s110_nrf51 migration document

On this page

Introduction to the s110_nrf51 migration document

About the document

This document describes how to migrate to new versions of the s110_nrf51. The s110_nrf51 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.

s110_nrf51_8.0.0

This section describes how to migrate to s110_nrf51_8.0.0 from s110_nrf51822_7.x.x.

Required changes

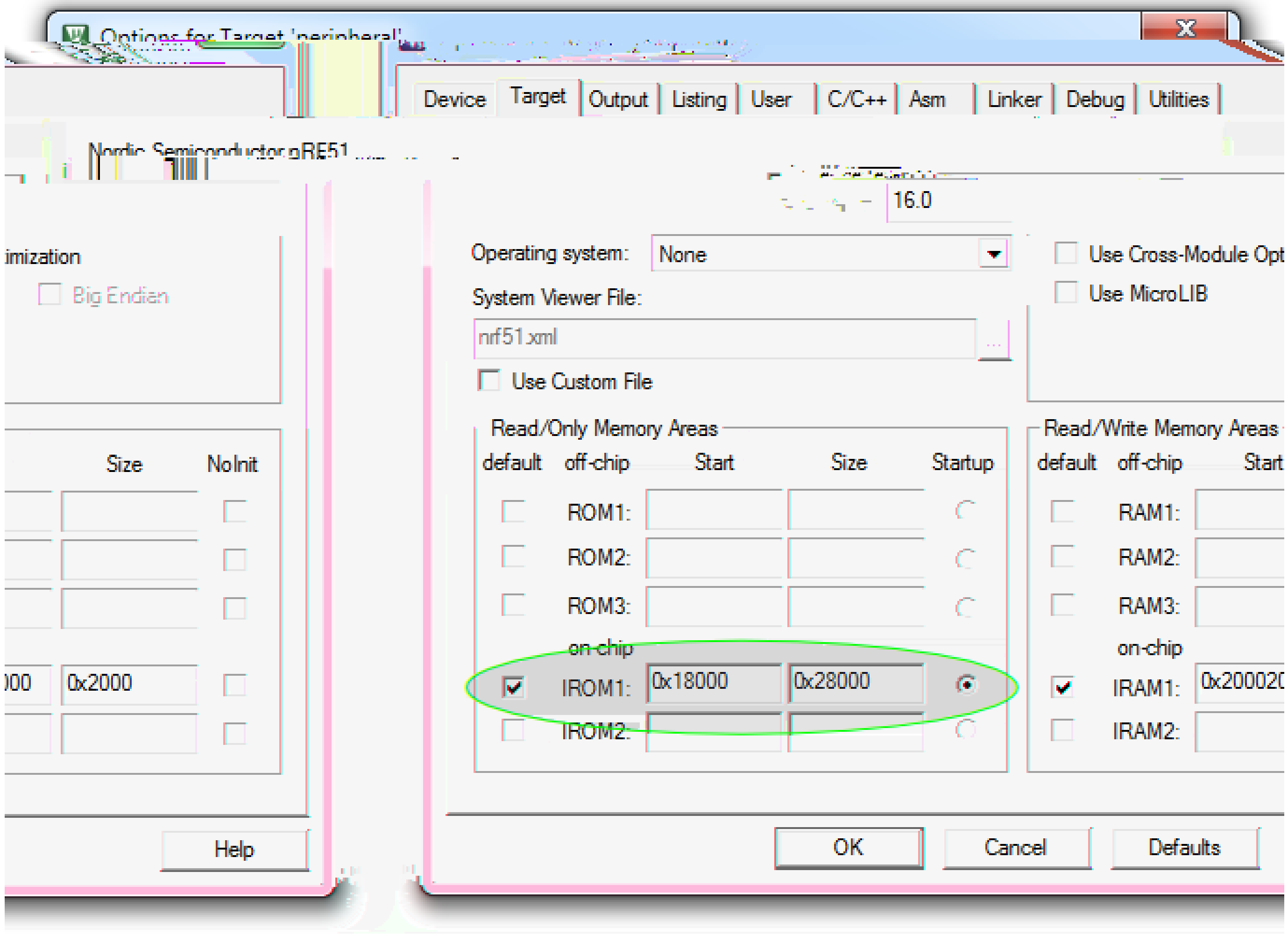
SoftDevice size:

The size of the SoftDevice has changed requiring a change to the application project file. The new size (including MBR) is 96 kB.

