

s14x_nrf5x migration document

Introduction to the s140_nrf52840 migration document

About the document

This document describes how to migrate to new versions of the s140 SoftDevices. The s140_nrf52840 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 5.1.0 from 5.0.0, then follow the instructions to migrate to 5.2.0 from 5.1.0.

Copyright (c) Nordic Semiconductor ASA. All rights reserved.

s140_nrf52840_5.0.0-3.alpha

This section describes how to migrate to s140_nrf52840_5.0.0-3.alpha from s140_nrf52840_5.0.0-2.alpha or s132_nrf52832_4.0.2.

New functionality

New Configuration API

A new configuration option, `BLE_GAP_CFG_ADV`, has been added to the `sd_ble_cfg_set()`. This option can be used to configure advertising sets. Currently this option is not used as this alpha release only supports one advertising set with 31 bytes of advertising or scan response data.

New defines and structures

Some new defines and structures have been added to `ble_gap.h`

```
/** @brief Default advertising and scan response max length. */
#define BLE_GAP_ADV_SR_MAX_LEN_DEFAULT (31)

/** @brief Maximum advertising or scan response data length. */
#define BLE_GAP_ADV_SR_MAX_DATA_LEN (1650)

/** @brief Maximum fragmentation size of an advertising or scan response packet. */
#define BLE_GAP_ADV_SR_MAX_FRAGMENTATION_SIZE (255)

/** @brief Default advertising set handle.
 *
 * Default advertising set handle. This handle identifies the default advertising
 * set,
```

```

* and shall be used when the application has not configured any custom advertising
sets.
* @sa ble_gap_cfg_adv_config_t */
#define BLE_GAP_ADV_SET_HANDLE_DEFAULT (0)

/** @brief Advertising set handle not set.
*
* Advertising set handle not set. If an additional advertising handle is required
this have to be set
* to configure additional advertising sets. @sa ble_gap_cfg_adv_config_t */
#define BLE_GAP_ADV_SET_HANDLE_NOT_SET (0xFF)

/**@defgroup BLE_GAP_ADV_DATA_STATUS GAP Advertising data status
* @{ */
#define BLE_GAP_ADV_DATA_STATUS_COMPLETE 0x00 /**< All data in the
advertising event have been received. */
#define BLE_GAP_ADV_DATA_STATUS_INCOMPLETE_MORE_DATA 0x01 /**< More data to be
received. */
#define BLE_GAP_ADV_DATA_STATUS_INCOMPLETE_TRUNCATED 0x02 /**< Missing data, no
more to be received. */
/**@} */

/**@defgroup BLE_GAP_SCAN_FILTER_POLICIES GAP Scanner filter policies
* @{ */
#define BLE_GAP_SCAN_FP_ACCEPT_ALL 0x00 /**< Accept all
advertising packets except directed advertising packets not addressed to this
device. */
#define BLE_GAP_SCAN_FP_WHITELIST 0x01 /**< Accept
advertising packets from devices in the whitelist except directed advertising
packets not addressed to this device. */
#define BLE_GAP_SCAN_FP_ALL_NOT_RESOLVED_DIRECTED 0x02 /**< Accept all
advertising packets specified in @ref BLE_GAP_SCAN_FP_ACCEPT_ALL. In addition,
accept directed advertising packets,
where the
initiator's address is a resolvable private address that cannot be resolved. */
#define BLE_GAP_SCAN_FP_WHITELIST_NOT_RESOLVED_DIRECTED 0x03 /**< Accept all
advertising packets specified in @ref BLE_GAP_SCAN_FP_WHITELIST. In addition,
accept directed advertising packets,
where the
initiator's address is a resolvable private address that cannot be resolved. */
/**@} */

/**@defgroup BLE_GAP_SCAN_DUPLICATES_POLICIES GAP Scanner filter duplicates
policies.
* @{ */
#define BLE_GAP_SCAN_DUPLICATES_REPORT 0x00 /**< Duplicate filtering
disabled. */
#define BLE_GAP_SCAN_DUPLICATES_SUPPRESS 0x01 /**< Duplicate filtering
enabled. */
#define BLE_GAP_SCAN_DUPLICATES_ONCE_PER_PERIOD 0x02 /**< Duplicate filtering
enabled, reset for each scan period. */
/**@} */

/**@brief Advertising event properties. */

