Содержание

[Введение 4](#_Toc516159649)

[Глава 1 Теоретическая часть 6](#_Toc516159650)

[1.1 Предпроектное обследование 6](#_Toc516159651)

[1.2 Характеристика инструментальных средств разработки 9](#_Toc516159652)

[Глава 2 Практическая часть 13](#_Toc516159653)

[2.1 Постановка задачи 13](#_Toc516159654)

[2.2 Анализ требований и определение спецификаций программного обеспечения 16](#_Toc516159655)

[2.3 Проектирование программного обеспечения 19](#_Toc516159656)

[2.4 Разработка пользовательских интерфейсов программного обеспечения 22](#_Toc516159657)

[2.5 Тестирование и отладка программного обеспечения 26](#_Toc516159658)

[2.6 Руководство по использованию программы 33](#_Toc516159659)

[Заключение 41](#_Toc516159660)

[Список литературы 43](#_Toc516159661)

[Приложение А 45](#_Toc516159662)

[Приложение Б 49](#_Toc516159663)

[Приложение В 56](#_Toc516159664)

# Введение

В торговле товарами и услугами для фиксации производимой операции используется контрольно-кассовая техника (ККТ). ККТ, помимо того, что формирует чек, который получает покупатель в печатном виде, ещё и записывает на фискальный накопитель (ФН) все проведённые фискальные документы.

В случае если ККТ было зарегистрировано таким образом, что данные, которые хранит ФН не передавались оператору фискальных данных (ОФД), владельцу ККТ после закрытия ФН необходимо будет сформировать отчёт о считывании фискальных данных и передать его в Федеральную налоговую службу (ФНС) вместе с отчётом на бумажном носителе.

На данный момент в компании «ПИЛОТ РБС» применяется программа, разработанная сторонней группой разработчиков и работающая напрямую с ФН через специальный переходник. Для клиентов компании такой способ формирования отчёта показался неудобным, и было принято решение разработать программный комплекс ReportFNSUtility, который будет включать в себя следующие модули:

* модуль формирования «Отчета о считывания фискальных данных» из данных длительного хранения ККТ;
* модуль дешифровки и вывода данных «Отчета о считывании фискальных данных».

Целью данной выпускной квалификационной работы является разработка модуля дешифровки и вывода данных «Отчета о считывании фискальных данных» программного комплекса ReportFNSUtility.

Для достижения поставленной цели необходимо решить следующие задачи:

* на основе анализа документации и существующего программного обеспечения разработать функциональные и нефункциональные требования к разрабатываемому модулю;
* обосновать выбор инструментальных средств разработки;
* составить необходимые диаграммы и схемы на основе заявленных требований;
* разработать и интегрировать модуль в программный комплекс;
* произвести тестирование и отладку модуля;
* разработать руководство по использованию программного модуля различными категориями пользователей, а именно руководство пользователя, руководство программиста и руководство системного администратора.

Также в процессе работы над дипломным проектом необходимо изучить специфику предметной области, а именно проанализировать такие документы как:

* Федеральный закон от 22.05.2003 N 54-ФЗ "О применении контрольно-кассовой техники при осуществлении наличных денежных расчетов и (или) расчетов с использованием электронных средств платежа" (далее – 54-ФЗ);
* Приказ ФНС России от 21.03.2017 N ММВ-7-20/229@ "Об утверждении дополнительных реквизитов фискальных документов и форматов фискальных документов, обязательных к использованию" (далее – Приказ ММВ-7-20/229).

Для разработки модуля выбраны следующие инструментальные средства:

* язык программирования C# версии 7.0 .Net Framework 4.6;
* среда разработки Visual Studio 2017.

Разрабатываемый программный модуль в качестве входных данных использует выходные данные модуля формирования «Отчета о считывания фискальных данных» из данных длительного хранения ККТ, что отражено в структурной схеме программного комплекса ReportFNSUtility.

# Глава 1 Теоретическая часть

## 1.1 Предпроектное обследование

В соответствии с федеральным законом 54-ФЗ в торговле товарами и услугами для фиксации производимых операций используется контрольно-кассовая техника (ККТ) [12]. ККТ выполняет две функции:

* формирует фискальные и нефискальные документы;
* записывает на фискальный накопитель (ФН) все проведённые фискальные документы.

К фискальным документам относятся:

* кассовый чек (приход/расход/возврат прихода/возврат расхода);
* кассовый чек коррекции (приход/расход);
* открытие/закрытие смены;
* регистрация/перерегистрация;
* закрытие ФН;
* отчет о текущем состоянии расчетов.

По требованию покупателя, а также в зависимости от функциональной возможности ККТ покупателю может быть направлен кассовый чек в электронной форме на предоставленный им абонентский номер либо адрес электронной почты.

ККТ может быть зарегистрирована в трех режимах [11]:

* «автономный режим» – режим функционирования ККТ, не предусматривающий передачу фискальных данных (ФД) в налоговые органы в электронной форме через оператора фискальных данных (ОФД);
* «режим передачи данных» – режим функционирования ККТ, предусматривающий обязательную передачу ФД в налоговые органы в электронной форме через ОФД;
* «автоматический режим» – режим применения ККТ в составе автоматического устройства для расчетов при осуществлении расчетов с покупателем в автоматическом режиме с применением ККТ без участия уполномоченного лица.

Фискальный накопитель – это основная память в кассовом аппарате, он берет на себя две основные функции:

* хранение всех кассовых чеков в собственной памяти;
* шифрование сохраненных данных (кассовых чеков) и формирование специального кода для их проверки.

Организациям и предпринимателям разрешено не передавать сведения о расчетах по кассе через операторов фискальных данных в ФНС России, если они осуществляют деятельность в отдаленных от сетей связи местностях. Перечень таких местностей определяют местные власти, исходя из критерия: численность населения до 10 тысяч человек. Этот критерий введен Приказом Минкомсвязи России от 05.12.2016 N 616 "Об утверждении критерия определения отдаленных от сетей связи местностей". Таким клиентам необходимо формировать отчет о считывании фискальных данных и сдавать его в ФНС России вместе с отчётом на бумажном носителе.

Пользователи при перерегистрации ККТ в связи с заменой фискального накопителя и снятии с регистрационного учета контрольно-кассовой техники, которая применялась в автономном режиме, обязаны осуществить считывание фискальных данных всех фискальных документов, содержащихся в фискальном накопителе, и представить эти фискальные данные в налоговые органы вместе с заявлением о перерегистрации контрольно-кассовой техники или о снятии контрольно-кассовой техники с регистрационного учета на бумажном носителе или через кабинет контрольно-кассовой техники. Датой подачи отчета в электронной форме считается дата его размещения в кабинете ККТ.

Фискальные данные, хранящиеся в памяти ФН не менее 5 лет (далее – фискальные данные длительного хранения), должны считываться из памяти ФН и включаться в состав «Отчета о считывании фискальных данных», с использованием программы для выгрузки данных из ФН.

Отчет о считывании ФД должен иметь заголовок, содержащий поля записи, описанные в таблице 1.1.

Таблица 1.1 – Структура Отчета о считывании ФД

|  |  |  |  |
| --- | --- | --- | --- |
| Поле записи | Тип | Формат данных | Длина |
| Наименование файла выгрузки | Текст | String | 53 |
| Программа выгрузки | Текст | String | 256 |
| Регистрационный номер ККТ | Текст | String | 20 |
| Номер ФН | Текст | String | 16 |
| Номер версии ФФД | Целое | byte | 1 |
| Количество смен | Целое | UInt32 | 4 |
| Количество фискальных документов | Целое | UInt32 | 4 |
| Контрольная сумма файла выгрузки | Целое | UInt32 | 4 |

Поля записи заголовка должны иметь фиксированную длину в байтах, указанную в таблице 1.1. Значения полей записи, содержание которых определяется значением соответствующего реквизита, должны указываться в заголовке без поля «тег» и поля «длина».

Данные о проведённых документах в свою очередь должны быть представлены в форматах:

* TLV – метод записи данных в электронной форме в виде структуры, состоящей из трех полей: тип-длина-значение (tag-length-value), когда значение представлено данными установленного формата;
* STLV – метод записи данных в электронной форме в виде структуры, состоящей из трех полей: тип-длина-значение (tag-length-value), когда значение, в свою очередь, представляется в виде одной или нескольких TLV-структур.

Передо мной была поставлена цель разработки модуля дешифровки и вывода данных отчета о считывании ФД. В связи с тем, что у одного документа может быть несколько дочерних элементов (описание, подтверждение передачи) было принято решение отображать документы на экране в виде дерева, раскрывающегося при нажатии на кнопки «+». Из-за больших объёмов данных необходимо предусмотреть возможность остановки процесса считывания. Для реализации этого будет использован механизм многопоточности процесса.

Для того, чтобы пользователю не пришлось просматривать все записанные документы, для проверки их корректности, было решено рассчитывать статистику по количеству и суммам записанных в файле документов.

## 1.2 Характеристика инструментальных средств разработки

Разрабатываемый модуль является частью программного комплекса ReportFNSUtility. В компании для разработки данного программного комплекса было решено использовать среду программирования Visual Studio 2017 и язык программирования С# версии 7.0 .NET Framework 4.6. Часть необходимых методов для взаимодействия с TLV-структурами была взята из разрабатываемой в компании «Пилот РБС» библиотеки «Fw16.dll».

Для удобства взаимодействия внутри команды разработчиков программного комплекса было решено использовать систему контроля версий Git с использованием веб-сервиса GitHub.

Для построения схем и диаграмм при составлении технического задания и сопровождающей документации было решено использовать Dia Gnome. Dia – свободный кроссплатформенный редактор диаграмм, являющийся частью GNOME Office. Данный редактор схем и диаграмм был выбран исходя из того, что он является бесплатным и имеет удобный и простой интерфейс. В этом редакторе имеется множество различных фигур, необходимых для проектирования не только программных продуктов, но и радиоэлектронных элементов, сетевых диаграмм и прочих схем. Помимо стандартных элементов в редактор могут быть добавлены новые с помощью файлов формата, основанного на расширяемом языке разметки XML.

Visual Studio 2017 – среда разработки, используемая для программного комплекса ReportFNSUtility, является последней версией из линейки продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Продукты из этой линейки позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом.

Visual Studio 2017 включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий (СКВ) исходного кода.

СКВ – это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем у разработчика была возможность вернуться к определённым старым версиям этих файлов.

СКВ предоставляет следующие возможности:

* поддержка хранения файлов в репозитории;
* поддержка истории версий файлов в репозитории;
* нахождение конфликтов при изменении исходного кода и обеспечение синхронизации при работе в многопользовательской среде разработки;
* отслеживание авторов изменений.

Git – распределенная система контроля версий, разработанная Линусом Торвальдсом. Изначально Git предназначалась для использования в процессе разработки ядра Linux, но позже стала использоваться и во многих других проектах (X.org и Ruby on Rails, Drupal).

В проекте была использована СКВ Git с использованием бесплатного сервиса хостинга Git-репозиториев Github.com. На данный момент Git является самой быстрой распределенной системой, использующей самое компактное хранилище ревизий. Помимо этого для Visual Studio 2017 существует расширение, которое позволяет делать так называемые коммиты (commits), создавать ветки и соединять их непосредственно в самой студии с помощью интуитивно понятного интерфейса.

Язык программирования C#, используемый в разработке программного комплекса ReportFNSUtility, является объектно-ориентированным. C# относится к семье языков с C-подобным синтаксисом, его синтаксис наиболее близок к C++ и Java [13]. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Ниже перечислены основные достоинства языка С#[17]:

* является объектно-ориентированным языком, где типы, встроенные в язык, представлены классами;
* предоставляет возможности наследования и универсализации;
* является наследником языков C/C++, сохраняя лучшие черты этих популярных языков программирования;
* является простым и надёжным, поскольку на C# хотя и допускаются, но не поощряются такие опасные свойства С++, как указатели, адресация, разыменование, адресная арифметика;
* предоставляет возможность создавать многопоточные программы;
* создавался параллельно с каркасом Framework .Net и в полной мере учитывает все его возможности – поддержку как Framework Class Library (FCL), так и Common Language Runtime (CLR);
* благодаря каркасу Framework .Net, ставшему надстройкой над операционной системой, программисты C# получают те же преимущества работы с виртуальной машиной, что и программисты Java. Эффективность кода даже повышается, поскольку исполнительная среда CLR представляет собой компилятор промежуточного языка, в то время как виртуальная Java-машина является интерпретатором байт-кода.

В разрабатываемом программном модуле использованы такие структуры и конструкции языка программирования C#, как:

* классы и вложенные классы;
* перечисления;
* потоки;
* потоки данных;
* перегрузка индексатора.

Одним из ключевых аспектов в современном программировании является многопоточность. Ключевым понятием при работе с многопоточностью является поток. Поток представляет некоторую часть кода программы. При выполнении программы каждому потоку выделяется определенный квант времени. И при помощи многопоточности мы можем выделить в приложении несколько потоков, которые будут выполнять различные задачи одновременно. Благодаря потокам мы можем выделить отправку запроса или любую другую задачу, которая может долго обрабатываться, в отдельный поток. Поэтому клиент-серверные приложения практически не мыслимы без многопоточности.

В разрабатываемом модуле многопоточность будет играть важную роль. Благодаря ей пользователь сможет взаимодействовать с экранными формами, пока происходит расчёт статистики, а именно он сможет выводить документы и просматривать их содержимое в виде древовидной структуры. Помимо этого, многопоточность позволяет при необходимости прерывать процесс подсчёта статистики и выведение документов на экран.

# Глава 2 Практическая часть

## 2.1 Постановка задачи

### 2.1.1 Описание входных данных

Входными данными являются:

* путь к файлу отчёта о считывании фискальных данных;
* номер начального документа, с которого необходимо вывести подробную информацию;
* номер конечного документа, с которого необходимо вывести подробную информацию.

Структура входных данных представлена в таблице 2.1.

Таблица 2.1 – Входные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Идентификатор | Тип данных | Размер |
| Путь к файлу отчёта | Patch | String | 255 Байт |
| Номер начального документа | \_start | UInt32 | 4 Байта |
| Номер конечного документа | \_end | UInt32 | 4 Байта |

### 2.1.2 Описание выходных данных

Выходными данными являются:

* данные заголовка отчёта о считывании фискальных данных;
* статистика о количестве и сумме документов;
* подробная информация о записанных документах в отчёте.

Структура выходных данных представлена в таблице 2.2.

### 2.1.3 Функциональные требования

Разрабатываемый программный модуль должен проверять контрольную сумму, вырезая её из потока, и формировать новую контрольную сумму, используя метод хеширования CRC-32. В случае несовпадения контрольных сумм пользователь должен получить сообщение о повреждении файла.

Таблица 2.2 – Выходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Идентификатор | Тип данных | Размер |
| Наименование файла выгрузки | Name | String | 53 Байта |
| Программа выгрузки | NameProgram | String | 255 Байт |
| Регистрационный номер ККТ | NumberECR | String | 20 Байт |
| Номер ФН | NumberFS | String | 16 Байт |
| Номер версии ФФД | NumberFFD | byte | 1 Байт |
| Количество смен | CountShift | UInt32 | 4 Байта |
| Количество фискальных документов | CountFiscalDoc | UInt32 | 4 Байта |
| Контрольная сумма файла выгрузки | Hash | UInt32 | 4 Байта |
| Статистика | stats[12] | decimal[] | 128 Байт |
| Документы | Item | TreeNode | - |

Разрабатываемый программный модуль должен считывать заголовок файла, состоящего из 8 полей, описание и типы данных которых представлены в Приложении Б, и выводить данные на экран. В случае возникновения ошибок преобразования данных из бинарного формата в строковый или целочисленный тип считывание информации из потока должно быть прервано, и на экране должно появиться сообщение о том, что заголовок был сформирован некорректно.

Программный модуль должен считывать документы, записанные в виде STLV- и TLV-структур, и составлять список всех позиций документов в потоке для быстрого считывания их по мере необходимости. В случае выхода за пределы потока из-за ошибочной длины или непредвиденного значения тега должно быть выведено сообщение о том, что дерево тегов было сформировано некорректно и считывание данных из потока должно прекратиться.

После успешного выполнения трёх предыдущих пунктов программный модуль должен запускать в отдельном потоке подсчёт статистики по количеству и сумме записанных документов и выводить информацию на экран.

Также после успешного выполнения трёх первых пунктов программный модуль должен предоставлять пользователю возможность запустить отображение определённого диапазона документов в виде древовидной структуры с формированием «ветвей», содержащих данные документов, записанных в файле.

