Nordic Semiconductor Sniffer API Guide

Version 0.2

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 from mypage at nordicsemi.com on purchase of the nRF51822, nRF51422 and nRF8001 development kits. The Python API documented is currently available only for Windows. The intent of this document is to support the porting of the Sniffer API to non-windows platforms like OS X and Linux.

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

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 three parts as seen in Figure 1, where the API replaces the console app as the controller and hub of communication.

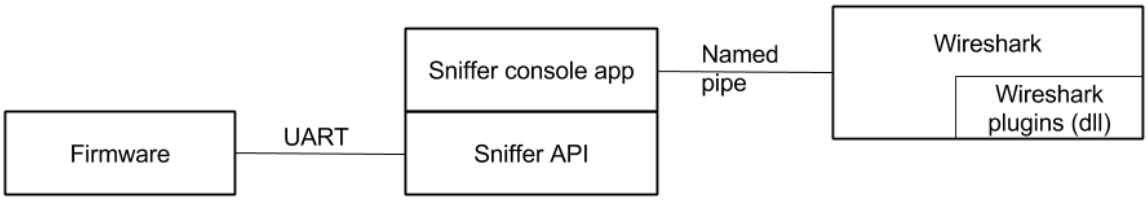


Figure 1 - The parts of the sniffer.

The Wireshark plugin code is included in the API.

Dependencies

The API has been developed using Python 2.7.6 32 bit. 64 bit is untested but might work. The API also requires one third party Python library:

1. Pyserial (cross platform) version 2.7. Get the installer if you are on Windows.
   http://pyserial.sourceforge.net

In addition, you must get nRF Sniffer version 0.9.7, and make sure it connects to the firmware.

See the Sniffer User Guide included with the nRF Sniffer for more 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. Instantiate the Sniffer class with e.g.
   ```
   mySniffer = Sniffer()
   ```
5. 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.

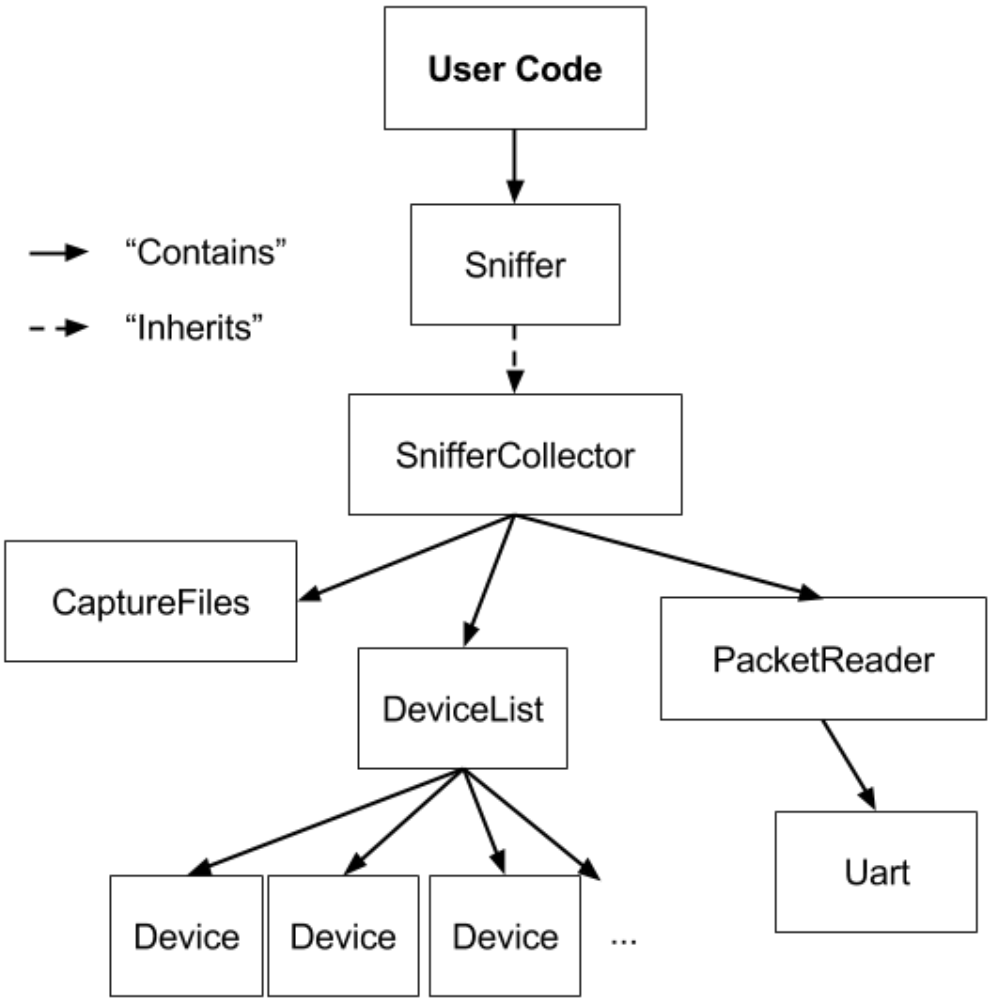


Figure 2- Object hierarchy behind the Sniffer API

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 Pipe thread which is used to connect the named pipe dynamically.
2. The LogFlusher thread which regularly flushes the log to file.
3. The Sniffer thread. This is the main thread which handles everything else, including the flow of packets described above.