For Keil this means:

- Go into the properties of the project and find the Target tab
- Change IROM1 Start to `0x18000`
- Ensure that the IROM1 size is no more than `0x28000` (for the 256 kB IC variants).

If the project uses a scatter file instead of the settings from the Target tab, the scatter file must be updated accordingly.



Options for Target 'peripheral'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Nordic Semiconductor nRF51

16.0

Operating system: None

System Viewer File: nrf51.xml

☐ Use Custom File

☐ Use Cross-Module Opt

☐ Use MicroLIB

imization

☐ Big Endian

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
✓	IROM1:	0x18000	0x28000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x20002	0x2000	☐
☐	IRAM2:			☐

SVC number changes:

The SVC numbers in use by the stack have been changed so the application needs to be recompiled against the new header files.

The `NRF_POWER_DCDC_MODES` enumeration has been simplified:

Instead of the previous `OFF`, `ON` and `AUTOMATIC` modes, it can now only be set to `NRF_POWER_DCDC_DISABLE` or `NRF_POWER_DCDC_ENABLE`. This affects the `sd_power_dcdc_mode_set()` SV call. Note that the DC/DC converter is only supported on nRF51 series IC revision 3.

The GATT Server Attribute Table size can now be configured:

It is now possible to configure the size of the GATT Server Attribute table by using the parameter `attr_tab_size` in `ble_gatts_enable_params_t` when calling `sd_ble_enable()`.

Configuration of the Attribute Table size is optional. The default Attribute Table size (`0x700` bytes) will be used if `attr_tab_size` is set to `BLE_GATTS_ATTR_TAB_SIZE_DEFAULT`. The default size is equal to the Attribute Table size in previous version of the s110 SoftDevice.

If the application wishes to use an Attribute Table size different from the default one, the linker configuration of the application **must be changed** accordingly to match the total amount of RAM used by the SoftDevice. The following formula can be used to calculate the amount of RAM that the SoftDevice (together with the MBR) will use and therefore the address in RAM where the application memory area begins:

```
SoftDevice RAM size = (0x1900 + attr_tab_size) default: 0x2000
```

The address at which the application memory area begins is therefore:

```
Application RAM begin = (0x20001900 + attr_tab_size) default: 0x20002000
```

`ble_gap_adv_params_t` now contains a additional `channel_mask` field that must be filled in:

When calling `sd_ble_gap_adv_start()`, a pointer to an instance of `ble_gap_adv_params_t` must be provided. This structure now contains an additional field (`channel_mask`) allowing the application to disable specific advertising channels, which must be filled in by the application. To enable all advertising channels simply set this whole field to zero. The SoftDevice behavior will then remain unchanged from previous versions.

The `BLE_GAP_EVT_CONNECTED` event now includes the device's own address:

The new `own_addr` field in the `ble_gap_evt_connected_t` structure allows the application to find out which address was used to establish a particular connection, which can be useful when using privacy features.

The GAP options member name has changed:

The `ble_gap_opt_t` instance inside `ble_opt_t` has been renamed from `gap` to `gap_opt`.

The GAP advertising timeout source macro has been renamed:

The `BLE_GAP_TIMEOUT_SRC_ADVERTISEMENT` macro has been renamed to `BLE_GAP_TIMEOUT_SRC_ADVERTISING`.

The connection RSSI interface has been expanded:

Two new parameters to the `sd_ble_gap_rssi_start()` SV call, `threshold_dbm` and `skip_count`, now allow the application to specify how often and under which conditions it wishes to receive connection RSSI events from the SoftDevice. Set both this new parameters to 0 if you wish to receive the same number of events as with previous versions of the SoftDevice.

The new `sd_ble_gap_rssi_get()` SV call allows applications to poll the connection RSSI of the last connection interval.

The maximum value for slave latency has been reduced:

The `BLE_GAP_CP_SLAVE_LATENCY_MAX` value has been changed from `0x03E8` to `0x01F3` to comply with versions 4.1 and above of the Bluetooth specification.

The security APIs have been redesigned and improved:

All the application-facing security interfaces have been modified and improved for the benefit of speed, memory efficiency, future-proofing and compatibility with other SoftDevice series. All application developers will have to adapt to the new security APIs.