### 2.1.4 Нефункциональные требования

Ниже перечислены требования к интерфейсу разрабатываемого программного модуля:

* приложение должно обладать интуитивно понятным графическим интерфейсом;
* приложение должно быть оформлено в строгом минималистичном стиле;
* данные статистики должны отображаться на отдельной вкладке;
* данные документов должны отображаться в виде древовидной структуры;
* данные заголовка должны быть постоянно видны пользователю;
* кнопки запуска процесса должны использоваться и для остановки процесса;
* текущее состояние процесса выполнения подсчёта статистики должно отображаться на полосе загрузки.

Программный модуль должен быть реализован на языке программирования C# версии .Net Framework 4.6 в среде программирования Visual Studio 2017.

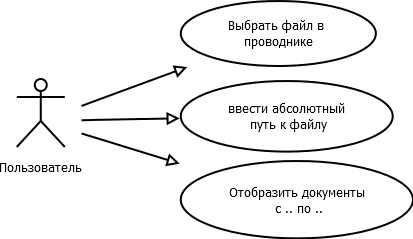
Ниже перечислены требования к надёжности программного модуля:

* достоверность фискальных данных обеспечивается за счёт невозможности редактирования документов в программе;
* целостность данных обеспечивается за счёт проверки контрольной суммы путём вырезания её из потока файла и формирования проверочной контрольной суммы.

## 2.2 Анализ требований и определение спецификаций программного обеспечения

### 2.2.1 Диаграмма вариантов использования

На рис. 1 (а также в Приложении А на рис. А.1) изображена диаграмма вариантов использования. На ней представлено взаимодействие пользователя с разрабатываемым модулем. Пользователь может ввести абсолютный путь к файлу отчёта или выбрать файл в диалоговом окне. Путь к файлу, полученный одним из двух способов, необходим для считывания информации. Третий вариант использования – это отображение необходимых документов на экране.

  
Рис. 1 – Диаграмма вариантов использования

### 2.2.2 Диаграмма потоков данных

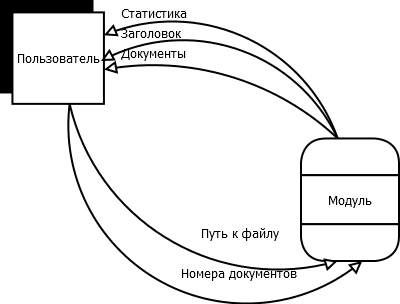
На рис. 2 (а также в Приложении А на рис. А.2) изображена контекстная диаграмма потоков данных. На ней представлено, какие данные пользователь передаёт в программу, а какие – получает на выходе.

Входные данные:

* путь к файлу;
* номера документов.

Выходные данные:

* статистика;
* заголовок;
* содержимое документов.

  
Рис. 2 – Диаграмма потоков данных (контекстная)

### 2.2.3 Функциональная диаграмма

На рис. 3 (а также в Приложении А на рис. А.3) изображена контекстная функциональная диаграмма. На ней отражены:

* исходные данные:

1. путь к файлу;
2. номера документов;

* результаты:

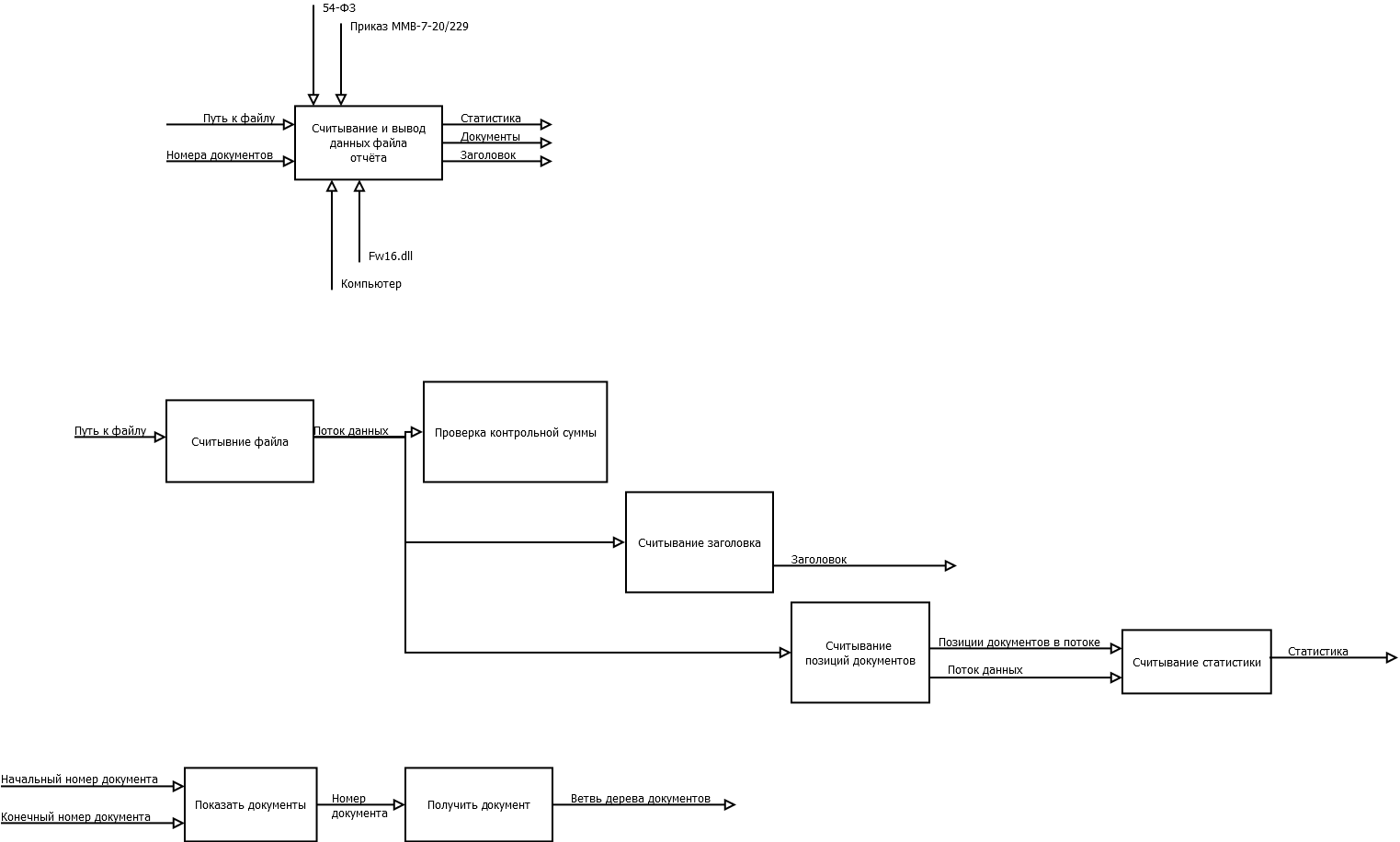
1. статистика – данные о количестве и сумме проводимых документов;
2. документы – данные, представленные в виде древовидной структуры;
3. заголовок – восемь полей, описанных в приложении Б;

* управление:

1. 54-ФЗ;
2. приказ ММВ-7-20/229;

* механизмы:

1. компьютер;
2. Fw16.dll – сторонняя библиотека разработанная «Пилот РБС».

  
Рис. 3 – Функциональная диаграмма (контекстная)

В приложении А на рис. А.4 изображена детализированная функциональная диаграмма. Пользователь вводит путь к файлу и запускает считывание файла.

После этого путь передаётся в функцию считывания файла, которая в свою очередь передаёт поток данных в:

* функцию проверки контрольной суммы (в случае непрохождения проверки процесс считывания файла будет прерван и выведено сообщение об ошибке);
* функцию считывания заголовка (в случае обнаружения ошибок в данных заголовка процесс считывания файла будет прерван и выведено сообщение об ошибке);
* функцию считывания позиций документов (в случае обнаружения ошибок в TLV структурах, записанных в файле, процесс считывания будет прерван и выведено сообщение об ошибке).

Функция считывания позиций документа передаёт поток данных и позиции в функцию считывания статистики. Функция считывания статистики выводит статистику на экран.

В функцию отображения документов передаются начальный и конечный номер диапазона выводимых документов. Выходные данные этой функции являются входными для функции получения документов. Функция получения документов выводит «ветвь дерева» документов на экран.

### 2.2.4 Диаграмма классов

На рис. 4 (а также в Приложении А на рис. А.5) изображена диаграмма классов. В главном классе программного комплекса «ReportFNSUtility» (Program) находится основной класс модуля чтения файла отчёта о считывании фискальных данных – ReportReader. Отчёт о считывании фискальных данных представлен в программе в виде класса ReportFNS и вложенных в него классов Header и TreeOfTags.

Header отвечает за:

* считывание данных заголовка;
* хранение данных заголовка;
* формирование контрольной суммы;
* проверку контрольной суммы.

TreeOfTags отвечает за:

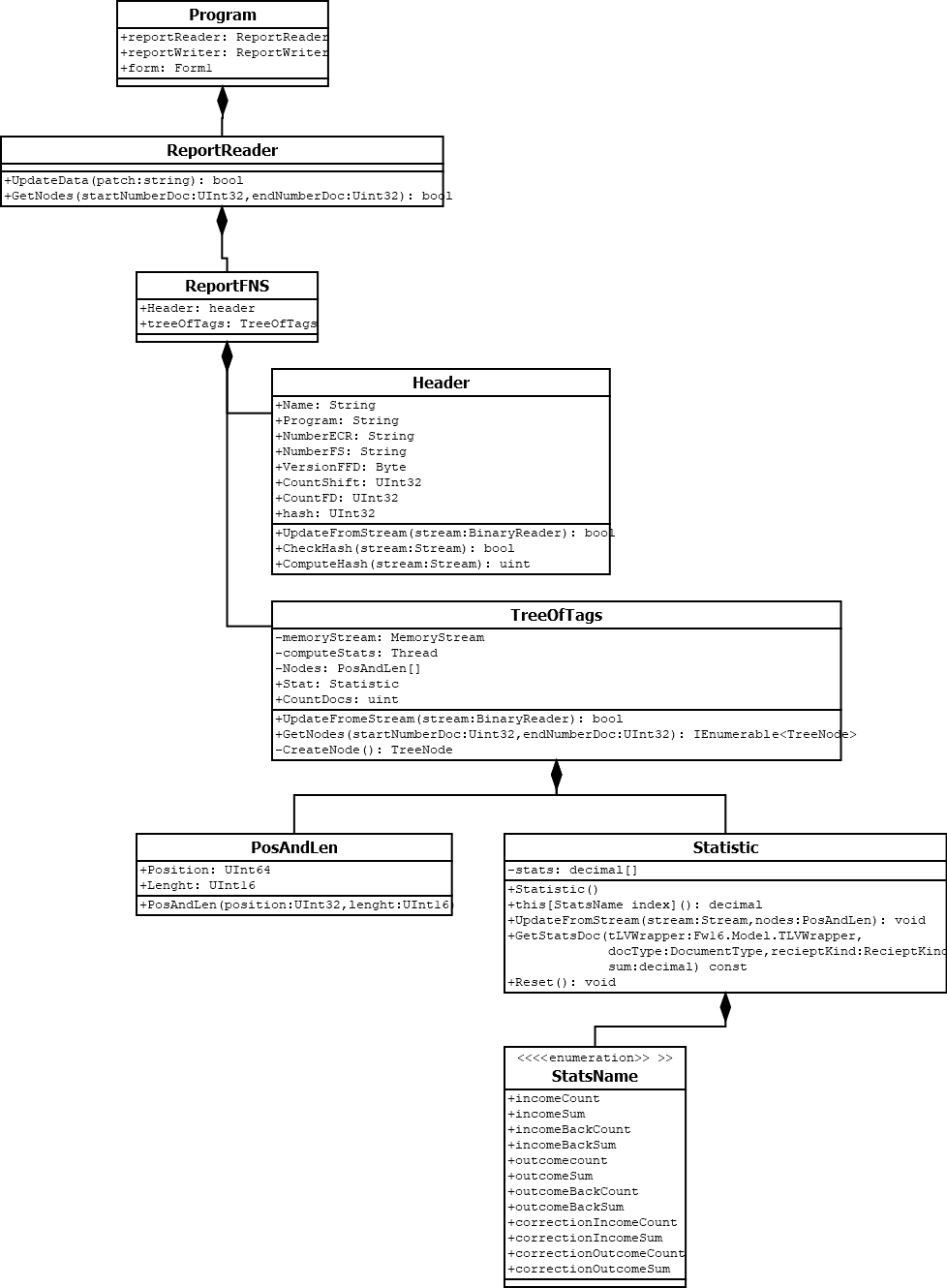
* считывание документов, хранящихся в основной части отчёта;
* формирование из данных документов веток для элемента формы древовидного представления информации;
* запуск подсчёта статистики.

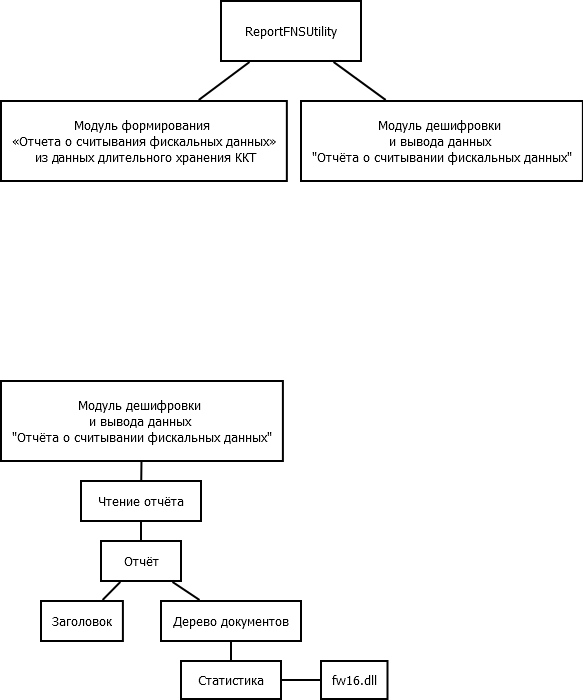
Statistic – класс, вложенный в класс TreeOfTags, отвечает за подсчёт статистики. В классе Statistic имеется перечисление с наименованием данных статистики.

## 2.3 Проектирование программного обеспечения

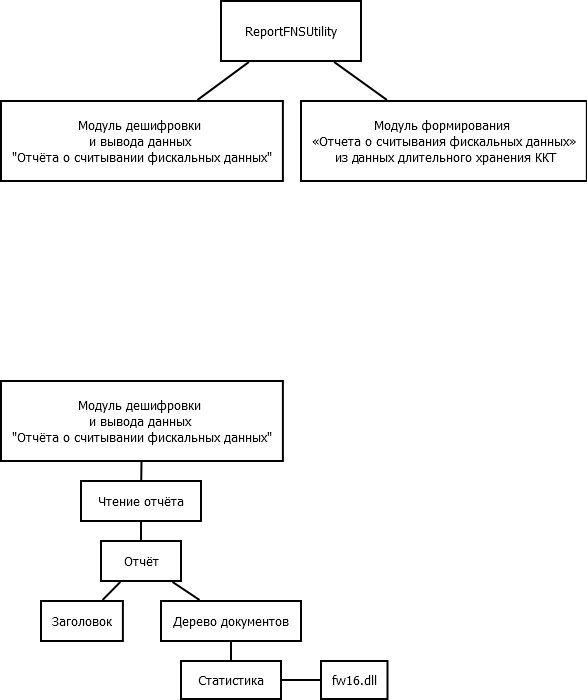
На рис. 5 (а также в Приложении А на рис. А.6) представлена структурная схема программного комплекса «ReportFNSUtility», частью которой является разрабатываемый мною модуль. На схеме видно, что программный комплекс состоит из двух модулей:

* модуль формирования «Отчета о считывания фискальных данных» из данных длительного хранения ККТ;
* модуль дешифровки и вывода данных «Отчета о считывании фискальных данных».

  
Рис. 4 – Диаграмма классов

Рис. 5 – Структурная схема комплекса «ReportFNSUtility»

На рис. 6 (а также в Приложении А на рис. А.7) представлена структурная схема разрабатываемого мною программного модуля дешифровки и вывода данных «Отчета о считывании фискальных данных». На ней показано, что модуль включает в себя элемент структуры «чтение отчёта», который в свою очередь включает в себя «Заголовок» и «Дерево документов». «Дерево документов» включает в себя «Статистику», которая использует методы библиотеки «Fw16.dll».

