# Общие положения

Предлагаемое решение основано на следующих утверждениях:

* фильтруются, исправляются и генерируются только исходящие UDP-пакеты, которые посылаются POS-сервером на ILASST-сервер;
* входящие UDP-пакеты, которые посылаются ILASST-сервером на POS-сервер, не перехватываются и не обрабатываются;
* для обработки и исправления исходящих UDP-пакетов используются только данные, которые доступны в POS DB;
* следующие NMEA строки (sentences) признаются потенциально содержащими ошибочные данные: PANSSY, PANSPT, PANSHT;
* для исправления ошибочных данных в PANSSY и PANSPT строках требуется обработка непосредственно исходящих UDP-пакетов;
* для исправления ошибочных данных в PANSHT строках требуются изменения только в POS DB.

# Модель обработки данных

В этом разделе предлагается модель минимально необходимых данных, вычислений и исправлений для решения обсуждаемой проблемы. Здесь не рассматриваются вопросы оптимизации и технические детали реализации.

## Таблица pos.pos

Предлагаются следующие правила для исправления записей в таблице ‘pos.pos’:

* каждый раз при добавлении новой записи в таблицу ‘pos.pos’ проверяется статус соответствующего тега в таблице ‘pos.tags’;
* если статус тега равен ‘ok’, то запись не изменяется;
* если статус тега равен ‘logged\_off’, то номер сектора в записи меняется на значение 67 (сектор ‘Landgang’);
* если статус тега равен ‘inactive’, то запись удаляется из таблицы (гипотетический случай, во время исследования не был обнаружен).

Далее в этом документе речь всегда идёт об уже исправленных записях и значениях из таблицы ‘pos.pos’.

## Кэш

Кэш содержит подмножество данных извлекаемых из POS DB, которые необходимы для исправления и генерирования UDP-пакетов. Предлагается следующий набор данных для кэширования:

cash\_tags[NT] - целочисленный массив из NT элементов (по количеству тегов в системе, обычно 450), где каждый элемент массива соответствует тегу и содержит порядковый номер сектора, в котором в данный момент зарегистрирован этот тег, или 0 - если тег не активирован.

cash\_sectors[NS] - целочисленный массив из NS элементов (по количеству секторов в системе, обычно 67), где каждый элемент массива соответствует сектору и содержит статус этого сектора в данный момент (0 - неактивный, 1 - активный).

Для оптимизации вычислений и уменьшения задержек при обработке UDP-пакетов кэш может содержать дополнительные наборы данных, которые, однако, не являются необходимыми для обсуждаемой здесь модели.

## Инициализация кэша

Первоначальная инициализация кэша производится по аналогии с первоначальной инициализацией SIS POS Service (данные получены в результате анализа SQL запросов к POS DB во время старта сервиса). Для инициализации используются только таблицы ‘pos.tags’ и ‘pos.sectors’.

Массив ‘cash\_tags’ инициализируется на основе данных из таблицы ‘pos.tags’ следующим образом:

* элементы массива, которые соответствуют тегам со статусом ‘inactive’ в таблице ‘pos.tags’, устанавливаются в значение 0 (сектор ‘Interne Verwendung’);
* элементы массива, которые соответствуют тегам со статусом ‘logged\_off’ в таблице ‘pos.tags’, устанавливаются в значение 67 (сектор ‘Landgang’);
* элементы массива, которые соответствуют тегам со статусом ‘ok’ в таблице ‘pos.tags’, устанавливаются в значение 66 (сектор ‘Außendeck’).

Массив ‘cash\_sectors’ инициализируется на основе данных из таблицы ‘pos.sectors’ следующим образом:

* элементы массива, которые соответствуют секторам без transition-сектора в таблице ‘pos.sectors’, устанавливаются в значение 1 (активный);
* элементы массива, которые соответствуют секторам c существующим transition-сектором в таблице ‘pos.sectors’, устанавливаются в значение 0 (неактивный).

## Обновление кэша

Обновление кэша происходит при изменении/добавлении данных в таблицах ‘pos.tags’, ‘pos.pos’ и ‘pos.sector\_status’.

