s140_nrf52_7.2.0 migration document

Introduction to the s140_nrf52 migration document

About the document

This document describes how to migrate to new versions of the s140 SoftDevice. The s140_nrf52 release notes should be read in conjunction with this document.

For each version, we have the following sections:

- "Required changes" describes the changes that need to be done in the application when migrating from an older version of the SoftDevice.
- "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.

s140_nrf52_7.2.0

This section describes how to use the new features of s140_nrf52_7.2.0 when migrating from s140_nrf52_7.0.1. As with all minor releases, the s140_nrf52_7.2.0 is binary compatible with s140_nrf52_7.0.1.

New functionality

Efficient discovery of 128-bit UUIDs

By default, any discovered 128-bit UUIDs that are not present in the Vendor Specific UUID table, will have the `ble_uuid_t::type` set to `BLE_UUID_TYPE_UNKNOWN`.

To change this default behavior and enable the automatic insertion of discovered 128-bit UUIDs to the Vendor Specific UUID table, the following option can be used:

```
sd_ble_opt_set(BLE_GATTC_OPT_UUID_DISC, &(ble_opt_t){.gattc_opt.uuid_disc.auto_add_vs_enable = 1});
```

s140_nrf52_7.0.1

This section describes how to use the new features of s140_nrf52_7.0.1 when migrating from s140_nrf52_6.1.1. The s140_nrf52_7.0.1 has changed the API compared to s140_nrf52_6.1.1 which requires applications to be recompiled.

Required changes

The application can no longer use the option `BLE_COMMON_OPT_ADV_SCHED_CFG`. The advertiser will always use improved scheduling. This was previously defined as `ADV_SCHED_CFG_IMPROVED`.

The macros `NRF_SOC_APP_PPI_CHANNELS_SD_DISABLED_MSK`, `NRF_SOC_APP_PPI_CHANNELS_SD_ENABLED_MSK`, `NRF_SOC_APP_PPI_GROUPS_SD_DISABLED_MSK`, and `NRF_SOC_APP_PPI_GROUPS_SD_ENABLED_MSK` are removed. The application can use the macros `NRF_SOC_SD_PPI_CHANNELS_SD_ENABLED_MSK` and `NRF_SOC_SD_PPI_GROUPS_SD_ENABLED_MSK` to deduce the PPI channels and groups available to the application.

New functionality

Connection event trigger

When enabled, this feature will trigger a task at the start of connection events. The application can configure the SoftDevice to trigger a task every N connection events starting from a given connection event counter.

API Updates

- `sd_ble_gap_next_conn_evt_counter_get()`. This API can be used to retrieve the next connection event counter.
- `sd_ble_gap_conn_evt_trigger_start()`, `sd_ble_gap_conn_evt_trigger_stop()`. These APIs can be used to start and stop triggering a task on connection events.

Usage

The code snippet below illustrates how to configure the SoftDevice to toggle the GPIO pin 13, every second connection event, starting at connection event 10. The code snippet stops the connection event trigger when the connection parameters are updated.

```
void on_ble_evt(const ble_evt_t * p_ble_evt)
{
  if (p_ble_evt->header.evt_id == BLE_GAP_EVT_CONNECTED)
  {
    uint16_t conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
```

```
    ble_gap_conn_event_trigger_t trigger_params;
    trigger_params.ppi_ch_id = 0;
    trigger_params.task_endpoint = &NRF_GPIOTE->TASKS_OUT[0];
    trigger_params.conn_evt_counter_start = 10;
    trigger_params.period_in_events = 2;

    sd_ble_gap_conn_evt_trigger_start(conn_handle, &trigger_params);
  }
  else if (p_ble_evt->header.evt_id == BLE_GAP_EVT_CONN_PARAM_UPDATE)
  {
    uint16_t conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
    sd_ble_gap_conn_evt_trigger_stop(conn_handle);
  }
}

int main(void)
{
  /* Configure GPIOTE */
  NRF_GPIO->DIRSET = (1 << 13);
  NRF_GPIOTE->CONFIG[0] = (GPIOTE_CONFIG_POLARITY_Toggle << GPIOTE_CONFIG_POLARITY_Pos)
                          | (13 << GPIOTE_CONFIG_PSEL_Pos)
                          | (GPIOTE_CONFIG_MODE_Task << GPIOTE_CONFIG_MODE_Pos);

  /* Enable the BLE Stack and connect device */
  sd_ble_enable(...);
  sd_ble_gap_connect(...);

  [...]
}
```