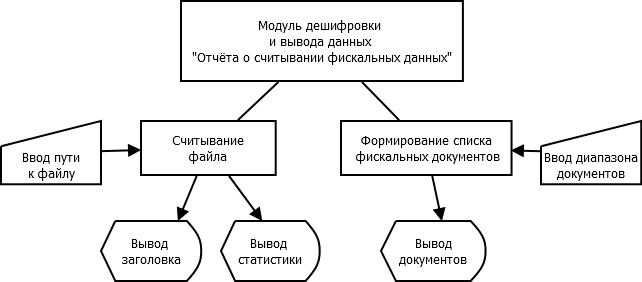
  
Рис. 6 – Структурная схема модуля

На рис. 7 (а также в Приложении А на рис. А.8) изображена функциональная схема. Модуль дешифровки и вывода данных «Отчета о считывании фискальных данных» имеет две функции:

* считывание файла;
* формирование списка фискальных документов.

В функцию считывания файла вводится путь к файлу. На экран выводится статистика и данные заголовка.

В функцию формирования списка фискальных документов вводится диапазон выводимых документов. На экран выводятся данные документов.

  
Рис. 7 – Функциональная схема

## 2.4 Разработка пользовательских интерфейсов программного обеспечения

На рис. 8 изображено главное окно программного комплекса «ReportFNSUtility». В левой части окна находятся сгруппированные поля заголовка считанного файла «Отчёта о считывании фискальных данных», доступные только для чтения. Ниже сгруппированы элементы для взаимодействия с модулем формирования отчёта.

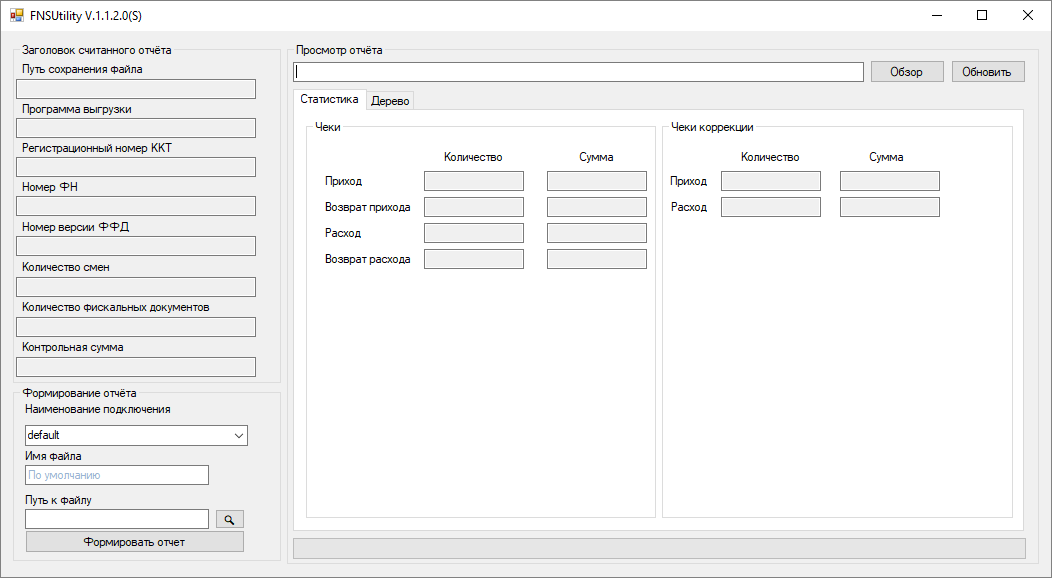
В основной части находятся:

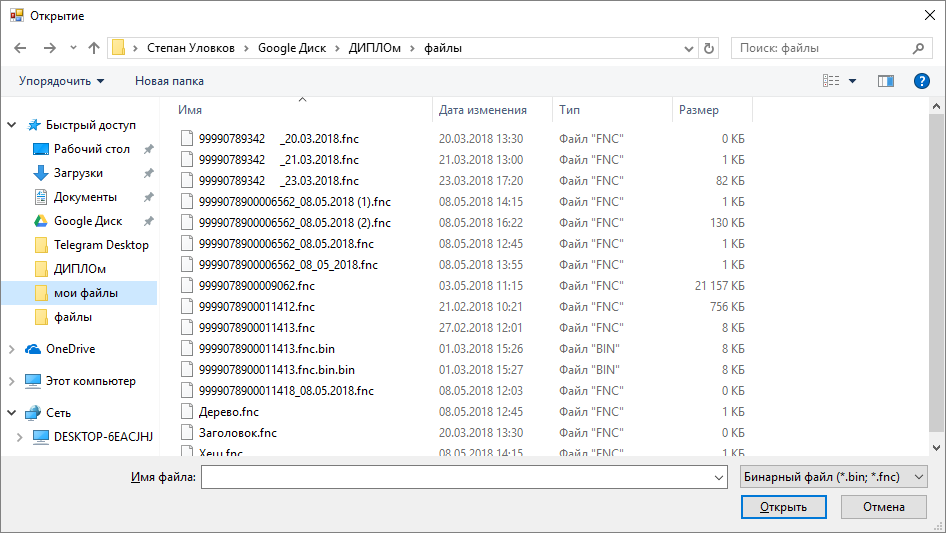
* поле для ввода пути к файлу отчёта;
* кнопка «Обзор», вызывающая диалоговое окно выбора файла   
  (рис. 9);
* кнопка «Обновить», запускающая процесс считывания данных из файла, путь к которому указан в поле;
* вкладка «Статистика», отражающая статистику количества и суммы записанных в файле чеков и чеков коррекции;
* вкладка «Дерево»;
* полоса загрузки, которая заполняется по мере подсчёта статистики.

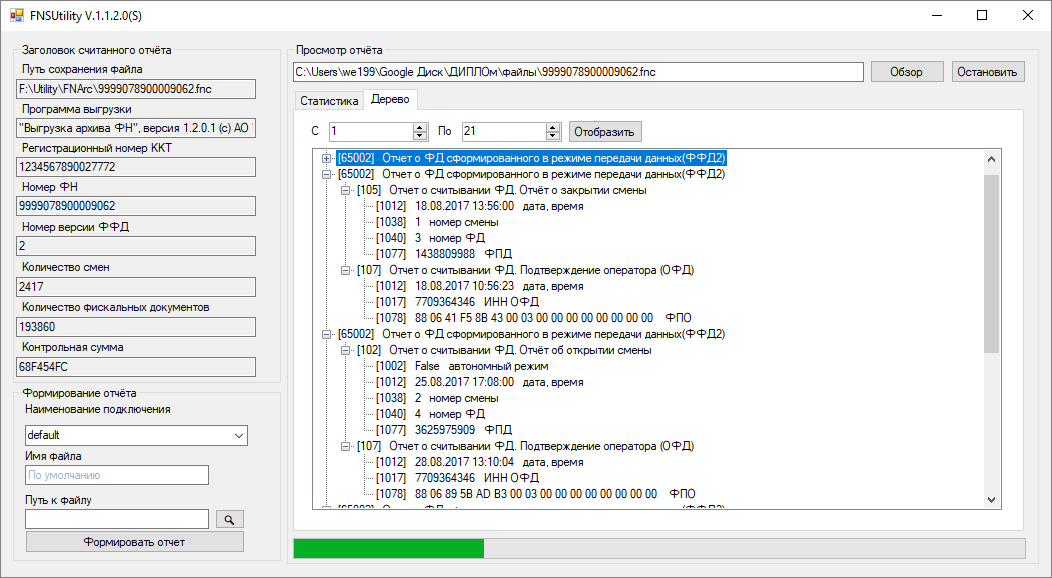
На рис. 10 представлен пример отображения документов в виде древовидной структуры. Данные записаны следующим образом:

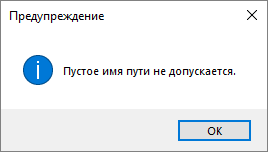
1. тег в квадратных скобках;
2. значение, если это TLV-структура;
3. описание тега.

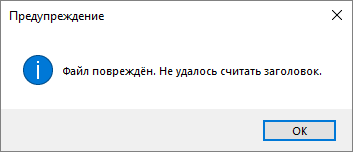
На рис. 11–15 представлены сообщения, оповещающие пользователя об обнаруженных ошибках в ходе считывания файла и вывода документов в виде древовидной структуры.

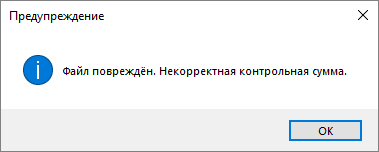
Рис. 8 – Главное окно программного комплекса «ReportFNSUtility»

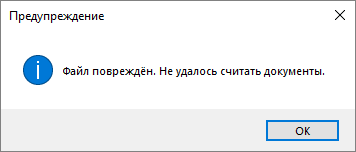
Рис. 9 – Диалоговое окно выбора файла

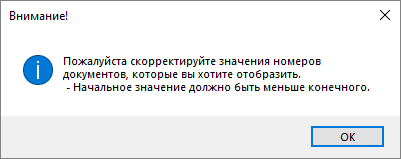
Рис. 10 – Дерево документов

  
Рис. 11 – предупреждение о пустом имени файла

  
Рис. 12 – предупреждение о некорректном заголовке

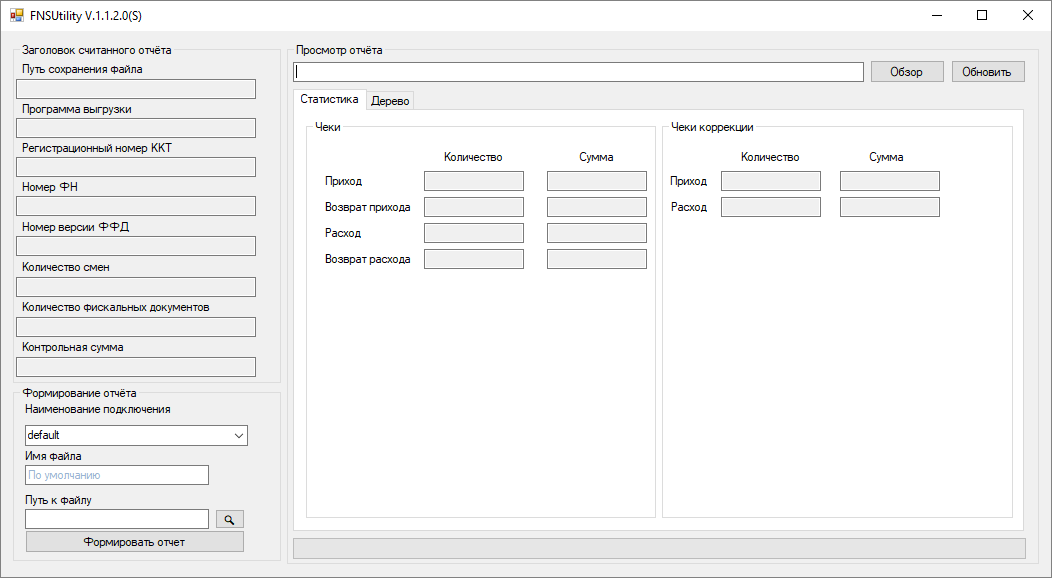
  
Рис. 13 – предупреждение о некорректной контрольной сумме

  
Рис. 14 – предупреждение о повреждении записи документов

  
Рис. 15 – Предупреждение о некорректном заполнении номеров выводимых документов

## 2.5 Тестирование и отладка программного обеспечения

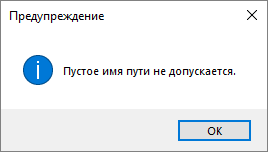
Проведём тестирование разрабатываемого модуля. После запуска программного комплекса «ReportFNSUtility» мы попадаем в главное окно, интерфейс которого можно увидеть на рис. 16.

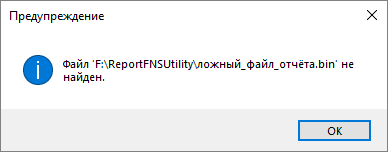
Рис. 16 – Главное окно программного комплекса «ReportFNSUtility»

Произведём тестирование с некорректным путём к файлу. Используем пустое значение в качестве пути к файлу и наименование несуществующего файла («ложный\_файл\_отчёта.bin»). В первом случае было выведено сообщение о том, что пустой путь не допускается, результат отображён на рис. 17. Во втором случае так как мы указали только название файла без полного пути, модуль будет искать файл в каталоге, где находится исполняемый файл программного комплекса. В связи с тем, что такого фала в каталоге нет, было выведено сообщение об ошибке, которое можно увидеть на рис. 18.

Следующим шагом проведём тестирование на трёх повреждённых файлах:

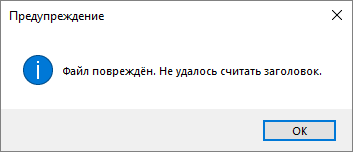
* «F:\ReportFNSUtility\files\Заголовок.fnc»;
* «F:\ReportFNSUtility\files\Хеш.fnc»;
* «F:\ReportFNSUtility\files\Дерево.fnc».

  
Рис. 17 – Предупреждение о пустом имени файла

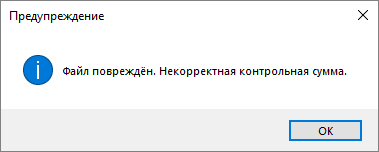
  
Рис. 18 – Предупреждение о несуществующем файле

В первом файле повреждён заголовок, а именно записана только часть данных. Во втором файле были внесены изменения сторонней программой без пересчёта контрольной суммы. В третьем файле была записана некорректная длина в TLV-структуре.

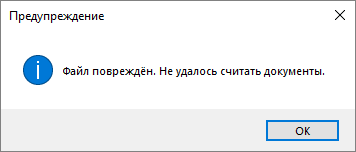
При обновлении данных из первого файла было выведено сообщение о том, что файл был повреждён и программе не удалось прочитать заголовок. Результат можно видеть на рис. 19.

  
Рис. 19 – Предупреждение о некорректном заголовке

При обновлении данных из второго файла было выведено сообщение о том, что файл был повреждён контрольная сумма не совпала с вновь сформированной. Результат можно видеть на рис. 20.

  
Рис. 20 – Предупреждение о некорректной контрольной сумме

При обновлении данных из третьего файла было выведено сообщение о том, что файл был повреждён и программе не удалось считать документы. Результат можно видеть на рис. 21.

  
Рис. 21 – Предупреждение о повреждении записи документов

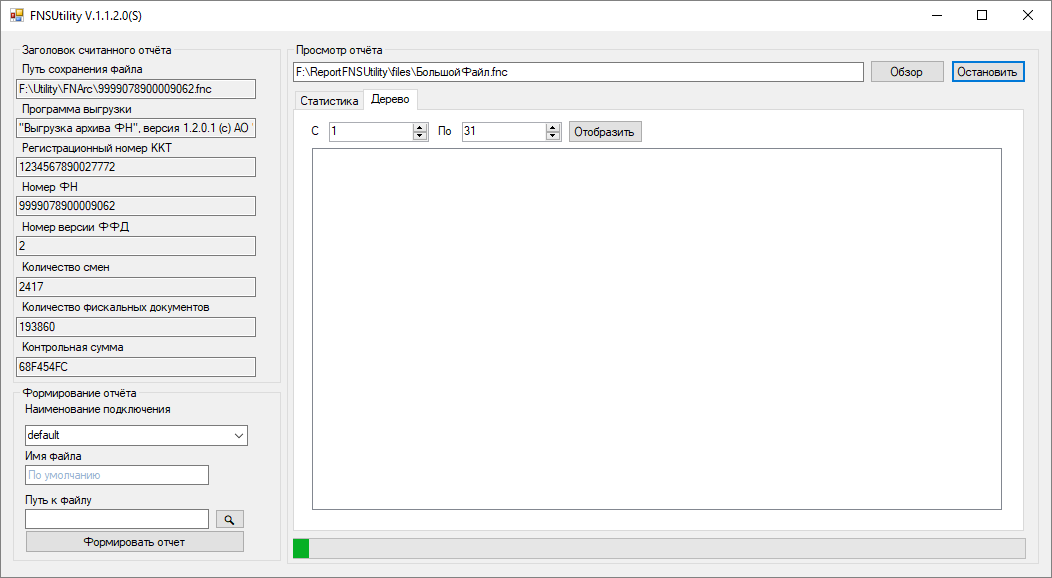
Теперь произведём тестирование на корректном файле. В поле пути введём «F:\ReportFNSUtility\files\БольшойФайл.fnc» и запустим обновление данных. После нажатия на кнопку произошли следующие изменения на форме:

* кнопка «Обновить» изменила надпись на «Остановить»;
* поля данных заголовка заполнились данными из файла;
* на вкладке «Дерево» кнопка «Отобразить» стала доступна;
* в нижней части окна начала заполняться полоса загрузки.

Описанные изменения можно увидеть на рис. 22.

