ТИТУЛЬНЫЙ ЛИСТЗміст

[Вступ 4](#_Toc407209371)

[Абревіатури 5](#_Toc407209372)

[1 Огляд Використаних Бібліотек та програмного забезпечення 6](#_Toc407209373)

[1.1 IDE 6](#_Toc407209374)

[1.2 Конструювання GUI 7](#_Toc407209375)

[1.3 Використані бібліотеки 8](#_Toc407209376)

[1.3.1 Apache Commons Collections 9](#_Toc407209377)

[1.3.2 Apache Commons Lang 9](#_Toc407209378)

[1.3.3 ControlsFX 9](#_Toc407209379)

[1.3.4 Google Gson 10](#_Toc407209380)

[1.3.5 JavaHIDAPI 10](#_Toc407209381)

[1.3.6 jSSC 11](#_Toc407209382)

[1.3.7 Reflections 11](#_Toc407209383)

[1.3.8 usb4java 12](#_Toc407209384)

[2 Реалізація Протоколів і інтерфейсу 13](#_Toc407209385)

[2.1 Загальна схема роботи програми 13](#_Toc407209386)

[2.2 Пошук пристроїв 13](#_Toc407209387)

[2.3 Абстрактний клас Device та його реалізації 14](#_Toc407209388)

[2.3.1 Статичний фабричний метод 14](#_Toc407209389)

[2.3.2 Конкретні реалізації 15](#_Toc407209390)

[2.4 Взаємодія з пристроєм 17](#_Toc407209391)

[2.4.1 Прийняття даних з пристрою та генерування пакету 18](#_Toc407209392)

[2.5 Генерування даних для аналізу 19](#_Toc407209393)

[2.6 Реєстрація повідомлень 21](#_Toc407209394)

[2.7 Використання вбудованого Wi-Fi адаптера для сканування каналів 23](#_Toc407209395)

[2.8 Графічний інтерфейс 25](#_Toc407209396)

[2.8.1 Меню налаштувань 26](#_Toc407209397)

[2.8.2 Файли ресурсів 29](#_Toc407209398)

[2.9 Допоміжні класи 29](#_Toc407209399)

[3 Робота з аналізаторами спектру 31](#_Toc407209400)

[3.1 MetaGeek Wi-Spy 2.4i (Gen 1) 31](#_Toc407209401)

[3.1.1 Використання високорівневих функцій бібліотеки usb4java 31](#_Toc407209402)

[3.1.2 Використання низькорівневих функцій бібліотеки usb4java 32](#_Toc407209403)

[3.1.3 Використання JNI 32](#_Toc407209404)

[3.1.4 Розбір даних з пристрою 33](#_Toc407209405)

[3.2 MetaGeek Wi-Spy 2.4x2 34](#_Toc407209406)

[3.2.1 Ініціалізація 34](#_Toc407209407)

[3.2.2 Розбір даних з пристрою 35](#_Toc407209408)

[3.3 Texas Instruments ez430-RF2500 35](#_Toc407209409)

[3.3.1 Підключення до MDRV 40](#_Toc407209410)

[3.4 Ubiquiti AirView2 40](#_Toc407209411)

[3.4.1 Ініціалізація 40](#_Toc407209412)

[3.4.2 Розбір даних з пристрою 40](#_Toc407209413)

[3.5 Unigen ISM Sniffer (Wi-detector) 41](#_Toc407209414)

[3.6 Pololu Wixel 41](#_Toc407209415)

[Висновки 42](#_Toc407209416)

[Список літератури 43](#_Toc407209417)

[Додатки 44](#_Toc407209418)

# Вступ

# Абревіатури

|  |  |
| --- | --- |
| API | Application Programming Interface |
| CSS | Cascading Style Sheets |
| COM | Communication Port, Serial Port |
| CVS | Concurrent Versioning System |
| GNU | GNU’s Not UNIX |
| GNU GPL | GNU General Public License |
| HID | Human Interface Device |
| IRP | I/O request packet |
| JDK | Java Development Kit |
| JNI | Java Native Interface |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| NIO | Non-blocking I/O |
| PID | Product Identifier |
| RSSI | Received Signal Strength Indication |
| UI | User interface |
| USB | Universal Serial Bus |
| VID | Vendor Identifier |
|  |  |
| ОС | Операційна система |
|  |  |

1. Огляд Використаних Бібліотек та програмного забезпечення
   1. IDE

Для розробки проекту було використано IDE IntelliJ IDEA Community Edition (див. рис. 1). IntelliJ IDEA — комерційне інтегроване середовище розробки для Java від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії "Community Edition" і повнофункціональної комерційної версії "Ultimate Edition", для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні складання підготовлені для Linux, Mac OS X і Windows.

Community версія середовища IntelliJ IDEA підтримує інструменти для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial, Git і GitHub, засоби складання Maven і Ant, мови програмування Java, Java ME, Scala, Clojure, Groovy і Dart. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань.



1. Процес налагодження програми в IntelliJ IDEA
   1. Конструювання GUI

Для конструювання графічного інтерфейсу технологія JavaFX використовує формат розмітки FXML - це декларативна мова на основі XML, яка створена корпорацією Oracle для визначення інтерфейсу користувача JavaFX 2.0. Oracle надає спеціальну програму для роботи з FXML - JavaFX Scene Builder 2.0. Приклад роботи програми JavaFX Scene Builder можна побачити на рис. 2.



1. Конструювання файлу MainWindow.fxml у JavaFX Scene Builder
   1. Використані бібліотеки

В даному проекті використовуються бібліотеки сторонніх розробників. Для завантаження бібліотек було використано Maven Central Repository.

Бібліотеки для роботи з пристроями:

* JavaHIDAPI
* jSSC
* usb4java

Бібліотека для роботи з інтерфейсом користувача:

* ControlsFX

Бібліотека для роботи с форматом JSON:

* Google Gson

Бібліотеки, які розширюють стандартні можливості Java та підвищують якість коду в цілому:

* Apache Commons Collections
* Apache Commons Lang
* Reflections
  + 1. Apache Commons Collections

Java Collections Framework був важливим доповненням в JDK 1.2. Він додав, багато потужних структур даних, які прискорюють розробку найбільш значущих Java-додатків. Commons-Collections розвинули класи JDK шляхом надання нових інтерфейсів, реалізацій і утиліт. Надається під Apache License.

В даному проекті ця бібліотека використовується клас пакету BidiMap, який представляє собою мапу, ключ якої може буди використаний як значення, а значення як ключ.

* + 1. Apache Commons Lang

Стандартні Java-бібліотеки не в змозі забезпечити достатньо методів для маніпулювання основними класами. Apache Commons Lang надає ці додаткові методи.

Lang надає безліч допоміжних утиліт для java.lang API, зокрема методів маніпуляції String, основні чисельні методи, reflection, concurrency, створення і серіалізация властивостей системи. Крім того, він містить основні вдосконалення java.util.Date. Надається під Apache License.

В даному проекті ця бібліотека використовується для більш зручного визначення ОС, на якій запущена програма (клас org.apache.commons.lang3.  
SystemUtils).

* + 1. ControlsFX

ControlsFX є проект з відкритим кодом для JavaFX, яка покликана забезпечити дійсно високоякісний UI та інші інструменти, які доповнюють JavaFX. Бібліотека має JavaDoc документацію високої якості. Надається під BSD 3-Clause License.

В даному проекті бібліотека використовується як зручний засіб відображення діалогових вікон.

* + 1. Google Gson

Gson це бібліотека для перетворення об’єктів Java у формат JSON. Вона також може бути використана для перетворення рядка JSON до еквівалентного об’єкта Java. Gson може працювати з довільними Java об’єктами, включаючи вже існуючі об’єкти, на які не мають сирцевого коду. Надається під Apache License 2.0.

