# s112_nrf5x migration document

## Introduction to the s112_nrf5x migration document

### About the document

This document describes how to migrate to new versions of the s112_nrf52 SoftDevices. The s112_nrf52 release notes should be read in conjunction with this document.

For each version, we have the following sections:

- "Required changes" describes how an application would have used the previous version of the SoftDevice and how it must now use this version for the given change.
- "New functionality" describes how to use new features and functionality offered by this version of the SoftDevice. **Note:** Not all new functionality may be covered; the release notes will contain a full list of new features and functionality.

Each section describes how to migrate to a given version from the previous version. If you are migrating to the current version from the previous version, follow the instructions in that section. To migrate between versions that are more than one version apart, follow the migration steps for all intermediate versions in order.

**Example:** To migrate from version 5.0.0 to version 5.2.0, first follow the instructions to migrate to version 5.1.0 from version 5.0.0, then follow the instructions to migrate to version 5.2.0 from version 5.1.0.

# s112_nrf52_6.0.0

This section describes how to migrate to s112_nrf52_6.0.0 from s132_nrf52_5.1.0 (which is API compatible with s112_nrf52810_5.1.0).

Notes:

- s112_nrf52_6.0.0 has changed the API compared to s132_nrf52_5.1.0 which requires applications to be recompiled.
- s112_nrf52_6.0.0 includes some features that are not Bluetooth qualified. For more information, see the release notes.

## New functionality

### Write to SoftDevice protected registers

A new API, `sd_protected_register_write()`, has been added to give the application the possibility to write to a register that is write-protected by the SoftDevice. A write-protected peripheral shall only be accessed through the SoftDevice API when the SoftDevice is enabled.

The new API supports writing to the Block Protection (`BPROT`) peripheral. The application can use `sd_protected_register_write()` ins tead of `sd_flash_protect()` to set the flash protection configuration registers.

**Usage**

```
/* Old API: */
errcode = sd_flash_protect(value0, value1, value2, value3)

/* New API: */
errcode = sd_protected_register_write(&(NRF_BPROT->CONFIG0), value0)
errcode = sd_protected_register_write(&(NRF_BPROT->CONFIG1), value1)
errcode = sd_protected_register_write(&(NRF_BPROT->CONFIG2), value2)
errcode = sd_protected_register_write(&(NRF_BPROT->CONFIG3), value3)
```

## Required changes

### Updated advertiser API

`sd_ble_gap_adv_data_set()` has been removed.

A new API, `sd_ble_gap_adv_set_configure()`, has been added with the following functionalities:

- Configuring and updating the advertising parameters of an advertising set.
- Setting, clearing, or updating advertising and scan response data.

  Note: The advertising data must be kept alive in memory until advertising is terminated. Not doing so will lead to undefined behavior.
  Note: Updating advertising data while advertising can only be done by providing new advertising data buffers.

#### Configuring and updating an advertising set

*Advertising Set* is a term introduced in Bluetooth Core Specification v5.0.

Each advertising set is identified by an advertising handle. To configure a new advertising set and obtain a new advertising handle, `sd_ble_gap_adv_set_configure()` should be called with a pointer `p_adv_handle` pointing to an advertising handle set to `BLE_GAP_ADV_SET_HANDLE_NOT_SET`.

To update an existing advertising set, `sd_ble_gap_adv_set_configure()` should be called with a previously configured advertising handle.

Note: Currently only one advertising set can be configured in the SoftDevice.

## Configuring advertising parameters for an advertising set

Setting advertising parameters has been moved from `sd_ble_gap_adv_start()` to `sd_ble_gap_adv_set_configure()`.