До окончания считывания прервём процесс, нажав на кнопку «Остановить». В результате:

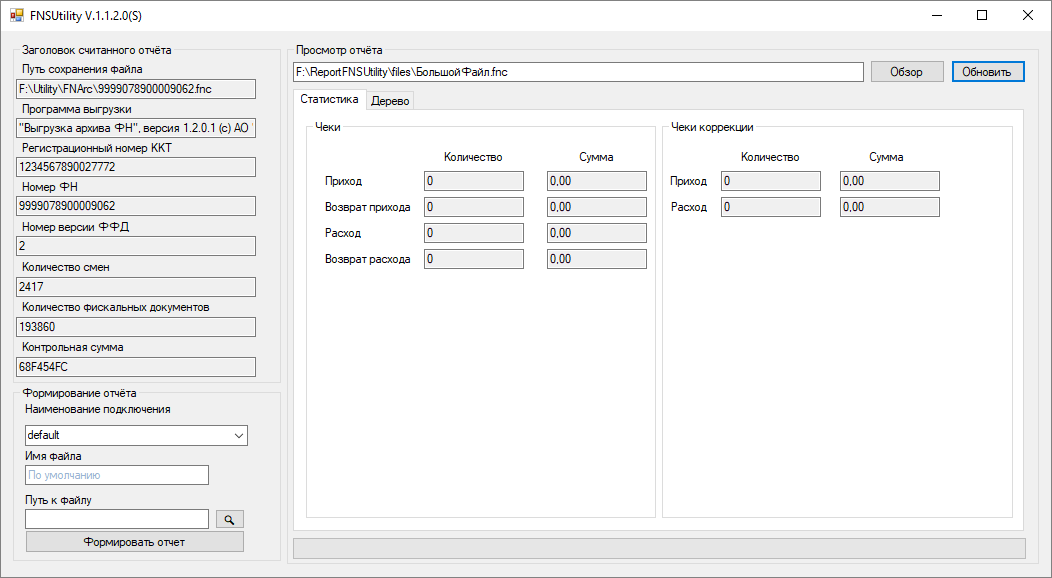
* полоса загрузки обнулилась;
* данные заголовка не обнулились;
* кнопка «Отобразить» осталась доступной;

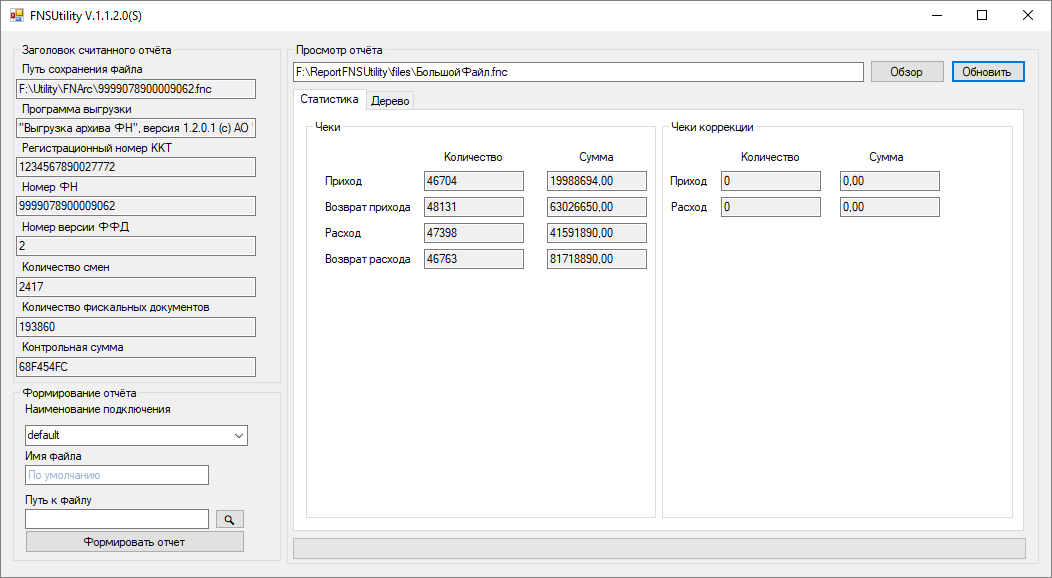
Рис. 22 – Вкладка «Дерево» после считывания файла небольшого объёма

* кнопка «Остановить» изменила надпись на «Обновить»;
* данные статистики не были заполнены.

Описанные изменения можно увидеть на рис. 23.

Считаем файл до конца. В результате поля статистики будут заполнены данными, полученными из файла. Результат можно увидеть на рис. 24.

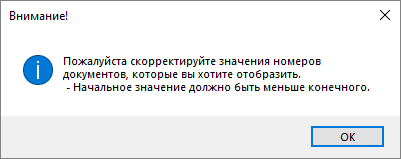
Рис. 23 – Вкладка «Статистика» после прерывания считывания файла

Рис. 24 – Вкладка «Статистика» после завершения считывания файла

Произведём вывод документов. Введём номер начального документа больше, чем номер конечного:

* начальный: 11;
* конечный: 1.

В результате было выведено сообщение о некорректном заполнении, которое можно увидеть на рис. 25.

  
Рис. 25 – Предупреждение о некорректном заполнении номеров выводимых документов

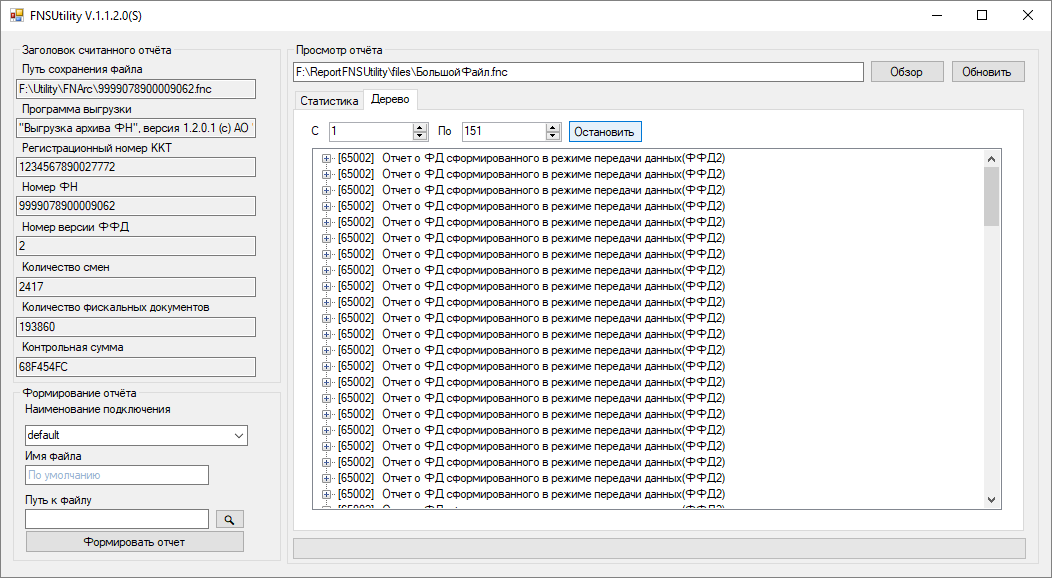
Введём корректные данные в поля номеров документов, а именно:

* начальный: 1;
* конечный: 151.

В результате на экранной форме произошли следующие изменения:

* кнопка «Отобразить» изменила надпись на «Остановить»;
* постепенно элемент древовидного представления данных начал заполняться документами.

Описанные изменения можно увидеть на рис. 26.

Рис. 26 – Формирование документов в виде древовидной структуры

Если прервать данный процесс, нажав на кнопку «Остановить», процесс формирования документов прекратится, но элемент древовидного представления информации не будет очищен, что позволит просматривать документы.

Считанные данные изменять на форме не представляется возможным. Вводить пустое или буквенное значение в поля номеров документов не представляется возможным.

Отчёт по всем вводимым данным и соответствующим реакциям программного модуля можно увидеть в таблице 2.3.

Подведём итоги. Модуль выполняет все поставленные задачи. На некорректные данные модуль реагирует соответствующими сообщениями либо не даёт их вводить. Ошибок в ходе тестирования обнаружено не было. Тестирование можно завершить, сделав вывод о том, что оно прошло успешно.

Таблица 2.3 – Отчёт по вводимым данным и соответствующим реакциям

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Входные данные | Вводимое значение | Реакция программы | |
| 1 | Путь | «» | Сообщение о запрете ввода пустого пути | |
| 2 | Путь | «ложный\_файл\_отчёта.bin» | Сообщение о несуществующем файле | |
| 3 | Путь | «F:\ReportFNSUtility\files\Заголовок.fnc» | Сообщение о повреждённом заголовке |
| 4 | Путь | «F:\ReportFNSUtility\files\Хеш.fnc» | Сообщение о некорректной контрольной сумме |
| 5 | Путь | «F:\ReportFNSUtility\files\Дерево.fnc» | Сообщение о невозможности считать документы |
| 6 | Путь | «F:\ReportFNSUtility\files\БольшойФайл.fnc» | Вывод данных заголовка, статистики, разблокировка вывода документов |
| 7 | Номер начального документа, номер конечного документа | 11, 1 | Сообщение о некорректном вводе номеров документов |
| 8 | Номер начального документа, номер конечного документа | 1, 151 | Вывод документов в виде древовидной структуры |

## 2.6 Руководство по использованию программы

### 2.6.1 Руководство системного программиста

#### 2.6.1.1 Общие сведения о программе

Модуль дешифровки и вывода данных «Отчета о считывании фискальных данных» является частью программного комплекса «ReportFNSUtility». Модуль предназначен для проверки целостности файла отчёта и корректности данных, записанных в нём.

Модуль выполняет следующие функции:

* считывает файл отчёта;
* проверяет контрольную сумму;
* выводит на экран данные заголовка;
* считывает и выводит данные статистики;
* выводит заданный диапазон документов в элемент древовидного представления информации.

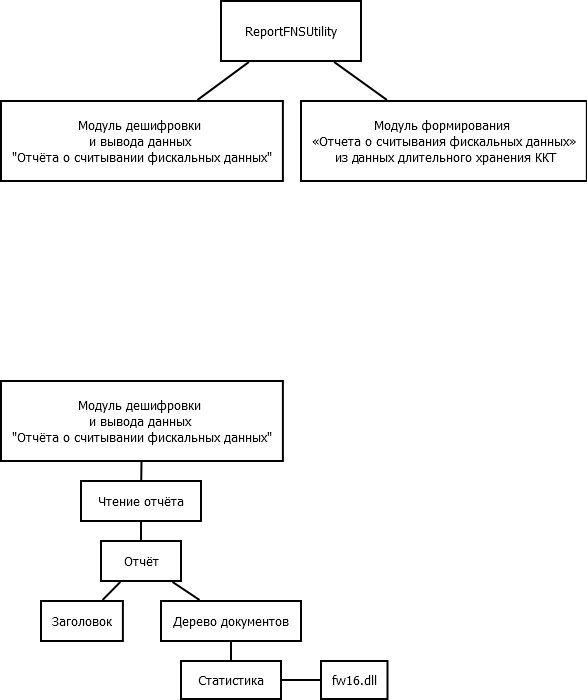
Модуль использует стороннюю библиотеку «Fw16.dll», разрабатываемую компанией «Пилот РБС».

#### 2.6.1.2 Структура программы

На рис. 27 представлена структурная схема разрабатываемого модуля. На ней показано, что модуль включает в себя элемент структуры «чтение отчёта», который в свою очередь включает в себя «Заголовок» и «Дерево документов». «Дерево документов» включает в себя «Статистику», которая использует методы библиотеки «Fw16.dll».

#### 2.6.1.3 Настройка программы

Модуль не требует дополнительных настроек для начала использования.

  
Рис. 27 – Структурная схема модуля

#### 2.6.1.4 Проверка программы

Для проверки работоспособности модуля необходимо считать файл «Отчёта о считывании фискальных данных». Для этого в поле пути вводим абсолютный путь к файлу и нажимаем кнопку «Обновить». В случае обнаружения ошибок в сформированном файле будет выведено соответствующее сообщение (смотрите пункт 2.6.3.4 Сообщения пользователю). В случае удачного прохождения проверок модуль должен вывести данные заголовка в соответствующие поля и после того, как полоса загрузки дойдёт до конца, на вкладке «Статистика» отобразятся данные статистики записанных документов. Помимо этого модуль должен корректно выводить данные о документах в виде древовидной структуры при нажатии на кнопку «Отобразить», очищая поле вывода данных каждый раз перед очередным отображением информации.

#### 2.6.1.5 Сообщения системному программисту

Сообщения системному программисту отсутствуют.

### 2.6.2 Руководство программиста

#### 2.6.2.1 Назначение и условия применения программы

Модуль дешифровки и вывода данных «Отчета о считывании фискальных данных» является частью программного комплекса «ReportFNSUtility». Модуль предназначен для проверки целостности файла отчёта и корректности данных, записанных в нём.

Модуль выполняет следующие функции:

* считывает файл отчёта;
* проверяет контрольную сумму;
* выводит на экран данные заголовка;
* считывает и выводит данные статистики;
* выводит заданный диапазон документов в виде древовидной структуры.

Модуль использует стороннюю библиотеку «Fw16.dll», разрабатываемую компанией «Пилот РБС».

Системные требования для выполнения программного комплекса «ReportFNSUtility»:

- ОС: 32-битная Windows 10, 8, 7, Vista, XP;

- процессор: Intel Core i3-4030U 1.90 GHz;

- ОЗУ: 4 ГБ;

- DirectX: видеокарта, совместимая с версией 11.0 или аналогичная ей;

- свободное место на жестком диске: 512 МБ.

#### 2.6.2.2 Характеристики программы

Инструментальные средства:

* язык программирования C# версии .Net Framework 4.6;
* среда разработки Visual Studio 2017.

В оперативной памяти занимает 17Мб памяти без считанного файла.

#### 2.6.2.3 Обращение к программе

Взаимодействие пользователя с программным модулем происходит посредством запуска исполняемого файла «ReportFNSUtilty.exe».

#### 2.6.2.4 Входные и выходные данные

Входными данными являются:

* путь к файлу отчёта о считывании фискальных данных;
* номер начального документа, с которого необходимо вывести подробную информацию;
* номер конечного документа, с которого необходимо вывести подробную информацию.

Структура входных данных представлена в таблице 2.4.

Таблица 2.4 – Входные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Идентификатор | Тип данных | Размер |
| Путь к файлу отчёта | patch | String | 255 Байт |
| Номер начального документа | \_start | UInt32 | 4 Байта |
| Номер конечного документа | \_end | UInt32 | 4 Байта |

Выходными данными являются:

* данные заголовка отчёта о считывании фискальных данных;
* статистика о количестве и сумме документов;
* подробная информация о записанных документах в отчёте.

Структура выходных данных представлена в таблице 2.5.

Таблица 2.5 – Выходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Идентификатор | Тип данных | Размер |
| Наименование файла выгрузки | Name | String | 53 Байта |
| Программа выгрузки | NameProgram | String | 256 Байт |
| Регистрационный номер ККТ | NumberECR | String | 20 Байт |
| Номер ФН | NumberFS | String | 16 Байт |
| Номер версии ФФД | NumberFFD | byte | 1 Байт |
| Количество смен | CountShift | UInt32 | 4 Байта |
| Количество фискальных документов | CountFiscalDoc | UInt32 | 4 Байта |
| Контрольная сумма файла выгрузки | Hash | UInt32 | 4 Байта |
| Статистика | stats[12] | decimal[] | 128 Байт |
| Документы | item | TreeNode | - |

#### 2.6.2.5 Сообщения

Сообщения программисту отсутствуют.

### 2.6.3 Руководство пользователя

#### 2.6.3.1 Назначение программы

Модуль дешифровки и вывода данных «Отчета о считывании фискальных данных» является частью программного комплекса «ReportFNSUtility». Модуль предназначен для проверки целостности файла отчёта и корректности данных, записанных в нём.

Модуль выполняет следующие функции:

* считывает файл отчёта;
* проверяет контрольную сумму;
* выводит на экран данные заголовка;
* считывает и выводит данные статистики;
* выводит заданный диапазон документов в виде древовидной структуры.

#### 2.6.3.2 Условия выполнения программы

Системные требования для выполнения программного комплекса «ReportFNSUtility»:

- ОС: 32-битная Windows 10, 8, 7, Vista, XP;

- процессор: Intel Core i3-4030U 1.90 GHz;

- ОЗУ: 4 ГБ;

- DirectX: видеокарта, совместимая с версией 11.0 или аналогичная ей;

- свободное место на жестком диске: 512 МБ.

#### 2.6.3.3 Выполнение программы

Взаимодействие пользователя с модулем происходит посредством запуска исполняемого файла программного комплекса «ReportFNSUtilty.exe».