Configurable inclusion of Central Address Resolution (CAR) characteristic and Peripheral Preferred Connection Parameters (PPCP)

API Updates

- `BLE_GAP_CFG_CAR_INCL_CONFIG`. This allows the application to include or exclude the CAR characteristic from the GAP Service.
- `BLE_GAP_CFG_PPCP_INCL_CONFIG` . This allows the application to include or exclude the PPCP characteristic from the GAP Service.

For the above inclusion configuration APIs, the application can use:

- `BLE_GAP_CHAR_INCL_CONFIG_INCLUDE:` The characteristic is included.
- `BLE_GAP_CHAR_INCL_CONFIG_EXCLUDE_WITH_SPACE: T` he characteristic is excluded, but the SoftDevice will reserve the attribute handles which are otherwise used for this characteristic.
- `BLE_GAP_CHAR_INCL_CONFIG_EXCLUDE_WITHOUT_SPACE:` The characteristic is excluded.

When CAR is excluded and the SoftDevice is configured to support the central role:

- It is not possible to distribute own IRK.
- It is not possible to enable privacy.

Usage

The code snippet below illustrates how to configure the SoftDevice to exclude both CAR and PPCP from the GAP Service.

```
int main(void)
{
  ble_cfg_t cfg;


  /* Exclude CAR from GAP service, but reserve the ATT Handles that will otherwise be used up by CAR. */
  cfg.gap_cfg.car_include_cfg = BLE_GAP_CHAR_INCL_CONFIG_EXCLUDE_WITH_SPACE;
  sd_ble_cfg_set(BLE_GAP_CFG_CAR_INCL_CONFIG, &cfg, ..);


  /* Exclude PPCP from GAP service, but reserve the ATT Handles that will otherwise be used up by PPCP. */
  cfg.gap_cfg.ppcp_include_cfg= BLE_GAP_CHAR_INCL_CONFIG_EXCLUDE_WITH_SPACE;
  sd_ble_cfg_set(BLE_GAP_CFG_PPCP_INCL_CONFIG, &cfg, ..);

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

  [...]
}
```

s140_nrf52_6.1.0

This section describes how to use the new features of s140_nrf52_6.1.0 when migrating from s140_nrf52_6.0.0. As with all minor releases, the s140_nrf52_6.1.0 is binary compatible with s140_nrf52_6.0.0. Hence existing applications running on s140_nrf52_6.0.0 need not be recompiled unless the new features are needed. Advertising extensions and LE Coded PHY are now fully tested and qualified features.

New functionality

Scanning on two PHYs

Using a single call to sd_ble_gap_scan_start(), the application can make the SoftDevice scan for advertisers advertising on both LE 1M PHY and LE Coded PHY as primary advertising channels. For scanning on two PHYs, the API expects the interval parameter to be larger than or equal to twice the scan window and the extended flag to be set to 1.

The application can also use `sd_ble_gap_connect()` to scan on two PHYs before connecting to a peer peripheral. This is useful when the application does not know the PHY on which the peer peripheral is advertising.

Usage

```
static uint8_t raw_scan_buffer[BLE_GAP_SCAN_BUFFER_EXTENDED_MIN];
static ble_data_t scan_buffer =
    {
     .p_data = raw_scan_buffer,
     .len = sizeof(raw_scan_buffer)
    };
static uint16_t scan_window = 0x00A0; /* Corresponds to 100 ms */

int main(void)
{
  ble_gap_scan_params_t scan_params=
    {
     .extended      = 1, /* Enable extended scanning. */
     .scan_phys     = BLE_GAP_PHY_1MBPS | BLE_GAP_PHY_CODED ,
     .timeout       = BLE_GAP_SCAN_TIMEOUT_UNLIMITED,
```

```
      .window        = scan_window,
      .interval      = (scan_window * 2), /* Interval should be at least twice the scan window since the scanning
is requested for two PHYs. */
    };

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

  /* Start scanning */
  sd_ble_gap_scan_start(&scan_params, &scan_buffer);


  /* Stop scanning */
  sd_ble_gap_scan_stop();


  /* Create a connection to a peer that is advertising on either LE 1M PHY or LE Coded PHY. */
  sd_ble_gap_connect(..., &scan_params, ...);

  [...]
}
```

