NORDIC SEMICONDUCTOR

Getting started with

BLUETOOTH LOW ENERGY DEVELOPMENT

TABLE OF CONTENTS

INTRODUCTION

Making a product with Bluetooth low energy technology has become easy. Everything you need from the chip to the protocol stack and examples of use is provided by the chip vendor. And even if the technology is advanced, it is easy to use through standardized profiles and API's. This document will highlight what Nordic Semiconductor has to offer when you are about to start your Bluetooth low energy product development and some basics about Bluetooth low energy.

WHAT NORDIC SEMICONDUCTOR HAS TO OFFER

In this chapter we will show you everything you need to get started developing a product based on one of the ICs in the nRF52 Series.

We have a Software Development Kit (SDK) and the precompiled Bluetooth low energy protocol stack, called a SoftDevice, with API documentation and the rest of our software documentation that covers the entire nRF52 Series software development process.

When you start designing your hardware, it is essential that you follow our PCB layout guidelines that are documented in the nRF52832 datasheet to achieve the best RF performance and best chance to pass regulatory qualification tests.

Let us look at some of the tools Nordic Semiconductor has to offer.

HARDWARE

The nRF52 Development Kit is an ideal starting point for your development process, allowing you to quickly set up the software toolchain and develop, debug and test your prototype.

Our development tools are designed to make your job of getting a design to production easier. The basis for our development environments is the development kit with its wide range of functionality.

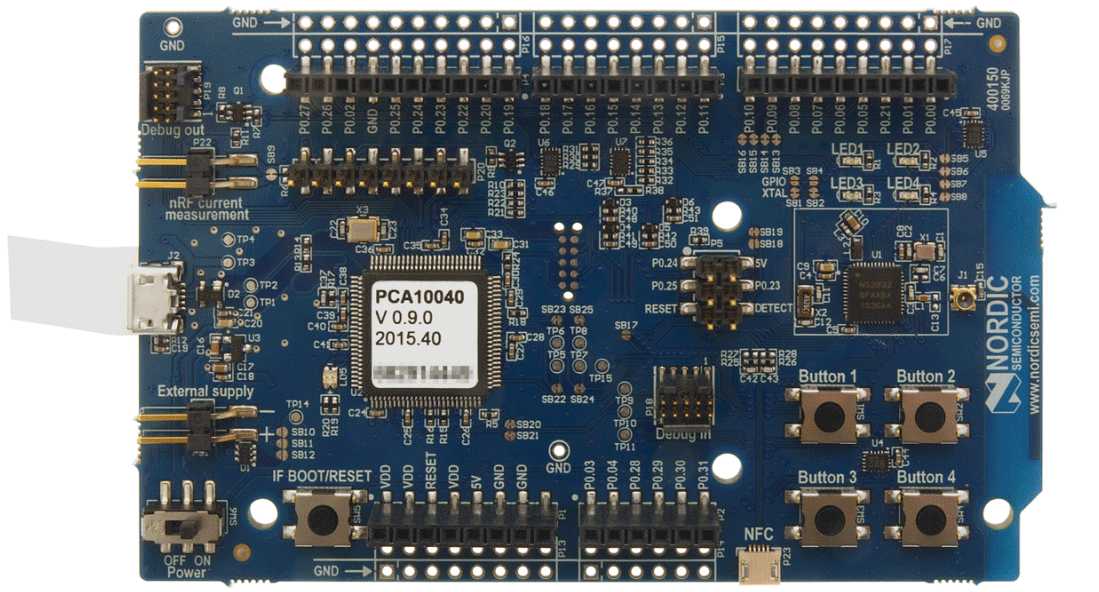
The best setup is to use the development kit together with a computer and Bluetooth low energy dongle and the Master Control Panel as the peer device.

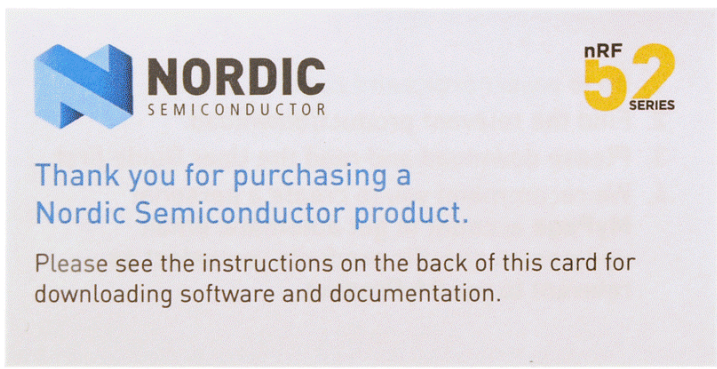
In addition to hardware, the nRF52 Development Kit consists of firmware source code, documentation, hardware schematics, and layout files.

The key features of the development kit are:

- nRF52832 flash-based Bluetooth® low energy SoC solution
- Buttons and LEDs for user interaction
- I/O interface for Arduino form factor plug-in modules
- SEGGER J-Link OB Debugger with debug out functionality
- Virtual COM Port interface via UART
- Drag-and-drop Mass Storage Device (MSD) programming
- Supporting NFC-A listen mode support

For access to firmware source code, hardware schematics, and layout files, see www.nordicsemi.com.





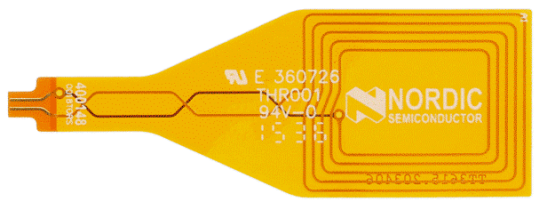


Figure 1. 1 x nRF52 Development Kit board (PCA10040) and 1 x NFC adhesive tag

The Arduino form factor interface enables you to take advantage of the large number of Arduino shields available.

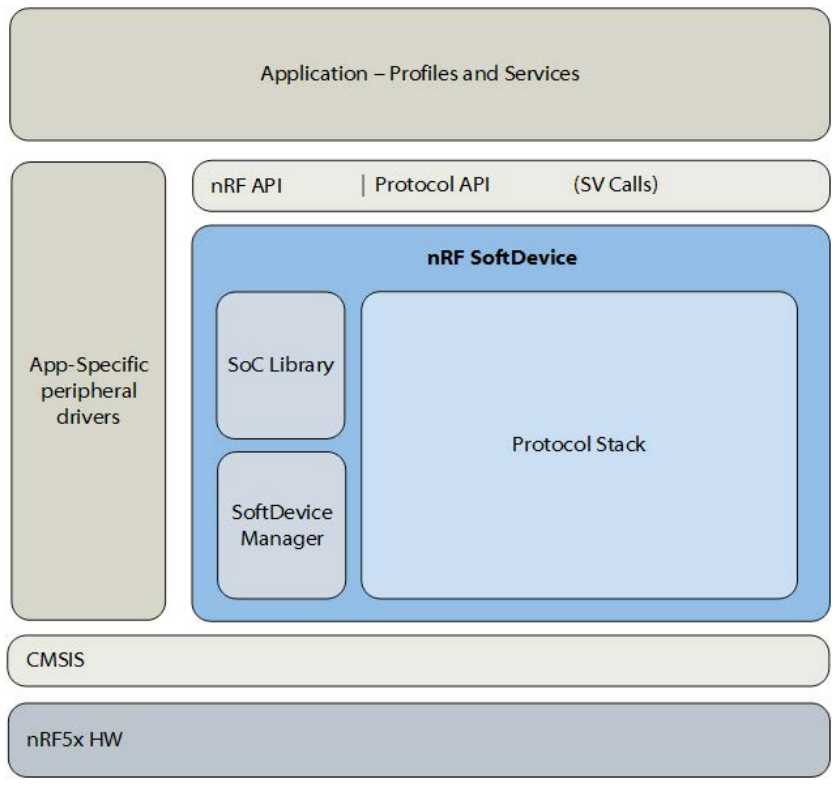
SOFTWARE

SOFTDEVICE

A SoftDevice is a pre-compiled and pre-linked binary software library implementing a wireless protocol developed by Nordic Semiconductor.

Since the SoftDevice is pre-compiled and pre-linked there is no compile time dependency between the SoftDevice and the application, and the two are developed completely independently. This speeds up development of the application, and allows the SoftDevice to be tested separately.

