Android SDK Documentation

Version	Date	Editor
0.9	08/19/2014	Ronny Lerch
1.0	08/21/2014	Brandon Jones

1. Purpose
2. Structure
    1. *UML Diagrams*
    2. *Object Description*
    3. *Sequence Diagrams*
        1. *Advertisment*
        2. *Le Sticknfind*
    4. *Callbacks*
3. Use Cases
4. Examples
    1. *Configure Advertisement Data*
    2. *Read Advertisement Data*

1. Purpose

The Android SDK simplifies BLE tag operations:
(1) Connection handling
(2) Configuration
(3) Firmware updating.

Android BLE functions are wrapped and exposed to the corresponding Handler calls. A certain number of callbacks need to be implemented additionally to the existing Android BLE callbacks.

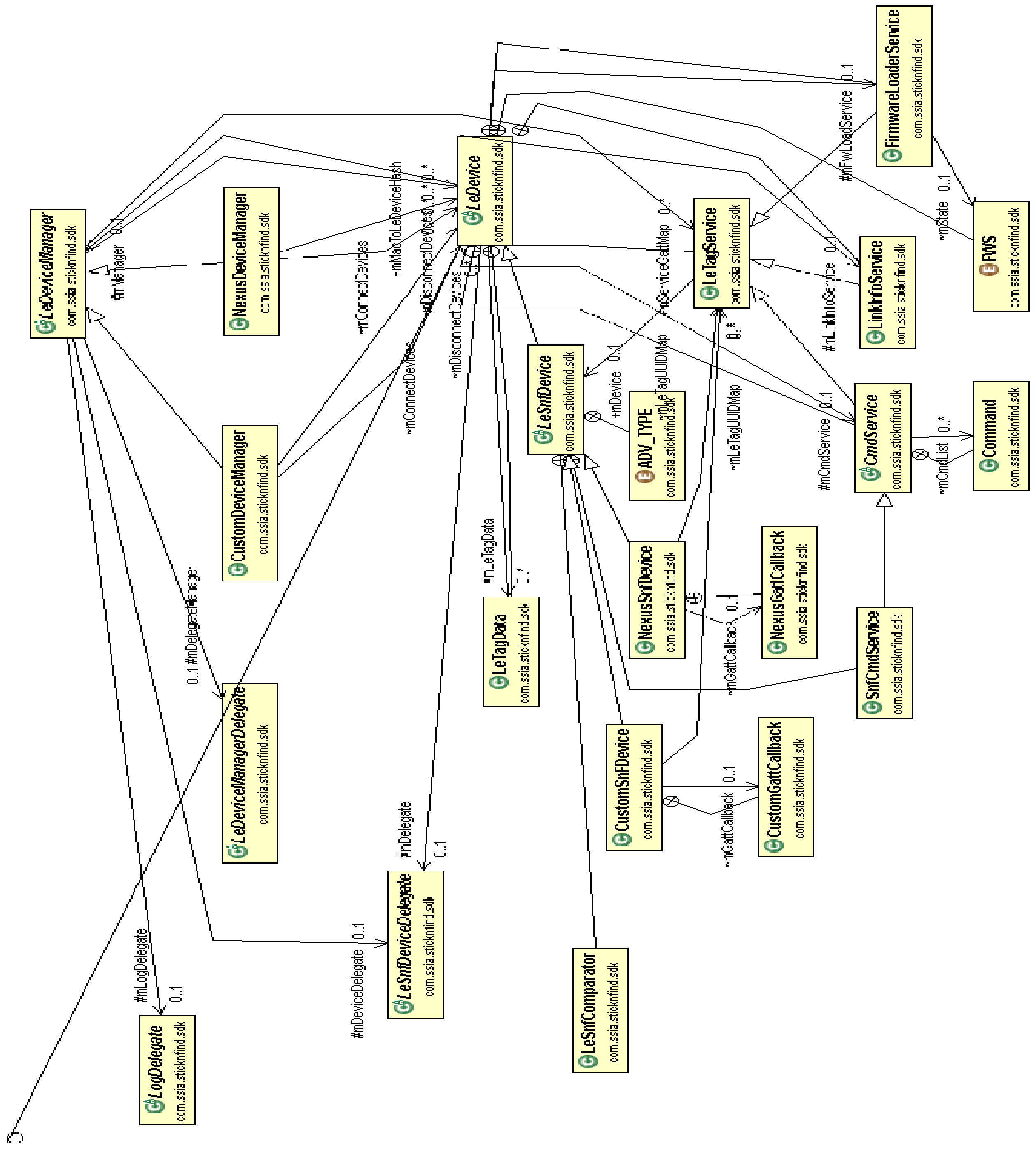
Please refer to the Android Development Website for more details.
A basic BLE skill level is highly recommended, as the SticknFind SDK assumes basic BLE knowledge.