Support for advertising with up to 255 bytes of advertising data

The SoftDevice now supports advertising up to 255 bytes of advertising data. The macro `BLE_GAP_ADV_SET_DATA_SIZE_EXTENDED_MAX_SUPPORTED` is added to indicate this. For connectable extended advertising, the maximum advertising data size is 238 bytes, as indicated by `BLE_GAP_ADV_SET_DATA_SIZE_EXTENDED_CONNECTABLE_MAX_SUPPORTED` .

Usage

Extended Non-Connectable Non-Scannable Advertising with 255 bytes of Advertising data

```
static uint8_t raw_adv_data_data_buffer[BLE_GAP_ADV_SET_DATA_SIZE_EXTENDED_MAX_SUPPORTED];
static ble_gap_adv_data_t adv_data  =
    {
     .adv_data.p_data = raw_adv_data_data_buffer,
     .adv_data.len = sizeof(raw_adv_data_data_buffer)
    };

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_EXTENDED_NONCONNECTABLE_NONSCANNABLE_UNDIRECTED
      },
      .interval              = BLE_GAP_ADV_INTERVAL_MAX,
      .duration              = BLE_GAP_ADV_TIMEOUT_LIMITED_MAX,
      .channel_mask          = {0},
      .max_adv_evts          = 0,
      .filter_policy         = BLE_GAP_ADV_FP_ANY,
      .primary_phy           = BLE_GAP_PHY_1MBPS,
      .secondary_phy         = BLE_GAP_PHY_2MBPS,
    };

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

  [...]
  sd_ble_gap_adv_set_configure(&adv_handle, &adv_data, &adv_params);

  /* Start advertising */
```

```
  sd_ble_gap_adv_start(adv_handle, BLE_CONN_CFG_TAG_DEFAULT);

  [...]
}
```

Extended Scannable Advertising with 255 bytes of Scan Response data

```
static uint8_t raw_scan_rsp_data_buffer[BLE_GAP_ADV_SET_DATA_SIZE_EXTENDED_MAX_SUPPORTED];
static ble_gap_adv_data_t adv_data  =
    {
     .scan_rsp_data.p_data = raw_scan_rsp_data_buffer,
     .scan_rsp_data.len = sizeof(raw_scan_rsp_data_buffer)
    };

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_EXTENDED_NONCONNECTABLE_SCANNABLE_UNDIRECTED
     },
     .interval              = BLE_GAP_ADV_INTERVAL_MAX,
     .duration              = BLE_GAP_ADV_TIMEOUT_LIMITED_MAX,
     .channel_mask          = {0},
     .max_adv_evts          = 0,
     .filter_policy         = BLE_GAP_ADV_FP_ANY,
     .primary_phy           = BLE_GAP_PHY_1MBPS,
     .secondary_phy         = BLE_GAP_PHY_2MBPS,
```

```
    };

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

  [...]
  sd_ble_gap_adv_set_configure(&adv_handle, &adv_data, &adv_params);

  /* Start advertising */
  sd_ble_gap_adv_start(adv_handle, BLE_CONN_CFG_TAG_DEFAULT);

  [...]
}
```

Support for receiving up to 255 bytes of advertising data

The SoftDevice now supports receiving up to 255 bytes of advertising data as a scanner. The macro `BLE_GAP_SCAN_BUFFER_EXTENDED_MAX_SUPPORTED` is added to indicate this.