```

```

typedef struct
{
    uint16_t connectable : 1; /**< Connectable advertising event. */
    uint16_t scannable : 1; /**< Scannable advertising event. */
    uint16_t directed : 1; /**< Directed advertising event. */
    uint16_t high_duty : 1; /**< High duty cycle directed advertising. Only
applicable for directed advertising event using legacy PDUs. */
    uint16_t legacy_pdu : 1; /**< Advertise using legacy advertising PDUs. @note If
ble_gap_cfg_adv_config_t::use_adv_ext has not been configured
                                on the advertising handle corresponding to this
advertising set, then legacy_pdu shall be set to 1.*/
    uint16_t anonymous : 1; /**< Omit advertiser's address from all PDUs. */
    uint16_t tx_power : 1; /**< Include TxPower in the extended header of the
advertising PDU. */
    uint16_t reserved : 9; /**< Reserved for future use. */
} ble_gap_adv_properties_t;

/**@brief Advertising report type. */
typedef struct
{
    uint16_t connectable : 1; /**< Connectable advertising event type. */
    uint16_t scannable : 1; /**< Scannable advertising event type. */
    uint16_t directed : 1; /**< Directed advertising event type. */
    uint16_t scan_response : 1; /**< Scan response. */
    uint16_t legacy_pdu : 1; /**< Legacy advertising PDU. */
    uint16_t status : 2; /**< Data status. See @ref BLE_GAP_ADV_DATA_STATUS.
*/
    uint16_t reserved : 9; /**< Reserved for future use. */
} ble_gap_adv_report_type_t;

/**
 * @brief Configuration of an advertising set, set with @ref sd_ble_cfg_set.
 *
 * @note This configuration can be set multiple times, and each time it will
reserve memory required for the advertising configuration. If adv_handle
 * has been set to @ref BLE_GAP_ADV_SET_HANDLE_NOT_SET, it will return a new
advertising set handle. The first call to this function will replace
 * the default advertising configuration. If the adv_handle has been set to
something other than @ref BLE_GAP_ADV_SET_HANDLE_NOT_SET then the
 * advertising configuration will be updated to the maximum size required
between those subsequent calls.
 * The default advertising configuration handle is @ref
BLE_GAP_ADV_SET_HANDLE_DEFAULT with @ref BLE_GAP_ADV_SR_MAX_LEN_DEFAULT.
 *
 * @retval ::NRF_ERROR_INVALID_PARAM Invalid parameters.
 */
typedef struct
{
    uint8_t *p_adv_handle; /**< Pointer to store the advertising handle for
this configuration. */
    uint16_t adv_data_size; /**< Maximum advertising data size. If size is
larger than @ref BLE_GAP_ADV_SR_MAX_LEN_DEFAULT then advertising extension will be
used. */
    uint16_t scan_response_size; /**< Maximum scan response data size required. If
size is larger than @ref BLE_GAP_ADV_SR_MAX_LEN_DEFAULT then advertising extension
will be used. */
    uint8_t use_adv_ext:1; /**< If set, it configures the advertng set to use

```

```
advertising extension. */
} ble_gap_cfg_adv_config_t;

/**@brief Data structure. */
typedef struct
{
    uint8_t      *p_data;  /**< Pointer to the data provided to/from the application.
*/
```

```
uint16_t    len;    /**< Total length of the data. */
} ble_data_t;
```

Required changes

Updated advertising API

The define `BLE_GAP_ADV_NONCON_INTERVAL_MIN` has been removed.

The define `BLE_GAP_ADV_INTERVAL_MAX` has been increased from `0x4000` to `0xFFFFF`.

`ble_gap_scan_params_t::timeout` and `ble_gap_adv_params_t::timeout` have been renamed `ble_gap_scan_params_t::duration` and `ble_gap_adv_params_t::duration`, and their units have been changed from seconds to 10ms units.

`ble_gap_adv_params_t::type` has been changed to `ble_gap_adv_params_t::properties` and is of the new type `ble_gap_adv_properties_t`. To advertise with legacy packets, the advertising properties have to be configured as follows:

```
ble_gap_adv_params_t adv_params = {0};

// BLE_GAP_ADV_TYPE_ADV_IND
memset(&adv_params, 0, sizeof(adv_params));
adv_params.properties.connectable = 1;
adv_params.properties.scannable   = 1;
adv_params.properties.legacy_pdu  = 1;

//BLE_GAP_ADV_TYPE_ADV_DIRECT_IND
memset(&adv_params, 0, sizeof(adv_params));
adv_params.properties.connectable = 1;
adv_params.properties.directed    = 1;
adv_params.properties.legacy_pdu  = 1;

//BLE_GAP_ADV_TYPE_ADV_SCAN_IND
memset(&adv_params, 0, sizeof(adv_params));
adv_params.properties.scannable   = 1;
adv_params.properties.legacy_pdu  = 1;

//BLE_GAP_ADV_TYPE_ADV_NONCON_IND
memset(&adv_params, 0, sizeof(adv_params));
adv_params.properties.legacy_pdu  = 1;
```

`ble_gap_adv_params_t` has several new parameters:

```
/**@brief GAP advertising parameters. */

typedef struct
{
    ble_gap_addr_t const    *p_peer_addr;    /**< Address of a known peer.
                                             - When privacy is enabled
and the local device use @ref BLE_GAP_ADDR_TYPE_RANDOM_PRIVATE_RESOLVABLE
addresses, the device identity list is searched for a matching
                                             entry. If the local IRK
for that device identity is set, the local IRK for that device will be used to
```

generate the advertiser address field in the advertise packet.

