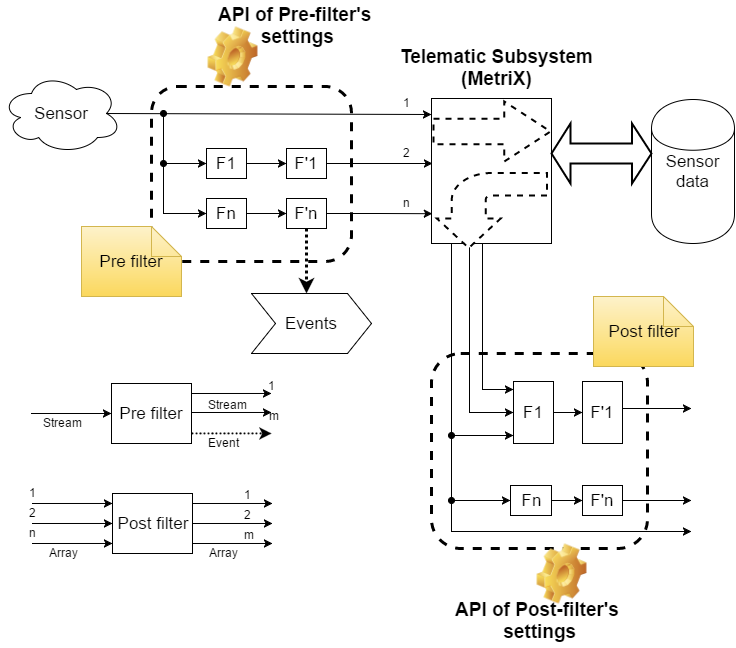
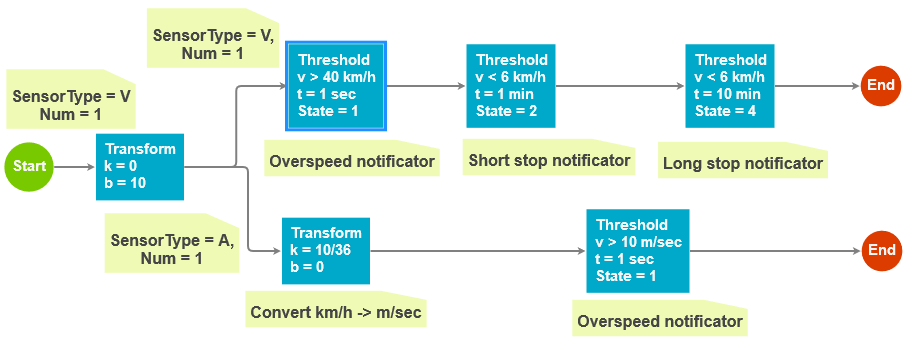
Фильтры обработки данных

TotalAPI





# Виды фильтров

## По результату и способу обработки потока данных

1. **Pre-фильтр** – изменяет входящий поток данных до записи в БД.
2. **Online-фильтр** – не меняет входящий поток, но выполняет дополнительные действия на его основании. (Это может быть генерация специализированных событий или изменение состояния устройства/сенсора).
3. **Post-фильтр** – обрабатывает массив данных, полученный при чтении из БД при запросе пользователем.

Можно заметить, что по большому счёту **Pre-фильтр** и **Online-фильтр** мало чем отличаются друг от друга. Оба работают с входящим потоком данных до их записи в БД и вообще теоретически один фильтр может выполнять обе функции (например, обрезка некорректных значений и уведомление об этом).

Поэтому можно включить **Online-фильтр** в **Pre-фильтр**:

1. **Pre-фильтр** – может изменять входящий ***поток данных*** и выполнять дополнительные действия на его основании. (Это может быть генерация специализированных событий или изменение состояния устройства/сенсора) до записи в БД.
2. **Post-фильтр** – обрабатывает ***массив данных***, полученный при чтении из БД при запросе пользователем.

Фильтры обоих типов могут быть объединены в один конвейер, когда выходные данные одного фильтра поступают на вход другого фильтра.