Characteristic operations like read and write are built into the main code base. Callbacks are provided to leave it up to you, the developer, to take appropriate action for your given use case.

Restricting BLE functions was not a goal in the development of the SDK. StickNFind's requirement was transparency, usability and minimal limitation on what a consumer of the SDK could control.

The SDK requires a basic knowledge of the BLE core specification to perform certain functionalities like manipulation of advertisement data.

2.1 UML Diagram



The UML Diagram shows the relation of the SDK Objects.

2.2 Object Description:

(i) LeDeviceManager:

- Abstract super class for NexusDeviceManger & CustomDeviceManager
- Filters for Broadcast data
- Loads firmware files

(ii) LogDelegate

- Implementation of Log callbacks

(iii) LeDeviceManagerDelegate

- Implementation of DeviceManager callbacks

(iv) CustomDeviceManager

- Controls connection and scan process
- Houses specific StickNFind methods

(iv) LeDevice

- Abstract superclass for LeSnfDevice
- Including basic functionalities for BLE tags
  - read battery & temperature
  - set alert
  - Configure connection rate and latency
  - Handling of authentication key
  - Enable notifications
- Service objects implementation
  - Command Service
  - Link Info Service
  - Firmware Service

(v) LeSnfDeviceDelegate

- Implementation of LeSnfDevice callbacks

(viii) LeTagData

- Structure for Commands

(ix) LeSnfDevice

- Implementation of StickNFind specific BLE tag functions
  - Configuration beacon data & settings
  - Factory reset
  - Temperature calibration
  - Configuration of device name

    - Configuration of melody
    - Requesting of device ID, advertisement key, advertisement data, temperature, battery, acceleration configuration, advertisement settings, system time, rssi calibration

(x) LeSnfComparator

- Comparator function for temperature log

(xi) CustomSnfDevice

- Extends to LeSnfDevice
- Handling of connection & discovery process
- Adds Custom methods for third party
- Contains CustumGattCallback for BluetoothGattCallbacks

(xii) NexusSnfDevice

- Extends to LeSnfDevice
- Handling of connection & discovery process
- Contains NexusGattCallback for BluetoothGattCallbacks

(xiii) LeTagService

- Superclass for all BLE Services
- Includes all Characteristic callbacks on onWrite, onRead, onNotification

(xiv) CmdService

- Command Service handles all command type characteristic operations
    - Alert
    - Connection settings
    - Latency configuration
    - Authentication mechanism
    - Advertisement conversions
    - Calibrations
    - Acceleration functions

(xv) LinkInfoService

- Implementation of remote rssi functionality

(xvi) FirmwareLoaderService

- Implementation of firmware update functionality

(xvii) SnfCmdService

- Extends to CmdService
- Handles all LeSnfDevice specific Commands & Callbacks
    - Advertisement callbacks
    - Device ID callback