The unique hardware and software supported framework, in which it executes, provides run-time isolation and determinism in its behavior. These characteristics make the interface comparable to that of a hardware device, providing clear separation between the application and the protocol stack. The SoftDevice Application Program Interface (API) is available to applications as a high-level programming language interface, provided as a standard C header file.



The SoftDevice and application firmware can be updated in the field by Device Firmware update (DFU.) This is a feature that enables you to extend your product lifetime by adding new features even after the product has been put to use by the end customer.

SOFTWARE DEVELOPMENT KIT

Software development on the nRF52 series is mainly supported through our Software Development Kit (SDK) which contains examples and libraries utilizing the different features in and wireless protocols supported by the nRF52832. However, we have an extensive range of supporting software tools to help you with testing and programming on your chip.

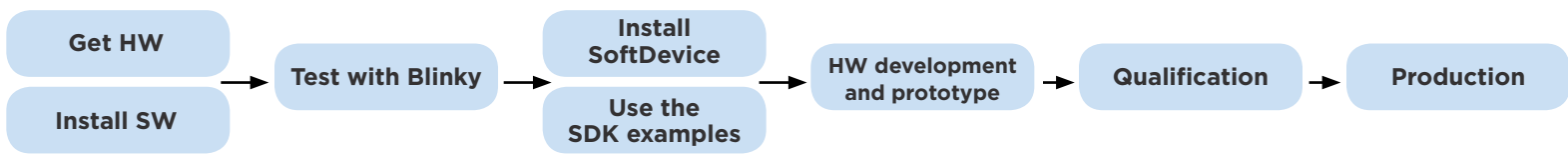
- **nRF52 SDK:** The nRF52 Software Development Kit (SDK) provides source code of examples and libraries forming the base of your application development.
- **nRF5x tools:** nRF5x Tools is a package that contains JLinkARM, JLink CDC, nRFjprog, and mergehex. nRFjprog is a command line tool for programming nRF5x series chips. It is also useful in a production setup. nRF5x Tools will be installed together with nRFgo Studio.
- **Master Control Panel:** The Master Control Panel application is a software tool that is used to act as a Bluetooth® low energy peer device.
    - **Master Control Panel:** (64-bit version) (32-bit version) Master Control Panel for Windows is a software tool that is used with the nRF51 Dongle (PCA10031) to act as a Bluetooth low energy peer device. You can test your application's wireless connection with this tool. The Master Control Panel supports programming of SEGGER J-Link based nRF52 devices. For more information, see the help files in the Master Control Panel folder (default directory:" C:\Program Files (x86)\Nordic Semiconductor\Master Control Panel").
    - **nRF Master Control Panel for Android 4.3 or later:** nRF Master Control Panel for Android 4.3 or later is a powerful generic tool that allows you to scan and explore your Bluetooth Smart devices and communicate with them on an Android phone. MCP supports a number of Bluetooth SIG adopted profiles together with the Device Firmware Update (DFU) profile from Nordic Semiconductor.
- **nRFgo Studio:** This is a tool for programming and configuring devices. It supports the programming of SoftDevices, applications, and bootloaders, and can be used to run TX and RX tests on the chip to test the RF performance.
- **S132 SoftDevice:** Bluetooth Smart concurrent multi-link protocol stack solution supporting simultaneous Central/ Peripheral/Broadcaster/Observer role connections. For more information, see the S132 SoftDevice Specification and the nRF5 SDK.

We also recommend some third party software tools that are useful when developing with our products:

- **Keil MDK-ARM Development Kit**: Keil MDK-ARM Development Kit is a development environment specifically designed for microcontroller applications that lets you develop using the nRF52 SDK application and example files.
- **SEGGER J-Link Software:** The J-Link software is required to debug using the J-Link hardware that is packaged with our development kits.

HOW TO GET STARTED

The figure below shows the flow we recommend our customers to follow during their development.



- Get HW. This links to a web page that shows sources where you can buy the nRF52 DK online.
- Install SW. Shows a list of all our Software tools and where to get them.
- Test with Blinky. Shows you how to run an example on the nRF52 DK to ensure that both the toolchain and hardware is set up properly.
- Use the SDK examples. Shows all the examples we have available for the nRF5 series.
- Install SoftDevice: This is done by using the nRFgo Studio tool and the procedure described in the nRFgo Studio help file.
- HW development and prototyping: We provide a reference layout for the nRF52 device. This must be followed in order to achieve the best RF performance and to pass regulatory qualification tests. Nordic Semiconductor offers layout reviews and prototype RF tuning as part of our technical support service.
- Qualification and production: Both testing for qualification and production testing can be done by using Direct Test Mode (DTM.) The linked application note shows how you do this in production. This procedure also include FCC/ETSI (spurious testing).

If you have any questions regarding development with our products, please visit our online development community, DevZone, or the support ticket system at www.nordicsemi.no.

BLUETOOTH LOW ENERGY BASICS

When you have become familiar with the development tools Nordic Semiconductor has to offer, you might want to dig a bit more into the Bluetooth low energy technology. This chapter covers the basics.

We will try to provide the link between what we describe here and where in the Bluetooth Core Spec v4.2 it is defined. References to the core spec uses the following format: Section x.x Part x Vol x .

Bluetooth Smart defines 4 GAP roles: Broadcaster, Observer, Central, Peripheral [Section 6.2 Vol 1 Part A] and 5 Link layer states: Standby, Advertising, Scanning, Initiating, Connection [Section 1.1 Vol 6 Part B]. One device may have one or multiple roles, working in one or multiple states at the same time.

THEORY ABOUT ADVERTISING

I. What is advertising

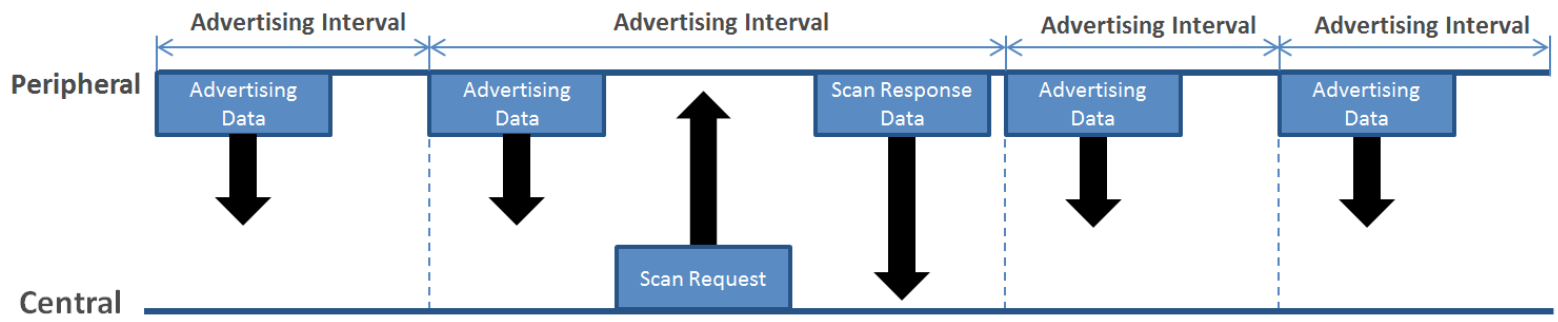
Advertising is the act of broadcasting data. The purpose of this is for device discovery and data publishing. There are 2 types of data packets that can be transmitted, Advertising packet and Scan Response packet, **each can have up to 31 bytes payload**. The advertiser address is included in the broadcast data in addition to the payload.

The advertiser constantly broadcasts the advertising packets with an advertising interval. Advertising interval can be changed on the fly. There is a requirement for minimum and maximum advertising interval.

For normal, undirected advertising, the advertising intervals ranges from **20ms to 10.24s** (Section 7.8.5 Part E Vol2). We will go into detail on different types of advertising in next section.

The scan response packet is transmitted by the advertiser when it receives a scan request from a scanner. The advertiser has to enter a RX period to wait for the scan request. This RX period can be used to receive Connect Request as well. But we will not discuss about connection in this document.

The following figure describe how the advertising packet and scan response packet is sent.