Key distribution selection is now configurable by the application:

Two instances of a new `ble_gap_sec_kdist_t` structure are now part of `ble_gap_sec_params_t` and allow the application to select which keys will be distributed by each side during a bonding procedure. Applications are therefore now required to carefully select which keys will be necessary during the bond's lifetime. The application should check the keys requested by the peer (propagated to the application inside the `ble_gap_sec_params_t` present in the `ble_gap_evt_sec_params_request_t` structure) before selecting the keys that are to be finally exchanged. The key selection needs to be included in the security parameters passed to `sd_ble_gap_sec_params_reply()`.

The recommended key distribution selection for applications is shown below. This will distribute the LTK and IRK in both directions when bonding and only when requested by the peer:

```
case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
 if(bonding)
 {
 peer_params = p_ble_evt->evt.gap_evt.params.sec_params_request.peer_params;
  own_params.kdist_periph.enc = peer_params.kdist_periph.enc;
  own_params.kdist_periph.id = peer_params.kdist_periph.id;
  own_params.kdist_periph.sign = 0;
  own_params.kdist_central.enc = peer_params.kdist_central.enc;
  own_params.kdist_central.id = peer_params.kdist_central.id;
  own_params.kdist_central.sign = 0;
  sd_ble_gap_sec_params_reply(.., &own_params, ...);
 }
 break;
```

To emulate the behavior of previous SoftDevices, the key selection would be performed as follows:

```
case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
 if(bonding)
 {
 peer_params = p_ble_evt->evt.gap_evt.params.sec_params_request.peer_params;
  own_params.kdist_periph.enc = peer_params.kdist_periph.enc;
  own_params.kdist_periph.id = peer_params.kdist_periph.id;
  own_params.kdist_periph.sign = 0;
  own_params.kdist_central.enc = 0;
  own_params.kdist_central.id = peer_params.kdist_central.id;
  own_params.kdist_central.sign = peer_params.kdist_central.sign;
  sd_ble_gap_sec_params_reply(.., &own_params, ...);
 }
 break;
```

The application can no longer specify a timeout for security procedures:

The `timeout` field has been removed from `ble_gap_sec_params_t`, and the timeout length is now set internally by the stack according to the specification.

All encryption keys are now identified using EDIV/RAND pairs:

The former LTK identifier, `div`, has now been removed from `ble_gap_enc_info_t` and instead LTKs are now identified using a `ble_gap_mas` ter_id_t instance (EDIV/RAND pair). The application must now use the new master_id field in the ble_gap_evt_sec_info_request_t in stead of the former div field to uniquely identify the LTK for that peer device.

Exchanged keys are now stored directly in application-provided memory:

The application now supplies pointers to memory to store exchanged keys instead of receiving them in the BLE_GAP_EVT_AUTH_STATUS event. A number of changes are made to support this:

- Two new structures have been added to the API for this purpose, ble_gap_enc_key_t and ble_gap_id_key_t.
- The ble_gap_sec_keys_t has been refactored from a bitfield into a set of pointers to keys in application memory. p_enc_key, p_id_key and p_sign_key are all pointers that must be initialized by the application to point to its own instances of key structures in memory.
- A new structure, ble_gap_sec_keyset_t, has been introduced and is required as a parameter to sd_ble_gap_sec_params_reply(). This allows the application to provide the dual set of pointers to the stack (one for each of the devices participating in the bonding procedure). The keys are stored directly in the application's instances as soon as they are received over the air or generated and sent

Please note that a special exception applies to the IRK distributed by the peripheral (ble_gap_sec_keyset_t::keys_periph::p_id_key). If this p_id_key pointer is initialized to a valid value, the IRK and device address pointed to will be distributed over the air by the SoftDevice to the peer device. If, however, the application sets p_id_key to NULL, the device's currently configured IRK and device address will be distributed. Most applications will want to set this pointer to NULL unless they plan to provide their own IRK and device address instead of using the one generated by the SoftDevice or configured using the BLE_GAP_OPT_PRIVACY option and the sd_ble_gap_address_set() SV call.

The authentication (pairing or bonding) procedure result structure has been revamped:

The existing ble_gap_evt_auth_status_t structure, which encapsulates a BLE_GAP_EVT_AUTH_STATUS event, has been modified to fit the new key storage model. A new bonded field has been added that allows the application to check whether the procedure ended up in the generation of a new bond. A new structure, ble_gap_sec_kdist_t, which is simply a bitfield of the keys distributed during a bonding procedure has been added to the API and two instances of it have been included in the ble_gap_evt_auth_status_t structure as the kdist_periph and kdist_central fields. The application can check those new fields to find out which keys ended up being distributed during the bonding procedure after it has completed.

