Семестровая работа по курсу «Документоведение».

Василов Артур,

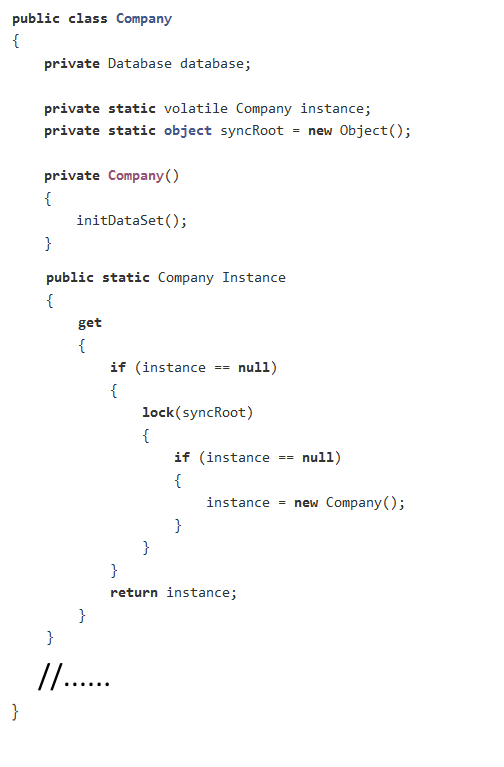
Группа 09-311,

2014г.

Разработанное приложение имитирует систему документооборота аутсорсинговой компании. Для примера используются 4 документа – договор о найме сотрудника, проектный договор, отчет сотрудника и финансовая декларация (в простой форме – доход-расход-прибыль).

Рассмотрим, как была реализована работа с системой хранения документов. Примечание: код будет приводиться фрагментами, полная версия доступна в гит-репозиторие. [Ссылка.](https://github.com/ArturVasilov/PracticeITcs/tree/master/DocumentsSecurity/DocumentsSecurity)

Во-первых, для удобной работы нам надо создать класс, для обработки всех запросов, операций поиска, удаления и добавления документов. Этот класс должен иметь только один экземпляр и содержать все необходимые методы. Таким образом, этот класс должен реализовывать паттерн Singleton, а также частично являться DAO (data access object). Я использую ленивую инициализацию и механизм двойной блокировки. Также я добавил несколько методов, которые можно выполнять из других классов (я опускаю комментарии в коде, так как в комментировании нуждается только плохой код):