## По способу формирования конвейера

1. **Линейный** – один вход один выход. Поток данных одного типа датчика остаётся потоком данных этого же типа датчика. При этом данные этого потока могут быть изменены.   
   Очевидно, что **Post-фильтры** могут быть только линейными.  
   **Пример:** Фильтр прореживания (оставляет каждое N-ое значение).
2. **Разветвитель** – один вход несколько выходов. Поток данных одного типа датчика расширяется на несколько потоков датчиков других типов (возможно в том числе и датчика исходного типа). Если выходные потоки данных не будут меняться (кроме информации о типе датчика), то, по сути, это даже не фильтр, а правило построения конвейера. Разбивая один поток на несколько, мы будем применять к каждому из них разные линейные фильтры и в результате получать разные результаты для разных видов датчиков.  
   **Пример:** Данные реального температурного датчика, разбиваются на два потока – поток температурного датчика и поток датчика средней температуры за период (это виртуальный датчик). Применяя ко второму потоку **линейный Pre-фильтр** медианы за N минут, мы сможем записать в БД дополнительный поток данных и получать в последствии значения о средней температуре без пост-вычислений.
3. **Агрегатор** – несколько входов один выход. Несколько потоков разных типов датчиков объединяются в один поток виртуального датчика другого типа.  
   **Пример:** Значения нескольких температурных датчиков объединяются в один виртуальный датчик средней температуры.

Для упрощения можно считать **разветвитель** не фильтром, а правилом конвейера (см. примечание к п.2), а **линейный** фильтр – частным случаем **агрегатора**, когда агрегируется только один поток.

## Интерфейсы и регистрация фильтра в системе

/// <summary>

/// Base data filter interface.

/// </summary>

public interface IDataProcessingFilter

{

/// <summary>

/// Filter identifier.

/// </summary>

string Id { get; }

/// <summary>

/// Filter name.

/// </summary>

string Name { get; }

/// <summary>

/// Filter description.

/// </summary>

string Description { get; }

/// <summary>

/// Type, distributing the parameters class of current filter.

/// This type should have public constructor without parameters and be serializeable.

/// Even if there is no parameters - it should return the type of an empty class.

/// Types of parameters should be unique across different filters.

/// </summary>

[NotNull]

Type FilterParametersType { get; }

}

/// <summary>

/// Interface representing the series of data points

/// </summary>

public interface ISeriesDataPoints : IEnumerable

{

/// <summary>

/// Custom user data. Do not transfered over the network.

/// </summary>

object UserData { get; set; }

/// <summary>

/// The series of data points.

/// </summary>

IList Series { get; }

/// <summary>

/// Additional information - could be expanded by filters during processing.

/// </summary>

IDictionary<string, object> AdditionalInfo { get; }

}

Несмотря на то, что данные координат можно трактовать как связанный между собой набор датчиков, удобнее оперировать данными координат как единым целым. В этом случае система должна иметь отдельный набор фильтров для обработки координат и отдельный набор фильтров для обработки данных датчиков.

Любой TotalApi-фильтр **координат** должен реализовывать один (или оба) из интерфейсов: ICoordinatesProcessingPreFilter и ICoordinatesProcessingPostFilter, которые являются наследниками от IDataProcessingFilter:

/// <summary>

/// Coordinates processing pre-filter interface.

/// </summary>

public interface ICoordinatesProcessingPreFilter : IDataProcessingFilter

{

/// <summary>

/// Gets the flag that filter is notifier and doesn’t continue pipeline stream.

/// The filter returning true never calls OnPreProcessed action.

/// </summary>

bool IsNotifier { get; }

/// <summary>

/// Starts processing of passing series of data.

/// </summary>

/// <param name="series">Series of data with metadata</param>

/// <param name="prms">Filter parameters</param>

void PreProcess(ISeriesDataPoints series, object prms);

/// <summary>

/// Calls when processing is finished and result data can be sent onward.

/// </summary>

Action<IDataProcessingFilter, ISeriesDataPoints> OnPreProcessed { get; }

}

/// <summary>

/// Coordinates processing post-filter interface.

/// </summary>

public interface ICoordinatesProcessingPostFilter : IDataProcessingFilter

{

/// <summary>

/// Starts processing of passing series of data.

/// </summary>

/// <param name="series">Series of data with metadata</param>

/// <param name="prms">Filter parameters</param>

void PostProcess(ISeriesDataPoints series, object prms);

}

Любой TotalApi-фильтр **датчиков** должен реализовывать один (или оба) из интерфейсов: ISensorDataProcessingPreFilter и ISensorDataProcessingPostFilter, которые являются наследниками от IDataProcessingFilter:

/// <summary>

/// Sensor data processing pre-filter interface.

/// </summary>

public interface ISensorDataProcessingPreFilter : IDataProcessingFilter

{

/// <summary>

/// Gets the flag that filter is notifier and doesn’t continue pipeline stream.

/// The filter returning true never calls OnPreProcessed action.

/// </summary>

bool IsNotifier { get; }

/// <summary>

/// Starts processing of passing series of data.