- If @ref ble_gap_adv_properties_t::directed is set, this must be set to the targeted initiator. If the initiator is in the device identity list, the peer IRK for that device will be used to generate the initiator address field in the ADV_DIRECT_IND packet. */

ble_gap_adv_properties_t properties; /**< Advertising event properties. See @ref ble_gap_adv_properties_t. */

uint32_t interval; /**< Advertising interval. See @ref BLE_GAP_ADV_INTERVALS.

- If @ref ble_gap_adv_properties_t::directed and @ref ble_gap_adv_properties_t::high_duty, this parameter is ignored. */

uint16_t duration; /**< Advertising duration between 0x0001 and 0xFFFF in 10ms units. Setting the value to 0x0000 disables the timeout.

Advertising will be automatically stopped when the duration specified by this parameter (if not 0x0000) is reached. @sa BLE_GAP_ADV_TIMEOUT_VALUES.

@note If @ref ble_gap_adv_properties_t::directed and @ref ble_gap_adv_properties_t::high_duty are set, this parameter is ignored. */

uint8_t max_ext_adv; /**< Maximum extended advertising events that shall be sent prior to disabling the extended advertising. Setting the value to 0 disables the limitation.

Advertising will be automatically stopped when the count of extended advertising events specified by this parameter (if not 0) is reached.

@note If @ref ble_gap_adv_properties_t::directed and @ref ble_gap_adv_properties_t::high_duty are set, this parameter is ignored.

@note max_ext_adv will be ignored if @ref ble_gap_adv_properties_t::legacy_pdu is set.*/

ble_gap_adv_ch_mask_t channel_mask; /**< Advertising channel mask for the primary channels. See @ref ble_gap_adv_ch_mask_t. */

uint8_t fp; /**< Filter Policy, see @ref BLE_GAP_ADV_FILTER_POLICIES. */

uint8_t primary_phy; /**< Indicates the PHY on which the advertising packets are transmitted on the primary advertising channel. See @ref BLE_GAP_PHYS.

@note The primary_phy shall indicate @ref BLE_GAP_PHY_1MBPS if @ref ble_gap_adv_properties_t::legacy_pdu is set. */

uint8_t secondary_phy; /**< Indicates the PHY on which the advertising packets are transmitted on the secondary advertising channel. See @ref BLE_GAP_PHYS.

@note This is the PHY that will be used to create connection and send AUX_ADV_IND packets on. secondary_phy will be ignored when @ref ble_gap_adv_properties_t::legacy_pdu is set. */

uint8_t secondary_max_skip; /**< Maximum advertising events the controller can skip before sending the AUX_ADV_IND packets on the secondary channel.

@note secondary_max_skip will be ignored if @ref ble_gap_adv_properties_t::legacy_pdu is set. */

uint8_t advertising_sid:7; /**< Advertising Set ID to distinguish between advertising data transmitted by this device. @note

```
advertising_sid will be ignored if @ref ble_gap_adv_properties_t::legacy_pdu is
set. */
uint8_t          scan_req_notification:1; /**< Enable scan request
notifications for this advertising set. */
uint8_t          adv_fragmentation_len;    /**< Maximum PDU length of
advertising and scan response packets. If set to 0 @ref
BLE_GAP_ADV_SR_MAX_FRAGMENTATION_SIZE will be used.
```

```
@note adv_fragmentation_len
will be ignored if @ref ble_gap_adv_properties_t::legacy_pdu is set.*/
} ble_gap_adv_params_t;
```

The `ble_gap_adv_params_t::primary_phy` has to be set to `BLE_GAP_PHY_1MBPS` for legacy advertising. It can be set to `BLE_GAP_PHY_1MBPS` or `BLE_GAP_PHY_CODED` for extended advertising.

The `ble_gap_adv_params_t::secondary_phy` can be ignored for legacy advertising. It can be set to `BLE_GAP_PHY_1MBPS`, `BLE_GAP_PHY_2MBPS`, or `BLE_GAP_PHY_CODED` for extended advertising.

The following fields are not used in this alpha and should be set to 0:

- `ble_gap_adv_params_t::max_ext_adv`
- `ble_gap_adv_params_t::secondary_max_skip`
- `ble_gap_adv_params_t::advertising_sid`
- `ble_gap_adv_params_t::scan_req`
- `ble_gap_adv_params_t::fragmentation_len`