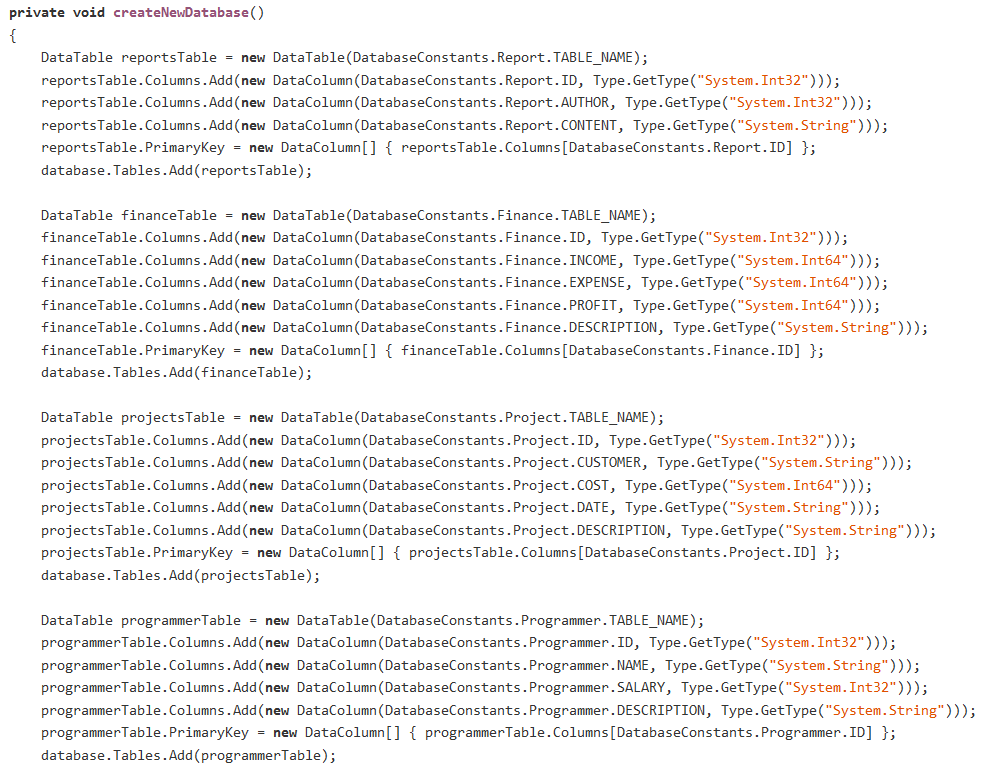


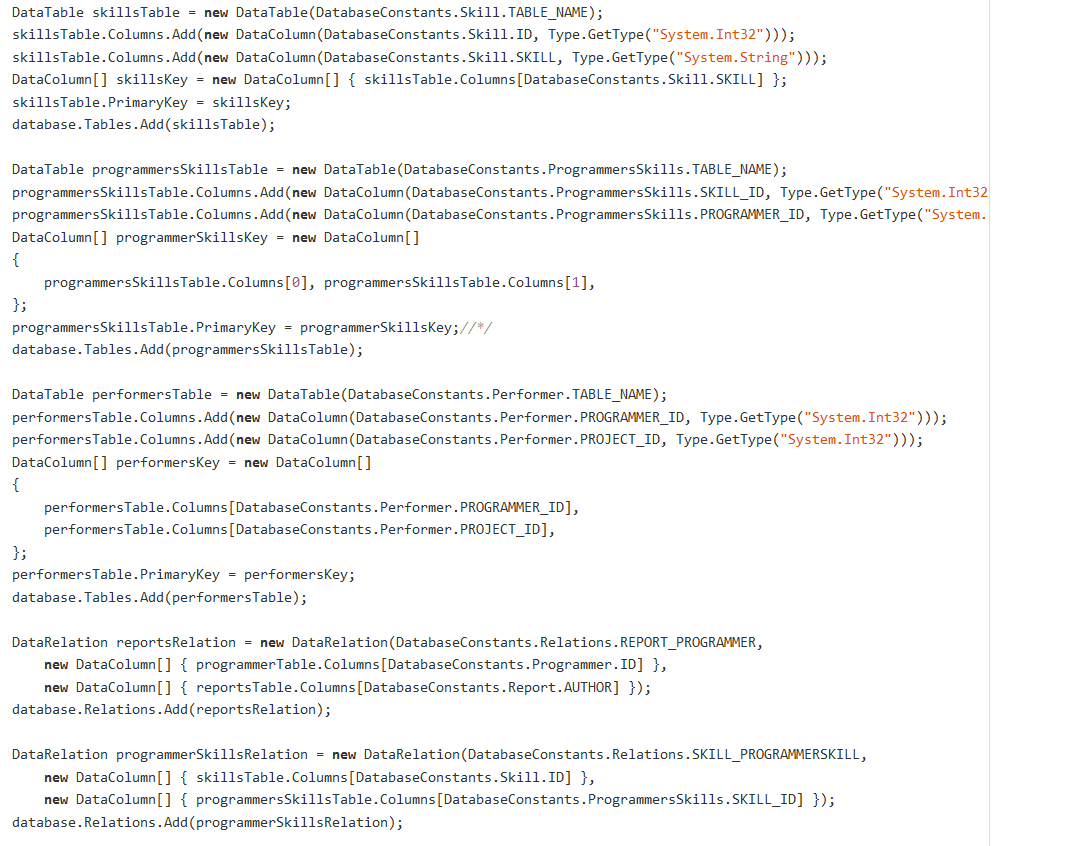
Как видно, все методы вызывают соответствующие методы в классе Database. Этот класс уже непосредственно выполняет работу с классом DataSet.