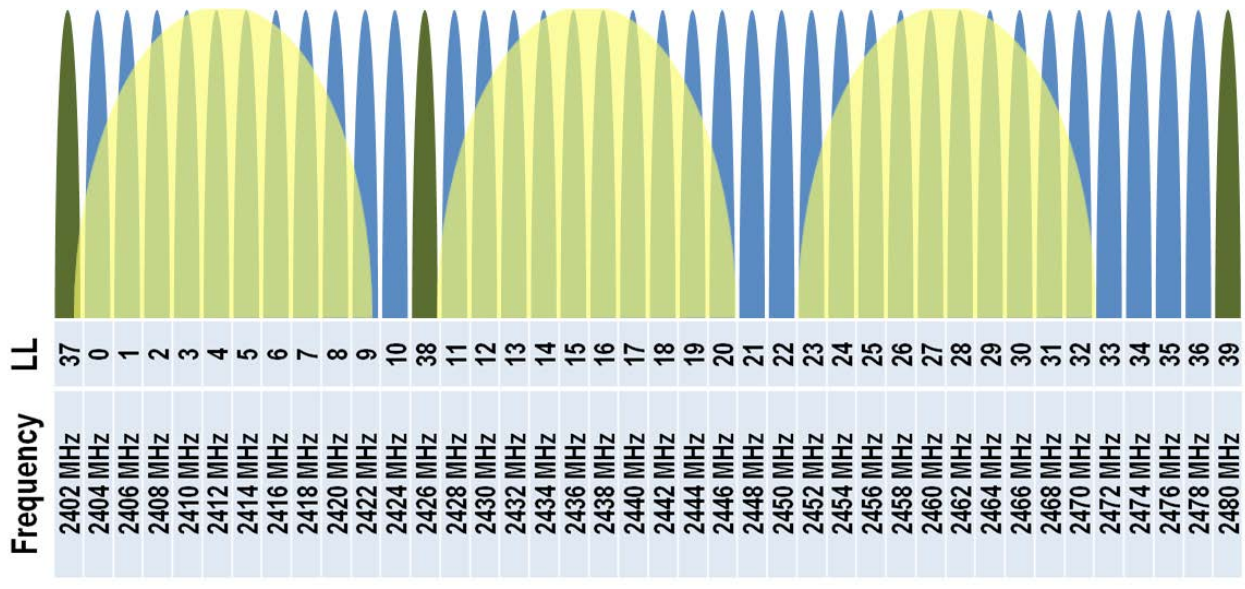


In some applications when we do not expect a connection and do not have extra data in scan response packet, we can advertise in non-connectable mode and can skip the RX period to save power. The beacon application is one of the use cases.

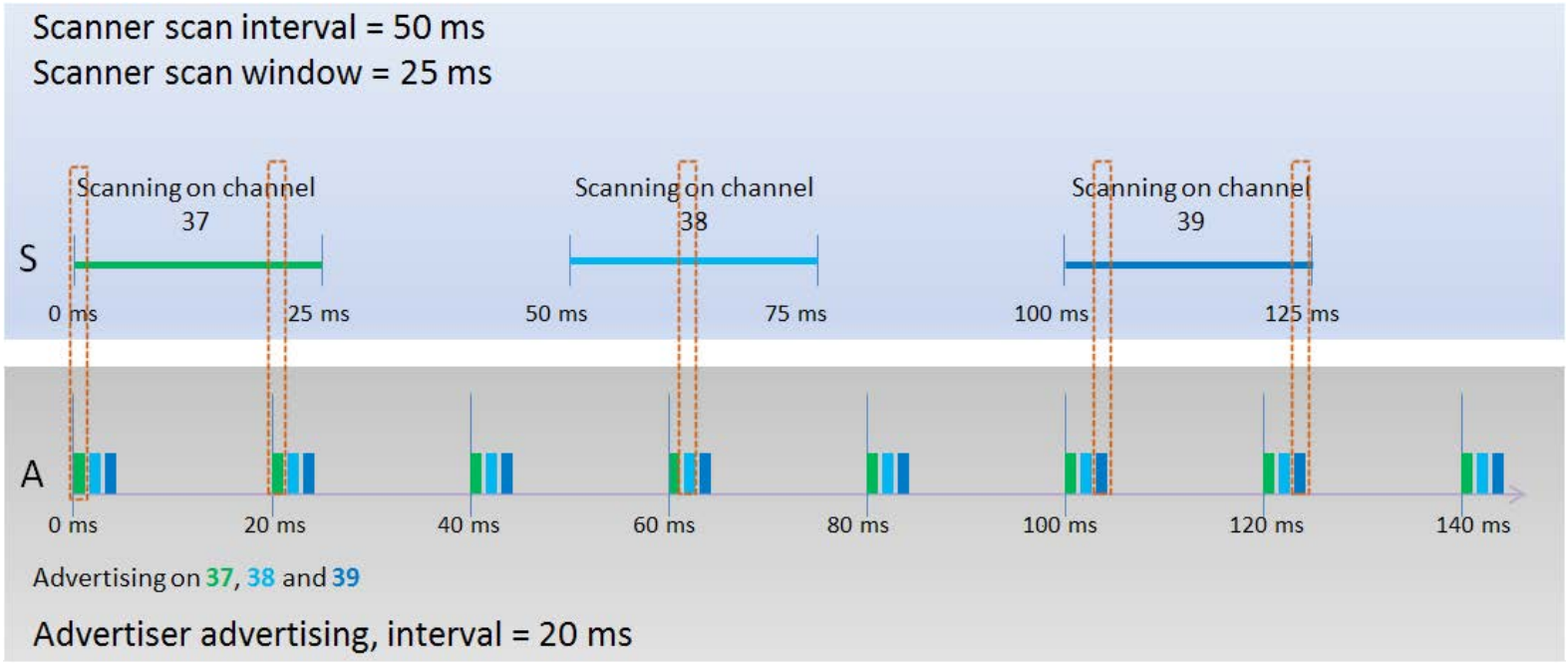
Note that all Advertising packet, Scan Request packet, Scan Response packet share the same on-air Access Address = **0x8E89BED6**. This common address allows any device to scan and receive advertising/scan response data.

Bluetooth Smart uses 40 RF channels in the ISM band (2.4GHz). These RF channels have center frequencies 2402 + k*2MHz where k ranges from 0 to 39. Note that k is not the same as ”Channel Index”, or channel number (Section 1.4 Vol 6 Part B).

Three of them is dedicated for advertising which is channel 37 (2402MHz), 38 (2426MHz) and 39 (2480MHz). They were selected to avoid interference with the busy channels used by Wifi. This figure shows the 3 advertising channels, 37 data channels, and the curved shapes are wifi busy channels.