Updated scanning and connection API

`ble_gap_scan_params_t` has received some new parameters. `ble_gap_scan_params_t::use_whitelist` and `ble_gap_scan_params_t::adv_dir_report` have been combined into `ble_gap_scan_params_t::filter_policy` which should be set to a value from `BLE_GAP_SCAN_FILTER_POLICIES`.

```
/**@brief GAP scanning parameters. */
typedef struct
{
    uint8_t active          : 1; /**< If 1, perform active scanning (scan
requests). */
    uint8_t filter_policy   : 2; /**< Scanning filter policy. See @ref
BLE_GAP_SCAN_FILTER_POLICIES. */
    uint8_t filter_duplicates: 2; /**< Filter duplicates. @ref
BLE_GAP_SCAN_DUPLICATES_POLICIES. */
    uint8_t scan_phy;       /**< PHY to scan on. See @ref BLE_GAP_PHYS. */
    uint16_t interval;      /**< Scan interval. See @ref
BLE_GAP_SCAN_INTERVALS. */
    uint16_t window;        /**< Scan window. See @ref BLE_GAP_SCAN_WINDOW.
*/
    uint16_t duration;      /**< Duration of a scanning session in units of
10ms. Range: 0x0001 - 0xFFFF (10ms to 10.9225m). If set to 0x0000, scanning will
continue until it is explicitly disabled. @sa sd_ble_gap_connect @sa
sd_ble_gap_scan_stop */
    uint16_t period;        /**< Time interval between two subsequent
scanning sessions in units of 1.28s. Range: 0x0001 - 0xFFFF (1.28s - 83,884.8s).
If @ref ble_gap_scan_params_t::duration is
not 0x0000, the time specified by Period must be larger than the time
specified by @ref
ble_gap_scan_params_t::duration. If Period is set to 0x0000, scanning will
automatically end after the time specified by Duration is expired. */
} ble_gap_scan_params_t;
```

`ble_gap_scan_params_t::scan_phy` has to be set to either `BLE_GAP_PHY_1MBPS` or `BLE_GAP_PHY_CODED`. `ble_gap_scan_params_t::period` and `ble_gap_scan_params_t::filter_duplicates` are not used in this alpha and shall be set to 0.

The defines `BLE_GAP_SCAN_INTERVAL_MAX` and `BLE_GAP_SCAN_WINDOW_MAX` have been increased from `0x4000` to `0xFFFF`.

`ble_gap_adv_report_t` has been modified and has some new parameters.


```

/**@brief Event structure for @ref BLE_GAP_EVT_ADV_REPORT. */
typedef struct
{
    ble_gap_adv_report_type_t type;                /**< Advertising
report type. See @ref ble_gap_adv_report_type_t. */
    ble_gap_addr_t peer_addr;                    /**< Bluetooth
address of the peer device. If the peer_addr resolved: @ref
ble_gap_addr_t::addr_id_peer is set to 1
                                                    and the
address is the device's identity address. */
    ble_gap_addr_t direct_addr;                  /**< Set when the
scanner is unable to resolve the private resolvable address of the initiator field
of a directed advertisement
                                                    packet and
the scanner has been enabled to report this with either @ref
BLE_GAP_SCAN_FP_ALL_NOT_RESOLVED_DIRECTED, or @ref
BLE_GAP_SCAN_FP_WHITELIST_NOT_RESOLVED_DIRECTED. */
    uint8_t primary_phy;                        /**< Indicates
the PHY on which the advertising packets are received on the primary advertising
channel. See @ref BLE_GAP_PHYS. */
    uint8_t secondary_phy;                      /**< Indicates
the PHY on witch the advertising packets are received on the secondary advertising
channel. See @ref BLE_GAP_PHYS. */
    uint16_t periodic_interval;                  /**< If periodic
advertising exists, as part of this advertising set, the periodic_interval
specifies the interval of the periodic advertising,
                                                    in 1.25ms
units. If set to 0, it indicates that no periodic advertising exists as part of
this set. */
    int8_t tx_power;                            /**< TX Power
reported by the advertiser. */
    int8_t rssi;                                /**< Received
Signal Strength Indication in dBm. */
    uint8_t set_id;                             /**< Set ID of
received advertising report. */
    uint8_t dlen;                               /**< Advertising
or scan response data length. */
    uint8_t data[BLE_GAP_ADV_SR_MAX_LEN_DEFAULT]; /**< Advertising
or scan response data. */
} ble_gap_evt_adv_report_t;

```

`ble_gap_adv_report_t::type` has been changed from `uint8_t` to `ble_gap_adv_report_type_t`. If `ble_gap_adv_report_type_t::legacy_pdu` is set, then the following parameters can be ignored:

- `ble_gap_adv_report_t::secondary_phy` (will be set to be `BLE_GAP_PHY_NOT_SET` if `legacy_pdu` is set)
- `ble_gap_adv_report_t::periodic_interval` (currently not supported)
- `ble_gap_adv_report_t::tx_power` (currently not supported, set to 0x7F)
- `ble_gap_adv_report_t::set_id` (currently not supported)

`sd_ble_gap_adv_data_set` has been changed to expect an advertising handle in addition to two `ble_data_t` structures.

Usage:

```
uint8_t adv_array[] = {<advertising data>;
ble_data_t adv_data = {.p_data=adv_array, .len=sizeof(adv_array)};

uint8_t sr_array[] = {<scan response data>;
ble_data_t sr_data = {.p_data=sr_array, .len=sizeof(sr_array)};

uint32_t errcode = sd_ble_gap_adv_data_set(BLE_GAP_ADV_SET_HANDLE_DEFAULT,
&adv_data, &sr_data);
```

`sd_ble_gap_adv_start` and `sd_ble_gap_adv_stop` now expect an advertising handle as the first argument, and currently it should be set to `BLE_GAP_ADV_SET_HANDLE_DEFAULT`.