Каждый раз, когда статус тега изменяется в таблице ‘pos.tags’, соответствующий элемент массива ‘cash\_tags’ изменяется по правилу, описанному в разделе “Инициализация кэша”.

Каждый раз, когда в таблицу ‘pos.pos’ добавляется новая запись (исправленная), соответствующий элемент массива ‘cash\_tags’ изменяется на значение сектора из этой записи.

Каждый раз, когда в таблицу ‘pos.sector\_status’ добавляется новая запись, соответствующий элемент массива ‘cash\_sectors’ изменяется на значение статуса из этой записи (‘ok’ = 1, ‘mailformed’ = 0).

## PANSSY строки

Используя только данные из POS DB невозможно вычислить или получить значения всех полей для PANSSY строки (например ‘POS2 Serverstatus’ или ‘INIT-POS’). Поэтому предлагается исправлять оригинальную PANSSY строку.

При получении UDP-пакета, который содержит PANSSY строку:

* значение поля ‘Gesamtanzahl TAGs’ меняется на количество тегов из массива ‘cash\_tags’, которые зарегистрированы в секторах с 1 по 66 включительно;
* значение поля ‘Anzahl vermisster TAGS’ меняется на количество тегов из массива ‘cash\_tags’, которые зарегистрированы в секторе 66 (сектор ‘Außendeck’).

## PANSPT строки

В данном случае используя только данные из кэша можно вычислить значения всех полей PANSPT строки для сектора S по следующим правилам:

* значение поля ‘Anzahl der TAGs im Sektor’ устанавливается равным количеству тегов из массива ‘cash\_tags’, которые зарегистрированы в секторе S;
* значение поля ‘Sektor Status’ устанавливается равным значению элемента массива ‘cash\_sectors’, который соответствует сектору S.

С точки зрения обработки UDP-пакетов возможны 2 варианта.

### Вариант №1 (удаление + генерация)

При получении UDP-пакета удалять все PANSPT строки, которые содержатся в нём.

И независимо генерировать UDP-пакеты с PANSPT строками в следующих случаях:

* каждый раз, когда элемент массива ‘cash\_tags’ изменяет значение (на отличное от 0), генерировать новый UDP-пакет с PANSPT строкой;
* каждые ‘pos.settings.msg\_interval\_fast’ секунд генерировать UDP-пакет(ы) с PANSPT строками для секторов с 1 по 67 включительно.

### Вариант №2 (исправление)

При получении UDP-пакета, который содержит PANSPT строки, исправлять все поля, используя данные из кэша (по правилам описанным выше).

## PANSHT строки

Дополнительной обработки полей из PANSHT строк не требуется, поскольку они формируются только из записей таблиц ‘pos.pos’ и ‘pos.sector\_status’. А записи таблицы ‘pos.pos’ уже содержат исправленные данные (как было описано выше).

# Техническая реализация

Ниже на диаграмме представлены основные компоненты предлагаемой технической реализации и связи между ними:

[!!! ВСТАВИТЬ ДИАГРАММУ СЮДА ИЛИ СОСЛАТЬСЯ НА ОТДЕЛЬНЫЙ ДОКУМЕНТ !!!]

Представленная техническая реализация состоит из 4 основных компонентов:

* SQL скрипт для POS DB;
* UDF Messenger DLL (Windows API, C++);
* UDP NMEA Proxy Service (.NET, С#);
* Windows Installer (опционально, на диаграмме отсутствует).

Ключевая особенность предлагаемой технической реализации - это использование технологии MySQL UDF, которая позволяет избежать частых периодических SELECT запросов к POS DB, а вместо этого использовать специальные сообщения для обновления кэша данных в UDP NMEA Proxy Service. Такое решение позволяет снизить нагрузку на MySQL Server и на всю систему в целом. Более подробная документация о технологии MySQL UDF доступна по ссылке:

<https://dev.mysql.com/doc/refman/5.7/en/adding-functions.html>

## SQL скрипт для POS DB

SQL скрипт содержит все изменения, которые необходимо внести в схему POS DB:

* триггеры для таблиц ‘pos.tags’, ‘pos.pos’ и ‘pos.sector\_status’;
* определения UDF: TAGMSG, POSMSG и SECMSG;
* таблицы ‘pos.real\_illast’ и ‘pos.proxy\_settings’.