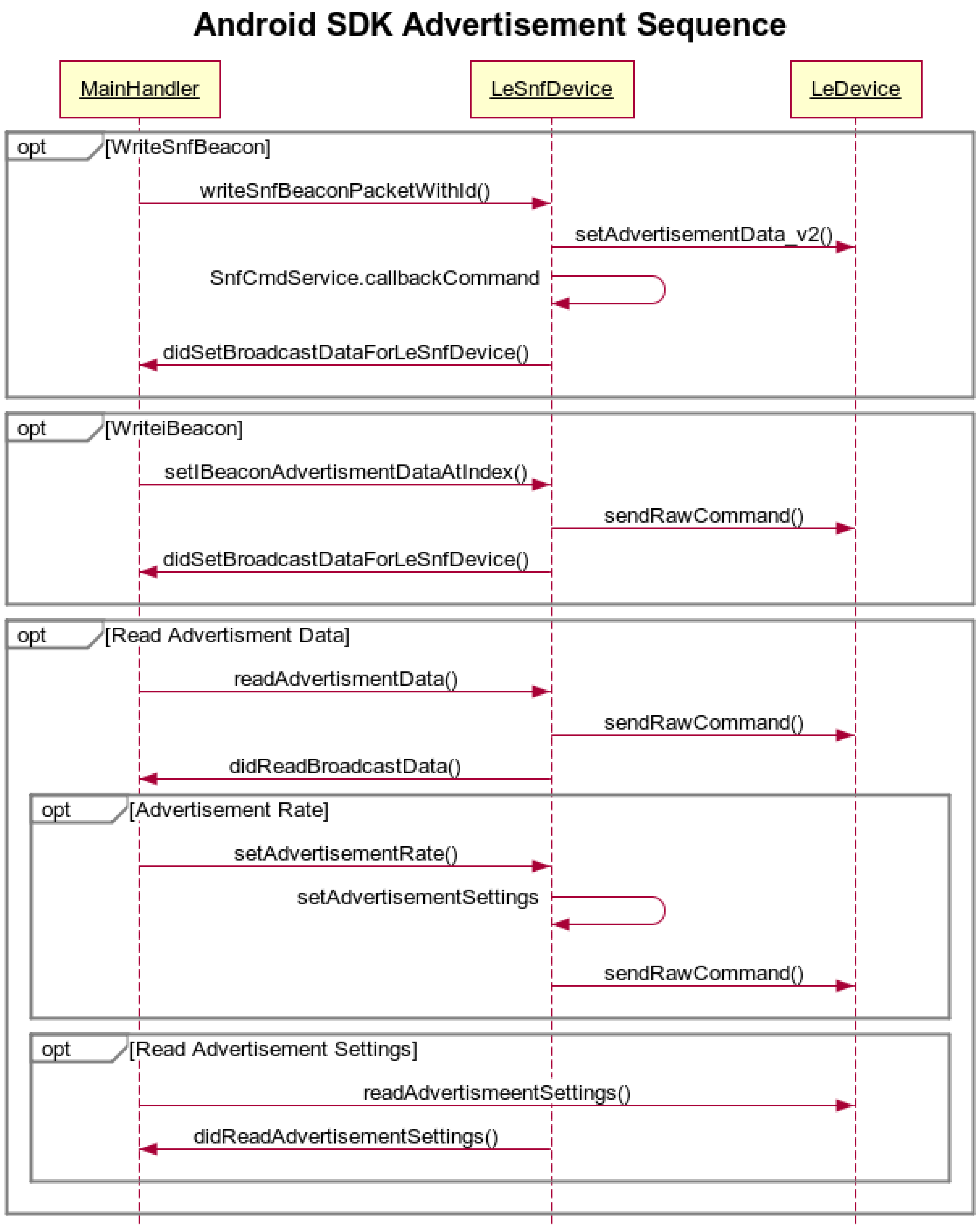
    - Device property requests like battery type, beacon kind, calibration values and authentication verification