Пользователь попадает в главное окно программного комплекса. Для того, чтобы начать считывать файл «Отчёта о считывании фискальных данных», пользователю необходимо указать путь к файлу. Сделать это можно тремя способами:

* указать в поле абсолютный путь к файлу;
* указать в поле путь к файлу относительно исполняемого файла;
* нажать на кнопку «Обзор» и выбрать файл в диалоговом окне.

В третьем случае пользователь может выбрать файл формата «.fnc» или «.bin». Если отчёт имеет другой формат, пользователю необходимо использовать один из первых двух способов.

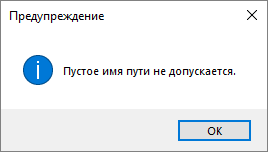
После начала считывания файла в случае нахождения ошибок в файле на экран будет выведено соответствующее сообщение (смотрите пункт 2.6.3.4 Сообщения пользователю). Если файл записан корректно, на форму будут выведены данные заголовка, станет доступно отображение данных документов и начнётся вычисление статистики. Прогресс вычислений отражается на полосе загрузки в нижней части экранной формы. После завершения вычислений статистика будет отображена на вкладке «Статистика».

На вкладке «Дерево» пользователь может вывести данные о документах, записанные в файле «Отчёта о считывании фискальных данных». Для этого пользователю необходимо ввести начальный и конечный номер диапазона документов, который он хочет отобразить на элементе древовидного представления информации и нажать кнопку «Отобразить».

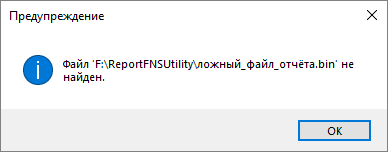
Процесс подсчёта статистики и процесс вывода данных о документах можно при необходимости остановить, нажав на ту же кнопку, поменявшую надпись на «Остановить».

#### 2.6.3.4 Сообщения пользователю

В случае если поле пути не было заполнено, программный модуль выведет на экран сообщение об ошибке, которое можно увидеть на рис. 28.

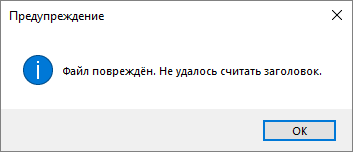
  
Рис. 28 – Предупреждение о пустом имени файла

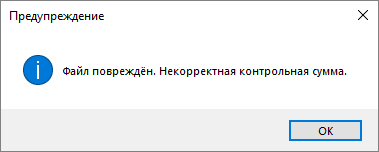
В случае если пользователь ввёл некорректный путь к файлу, будет выведено соответствующее сообщение, пример которого можно увидеть на рис. 29.

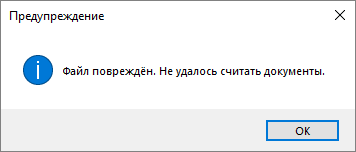
  
Рис. 29 – Предупреждение о несуществующем файле

На рис. 30–32 представлены сообщения о трёх возможных ошибках в файле «Отчёта о считывании фискальных данных», а именно:

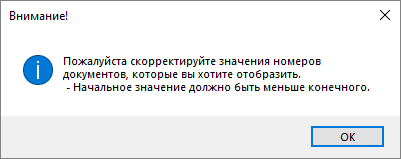
* некорректный заголовок;
* несоответствие контрольной суммы;
* повреждение документов.

  
Рис. 30 – Предупреждение о некорректном заголовке

  
Рис. 31 – Предупреждение о некорректной контрольной сумме

  
Рис. 32 – Предупреждение о повреждении записи документов

На рис. 33 представлено сообщение об ошибке некорректного заполнения полей, отвечающих за диапазон выводимых документов.

  
Рис. 33 – Предупреждение о некорректном заполнении номеров выводимых документов

# Заключение

В начале разработки передо мной стояло множество сложных задач, которые необходимо было решить. Важно было изучить абсолютно новую для меня обширную предметную область, изучить механизм реализации многопоточности на языке программирования C# и некоторые не используемые мною ранее конструкции.

Для начала мне пришлось изучить юридический аспект взаимодействия с контрольно-кассовой техникой. На данный момент ФНС России постепенно обязывает все юридические лица использовать ККТ при торговле за наличный расчёт или по банковской карте. К 1 июля 2019 года это требование станет обязательным для всех без исключения.

В процессе подготовки выпускной квалификационной работы мною было выявлено, что ККТ имеет три разных режима регистрации. Для регистрации без отправки фискальных данных, а именно в «Автономном режиме», необходимо оформить соответствующий документ, в котором указано, что торговля происходит в удалённом регионе и возможности передавать данные по сети не представляется возможным. При такой регистрации необходимо формировать отчёт о считывании фискальных данных и вместе с отчётом на бумажном носителе предоставлять в ФНС России.

Следующим шагом изучения предметной области было ознакомление со структурой отчёта, используемыми типами и структурами данных. Мною были рассмотрены готовые файлы «Отчёта о считывании фискальных данных» и изучен приказ ММВ-7-20/229, в котором подробно расписаны все аспекты формирования «Отчёта о считывании фискальных данных».

Отчёт состоит из заголовка и документов, записанных в виде TLV- и STLV-структур. Заголовок, в свою очередь, состоит из восьми полей определённой длины и типа данных.

После обследования предметной области мне необходимо было изучить механизм реализации многопоточности на языке программирования C#. Этот механизм был важен в реализации модуля, так как при больших объёмах данных экранная форма переставала бы «отвечать». Многопоточность позволила мне реализовать параллельное считывание статистики и взаимодействие пользователя с формой. Изучение данного механизма далось мне непросто, но сильно расширило возможности разрабатываемого мною программного обеспечения.

Следующим этапом было создание диаграмм и схем. После тщательного изучения предметной области и изучения сложных конструкций языка C# выполнение данного этапа не составил для меня никакого труда.

По разработанным схемам и диаграммам реализация модуля заняла намного меньше времени, чем бы она заняла при разработке без предварительного проектирования.

В ходе промежуточного тестирования были устранены все недочёты в реализации, а также опечатки. Итоговое тестирование не выявило ошибок.

В результате проделанной работы была изучена обширная предметная область, а именно использование ККТ в торговле. Изучены ранее не известные мне конструкции языка программирования C#, а также был разработан программный модуль, который выполняет следующие функции:

* считывает файл отчёта;
* проверяет контрольную сумму;
* выводит на экран данные заголовка;
* считывает и выводит данные статистики;
* выводит заданный диапазон документов в виде древовидной структуры.

Разработанный модуль отвечает всем заявленным требованиям, впоследствии он будет изменяться и дополняться в зависимости от изменений в законодательстве РФ, требований современной экономики и пользователей. Считаю выпускную работу успешной и завершённой.

# Список литературы

Законодательные и нормативные акты:

1. ГОСТ 2.316-2008 Правила нанесения надписей, технических тебований и таблиц на графических документах
2. ГОСТ 7.1. – 2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2004. – 169 с.
3. ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание.
4. ГОСТ 7.32 – 2001. Система стандартов по информацию, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: ИПК Издательство стандартов, 2001. – 21 с.
5. ГОСТ 712-93 Библиографическая запись. Сокращение слов на русском языке.
6. ГОСТ 8 417 2002 Единицы величин
7. ГОСТ Р 21.1101-2013 Основные требования к проектной и рабочей документации
8. ГОСТ СН 528-80 Перечень единиц физических величин
9. ГОСТ\_2.105-95 Общие требования к текстовым документам
10. Единая система программной документации. – М.: Стандартинформ, 2005. – 128 с.
11. Приказ ФНС России от 21.03.2017 N ММВ-7-20/229@ "Об утверждении дополнительных реквизитов фискальных документов и форматов фискальных документов, обязательных к использованию" – Режим доступа: http://www.consultant.ru/cons/cgi/online.cgi
12. Федеральный закон от 22 мая 2003 года N 54-ФЗ "О применении контрольно-кассовой техники при осуществлении наличных денежных расчетов и расчетов с использованием платежных карт" – [Электронный ресурс]. – Режим доступа: http://www.consultant.ru/cons/cgi/online.cgi

Учебная и научная литература:

1. Гуриков С.Р. Введение в программирование на языке Visual C#: учебное пособие. – М.: ФОРУМ: ИНФРА-М, 2013.
2. Макконелл С. Совершенный код – Microsoft Press, Русская Редакция, 2013.
3. Рудаков А.В. Технология разработки программных продуктов: учебник для студ. учреждений сред. проф. образования. – М.: Академия, 2013.
4. Рудаков А.В., Федорова Г.Н. Технология разработки программных продуктов. Практикум: учеб. пособие для студ. учреждений сред. проф. образования. – М.: Академия, 2013.
5. Билл Вагнер. Наиболее эффективное программирование на C#. 50 способов улучшения кода. – Вильямс, 2017
6. Богачёв К.Ю. Программирование. Основы параллельного программирования. – Бином. Лаборатория знаний, 2015

Интернет-источники

1. Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7. – [Электронный ресурс]. – Режим доступа: http://metanit.com/sharp/tutorial/

# Приложение А

СХЕМЫ И ДИАГРАММЫ

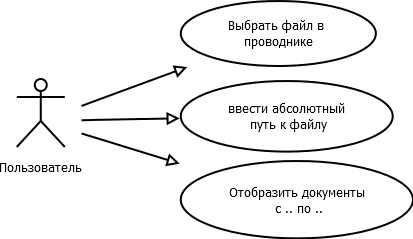
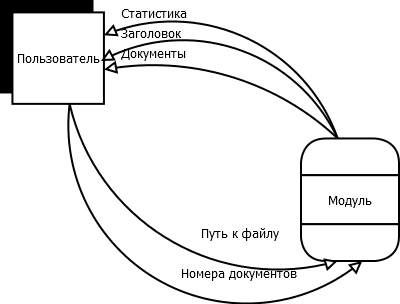
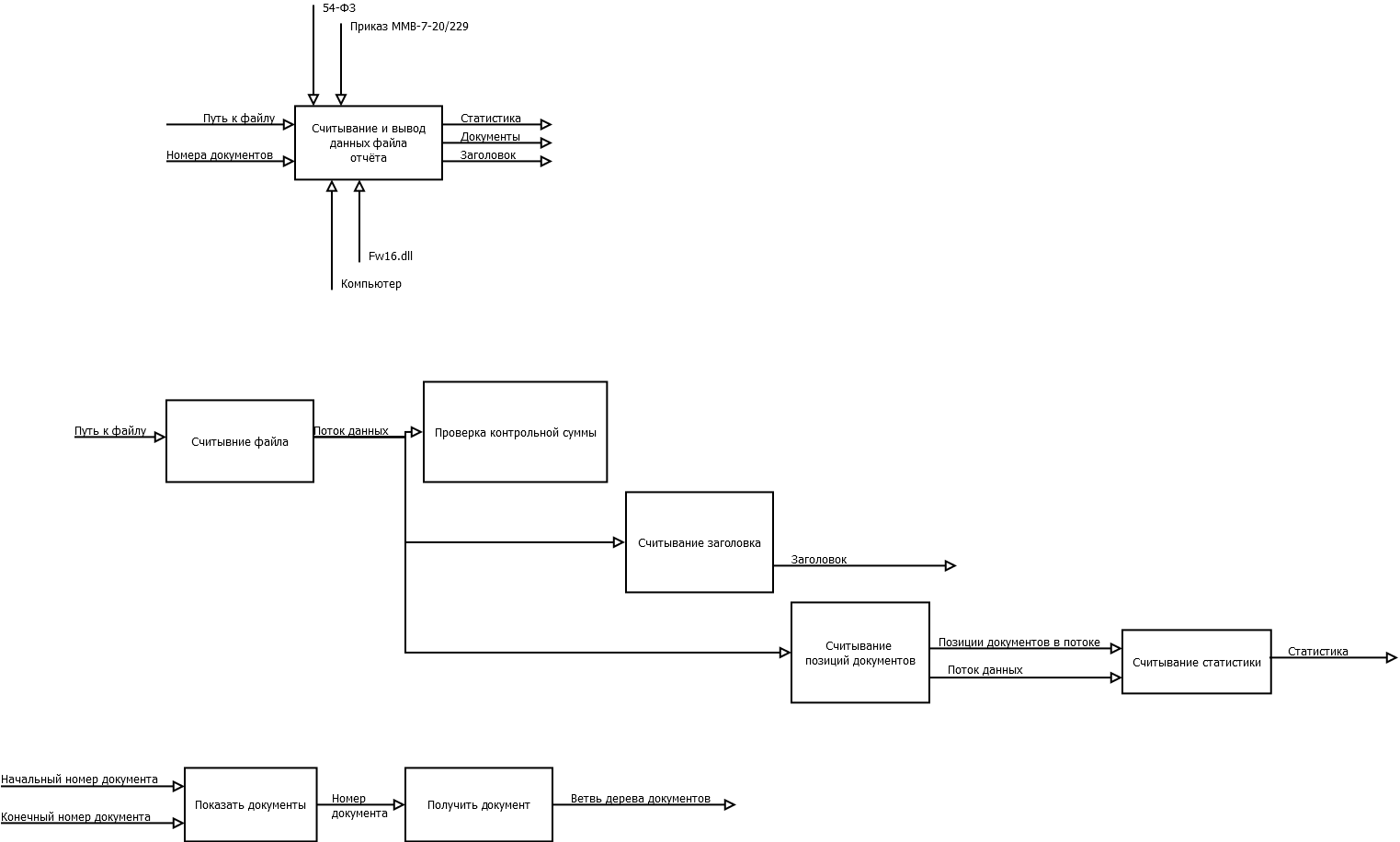
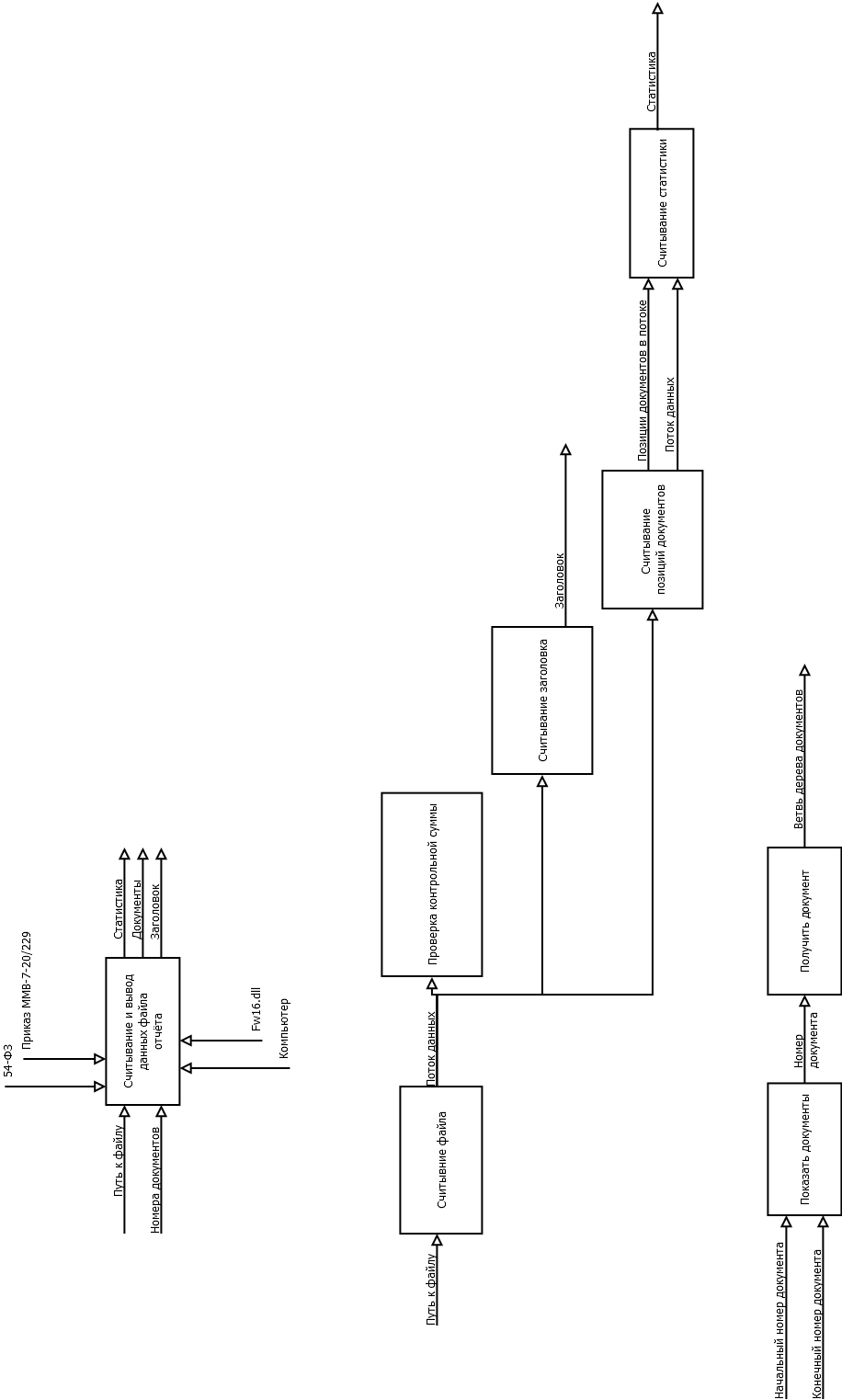
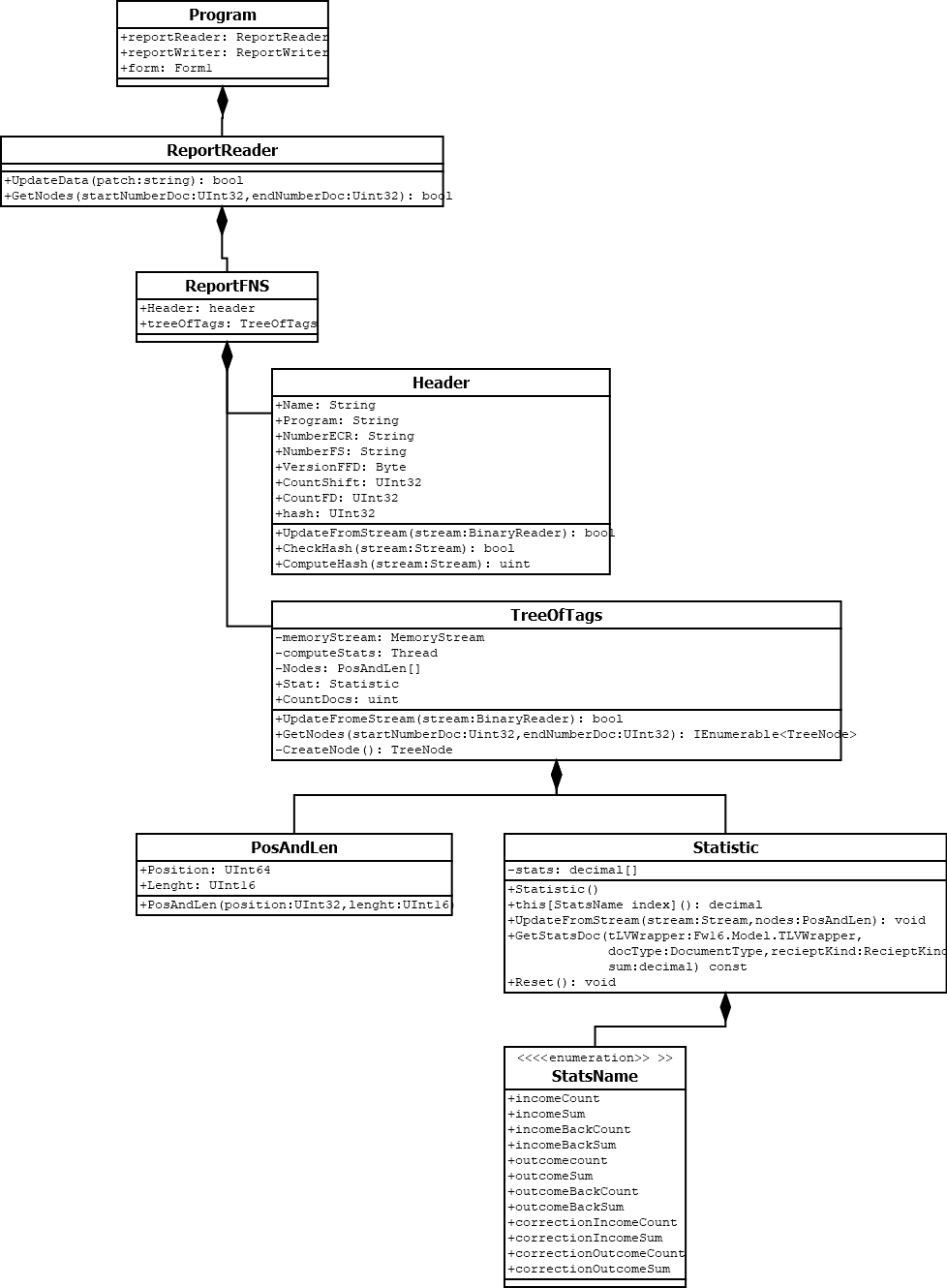