/// </summary>

/// <param name="series">Series of data with metadata</param>

/// <param name="prms">Filter parameters</param>

void PreProcess(ISeriesDataPoints series, object prms);

/// <summary>

/// Calls when processing is finished and result data can be sent onward.

/// </summary>

Action<IDataProcessingFilter, ISeriesDataPoints> OnPreProcessed { get; }

}

/// <summary>

/// Sensor data processing post-filter interface.

/// </summary>

public interface ISensorDataProcessingPostFilter : IDataProcessingFilter

{

/// <summary>

/// Starts processing of passing series of data.

/// </summary>

/// <param name="series">Series of data with metadata</param>

/// <param name="prms">Filter parameters</param>

void PostProcess(ISeriesDataPoints series, object prms);

}

Информация о том, в каком порядке фильтры должны быть объединены в конвейер, с какими параметрами и какая последовательность должна применяться к какому из типов датчиков, задаётся пользователем и хранится в БД телематического модуля.

Фактически пользователь определяет свои наборы фильтров, задавая их последовательность и параметры, а потом указывает к какому типу датчика какой из созданных наборов применить.

Также есть возможность трактовать созданный набор как новый фильтр. Т.е. использовать его в качестве фильтра при создании другого набора. Таким образом можно создавать новые фильтры на основании существующих.

В случае **Post-фильтров**, пользователь может указать в качестве параметра вызова набор кастомный фильтров, которые должны быть применены для постобработки запрошенных данных, либо указать идентификатор одного из определённых ранее наборов, либо не использовать **Post-фильтры** вообще.

Прогонку данных через конвейер фильтров осуществляет **менеджер фильтров**. Его можно представить, как один фильтр, который получает на вход данные, а на выходе записывает их в БД (для **Pre-фильтров**) или возвращает обработанные данные (для **Post-фильтров**).

В свою очередь разработчик фильтров об этом знать не должен – каждый из фильтров должен работать, считая, что других фильтров в системе нет.

**Менеджер фильтров** осуществляет автоматическую регистрацию фильтров в системе. Для того, чтобы фильтр был автоматически зарегистрирован в системе необходимо, чтобы в сборке, содержащей **менеджер фильтров**, были реализации соответствующих интерфейсов (ICoordinatesProcessingPreFilter, ICoordinatesProcessingPostFilter, ISensorDataProcessingPreFilter, ISensorDataProcessingPostFilter), экспортированные средствами **MEF**.

Интерфейс телематического модуля соответственно расширяется методами, позволяющими конфигурировать фильтры.

## Масштабирование фильтров

В простейшем случае для масштабирования мы можем использовать несколько телематических модулей, каждый из которых в один отрезок времени принимает данные разных датчиков, обрабатывают их, используя одинаковые настройки очереди **Pre-фильтров** и записывают их в одну базу данных (или в разные кластеры одной и той же базы данных).

Можно также принимать данные в одной подсистеме, а отдавать их на обработку в другую подсистему. Однако такой механизм не принесёт ускорения, т.к. мы должны передавать данные всегда в одну и ту же подсистему, чтобы правильно выполнялись алгоритмы пре-фильтров, работающие с непрерывным потоком данных. А в этом случае такое “масштабирование” приведёт только к увеличению трафика и нагрузке на процессоры обоих подсистем.

Однако существует ещё одна возможность масштабирования – это выполнение **Online‑фильтров**, являющихся подмножеством **Post-фильтров**, в отдельной подсистеме (но только всегда в одной и той же). Т.к. результатом работы **Online‑фильтров** являются уведомления, и чистые **Online‑фильтры** являются окончанием ветки конвейера обработки данных, можно выполнить эту обработку в другой подсистеме, если таковая присутствует, а сама обработка ресурсоёмка.

Реализация такого механизма требует расширения интерфейса телематического модуля, а также небольшого изменения в интерфейсе **Pre-фильтра** (туда следует добавить явный признак того, что данный фильтр является **Online-фильтром**, т.е. не пишет ничего в БД).