Usage

```
static uint8_t raw_scan_buffer[BLE_GAP_SCAN_BUFFER_EXTENDED_MAX_SUPPORTED];
static ble_data_t scan_buffer =
    {
     .p_data = raw_scan_buffer,
     .len = sizeof(raw_scan_buffer)
    };
static uint16_t scan_window = 0x00A0; /* Corresponds to 100 ms */

int main(void)
{
  ble_gap_scan_params_t scan_params=
    {
     .extended      = 1,        /* Enable extended scanning to be able to receive large advertising data. */
     .scan_phys     = BLE_GAP_PHY_1MBPS | BLE_GAP_PHY_CODED,
     .timeout       = BLE_GAP_SCAN_TIMEOUT_UNLIMITED,
     .window        = scan_window,
     .interval      = BLE_GAP_SCAN_INTERVAL_MAX,
     .channel_mask  = {0}, /* Scanning on all the primary channels */
     .filter_policy = BLE_GAP_SCAN_FP_ACCEPT_ALL
    };

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

  /* Start scanning */
  sd_ble_gap_scan_start(&scan_params, &scan_buffer);

  [...]
}
```

API for removing a Vendor Specific base UUID

Using `sd_ble_uuid_vs_remove()`, the application can now remove a Vendor Specific base UUID that has been added with `sd_ble_uuid_vs_add()`. This allows the application to reuse memory allocated for Vendor Specific base UUIDs. The application must provide a pointer to the UUID type to be removed as an input parameter to `sd_ble_uuid_vs_remove()`. The UUID type must not be in use by the ATT Server. A limitation with the current implementation is that the input parameter can only point to `BLE_UUID_TYPE_UNKNOWN` or the last added UUID type.

API to enable or disable extended RC calibration

Extended RC calibration is a new SoftDevice feature that performs additional RC oscillator drift detection and calibration when the SoftDevice is acting as a peripheral and the RC oscillator is used as the SoftDevice clock source. The extended RC calibration is performed in addition to the periodic calibration which is configured when calling `sd_softdevice_enable()`. If using only peripheral connections, the periodic calibration can then be configured with a much longer interval because the peripheral can detect and adjust automatically to clock drift and calibrate when required.

The extended RC calibration is enabled by default. The option `BLE_COMMON_OPT_EXTENDED_RC_CAL` is added to the BLE option API, allowing the application to enable or disable this feature. When using this API, set `ble_common_opt_t::extended_rc_cal::enable` to '1' to enable, or to '0' to disable.

API to get the advertiser Bluetooth device address

A new API `sd_ble_gap_adv_addr_get()` enables the application to get the local Bluetooth device address that is used by the advertiser. The application must provide the advertising handle of the advertiser for the `adv_handle` input parameter, and a pointer to an address structure `p_addr` to be used as the output parameter. The function may only be called when advertising is enabled.

Note: If privacy is enabled , the SoftDevice will generate a new private address every `ble_gap_privacy_params_t::private_addr_cycle_s`, which is configured when calling `sd_ble_gap_privacy_set()`. Depending on when sd_ble_gap_adv_addr_get() is called, the returned address may not be the address that is currently used by the advertiser.

Hardware resource usage API

The API now contains new macros to inform the application about the hardware resources used by the SoftDevice.

- The macro `__NRF_NVIC_SD_IRQ_PRIOS` indicates the interrupt priority levels used by the SoftDevice.
- The macro `__NRF_NVIC_APP_IRQ_PRIOS` indicates the interrupt priority levels available to the application.
- The macros `NRF_SOC_SD_PPI_CHANNELS_SD_ENABLED_MSK` and `NRF_SOC_SD_PPI_CHANNELS_SD_DISABLED_MSK` can be used to identify the PPI channels reserved by the SoftDevice when the SoftDevice is enabled or disabled respectively.
- The macros `NRF_SOC_APP_PPI_CHANNELS_SD_ENABLED_MSK` and `NRF_SOC_APP_PPI_CHANNELS_SD_DISABLED_MSK` can be used to identify the PPI channels available to the application when the SoftDevice is enabled or disabled respectively.
- The macros `NRF_SOC_SD_PPI_GROUPS_SD_ENABLED_MSK` and `NRF_SOC_SD_PPI_GROUPS_SD_DISABLED_MSK` can be used to identify the PPI groups reserved by the SoftDevice when the SoftDevice is enabled or disabled respectively.
- The macros `NRF_SOC_APP_PPI_GROUPS_SD_ENABLED_MSK` and `NRF_SOC_APP_PPI_GROUPS_SD_DISABLED_MSK` can be used to identify the PPI groups available to the application when the SoftDevice is enabled or disabled respectively.