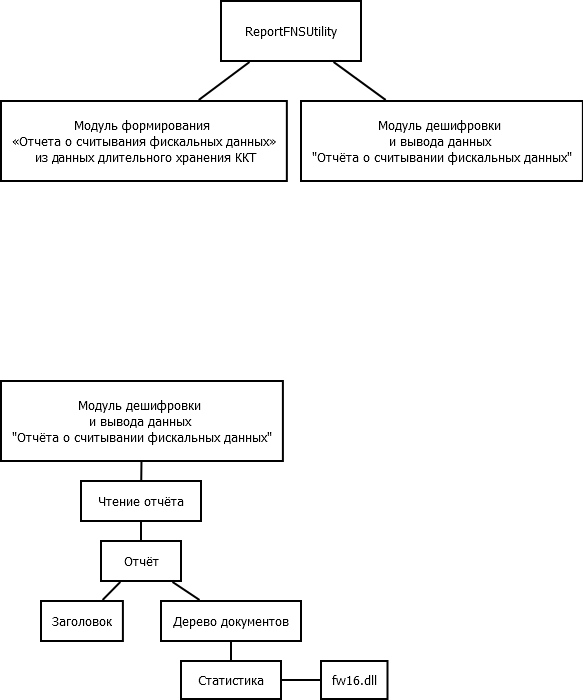
Рис. А.1 – Диаграмма вариантов использования

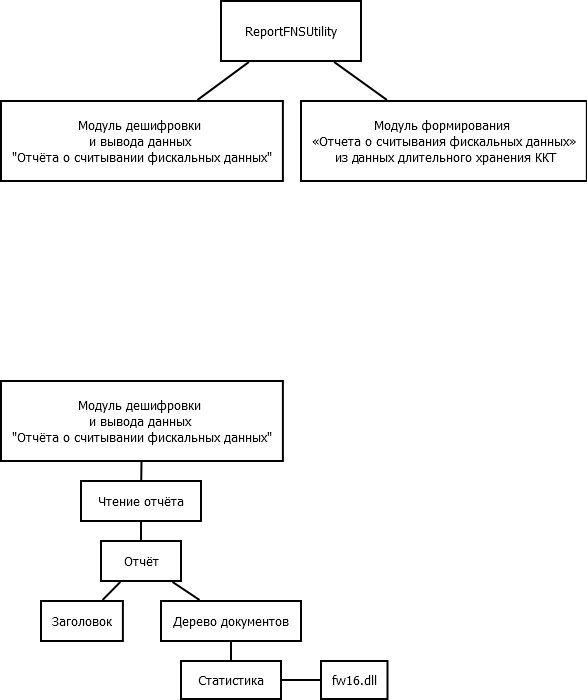
  
Рис. А.2 – Диаграмма потоков данных (контекстная)

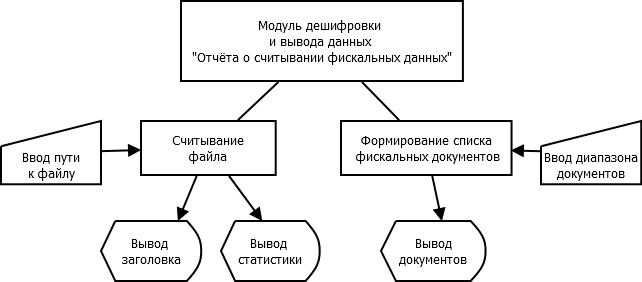
  
Рис. А.3 – Функциональная диаграмма (контекстная)

  
Рис. А.4 – Функциональная диаграмма (детализированная)

  
Рис. А.5 – Диаграмма классов

  
Рис. А.6 – Структурная схема комплекса «ReportFNSUtility»

  
Рис. А.7 – Структурная схема модуля

  
 Рис. А.8 – Функциональная схема

# Приложение Б

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

**Общие сведения**

В данном документе создаваемое программное обеспечение называется «Утилита формирования отчёта о считывании фискальных документов» (далее – Утилита). Принято решение разрабатывать данную утилиту в связи с обновлением требований по выгрузке фискальных данных описанных в «Приказе ФНС России от 21.03.2017 N ММВ-7-20/229@».

**Цели создания утилиты.**

1. Формировать Отчет о считывании фискальных данных всех фискальных документов, содержащихся в ФН (далее – Отчет о считывании ФД).
2. Расшифровывать бинарный файл отчёта для просмотра его содержимого.

**Требования к программному обеспечению**

Утилита должна отвечать следующим требованиям:

1. Формировать файл отчета путём вызова из командной строки:

* С указанием параметров, необходимых для подключения к ККТ.
* С выводом данных в файл по указанному пути с именем <НомерФН\_YYMMDD>.bin.
* С возможностью перезаписи имеющегося файла только при наличии явно заданного параметра «перезаписать имеющийся файл».

1. При запуске без параметров – отображать пользовательский интерфейс (окно), позволяющий:

* Подключиться к ККТ и сформировать файл отчета с указанным пользователем именем по указанному им пути.
* «открыть» выбранный пользователем, ранее созданный файл отчета для просмотра его содержимого.
* Режим просмотра файла отчета должен позволить пользователю быстро находить значимую информацию отчета путем фильтрации\поиска.

1. В файл отчёта о считывании ФД необходимо занести:
   1. Заголовок отчета о считывании ФД
   2. Фискальные документы

* Отчёты о регистрации
* Отчет об изменении параметров регистрации
* Отчет о закрытии фискального накопителя
  1. Фискальные данные ФД
* Отчет об открытии смены
* Отчета о текущем состоянии расчетов
* Кассовый чек
* Кассовый чек коррекции
* БСО
* БСО коррекции
* Отчет о закрытии смены
* Подтверждение оператора

Структуры записи вышеперечисленных документов в отчёт о считывании ФД описаны в таблицах в прилагаемом файле «Структура отчёта» с перечислением тегов реквизитов.

**Диаграмма вариантов использования**

Пользователь может запустить программное обеспечение двумя способами   
(Рис. 1):

1. Через exe-файл. Будет запущена windows форма, с помощью которой пользователь сможет формировать отчёт или открывать уже сформированные файлы для просмотра записанной информации в наглядном виде.
2. Через консоль с передачей параметров. Пользователь может запустить программу из консоли для формирования отчёта без вызова windows формы.

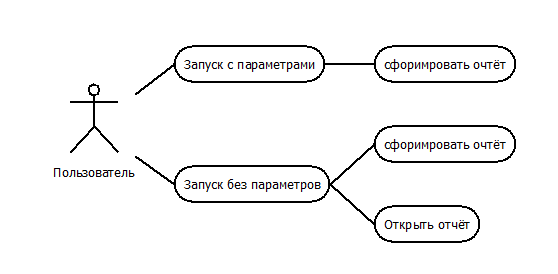


Рис. 1 – Диаграмма вариантов использования

**Схематичное представление данных отчёта.**

В объекте отчёта будут два поля заголовок и данные длительного хранения. Данные длительного хранения содержат в себе все виды документов перечисленные в приложении к приказу 2620-2 (Рис. 2).

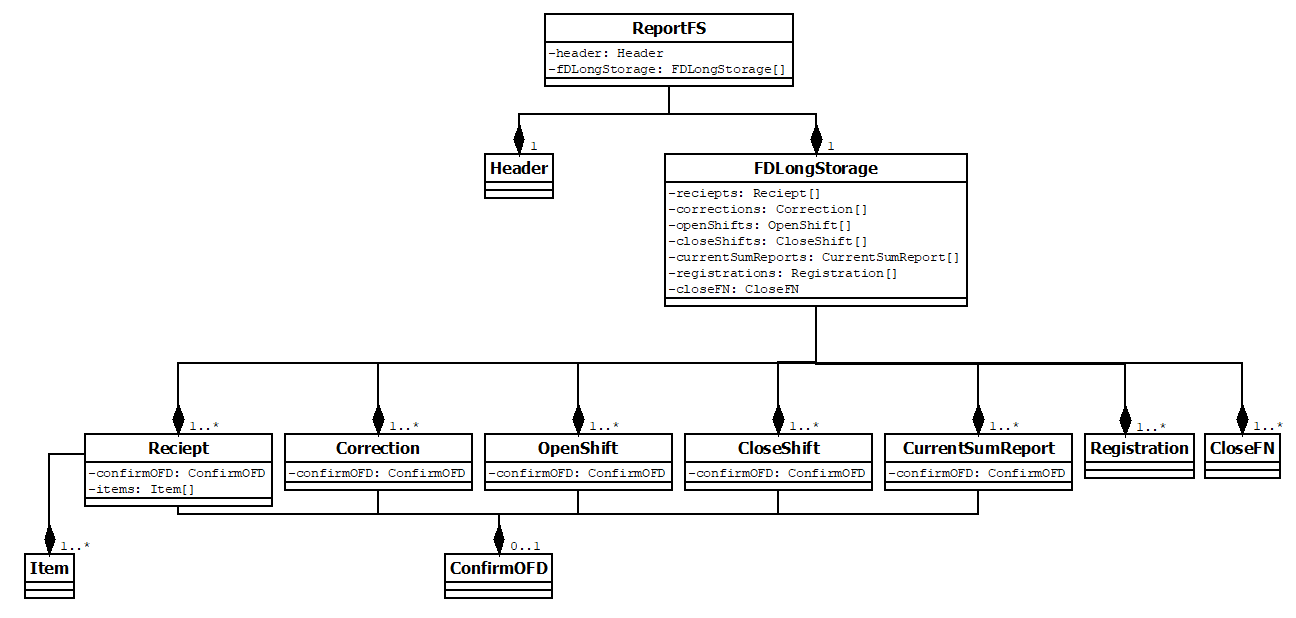


Рис. 2 – Бизнес-схема представления данных в утилите

**Схема классов решения**

В утилите вышеописанная структура будет представлена 5 классами отчёт, заголовок, базовый класс STLV структуры и два дочерних представляющие собой реализацию TLV и STLV структур в утилите (Рис. 3).

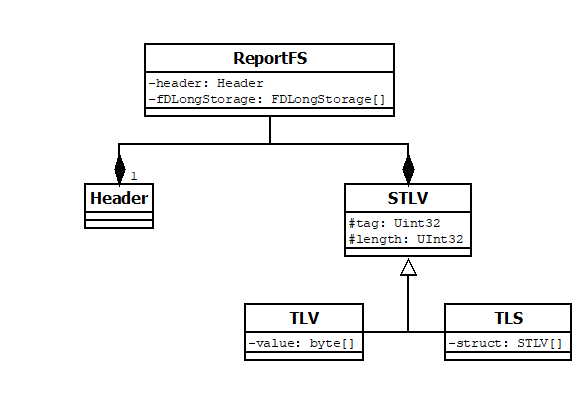


Рис. 3 – Схема классов решения

**Диаграмма потоков данных (контекстная)**

На рис. 4 изображена диаграмма потоков данных. Пользователь вводит путь к создаваемому файлу, способ подключения и ReportFNSUtility создаёт файл отчёта или вводит путь к файлу и способ чтения файла и ReportFNSUtility выводит содержимое файла.

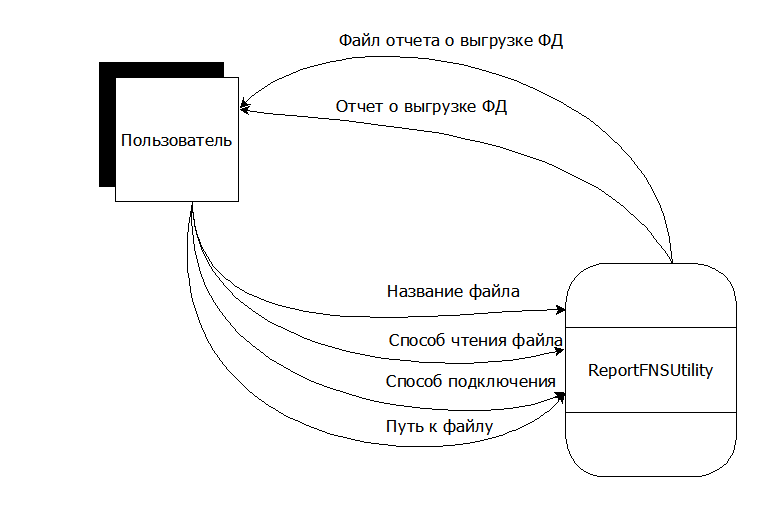


Рис. 4 – Диаграмма потоков данных

**Диаграмма потоков данных (детализированная)**

На рис. 5 изобрежена детализированная диаграмма потоков данных. Пользователь заполняет поля название файла и путь к файлу и выбирает способ чтения файла. Потом после выбора режима (чтение или запись файла) при выборе чтения название файла передаётся в процедуру заполнения дерева тегов, которая в свою очередь передаёт сформированное дерево тегов в процедуру вывода отчёта. При выборе записи пользователь вводит данные о способе подключения, и процедура, приняв название файла из процедуры выбора режима формирует дерево тегов и передаёт в процедуру создания файла.