## Список основных фильтров координат

|  |  |  |  |
| --- | --- | --- | --- |
| Фильтр | Тип | Параметры | Описание |
| Упрощение | Pre/Post | Tolerance - Smoothing coefficient (1-100). | Исключает точки, лежащие на одной прямой. |
| Скопления | Pre/Post | Radius - Minimal speed in km/h to determine the stop. MinPointsCount - Minimal coordinate points count which could be interpreted as congestion. | Объединяет выбросы координат вокруг реальной точки остановки в одну точку. |
| Разрывы | Online | MinDistance - Minimum distance in km between coordinate points for detecting track break. MinStopDuration - Minimal stop time. Calculated values that are less then this are determined as no stop.  AvgSpeed - Average speed in km/h.  DeviceStatusState – The bit number to set in last device status state property. | Устанавливает состояние отсутствия приёма валидных координат из-за ряда причин (нет GPS-соединения, нет поступающих данных и т.д.) |
| Разрывы | Post | MinDistance - Minimum distance in km between coordinate points for detecting track break. MinStopDuration - Minimal stop time. Calculated values that are less then this are determined as no stop.  AvgSpeed - Average speed in km/h. | Помечает соответствующие точки в результирующем массиве, что это точки разрыва. |
| Остановки | Online | MinSpeed - Minimal speed in km/h to determine the stop. MinStopDuration - Minimal stop time. Calculated values that are less then this are determined as no stop.  DeviceStatusState – The bit number to set in last device status state property. | Устанавливает состояние факта остановки устройства слежения. |
| Остановки | Post | MinSpeed - Minimal speed in km/h to determine the stop. MinStopDuration - Minimal stop time. Calculated values that are less then this are determined as no stop.  FilterAction - Actions for a filter over the track. | Помечает соответствующие точки в результирующем массиве, что это точки остановки. |
| Превышение скорости | Online | MaxSpeed - Maximum allowed speed in km/h. Speed that is greater than this value would be interpreted as over speed. Tolerance - Smoothing coefficient (1-100). Bigger value - less precise computation would be applied.  DeviceStatusState – The bit number to set in last device status state property. | Устанавливает состояние факта превышения скорости устройства слежения. |
| Превышение скорости | Post | MaxSpeed - Maximum allowed speed in km/h. Speed that is greater then this value would be interpreted as over speed. Tolerance - Smoothing coefficient (1-100). Bigger value - less precise computation would be applied. | Помечает соответствующие точки в результирующем массиве, что это точки превышения скорости. |
| Отсутствие координат | Online | Period - The minimum period without data before setting the state.  DeviceStatusState – The bit number to set in last device status state property. | Устанавливает состояние факта отсутствия поступления данных с устройства слежения. |

Видно, что один и тот же фильтр может быть одновременно как **Pre-фильтром**, так и Post-фильтром, по крайней мере по алгоритму и основному набору параметров, однако даже в таком случае их назначение, полный набор параметров и, возможно, реализация алгоритма может отличаться, так что их можно считать фактически разными фильтрами.

Внимательно присмотревшись, можно заметить, что **фильтр** **превышения скорости** и **фильтр** **остановок** можно объединить в один **фильтр** **скорости**, указывая в качестве параметров верхнюю и нижнюю границы допустимой скорости. Тогда при одном наборе параметров – это будет **фильтр остановок**, а при другом – **фильтр превышения скорости**.

## Список основных фильтров датчиков

|  |  |  |
| --- | --- | --- |
| Фильтр | Тип | Описание |
| Фильтр Кальмана | Pre/Post | Фильтрация шумов алгоритмом **Кальмана** |
| Определение аномалий | Post | Определение аномалий на выбранном множестве данных |
| Калибровочный | Post | Калибровка значений согласно переданной калибровочной таблицы |
| Вычисление производной | Pre/Post | Вычисление первой производной выбранного множества данных |
| LTTB-фильтр | Post | Фильтрация шумов методом **Largest-Triangle-Three-Buckets** |
| Медианный | Pre/Post | Усреднение результатов с методом медианы |
| Фильтр нормализации | Post | Нормализация данных в диапазон 0…1 |
| Определения пиков | Post | Определение пиков на выбранном множестве данных |
| Фильтр Савицкого-Голея | Post | Фильтрация шумов методом **Савицкого-Голея** |
| Разрыва данных | Post | Определение разрывов на выбранном множестве данных |
| Упрощения | Pre/Post | Упрощение выбранного множества данных таким образом, чтобы примерно одинаковые данные были исключены из результирующей выборки |
| Превышения порога | Online | Уведомление о выходе значения за указанные границы |
| Скользящего среднего | Pre/Post | Усреднение результатов с методом скользящего среднего |
| Преобразования | Pre/Post | Выполняет полиномиальное преобразование на переданными данными:  **y** = p0 + p1·**x** + p2·**x**2 + … + pn·**x**n |
| Отсутствия данных | Online | Уведомление об отсутствии данных с датчика в течение указанного периода времени |