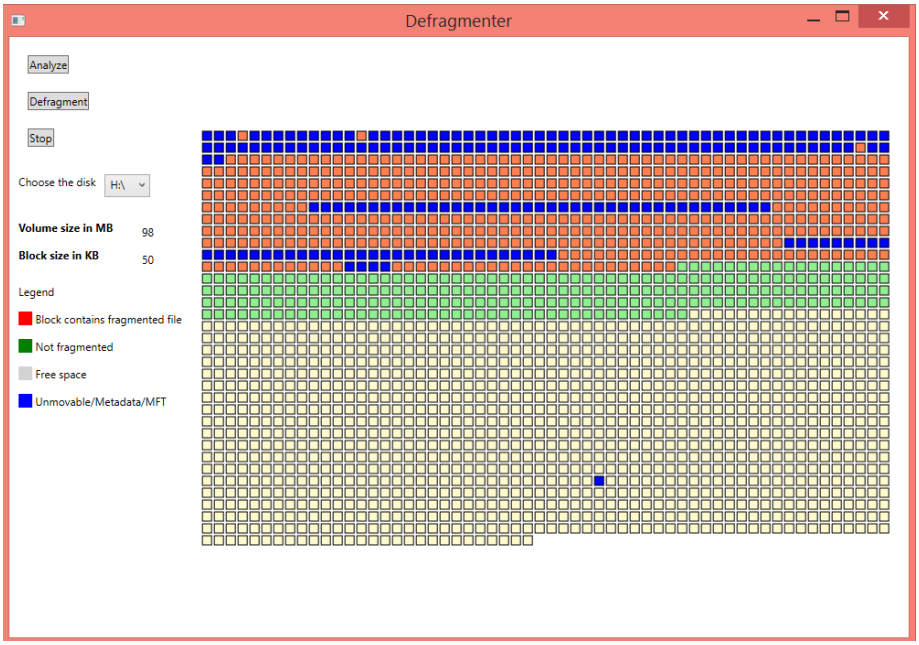


Рисунок 28 Карта раздела после дефрагментации

Заключение

В ходе работы над курсовым проектом произошло ознакомление со структурой файловой системы, документацией MSDN, платформой .NET, графическими возможностями технологии WPF, погружение в системные процессы маршалинга, сервиса взаимодействия управляемого и неуправляемого кода, работы операционной системы с файловой системой.

Результатом работы стало приложение и библиотека для дефрагментации жесткого диска, которые прошли тестирование, показавшее пригодность и работоспособность продукта.

Список использованных источников

1. Таненбаум, Э. С.Современные операционные системы = Modern Operating Systems.— 4-е изд.—СПб.:Питер, 2017.— 1120с.—ISBN 978-5-496-01395-6.
2. Леденев, А. В. Семенов, И. А. Сторожевых, В. А. Динамически загружаемые библиотеки: структура, архитектура и применение [Текст] / А.В. Леденев, И.А. Семенов, В.А. Сторожевых // Прикладная информатика. – 2008. — № 2(14). – С. 31-84.
3. https://msdn.microsoft.com