Переваги Gson:

* забезпечує прості методи toJson і fromJson для перетворення Java об’єктів в JSON і навпаки;
* вже існуючі об’єкти можуть бути перетворені в та з JSON;
* розширена підтримка Java Generics;
* користувацькі подання для об’єктів.

В даному проекті використовується для підтримки функціонування функцій налаштувань програми та зберігання та програвання збережених треків.

* + 1. JavaHIDAPI

Java HID API є JNI, що дозволяє використовувати бібліотеку HIDAPI з Java коду. HIDAPI є мультиплатформною бібліотекою, яка дозволяє додатку взаємодіяти з USB та Bluetooth пристроями HID-класу під Windows, Linux і Mac OS X.

Після створення екземпляра HIDManager можна використовувати деякі з його методів. Метод listDevices повертає список активних в даний момент часу HID. Кожен пристрій представлено екземпляром класу HIDDeviceInfo, який містить інформацію про пристрої. Щоб відкрити пристрій потрібно викликати метод Open.

HIDManager також надає кілька зручних методів для швидкого пошуку і відкриття пристрою або шляху (openByPath) або через vendor id / product id / serial number (openById).

Кожний відкритий пристрій представлено класом HIDDevice. Якщо пристрій відкрито кілька разів, HIDDevice буде однаковий, але безпека потоків не гарантується.

HIDAPI може бути використаний в рамках однієї з трьох ліцензій:

* GNU Public License, версія 3.0;
* BSD-стиль ліцензії;
* оригінал ліцензії HIDAPI.
  + 1. jSSC

jSSC (Java Simple Serial Connector) - бібліотека для роботи з COM портами з Java. jSSC підтримує Win32 (Win98-Win8), Win64, Linux (x86, x86-64, ARM), Solaris (x86, x86-64), Mac OS X 10.5 і вище (x86, x86-64, PPC, PPC64). Надається під GNU Lesser GPL.

В даному проекті ця бібліотека використовується як основний засіб для взаємодії з COM-пристроями (Pololu Wixel, TI ez430-RF2500, Ubiquiti AirView2).

* + 1. Reflections

В інформатиці відбиття або означає процес, під час якого програма може відстежувати і модифікувати власну структуру і поведінку під час виконання. Парадигма програмування, покладена в основу відображення, називається рефлексивним програмуванням. Це один з видів метапрограмування.

У більшості сучасних комп’ютерних архітектур програмні інструкції (код) зберігаються як дані. Різниця між кодом і даними в тому, що виконуючи код, комп’ютери обробляють дані. Тобто інструкції виконуються, а дані обробляються так, як написано цими інструкціями. Однак програми, написані за допомогою деяких мов, здатні обробляти власні інструкції як дані і виконувати, таким чином, рефлексивні модифікації. Такі самомодифікуючі програми в основному створюються за допомогою високорівневих мов програмування, що використовують віртуальні машини (наприклад, Smalltalk, скриптові мови).

Reflections сканує директорію класів, індексує метадані та дозволяє получати доступ до них під час виконання програми. Надається під Other Open Source License.

Використовуючи Reflections ви можете запросити такі метадані:

* підтипи певного типу;
* типи / constructos / методи / поля з анотацією;
* отримати всі погодження ресурсів відповідні регулярному виразу;
* отримати всі методи з конкретною сигнатурою, параметрами та типом повернення.

В даному проекті ця бібліотека використовується для знаходження всіх підкласів класу Device, тим самим генеруючи список всіх підтримуваних профілів пристроїв.

* + 1. usb4java

usb4java це бібліотека Java для доступу до USB-пристроїв. Вона заснована на native libusb 1.0 і використовує Java NIO буфери для обміну даними між libusb і Java. usb4java також підтримує стандарт javax-USB (JSR-80) через розширення usb4java-javax.

Підтримує платформи Linux (x86 32/64 біт, ARM 32 біт), OS X (x86 32/64 біт) і Windows (тільки x86 32/64 біт). Але інші платформи можуть працювати так само добре (якщо є Java 6 і підтримуються libusb) шляхом компіляції бібліотеки JNI вручну. Надається під LGPL.

В даному проекті бібліотека використовується тільки у якості допоміжної.

1. Реалізація Протоколів і інтерфейсу
   1. Загальна схема роботи програми

Загальна схема роботи програми виглядає наступним чином (див. рис. 3):



1. Загальна схема роботи програми

При підключені пристрою до системи програма ідентифікує його та активує подію підключення пристрою. Підписчик на подію DeviceConnectionHandler створює новий потік для роботи з підключеним пристроєм. Потік для роботи з пристроєм генерує подію генерування нового пакету. PacketAnalysis аналізує отримані дані та генерує подію, яка перехоплюється системою графічного відображення даних. PacketLogger зберігає пакети для можливості повторного використання.

* 1. Пошук пристроїв

DeviceConnectionListener (див. додаток 1) — клас, задача якого сканувати систему на предмет підключень пристроїв через задані проміжки часу (за замовчуванням — 1 с).

Клас реалізує патерн програмування Singleton, тому що немає сенсу запускати в одній програмі декілька екземплярів цього класу. Також реалізує патерн програмування Observer, за допомогою якого відбувається нотифікація підписчиків на подію підключення пристрою. Клас DeviceConnectionHandler (див. додаток 2) підписується на подію підключення пристрою, викликає статичний метод для пошуку конкретного пристрою, яке підтримує програма та, у випадку успіху, запускає потік для роботи з пристроєм.

* 1. Абстрактний клас Device та його реалізації

Device — клас, який абстрагує загальну поведінку для всіх пристроїв (див. додаток 3).

* + 1. Статичний фабричний метод

Звичайний спосіб отримання екземпляру класу — відкритий конструктор. Існує ще один метод: клас може забезпечити статичний фабричний метод, який повертає екземпляр класу.

Однією з переваг статичних фабричних методів є те, що на відміну від конструкторів, вони мають імена. Друга перевага статичних фабричних методів: вони не зобов’язані створювати новий об’єкт при виклику. Третя перевага статичних фабричних методів є те, що на відміну від конструкторів, вони можуть повернути об’єкт будь-якого підтипу.

У класі Device використовується статичний фабричний метод за наступною сигнатурою:

public static Device getConcreteDevice(DeviceInfo deviceInfo)

Клас DeviceConnectionHandler викликає фабричний метод getConcreteDevice з параметром DeviceInfo в якому знаходиться інформація о пристрої.

Задача методу getConcreteDevice: на основі даних з DeviceInfo повернути ініціалізований конкретний екземпляр класу пристрою. Це досягається за допомогою спеціального механізму Reflection. Використовуючи Reflection реалізовано пошук конкретного класу по всім можливим нащадкам абстрактного класу Device. Такий підхід дає можливість додавати підтримку реалізацій нових пристроїв не чіпаючи при цьому інші класи.

* + 1. Конкретні реалізації

Для реалізації конкретного пристрою треба заповнити шаблон DeviceTemplate (див. додаток 4):

public class DeviceTemplate extends Device

{

public final static String FRIENDLY\_NAME = "";

public final static String VENDOR\_ID = "";

public final static String PRODUCT\_ID = "";

public final static float INITIAL\_FREQUENCY = 2400f;

public final static float CHANNEL\_SPACING = 0f;

public final static byte[] END\_PACKET\_SEQUENCE = new byte[]{};

public final static boolean MANUAL\_DEVICE\_CONTROL = false;

@Override

public void initializeDevice()

{ }

@Override

public ArrayList<Byte> parse(ArrayList<Byte> dataToParse)

{

ArrayList<Byte> finalArray = new ArrayList<>(dataToParse);

return finalArray;

}

@Override

public byte[] customReadMethod()

{

return new byte[0];

}

}

Поле FRIENDLY\_NAME використовується для ідентифікації пристрою у графічному інтерфейсі.

Поле VENDOR\_ID використовується для зберігання ідентифікатору виробника пристрою. Повинен бути у шістнадцятковій системі числення. На приклад "1FFB".