The security information reply now accepts IRKs:

The existing sd_ble_gap_sec_info_reply() SV call now takes an additional parameter in the form of p_id_info, a pointer to an optional IRK. This can be set to NULL by the application as it's currently ignored by the SoftDevice.

The GATTS set and get operations for local values have changed:

The new function prototypes require a connection handle (since this is required for certain multi-value attributes) and also use a new structure ble_gatts_value_t for parameter input:

- sd_ble_gatts_value_set(uint16_t conn_handle, uint16_t handle, ble_gatts_value_t *p_value)
- sd_ble_gatts_value_get(uint16_t conn_handle, uint16_t handle, ble_gatts_value_t *p_value)

The GATTS set and get operations for system attributes have changed:

The new function prototypes include an additional parameter, flags, to allow for partial retrieval or storage of system attributes. If the application does not want to make use of this new functionality, it can simply set the flags parameter to 0.

- sd_ble_gatts_sys_attr_set(uint16_t conn_handle, uint8_t const *p_sys_attr_data, uint16_t len, uint32_t flags)
- sd_ble_gatts_sys_attr_get(uint16_t conn_handle, uint8_t *p_sys_attr_data, uint16_t *p_len, uint32_t flags)

The GATTC write parameters have been slightly modified:

The field flags within the ble_gattc_write_params_t has been moved up in the structure for better alignment.

New functionality

Scan Request reports are now available:

An application can now request to receive events whenever an observer issues a scan request by using the new `BLE_GAP_OPT_SCAN_REQ_REPORT` option. The scan request reports will be propagated to the application through the new `ble_gap_evt_scan_req_report_t`, and more information on the feature can be found under the documentation for the new option structure `ble_gap_opt_scan_req_report_t`.

The connection channel map is now retrievable:

Applications can now retrieve the channel map used for connections by using the new `BLE_GAP_OPT_CH_MAP` option. This feature is completely independent of the advertising channel mask listed in the "Required changes" section of this document. See documentation of `ble_gap_opt_ch_map_t` for more information.

The SoftDevice info structure is now documented:

A set of new macros have been introduced to access the SoftDevice info structure directly from hex or bin SoftDevice images. This can be useful when doing device firmware upgrade or when generic information about a SoftDevice in binary form is to be retrieved. See `nrf_sdm.h` for details on the new macros listed below.

- `MBR_SIZE`
- `SOFTDEVICE_INFO_STRUCT_OFFSET`
- `SOFTDEVICE_INFO_STRUCT_ADDRESS`
- `SOFTDEVICE_INFO_STRUCT_OFFSET`
- `SD_FWID_OFFSET`
- `SD_SIZE_GET()`
- `SD_FWID_GET()`

s110_nrf51822_7.0.0

This section describes how to migrate to s110_nrf51822_7.0.0 from s110_nrf51822_6.0.0.