Clock configuration rename.

`nrf_clock_lf_cfg_t::xtal_accuracy` has been renamed `nrf_clock_lf_cfg_t::accuracy`, and the following defines have been renamed:

Old Define	New Define
<code>NRF_CLOCK_LF_XTAL_ACCURACY_250_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_250_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_500_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_500_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_150_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_150_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_100_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_100_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_75_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_75_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_50_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_50_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_30_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_30_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_20_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_20_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_10_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_10_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_5_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_5_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_2_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_2_PPM</code>
<code>NRF_CLOCK_LF_XTAL_ACCURACY_1_PPM</code>	<code>NRF_CLOCK_LF_ACCURACY_1_PPM</code>

s140_nrf52840_5.0.0-2.alpha

This section describes how to migrate to s140_nrf52840_5.0.0-2.alpha from s140_nrf52840_5.0.0-1.alpha.

Required changes

SoftDevice RAM usage

The RAM usage of the SoftDevice has changed. `sd_ble_enable()` should be used to find the `APP_RAM_BASE` for a particular configuration.

New configuration API

Configuration parameters passed to `sd_ble_enable()` have been moved to the SoftDevice configuration API.

API updates

- A new SV call `sd_ble_cfg_set()` is added to set the configuration. This API can be called many times to configure different parts of the BLE stack. All configurations are optional. Configuration parameters not set by this API will take their default values.
- The SV call parameter `ble_enable_params_t * p_ble_enable_params` is removed from `sd_ble_enable()`. The SV call `sd_ble_cfg_set()` must be used instead. The parameters of this call are given in the following table:

Old API: <code>ble_enable_params_t</code> member	New API: <code>cfg_id</code> in <code>sd_ble_cfg_set()</code>
<code>common_enable_params.vs_uuid_count</code>	<code>BLE_COMMON_CFG_VS_UUID</code>
<code>common_enable_params.p_conn_bw_counts</code>	<code>BLE_CONN_CFG_GAP (*)</code>
<code>gap_enable_params.periph_conn_count</code> <code>gap_enable_params.central_conn_count</code> <code>gap_enable_params.central_sec_count</code>	<code>BLE_GAP_CFG_ROLE_COUNT</code>
<code>gap_enable_params.p_device_name</code>	<code>BLE_GAP_CFG_DEVICE_NAME</code>
<code>gatt_enable_params</code>	<code>BLE_CONN_CFG_GATT (*)</code>
<code>gatts_enable_params.service_changed</code>	<code>BLE_GATTS_CFG_SERVICE_CHANGED</code>
<code>gatts_enable_params.attr_tab_size</code>	<code>BLE_GATTS_CFG_ATTR_TAB_SIZE</code>

(*) These configurations can be set per link.

Usage

Example pseudo code to set per link ATT_MTU using the new configuration API:

```
const uint16_t client_rx_mtu = 158;
const uint32_t long_att_conn_cfg_tag = 1;

/* set ATT_MTU for connections identified by long_att_conn_cfg_tag */
ble_cfg_t cfg;
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = long_att_conn_cfg_tag;
cfg.conn_cfg.params.gatt_conn_cfg.att_mtu = client_rx_mtu;
sd_ble_cfg_set(BLE_CONN_CFG_GATT, &cfg, ...);

/* Enable the BLE Stack */
sd_ble_enable(...);

[...]

uint16_t long_att_conn_handle;
/* Establish connection with long_att_conn_cfg_tag */
sd_ble_gap_adv_start(..., long_att_conn_cfg_tag);

[...]

/* Establish connection with BLE_CONN_CFG_TAG_DEFAULT, it will use default ATT_MTU
of 23 bytes */
sd_ble_gap_connect(..., BLE_CONN_CFG_TAG_DEFAULT);

[...]

/* Start ATT_MTU exchange */
sd_ble_gattc_exchange_mtu_request(long_att_conn_handle, client_rx_mtu);
```

BLE bandwidth configuration

The BLE bandwidth configuration and application packet concept has been changed. Previously, the application could specify a bandwidth setting, which would result in a given queue size and a corresponding given radio time allocated. Now the queue sizes and the allocated radio time have been separated. The application can now configure:

- Event length
- Write without response queue size
- Handle Value Notification queue size

These settings are configurable per link.

Note that now the configured queue sizes are not directly related to on-air bandwidth:

- The application can configure one single packet to be queued in the SoftDevice, but still achieve full throughput if the application can queue packets fast enough during connection events.
- Even if the application configures a large number of packets to be queued, not all of them will be sent during a single connection event if the configured event length is not large enough to send the packets.