Other additions to the API

- The macro `SD_VARIANT_ID` indicates the SoftDevice variant.
- The macro `SD_FLASH_SIZE` indicates the amount of flash memory used by the SoftDevice.

s140_nrf52_6.0.0

This section describes how to migrate to s140_nrf52_6.0.0 from s132_nrf52_5.1.0.

Notes:

- s140_nrf52_6.0.0 has changed the API compared to s132_nrf52_5.1.0 which requires applications to be recompiled.
- s140_nrf52_6.0.0 includes some features that are not Bluetooth qualified. For more information, see the release notes.

New functionality

Quality of Service (QoS) channel survey

This feature provides measurements of the energy levels on the Bluetooth Low Energy channels to the application. The application can use this information to determine the noise floor on a per channel basis and set an adapted channel map to avoid busy channels.

When the feature is enabled, `BLE_GAP_EVT_QOS_CHANNEL_SURVEY_REPORT` events will periodically report the measured energy levels for each channel. The channel energy is reported in `ble_gap_evt_qos_channel_survey_report_t::channel_energy[BLE_GAP_CHANNEL_COUNT]` , indexed by the Channel Index. The SoftDevice will attempt to measure energy levels and deliver reports with the average interval specified in `interval_us` .

Note: To make the channel survey feature available to the application, `ble_gap_cfg_role_count_t::qos_channel_survey_role_available` must be set. This is done using the `sd_ble_cfg_set()` API.

The event structures for `BLE_GAP_EVT_RSSI_CHANGED` and `BLE_GAP_EVT_ADV_REPORT` have been changed to provide the application the channel number for reported Received Signal Strength Indication (RSSI) measurements. For more information, see Updated RSSI API in the Required changes section.

API Updates

- A new boolean flag, `ble_gap_cfg_role_count_t::qos_channel_survey_role_available`, must be set in the SoftDevice role configuration API to make the channel survey available for the application.
- Two new SV calls have been added to start and stop the channel survey:
  - `sd_ble_gap_qos_channel_survey_start()`
  - `sd_ble_gap_qos_channel_survey_stop()`

Usage

```
/* Make Channel Survey feature available to the application */
ble_cfg_t cfg;
cfg.role_count.qos_channel_survey_role_available = 1;
sd_ble_cfg_set(..., &cfg, ...);
```

```
/* Start receiving channel survey continuously. */
uint32_t errcode;
errcode = sd_ble_gap_qos_channel_survey_start(BLE_GAP_QOS_CHANNEL_SURVEY_INTERVAL_CONTINUOUS);
```

```
int8_t rssi;
/* A new measurement is ready. */
case BLE_GAP_EVT_QOS_CHANNEL_SURVEY_REPORT:
{
  for (i = 0; i < BLE_GAP_CHANNEL_COUNT; i++)
  {
     rssi = p_ble_evt->evt.gap_evt.params.qos_channel_survey_report.channel_energy[i];
  }
}
```

```
/* Stop receiving channel survey. */
errcode = sd_ble_gap_qos_channel_survey_stop()
```

Advertising Extensions

The LE Advertising Extensions feature has limited support in this SoftDevice that can be enabled with the new advertiser and scanner API. The feature may not function as specified, and may contain issues. For more information, see the release notes.

Extended Advertiser

Extended advertising can be enabled by assigning an `_EXTENDED_` advertising type to the `ble_gap_adv_params_t::properties::type` .

The extended advertising types are:

`BLE_GAP_ADV_TYPE_EXTENDED_CONNECTABLE_NONSCANNABLE_UNDIRECTED`

`BLE_GAP_ADV_TYPE_EXTENDED_CONNECTABLE_NONSCANNABLE_DIRECTED`

`BLE_GAP_ADV_TYPE_EXTENDED_NONCONNECTABLE_SCANNABLE_UNDIRECTED`

`BLE_GAP_ADV_TYPE_EXTENDED_NONCONNECTABLE_SCANNABLE_DIRECTED`

`BLE_GAP_ADV_TYPE_EXTENDED_NONCONNECTABLE_NONSCANNABLE_UNDIRECTED`

`BLE_GAP_ADV_TYPE_EXTENDED_NONCONNECTABLE_NONSCANNABLE_DIRECTED`

New parameters in the API that are relevant for extended advertising:

- `ble_gap_adv_params_t::properties::anonymous`
  - If this flag is set to 1, the advertiser's address will be omitted from all PDUs. This is only available for extended advertising event types.
- `ble_gap_adv_params_t::primary_phy`
  - Indicates the PHY on which the primary advertising channel packets are transmitted.
  - For extended advertising event types, this can be set to `BLE_GAP_PHY_AUTO` , `BLE_GAP_PHY_1MBIT` , or `BLE_GAP_PHY_CODED` if supported by the SoftDevice.
- `ble_gap_adv_params_t::secondary_phy`
  - Indicates the PHY on which the auxiliary PDUs will be sent.
  - Can be set to `BLE_GAP_PHY_AUTO` , `BLE_GAP_PHY_1MBPS` , `BLE_GAP_PHY_2MBPS` , or `BLE_GAP_PHY_CODED` if supported by the SoftDevice.
- `ble_gap_adv_params_t::set_id`
  - This value is used as the Advertising Set ID in the AdvDataInfo field of the PDU.

Extended Scanner

Scanning of extended advertising PDUs can be enabled by setting the `ble_gap_scan_params_t::extended` flag to 1 for the scan parameters provided to `sd_ble_gap_scan_start()` . If set to 1, both legacy and extended advertising PDUs will be scanned. If the flag is set to 0, all extended advertising PDUs will be ignored by the scanner. Correspondingly, to connect to a peer that is advertising with extended advertising PDUs, set the `ble_gap_scan_params_t::extended` flag to 1 for the scan parameters provided to `sd_ble_gap_connect()` .

New parameters in the API that are relevant for extended scanning:

- `ble_gap_scan_params_t::report_incomplete_evts`
  - This option is currently not supported.
- `ble_gap_evt_adv_report_t::type::extended_pdu`
  - Will be set to 1 if an extended advertising set is received.
- `ble_gap_evt_adv_report_t::tx_power`
  - The transmit power reported by the advertising in the last packet header received. The TX power field is present only in some extended advertising PDUs.
- `ble_gap_evt_adv_report_t::aux_pointer`
  - The offset and PHY of the next advertising packet in this extended advertising set.
  - This field will only be set if `ble_gap_evt_adv_report_t::type::status` is set to `BLE_GAP_ADV_DATA_STATUS_INCOMPLETE_MORE_DATA` .
- `ble_gap_evt_adv_report_t::set_id`
  - Set ID of the received advertising data. This is only present in some extended advertising PDUs.
- `ble_gap_evt_adv_report_t::data_id`
  - Data ID of the received advertising data. This is only present in some extended advertising PDUs.

Access to USB power handling registers

The SoftDevice provides new APIs allowing the application to enable or disable USB power interrupts . It is also now possible to read the value of the USB supply status register.

API Updates

- Four new APIs have been added
  - `sd_power_usbpwrrdy_enable()` : Enable or disable the USB power ready event.
    - When enabled, the `NRF_EVT_POWER_USB_POWER_READY` event will be raised when USB 3.3 V supply is ready.
  - `sd_power_usbdetected_enable()` : Enable or disable the USB power detected event.
    - When enabled, the `NRF_EVT_POWER_USB_DETECTED` event will be raised when voltage supply is detected on the VBUS pin.
  - `sd_power_usbremoved_enable()` : Enable or disable the USB power removed event.
    - When enabled, the `NRF_EVT_POWER_USB_REMOVED` event will be raised when voltage supply removed from the VBUS pin.
  - `sd_power_usbregstatus_get()` : Get the USB supply status register content.

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 Access Control Lists (ACL) peripheral which is designed to assign and enforce access permissions to different regions of the on-chip flash memory map. Therefore, `sd_flash_protect()` has been removed in this SoftDevice.

Usage

```
uint32_t errcode;
/* Set the start address of the flash page to 0x10000 */
errcode = sd_protected_register_write(&(NRF_ACL->ACL[0].ADDR), 0x10000);

if (errcode == NRF_SUCCESS)
{
  /* Set the size of the region to protect to 0x1000 */
  errcode = sd_protected_register_write(&(NRF_ACL->ACL[0].SIZE), 0x1000);
}

if (errcode == NRF_SUCCESS)
{
  /* Set the permission for the protected region to read/write protected */
  errcode = sd_protected_register_write(&(NRF_ACL->ACL[0].PERM), (ACL_ACL_PERM_READ_Msk |
ACL_ACL_PERM_WRITE_Msk) );
}
```

Configure power failure levels for high voltage

A new API, `sd_power_pof_thresholdvddh_set()` , has been added to give the application the possibility to set the power failure comparator threshold for high voltage.

