# Техническое задание

# Многопоточный обработчик файлов с расширяемой функциональностью

## Платформа .NET Framework 4.7.2 C#

## Среда Windows 10 или Server 2012 и выше

# Заказчик: Никита Гуров

# Исполнитель: Андрей Куликов

# Май 2020

1. Название

Название проекта: «Многопоточный обработчик файлов с расширяемой функциональностью».

1. Платформа

Выполнить на платформе .NET Framework версии не ниже 4.7.2 на языке C#.

1. Постановка задачи

Разработать библиотеку классов, которая предоставляет сервис для многопоточного поиска файлов в различных хранилищах, так локальных, так и сетевых, и последующей обработки данных из файлов с помощью внешних по отношению к библиотеке модулей. В библиотеке должны быть предусмотрены сервисы для создания поисковых сессий с заданными шаблонами поиска и организации потоков обработки данных.

В библиотеке должна быть предусмотрена возможность дальнейшего расширения функциональности при помощи дополнительных модулей, при этом библиотека должна предусматривать сервисы для подключения дополнительных модулей и включения в систему обработки данных.

Библиотека должна быть реализована в виде DLL на платформе .NET Framework версии не ниже 4.7.2 с поддержкой длинных имён файлов. В состав библиотеки также должен входить набор unit-тестов.

1. требования к функциям

Библиотека должна предоставлять сервисы для выполнения следующих функций:

1. Параллельное (асинхронное) выполнение операций над файлами по следующим правилам:
   1. Последовательность операций (вызова плагинов) задаётся пользователем;
   2. Cами операции определяются во внешних модулях (плагинах) к основному приложению, т.е. необходимо разработать интерфейс (API) для подключаемых модулей с возможностью сохранения данных в контексте приложения (разработка самих плагинов не требуется, за исключением поиска и копирования файлов).
2. Поиск и сбор информации (имя, атрибуты и основные свойства) о файлах, хранящихся на физических и сетевых носителях (с поддержкой длинных путей):
   1. Искать по списку папок (с исключениями);
   2. Поиск параллельный – найденные файлы незамедлительно поступают в очередь для последующей обработки.
3. Копирование в локальное хранилище:
   1. Копировать по маске, заданной для конкретной папки;
   2. Обеспечить синхронизацию, т.е. копировать только новые и изменённые файлы, старые и удаленные переносить в архив;
   3. Возможность добавлять файлы для копирования в процессе обработки.
4. Возможность использования разных очередей с ограничением параллелизма для различных задач:
   1. Преимущественно вычислительные операции;
   2. Интенсивное использование устройств ввода-вывода.
5. Должно быть предусмотрено оповещение о прогрессе выполнения с помощью событий.
6. Должна быть предусмотрена возможность приостановить или прервать процесс.
7. Должна быть предусмотрена возможность обработки ошибок с сохранением в контекст и лог-файл.
8. Должно быть предусмотрено сохранение текущего статуса выполнения задач с возможностью возобновления работы после прерывания процесса.
9. Хранение сохранённых данных относительно текущего статуса, конфигурации и т.п. в базе данных не требуется. Достаточно обойтись простым сохранением одного или группы файлов в локальном хранилище. Формат файлов выбирает разработчик.
10. требования к исполнению

Сроки выполнения: 1 месяц.

Стоимость: договорная.

Сроки тестирования и устранения ошибок: 2 недели после выполнения проекта, при этом:

* Заказчик обязуется провести приёмку в течение 4 рабочих дней после окончания разработки;
* Подрядчик обязуется устранить ошибки в течение последующих 4 дней в случае выявления ошибок.

В целях контроля над выполнением проекта предусматривается разбиение на подзадачи:

1. Проработка и утверждение концепции (20%),
2. Разработка параллельного обработчика (20%),
3. Разработка поиска и хранения файлов (20%),
4. Разработка API (20%),
5. Разработка тестов (20%).

Дополнительные требования:

* Проект вести в репозитории GitHub с регулярными коммитами,
* Обсуждение прогресса и деталей выполнения проводить не реже 1-го раза в неделю.

1. Описание концепции
   1. Наименование проекта

Наименование проекта и полное наименование проектируемой библиотеки – Файловый процессор (FileProcessor), наименование файла библиотеки после компиляции и сборки – fp.dll.