Required changes

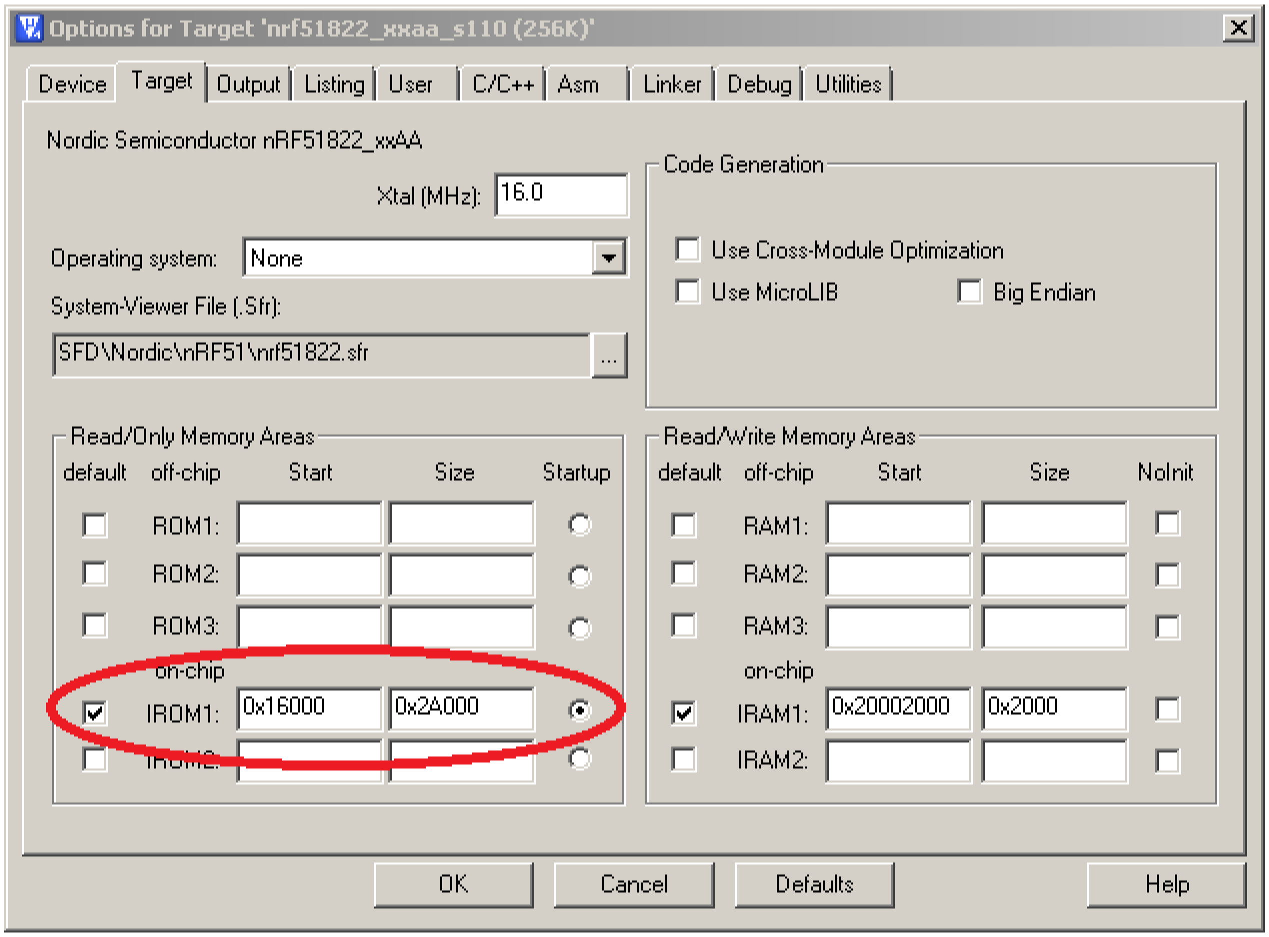
SoftDevice size

The size of the SoftDevice has changed requiring a change to the application project file.

For Keil this means:

- Go into the properties of the project and find the Target tab
- Change IROM1 Start to `0x16000`
- Ensure that the IROM1 size is no more than `0x2A000`

If the project uses a scatter file instead of the settings from the Target tab, the scatter file must be updated accordingly.


Options for Target 'nrf51822_xxaa_s110 (256K)'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Nordic Semiconductor nRF51822_xxAA

Xtal (MHz): 16.0

Operating system: None

System-Viewer File (.Sfr): SFD\Nordic\nRF51\nrf51822.sfr

Code Generation
- [ ] Use Cross-Module Optimization
- [ ] Use MicroLIB
- [ ] Big Endian

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
✓	IROM1:	0x16000	0x2A000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x20002000	0x2000	☐
☐	IRAM2:			☐

OK | Cancel | Defaults | Help

SVC number changes

The SVC numbers in use by the stack have been changed so the application needs to be recompiled against the new header files.

An additional call has to be made after `sd_softdevice_enable()` before any BLE related functionality in the SoftDevice can be used:

- `sd_ble_enable(p_ble_enable_params)`

Using this call, the application can select whether to include the Service Changed characteristic in the GATT Server. The default in all previous releases has been to include the Service Changed characteristic, but this affects how GATT clients behave. Specifically, it requires clients to subscribe to this attribute and not to cache attribute handles between connections unless the devices are bonded. If the application does not need to change the structure of the GATT server attributes at runtime this adds unnecessary complexity to the interaction with peer clients. If the SoftDevice is enabled with the Service Changed Characteristics turned off, then clients are allowed to cache attribute handles making applications simpler on both sides. Please see the SoftDevice API documentation for details.