Поле PRODUCT\_ID використовується для зберігання ідентифікатору пристрою. Повинен бути у шістнадцятковій системі числення.

Поле INITIAL\_FREQUENCY використовується для зберігання мінімальної частоти, яку пристрій може бачити (Base Frequency). Значення береться з документації к пристрою. На приклад 2400f.

Поле CHANNEL\_SPACING використовується для визначення між каналами. Значення береться з документації к пристрою. На приклад 327.450980f.

Поле END\_PACKET\_SEQUENCE використовується для зберігання символів кінця пакету. Символи кінця пакету використовуються класом RxRawDataReceiver для генерування пакетів зі значеннями RSSI.

Поле MANUAL\_DEVICE\_CONTROL встановлюється, коли треба отримати прямий контроль над USBHID. Це потрібно у ситуаціях, коли бібліотека за замовчуванням не спрацьовує за якихось причин. У цьому режимі програма не відкриватиме та не робитиме спроб зчитати з пристрою. Метод customReadMethod активується. Приклад – клас MetaGeekWiSpyGen1.

Метод initializeDevice з наступною сигнатурою:

public void initializeDevice()

використовується для ініціалізації пристрою, якщо це потрібно. Інакше тіло методу можна залишити пустим.

Метод parse з наступною сигнатурою:

public ArrayList<Byte> parse(ArrayList<Byte> dataToParse)

використовується для розбору даних, які були сформовані пристроєм. Формат повернення – масив байтів, кожен елемент якого – значення RSSI.

Метод customReadMethod з наступною сигнатурою:

public byte[] customReadMethod()

{

return new byte[0];

}

використовується для перевизначення стандартної поведінки USBHID. Для використання цього методу треба встановити прапорець MANUAL\_DEVICE\_CONTROL.

У шаблоні DeviceTemplate реалізована JavaDoc документація за допомогою якої кожен може додати в програму підтримку свого пристрою.

* 1. Взаємодія з пристроєм

Основним класом для взаємодії програми з пристроєм є абстрактний клас DeviceCommunication (див. додаток 5). Він реалізує шаблон програмування Singleton та має статичний фабричний метод getInstance, який повертає конкретну реалізацію, залежно від переданого параметра DeviceInfo. Сигнатура функції getInstance:

public static DeviceCommunication getInstance(DeviceInfo deviceInfo)

Всього є три конкретні реалізації абстрактного класу DeviceCommunication:

* COMDeviceCommunication – для взаємодії з COM-пристроями;
* HIDDeviceCommunication – для взаємодії з USBHID;
* DummyDeviceCommunication – для реалізації тестового програмного пристрою.

DeviceCommunication спроектований для роботи в окремому потоку за допомогою реалізації інтерфейсу Runnable. Код старту потоку наведено нижче:

Thread thread = new Thread(device.getDeviceCommunication());

thread.setName(device.getDeviceInfo().getFriendlyNameWithId());

thread.setDaemon(true);

thread.start();

У першому рядку створюється об’єкт потоку з параметром ініціалізованого конкретного класу DeviceCommunication.

У другому рядку задається ім’я потоку для його ідентифікації у разі виникнення виключення.

Методом setDaemon потік помічається як потік користувача. JVM завершує свою роботу тільки тоді, коли всі потоки, що залишилися, помічені як потоки користувача. Це запобігає ситуаціям, коли програма не може бути завершена оскільки один з потоків взаємодії з пристроєм не відповідає.

* + 1. Прийняття даних з пристрою та генерування пакету

Кожен об’єкт класу DeviceCommunication має вкладений клас RxRawDataReceiver,основна задача якого приймати сирі дані від пристрою, складати їх в пакет (масив значень RSSI) та сповіщати про складання нового пакету всіх підписчиків – класу реєстрації пакетів PacketLogger та класу аналізу пакетів PacketAnalysis.

Основний принцип роботи полягає у виклику методу receiveRawData з наступною реалізацією:

public void receiveRawData(byte[] rawData)

{

for (byte byteProcess : rawData)

processReceivedByte(byteProcess);

}

Цей метод працює у якості буфера для метода processReceivedByte, що приймає один байт. processReceivedByte накопичує прийнятий байт у буфері із якого у подальшому генерується пакет з RSSI. Для визначення кінця пакету використовується поле END\_PACKET\_SEQUENCE. При знаходженні потрібної послідовності послідовність переноситься у початок пакету (для ситуацій, коли послідовність кінця пакету знаходиться у початку наступного фізичного пакету з пристрою) та викликається методи, які зберігають зібранні дані та сповіщають підписчиків.

* 1. Генерування даних для аналізу

За генерування даних для аналізу відповідає клас PacketAnalysis. PacketAnalysis генерує наступні масиви даних: мода, медіана, середнє арифметичне, максимальне.

Для генерування масиву максимальних значень використовується спеціальна функція, яка приймає як параметри поточний та попередній масив значень, порівнює значення цих двох масивів та заносить більше значення у масив, який повертається.

Для генерування масиву з середніми арифметичними значеннями використовується формула, де – значення RSSI на заданій частоті, а – загальна кількість пакетів зі значеннями RSSI для конкретного пристрою:

Основною проблема була в лінійному збільшенні часу на розрахунок оскільки кількість пакетів зі значеннями RSSI постійно збільшується, що лінійно збільшує кількість ітерацій. Рішення наступне: для запобігання повторного розрахунку усіх значень при кожному новому пакеті введено структуру даних, яка зберігає суму усіх попередніх значень RSSI () для кожної частоти для конкретного пристрою та загальну кількість прийнятих пакетів (див. рис. 4). Це дозволяє, при надходженні нового масиву RSSI, просто додати нове значення до вже наявної суми та розділити на загальну кількість.

Для підрахунку моди треба визначити значення, що трапляється найчастіше. Якщо сканувати кожен раз всі надіслані пакети, то, як і у випадку з середнім значенням, складність розрахунку буде лінійно зростати. Тому введено допоміжну структуру наступного виду:

HashMap < [Пристрій] -> ArrayList < HashMap < [RSSI] -> [Кількість] > > >

Кожному конкретному пристрою відповідає масив мап з RSSI у якості ключа та кількістю значень RSSI у якості значення. На приклад маємо наступні значення (див. рис. 4): , які відповідають наступному матричному представленню, у якому кожному пристрою (Device) відповідає масив (ArrayList, рядки матриці), в якому знаходиться мапа з відношенням значень RSSI до їх кількості:

При кожному надходженні пакету до поточної кількості значень RSSI до дається 1 у випадку колізії. Така структура дає можливість використовувати накопичені значення для підрахунку моди з постійною швидкістю – потрібно лише взяти ключ з найбільшою кількістю значень. Ця ж структура використовується і для пошуку медіани – масив сортується та береться значення, яке знаходиться посередині масиву.



1. Схематичне відображення масивів RSSI у часі
   1. Реєстрація повідомлень

Для реєстрації повідомлень використовується вбудований клас java.util.logging.Logger пакету java.util.logging. Об’єкт Logger використовується для запису повідомлень для конкретної системи або компонента програми. Logger, як правило, використовує ієрархічну структуру імен. Імена можуть бути довільними, як правило, встановлюються на основі імені пакета чи імені класу. Крім того, можна створювати «анонімний» Logger, що не зберігаються в просторі імен реєстратора.

Об’єкти Logger може бути отримана шляхом виклику getLogger factory methods, який створить новий Logger або поверне підходящий існуючий. Важливо відзначити, що Logger повернений одним з методів getLogger може бути знищений при збірці сміття в будь-який момент, якщо на нього немає посилання.

Повідомлення будуть пересилатися на зареєстровані оброблювачі, який може пересилати повідомлення в різні напрямки, в тому числі консолі, файли, журнали ОС і т. п.

Реєстратор повідомлень підтримує декілька рівнів повідомлень: OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST.