(xviii) Command

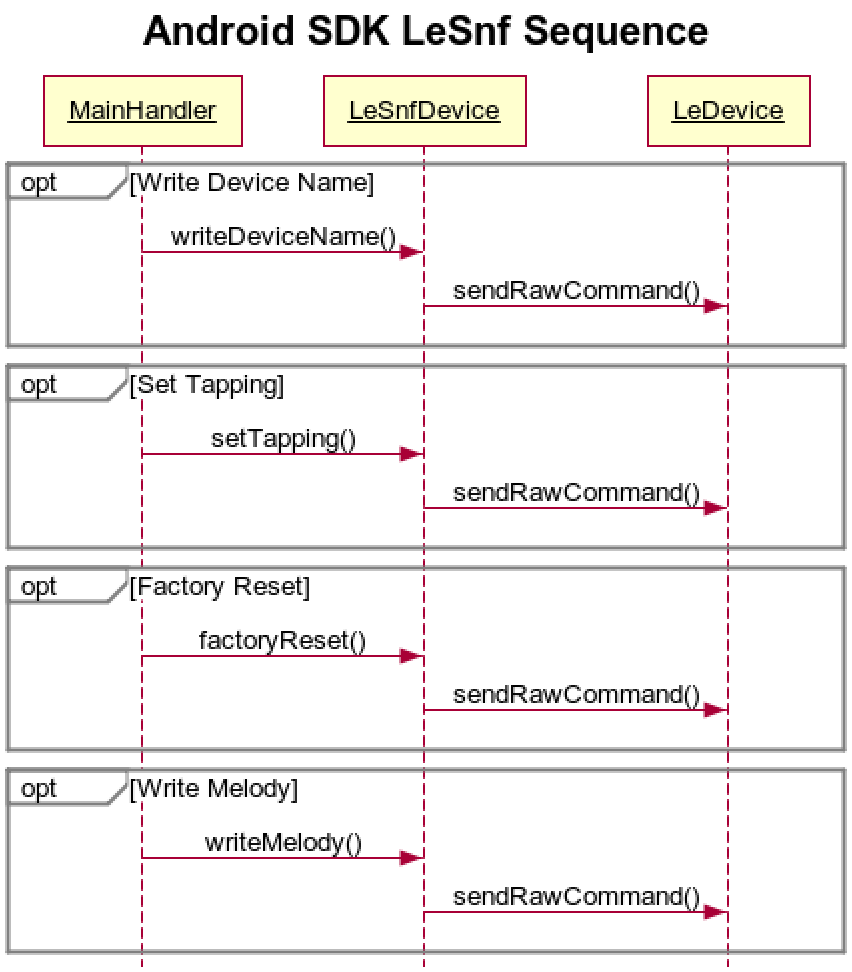
- Inner class of LeDevice
- Contains the structure of the command
    - byte array for buffer
    - info index
    - offset value

2.3 Sequence Diagrams

2.3.1 Advertisement


Android SDK Advertisement Sequence
MainHandler
LeSnfDevice
LeDevice
opt [WriteSnfBeacon]
writeSnfBeaconPacketWithId()
setAdvertisementData_v2()
SnfCmdService.callbackCommand
didSetBroadcastDataForLeSnfDevice()
opt [WriteiBeacon]
setIBeaconAdvertismentDataAtIndex()
sendRawCommand()
didSetBroadcastDataForLeSnfDevice()
opt [Read Advertisment Data]
readAdvertismentData()
sendRawCommand()
didReadBroadcastData()
opt [Advertisement Rate]
setAdvertisementRate()
setAdvertisementSettings
sendRawCommand()
opt [Read Advertisement Settings]
readAdvertismeentSettings()
didReadAdvertisementSettings()

2.3.2 Le Sticknfind



2.4 Callbacks

The Android SDK is wrapping native characteristic callback like onCharacteristicRead, onCharacteristicWrite etc. Additional StickNFind callbacks are related to corresponding combined service commands (CSCMD).

For example:

readAdvertisementData() is using internal the CSCMD_ADV_SETTINGS command and result in a didReadBroadcastData callback.

Similar operations build the entire callback state machine.

3. Use Cases

3.1 Advertisement Data Configuration

Our unique BLE tag firmware includes a generic packet engine which allows to customize the advertising payload.
The SDK includes a method setAdvertisementData_v2 with following parameters:

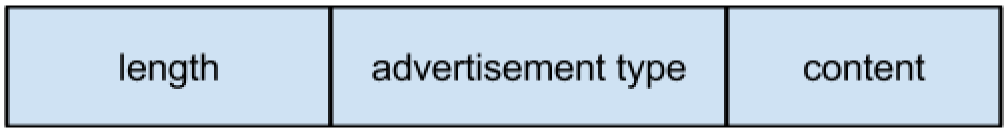
```
 * @param data - data information
 * @param atIndex - device specific (0 - 3 Ble Specification) / 4 - 5 custom
 * @param intervalSkip - advertisement slot to skip
 * @param hwAddr - in case should transmit with different hwaddresses
 * @param hwAddrPublic - activate hw address
 * @param advType - use ADV_TYPE(Connectable, None_Connectable, Scanable)
WARNING: NONE_CONNECTABLE setup results in configuring the beacon to be no more
connectable
 * @param txPower - see below channel_txpower params
 * @param dynOfs - offset index within data where dynamic data start
 * @param dynLength - length off dynamic data array
 * @param kidx - advertisement key index to use for encryption(Advisory Name requires 0x80
index)
 * @param encrLength - amount of data fields to encrypt
 * @param encrOfs - start of index of entire packet( index 7 is start of data field)
 * @param rlen - random length -> amount of data to randomly encrypt
 *
 * For channel_txpower params:
 * lowest 4 bits -> 0:-40dBm
 *                  1:-20dBm
 *                  2:-16dBm
 *                  3:-12dBm
 *                  4:-8dBm
 *                  5:-4dBm
 *                  6: 0dBm
 *                  7:+4dBm
 * next 4 bits -> enables channels: 0x10:ch37
 *                                  0x20:ch38
 *                                  0x40:ch39
```

The parameters root is the BLE Core Specification which can be found at bluetooth.org. See section *"9 OPERATIONAL MODES AND PROCEDURES – LE PHYSICAL TRANSPORT".* Additional the Core Specification Supplement.

Please find some additional BLE information below:

@param data - data information

The raw payload data format. The byte array is 31 bytes long.
Each sequence has following format:



Following types are available:

```
ADTYPE_FLAGS                    = 0x01; // Discovery Mode
ADTYPE_16BIT_MORE               = 0x02; // Service: More 16-bit UUIDs available
ADTYPE_16BIT_COMPLETE           = 0x03; // Service: Complete list of 16-bit UUIDs
ADTYPE_32BIT_MORE               = 0x04; // Service: More 32-bit UUIDs available
ADTYPE_32BIT_COMPLETE           = 0x05; // Service: Complete list of 32-bit UUIDs
ADTYPE_128BIT_MORE              = 0x06; // Service: More 128-bit UUIDs available
ADTYPE_128BIT_COMPLETE          = 0x07; // Service: Complete list of 128-bit UUIDs
ADTYPE_LOCAL_NAME_SHORT         = 0x08; // Shortened local name
ADTYPE_LOCAL_NAME_COMPLETE      = 0x09; // Complete local name
ADTYPE_POWER_LEVEL              = 0x0A; // TX Power Level: 0xXX: -127 to +127 dBm
ADTYPE_OOB_CLASS_OF_DEVICE      = 0x0D; // Simple Pairing OOB Tag: Class of device (3 octets)
ADTYPE_OOB_SIMPLE_PAIRING_HASHC   = 0x0E; //Simple Pairing OOB Tag: Simple Pairing Hash C (16
octets)
ADTYPE_OOB_SIMPLE_PAIRING_RANDR   = 0x0F; // Simple Pairing OOB Tag: Simple Pairing Randomizer R
(16 octets)
ADTYPE_SM_TK                    = 0x10; // Security Manager TK Value
ADTYPE_SM_OOB_FLAG              = 0x11; // Secutiry Manager OOB Flags
ADTYPE_SLAVE_CONN_INTERVAL_RANGE   = 0x12; // Min and Max values of the connection interval (2 octets
Min, 2 octets Max) (0xFFFF indicates no conn interval min or max)
ADTYPE_SIGNED_DATA              = 0x13; // Signed Data field
ADTYPE_SERVICES_LIST_16BIT      = 0x14; // Service Solicitation: list of 16-bit Service UUIDs
ADTYPE_SERVICES_LIST_128BIT     = 0x15; // Service Solicitation: list of 128-bit Service UUIDs
ADTYPE_SERVICE_DATA             = 0x16; // Service Data
ADTYPE_MANUFACTURER_SPECIFIC    = 0xFF;
```

The length is related to the advertisement type.