### Триггеры

Триггер для таблицы ‘pos.tags’ вызывается каждый раз, когда статус тега изменяется. Этот триггер вызывает UDF TAGMSG.

Триггер для таблицы ‘pos.pos’ вызывается каждый раз, когда в таблицу добавляется новая запись. Этот триггер исправляет запись в соответствии с правилами, описанными в разделе “Таблица pos.pos”, а также вызывает UDF POSMSG.

Триггер для таблицы ‘pos.sector\_status’ вызывается каждый раз, когда в таблицу добавляется новая запись. Этот триггер вызывает UDF SECMSG.

### Определения UDF

UDF TAGMSG посылает сообщение об изменении статуса тега и принимает следующие аргументы:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Reference** |
| TAGID | INTEGER | pos.tags.id |
| STATUS | INTEGER | pos.tags.status |

UDF POSMSG посылает сообщение о регистрации тега в секторе и принимает следующие аргументы:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Reference** |
| TAGID | INTEGER | pos.pos.tagid |
| SECTORID | INTEGER | pos.pos.sectorid |

UDF SECMSG посылает сообщение об изменении статуса сектора и принимает следующие аргументы:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Reference** |
| SECTORID | INTEGER | pos.sector\_status.sectorid |
| STATUS | INTEGER | pos.sector\_status.status |

### Таблицы

Поскольку записи таблицы ‘pos.illast’ теперь будут содержать IP-адрес и порт UDP NMEA Proxy Service, то добавляется новая аналогичная таблица ‘pos.real\_illast’, которая будет содержать настоящие IP-адреса и порты ILASST серверов.

Также предлагается добавить новую таблицу ‘pos.proxy\_settings’ для хранения некоторых настроек для UDP NMEA Proxy Service. Определение этой таблицы полностью аналогично определению таблицы ‘pos.settings’. Полный список доступных настроек будет определён после окончания разработки UDP NMEA Proxy Service.

## UDF Messenger DLL

UDF Messenger DLL принимает сообщения, которые генерируются триггерами через вызов UDF, и помещает их в Named Pipe. Документация о Named Pipe доступна по ссылке:

<https://msdn.microsoft.com/en-us/library/windows/desktop/aa365590(v=vs.85).aspx>

Сообщения от UDF TAGMSG помещаются в Named Pipe в следующем формате:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Length** | **Value** |
| MessageType | char[4] | 4 bytes | “TAG\0” |
| TagId | DWORD | 4 bytes | TAGMSG.TAGID |
| TagStatus | DWORD | 4 bytes | TAGMSG.STATUS |

Сообщения от UDF POSMSG помещаются в Named Pipe в следующем формате:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Length** | **Value** |
| MessageType | char[4] | 4 bytes | “POS\0” |
| TagId | DWORD | 4 bytes | POSMSG.TAGID |
| SectorId | DWORD | 4 bytes | POSMSG.SECTORID |

Сообщения от UDF SECMSG помещаются в Named Pipe в следующем формате:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Length** | **Value** |
| MessageType | char[4] | 4 bytes | “SEC\0” |
| SectorId | DWORD | 4 bytes | SECMSG.SECTORID |
| SectorStatus | DWORD | 4 bytes | SECMSG.STATUS |

UDF Messenger DLL будет реализован на языке программирования C++ c использованием Windows API.

## UDP NMEA Proxy Service

UDP NMEA Proxy Service выполняет 2 основные задачи:

* инициализирует и поддерживает в актуальном состоянии кэш необходимых данных;
* перехватывает и обрабатывает исходящие UDP-пакеты, используя данные из кэша.

Предлагается реализовать UDP NMEA Proxy Service на языке программирования C# с использованием .NET Framework 3.5 или 4 (эти версии входят в системные требования POS сервера и поэтому уже должны быть установлены на каждом POS сервере). Использование .NET Framework позволит существенно сократить время разработки и упростить дальнейшую поддержку.

### Кэш данных

Кэш данных содержится в памяти UDP NMEA Proxy Service и инициализируется заново при каждом старте сервиса.

