Nordic Semiconductor Sniffer API Guide

Version 0.5

The Sniffer API guide provides the documentation of the Python API used to interface with the nRF Sniffer for Bluetooth low energy. The nRF Sniffer is available for download at nordicsemi.com.

Revision History

Revision	**Changes**
0.1	Initial version
0.2	Added description of LED and GPIO.
0.3	Updated documentation to reflect API changes after 0.9.7
0.4	Updated documentation for version 2.0.0
0.5	Updated documentation for version 3.0.0

Introduction

The Sniffer API is a Python API that allows scripted use of the Nordic Semiconductor BLE Sniffer. It allows discovery of devices and sniffing of a single device. It provides access to all the BLE packets received by the sniffer and the devices discovered.

The sniffer consists of four parts as seen in Figure 1. Code that uses the Sniffer API directly will effectively replace the “Sniffer extcap” and “Wireshark” components.

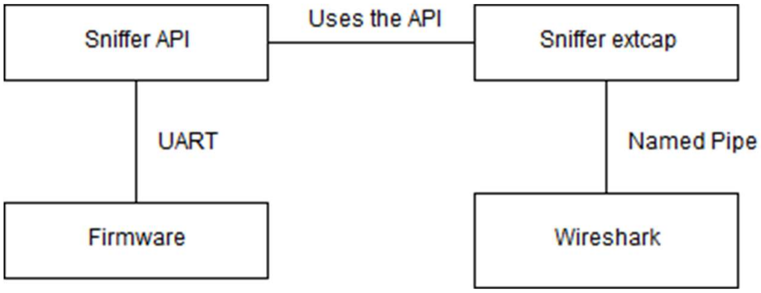


Figure 1 - The parts of the sniffer.

The “Sniffer extcap” is included under extcap/nrf_sniffer.py (and the helper script extcap/nrf_sniffer.bat for Windows). See https://www.wireshark.org/docs/wsdg_html_chunked/ChCaptureExtcap.html for more information on how the extcap system works. Note: you do **not** need to use the extcap if you are using the Sniffer API directly.

Dependencies

The API has been developed using Python 3.7, but should be compatible with any Python runtime 3.5 and above. It is however **not** compatible with Python 2. The API also requires one third party Python library:

1. Pyserial (cross platform) version 3.4 or higher. Download using pip: “pip install pyserial>=3.4” or use the included “requirements.txt”.

See the Sniffer User Guide for additional information.

Using the Sniffer API

Getting Started

1. Install dependencies.
2. Include the SnifferAPI folder in your Python project.
3. Import the API with
   ```
   from SnifferAPI import Sniffer
   ```
4. Optional: discover a list of connected sniffers using:
   ```
   from SnifferAPI import UART
   ports = UART.find_sniffer()
   ```
5. Instantiate the Sniffer class with e.g
   ```
         # For firmware < 2.0.0, use baudrate=460800 or omit
   baudrate
         mySniffer = Sniffer(portnum=”COM40”, baudrate=1000000)
   ```

6. Start the Sniffer with

```
mySniffer.start()
```

`example.py` is an example program with explanations in the comments.

Overview

The API consists of 5 classes in 3 files: The Sniffer class in Sniffer.py, the DeviceList and Device classes in Devices.py, and the Packet and BlePacket classes in Packet.py. The exceptions in Exceptions.py are also part of the API. The entry point for the API is the Sniffer class (retrieve packets and devices through the methods in Sniffer). The last pages of this document (and also the documentation.html file) contain a complete documentation of the API.

An overview of the levels below the Sniffer module

Object/Module hierarchy

During normal operation, the Sniffer object interfaces only to the SnifferCollector object which acts as a hub for the flow of packets. The SnifferCollector object reads packets from UART through its PacketReader object and sends packets over named pipe to Wireshark. It also stores all packets in a capture (.pcap) file through its CaptureFileHandler object, and keeps an internal buffer of packets. In addition, the SnifferCollector object keeps a list of devices which are advertising in the vicinity.