Payload data arrays require to contain the ADTYPE_FLAGS and the ADTYPE_16BIT_MORE(see 1.1 Service UUID within CSS document).
The remaining 24 bytes can be used to advertise:
(1) ADTYPE_LOCAL_NAME_SHORT
(2) ADTYPE_LOCAL_NAME_COMPLETE
(3) ADTYPE_MANUFACTURER_SPECIFIC

for example.
Special features are available in this case:

LeSnfDeviceBroadcastV2dynBatteryLevel	*= 0x01;*
LeSnfDeviceBroadcastV2dynBatteryVoltage	*= 0x02;*
LeSnfDeviceBroadcastV2dynTemperature	*= 0x03;*
LeSnfDeviceBroadcastV2dynLinkLossCounter	*= 0x04;*
LeSnfDeviceBroadcastV2dynScanDeviceCount	*= 0x05;*
LeSnfDeviceBroadcastV2dynScanTime	*= 0x06;*
LeSnfDeviceBroadcastV2dynStateTime	*= 0x07;*
LeSnfDeviceBroadcastV2dynGbcnID	*= 0x08;*
LeSnfDeviceBroadcastV2dynGbcnMAC	*= 0x09;*
LeSnfDeviceBroadcastV2dynRssiValue	*= 0x0C;*
LeSnfDeviceBroadcastV2dynRssiCalibration	*= 0x0D;*
LeSnfDeviceBroadcastV2dynRssiCalibrationCh	*= 0x0E;*
LeSnfDeviceBroadcastV2dynRssiCalibrationChId	*= 0x0F;*
LeSnfDeviceBroadcastV2dynRssiValueDec3B	*= 0x10;*
LeSnfDeviceBroadcastV2dynGbcnIDV2	*= 0x11;*
LeSnfDeviceBroadcastV2dynButtonCounter	*= 0x20;*
LeSnfDeviceBroadcastV2dynStartupCounter	*= 0x24;*
LeSnfDeviceBroadcastV2dynPacketCounter	*= 0x28;*
LeSnfDeviceBroadcastV2dynAdvCounter	*= 0x2C;*
LeSnfDeviceBroadcastV2dynScanCounter	*= 0x30;*
LeSnfDeviceBroadcastV2dynFirmwareRevision	*= 0x34;*
LeSnfDeviceBroadcastV2dynSystemTime	*= 0x38;*
LeSnfDeviceBroadcastV2dynTapCounter	*= 0x3C;*

Caps in between align to the corresponding size of this feature. System time for example is 4 bytes long because of 0x38, 0x39, 0x3A, 0x3B with LSB.
The special features are declared by using the *dynOfs* and *dynLength*.

@param atIndex

Advertisement index is are reserved (value 0 - 3) or user specific (value 4 - 5).
3 -> iBeacon payload.
4 -> sBeacon payload

@param intervalSkip

See section *"1.1 OVERVIEW OF BR/EDR OPERATION"*.
IntervalSkip is related to the time slots which are configured by the transmission interval. Skip offers the possibility to transfer several advertisement payloads with a single bletag. A value of "2" would refer to use every second slot or "5" specifies each 5th slot.

@param hwAddr

Allows to define a custom bluetooth hardware address. A byte array with length 6 is required.

@param hwAddrPublic

Enables the hwAddr.

@param advType

Configures the advertisement type:
(1) Connectable -> BLE tag is able to connect
(2) None_Connectable -> BLE tag denies any connection attempt
(3) Scannable -> BLE tag is able to receive scan responses

@param txPower

Configuration of advertisement tx power:

```
lowest 4 bits -> 0:-40dBm
          *      1:-20dBm
          *      2:-16dBm
          *      3:-12dBm
          *      4:-8dBm
          *      5:-4dBm
          *      6: 0dBm
          *      7:+4dBm
 * next 4 bits -> enables channels: 0x10:ch37
          *                  0x20:ch38
          *                  0x40:ch39
          * example:
          * 0x76 -> ch37-39 are sending with 0dBm
```

@param kidx

Key index to point to a previous written key with the *writeAdvertisementKey* method. Protected keys must be have the highest bit set(0x80).

@param encrLength

Declares the length of the encryption data within the data byte array.

@param encrOfs

Declares the start of the encryption within the data byte array.