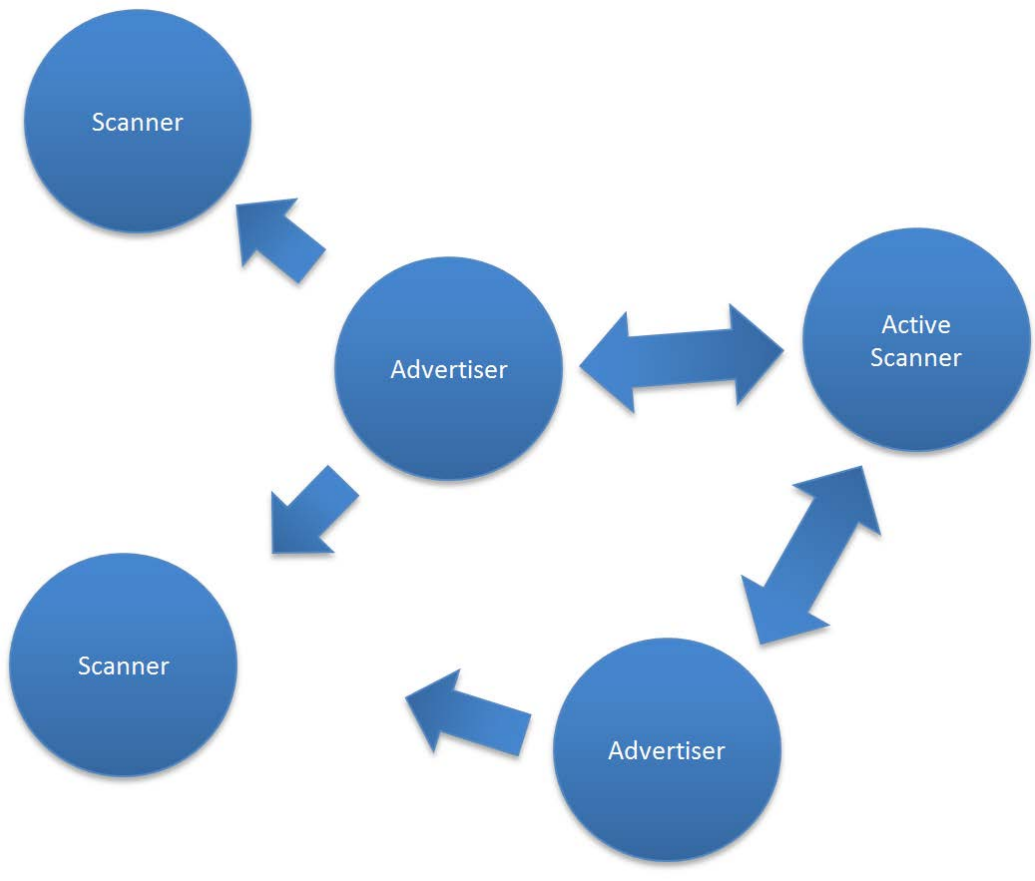


In the implementation of our stack, by default we transmit the advertising packet in all 3 channels on every advertising event, on channel 37, 38, 39 respectively. Next figure shows how we do that, and how the advertising packet on a channel is captured by the scanner, which scan on one of the three channel at a time.



II. Broadcast topology

When advertising, the network topology is Broadcast topology. There could be multiple advertisers and multiple scanner at the same time. It is a connection-less network:



Note that one device can do scanning and advertising simultaneously. And one can be in a connection with a central or peripheral and can do advertising at the same time.

The only packet the active scanner can send to the advertiser is the Scan Request packet, which contain only the scanner address. Passive scanner does not do Scan Request.

III. Advertising types and advertising with whitelist

There are 4 defined types of Advertising (Section 2.3.1 Vol 6 Part B):

- **ADV_IND: connectable undirected advertising**. This is the normal advertising type where any device can send scan response packet and connect request to the advertiser.
- **ADV_DIRECT_IND: connectable directed advertising**. You use this to direct your advertise packet to one specific central to ask for connection. The packet is still a broadcast packet but other scanners will ignore the packet if the peer address is not matched with them. And connect request or scan request from unmatched central will be ignored by the advertiser. Directed advertising usually comes with high duty cycle with interval fix at 3.75ms. For low duty cycle directed advertising, it is configurable and should be <10ms. (Section 4.4.2 Part B Vol 6).
- **ADV_SCAN_IND: scannable undirected advertising**. This advertising packet won't accept connect request but accept scan request.
- **ADV_NONCONN_IND: non-connectable undirected advertising**. This is non RX mode, which mean the advertiser will not accept any connect request or scan request. Staying in this mode the advertiser doesn't need to switch to receiver mode and can save power. The main application for this is beacon application, where maximize battery life time is most important and the beacon does not need to interact with the scanner.

Advertise with Whitelist

The advertiser can use a whitelist to limit the interaction to a number of scanner/central device. The whitelist contains an array of the peer device addresses or IRK numbers (when central use resolvable random address). It will reject packets from scanners/centrals whose addresses are not in the list. Whitelist can be configured to filter scan request packets, connect request packets or both.

SETTING UP AND START ADVERTISING WITH OUR SOFTDEVICE

I. APIs provided by the softdevice:

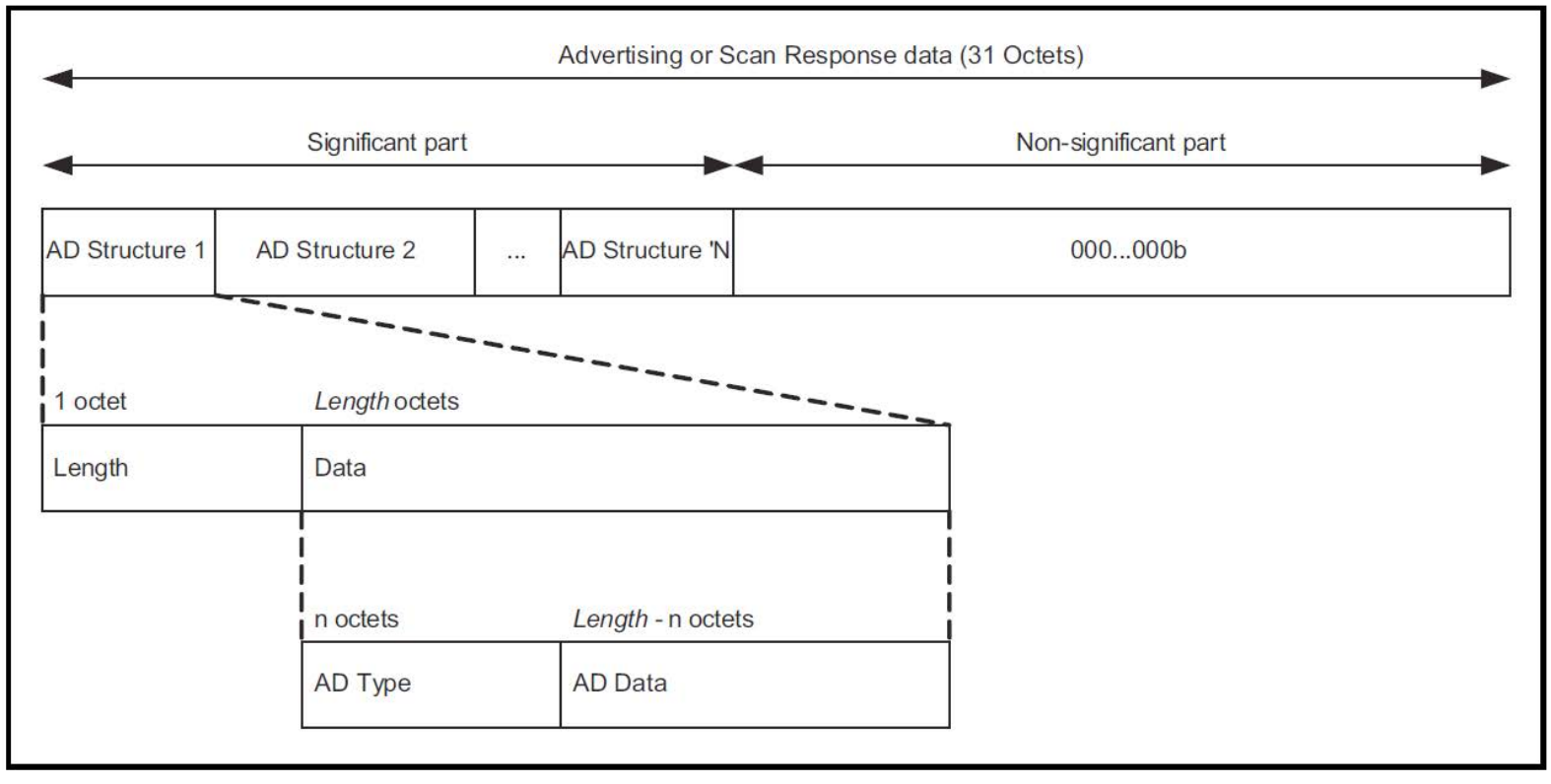
- **Setting up advertising data and the scan response data:**

  sd_ble_gap_adv_data_set(uint8_t const p_data, uint8_t dlen, uint8_t const p_sr_data, uint8_t srdlen)

- **Start advertising with the parameters:**

  sd_ble_gap_adv_start(ble_gap_adv_params_t const *p_adv_params)

Here is how the 31 bytes Advertising data and Scan response data should look like (Chapter 11 Part C Vol 3).



It is up to the application to prepare the advertising data and set it in the softdevice using **sd_ble_gap_adv_data_set().**

Note:

- The data in the input parameter is an array of uint8_t. You need to encode data to match with this.
- You should set the length of the advertising data (dlen) to match with the significant part length so that the non-significant part (zero padding) will not be transferred over the air.
- 31 bytes includes also the overhead so the actual payload for application is 27 bytes.