API updates

- The `ble_enable_params_t::common_enable_params.p_conn_bw_counts` parameter of the `sd_ble_enable()` SV call is replaced by the `sd_ble_cfg_set()` SV call with `cfg_id` parameter set to `BLE_CONN_CFG_GAP`. The following table shows how the old bandwidth configuration corresponds to the new one for the default ATT_MTU:

Old API: <code>BLE_CONN_BWS</code>	New API: <code>ble_gap_conn_cfg_t::event_length</code> in <code>sd_ble_cfg_set()</code>
<code>BLE_CONN_BW_LOW</code>	<code>BLE_GAP_EVENT_LENGTH_MIN</code>
<code>BLE_CONN_BW_MID</code>	<code>BLE_GAP_EVENT_LENGTH_DEFAULT</code>
<code>BLE_CONN_BW_HIGH</code>	6

The bandwidth configuration is further described in the SDS.

- The `BLE_COMMON_OPT_CONN_BW` option is removed. Instead, during connection creation, the application should supply the `conn_cfg_tag` defined by the `ble_conn_cfg_t::conn_cfg_tag` parameter in the `sd_ble_cfg_set()` SV call.
- A new parameter `conn_cfg_tag` is added to `sd_ble_gap_adv_start()` and `sd_ble_gap_connect()` SV calls. To create a connection with a default configuration, `BLE_CONN_CFG_TAG_DEFAULT` should be provided in this parameter.
- The `BLE_EVT_TX_COMPLETE` event is split on two events: `BLE_GATTC_EVT_WRITE_CMD_TX_COMPLETE` and `BLE_GATTS_EVT_HVN_TX_COMPLETE`.
- The SV call `sd_ble_tx_packet_count_get()` is removed. Instead, the application can now configure packet counts per link, using the SV call `sd_ble_cfg_set()` with the `cfg_id` parameter set to `BLE_CONN_CFG_GATTC` and `BLE_CONN_CFG_GATTS`.

Usage

Example pseudo code to set configuration that corresponds to the old `BLE_CONN_BW_HIGH` bandwidth configuration both in throughput and packet queueing capability:

```

const uint32_t high_bw_conn_cfg_tag = 1;
ble_cfg_t cfg;

/* configure connections identified by high_bw_conn_cfg_tag */

/* set connection event length */
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = high_bw_conn_cfg_tag;
cfg.conn_cfg.params.gap_conn_cfg.event_length = 6; /* 6 * 1.25 ms = 7.5 ms
corresponds to the old BLE_CONN_BW_HIGH for default ATT_MTU */
cfg.conn_cfg.params.gap_conn_cfg.conn_count = 1; /* application needs one link
with this configuration */
sd_ble_cfg_set(BLE_CONN_CFG_GAP, &cfg, ...);

/* set HVN queue size */
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = high_bw_conn_cfg_tag;
cfg.conn_cfg.params.gatts_conn_cfg.hvn_tx_queue_size = 7; /* application wants to
queue 7 HVNs */
sd_ble_cfg_set(BLE_CONN_CFG_GATTS, &cfg, ...);

/* set WRITE_CMD queue size */
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = high_bw_conn_cfg_tag;
cfg.conn_cfg.params.gattc_conn_cfg.write_cmd_tx_queue_size = 0; /* application is
not going to send WRITE_CMD, so set to 0 to save memory */
sd_ble_cfg_set(BLE_CONN_CFG_GATTC, &cfg, ...);

/* Enable the BLE Stack */
sd_ble_enable(...);

[...]

uint16_t high_bw_conn_handle;
/* Establish connection with high_bw_conn_cfg_tag */
sd_ble_gap_adv_start(..., high_bw_conn_cfg_tag);

```

Data Length Update Procedure

The application now has to respond to the Data Length Update Procedure when initiated by the peer. See the description of the Data Length Update Procedure in the New functionality section for more details.

Required changes:

```

case BLE_GAP_EVT_DATA_LENGTH_UPDATE_REQUEST:
{
    /* Allow SoftDevice to choose Data Length Update Procedure parameters
    automatically. */
    sd_ble_gap_data_length_update(p_ble_evt->evt.gap_evt.conn_handle, NULL, NULL);
    break;
}
case BLE_GAP_EVT_DATA_LENGTH_UPDATE:
{
    /* Data Length Update Procedure completed, see
    p_ble_evt->evt.gap_evt.params.data_length_update.effective_params for negotiated
    parameters. */
    break;
}

```

Access to RAM[x].POWER registers

SoftDevice APIs are updated to provide access to the RAM[x].POWER registers instead of the deprecated RAMON/RAMONB.

API updates

- `sd_power_ramon_set()` SV call is replaced with `sd_power_ram_power_set()`.
- `sd_power_ramon_clr()` SV call is replaced with `sd_power_ram_power_clr()`.
- `sd_power_ramon_get()` SV call is replaced with `sd_power_ram_power_get()`.

API rename

Some APIs were renamed. Applications that use the old names must be updated.