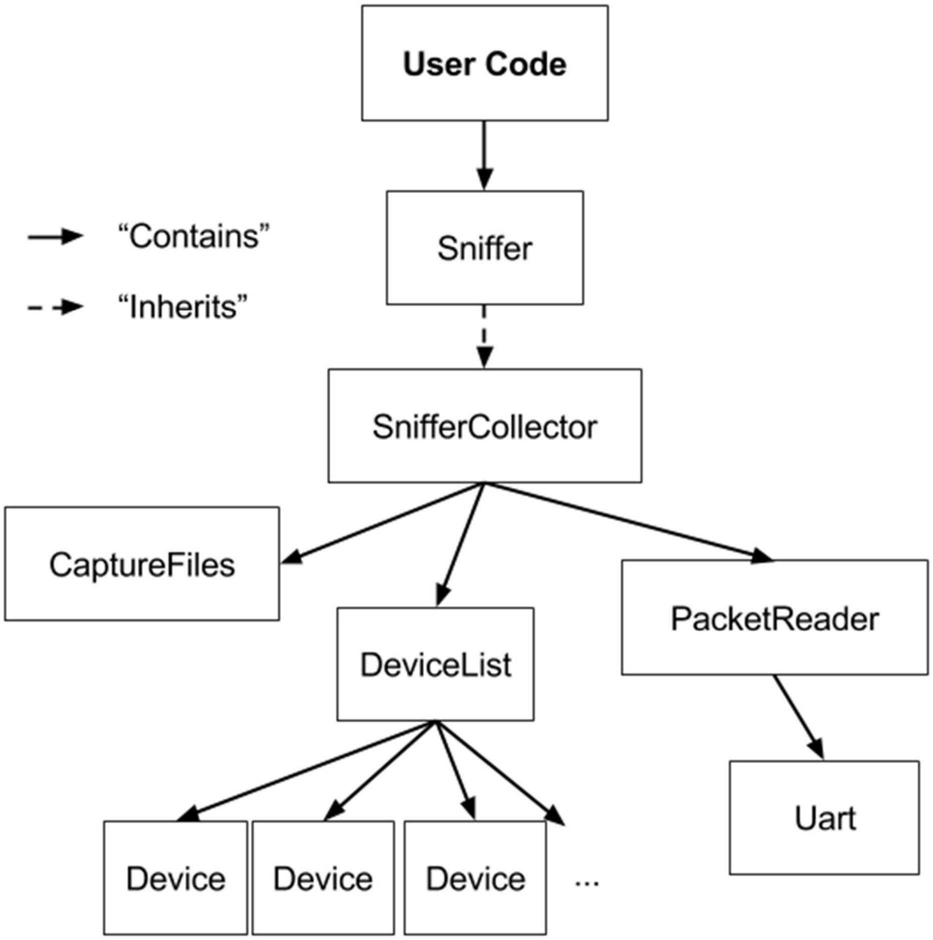


Figure 2- Object hierarchy behind the Sniffer API

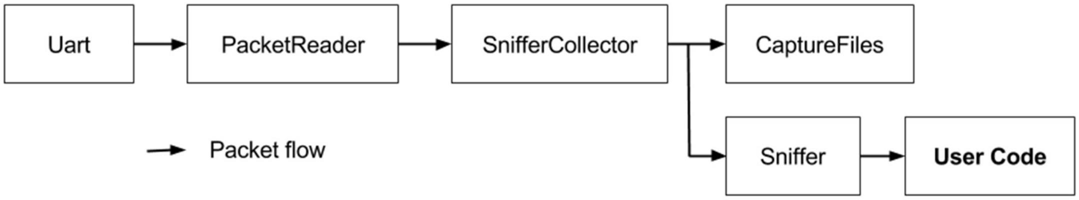


Figure 3- The flow of packets through the API.

Note: Command packet flow from the SnifferCollector to the UART is not represented in the above diagram.

Threads of operation

The Sniffer system contains 3 separate threads which are running in addition to the main context (user thread). They are:

1. The LogFlusher thread which regularly flushes the log to file.
2. The Uart thread which is used to read data from the sniffer's serial port.
3. The Sniffer thread. This is the main thread which uses the data read from the serial port to parse packets and distribute them to the User Code and the rest of the sniffer system.

OS specific code

The API should not contain any OS specific code. The modules that previously had OS specific code have been removed in a previous version of the API.