* 1. Термины и определения

В настоящей концепции применены следующие термины и определения:

1. **Приложение** – вызывающее приложение, загрузившее и использующее функции библиотеки.
2. **Сервис или Служба (Service)** – программный объект, изменяющий другие программные объекты, их состояние, связи, конфигурацию и пр. Является экземпляром класса .NET, содержит данные и методы.
3. **Объект данных или Объект (Data Object, Object)** – программный объект, основным назначением которого является хранить представляемый им набор данных. Может изменять только собственные конфигурацию и состояние. Является экземпляром класса .NET, содержит данные и методы.
4. **Программный пользовательский интерфейс (User API)** – совокупный набор методов и полей библиотеки, доступный внешнему приложению, использующему библиотеку.
5. **Системный программный интерфейс (.NET Framework API)** – программный интерфейс платформы .NET Framework.
6. **Локальное хранилище (Local FileStore)** – локальная директория компьютера, на котором развёрнута сессия приложения и в которое приложение копирует файлы из исходных хранилищ для дальнейшей обработки.
7. **Исходное хранилище (Original FileStore)** – сетевая или локальная директория по отношению к компьютеру, на котором развёрнута сессия приложения, в которой хранятся исходные экземпляры файлов, предназначенных для обработки.



Рис. 1. Условные обозначения

1. Чтобы отличать имена классов библиотеки от имён объектов, имя класса должно начинаться с буквы N. Таким образом, NFrame – имя класса сервиса Фрейм, Frame – имя экземпляра класса NFrame. На схемах наименования Frame, Conveyor и т.п. обозначают программные объекты приложения соответствующих классов.
2. Условные обозначения, применённые на схемах, приведены на рис. 1.
   1. Иерархия сервисов

На рис. 1 представлена иерархия программных объектов служб, предоставляемых библиотекой.



Рис. 2. Иерархия сервисов библиотеки FileProcessor

Иерархическая структура сервисов проекта FileProcessor состоит из 3-х уровней:

1. **Фрейм** (**Frame**, **класс** **NFrame**) – программный объект, который отвечает за действия, общие для всего приложения:
   1. Разворачивает в памяти программные объекты в соответствии с текущей конфигурацией;
   2. Обеспечивает загрузку и сохранение текущей конфигурации;
   3. Предоставляет методы для изменения текущей конфигурации;
   4. Предоставляет доступ к конвейерам;
   5. Обеспечивает ведение лога с сохранением в файл или выводом в пользовательское устройство вывода.

Более подробно структура и логика работы сервиса Frame описана в разделе 6.4.

1. **Конвейер** (**Conveyor**, **класс** **NConveyor**) – программный объект, реализующий один логический поток операций обработки, начиная от поиска файлов и далее по этапам. Конвейеров в приложении может быть более одного, конвейеры работают параллельно и независимо друг от друга. Зависание или крушение одного конвейера не должно приводить к зависанию или крушению приложения.

Более подробно структура и логика работы сервиса Conveyor описана в разделе 6.5.

1. **Обработчик (Processor, класс NProcessor)** – программный объект, реализующий определённый алгоритм обработки файлов. Примерами обработчиков являются: копировщик файлов, подсчёт контрольной суммы файла, заполнение пустых полей в таблице, удаление дублированных строк в таблице и т.п.

Более подробно структура и логика работы сервиса Processor описана в разделе 6.6.

* 1. Сервис Frame (класс NFrame)

Структурная схема программного объекта службы Frame представлена на рис. 2.



Рис. 3. Структурная схема сервиса Frame

Сервис Frame предназначен для разворачивания конвейеров в соответствии с конфигурацией, добавления и удаления конвейеров, загрузки и чтения конфигурации, ведения общего лога.

Фрейм обеспечивает так же ведение реестра обработчиков, которые могут быть использованы конвейерами.

В составе фрейма существуют следующие программные объекты:

1. **Конвейер** (**Conveyor**, **класс** **NConveyor**) – реализует один поток обработки файлов. Является главным программным объектом в составе фрейма. Фрейм обеспечивает существование и одновременную работу нескольких конвейеров.

Каждому конвейеру назначается свой приоритет из набора системных приоритетов задач .NET.