Данные необходимые для инициализации кэша извлекаются из POS DB посредством SELECT запросов из таблиц ‘pos.tags’ и ‘pos.sectors’. Далее кэш инициализируется по правилам, описанным в разделе “Инициализация кэша”.

После инициализации кэш постоянно обновляется для поддержания в актуальном состоянии. Для этого используются данные, которые содержатся в сообщениях, извлекаемых из Named Pipe. **Для обновления кэша не используются запросы к POS DB!** Кэш обновляется по правилам, описанным в разделе “Обновление кэша”.

### Обработка UDP-пакетов

UDP NMEA Proxy Service должен перехватывать все исходящие UDP-пакеты от SIS POS Service. Для этого в таблицу ‘pos.ilasst’ вместо записей для ILASST серверов вносятся записи для UDP NMEA Proxy Service. В свою очередь настоящие записи для ILASST серверов переносятся в таблицу ‘pos.real\_ilasst’.

Каждый перехваченный UDP-пакет проверяется на наличие потенциально ошибочной NMEA строки (PANSSY или PANSPT). Если такая строка не обнаружена, то UDP-пакет пересылается дальше на ILASST сервер без изменений.

Если обнаружена потенциально ошибочная NMEA строка, то оригинальный UDP-пакет не пересылается дальше, а каждая обнаруженная строка обрабатывается по правилам , которые описаны в разделах “PANSSY строки” и “PANSPT строки”. После этого формируется новый UDP-пакет (или в случае превышения максимального размера - 2 UDP-пакета), который пересылается дальше на ILASST сервер. При этом должен сохраняться оригинальный порядок всех перехваченных UDP-пакетов и всех NMEA -строк содержащихся в них.

### Режимы взаимодействия с ILASST

SIS POS Service поддерживает 2 основных режима взаимодействия с ILASST сервером:

* ‘Normalbetrieb ILASST’ - режим, в котором пересылаются все типы NMEA строк;
* ‘Anfrage des Gesamtstatus’ - режим, в котором пересылаются только PANSSY и PANSST строки.

Текущее состояние этого режима может быть считано из перехваченной PANSSY строки (поле H) и сохранено в памяти UDP NMEA Proxy Service. Далее текущий режим должен учитываться в случае независимой генерации PANSPT строк (см. вариант №1 из раздела “PANSPT строки”).

### Зависимости между сервисами

Для правильной работы рассматриваемого решения необходимо чтобы UDP NMEA Proxy Service всегда стартовал раньше SIS POS Service. Для этого UDP NMEA Proxy Service должен быть добавлен в список зависимостей SIS POS Service:

<https://serverfault.com/questions/24821/how-to-add-dependency-on-a-windows-service-after-the-service-is-installed>

## Windows Installer

Это опциональный компонент, который может быть разработан при необходимости. Вместо него может быть подготовлена пошаговая инструкция по ручной установке и настройке всех компонентов. Конкретный выбор зависит от количества POS серверов, на которых нужно устранить обсуждаемую проблему.

Основные задачи, которые могут быть выполнены Windows Installer, следующие:

* обновление схемы POS DB;
* настройка адресов ILASST серверов;
* установка UDF Messenger DLL;
* регистрация и настройка UDP NMEA Proxy Service;
* обновление зависимостей для POS сервисов;
* автоматическое удаление установленных компонентов и восстановление исходного состояния POS сервера.

# Оценки

Ниже в таблице приведены оценки трудозатрат для предлагаемого решения:

|  |  |  |
| --- | --- | --- |
| **Задача / Компонент** | **Оценка (man-days)** | **Примечание** |
| SQL-скрипт | 1 | SQL, 3 триггера, 3 UDF, 2 таблицы |
| UDF Messenger DLL | 3 | C++, Windows API, DLL, 3 UDF, Named Pipe, Thread safety |
| UDP NMEA Proxy Service | 5 | С#, .NET Framework, Cash, UDP-Filter, UDP-Generator, Named Pipe, SQL запросы, Service |
| Windows Installer | Кто будет делать? | Опциональный компонент |
| Тестирование в виртуальном окружении | 3 + 2 (?) | Зависит от наличия Windows Installer |

# Риски

TBD