After you have set-up the advertising packet you can tell the softdevice to start advertising by calling **sd_ble_gap_adv_start()**. For this call, you need to configure:

- Advertising interval (the period between each advertising)
- Advertising timeout: how long you want to advertise. You will receive BLE_GAP_EVT_TIMEOUT event after this timeout.
- Advertising types (connectable, non-connectable, directed, etc),
- The peer address if you do directed advertising,
- The whitelist list if you have.
- Filter policy: Choose how to use the whitelist, filter scan request, connect request or both.
- Channel(s) you want to advertise, you can choose one of or two of or all three channels to advertise. We recommend that you use all three advertising channels.

II. GAP events you may receive when advertising:

- BLE_GAP_EVT_TIMEOUT: Occurs when the advertising timeout is passed. The application can decide to continue advertising in different mode or to enter sleep mode. The application should check the src parameter to check if the timeout event is from advertising timeout or not. See BLE_GAP_TIMEOUT_SOURCES list.
- BLE_GAP_EVT_SCAN_REQ_REPORT: The application receives this event when there is a scan request received by the advertiser. The event comes with address of the peer device and RSSI value. Note: you only get this event if you enable it using the option API sd_ble_opt_set().
- BLE_GAP_EVT_CONNECTED: You receive this when there is a central device send connect request and the connection is established.

III. Example code

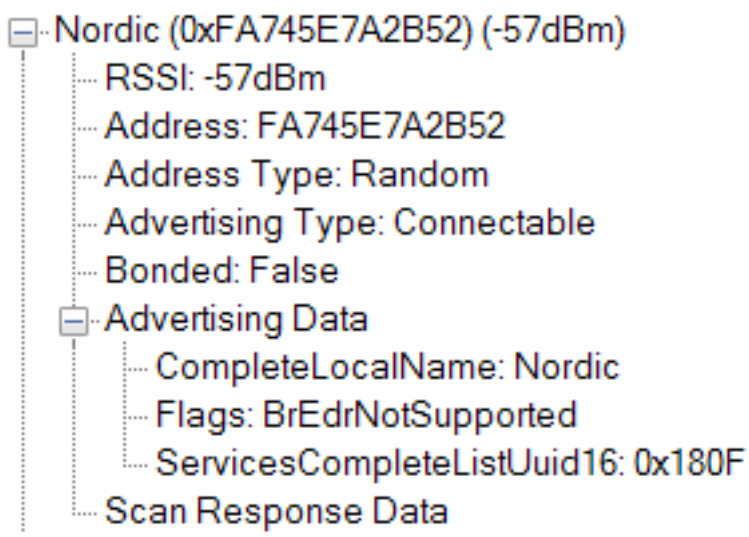
The following code (for S110v8.0/S130v1.0) will setup the advertising packet with device name, flag, Battery service UUID, and advertise with interval = 40ms (0.625 ms unit) and advertising timeout = 180 seconds. It advertises with whitelist applied for connect request. And the only peer address in the whitelist is 0xE05FDAECA271.

```
uint32_t             err_code;
ble_gap_adv_params_t adv_params;
ble_gap_whitelist_t  whitelist;
ble_gap_addr_t    * p_whitelist_addr[1];
ble_gap_addr_t        whitelist_addr={BLE_GAP_ADDR_TYPE_RANDOM_STATIC,{0x71,0xA2,0xEC,0xDA,
0x5F,0xE0}} ;
uint8_t addr[6] = {0x71,0xA2,0xEC,0xDA,0x5F,0xE0};
uint8_t adv_data[15] = {0x07,0x09,0x4E,0x6F,0x72,0x64,0x69,0x63,0x02,0x01,0x04,0x03,0x03,
0x0F,0x18};
uint8_t adv_data_length = 15;

//Setting up the advertising data with scan response data = Null
err_code = sd_ble_gap_adv_data_set(adv_data, adv_data_length, NULL, NULL);

//Configure the advertising parameter and whitelist
memset(&adv_params, 0, sizeof(adv_params));
adv_params.type        = BLE_GAP_ADV_TYPE_ADV_IND;
adv_params.p_peer_addr = NULL;
adv_params.interval    = 64;
adv_params.timeout     = 180;
p_whitelist_addr[0] = &whitelist_addr;
whitelist.addr_count = 1;
whitelist.pp_addrs   =  p_whitelist_addr;
whitelist.pp_irks = NULL;
whitelist.irk_count =0;
adv_params.fp          = BLE_GAP_ADV_FP_FILTER_CONNREQ;
adv_params.p_whitelist = &whitelist;
err_code = sd_ble_gap_adv_start(&adv_params);
```

Here is what the scanner sees when receiving the advertising packet:



Breaking down the raw advertising data:

```
{0x07,0x09,0x4E,0x6F,0x72,0x64,0x69,0x63,0x02,0x01,0x04,0x03,0x03,0x0F,0x18}
```

```
0x07 = length 7 octets
0x09 = AD type Complete Local Name
0x4E,0x6F,0x72,0x64,0x69,0x63 = “Nordic” in ASCII

0x02 = length 2 octects
0x01 = AD type flags
0x04 = Flag BLE_GAP_ADV_FLAG_BR_EDR_NOT_SUPPORTED

0x03 =length 3 octects
0x03 = AD type Complete List of 16-bit Service Class UUIDs
0x0F,0x18 = Battery Service
```

List of AD types can be found here.

To make the code short and simple, the check of the return code (err_code) of the API call is not added here. When you implement your code, make sure the return code of each API is checked and it should return NRF_SUCCESS (0x00).

The examples in our SDK provide an Advertising module (ble_advertising.c) with abstraction layers and predefined modes. Documentation can be found here.

In addition we provide the advertising data encoding module (ble_advdata.c) that you can use to generate the advertising/scan response packet.

CONNECTION ESTABLISHING AND TERMINATING

Recap from previous chapter: the advertiser broadcasts advertising packets so that scanner and initiator can receive the advertising data or initiate a connection.

I. Initializing

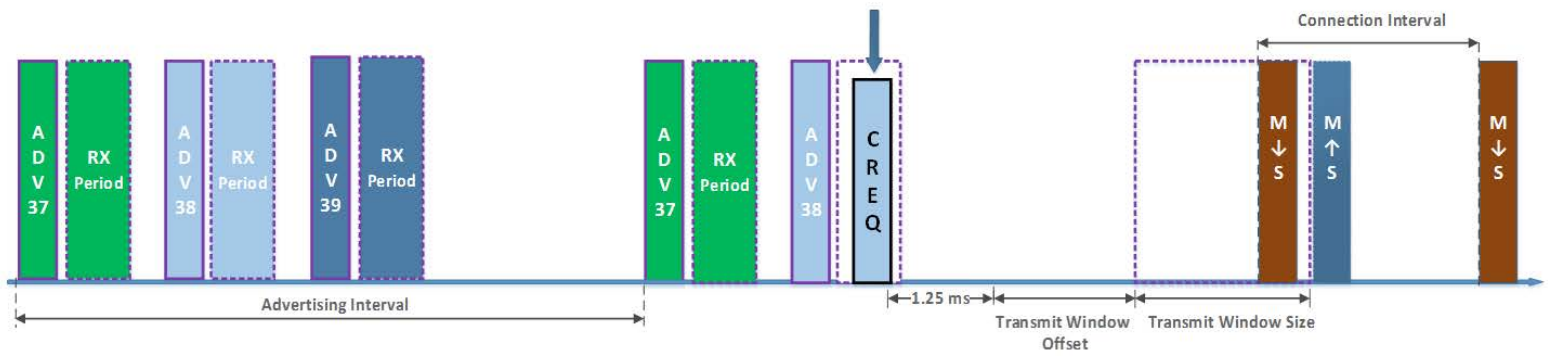
A connection is initialized with a CONNECT REQUEST.

A central that wants to start a connection has to scan for an advertising packet and send a connect request packet after it receive the advertising packet from the advertiser it is looking for. The connect request must be sent within the RX windows opened by the advertiser right after each advertising packet transmission [Section 4.2.2.5 Vol1 Part A]. This is shown in the figure below.

When the advertiser receives a connect request, and if the connection request is accepted, the advertiser should stop advertising in the next channel and/or next advertising interval then follow the parameter in the connect request packet to start a connection. The advertiser accepts a connect request when it is advertising in connectable mode and with one of following cases:

- Advertising with no whitelist
- With whitelist and the initiator's address is in the whitelist
- Directed advertising and the address of the initiator matches

The following figure shows an example when an advertiser receives a connect request on channel 38. It stops advertising when it receives the packet, and a connection is established with the parameter included in the connect request. We will discuss these parameters in the next section.