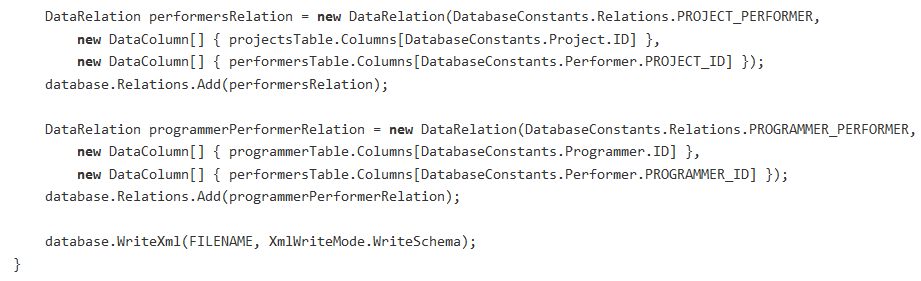
Класс DataSet (основные фрагменты кода):



Метод создания базы, создает все таблицы и связи:

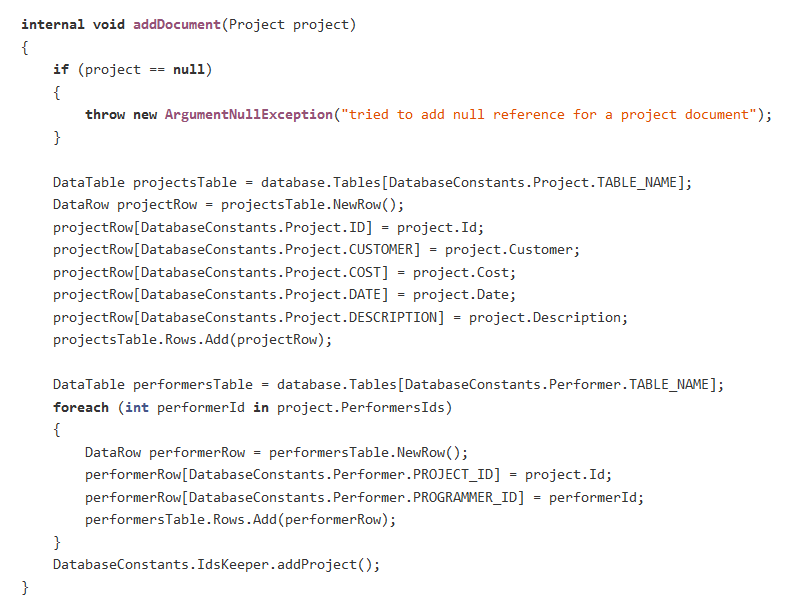




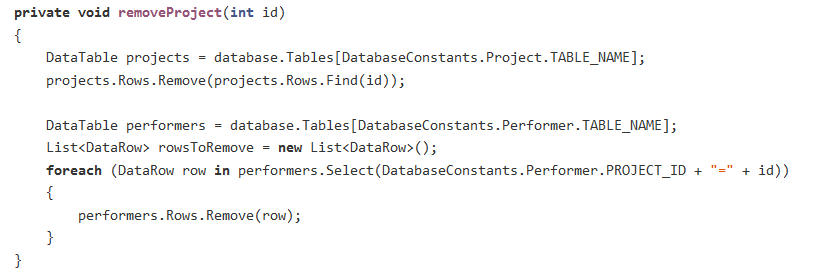


Рассмотрим методы добавления / удаления, редактирования, получения на примере проектного договора.