@param rlen

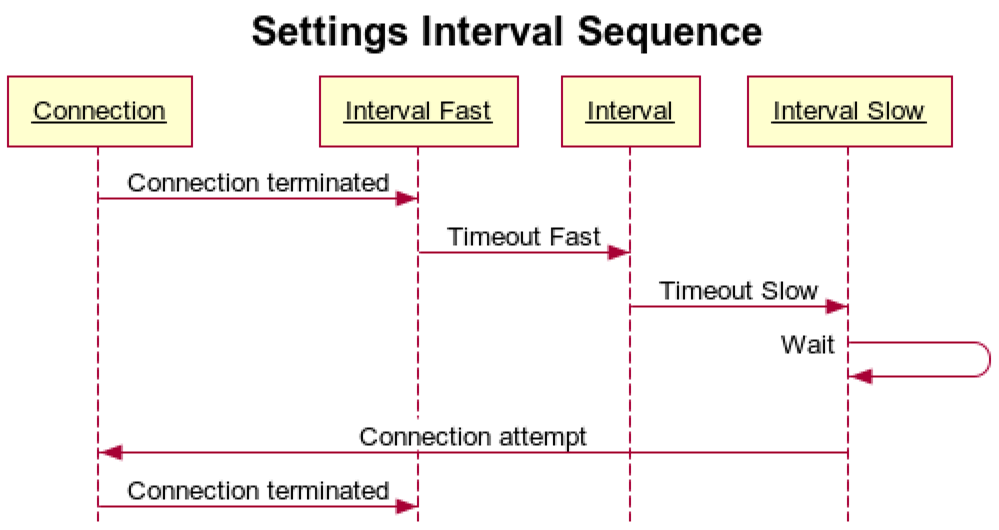
Amount of data used for checksum to verify successful decryption process

3.2 Advertisement Settings Configuration

StickNFind BLE tag support a wide range of settings.

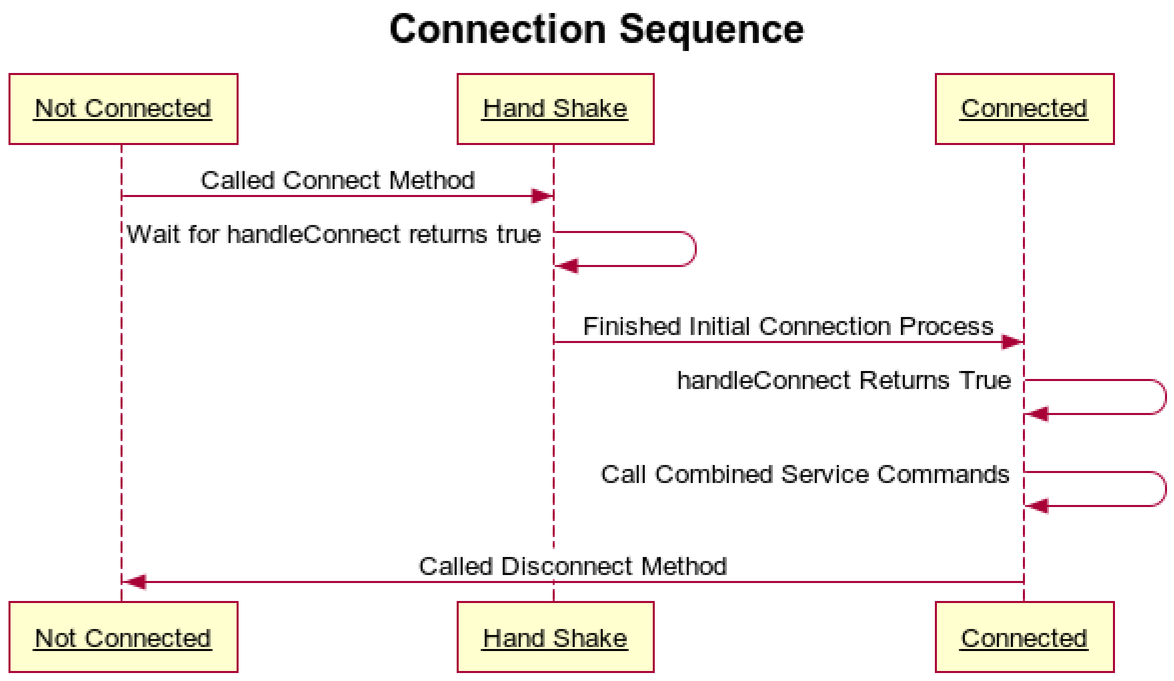
** @param interval_fast - 1.25ms steps*
** @param timeout_fast - max 3000 in s*
** @param interval - 1.25ms steps*
** @param timeout_slow - min 2000 in s - 0 disables timeout_slow*
** @param interval_slow - 1.25ms steps*
** @param channel_txpower_noconn - set channel_txpower for no connectable advertisement*
** @param channel_txpower_conn - set channel_txpower for connectable advertisement*