The first connection packet (connect event) should happen inside a transmit window defined by the transmit window size and the transmit window offset[4.5.3 Vol6 PartB]. After that, the Advertiser becomes Slave and the Initiator becomes Master. The master will keep sending connection event packet on every connection interval and the Slave in accordance will open the RX windows on every connection interval after the first connection event, and then send a connection event packet back to the master on each connection event. This connection event packet act as an ACK. It can contain data payload or not.

Note that the master has no way to know if the connect request is accepted by the advertiser or not, so it will assume that the connection is accepted and keep sending connection event packets, in our case it will keep sending 6 packets (6 connection events) and stop if there is a connect event packet back from the slave to ACK. Same with the slave, if it would not receive any packet within the period of first 6 connection interval, it will terminate the connection [4.5.2 Vol 6 Part B].

II. Connection parameters

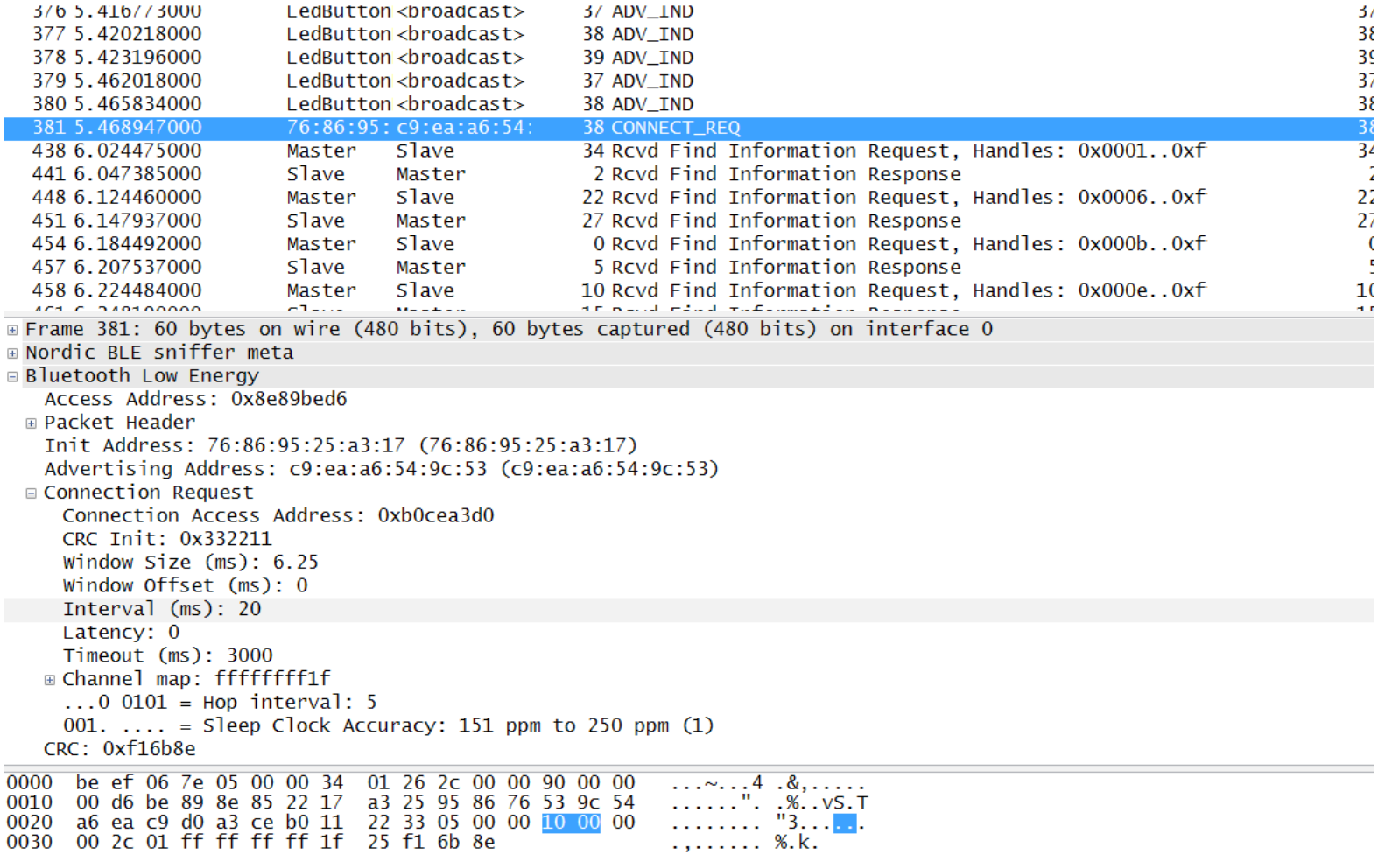
Connection parameters are crucial for a connection to be established. They provide information on how should the master and the slave communicate. The connection parameters are included in the Connect Request payload as follow [2.3.3.1 Vol6 PartB]:

LLData									
AA (4 octets)	CRCInit (3 octets)	WinSize (1 octet)	WinOffset (2 octets)	Interval (2 octets)	Latency (2 octets)	Timeout (2 octets)	ChM (5 octets)	Hop (5 bits)	SCA (3 bits)

Figure 2.11: LLData field structure in CONNECT_REQ PDU's payload

- **Interval:** defines the interval of the connection. How frequently the master will send a connection event packet to slave. Connection interval = interval * 1.25ms.
- **Latency:** Slave latency. The slave can skip waking up and response to the connection event from master to slave. The latency is the number of connect event the slave can skip. This is to save power on the slave side. When it has no data it can skip some connection events. But the sleeping period should not be too long so that the connection will timeout.
- **Timeout:** How long would the master keep sending connection event without response from slave before terminate the connection.
- **ChM:** Channel map, which channels will be used and which will not be used. 37 LSB bits represent the 37 channels can be used for connection packets. See PART 1 for the list of 40 channels for both advertising and connection.
- **Hop:** The hop increment in the data channel selection. How big should the channel be jumped in side the channel map list. The algorithm is at Section 4.5.8.2 Vol 6 Part B
- **SCA:** The worst case of the Sleep clock accuracy on the master. The slave uses this value in combination with the tolerance of its own sleep clock to determine how big should its RX window be opened to cope with the inaccuracy of the clock on the master. The higher the SCA the more power consumption the slave uses.
- **AA:** Access address of the connection, each connection need an unique access address that both peer devices will use instead of using BED6 address as in advertising.
- **CRC Init:** initialization value for the CRC calculation for linklayer
- **WinSize:** TransmitWindowSize = WinSize * 1.25 ms
- **WinOffset:** TransmitWindowOffset = WinOffset * 1.25 ms

The screenshot below shows an example of a sniffer trace, where you can see the connect request packet. You can find information about the connection in the connect request packet, including connection interval, connection timeout, etc, as discussed above. This is very useful for debugging and analyzing a connection.



III. Whitelisting

There are two different whitelists that can be involved when initiating the connection:

- The whitelist on the initiator limits the number advertisers it should look for and send connect request to.
- The whitelist on the advertiser has the list of initiator that it should accept the connect request (and/or scan request) from.

IIII. Connection Terminating

A connection can be terminated when:

- One of the peers voluntary terminates by sending LL_TERMINATE_IND PDU
- Connection timed out. This is when one of the peer devices failed to ACK or send a data channel packet within the connection timeout period.
- It is failed to established. This is when the connection request is not received by the advertiser or if the advertiser rejected it.

There are several error codes can be used for the LL_TERMINATE_IND, this will match with the value of the **reason** variable in ble_gap_evt_disconnected_t struct when you receive BLE_GAP_EVT_DISCONNECTED event. Below is the most common **reason** to receive:

- REMOTE USER TERMINATED CONNECTION (0x13): The remote device/user terminated the connection.
- BLE_HCI_CONNECTION_TIMEOUT (0x08): Connection timeout
- CONNECTION TERMINATED BY LOCAL HOST (0x16): Disconnected because the local device terminated connection.
- CONNECTION FAILED TO BE ESTABLISHED (0x3E): This is when the slave did not receive the connect request or if it reject that connect request.
- UNACCEPTABLE CONNECTION PARAMETERS (0x3B): This is when the slave want to disconnect because it won't accept the connection parameters from master.