В даному проекті за реєстрацію повідомлень відповідає клас ApplicationLogger. Приклад використання:

ApplicationLogger.setup();

ApplicationLogger.LOGGER.info("MetaGeek\_WiSpy24x2 has been initialized");

Метод ApplicationLogger.setup викликається один раз при старті програми для ініціалізації об’єкту реєстратора повідомлень та створення log-файлу. У класі MyLogFormatter, який розширює клас Formatter, вказано формат виводу за допомогою пере визначення наступної функції, яка викликається для кожної реєстрації повідомлення:

@Override

public String format(LogRecord record)

Можливий варіант зареєстрованих повідомлень виглядає так:

16.12.14 12:55:48.523 [INFO] [logging.ApplicationLogger.setup] Logger has initialized

16.12.14 12:55:48.572 [INFO] [updater.UpdateChecker.initCurrentVersion] Current version: Version{major=0, minor=3}

16.12.14 12:55:49.385 [INFO] [connectionlistener.DeviceConnectionListener.runSchedule] Listening schedule has started. Waiting for devices...

16.12.14 12:55:49.397 [WARNING] [wirelessadapter.WirelessAdapterCommunication.searchWirelessAdapters] phy#0

16.12.14 12:55:49.418 [INFO] [connectionlistener.DeviceConnectionListener.deviceConnectionEvent] USB Device CONNECTED

16.12.14 12:55:49.448 [INFO] [wirelessadapter.WirelessAdapter.setUpChannelToFrequencyMap] channelToFrequencyMap: {1=2412, 2=2417, 3=2422, 4=2427, 5=2432, 6=2437, 7=2442, 8=2447, 9=2452, 10=2457, 11=2462, 12=2467, 13=2472, 14=2484}

Формат виводу наступний:

[Date] [Time] [Logging Level] [The Class from which logger has been called] [Message]

* 1. Використання вбудованого Wi-Fi адаптера для сканування каналів

Якщо при завантажені програма встановлює, що в системі є бездротовий модуль з підтримкою Monitor mode, то програма автоматично розпочинає процедуру сканування, використовуючи заданий в меню налаштувань діапазон.

WirelessAdapterCommunication – клас, задача якого розпочати процедуру сканування. Для цього спочатку виконуються пошук адаптерів:

ArrayList<WirelessAdapter> wirelessAdapters = searchWirelessAdapters();

Якщо хоч один адаптер був знайдений, то викликається функція вибору адаптеру, яка виводить користувачу список із доступних адаптерів. Варто відмітити, що функція вибору адаптеру автоматично повертає адаптер, якщо він один:

WirelessAdapter wirelessAdapter = chooseWirelessAdapter(wirelessAdapters);

Після вибору адаптеру програма переводить його у Monitor mode, використовуючи вмонтовані в дистрибутив програми ifconfig та iw:

switchToMonitorMode(wirelessAdapter);

Далі розпочинається переключення каналів у заданому діапазоні та активація програми tcpdump, яка дозволяє захоплювати і аналізувати мережний трафік, що проходить через комп'ютер, на якому запущена ця програма:

startChannelSwitching(wirelessAdapter);

startListening(tcpDumpCommand, wirelessAdapter);

Приклад роботи наведено на наступному рисунку (див. рис. 5). Поточне значення RSSI підтверджує завантаженість діапазону частот, що підсвічується.



1. Приклад роботи мережевої карти у Monitor mode. Завантаженість в діапазоні 2,435 – 2,465 МГц
   1. Графічний інтерфейс

Графічний інтерфейс реалізовано за допомогою технології JavaFX. JavaFX — платформа та набір інструментів для створення насичених інтернет застосунків (англ. Rich Internet Applications, RIA) з можливістю підвантаження медіа та змісту.

Графічний інтерфейс програми зображено на рис. 6.



1. Графічний інтерфейс програми

Під номером 1 виділено елемент GUI LineChart, який відповідає за відображення та оновлення графіків. Вісь абсцис відповідає частоті та вимірюється в мегагерцах. Вісь ординат відповідає RSSI – рівню прийнятого сигналу.

Під номером 2 зображено список підключених пристроїв. Список також включає в себе кнопки управління відображенням графіків максимального, моди, медіани, середнього та поточного значень. Також користувач може включити чи виключити одразу всі графіки за допомогою CheckBox навпроти назви пристрою.

Під номером 3 виділено плаваючу підказку з поточним, виділеним курсором, значенням RSSI та відповідною частотою.

Під номером 4 зображено область, яка, при наведені на неї курсору, дає доступ до меню налаштувань.

При створенні пакет зі значеннями RSSI також створюється мітка часу, до якої прив’язаний створений пакет. Під номером 5 виділено слайдер, який дозволяє переміщуватись по шкалі часу.

* + 1. Меню налаштувань

На наступному рисунку можна побачити меню налаштувань.



1. Меню налаштувань

Кнопка **Open replay** потрібна для відкриття файлу з попередньо записаними даними.

Кнопка **Refresh chart** потрібна для очищення графіку від набраної інформації.

Кнопка **Show debug info** включає режим показу додаткової інформації, яку можна використовувати для налагодження та тестування програми.

Кнопка **Add Dummy** потрібна для додавання до програми тестового програмного пристрою.

CheckBox **Horizontal line** та **Vertical line** відповідають за відображення допоміжних ліній на графіку.

За замовчуванням програма відображає значення пакетів RSSI в реальному часі, тобто індикатор шкали часу завжди знаходиться вкінці шкали. CheckBox **Manual replay mode** дозволяє включити режим ручного управління часом.

CheckBox **Animation** дозволяє включити або виключити анімацію. На деяких слабких системах це є актуально.

У полі **Chart update delay** можна встановити інтервал оновлення графіку. Ця функція корисна для слабких систем та у випадках, коли треба аналізувати поточні дані з пристрою в реальному часі.

Наступний блок відповідає за налаштування відображення завантаженості каналу, що сканується бездротовим адаптером. У полі **Fade out after** вказується кількість секунд, до зникнення ефекту завантаженості каналу. У полі **Max opacity** вказується ступінь непрозорості ефекту завантаженості каналу. У полі **Fade up opacity** вказується коефіцієнт, що визначає як швидко буде ефект завантаженості каналу буде набирати непрозорості.

У наступному полі вказується канал чи діапазон каналів, що скануються. Відповідає наступному регулярному виразу:

"(?<channelStart>\\d{1,2}[^15-99]\*?)(-(?<channelEnd>\\d{1,2}[^15-99]\*?))?"

Тобто доступні значення для вводу від 1 до 14 включно. Щоб вказати діапазон значень треба скористатися символом тире. Канали будуть переключатися у тій послідовності, що вказана у полі.

При підключені пристрою в меню налаштувань автоматично генерується налаштування для щойно підключеного пристрою та зазвичай потрібні для калібрування пристрою. Поле **Channel spacing** відповідає за відстань між значеннями RSSI на графіку. Поле **RSSI shift** відповідає за зсув значень RSSI на графіку по шкалі абсцис.

* + 1. Файли ресурсів

Окрім файлів FXML, які потрібні для розмітки об’єктів інтерфейсу, було використано каскадні таблиці стилів (CSS) для налаштування стилю графічного об’єкту. На приклад, для налаштування стилю списку підключених пристроїв використано наступний код:

#chartLegendVbox{

-fx-background-color: rgba(255, 255, 255, 0.6);

-fx-background-radius: 5;

}

#chartLegendVbox – звернення до графічного об’єкту за його ідентифікатором.

-fx-background-color – задання кольору фону у форматі rgba.

-fx-background-radius – задання округлення кутів фону.