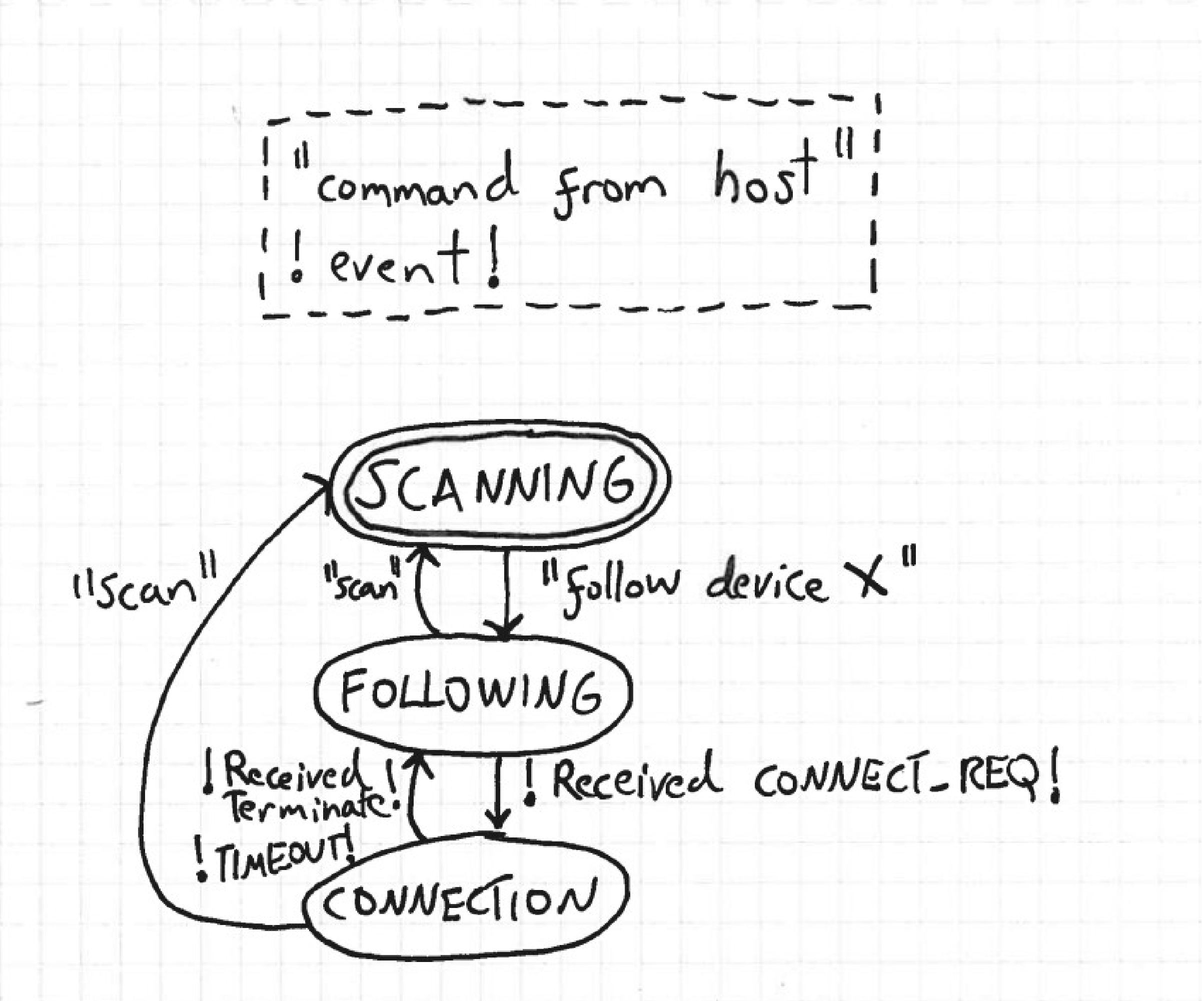
Establishing a connection between the API and the firmware

As explained below, the firmware sends received ADV packets in the SCANNING state. The Sniffer constructor can take the port number and the baudrate of the firmware as arguments. The UART.py file contains a helper function find_sniffer which lets you find the connected sniffers if you do not know their port names. Use a baudrate of 1 000 000 for sniffers of version 2.0.0 or greater, otherwise use a baudrate of 460 800.

Appendices

1. State change description

Firmware States (simplified)



SCANNING (Initial state):

- Scans advertiser packets.

State change: If the sniffer received a "follow device X" (REQ_FOLLOW) command, it will go to the FOLLOWING state.

FOLLOWING:

- Only packets from device X will be received.
- All packets sent by device X will be received.
- All SCAN_REQ packets directed to device X and corresponding SCAN_RSP packets will be picked up.
- All CONNECT_REQ packets directed to device X will also be picked up.

State change:

- If the sniffer receives a CONNECT_REQ packet, it will go to the CONNECTION state.
- If the sniffer received a "scan" command, it will go to the SCANNING state.

CONNECTION:

- The sniffer will follow the connection.
- All packets in the connection will be received.

State change:

- If a timeout occurs (no packets received for a length of time defined by the negotiated connection supervisor timeout, typically 4 seconds) the sniffer will go to the FOLLOWING state.
- If one of the devices in the connection terminate the connection the sniffer will go to the FOLLOWING state.
- If the sniffer received a “scan” (REQ_SCAN_CONT) command, it will go to the SCANNING state.

LED Configuration

State	LED1	LED2
SCANNING	Toggle when packet received	OFF
FOLLOWING	Toggle when packet received	OFF
CONNECTION	Toggle when packet received	ON