See `NRF_POWER_THRESHOLDVDDHS` for valid thresholds.

Enable DC/DC converter for REG0 stage

A new API, `sd_power_dcdc0_mode_set()` , has been added to give the application the possibility to enable the DC/DC regulator for the regulator stage 0 (REG0).

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 scanner API

The scanner API has been updated. The changes are as follows:

- `ble_gap_scan_params_t` has been changed:
  - A new flag, `extended`, has been added. If set to 1, the scanner will receive both legacy advertising packets and extended advertising packets. If set to 0, the extended advertising packets will be ignored.
  - The Observer channel map for primary advertising channels can be set through a new parameter `ble_gap_scan_params_t::channel_mask`. The parameter type `ble_gap_ch_mask_t` is the same as is used for setting advertiser channel map.
  - `use_whitelist` and `adv_dir_report` have been combined into `filter_policy`. See `BLE_GAP_SCAN_FILTER_POLICIES` for valid policies.
  - `scan_phys` has been added to let the application decide on which PHYs the scanner should receive packets. Set to `BLE_GAP_PHY_AUTO` or`BLE_GAP_PHY_1MBPS` if extended scanning is disabled .
  - `timeout` is now measured in 10 ms units.
- `sd_ble_gap_scan_start()` has a new input parameter, `p_adv_report_buffer`, which takes a pointer to an advertising report buffer that must be kept alive until the scanner is stopped. The minimum buffer size is either `BLE_GAP_SCAN_BUFFER_MIN` or `BLE_GAP_SCAN_BUFFER_EXTENDED_MIN` when extended scanning is enabled.
- When the application receives a `ble_gap_adv_report_t`, it must now resume scanning by calling `sd_ble_gap_scan_start()`.
- ble_gap_evt_adv_report_t has been updated:
  - `ble_gap_evt_adv_report_t ::type` has been redefined from `uint8_t` to `ble_gap_adv_report_type_t`.
  - `scan_rsp` flag has been removed. It is now included in `ble_gap_adv_report_type_t::scan_response`.
  - `data` and `dlen` have been replaced with `data` of type `ble_data_t`.
  - New fields have been added: and `aux_pointer`.
- `ble_gap_evt_timeout_t` now includes `adv_report_buffer` which is set when the scanner times out.
- `BLE_GAP_SCAN_INTERVAL_MAX` and `BLE_GAP_SCAN_WINDOW_MAX` have been increased from 0x4000 to 0xFFFF.
- `BLE_GAP_SCAN_TIMEOUT_MAX` has been removed.

Usage

```
static uint8_t raw_scan_buffer[BLE_GAP_SCAN_BUFFER_MIN];
static ble_data_t scan_buffer = {.p_data = raw_scan_buffer, .len = sizeof(raw_scan_buffer)};

void on_ble_evt(const ble_evt_t * p_evt)
{
  if (p_ble_evt->header.evt_id == BLE_GAP_EVT_ADV_REPORT)
  {
    ble_gap_evt_adv_report_t * p_report = &p_ble_evt->evt.gap_evt.params.adv_report;

    /* Read out data*/
    [...]

    /* Continue scanning. */
    sd_ble_gap_scan_start(NULL, &scan_buffer);
  }
}

int main(void)
{
  ble_gap_scan_params_t scan_params= {.extended      = 0,
                                                       .scan_phys     = BLE_GAP_PHY_AUTO,
                                      .timeout       = BLE_GAP_SCAN_TIMEOUT_UNLIMITED, /* Unlimited scanning */
                                      .interval      = BLE_GAP_SCAN_INTERVAL_MAX,
                                      .channel_mask  = {0}, /* Scanning on all the primary channels */
                                      .filter_policy = BLE_GAP_SCAN_FP_ACCEPT_ALL
                                     };
  /* Enable the BLE Stack */
  sd_ble_enable(...);

  /* Start scanning */
  sd_ble_gap_scan_start(&scan_params, &scan_buffer);
```

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

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_protect()` has been removed.