* 1. Допоміжні класи

В програмі використовуються спеціальні класи, в яких сформовані статичні методи. Ці методи використовуються у програмі у різних місцях. Класи можна знайти у пакеті com.rasalhague.mdrv.Utility. Наприклад, наступна функція повертає псевдовипадкове число типу int з заданого діапазону:

public static int randInt(int min, int max)

Для приведення PID чи VID пристрою до чотирьох символів використовується функція

public static String normalizePidVidToLength(String str)

Щоб виклику Runnable в JFX Thread і чекати доки він не завершиться можна використати функцію

public static void runAndWait(final Runnable run) throws InterruptedException, ExecutionException

Для завантаження Native бібліотек використано функцію, яка розпаковує обрану бібліотеку до тимчасової директорії,

public static void loadLibraryFromJar(String path) throws IOException

1. Робота з аналізаторами спектру
   1. MetaGeek Wi-Spy 2.4i (Gen 1)

MetaGeek Wi-Spy Gen 1 як і вся лінійка пристроїв MetaGeek Wi-Spy підключається до операційної системи як HID. Підключенні даного пристрою до програмного комплексу MDRV за допомогою стандартної бібліотеки JavaHIDAPI не вдалося ні під Linux ні під Windows. Ось так виглядає виключення:

java.io.IOException: The supplied user buffer is not valid for the requested operation.

at com.codeminders.hidapi.HIDDevice.read(Native Method)

З коду помилки видно, що функція читання, яка являє собою Native Method, вважає переданий їй буфер, в який треба зчитати данні, не дійснім.

Далі наведено шляхи вирішення проблеми.

* + 1. Використання високорівневих функцій бібліотеки usb4java

У різних бібліотеках реалізації можуть відрізнятися, тому було вирішено спробувати бібліотеку usb4java. Але при спробі виконати читання даних:

int received = pipe.syncSubmit(data);

програма зависає.

При спробі відіслати на пристрій IRP - структуру даних ядра Windows, яка забезпечує обмін даними між програмою та драйвером, а також між драйвером та драйвером:

irp = device.createUsbControlIrp((byte) (UsbConst.REQUESTTYPE\_DIRECTION\_IN |

UsbConst.REQUESTTYPE\_TYPE\_STANDARD |

UsbConst.REQUESTTYPE\_RECIPIENT\_DEVICE),

UsbConst.REQUEST\_CLEAR\_FEATURE, (short) 8, (short) 8);

irp.setData(data);

device.syncSubmit(irp);

виникає виключення:

javax.usb.UsbPlatformException: USB error 2: Unable to submit control message: Invalid parameter

Ситуація схожа з кореневою проблемою з буфером обміну.

* + 1. Використання низькорівневих функцій бібліотеки usb4java

При використанні низькорівневих функцій бібліотеки usb4java ситуація не змінилась – проблема з буфером обміну.

* + 1. Використання JNI

Основуючись на програмі Kismet Spectools, працювала з даним пристроєм, для вирішення проблеми було вирішено зробити JNI до С++. Ключовим фактором для роботи пристрою став виклик функції бібліотеки libusb

int libusb\_detach\_kernel\_driver (libusb\_device\_handle \*dev, int interface\_number)

яка працює тільки для ОС Linux. Вона виконує від’єднання драйверу ядра внаслідок чого можна зробити захват інтерфейсу:

libusb\_detach\_kernel\_driver(dev\_handle, 0)

libusb\_claim\_interface(dev\_handle, 0);

Після цього можна зчитати данні з пристрою:

libusb\_control\_transfer(dev\_handle, USB\_ENDPOINT\_IN + USB\_TYPE\_CLASS + USB\_RECIP\_INTERFACE, HID\_GET\_REPORT, (HID\_RT\_FEATURE << 8), 0, buf, buf\_size, TIMEOUT);

* + 1. Розбір даних з пристрою

Пристрій повертає пакети довжиною в 8 символів:

[0, 4, 2, 2, 1, 2, 1, 1]

[7, 2, 3, 7, 0, 0, 2, 0]

[14, 1, 2, 0, 1, 2, 2, 3]

[21, 1, 2, 1, 3, 2, 2, 1]

[28, 1, 3, 3, 2, 0, 0, 2]

[35, 1, 0, 1, 1, 1, 3, 1]

[42, 2, 1, 2, 2, 1, 1, 3]

[49, 2, 2, 1, 16, 31, 31, 16]

[56, 2, 1, 1, 3, 1, 2, 1]

[63, 3, 1, 3, 2, 1, 2, 3]

[70, 3, 1, 1, 2, 2, 1, 2]

[77, 1, 2, 0, 2, 2, 1, 0]

Перший елемент масиву можна інтерпретувати як кількість вже переданих значень RSSI, чи як порядковий номер наступного значення RSSI. Від другого до восьмого включно — значення RSSI.

Після того, як пристрій пробігає по всьому спектру, данні приводяться до чисельного типу та заносяться у масив, до якого, у кінець, підмішується символ кінця пакету, заданий константою END\_PACKET\_SEQUENCE:

END\_PACKET\_SEQUENCE = new byte[]{-1};

[4, 2, 2, 1, 2, 1, 1, 2, 3, 7, 0, 0, 2, 0, 1, 2, 0, 1, 2, 2, 3, 1, 2, 1, 3, 2, 2, 1, 1, 3, 3, 2, 0, 0, 2, 1, 0, 1, 1, 1, 3, 1, 2, 1, 2, 2, 1, 1, 3, 2, 2, 1, 16, 31, 31, 16, 2, 1, 1, 3, 1, 2, 1, 3, 1, 3, 2, 1, 2, 3, 3, 1, 1, 2, 2, 1, 2, 1, 2, 0, 2, 2, 1, 0, -1]

* 1. MetaGeek Wi-Spy 2.4x2
     1. Ініціалізація

Для того, щоб ініціалізувати пристрій потрібно знати спеціальну послідовність ініціалізації, яку потрібно передати пристрою. Послідовність ініціалізації представляє собою набір команд, які пристрій здатний сприймати. У даному випадку офіційної документації немає, тому використаємо сніффер USB трафіку USBlyzer у поєднанні з програмою MetaGeek Chanalyzer, у якій вже реалізовано ініціалізацію пристрою (див. рис. 8).



1. . Сніффінг USB трафіку для визначення послідовності ініціалізації

При спробі відправити команду ініціалізації до пристрою під ОС Windows виникає наступне виключення:

java.io.IOException: The parameter is incorrect.

at com.codeminders.hidapi.HIDDevice.write(Native Method)

яке говорить про те, що в Native Method був переданий не коректний параметр. Треба відмітити, що той самий програмний код коректно працює під Linux та не працює під Windows. У подальшому можлива спроба виправити це за допомогою JNI, а поки що цей пристрій підтримується тільки на Linux.

* + 1. Розбір даних з пристрою

За одну передачу пристрій видає наступну послідовність:

[74, 0, 0, 0, 0, 33, 67, 67, 69, 66, 68, 65, 67, 66, 66, 64, 67, 67, 67, 67, 67, 67, 70, 62, 68, 68, 64, 66, 68, 68, 67, 68, 67, 68, 64, 67, 68, 68, 65, 67, 66, 68, 69, 65, 66, 66, 68, 68, 65, 69, 67, 66, 67, 66, 67, 68, 66, 66, 66, 68, 67, 67, 68, 65, 74, 59]

Структура пакету схожа на MetaGeek Wi-Spy 2.4i Gen 1. Перший елемент масиву можна інтерпретувати як кількість вже переданих значень RSSI, чи як порядковий номер наступного пакету значень RSSI. Від другого до п’ятого включно — сервісна інформація. Всі інші — значення RSSI зі здвигом на 170 одиниць.

* 1. Texas Instruments ez430-RF2500

Перед застосуванням, пристрій треба було попередньо налаштувати. Проблема була в тому, що пристрій не бачив частину спектру. Далі наводяться графіки, що підтверджують проблему.



1. . 1-й канал під загрузкою



1. . 6-й канал під загрузкою



1. . 11-й канал під загрузкою

З графіків вище видно, що діапазон частот до шостого каналу, тобто частоти 2401–2426, а також після 11-го каналу, тобто частоти 2473–2483 випадають із зони видимості пристрою.