API updates

- `BLE_EVTS_PTR_ALIGNMENT` is renamed to `BLE_EVT_PTR_ALIGNMENT`.
- `BLE_EVTS_LEN_MAX` is renamed to `BLE_EVT_LEN_MAX`.
- `GATT_MTU_SIZE_DEFAULT` is renamed to `BLE_GATT_ATT_MTU_DEFAULT`.
- The GAP option `BLE_GAP_OPT_COMPAT_MODE` is renamed to `BLE_GAP_OPT_COMPAT_MODE_1`.
- `ble_gap_opt_compat_mode_t` structure is renamed to `ble_gap_opt_compat_mode_1_t`.
- `ble_gap_opt_compat_mode_t::mode_1_enable` structure member is renamed to `ble_gap_opt_compat_mode_1_t::enable`.
- `ble_gap_opt_t::compat_mode` structure member is renamed to `ble_gap_opt_t::compat_mode_1`.

Proprietary L2CAP API removed

The proprietary API for sending and receiving data over L2CAP is removed.

API updates

- The SV calls `sd_ble_l2cap_cid_register()`, `sd_ble_l2cap_cid_unregister()`, and `sd_ble_l2cap_tx()` are removed.
- `BLE_L2CAP_EVT_RX` event is removed.
- The following defines are removed: `BLE_L2CAP_MTU_DEF`, `BLE_L2CAP_CID_INVALID`, `BLE_L2CAP_CID_DYN_BASE`, `BLE_L2CAP_CID_DYN_MAX`.

New functionality

Data Length Update Procedure

The application is given control of the Data Length Update Procedure. The application can initiate the procedure and has to respond when initiated by the peer.

API updates

- A new SV call `sd_ble_gap_data_length_update()` is added to initiate or respond to a Data Length Update Procedure.
- The `BLE_EVT_DATA_LENGTH_CHANGED` event is replaced with `BLE_GAP_EVT_DATA_LENGTH_UPDATE`.
- A new event `BLE_GAP_EVT_DATA_LENGTH_UPDATE_REQUEST` is added to notify that a Data Length Update request has been received. `sd_ble_gap_data_length_update()` must be called by the application after this event has been received to continue the Data Length Update Procedure.
- The GAP option `BLE_GAP_OPT_EXT_LEN` is removed. The `sd_ble_gap_data_length_update()` SV call should be used instead.

Usage

- The Data Length Update Procedure can be initiated locally or by peer device.
- Following is the pseudo code for the case where Data Length Update Procedure is initiated by application:

```
const uint16_t client_rx_mtu = 247;
const uint32_t long_att_conn_cfg_tag = 1;

/* ATT_MTU must be configured first */
ble_cfg_t cfg;
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = long_att_conn_cfg_tag;
cfg.conn_cfg.params.gatt_conn_cfg.att_mtu = client_rx_mtu;
sd_ble_cfg_set(BLE_CONN_CFG_GATT, &cfg, ...);

/* Enable the BLE Stack */
sd_ble_enable(...);

[...]

uint16_t long_att_conn_handle;
/* Establish connection */
sd_ble_gap_adv_start(..., long_att_conn_cfg_tag);

[...]

/* Start Data Length Update Procedure, can be done without ATT_MTU exchange */
ble_gap_data_length_params_t params = {
    .max_tx_octets = client_rx_mtu + 4,
    .max_rx_octets = client_rx_mtu + 4,
    .max_tx_time_us = BLE_GAP_DATA_LENGTH_AUTO,
    .max_rx_time_us = BLE_GAP_DATA_LENGTH_AUTO
};
sd_ble_gap_data_length_update(long_att_conn_handle, &params, NULL);

[...]

case BLE_GAP_EVT_DATA_LENGTH_UPDATE:
{
    /* Data Length Update Procedure completed, see
    p_ble_evt->evt.gap_evt.params.data_length_update.effective_params for negotiated
    parameters. */
    break;
}
```

New compatibility mode

A new compatibility mode is added to enable interoperability with central devices that may initiate version exchange and feature exchange control procedures in parallel. To enable this mode, use the `sd_ble_opt_set()` SV call with the `opt_id` parameter set to `BLE_GAP_OPT_COMPAT_MODE_2`.

Slave latency configuration

It is now possible to disable and enable slave latency on an active peripheral link. To disable or re-enable slave latency, use the `sd_ble_opt_set()` SV call with the `opt_id` parameter set to `BLE_GAP_OPT_SLAVE_LATENCY_DISABLE`.