`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.

LE Coded PHY

Note: When `sd_ble_gap_phy_update` is used to reply to a PHY Update, depending on the peer's preferences, `BLE_GAP_PHY_AUTO` might result in the PHY to be changed to `BLE_GAP_PHY_CODED`. This PHY is not Bluetooth Qualified in this SoftDevice. For more information, see the release notes.

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 adverting 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 `0xFFFFFF`.

`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

NRF_CLOCK_LF_XTAL_ACCURACY_250_PPM	NRF_CLOCK_LF_ACCURACY_250_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_500_PPM	NRF_CLOCK_LF_ACCURACY_500_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_150_PPM	NRF_CLOCK_LF_ACCURACY_150_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_100_PPM	NRF_CLOCK_LF_ACCURACY_100_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_75_PPM	NRF_CLOCK_LF_ACCURACY_75_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_50_PPM	NRF_CLOCK_LF_ACCURACY_50_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_30_PPM	NRF_CLOCK_LF_ACCURACY_30_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_20_PPM	NRF_CLOCK_LF_ACCURACY_20_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_10_PPM	NRF_CLOCK_LF_ACCURACY_10_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_5_PPM	NRF_CLOCK_LF_ACCURACY_5_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_2_PPM	NRF_CLOCK_LF_ACCURACY_2_PPM
NRF_CLOCK_LF_XTAL_ACCURACY_1_PPM	NRF_CLOCK_LF_ACCURACY_1_PPM

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: `ble_enable_params_t` member	New API: `cfg_id` in `sd_ble_cfg_set()`
`common_enable_params.vs_uuid_count`	`BLE_COMMON_CFG_VS_UUID`
`common_enable_params.p_conn_bw_counts`	`BLE_CONN_CFG_GAP` (*)

`gap_enable_params.periph_conn_count`
`gap_enable_params.central_conn_count`
`gap_enable_params. central_sec_count`
	`BLE_GAP_CFG_ROLE_COUNT`

`gap_enable_params.p_device_name`	`BLE_GAP_CFG_DEVICE_NAME`
`gatt_enable_params`	`BLE_CONN_CFG_GATT` (*)
`gatts_enable_params.service_changed`	`BLE_GATTS_CFG_SERVICE_CHANGED`
`gatts_enable_params.attr_tab_size`	`BLE_GATTS_CFG_ATTR_TAB_SIZE`

(*) 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 correpsonding 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: `BLE_CONN_BWS`	New API: `ble_gap_conn_cfg_t::event_length` in `sd_ble_cfg_set()`
`BLE_CONN_BW_LOW`	`BLE_GAP_EVENT_LENGTH_MIN`
`BLE_CONN_BW_MID`	`BLE_GAP_EVENT_LENGTH_DEFAULT`
`BLE_CONN_BW_HIGH`	`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 giong 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.

static uint8_t raw_adv_data_data_buffer[BLE_GAP_ADV_SET_DATA_SIZE_EXTENDED_MAX_SUPPORTED]; /* 255 bytes of advertising data. */static ble_gap_adv_data_t adv_data = {.adv_data.p_data = raw_adv_data_data_buffer, .adv_data.len = sizeof(raw_adv_data_data_buffer)};
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_EXTENDED_NONCONNECTABLE_NONSCANNABLE_UNDIRECTED }, .interval = BLE_GAP_ADV_INTERVAL_MAX, .duration = BLE_GAP_ADV_TIMEOUT_LIMITED_MAX, .channel_mask = {0}, .max_adv_evts = 0, .filter_policy = BLE_GAP_ADV_FP_ANY, .primary_phy = BLE_GAP_PHY_AUTO, .secondary_phy = BLE_GAP_PHY_AUTO, }; /* Enable the BLE Stack */ sd_ble_enable(...);
  [...] sd_ble_gap_adv_set_configure(&adv_handle, &adv_data, &adv_params);
  /* Start advertising */ sd_ble_gap_adv_start(adv_handle, BLE_CONN_CFG_TAG_DEFAULT); [...]}