Для вирішення даної проблеми треба:

1. Визначити, які регістри відповідають за потрібні налаштування та встановити коректні параметри.
2. Прошити пристрій виправленим кодом.
3. Провести тестування.

У ході роботи над даним пристроєм було використано програмне забезпечення SmartRF Studio, та програмний код із статті Creating a Spectrum Analyzer to Measure Noise. Для компіляції програмного коду та відправки його до пам’яті пристрою було використано IDE IAR Embedded Workbench.

Пошук потрібних регістрів та встановлення коректних параметрів

У заготовочному файлі source\_code\drivers\mrfi\smartrf\CC2500\smartrf  
\_CC2500.h було знайдено об’явлення регістрів. Нижче приведені регістри, які відповідають налаштуванню Base Frequency:

#define SMARTRF\_SETTING\_FREQ2 0x5D

#define SMARTRF\_SETTING\_FREQ1 0x93

#define SMARTRF\_SETTING\_FREQ0 0xB1

Для перевірки введемо ці значення до SmartRF Studio.





1. . Перевірка значень регістрів у SmartRF Studio

Рис. 12 вказує, що при таких значеннях регістрів Base Frequency = 2433, що підтверджує проблему. Підставимо коректні значення Base Frequency:





1. . Коректні значення регістрів для Base Frequency

#define SMARTRF\_SETTING\_FREQ2 0x5С //old val 0x5D

#define SMARTRF\_SETTING\_FREQ1 0x58 //old val 0x93

#define SMARTRF\_SETTING\_FREQ0 0x9В //old val 0xB1

Тестування змін:



1. . Тестування скорегованого Base Frequency

Рис. 14 засвідчує коректність нових значень Base Frequency.

Далі скорегуємо значення регістрів Channel Spacing, який відповідає за відстань між дискетами, так, щоб Carrier frequency був близьким до максимального для даного пристрою (2484 МГц). Для цього використаємо поле Channel number та Carrier frequency (див. рис. 15).





1. . Корегування значень регістру Channel Spacing

Рис. 16 засвідчує коректність змін Channel Spacing.



1. . Тестування змін Channel Spacing
   * 1. Підключення до MDRV

Пристрій визначається системою як COM. Формат пакету дуже простий – спочатку йдуть значення RSSI, а у кінці – символ кінця строки (‘\n’). Пристрій ініціалізується самостійно при підключенні до комп’ютера.

* 1. Ubiquiti AirView2
     1. Ініціалізація

Визначається як COM. Пристрій ініціалізується за допомогою передачі на нього спеціальних послідовностей, які приведені нижче:

byte[] intByte = new byte[]{0x69, 0x6E, 0x74}; //int

byte[] bsByte = new byte[]{0x0A, 0x62, 0x73, 0x0A}; //.bs.

Відповідно до ASCII перша послідовність означає “int”, тобто “initialize”. Друга послідовність – “.bs.” – “begin scan”.

* + 1. Розбір даних з пристрою

Пристрій визначається системою як COM. Формат пакету дуже простий: спочатку йдуть значення RSSI, а у кінці — символ кінця строки (‘\n’).

* 1. Unigen ISM Sniffer (Wi-detector)

Визначається як USBHID. Пристрій ініціалізується та починає передавати значення RSSI самостійно, при підключенні до комп’ютера. Значення RSSI потрібно корегувати за наступною формулою:

(((aByte - 135) + 100) \* 1.428) – 100

Де aByte – передане пристроєм значення.

* 1. Pololu Wixel

Визначається як COM. Пристрій ініціалізується та починає передавати значення RSSI самостійно, при підключенні до комп’ютера. Пакет, як і у Ubiquiti AirView2 та TI ez430-RF2500 спочатку значення RSSI, а у кінці — символ кінця строки (‘\n’).

# Висновки

# Список літератури

1. Стаття Creating a Spectrum Analyzer to Measure Noise - <http://cnx.org/contents/35441aa1-e93c-4c5b-82fe-bc1aa16bf6fd@10.6:19/eZWSN:_Experimenting_with_Wire>

# Додатки

1. Лістинг класу DeviceConnectionListener

package com.rasalhague.mdrv.connectionlistener;

import com.codeminders.hidapi.HIDDeviceInfo;

import com.codeminders.hidapi.HIDManager;

import com.rasalhague.mdrv.device.core.DeviceInfo;

import com.rasalhague.mdrv.logging.ApplicationLogger;

import jssc.SerialPortList;

import java.io.IOException;

import java.util.\*;

/\*\*

\* The type Device connection listener.

\* <p>

\* Singleton. Listen for a new device connections.

\*/

public class DeviceConnectionListener {

private static final DeviceConnectionListener INSTANCE = new DeviceConnectionListener();

static {

com.codeminders.hidapi.ClassPathLibraryLoader.loadNativeHIDLibrary();

}

private final ArrayList<DeviceInfo> dummyDeviceList = new ArrayList<>();

private final ArrayList<DeviceInfo> connectedDeviceList = new ArrayList<>();

private final List<DeviceConnectionListenerI> listeners = new ArrayList<>();

private final long scanTimerPeriodMs = 1000;

private boolean isListening = false;

private Timer timer;

private DeviceConnectionListener() { }

/\*\*

\* Gets INSTANCE.

\*

\* @return the INSTANCE

\*/

public static DeviceConnectionListener getInstance() {

return INSTANCE;

}

/\*\*

\* Start listening.

\*/

public void startListening() {

runSchedule();

}

/\*\*

\* Stop listening.

\*/

public void stopListening() {

cancelSchedule();

}

/\*\*

\* Add dummy device.

\* <p>

\* DummyDevice name must be "DummyDevice [SeqNumb]"

\*/

public void addDummyDevice() {

Random random = new Random();

dummyDeviceList.add(new DeviceInfo("1111", "1111", "DummyDevice " + (dummyDeviceList.size() + 1),

"DummyPort " + (dummyDeviceList.size() + 1), DeviceInfo.DeviceType.DUMMY,

new byte[]{10}, 2399, 500));

}

/\*\*

\* Check for listening.

\*

\* @return the boolean

\*/

public boolean isListening() {

return isListening;

}

/\*\*

\* Add listener.

\*

\* @param toAdd

\* the to add

\*/

public void addListener(DeviceConnectionListenerI toAdd) {

listeners.add(toAdd);

}

private void runSchedule() {

if (!isListening) {

timer = new Timer();

TimerTask timerTask = new TimerTask() {

@Override public void run() {

scanForDeviceConnections();

}

};

long timerDelayMs = 0;

timer.schedule(timerTask, timerDelayMs, scanTimerPeriodMs);

isListening = true;

ApplicationLogger.LOGGER.info("Listening schedule has started. Waiting for devices...");

}

else {

cancelSchedule();

runSchedule();

}

}

private void cancelSchedule() {

timer.cancel();

isListening = false;

ApplicationLogger.LOGGER.info("Listening schedule has canceled.");

}

private void scanForDeviceConnections() {

ArrayList<DeviceInfo> combinedList = new ArrayList<>();

combinedList.addAll(getCOMPortsList());

combinedList.addAll(getHIDDevicesList());

combinedList.addAll(getDummyDeviceList());

updateConnectedDeviceList(combinedList);

}

/\*\*

\* Get current connected ports via ArrayList<DeviceInfo>

\*

\* @return current connected ports

\*/

private ArrayList<DeviceInfo> getCOMPortsList() {

//Get connected port names

String[] portNames = SerialPortList.getPortNames();

//Generate array list from portNames

ArrayList<DeviceInfo> deviceInfoList = new ArrayList<>();

for (String portName : portNames) {

deviceInfoList.add(new DeviceInfo(portName));

}

return deviceInfoList;

}

private ArrayList<DeviceInfo> getHIDDevicesList() {

try {

HIDDeviceInfo[] hidDeviceInfos = HIDManager.getInstance().listDevices();