Note that there are only two reasons should be used when calling sd_ble_gap_disconnect() which are REMOTE USER TERMINATED CONNECTION and UNACCEPTABLE CONNECTION PARAMETERS.

WHEN IN CONNECTION

I. Data channel packet format

[Section 2 Part B Vol 6]

On linklayer, there is only one packet format for both advertising data packet and data channel packet (connection packet) as follow:

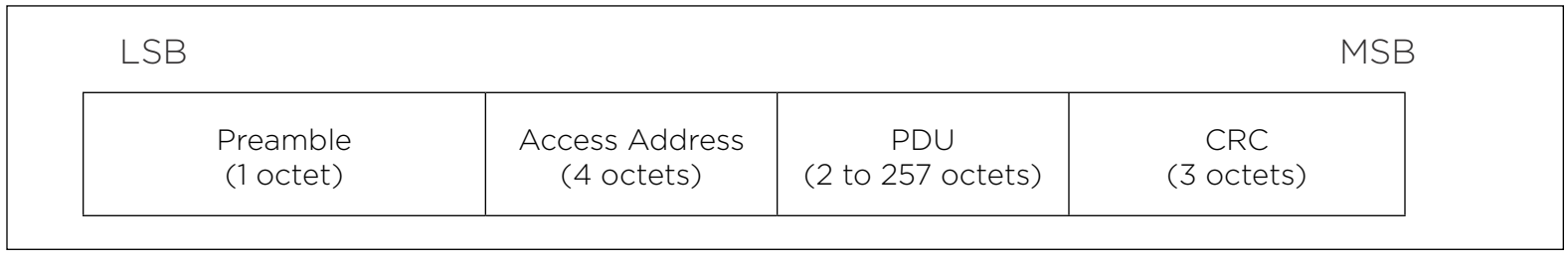


Figure 2.1: Link Layer packet format

The Access Address is the address provided in the CONNECT REQ packet.

The PDU (Protocol Data Unit) of a data channel is as follow:

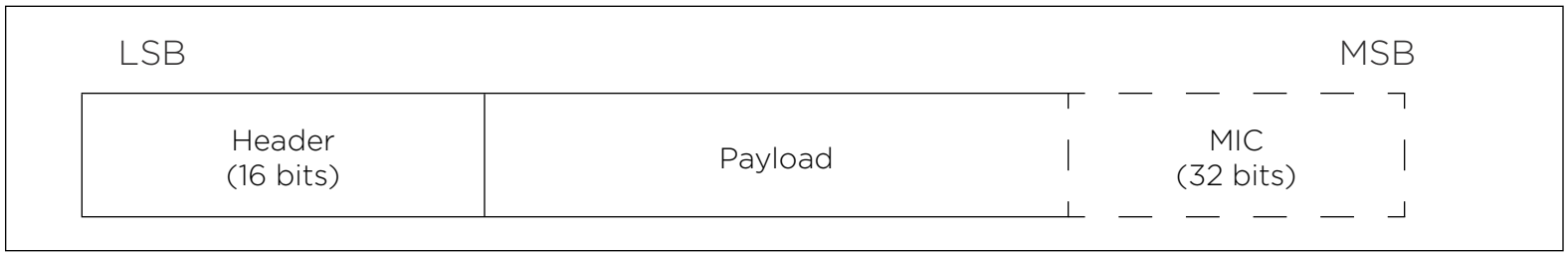


Figure 2.12: Data Channel PDU

The header of the PDU is composed of :

Header					
LLID (2 bits)	NESN (1 bit)	SN (1 bit)	MD (1 bit)	RFU (3 bits)	Length (8 bits)

Figure 2.13: Data Channel PDU header

- The LLID tells if it is an empty packet (just for acknowledgment) or data PDU or control PDU.
- NESN and SN is the sequence number for ACKing and flow control, that will be explained below.
- MD: More data bit. When receiving this the receiver knows that peer device want to send more data in this connection event. Our softdevices support up to 6 packets per connection event. Note that with S13x you have to configure the connection with high bandwidth to support 6 packets.
- Length: The length of the payload and MIC (for encrypted connection). Maximum payload MTU (at the moment) that our softdevices support on Link Layer is 27 bytes. There is a 4 byte header inside the PDU's payload, so there are 23 bytes max payload for L2CAP layer and with 3 byte header for ATT layer, there are 20 bytes actual payload for application. We will support longer L2CAP MTU in the future version of the S13x softdevice.

Note:

- The connection parameters can be changed during connection. It is the master's duty to select or update it. The Slave can however, request a change. But if the change is not accepted by the master, the request will be rejected. We will discuss about connection parameter update request in another document.

II. Acknowledgement and Flow Control

The link layer uses 2 bits for acknowledgement: transmitSeqNum and nextExpectedSeq-Num. We will call it sn and nesn to distinguish from SN and NESN bits from the packet.

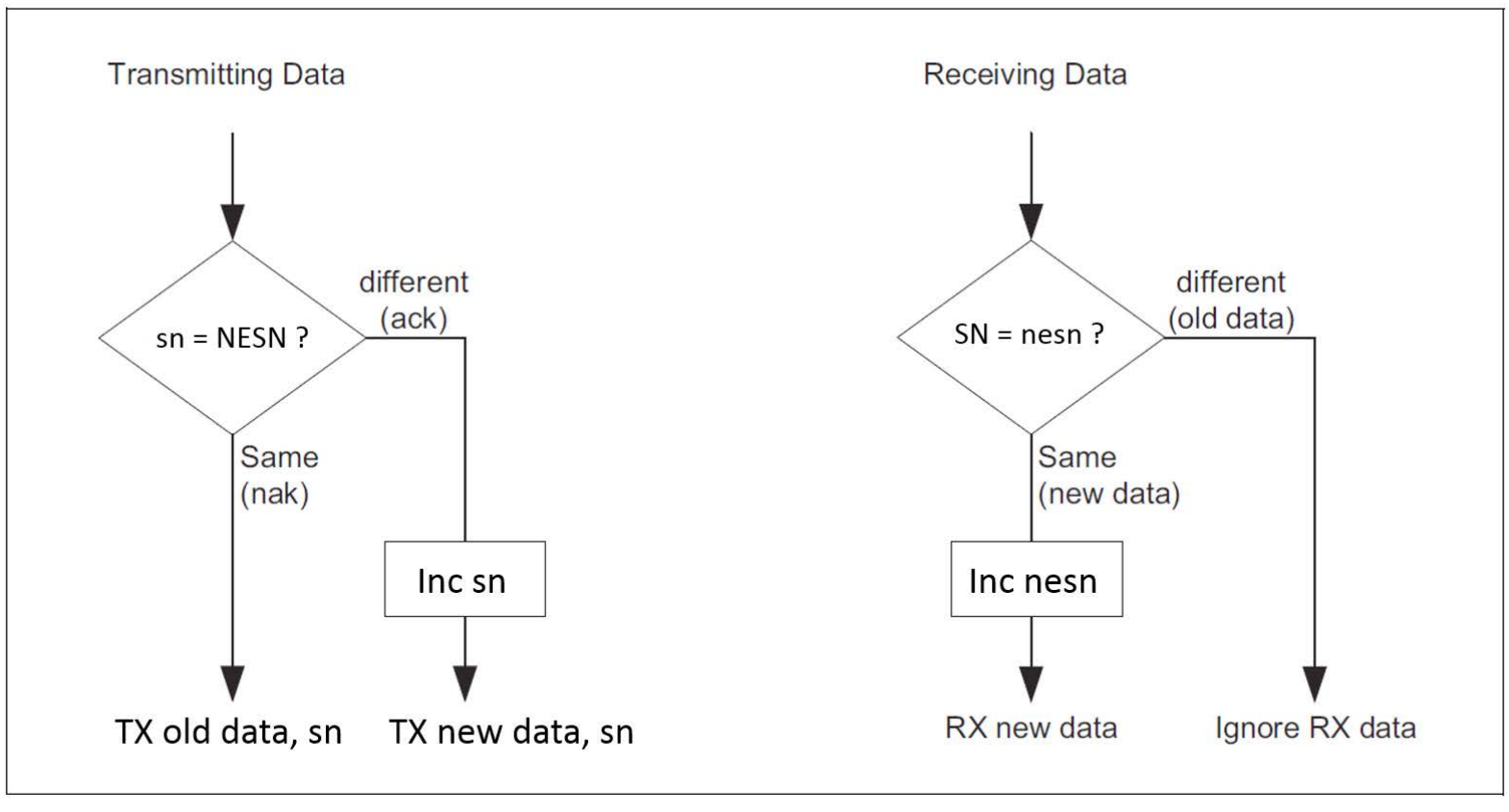
Basically, for both master and slave, when receiving a packet, the packet's NESN will be compared with the device's sn. If they are the same, then it is an NAK from the peer and the device will have to resend the previous TX data. If not, then it is an ACK, and the device will increase its sn and send new data and sn.