The content of `ble_gap_adv_params_t` has changed:

- `ble_gap_adv_params_t::type` has been removed.
- A new parameter, `properties`, of the new type `ble_gap_adv_properties_t` has been added.
  - The advertising type must now be set through `ble_gap_adv_properties_t::type`.
  - The advertising type definitions (`BLE_GAP_ADV_TYPES`) have changed, and new types have been added.
    The mapping from old to new advertising types is shown below. These advertising types are referred to as *legacy* advertising types:
    - `type = BLE_GAP_ADV_TYPE_ADV_IND -> properties.type = BLE_GAP_ADV_TYPE_CONNECTABLE_SCANNABLE_UNDIRECTED`
    - `type = BLE_GAP_ADV_TYPE_ADV_DIRECT_IND -> properties.type = BLE_GAP_ADV_TYPE_CONNECTABLE_NONSCANNABLE_DIRECTED_HIGH_DUTY_CYCLE` or `BLE_GAP_ADV_TYPE_CONNECTABLE_NONSCANNABLE_DIRECTED`
    - `type = BLE_GAP_ADV_TYPE_ADV_SCAN_IND -> properties.type = BLE_GAP_ADV_TYPE_NONCONNECTABLE_SCANNABLE_UNDIRECTED`
    - `type = BLE_GAP_ADV_TYPE_ADV_NONCONN_IND -> properties.type = BLE_GAP_ADV_TYPE_NONCONNECTABLE_NONSCANNABLE_UNDIRECTED`
- `ble_gap_adv_params_t::fp` has been renamed `ble_gap_adv_params_t::filter_policy`.
- `ble_gap_adv_params_t::timeout` has been renamed `ble_gap_adv_params_t::duration` and is now measured in 10 ms units.
- `ble_gap_adv_params_t::channel_mask` type has been changed from `ble_gap_adv_ch_mask_t` to the new type `ble_gap_ch_mask_t`.
  - Note: At least one of the primary channels that is channel index 37-39 must be set to 0.
  - Note: Masking away secondary channels is currently not supported.
  - The mapping from old type `ble_gap_adv_ch_mask_t` to the new type `ble_gap_ch_mask_t` is shown below:
    - `channel_mask.ch_37_off = 1` -> `channel_mask = 0x2000000000`
    - `channel_mask.ch_38_off = 1` -> `channel_mask = 0x4000000000`
    - `channel_mask.ch_39_off = 1` -> `channel_mask = 0x8000000000`
- `ble_gap_adv_params_t` has several new parameters:
  - `max_adv_evts` has been added to allow the application to advertise for a given number of advertising events.
  - `scan_req_notification` flag has been added to give the application the possibility to receive events of type `ble_gap_evt_scan_req_report_t`. This replaces `BLE_GAP_OPT_SCAN_REQ_REPORT`.
  - `primary_phy` and `secondary_phy` allow the application to select PHYs for primary and secondary advertising channels.
    - `primary_phy` should be set to `BLE_GAP_PHY_AUTO` or `BLE_GAP_PHY_1MBPS` for legacy advertising types. For extended advertising types, it should be set to `BLE_GAP_PHY_1MBPS` or `BLE_GAP_PHY_CODED` if supported by the SoftDevice.
    - `secondary_phy` can be ignored for legacy advertising. For extended advertising types, it should be set to `BLE_GAP_PHY_1MBPS`, `BLE_GAP_PHY_2MBPS`, or `BLE_GAP_PHY_CODED` if supported by the SoftDevice.
  - `set_id` has been added to allow the application to choose the set ID of an extended advertiser.

#### Other Advertising API changes

- `BLE_GAP_TIMEOUT_SRC_ADVERTISING` has been removed.
  - A new event, `BLE_GAP_EVT_ADVERTISING_SET_TERMINATED` with structure `ble_gap_evt_adv_set_terminated_t`, has been introduced to let the application know when and why an advertising set has terminated.
- A new configuration parameter, `ble_gap_cfg_role_count_t::adv_set_count`, has been introduced to set the maximum number of advertising sets.
  Note: The maximum number of supported advertising sets is `BLE_GAP_ADV_SET_COUNT_MAX`.
- `BLE_GAP_ADV_MAX_SIZE` has been replaced with `BLE_GAP_ADV_SET_DATA_SIZE_MAX`.
- `ble_gap_evt_connected_t` now includes `adv_handle` and `adv_data` of the new type `ble_gap_adv_data_t`. These are set when the device connects as a peripheral.
- `ble_gap_evt_scan_req_report_t` now includes `adv_handle`.
- `BLE_GAP_OPT_SCAN_REQ_REPORT` has been removed.
- `BLE_GAP_ADV_TIMEOUT_LIMITED_MAX` has been changed from 180 to 18000 as `sd_ble_gap_adv_params_t::duration` is now measured in 10 ms units.