//Generate array list from portNames

ArrayList<DeviceInfo> deviceInfoList = new ArrayList<>();

if (hidDeviceInfos != null) {

for (HIDDeviceInfo hidDeviceInfo : hidDeviceInfos) {

deviceInfoList.add(new DeviceInfo(hidDeviceInfo));

}

}

return deviceInfoList;

}

catch (IOException e) {

e.printStackTrace();

}

return null;

}

private ArrayList<DeviceInfo> getDummyDeviceList() {

return this.dummyDeviceList;

}

//region Observer implementation

/\*\*

\* Analise previously scan with current for find new conn or disconn

\*

\* @param scannedDevicesList

\*/

private void updateConnectedDeviceList(ArrayList<DeviceInfo> scannedDevicesList) {

//adding

for (DeviceInfo deviceInfo : scannedDevicesList) {

if (!connectedDeviceList.contains(deviceInfo)) {

connectedDeviceList.add(deviceInfo);

performDeviceConnectionEvent(deviceInfo, DeviceConnectionStateEnum.CONNECTED);

}

}

//removing

//create temp array coz ConcurrentModificationException

ArrayList<DeviceInfo> clone = new ArrayList<>();

clone = clone.getClass().cast(connectedDeviceList.clone());

for (DeviceInfo deviceInfo : clone) {

if (!scannedDevicesList.contains(deviceInfo)) {

performDeviceConnectionEvent(deviceInfo, DeviceConnectionStateEnum.DISCONNECTED);

connectedDeviceList.remove(deviceInfo);

}

}

}

private void performDeviceConnectionEvent(DeviceInfo deviceName, DeviceConnectionStateEnum connectionStateEnum) {

// Notify everybody that may be interested.

for (DeviceConnectionListenerI listenerI : listeners) {

listenerI.deviceConnectionEvent(deviceName, connectionStateEnum);

}

}

//endregion

}

1. Лістинг класу DeviceConnectionHandler

package com.rasalhague.mdrv.connectionlistener;

import com.rasalhague.mdrv.analysis.PacketAnalysis;

import com.rasalhague.mdrv.device.core.Device;

import com.rasalhague.mdrv.device.core.DeviceInfo;

import com.rasalhague.mdrv.logging.ApplicationLogger;

import com.rasalhague.mdrv.logging.PacketLogger;

import com.rits.cloning.Cloner;

public class DeviceConnectionHandler implements DeviceConnectionListenerI {

@Override

public void deviceConnectionEvent(DeviceInfo connectedDevice, DeviceConnectionStateEnum deviceConnectionStateEnum) {

DeviceInfo deviceInfoClone = new Cloner().deepClone(connectedDevice);

ApplicationLogger.LOGGER.info(deviceInfoClone.getName() + " " + deviceConnectionStateEnum);

if (deviceConnectionStateEnum == DeviceConnectionStateEnum.CONNECTED) {

//Call Factory method

Device device = Device.getConcreteDevice(deviceInfoClone);

//filter known device

if (device != null) {

device.getRxRawDataReceiver().addListener(PacketLogger.getInstance());

device.getRxRawDataReceiver().addListener(PacketAnalysis.getInstance());

Thread thread = new Thread(device.getDeviceCommunication());

thread.setName(device.getDeviceInfo().getFriendlyNameWithId());

thread.setDaemon(true);

thread.start();

}

else {

ApplicationLogger.LOGGER.info(deviceInfoClone.getName() +

" on " +

deviceInfoClone.getPortName() +

" ignored.");

}

}

}

}

1. Лістинг класу Device

package com.rasalhague.mdrv.device.core;

import com.rasalhague.mdrv.dev\_communication.DeviceCommunication;

import com.rasalhague.mdrv.dev\_communication.RxRawDataReceiver;

import com.rasalhague.mdrv.device.DeviceHistory;

import com.rasalhague.mdrv.logging.ApplicationLogger;

import org.reflections.Reflections;

import java.util.Set;

public abstract class Device implements DeviceTemplateI {

private static final String FRIENDLY\_NAME\_FIELD\_NAME = "FRIENDLY\_NAME";

private static final String VENDOR\_ID\_FIELD\_NAME = "VENDOR\_ID";

private static final String PRODUCT\_ID\_FIELD\_NAME = "PRODUCT\_ID";

private static final String CHANNEL\_SPACING\_FIELD\_NAME = "CHANNEL\_SPACING";

private static final String END\_PACKET\_SEQUENCE\_FIELD\_NAME = "END\_PACKET\_SEQUENCE";

private static final String INITIAL\_FREQUENCY\_FIELD\_NAME = "INITIAL\_FREQUENCY";

private static final String MANUAL\_DEVICE\_CONTROL\_FIELD\_NAME = "MANUAL\_DEVICE\_CONTROL";

private static final String REFLECTION\_INIT\_PATH = "com.rasalhague.mdrv.device";

private static DeviceHistory deviceHistory = new DeviceHistory();

protected DeviceCommunication deviceCommunication;

protected DeviceInfo deviceInfo;

public static Device getConcreteDevice(DeviceInfo deviceInfo) {

Reflections reflections = new Reflections(REFLECTION\_INIT\_PATH);

Set<Class<? extends Device>> devicesClassSet = reflections.getSubTypesOf(Device.class);

for (Class<? extends Device> concreteDeviceClass : devicesClassSet) {

try {

String vendorId = (String) concreteDeviceClass.getField(VENDOR\_ID\_FIELD\_NAME).get(null);

String productId = (String) concreteDeviceClass.getField(PRODUCT\_ID\_FIELD\_NAME).get(null);

String friendlyName = (String) concreteDeviceClass.getField(FRIENDLY\_NAME\_FIELD\_NAME).get(null);

float channelSpacing = concreteDeviceClass.getField(CHANNEL\_SPACING\_FIELD\_NAME).getFloat(null);

float initialFrequency = concreteDeviceClass.getField(INITIAL\_FREQUENCY\_FIELD\_NAME).getFloat(null);

byte[] endPacketSequence = (byte[]) concreteDeviceClass.getField(END\_PACKET\_SEQUENCE\_FIELD\_NAME).get(null);

boolean useCustomReadMethod = (boolean) concreteDeviceClass.getField(MANUAL\_DEVICE\_CONTROL\_FIELD\_NAME)

.get(null);

if (deviceInfo.getProductID().equals(productId) && deviceInfo.getVendorID().equals(vendorId)) {

Device device = concreteDeviceClass.newInstance();

deviceInfo.setSomeFields(friendlyName, endPacketSequence, initialFrequency, channelSpacing, device);

deviceInfo.setManualDeviceControl(useCustomReadMethod);

device.initializeObject(deviceInfo);

return device;

}

}

catch (IllegalAccessException | NoSuchFieldException | InstantiationException e) {

ApplicationLogger.LOGGER.severe(e.getMessage());

e.printStackTrace();

}

}

return null;

}

public void initializeObject(DeviceInfo deviceInfo) {

this.deviceInfo = deviceInfo;

this.deviceCommunication = DeviceCommunication.getInstance(deviceInfo);

deviceHistory.checkForCollision(this.deviceInfo);

}

@Override public int hashCode() {

return deviceInfo.hashCode();

}

@Override public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Device device = (Device) o;

if (!deviceInfo.equals(device.deviceInfo)) return false;

return true;

}

public DeviceCommunication getDeviceCommunication() {

return deviceCommunication;

}

public RxRawDataReceiver getRxRawDataReceiver() {

return deviceCommunication.getRxRawDataReceiver();

}

public DeviceInfo getDeviceInfo() {

return deviceInfo;

}

}

1. Лістинг класу DeviceTemplate

package com.rasalhague.mdrv.device.devices;

import com.rasalhague.mdrv.device.core.Device;

import java.util.ArrayList;

/\*\*

\* The Device class template. Use it for add new device support.