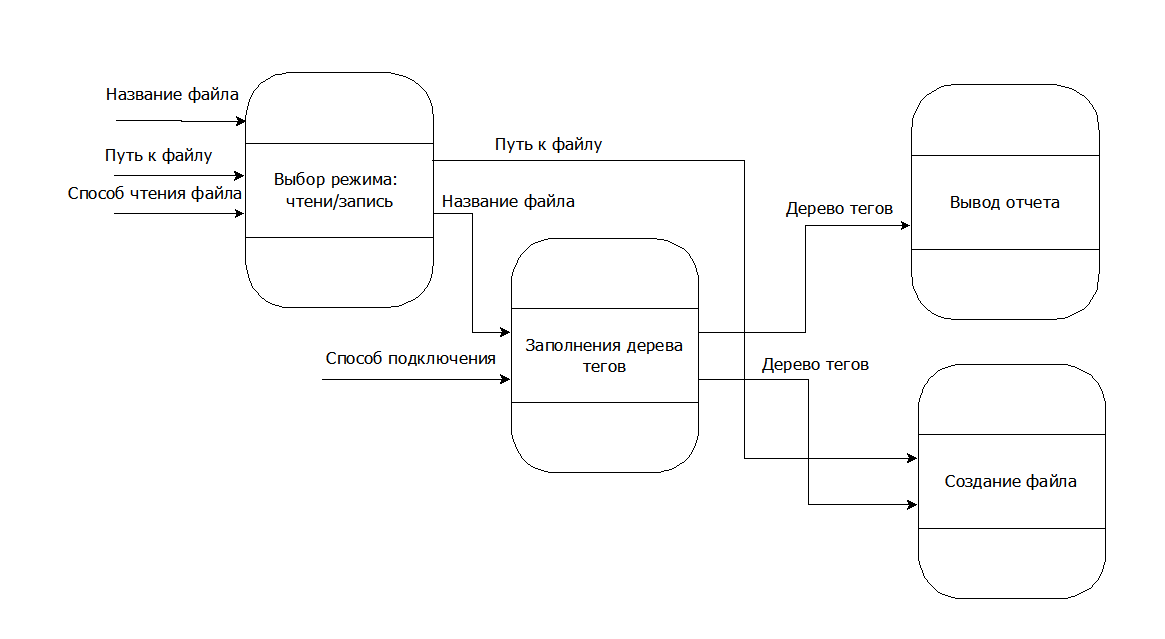


Рис. 5 – Детализированная диаграмма потоков данных

**Структурная схема**

На рис. 6 изображена структурная схема ReportFNSUtility. Пользователь может взаимодействовать с модулем чтения отчёта или с модулем написания отчёта.

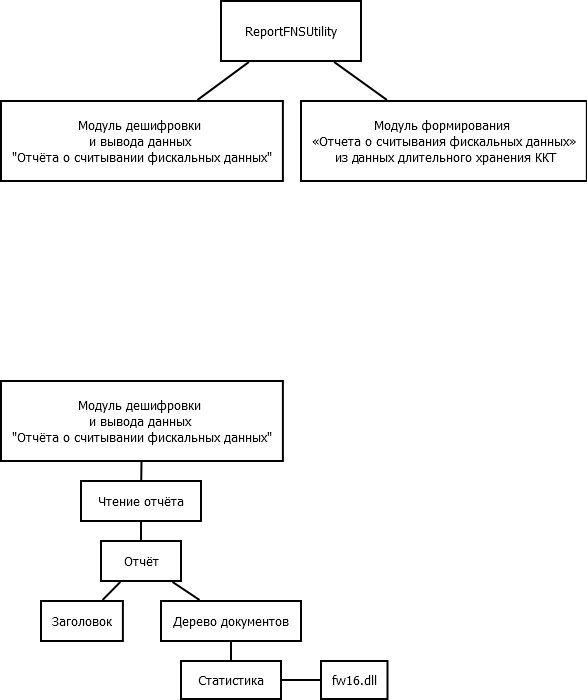


Рис. 6 – Структурная схема

**Функциональная схема**

На рис. 7 изображена функциональная схема. ReportFNSUtility включает в себя графическую оболочку используя которую пользователь вводит данные. В утилите имеются два модуля: модуль вывода отчёта и модуль создания отчёта. Модули формируют древо тегов, которое выводится в файл или на форму.

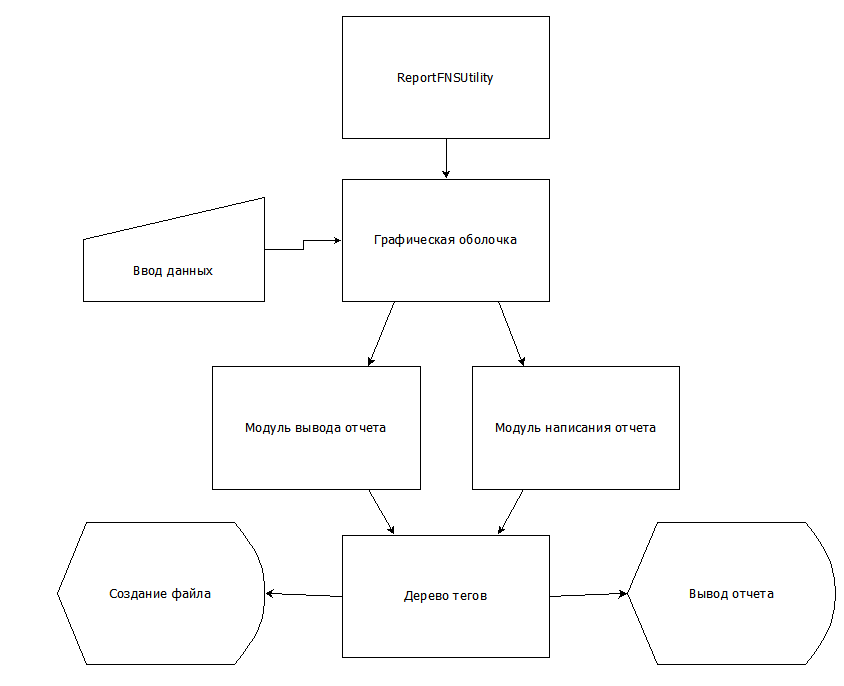


Рис. 7 – Функциональная схема

**Входные параметры при запуске из командной строки.**

Ниже перечислены возможные параметры при запуске из командной строки.

rw – перезаписать файл отчёта при совпадении имени

P<Номер\_Порта > - порт подключения к ККТ

R<Частота> - частота порта подключения к ККТ (Возможно только при указании порта)

D”<Абсолютный\_Путь\_К\_Директории >” – Путь к директории в которой будет создан файл отчёта.

**Пользовательский интерфейс**

Черновое представление windows формы (Рис. 4).

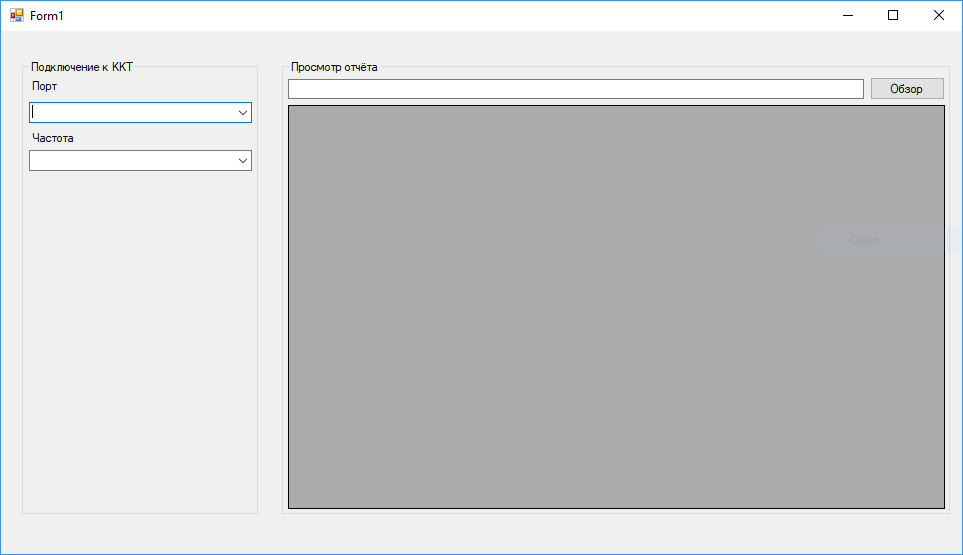


Рис. 4 – Пользовательский интерфейс

**План разработки**

Реализовать функционал расшифровки данных из файла отчёта

Реализовать функционал считывания данных из ККТ

Реализовать сортировку и фильтрацию данных выводимых в windows форму.

# Приложение В

ИСХОДНЫЙ КОД

Файл Form1.cs:

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using static ReportFNSUtility.ReportFNS.TreeOfTags.Statistic;

namespace ReportFNSUtility

{

public partial class Form1 : Form

{

Thread readReportThread, writeReportThread, showNodesThread;

public Thread computeStats;

WriteReport writeReport;

public static Form1 form = null;

Fw16.EcrCtrl ecrCtrl;

public Form1()

{

InitializeComponent();

form = this;

this.Text = Program.nameProgram;

TV\_TreeTags.TreeViewNodeSorter = new TreeSorter();

OpenFD\_binFile.InitialDirectory = Application.StartupPath;

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

Program.reportFNS.treeOfTags.StopComputeStats();

if (writeReportThread?.IsAlive ?? false)

{

writeReportThread?.Abort();

writeReportThread.Join();

(writeReport.ecrCtrl as IDisposable).Dispose();

writeReport.fileStream?.Close();

writeReport.writer?.Close();

}

if (readReportThread?.IsAlive ?? false)

{

readReportThread?.Abort();

readReportThread.Join();

}

}

public void UpdateProgressBar(int val, int max = 100)

{

progressBar1.Maximum = max;

if (val <= max)

{

progressBar1.Value = val;

}

else

{

progressBar1.Value = max;

}

}

private void Form1\_FormClosed(object sender, FormClosedEventArgs e)

{

(ecrCtrl as IDisposable)?.Dispose();

}

public void ReadHeader()

{

var \_header = Program.reportFNS.reportHeader;

TB\_Name.Text = \_header.Name;

TB\_Program.Text = \_header.NameProgram;

TB\_NumberECR.Text = \_header.NumberECR;

TB\_NumberFS.Text = \_header.NumberFS;

TB\_VersionFFD.Text = \_header.VersionFFD.ToString();

TB\_CountShift.Text = \_header.CountShift.ToString();

TB\_CountFiscalDoc.Text = \_header.CountFiscalDoc.ToString();

TB\_Hash.Text = \_header.Hash.ToString("X");

}

private void ClearHeder()

{

TB\_Name.Text = "";

TB\_Program.Text = "";

TB\_NumberECR.Text = "";

TB\_NumberFS.Text = "";

TB\_VersionFFD.Text = "";

TB\_CountShift.Text = "";

TB\_CountFiscalDoc.Text = "";

TB\_Hash.Text = "";

}

public void ReadStats()

{

var \_stat = Program.reportFNS.treeOfTags.stat;

TB\_CorrectionIncomeCount.Text = \_stat[StatsName.correctionIncomeCount].ToString();

TB\_CorrectionIncomeSum.Text = \_stat[StatsName.correctionIncomeSum].ToString("0.00");

TB\_CorrectionOutcomeCount.Text = \_stat[StatsName.correctionOutcomeCount].ToString();

TB\_CorrectionOutcomeSum.Text = \_stat[StatsName.correctionOutcomeSum].ToString("0.00");

TB\_IncomeBackCount.Text = \_stat[StatsName.incomeBackCount].ToString();

TB\_IncomeBackSum.Text = \_stat[StatsName.incomeBackSum].ToString("0.00");

TB\_IncomeCount.Text = \_stat[StatsName.incomeCount].ToString();

TB\_IncomeSum.Text = \_stat[StatsName.incomeSum].ToString("0.00");

TB\_OutcomeBackCount.Text = \_stat[StatsName.outcomeBackCount].ToString();

TB\_OutcomeBackSum.Text = \_stat[StatsName.outcomeBackSum].ToString("0.00");

TB\_OutcomeCount.Text = \_stat[StatsName.outcomeCount].ToString();

TB\_OutcomeSum.Text = \_stat[StatsName.outcomeSum].ToString("0.00");

}

private void NUD\_Validating(object sender, CancelEventArgs e)

{

if((sender as Control).Text == "") {

(sender as NumericUpDown).Value= (sender as NumericUpDown).Minimum;

(sender as Control).Text = (sender as NumericUpDown).Minimum.ToString();

}

}

////////////////////////////////////////////////////////////////

/// <summary>

/// Обработка нажатия на кнопку "обзор". Вывод окна выбора файла.

/// </summary>

private void B\_Browse\_Click(object sender, EventArgs e)

{

if (OpenFD\_binFile.ShowDialog() == DialogResult.OK)

TB\_Patch.Text = OpenFD\_binFile.FileName;

}

/// <summary>

/// Запускает или останавливает считывание файла в зависисмости от состояния потока.

/// </summary>

public void B\_UpdateStop\_Click(object sender, EventArgs e)

{

if ((readReportThread?.IsAlive ?? false) || (Program.reportFNS.treeOfTags.computeStats?.IsAlive ?? false))

{

Program.reportFNS.treeOfTags.StopComputeStats();

if (readReportThread?.IsAlive ?? false)

{

readReportThread?.Abort();

readReportThread.Join();

}

B\_UpdateStop.Text = "Обновить";

}

else

{

try

{

if (showNodesThread?.IsAlive ?? false)

{

showNodesThread.Abort();

showNodesThread.Join();

B\_ShowNodes.Text = "Отобразить";

}

B\_ShowNodes.Enabled = false;

ClearHeder();

TV\_TreeTags.Nodes.Clear();

readReportThread = new Thread((ThreadStart)delegate

{

try

{

Program.reportFNS.treeOfTags.StopComputeStats();

Program.reportReader.UpdateData(TB\_Patch.Text);

if (Program.reportFNS.treeOfTags.CountDocs > 0)

{

Program.form?.Invoke((MethodInvoker)delegate

{

B\_ShowNodes.Enabled = true;

NUD\_EndNumberDoc.Maximum = NUD\_StartNumberDoc.Maximum = Program.reportFNS.treeOfTags.CountDocs;

});

}

Program.form?.Invoke((MethodInvoker)delegate { ReadHeader(); });

}

catch (Exception ex)

{

Program.form?.Invoke((MethodInvoker)delegate

{

B\_ShowNodes.Enabled = false;

if (!(ex is ThreadAbortException))

MessageBox.Show(ex.Message, "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

B\_UpdateStop.Text = "Обновить";

});

}

});

readReportThread.Start();

B\_UpdateStop.Text = "Остановить";

}

catch (Exception ex)

{

if (!(ex is ThreadAbortException))

MessageBox.Show(ex.Message);

}

}

}

/// <summary>

/// Запускает или останавливает заполнение дерева в зависимости от состояния потока.