The sequence is the following:



4. Examples

The SDK is handling the connection attempt and will inform you with the *didReadDeviceId* callback when the initial handshake is completed.
Additional you can monitor the result have the *handleConnect* method.



Please follow above sequence to ensure connection with BLE tag.

4.1 Configure Advertisement Data

```
byte adv_template[] =
{
            0x02,0x01,0x06,
            0x03,0x02,0x04,0x19,
            0x17,(byte) 0xFF,(byte) 0xF9,0x00,

            0x13,0x01,0x02,0x03,0x04,

            LeSnfDeviceBroadcastV2dynSystemTime,
            LeSnfDeviceBroadcastV2dynSystemTime+1,
            LeSnfDeviceBroadcastV2dynSystemTime+2,
            LeSnfDeviceBroadcastV2dynSystemTime+3,
            LeSnfDeviceBroadcastV2dynScanDeviceCount,
            LeSnfDeviceBroadcastV2dynBatteryVoltage,
            LeSnfDeviceBroadcastV2dynTemperature,
            LeSnfDeviceBroadcastV2dynRssiCalibrationChId,
            0x00,0x00,0x00,0x00,0x00,0x00,0x00
};

setAdvertisementData_v2(   adv_template,
                           0x4,         //advertisement index
                           2,           //skip 1 slot
                           null,        //no specific bluetooth hardware addres
                           false,       //publish bluetooth hardware is off
                           LeSnfDevice.ADV_TYPE.ADV_CONNECTABLE,
                           0x75,        //use channel 0x37-0x39 with -4dBm transmission
                           16,          //dynamic payload offset
                           8,           /dynamic payload length
                           0x4,         //key index
                           12,          //encryption length
                           12,          //encryption offset
                           7);          //amount of data used for checksum
```

That's a typical example of generating an sBeacon payload.
The data array starts with:
(1) 0x02, 0x01, 0x6 -> two bytes data with the type advertisement flag and configuring a discoverable behavior
(2) 0x03, 0x02, 0x04, 0x19 -> three bytes of data for a service uuid with 16bit and more. The service id is 0x1904
(3) 0x17,(byte) 0xFF,(byte) 0xF9,0x00,
    0x13,0x01,0x02,0x03,0x04, -> 23 bytes follow for manufacture data follow
                               -> first 7 bytes are static
(4) Following 8 bytes are dynamic

```
   LeSnfDeviceBroadcastV2dynSystemTime,            //1. byte of system time
   LeSnfDeviceBroadcastV2dynSystemTime+1,          //2. byte of system time
   LeSnfDeviceBroadcastV2dynSystemTime+2,          //3. byte of system time
   LeSnfDeviceBroadcastV2dynSystemTime+3,          //4. byte of system time
   LeSnfDeviceBroadcastV2dynScanDeviceCount,       //Amount of ble devices detected
   LeSnfDeviceBroadcastV2dynBatteryVoltage,        //Battery voltage
   LeSnfDeviceBroadcastV2dynTemperature,           //Temperature
   LeSnfDeviceBroadcastV2dynRssiCalibrationChId    //Channel calibration value
```

4.2 Read Advertisement Data

The read advertisement data feature allows verification of previous executed write advertisement function.

Please find below an example:

public boolean readAdvertisementData(int index)

The index refers to the atIndex of the writeAdvertisementData_v2. 0 - 3 is ble specific and 4 - 5 user specific.
A successful read results in

didReadBroadcastData(LeDevice dev, byte [] data).

The LeDevice object refers to the associated bletag with the 31 byte array.