\*/

public class Dummy extends Device {

/\*\*

\* Use in GUI Labels for identify device.

\*/

public final static String FRIENDLY\_NAME = "Dummy";

/\*\*

\* Must be upper case in 16 base. F.e. "1FFB".

\*/

public final static String VENDOR\_ID = "1111";

/\*\*

\* Must be upper case in 16 base.

\*/

public final static String PRODUCT\_ID = "1111";

/\*\*

\* Minimal frequency that device can see. F.e. 2400f.

\*/

public final static float INITIAL\_FREQUENCY = 2400f;

/\*\*

\* Device channel spacing. F.e. 327.450980f.

\*/

public final static float CHANNEL\_SPACING = 500f;

/\*\*

\* Byte or sequence of byte that identify end of packet. "Packet" means RSSI set from INITIAL\_FREQUENCY to end of

\* device vision.

\*/

public final static byte[] END\_PACKET\_SEQUENCE = new byte[]{10};

/\*\*

\* Set this to TRUE only if you want to control device manually In this case the program will not try to open device

\* and read from customReadMethod() becomes active For example check MetaGeekWiSpyGen1.java class file Works only for

\* HIDUSB devices

\*/

public final static boolean MANUAL\_DEVICE\_CONTROL = false;

/\*\*

\* Use this method for initialize your device. If device does not need initialization - leave this blank

\*/

@Override public void initializeDevice() {

}

/\*\*

\* Use this method for parse data which you device out. Return format - Byte array. Every item - RSSI in format

\* "-100".

\*/

@Override public ArrayList<Byte> parse(ArrayList<Byte> dataToParse) {

ArrayList<Byte> finalArray = new ArrayList<>(dataToParse);

finalArray.remove(0);

return finalArray;

}

/\*\*

\* Use this method for override default HIDUSB / COM read behavior. In most cases its usable for HIDUSB devices, when

\* default com.codeminders.hidapi library read method does not work.

\* <p>

\* !!! IMPORTANT !!! If you want to use this method you need to set MANUAL\_DEVICE\_CONTROL field to TRUE

\*/

@Override public byte[] customReadMethod() {

return new byte[0];

}

}

1. Лістинг класу DeviceCommunication

package com.rasalhague.mdrv.dev\_communication;

import com.codeminders.hidapi.HIDDevice;

import com.rasalhague.mdrv.device.core.DeviceInfo;

import com.rasalhague.mdrv.logging.ApplicationLogger;

import jssc.SerialPort;

/\*\*

\* Result of the run method must be call to RxRawDataReceiver

\*/

public abstract class DeviceCommunication implements Runnable {

final DeviceInfo deviceInfo;

final RxRawDataReceiver rxRawDataReceiver;

public HIDDevice hidDevice;

public SerialPort serialPort;

/\*\*

\* Instantiates a new Device communication.

\*

\* @param devInfo

\* the dev info

\*/

protected DeviceCommunication(DeviceInfo devInfo) {

deviceInfo = devInfo;

rxRawDataReceiver = new RxRawDataReceiver(deviceInfo);

}

/\*\*

\* Factory method. Choose device.

\*

\* @param deviceInfo

\* the device info

\*

\* @return the instance

\*/

public static DeviceCommunication getInstance(DeviceInfo deviceInfo) {

if (deviceInfo.getDeviceType() == DeviceInfo.DeviceType.COM) {

return new COMDeviceCommunication(deviceInfo);

}

if (deviceInfo.getDeviceType() == DeviceInfo.DeviceType.HID) {

return new HIDDeviceCommunication(deviceInfo);

}

if (deviceInfo.getDeviceType() == DeviceInfo.DeviceType.DUMMY) {

return new DummyDeviceCommunication(deviceInfo);

}

return null;

}

/\*\*

\* Gets rx raw data receiver.

\*

\* @return the rx raw data receiver

\*/

public RxRawDataReceiver getRxRawDataReceiver() {

return rxRawDataReceiver;

}

/\*\*

\* Initialize device.

\*/

void initializeDevice() {

deviceInfo.getDevice().initializeDevice();

ApplicationLogger.LOGGER.info(deviceInfo.getName() + " has initialized.");

}

}

1. Лістинг класу ApplicationLogger

package com.rasalhague.mdrv.logging;

import com.rasalhague.mdrv.Utility.Utils;

import java.io.File;

import java.io.IOException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.MissingResourceException;

import java.util.logging.\*;

public class ApplicationLogger extends Logger {

private final static String LOGGER\_NAME = "ApplicationLogger";

// public final static Logger GLOBAL\_LOGGER = Logger.getLogger(Logger.GLOBAL\_LOGGER\_NAME);

public final static Logger LOGGER = new ApplicationLogger();

public static Logger getLogger() {

return LOGGER;

}

/\*\*

\* Protected method to construct a logger for a named subsystem.

\* <p>

\* The logger will be initially configured with a null Level and with useParentHandlers set to true.

\*

\* @throws MissingResourceException

\* if the resourceBundleName is non-null and no corresponding resource can be found.

\*/

private ApplicationLogger() {

super(ApplicationLogger.LOGGER\_NAME, null);

}

static public void setup() {

LOGGER.setLevel(Level.ALL);

try {

String fileName = "logs" + File.separator + Utils.addTimeStampToFileName("Application");

Utils.createFile(fileName);

//choose file header to add

FileHandler fileTxt = new FileHandler(fileName);

LOGGER.addHandler(fileTxt);

ConsoleHandler consoleHandler = new ConsoleHandler();

LOGGER.addHandler(consoleHandler);

setFormatterToLoggerHandlers(LOGGER, new MyLogFormatter());

LOGGER.info("Logger has initialized");

}

catch (SecurityException e) {

LOGGER.log(Level.SEVERE, "Cannot create file due to Security reason.", e);

}

catch (IOException e) {

LOGGER.log(Level.SEVERE, "Cannot create file due to IO error.", e);

}

// LogOutputStream.setup();

}

public synchronized static void addCustomHandler(Handler handler) {

LOGGER.addHandler(handler);

//update formatter

setFormatterToLoggerHandlers(LOGGER, new MyLogFormatter());

}

public synchronized static void closeHandlers() {

Handler[] handlers = LOGGER.getHandlers();

for (Handler handler : handlers) {

handler.close();

}

}

private static void setFormatterToLoggerHandlers(Logger logger, Formatter formatter) {

//For logger handlers

Handler[] handlers = logger.getHandlers();

for (Handler handler : handlers) {

handler.setFormatter(formatter);

}

//For logger parent handlers

Logger loggerParent = logger.getParent();

if (loggerParent != null) {

Handler[] loggerParentHandlers = loggerParent.getHandlers();

for (Handler handler : loggerParentHandlers) {

handler.setFormatter(formatter);

}

}

}

}

class MyLogFormatter extends Formatter {

// private final static String LOGGER\_NAME = "ApplicationLogger";

private static final SimpleDateFormat DATE\_FORMAT = new SimpleDateFormat("dd.MM.yy HH:mm:ss.SSS");

@Override public String format(LogRecord record) {

// LocationInfo locationInfo = new LocationInfo(new Throwable(), LOGGER\_NAME);

StringBuilder builder = new StringBuilder();

// String bracerOpen = "[";

// String bracerClose = "]";

// String dot = ".";

String separator = " ";

// String packageName = this.getClass().getPackage().getName();

String packageName = "com.rasalhague.mdrv";

builder.append(DATE\_FORMAT.format(new Date(record.getMillis())));

builder.append(separator);

builder.append("[").append(record.getLevel()).append("]");

builder.append(separator);

builder.append("[").append(record.getSourceClassName().replace(packageName + ".", "")).append(".")

.append(record.getSourceMethodName()).append("]");

builder.append(separator);

builder.append(formatMessage(record));

builder.append("\n");

return builder.toString();

}

}

1. Приклад обміну…