#### Usage

```
static uint8_t raw_adv_data_buffer1[BLE_GAP_ADV_SET_DATA_SIZE_MAX];
static uint8_t raw_scan_rsp_data_buffer1[BLE_GAP_ADV_SET_DATA_SIZE_MAX];
static ble_gap_adv_data_t adv_data1 = {.adv_data.p_data      =
```

```
raw_adv_data_buffer1,      .adv_data.len      = sizeof
(raw_adv_data_buffer1),
                                        .scan_rsp_data.p_data =
raw_scan_rsp_data_buffer1, .scan_rsp_data.len = sizeof
(raw_scan_rsp_data_buffer1)};

/* A second advertising data buffer for later updating advertising data
while advertising */
static uint8_t raw_adv_data_buffer2[BLE_GAP_ADV_SET_DATA_SIZE_MAX];
static uint8_t raw_scan_rsp_data_buffer2[BLE_GAP_ADV_SET_DATA_SIZE_MAX];
static ble_gap_adv_data_t adv_data2 = {.adv_data.p_data      =
raw_adv_data_buffer2,      .adv_data.len      = sizeof
(raw_adv_data_buffer2),
                                        .scan_rsp_data.p_data =
raw_scan_rsp_data_buffer2, .scan_rsp_data.len = sizeof
(raw_scan_rsp_data_buffer2)};

int main(void)
{
  uint8_t adv_handle = BLE_GAP_ADV_SET_HANDLE_NOT_SET;
  ble_gap_adv_params_t adv_params = {.properties={.
type=BLE_GAP_ADV_TYPE_CONNECTABLE_SCANNABLE_UNDIRECTED},
                                    .interval              =
BLE_GAP_ADV_INTERVAL_MAX,
                                    .duration              =
BLE_GAP_ADV_TIMEOUT_LIMITED_MAX,
                                    .channel_mask          = {0}, /*
Advertising on all the primary channels */
                                    .max_adv_evts          = 0,
                                    .filter_policy         =
BLE_GAP_ADV_FP_ANY,
                                    .primary_phy           =
BLE_GAP_PHY_AUTO,
                                    .scan_req_notification = 1
                                   };
  /* Enable the BLE Stack */
  sd_ble_enable(...);

  [...]
  sd_ble_gap_adv_set_configure(&adv_handle, &adv_data1, &adv_params);
  /* Start advertising */
  sd_ble_gap_adv_start(adv_handle, BLE_CONN_CFG_TAG_DEFAULT);

  [...]
  /* Update advertising data while advertising */
  sd_ble_gap_adv_set_configure(&adv_handle, &adv_data2, NULL);

  [...]
  /* Stop advertising */
  sd_ble_gap_adv_stop(adv_handle);
```

```
    [...]
}
```

## Updated RSSI API

The RSSI API has been changed so that the SoftDevice can provide the application with the channel index on which the reported RSSI measurements are made.

- `sd_ble_gap_rssi_get()` takes an additional parameter `p_ch_index`. For this parameter, provide a pointer to a location where the channel index for the RSSI measurement should be stored.
- The event structure for the `BLE_GAP_EVT_RSSI_CHANGED` event has a new parameter `ble_gap_evt_rssi_changed_t::ch_index`. This is the Data Channel Index (0-36) on which the RSSI is measured.
- The event structure for the `BLE_GAP_EVT_ADV_REPORT` event has a new parameter `ble_gap_evt_adv_report_t::ch_index`. This is the Channel Index (0-39) on which the last advertising packet is received. The corresponding measured RSSI for this packet can be read from `ble_gap_evt_adv_report_t::rssi`.

## TX power API

The TX power API now supports setting individual transmit power for each link or role.

- `sd_ble_gap_tx_power_set()` takes two new parameters, `role` and `handle`, in addition to `tx_power`. For available roles and TX power values, see ble_gap.h.

## Updated Flash API

`sd_flash_write()` now triggers a HardFault if the application tries to write to a protected page. `NRF_ERROR_FORBIDDEN` is returned if the application tries to write to a page outside application flash area.

`sd_flash_page_erase()` now triggers a HardFault if the application tries to erase a protected page. `NRF_ERROR_FORBIDDEN` is returned if the application tries to erase a page outside application flash area.