/// </summary>

private void B\_ShowNodes\_Click(object sender, EventArgs e)

{

if (showNodesThread?.IsAlive ?? false)

{

showNodesThread?.Abort();

showNodesThread.Join();

B\_ShowNodes.Text = "Отобразить";

}

else

{

try

{

TV\_TreeTags.Nodes.Clear();

UInt32 \_start = (UInt32)NUD\_StartNumberDoc.Value;

UInt32 \_end = (UInt32)NUD\_EndNumberDoc.Value;

if (\_start <= \_end && \_start != 0 && \_end != 0)

{

\_start--;

\_end--;

showNodesThread = new Thread((ThreadStart)delegate

{

if (!Program.reportReader.GetNodes(\_start, \_end))

{

Program.form?.Invoke((MethodInvoker)delegate

{

MessageBox.Show("Не удалось обновить дерево тегов согласно переданным параметрам.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

});

}

Program.form?.Invoke((MethodInvoker)delegate

{

B\_ShowNodes.Text = "Отобразить";

UpdateProgressBar(0);

});

});

showNodesThread.Start();

B\_ShowNodes.Text = "Остановить";

}

else

{

MessageBox.Show("Пожалуйста скорректируйте значения номеров \nдокументов, которые вы хотите отобразить.\n - Начальное значение должно быть меньше конечного.", "Внимание!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

/////////////////////////////////////////////////////////////////////////////////

}

}

Файл ReadReports.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Windows.Forms;

namespace ReportFNSUtility

{

class ReportReader

{

/// <summary>

/// Обновление данных в заголовке и дереве тегов из входной строки

/// </summary>

/// <param name="path">Строка, указывающая абсолютный путь к файлу</param>

public void UpdateData(string path)

{

FileStream \_fs = new FileStream(path, FileMode.Open);

MemoryStream stream = new MemoryStream();

\_fs.CopyTo(stream);

\_fs.Close();

try

{

if (!Program.reportFNS.reportHeader.UpdateFromStream(new BinaryReader(stream)))

{

throw new Exception("Файл повреждён. Не удалось считать заголовок.");

}

if (!Program.reportFNS.reportHeader.ChekHash(stream))

{

throw new Exception("Файл повреждён. Некорректная контрольная сумма.");

}

if (!Program.reportFNS.treeOfTags.UpdateFromStream(new BinaryReader(stream)))

{

throw new Exception("Файл повреждён. Не удалось считать документы.");

}

}

catch (Exception ex)

{

throw ex;

}

finally

{

stream.Close();

}

}

/// <summary>

/// Выводит документы в treeView

/// </summary>

/// <param name="startIndexDoc">Начальный индекс документа</param>

/// <param name="endIndexDoc">Конечный индекс документа</param>

/// <returns>УСпешность завершения операции</returns>

public bool GetNodes(UInt32 startIndexDoc, UInt32 endIndexDoc)

{

foreach (var item in Program.reportFNS.treeOfTags.GetNodes(startIndexDoc, endIndexDoc))

{

Program.form.Invoke((MethodInvoker)delegate { Program.form.TV\_TreeTags.Nodes.Add(item); });

}

return true;

}

}

}

Файл struct.cs:

using DamienG.Security.Cryptography;

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ReportFNSUtility

{

class ReportFNS

{

private ReportHeader \_reportHeader;

private TreeOfTags \_treeOfTags;

public ReportHeader reportHeader { get => \_reportHeader; }

public TreeOfTags treeOfTags { get => \_treeOfTags; }

public ReportFNS()

{

\_reportHeader = new ReportHeader();

\_treeOfTags = new TreeOfTags();

}

public class ReportHeader

{

private string name;

public string Name

{

get => name;

set

{

if (value?.Length >= 53)

{

this.name = value.Substring(0, 53);

}

else

{

this.name = string.Format($"{value,-53}");

}

}

}

private string nameProgram;

public string NameProgram

{

get => nameProgram;

set

{

if (value?.Length >= 256)

{

this.nameProgram = value.Substring(0, 256);

}

else

{

this.nameProgram = string.Format($"{value,-256}");

}

}

}

private string numberECR;

public string NumberECR

{

get => numberECR;

set

{

if (value?.Length >= 20)

{

this.numberECR = value.Substring(0, 20);

}

else

{

this.numberECR = string.Format($"{value,-20}");

}

}

}

private string numberFS;

public string NumberFS

{

get => numberFS;

set

{

if (value?.Length >= 16)

{

this.numberFS = value.Substring(0, 16);

}

else

{

this.numberFS = string.Format($"{value,-16}");

}

}

}

private byte versionFFD;

public byte VersionFFD

{

get => versionFFD;

set => versionFFD = value;

}

private uint countShift;

public uint CountShift

{

get => countShift;

set => countShift = value;

}

private uint countFiscalDoc;

public uint CountFiscalDoc

{

get => countFiscalDoc;

set => countFiscalDoc = value;

}

private uint hash;

public uint Hash

{

get => hash;

}

/// <summary>

/// Обновляет данные заголовка согласно переданному потоку чтения

/// </summary>

/// <param name="stream">поток чтения</param>

/// <returns></returns>

public bool UpdateFromStream(BinaryReader stream)

{

//Проеверка длиннны потока на присутствие в ней заголовка.

if (stream.BaseStream.Length >= 358)

{

stream.BaseStream.Seek(0, SeekOrigin.Begin);

//Считывание название документа

Encoding encoding = Encoding.GetEncoding(866);

byte[] name = new byte[53];

stream.Read(name, 0, 53);

Name = encoding.GetString(name);

//Считывание названия программы

byte[] programm = new byte[256];

stream.Read(programm, 0, 256);

NameProgram = encoding.GetString(programm);

//Считывание номера ККТ

byte[] numberECR = new byte[20];

stream.Read(numberECR, 0, 20);

NumberECR = encoding.GetString(numberECR);

//Считывание номер фискального накопителя

byte[] numberFS = new byte[16];

stream.Read(numberFS, 0, 16);

NumberFS = encoding.GetString(numberFS);

//Считывания версии ФФД, количества смен, количества фискальных документов и хеша

this.versionFFD = stream.ReadByte();

this.countShift = stream.ReadUInt32();

this.countFiscalDoc = stream.ReadUInt32();

this.hash = stream.ReadUInt32();

return true;

}

else

{

return false;

}

}

/// <summary>

/// Сверяет хеш написанный в файле и посчитанный программой.

/// </summary>

/// <param name="stream">Поток данных для вычисления и вычленения хеша</param>

/// <returns>bool, совпадают ли хеши.</returns>

public bool ChekHash(Stream stream)

{

if (stream.Length >= 358)

{

//подготовка

long \_pos = stream.Position;

stream.Seek(0, SeekOrigin.Begin);

MemoryStream \_tmpMemory = new MemoryStream();

byte[] \_tmpBytes = new byte[354];

//копирование потока в поток памяти без хеша

stream.Read(\_tmpBytes, 0, 354);

\_tmpMemory.Write(\_tmpBytes, 0, 354);

stream.Read(\_tmpBytes, 0, 4);

UInt32 \_hashInStream = BitConverter.ToUInt32(\_tmpBytes, 0);

stream.CopyTo(\_tmpMemory);

//Возвращаем позицию в потоке в исходную

stream.Seek(\_pos, SeekOrigin.Begin);

//Вычисление хеша и закрытие потока

uint \_hashCompute = ComputeHesh(\_tmpMemory);

\_tmpMemory.Close();

//Возврат результата сравнения

return \_hashCompute == \_hashInStream;

}

else

{

return false;

}

}

/// <summary>

/// Возвращает хеш CRC-32 для переданного потока

/// </summary>

/// <param name="stream">Поток данных</param>

/// <returns>Uint, Хеш для переданного потока</returns>

public static uint ComputeHesh(Stream stream)

{

long \_pos = stream.Position;

stream.Seek(0, SeekOrigin.Begin);

Crc32 crc32 = new Crc32();

byte[] \_hash = crc32.ComputeHash(stream);

Array.Reverse(\_hash);

stream.Seek(\_pos, SeekOrigin.Begin);

return BitConverter.ToUInt32(\_hash, 0);

}

}

public class TreeOfTags

{

MemoryStream memoryStream;

public Thread computeStats;

PosAndLen[] Nodes;

public readonly Statistic stat;

public uint CountDocs { get => (uint)Nodes.Length; }

public TreeOfTags()

{

stat = new Statistic();

}

/// <summary>

/// Создаёт копию переданного потока данных и вычисляет все позиции документов в потоке. После создаёт поток формирования статистики.

/// </summary>

/// <param name="stream"></param>

/// <returns></returns>

public bool UpdateFromStream(BinaryReader stream)

{

//подготовка переменных

memoryStream?.Close();

memoryStream = new MemoryStream();

//копирование потоков, создание потока чтения

stream.BaseStream.Seek(358, SeekOrigin.Begin);

stream.BaseStream.CopyTo(memoryStream);

if (memoryStream.Length < 8) return false;

memoryStream.Seek(0, SeekOrigin.Begin);

BinaryReader streamReader = new BinaryReader(memoryStream);

//формирование массива позиций и длинн документов в потоке

List<PosAndLen> \_tmpList = new List<PosAndLen>();

while (streamReader.BaseStream.Position != streamReader.BaseStream.Length)

{

UInt16 tag = streamReader.ReadUInt16();

UInt16 len = streamReader.ReadUInt16();

if (len > streamReader.BaseStream.Length - streamReader.BaseStream.Position) return false;

\_tmpList.Add(new PosAndLen((memoryStream.Position - 4), (short)(len + 4)));

streamReader.BaseStream.Seek(len, SeekOrigin.Current);

}

Nodes = \_tmpList.ToArray();

memoryStream.Seek(0, SeekOrigin.Begin);

computeStats = new Thread((ThreadStart)delegate

{

stat.Reset();

Program.form?.Invoke((MethodInvoker)delegate { Program.form.ReadStats(); });

stat.UpdateFromStream(memoryStream, Nodes);

Program.form?.Invoke((MethodInvoker)delegate

{

Program.form.B\_UpdateStop.Text = "Обновить";

Program.form.ReadStats();

Program.form.UpdateProgressBar(0);

});

});

computeStats.Start();

return true;

}

/// <summary>

/// Возвращает ветку представляющую один документ.

/// </summary>

/// <param name="startNumberDoc">Начальный индекс выводимых документов</param>

/// <param name="endNumberDoc">Конечный индекс выводимых документов</param>

/// <returns>Ветка, представляющая один документ</returns>

public IEnumerable<TreeNode> GetNodes(UInt32 startNumberDoc, UInt32 endNumberDoc)

{

for (uint i = startNumberDoc; i <= endNumberDoc; i++)

{

memoryStream.Seek(Nodes[i].Position, SeekOrigin.Begin);

byte[] \_tmp = new byte[Nodes[i].Length];

memoryStream.Read(\_tmp, 0, Nodes[i].Length);

yield return CreateNode(new Fw16.Model.TLVWrapper<Fw16.Model.TLVTag>(\_tmp));

}

}

/// <summary>

/// Рекурсивная функция формирующая ветвь

/// </summary>

/// <param name="tLVWrapper">tlVWraper, документа</param>

/// <returns></returns>

private TreeNode CreateNode(Fw16.Model.TLVWrapper<Fw16.Model.TLVTag> tLVWrapper)

{

TreeNode node;

if (tLVWrapper.Source.Tag == Fw16.Model.TLVTag.\_Anonymous)

{

return CreateNode((tLVWrapper.Value as List<Fw16.Model.TLVWrapper<Fw16.Model.TLVTag>>)[0]);

}

if (tLVWrapper.Value is List<Fw16.Model.TLVWrapper<Fw16.Model.TLVTag>> \_tmpList)

{

node = new TreeNode($"[{(int)tLVWrapper.Source.Tag}] {tLVWrapper.Description}");

foreach (var \_tmpWrap in \_tmpList)

{

node.Nodes.Add(CreateNode(\_tmpWrap));

}

}

else

{

string \_tmp = "";

if (tLVWrapper.Value is byte[] val)

{

foreach (var item in val)

{

\_tmp += item.ToString("X2") + " ";

}

}

node = new TreeNode($"[{(int)tLVWrapper.Source.Tag}] {(\_tmp.Length > 0 ? \_tmp : tLVWrapper.Value)} {tLVWrapper.Description}");

}

return node;

}

public void StopComputeStats()

{

if (computeStats?.IsAlive ?? false)

{

computeStats?.Abort();

computeStats?.Join();

Program.form?.Invoke((MethodInvoker)delegate { Program.form.UpdateProgressBar(0); });

}

}

public class PosAndLen

{

Int64 position;

Int16 length;

public long Position { get => position; }

public short Length { get => length; }

public PosAndLen(Int64 position, Int16 length)

{

this.position = position;

this.length = length;

}

}

public class Statistic

{

/// <summary>

/// Элементы статистики

/// </summary>

public enum StatsName

{

err = 0,

incomeCount = 1,

incomeSum,

incomeBackCount,

incomeBackSum,

outcomeCount,

outcomeSum,

outcomeBackCount,

outcomeBackSum,

correctionIncomeCount,

correctionIncomeSum,

correctionOutcomeCount,

correctionOutcomeSum

}

decimal[] stats;

public Statistic()

{

stats = new decimal[Enum.GetValues(typeof(StatsName)).Length];

}

/// <summary>

/// Возвращает значение указанного элемента статистики

/// </summary>

/// <param name="index">Наименование элемента статистики</param>

/// <returns>Decimal, значение элемента статистики</returns>

public decimal this[StatsName index]

{

get

{

return stats[(int)index];

}

set

{

stats[(int)index] = value;

}

}

/// <summary>

/// Создаёт копию переденного потока и вычисляет статистику для документов зашифрованных в потоке.

/// </summary>

/// <param name="stream">Поток данных</param>

/// <param name="nodes">Позиции и длянны в байтах документов в потоке</param>

public void UpdateFromStream(Stream stream, PosAndLen[] nodes)

{

MemoryStream memoryStream = new MemoryStream();

long \_pos = stream.Position;

stream.Seek(0, SeekOrigin.Begin);

stream.CopyTo(memoryStream);

stream.Seek(\_pos, SeekOrigin.Begin);

try

{

for (int i = 0; i < nodes.Length; i++)

{

Program.form?.Invoke((MethodInvoker)delegate { Program.form.UpdateProgressBar(i + 1, nodes.Length); });

Fs.Native.DocumentType docType = Fs.Native.DocumentType.NoDocument;

Fw16.Model.ReceiptKind receiptKind = Fw16.Model.ReceiptKind.NotAvailable;

decimal sum = 0;

memoryStream.Seek(nodes[i].Position, SeekOrigin.Begin);

byte[] \_tmp = new byte[nodes[i].Length];

memoryStream.Read(\_tmp, 0, nodes[i].Length);

foreach (var item in new Fw16.Model.TLVWrapper<Fw16.Model.TLVTag>(\_tmp).Value as List<Fw16.Model.TLVWrapper<Fw16.Model.TLVTag>>)

{

GetStatsDoc(item, ref docType, ref receiptKind, ref sum);

}

switch (docType)

{

case Fs.Native.DocumentType.Receipt:

switch (receiptKind)

{

case Fw16.Model.ReceiptKind.Income:

this[StatsName.incomeCount]++;

this[StatsName.incomeSum] += sum;

break;

case Fw16.Model.ReceiptKind.IncomeBack:

this[StatsName.incomeBackCount]++;

this[StatsName.incomeBackSum] += sum;

break;

case Fw16.Model.ReceiptKind.Outcome:

this[StatsName.outcomeCount]++;

this[StatsName.outcomeSum] += sum;

break;

case Fw16.Model.ReceiptKind.OutcomeBack:

this[StatsName.outcomeBackCount]++;

this[StatsName.outcomeBackSum] += sum;

break;

}

break;

case Fs.Native.DocumentType.ReciptCorrection:

switch (receiptKind)

{

case Fw16.Model.ReceiptKind.Income:

this[StatsName.correctionIncomeCount]++;

this[StatsName.correctionIncomeSum] += sum;

break;

case Fw16.Model.ReceiptKind.Outcome:

this[StatsName.correctionOutcomeCount]++;

this[StatsName.correctionOutcomeSum] += sum;

break;

}

break;

}

}

}

finally

{

memoryStream.Close();

}

}

/// <summary>

/// Рекурсивная функция подсчитывающая статистику в переданном документе

/// </summary>

/// <param name="tLVWrapper">Документ</param>

/// <param name="docType">Тип документа</param>

/// <param name="receiptKind">Тип рассчёта</param>

/// <param name="sum">Сумма</param>

public void GetStatsDoc(Fw16.Model.TLVWrapper<Fw16.Model.TLVTag> tLVWrapper, ref Fs.Native.DocumentType docType, ref Fw16.Model.ReceiptKind receiptKind, ref decimal sum)

{

switch ((int)tLVWrapper.Source.Tag)

{

case 103://чек

docType = Fs.Native.DocumentType.Receipt;

break;

case 131://чек коррекции

docType = Fs.Native.DocumentType.ReciptCorrection;

break;

case 1054://признак рассчёта

receiptKind = (Fw16.Model.ReceiptKind)tLVWrapper.Value;

break;

case 1020://сумма

sum = Decimal.Parse(tLVWrapper.Value.ToString());

break;

default:

break;

}

if (tLVWrapper.Value is List<Fw16.Model.TLVWrapper<Fw16.Model.TLVTag>> list)

{

foreach (var item in list)

{

GetStatsDoc(item, ref docType, ref receiptKind, ref sum);

}

}

}

/// <summary>

/// Сбрасывает всю статистику

/// </summary>

public void Reset()

{

for (int i = 0; i < stats.Length; i++)

{

stats[i] = 0;

}

}

}

}

}}