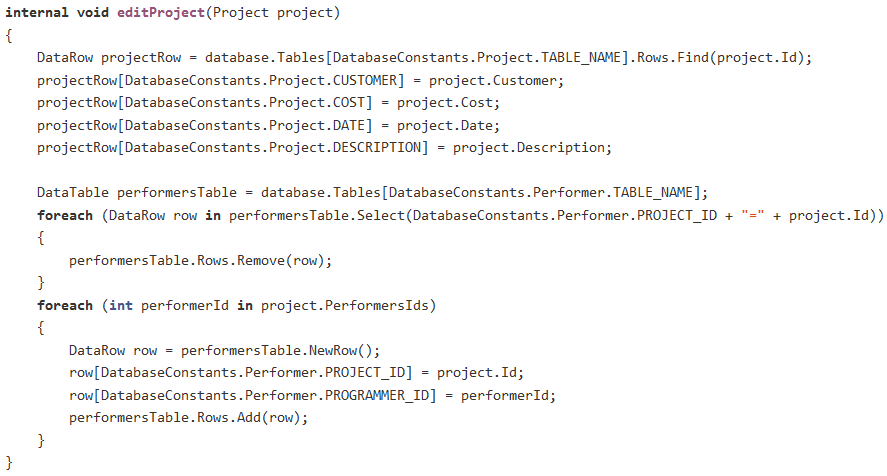
Добавление:



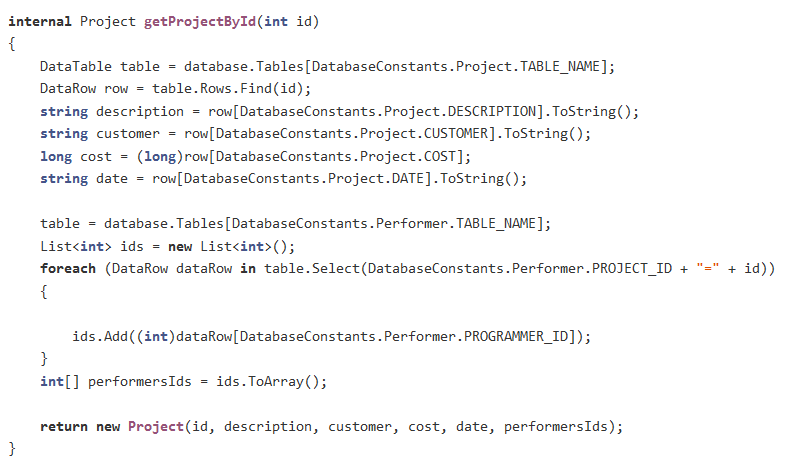
Удаление:



Редактирование:

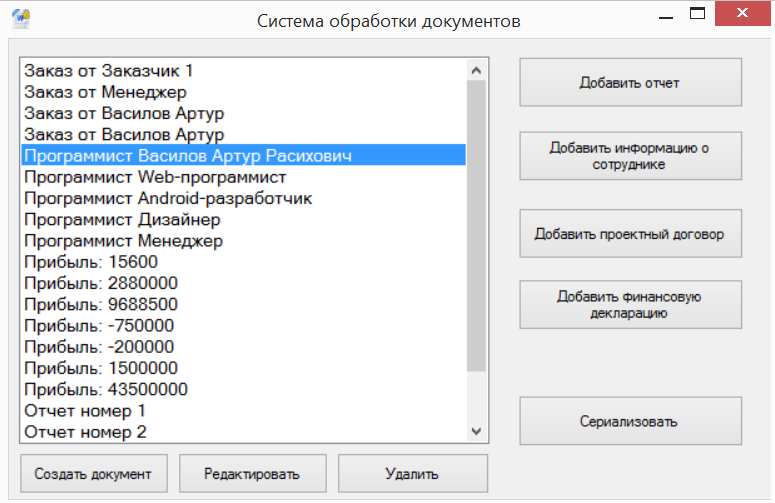


Получение:

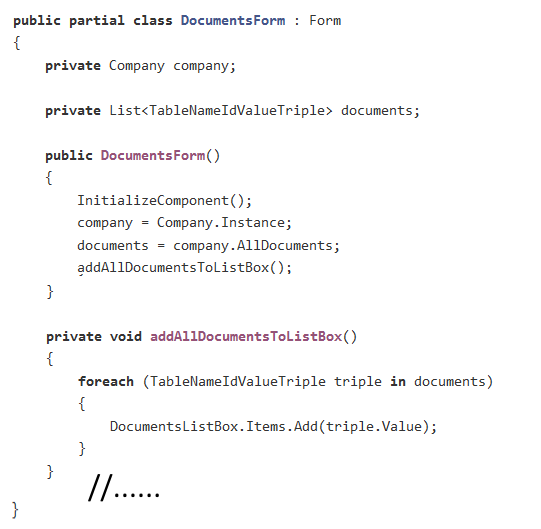
1

Для каждого документа создается отдельный класс, который наследуется от базового класса Document, который содержит общие поля и описание документа (т.е. все те поля, который должны содержаться во всех документах). Эти классы достаточно примитивны и просто содержат все нужные поля, рассматривать их не имеет особого смысла.

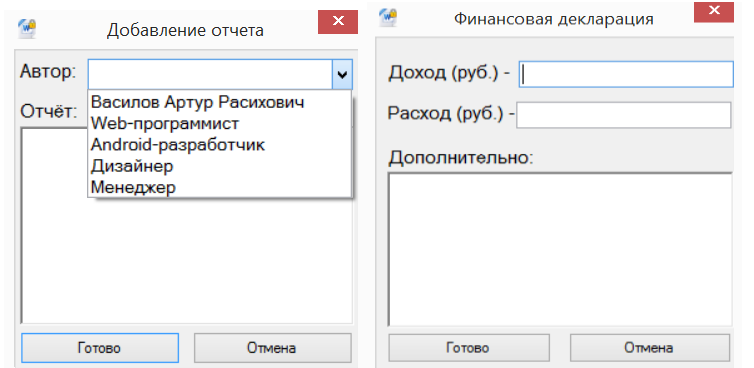
Приложение должно быть удобно в использовании. Для этого средствами Windows Forms был создан следующий интерфейс пользователя:



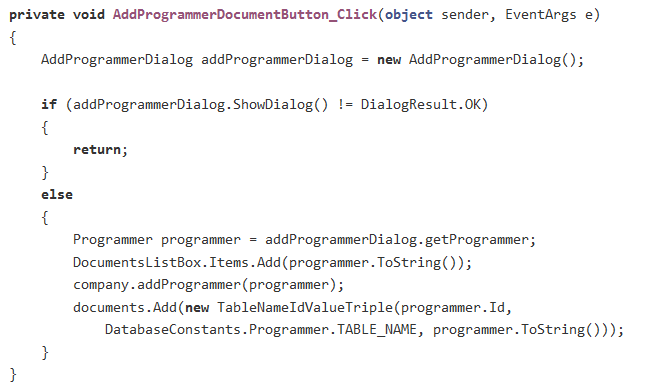
При запуске приложения все документы считываются из базы и добавляются в компонент ListBox на форме. Для этого можно выполнить примерный код:



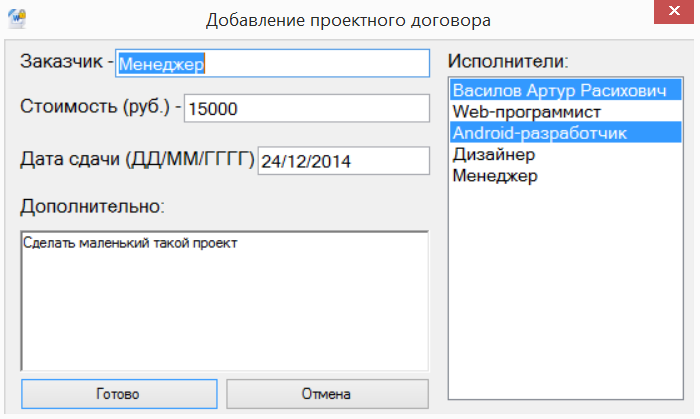
Создание документов выполняется с помощью соответствующих кнопок. По нажатию на них вызываются соответствующие диалоговые окна:



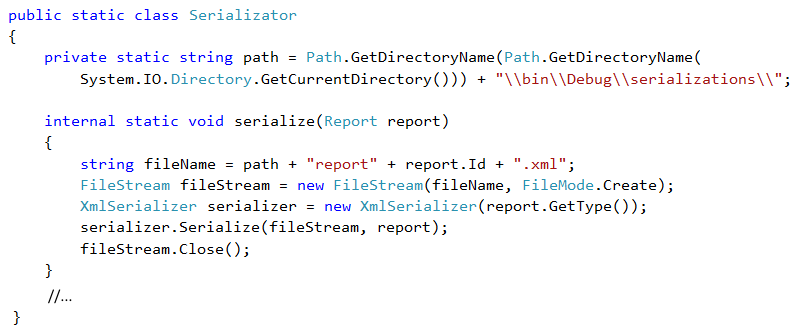
Код обработки нажатия на кнопку:



Сам класс AddProgrammerDialog считывает значение нужных полей по завершению ввода и генерирует соответствующий документ. Для редактирования используется этот же диалог с уже заполненными полями:

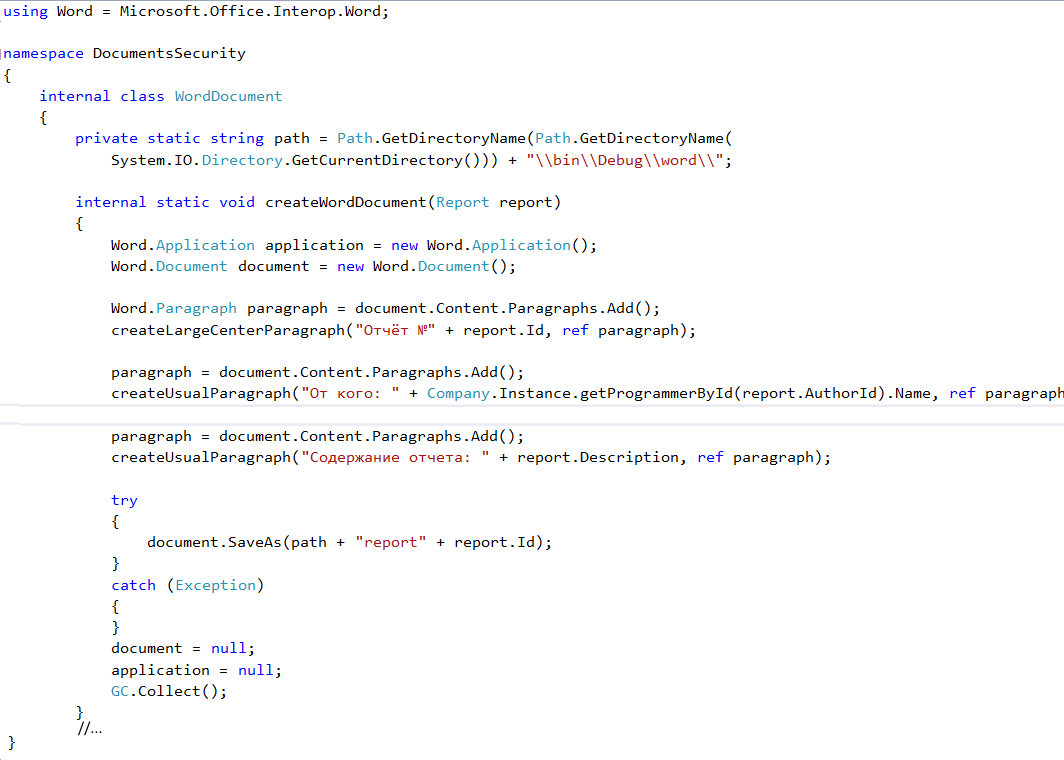


Пользователь может по желанию сериализовать документ для выполнения различных операций, например, отправка куда-либо и прочее. Для сериализации создадим отдельный класс с набором статических методов для каждого документа:

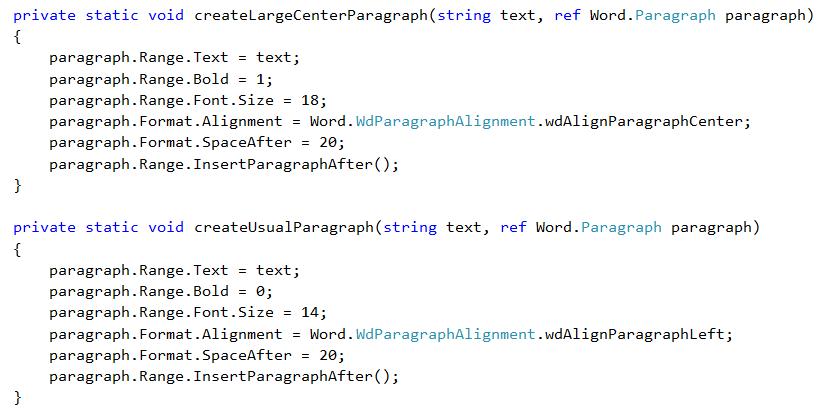


Сериализация остальных документов выполняется аналогично.

Кроме этого, мы должны иметь возможность генерации обычных документов для Microsoft Word. Это можно реализовать с помощью библиотеки Microsoft.Office; Пример:

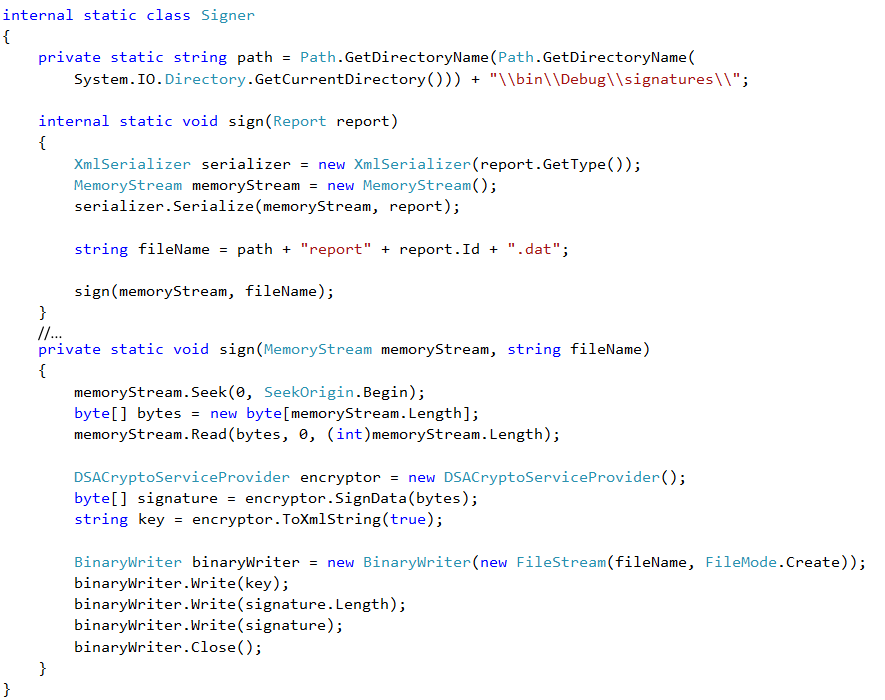


Для создания отдельных параграфов используются дополнительные методы:

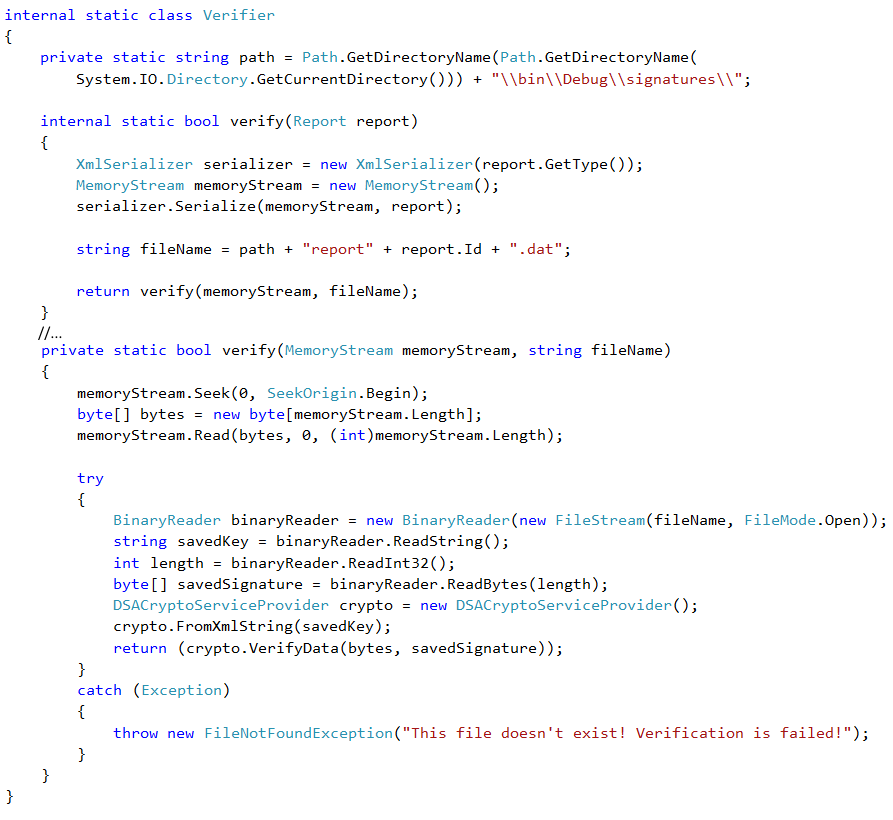


Для обеспечения мониторинга целостность документов, после создания или редактирования каждого документа необходимо генерировать его цифровую подпись. Если доступ к данным будет получен не из программы (например, кто-то напрямую отредактирует xml-файл), мы всегда сможем это узнать.

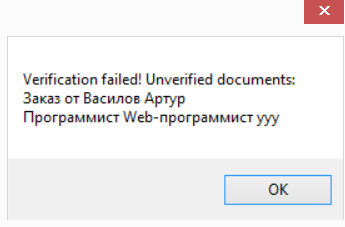
Для генерации цифровой подписи воспользуемся следующим классом:



Создадим еще один класс для проверки целостности документа. Он будет возвращать булевское значение, в зависимости от того, была ли нарушена целостность документа или нет:



Теперь при запуске программы мы выполняем верификацию всех документов и выводим список тех документов, которые ее не прошли:



Что позволяет нам быстро локализовать и исправить ошибку.

Таким образом, данная программа является достаточно неплохим средством для обработки документов определенной компании. Более того, ее можно адаптировать к любой системе документов. При этом необходимо будет переписать классы, которые работают непосредственно с документами, но не требуется продумывать новую архитектуру, что также достаточно удобно.

Данная программа создает также базовые основы для защиты документов. Конечно, это можно улучшать, храня всю информацию в зашифрованном виде.

Интерфейс пользователя также достаточно удобен в использовании (разумеется, он также нуждается в адаптации и улучшении при доработке программы).

Также крайне желательно использовать многопоточную обработку документов, так как даже задача извлечения данных из xml-файла и их верификация будет занимать непозволительно большое время при определенном количестве документов для того, чтобы выполняться в главном потоке.

Но все эти улучшения лежат за рамками курса, поэтому их возможная будущая реализация также не должна содержаться в данном отчете.