Support for high accuracy LFCLK oscillator source

It is now possible to configure the SoftDevice with higher accuracy LFCLK oscillator source. Four new levels are defined:

```
#define NRF_CLOCK_LF_XTAL_ACCURACY_10_PPM (8) /**< 10 ppm */
#define NRF_CLOCK_LF_XTAL_ACCURACY_5_PPM (9) /**< 5 ppm */
#define NRF_CLOCK_LF_XTAL_ACCURACY_2_PPM (10) /**< 2 ppm */
#define NRF_CLOCK_LF_XTAL_ACCURACY_1_PPM (11) /**< 1 ppm */
```

New power failure levels

It is now possible to configure the SoftDevice with all the new power failure levels introduced in NRF52. Levels that are added:

```
NRF_POWER_THRESHOLD_V17 /**< Set the power failure threshold to 1.7 V. */
NRF_POWER_THRESHOLD_V18 /**< Set the power failure threshold to 1.8 V. */
NRF_POWER_THRESHOLD_V19 /**< Set the power failure threshold to 1.9 V. */
NRF_POWER_THRESHOLD_V20 /**< Set the power failure threshold to 2.0 V. */
NRF_POWER_THRESHOLD_V22 /**< Set the power failure threshold to 2.2 V. */
NRF_POWER_THRESHOLD_V24 /**< Set the power failure threshold to 2.4 V. */
NRF_POWER_THRESHOLD_V26 /**< Set the power failure threshold to 2.6 V. */
NRF_POWER_THRESHOLD_V28 /**< Set the power failure threshold to 2.8 V. */
```


s140_nrf52840_5.0.0-1.alpha

This section describes how to migrate to s140_nrf52840_5.0.0-1.alpha from s132_nrf52_3.0.0. This SoftDevice is designed to take advantage of the new features of the nrf52840 chip.

Required changes

SoftDevice flash and RAM usage

The size of the SoftDevice has changed and therefore a change to the application project file is required.

For Keil this means:

1. Go into the properties of the project and find the Target tab
2. Change IROM1 Start to `0x20000`.

If the project uses a scatter file or linker script instead, then these must be updated accordingly.

The RAM usage of SoftDevice has also changed. `sd_ble_enable()` should be used to find the APP_RAM_BASE for a particular configuration.

Renamed defines

Some defines have been renamed to make the API more consistent. Any code using these defines has to be updated with the new names:

- `GATT_MTU_SIZE_DEFAULT` renamed to `BLE_GATT_MTU_SIZE_DEFAULT`
- `BLE_EVTS_LEN_MAX` renamed to `BLE_EVT_LEN_MAX`
- `BLE_EVTS_PTR_ALIGNMENT` renamed to `BLE_EVT_PTR_ALIGNMENT`

New functionality

Multiple PHYs

The SoftDevice introduces support for using multiple PHYs to adapt the speed and reliability of data transmission to the channel capacity. For higher throughput, a 2 Mbps PHY is supported. For higher reliability, a 125kbps Coded PHY is supported.

API updates

- A new GAP option, `BLE_GAP_OPT_PREFERRED_PHYS_SET`, has been added to indicate to the controller about which PHYs the controller shall prefer so it can respond to any requests to update PHYs by peers.
- A new SV call, `sd_ble_gap_phy_request()`, has been added to request the controller to attempt to change to a new PHY.
- A new event, `BLE_GAP_EVT_PHY_UPDATE`, has been added to indicate that the PHY of a connection has changed or that a local initiated PHY update procedure has finished.

Usage

Example pseudo code for setting the preferred PHYs for new connections

Note: This will only have an effect if the peer device initiates the procedure to change the PHY. The stack will not initiate a PHY Update procedure autonomously.

```

ble_opt_t opts;
opts.gap_opt.preferred_phys.tx_phys = BLE_GAP_PHY_1MBPS | BLE_GAP_PHY_2MBPS;
opts.gap_opt.preferred_phys.rx_phys = BLE_GAP_PHY_1MBPS | BLE_GAP_PHY_2MBPS;
TEST_SD_UTIL_NRF_SUCCESS_OR_ASSERT(sd_ble_opt_set(BLE_GAP_OPT_PREFERRED_PHYS_SET,
&opts) );

[ Advertise and connect / Scan and connect ]

```

Request the controller to attempt to change to a new PHY for an established connection:

```

ble_gap_phys_t phys = {BLE_GAP_PHY_CODED, BLE_GAP_PHY_CODED};
sd_ble_gap_phy_request(conn_handle, &phys);

```

Handle PHY Update event:

```

/* Handle the event */
case BLE_GAP_EVT_PHY_UPDATE:
    if (ble_event.evt.gap_evt.params.phy_update.status == BLE_HCI_STATUS_CODE_SUCCESS)
    {
        // The PHY was changed (after either the application or the peer requested it)
        // ble_event.evt.gap_evt.params.phy_update.tx_phy and
        ble_event.evt.gap_evt.params.phy_update.rx_phy contain the new PHYs
    }
    else
    {
        // A PHY update was requested which could not be performed successfully
    }
}

```

Higher TX power on nRF52840

The SoftDevice now supports configuring higher TX power to be used with nRF52840.

The following additional values are supported by the `sd_ble_gap_tx_power_set()` SV-call +2dBm, +5dBm, +6dBm, +7dBm, +8dBm, +9dBm.

These power levels can be used in the same way the existing power levels are used in the s132_nrf52_3.0.0 SoftDevice.