Due to an issue present in this release, the application is required to set the device address to the factory default right after calling `sd_ble_enable()`, this can be achieved with the following lines:

```
sd_ble_enable(p_ble_enable_params);
sd_ble_gap_address_get(&addr);
sd_ble_gap_address_set(BLE_GAP_ADDR_CYCLE_MODE_NONE, &addr);
```

`sd_ble_gap_address_set()` now takes an additional argument which is used to describe the private address cycle mode:

- `BLE_GAP_ADDR_CYCLE_MODE_NONE`
- `BLE_GAP_ADDR_CYCLE_MODE_AUTO`

To set all types of device addresses explicitly as with earlier versions of the SoftDevice, use

- `sd_ble_gap_address_set(BLE_GAP_ADDR_CYCLE_MODE_NONE, p_addr)`

To let the SoftDevice automatically cycle private addresses as defined by Bluetooth Core specification 4.1, use

- `sd_ble_gap_address_set(BLE_GAP_ADDR_CYCLE_MODE_AUTO, p_addr)`

where `p_addr->addr_type` is either `BLE_GAP_ADDR_TYPE_RANDOM_PRIVATE_RESOLVABLE` or `BLE_GAP_ADDR_TYPE_RANDOM_PRIVATE_NON_RESOLVABLE`.

Redirecting interrupts to an application from a bootloader has changed:

- `sd_softdevice_forward_to_application()` has been replaced with `sd_softdevice_vector_table_base_set(address)`

Interrupts can now be directed to anywhere in the application flash area. This also enables using more than one application. See the SoftDevice API documentation for details on how to use this call.

New functionality

The SoftDevice hex file no longer contains the SoftDevice size in the `UICR.CLENR0` register.

This means that the Memory Protection Unit is no longer configured to protect the SoftDevice code, memory space and protected peripherals, unless this is deliberately enabled. Memory protection must be disabled to allow Device Firmware Upgrade of the SoftDevice. However it may be useful to have the protection enabled during development to ease the detection of illegal memory and peripheral accesses.

- If the SoftDevice is programmed with nRFgo Studio 1.17 or newer, use the checkbox "Enable SoftDevice protection" in the "Program SoftDevice" dialog to enable or disable the protection.
- If the SoftDevice is programmed with nrfjprog.exe version 5.1.1 or newer using the --programs option, then protection is enabled.
- If the SoftDevice is programmed with nrfjprog.exe version 5.1.1 or newer using the --programs --dfu options, then protection is disabled.

The SoftDevice now includes a new Options API to allow the application to set and get advanced configuration options:

- `sd_ble_opt_get()`
- `sd_ble_opt_set()`

Three options are defined in this version of the SoftDevice:

- `BLE_GAP_OPT_LOCAL_CONN_LATENCY` can be used to override the Connection Latency specified by the Central.
- `BLE_GAP_OPT_PASSKEY` can be used to specify a 6-digit display passkey that will be used during pairing instead of a randomly generated one.
- `BLE_GAP_OPT_PRIVACY` can be used to tune the behaviour of the SoftDevice when advertising with private addresses.

Please see the SoftDevice API documentation for details on how to use the Options API.

The following types and definitions have been renamed:

- `SD_EVENT_IRQn` is now `SD_EVT_IRQn`
- `SD_EVENT_IRQHandler` is now `SD_EVT_IRQHandler`
- `SD_APP_EVENT_WAIT` is now `SD_APP_EVT_WAIT`
- `SD_EVENT_GET` is now `SD_EVT_GET`
- `NRF_SOC_EVENTS` is now `NRF_SOC_EVTS`
- `NRF_EVENT_HFCLKSTARTED` is now `NRF_EVT_HFCLKSTARTED`
- `NRF_EVENT_POWER_FAILURE_WARNING` is now `NRF_EVT_POWER_FAILURE_WARNING`
- `NRF_EVENT_NUMBER_OF_EVENTS` is now `NRF_EVT_NUMBER_OF_EVENTS`
- `sd_app_event_wait()` is now `sd_app_evt_wait()`
- `sd_event_get()` is now `sd_evt_get()`

The SoC framework must now be used to obtain temperature readings:

- `sd_temp_get()`

The application may no longer access the `NRF_TEMP` registers directly when the SoftDevice is enabled.