1. **Реестр обработчиков (ProcessorRegistry, класс NProcessorRegistry)** – представляет собой словарь, в котором хранится информация обо всех известных библиотеке обработчиках. Фрейм предоставляет методы для добавления и удаления обработчиков из реестра. Один или несколько обработчиков должны быть упакованы в библиотеку .NET dll. Фрейм умеет открывать библиотеку, распознавать обработчики и включать их в реестр, формируя уникальную запись для каждого обработчика. Запись в реестре должна содержать следующую информацию: имя обработчика, под которым он известен в системе, полное наименование класса обработчика, ссылка на файл библиотеки, в которой хранится код обработчика. Запись в реестре является объектом ProcessorContainer, класс NProcessorContainer.

Чтобы службы Файлового процессора могли правильно работать с обработчиками, каждый обработчик должен быть наследником базового класса NProcessor.

1. **Конфигурация фрейма (FrameConfig)** – набор полей фрейма, в которых хранится информация о текущей конфигурации фрейма – количество конвейеров, состояние каждого конвейера, ссылки на файлы конфигурации и реестра обработчиков и т.п.
2. **Менеджер фрейма (FrameManager)** – набор методов фрейма, которые обеспечивают следующие функции:
   1. Конструирование объекта фрейма в памяти при запуске приложения.
   2. Разрушение объекта фрейма и освобождение памяти при завершении приложения.
   3. Загрузку конфигурации фрейма и реестра обработчиков по команде User API и при запуске приложения.
   4. Сохранение конфигурации фрейма и реестра обработчиков по команде User API и при завершении приложения.
   5. Разворачивание набора конвейеров в памяти соответственно конфигурации при запуске приложения.
   6. Обеспечение работы каждого конвейера в отдельном потоке в соответствии с заданным приоритетом.
   7. Запуск, остановка, пауза и возобновление работы каждого конвейера по командам User API и при запуске приложения в соответствии с текущей конфигурацией.
   8. Оповещение конвейеров о прекращении работы приложения, чтобы последние смогли сохранить свои конфигурации.
   9. Обработка исключений, возникающих в конвейерах.
3. **Логгер фрейма (FrameLogger класс NFrameLogger)** – сервис – обработчик сообщений лога, поступающих из конвейеров и методов фрейма. Каждое сообщение является объектом LogMessage, класс NLogMessage. Логгер фрейма сохраняет сообщения в файл лога в хронологическом порядке. Логгер также предоставляет возможности для подключения внешних потребителей и получения сообщений лога программным объектам приложения, внешним по отношению к Файловому процессору, например, окну лога.
   1. Сервис Conveyor (класс NConveyor)

Структурная схема сервиса представлена на рис. 4.

Сервис Конвейер является самым сложным и насыщенным элементом библиотеки. Рассмотрим его структуру и логику работы подробнее.

Основная задача конвейера состоит в том, чтобы обеспечить разворачивание в памяти приложения набора обработчиков согласно текущей конфигурации, запустить их в заданной последовательности и обеспечить, чтобы выходы и входы последовательных обработчиков были соединены между собой. Обработчики 1 и 2 являются последовательными, если обработчик 2 в качестве входных данных использует результат работы обработчика 1 и не может приступить к обработке файла, пока обработчик 1 не закончит работать с ним. Каждый обработчик может быть запущен в несколько параллельных потоков, количество потоков определяется конфигурацией конвейера. Все потоки процессоров имеют тот же приоритет, что и поток контейнера.



Рис. 4. Структурная схема сервиса Conveyor

Конвейер может находиться в одном из следующих состояний:

* Остановлен – все процессоры, включая поисковик, остановлены;
* Работает – выполняются потоки поисковика и процессоров в зависимости от стадии обработки; по мере опустошения очередей некоторые потоки могут останавливаться; конвейер считается находящимся в состоянии «работает», пока работает хотя бы один поток конвейера;
* Пауза – конвейер был запущен и все процессоры приостановлены, текущее состояние сохраняется.