OS specific code

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

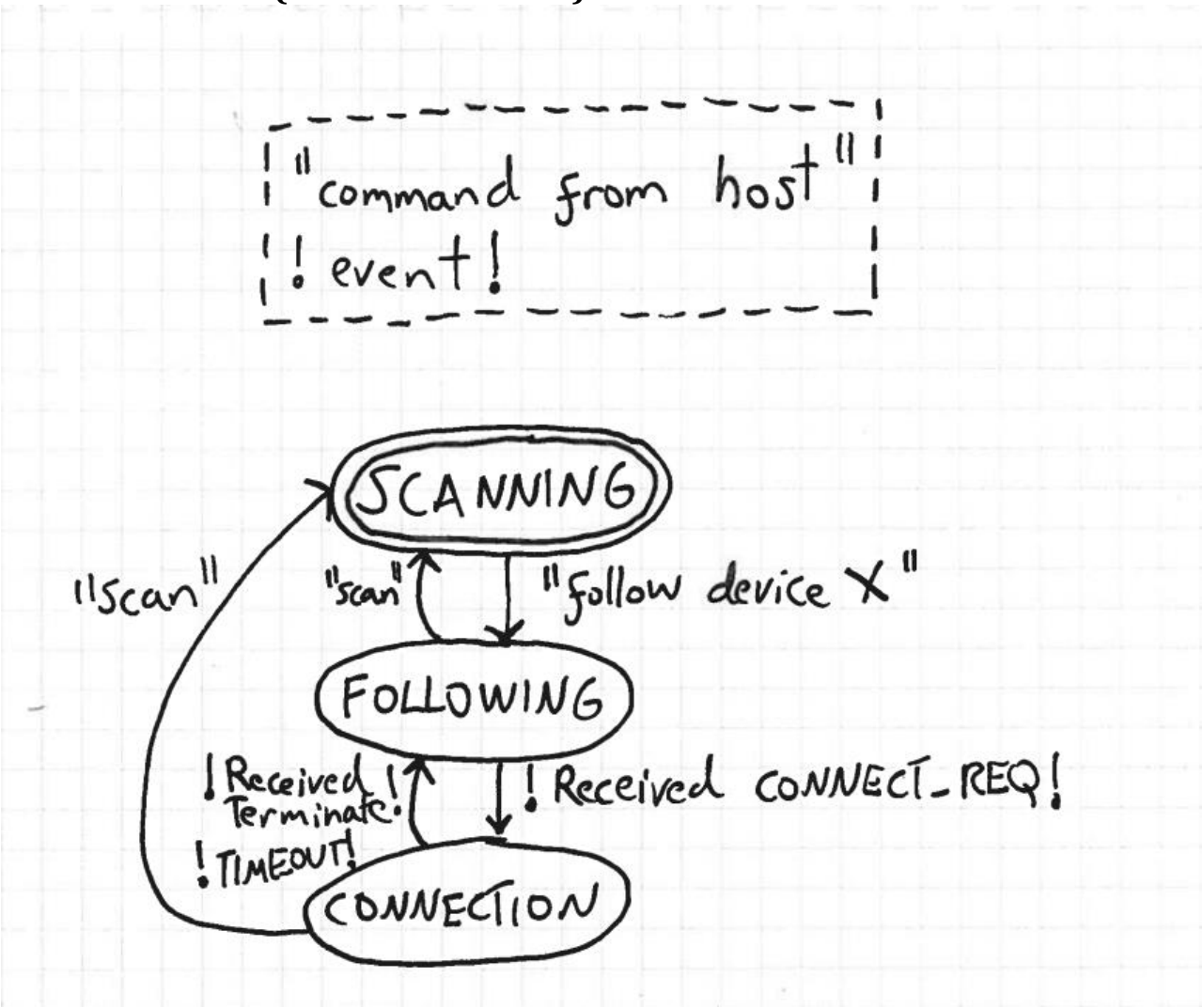
Establishing a connection between the API and the firmware

As explained below, the firmware sends PING_RSP packets in the SCANNING state. The Sniffer constructor can take the port number of the firmware as an argument. In this case, the API connects blindly to it. If no port is provided, the API opens all COM ports on the computer in succession and listens for PING_RSP packets to locate the correct port. When the PING_RSP packet is not received on a COM port, it closes the COM port.

Appendices

1. State change description
2. API documentation (also in documentation.html)
3. Description of UART protocol (also in sniffer_uart_protocol.xlsx)

Firmware States (nRF Sniffer v0.9.6)



SCANNING (Initial state):

- Scans advertiser packets.
- The sniffer will send a PING_RSP each 75ms to the host.

State change: If the sniffer received a "follow device X" 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 about 30 seconds) the sniffer will go to the FOLLOWING state.
- If one of the devices in the connection terminating the connection the sniffer will go to the FOLLOWING state.
- If the sniffer received a "scan" command, it will go to the SCANNING state.

LED Configuration (only valid for PCA10001)

State	LED0	LED1
SCANNING	OFF	Toggle when packet received
FOLLOWING	Toggle when packet received	OFF
CONNECTION	ON	Toggle when packet received

GPIO Behavior (only valid for PCA10001)

PIN	LOW – HIGH	HIGH - LOW
4	Finished receiving advertise packet from device being followed.	Enable RX for receiving CONNECT_REQ to followed device.
5	Start radio for receiving anchor point of connection event, ramp up required	Finished receiving ADDRESS bytes of anchor point in connection event.