Secondly, SN from the packet will also be compared to the device nesn, if it is the same then it is new RX data in the packet from the peer and the device will increase its nesn (means ACK) before sending next packet. [4.5.9 Vol 6 Part B]

Note:

- These two comparisons are independent, so a packet can be an NACK but still can have new data and vice versa.
- The SN and NESN of a packet that a device sent are equal to the sn and nesn it has at that moment.
- When a device cannot receive further packets, for example when there is no RX buffer left, it can NACK the packets by not increasing nesn. This will tell the peer device to retransmit the TX old data.

The following figure describes how it works:



The sniffer trace bellow shows an example of NACK packets. The master in this case only has one RX buffer, so it will NACK any further packet from the Slave in the same connection event.

No.	Time	Source	Destination	channel	Info	Value	channel	SN	NESN	Length
1033	11.984154000	Master	Slave	7	Empty Data PDU			7	1	1	26
1034	11.987129000	Slave	Master	7	Empty Data PDU			7	1	0	26
1035	12.004146000	Master	Slave	12	Empty Data PDU			12	0	0	26
1036	12.007684000	Slave	Master	12	Rcvd Handle Value Notification, Handle: 0x000e	00000		12	0	1	53
1037	12.010673000	Master	Slave	12	Empty Data PDU			12	1	1	26
1038	12.013822000	Slave	Master	12	Rcvd Handle Value Notification, Handle: 0x000e	01000		12	1	0	53
1039	12.024308000	Master	Slave	17	Empty Data PDU			17	0	1	26
1040	12.029878000	Slave	Master	17	Rcvd Handle Value Notification, Handle: 0x000e	01000		17	1	1	53
1041	12.033144000	Master	Slave	17	Empty Data PDU			17	1	0	26
1042	12.035858000	Slave	Master	17	Rcvd Handle Value Notification, Handle: 0x000e	02000		17	0	0	53
1043	12.044206000	Master	Slave	22	Empty Data PDU			22	0	0	26
1044	12.047876000	Slave	Master	22	Rcvd Handle Value Notification, Handle: 0x000e	02000		22	0	1	53
1045	12.050986000	Master	Slave	22	Empty Data PDU			22	1	1	26
1046	12.053533000	Slave	Master	22	Rcvd Handle Value Notification, Handle: 0x000e	03000		22	1	0	53
1047	12.064168000	Master	Slave	27	Empty Data PDU			27	0	1	26
1048	12.067715000	Slave	Master	27	Rcvd Handle Value Notification, Handle: 0x000e	03000		27	1	1	53
1049	12.070590000	Master	Slave	27	Empty Data PDU			27	1	0	26
1050	12.073167000	Slave	Master	27	Rcvd Handle Value Notification, Handle: 0x000e	04000		27	0	0	53
1051	12.084176000	Master	Slave	32	Empty Data PDU			32	0	0	26
1052	12.088229000	Slave	Master	32	Rcvd Handle Value Notification, Handle: 0x000e	04000		32	0	1	53
1053	12.091256000	Master	Slave	32	Empty Data PDU			32	1	1	26
1054	12.097817000	Slave	Master	32	Rcvd Handle Value Notification, Handle: 0x000e	05000		32	1	0	53
1055	12.104130000	Master	Slave	0	Empty Data PDU			0	0	1	26
1056	12.107844000	Slave	Master	0	Rcvd Handle Value Notification, Handle: 0x000e	05000		0	1	1	53
1057	12.110869000	Master	Slave	0	Empty Data PDU			0	1	0	26
1058	12.113372000	Slave	Master	0	Rcvd Handle Value Notification, Handle: 0x000e	06000		0	0	0	53
1059	12.124233000	Master	Slave	5	Empty Data PDU			5	0	0	26
1060	12.127900000	Slave	Master	5	Rcvd Handle Value Notification, Handle: 0x000e	06000		5	0	1	53
1061	12.137252000	Master	Slave	5	Empty Data PDU			5	1	1	26
1062	12.139870000	Slave	Master	5	Rcvd Handle Value Notification, Handle: 0x000e	07000		5	1	0	53
1063	12.144074000	Master	Slave	10	Empty Data PDU			10	0	1	26

As shown in the image, on channel 22 after the first notification from Slave (#1044), the master NACK the next notification (#1046) by not increasing nesn (NESN on #1045 = NESN on #1047). On the slave side, when receiving #1047, the NESN on that packet = sn on the slave, meaning the slave has to resend TX data. The data is resent on packet #1048.

SETTING UP A CONNECTION WITH NORDIC SOFTDEVICE

How a connection is established and maintained is controlled by GAP layer. You will find most APIs and Event for connection state in ble_gap.h file.

I. APIs provided by the softdevice:

- **Start scanning for advertising packet and report when receives**

  sd_ble_gap_scan_start(ble_gap_scan_params_t const *p_scan_params))

- **Start scanning for advertising packet and send a connect request if address matches:**

  sd_ble_gap_connect(ble_gap_addr_t const p_peer_addr, ble_gap_scan_params_t constp_scan_params, ble_gap_conn_params_t const *p_conn_params))

- **Teminate a connection:**

  sd_ble_gap_disconnect(uint16_t conn_handle, uint8_t hci_status_code))

- **Stop scanning:**

  sd_ble_gap_scan_stop()

- **Stop connecting:**

  sd_ble_gap_connect_cancel()

For scanning, we need to provide scan parameters to the Softdevice. These scan parameters include the scan window, scan interval, whitelist, active or passive scan. For connecting, beside scan parameters, we also have to provide the connection parameters which are connection interval, connection timeout, slave latency.

Note that only max_conn_interval will be used as connection interval but you still need to set:

7.5ms ≤ min_conn_interval ≤ max_conn_interval

II. GAP events you may receive when initiate/terminate connection:

BLE_GAP_EVT_ADV_REPORT: Occurs when you are scanning and get an advertising packet or a scan response packet. In this event, you can read the advertising data or scan response data and the address of the advertiser. From here, you can start a scan and connect command (sd_ble_gap_connect) to the peripheral device (based on the address) either by directly to the peer address, or by adding the peer address to the whitelist. The connect request will be sent the next time your device receive the advertising packet from the peer. Note that a peripheral device is unable to send connect requests.

BLE_GAP_EVT_CONNECTED

BLE_GAP_EVT_DISCONNECTED

Each BLE event comes with parameters of the type ble_evt_t. For connected event, you can find the role of the connection (if the role of the device is central or peripheral), the address of the peer, if it is in whitelist or not and if it is, which id in the white list it is (ble_gap_evt_connected_t) . For disconnected event, you can find the reason of the disconnection (ble_gap_evt_disconnected_t). This is important for debugging. The address of the peer device is not included, but you can use conn_handle inside ble_gap_evt_t to find which connection caused the event.

SOURCES TO MORE KNOWLEDGE ABOUT BLUETOOTH LOW ENERGY

There is a Bluetooth low energy introduction video here and on developer.bluetooth.org site. To get up to speed with Bluetooth low energy, there are a few books on the subject, e.g. this one, this one and this one. The first one includes chapters on how to create Bluetooth low energy apps for IOS and Android phones that connect and communicate with Bluetooth low energy devices. Additionally, if you want to look at these books right away, then they are both available on www.safaribooksonline.com, where they offer 30 day trial period. For in depth information on Bluetooth low energy, look at the Bluetooth core specification.

Telephone: +47 22 51 10 50

Postal address:
Nordic Semiconductor ASA
P.O. Box 436, Skøyen
0213 Oslo
Norway

For direct shipment/ visiting address:
Karenslyst Allé 5
0278 Oslo
Norway

www.nordicsemi.com