В состав контейнера входят следующие компоненты:

1. **Файловый поисковик (FileSearch класс NFileSearch)** – обязательный компонент конвейера, который всегда является первым в цепочке обработчиков. Поисковик получает на входе запрос поиска на языке стандартного поиска менеджера файлов Windows в виде текстовой строки. При запуске конвейера поисковик осуществляет поиск файлов согласно запроса в указанных хранилищах. Для каждого найденного файла создаётся объект FileContainer класса NFileContainer, в котором сохраняется ссылка на найденный исходный файл. Объекты FileContainer выдаются порциями по мере прогресса поиска и помещаются в реестр файловых контейнеров и в очередь на выходе поисковика.
2. **Очередь файловых контейнеров (Queue)** – объект данных, который обеспечивает приём, хранение и выдачу файловых контейнеров по принципу FIFO. Очередь обслуживает два включенных последовательно процессора, являясь выходной для первого (выдающего) и входной для второго (принимающего). Очередь содержит семафор, регулирующий постановку в ожидание и запуск потоков принимающего процессора по мере наполнения и опустошения очереди. Принимающий процессор запускается, как только в очереди появляется хотя бы один контейнер и переводится в ожидание, когда очередь становится пуста. Конвейер переводится в состояние «остановлен» автоматически, когда все очереди конвейера становятся пусты.

Выходная очередь поисковика является входной очередью последовательного за ним процессора. Поток (или потоки) последовательного за поисковиком процессора находится в ожидании и запускается, как только во входной очереди появляется хоты бы один файловый контейнер.

1. **Файловый контейнер (FileContainer класс NFileContainer)** – объект данных, содержащий информацию об одном файле, накопленную в ходе одной сессии работы конвейера. Файловый контейнер содержит следующие объекты:

* Идентификатор файлового контейнера, уникальный в пределах текущей сессии конвейера,
* Ссылка на исходный файл – результат поиска в исходном хранилище,
* Ссылка на результирующий файл – файл с результатом работы последнего полностью отработавшего на данный момент времени обработчика,
* Лог операций над файлом – список всех операций, проделанных с файлом во время текущей сессии конвейера с момента старта на текущий момент.

1. **Элемент лога операций над файлом (ContainerMessage класс** **NContainerMessage)** – базовый класс для всех элементов лога операций. Обработчики могут определять свои классы элементов лога, размещая там специфичную для себя информацию. В то же время все эелементы лога должны быть наследниками класса NContainerMessage, чтобы файловый контейнер мог обрабатывать их единым образом. Ошибки и исключения также пишутся в лог.

В состав данных базового элемента лога входят:

* Идентификатор элемента лога, уникальный в пределах текущей сессии конвейера,
* Время совершения операции,
* Идентификатор процессора, сделавшего запись,
* Тип операции,
* Ссылка на пользовательский объект.

1. **Реестр файловых контейнеров (ContainerRegistry класс NContainerRegistry)** – программный объект типа словарь, который содержит перечень находящихся в работе на данный момент файловых контейнеров и является общим для всех обработчиков конвейера. Наполнение реестра контейнеров обеспечивает поисковик, создавая и добавляя в реестр контейнер для каждого помещенного в свою выходную очередь файла. Чтобы не было коллизий, контейнер сначала помещается в реестр, а затем помещается в выходную очередь.
2. **Обработчик файла (процессор, Processor класс** **NProcessor)** – обработчик файла, который производит операции над файлом по определённому алгоритму. Обработчик может запускаться в несколько одновременных потоков, количество которых определяется конфигурацией конвейера. Все обработчики должны быть наследниками базового класса NProcessor. Конвейер обеспечивает запуск обработчиков в соответствующие моменты согласно текущей конфигурации. На схеме рис. 4 показан в качестве примера обработчик копирования файлов из исходного хранилища в локальное под названием FileCopy.

Обработчик может находиться в одном из следующих состояний:

* Остановлен – все потоки обработчика прекращены;
* Работает – выполняются все потоки обработчика; при опустошении входной очереди обработчик переходит в режим ожидания очереди;
* Ожидание очереди – все потоки обработчика ожидают, когда во входной очереди появится хотя бы один элемент; получивший входной элемент поток процессора переходит в режим «работает»; обработчик считается ожидающим, если все потоки процессора ожидают очередь; обработчик считается работающим, если выполняется хотя бы один поток обработчика;
* Пауза – процессор был запущен и все потоки приостановлены, текущее состояние сохраняется.