The SoC framework now presents an API to access flash memory safely during active BLE connections:

- `sd_flash_write()`
- `sd_flash_page_erase()`
- `sd_flash_protect()`

The application may no longer perform flash memory write, erase or protect operations directly using `NRF_NVMC` and `NRF_MPU->PROTENSET` registers when the SoftDevice is enabled.

Two new events required for GATTS Queued Writes have been added to the API and need to be handled by the application:

- `BLE_EVT_USER_MEM_REQUEST`
- `BLE_EVT_USER_MEM_RELEASE`

Those events will only be issued by the SoftDevice whenever a Write Long Characteristic Value (or Descriptor) or a Reliable Write procedure is started by the peer GATT client. Details on how to handle those events can be found in the Message Sequence Charts included with the SoftDevice documentation, including indications to making use of the new API function:

- `sd_ble_user_mem_reply(USER_MEMORY)`

If the application does not wish to allow any of the aforementioned procedures to be executed (as in previous versions of the SoftDevice), a NULL pointer should be passed into the SoftDevice upon reception of the `BLE_EVT_USER_MEM_REQUEST` event, using the new API call:

- `sd_ble_user_mem_reply(NULL)`

By passing in a `NULL` pointer the application has provided no memory and therefore prevents the execution of the procedures, so that subsequent `BLE_GATTS_EVT_WRITE` or `BLE_GATTS_EVT_RW_AUTHORIZE_REQUEST` with the op field set to one of the new values (`BLE_GATTS_OP_PREP_WRITE_REQ`, `BLE_GATTS_OP_EXEC_WRITE_REQ_CANCEL` or `BLE_GATTS_OP_EXEC_WRITE_REQ_NOW`) can be safely ignored or denied respectively.

The following definition has been removed from the header files, since it is no longer required:

- `BLE_GAP_DEVNAME_MAX_WR_LEN`

Use this instead

- `BLE_GAP_DEVNAME_MAX_LEN`

The following structure member has been removed:

- `ble_gap_evt_disconnected_t.peer_addr`

New functionality

The following structure member has been added to allow the application to obtain the handle provided by the peer in an ATT Error Response:

- `ble_gattc_evt_t.error_handle`

Along with the support for GATTS Write Long and Reliable Write procedures, 3 new GATTS operation types have been added:

- `BLE_GATTS_OP_PREP_WRITE_REQ`
- `BLE_GATTS_OP_EXEC_WRITE_REQ_CANCEL`
- `BLE_GATTS_OP_EXEC_WRITE_REQ_NOW`

And 2 new GATT operation types and flags for GATTC Write Long and Reliable Write support:

- `BLE_GATT_OP_PREP_WRITE_REQ`
- `BLE_GATT_OP_EXEC_WRITE_REQ`
- `BLE_GATT_EXEC_WRITE_FLAG_PREPARED_CANCEL`
- `BLE_GATT_EXEC_WRITE_FLAG_PREPARED_WRITE`

The following structure member has been added for GATTC Execute Write Requests:

- `ble_gattc_write_params_t.flags`

The following structure member has been added for GATTC Prepare Write Response:

- `ble_gattc_evt_write_rsp_t.offset`

The following 2 functions now return the complete length of their respective arrays:

- `sd_ble_gap_device_name_get`
- `sd_ble_gatts_value_get`

In practical terms what this means is that the application is able to detect that the device name or the value of a certain attribute will not fit in the buffer it has provided to the SoftDevice:

Device name is currently: "`Nordic Semiconductor`" (20 bytes)

```
uint8_t buffer[10];
uint16_t len = sizeof(buffer);
uint32_t errcode;
errcode = sd_ble_gap_device_name_get(buffer, &len);
```

Here the output would be:

```
errcode = NRF_SUCCESS
len = 20
buffer = "Nordic Sem"
```

Additionally, the application can pass `NULL` as `p_dev_name` or `p_value` to obtain the full length.

The maximum number of 128-bit Vendor Specific UUIDs has been increased from 5 to 10, and is now exposed to the application:

- `BLE_UUID_VS_MAX_COUNT`

s110_nrf51822_5.2.0

This section describes how to migrate to s110_nrf51822_5.2.0 from s110_nrf51822_5.1.0.

Required changes

Due to changes in SoftDevice API header files, applications may need to include nrf51_bitfields.h and/or nrf51_deprecated.h explicitly.

s110_nrf51822_5.1.0

This section describes how to migrate to s110_nrf51822_5.1.0 from s110_nrf51822_5.0.0.

Required changes

No changes are required due to this upgrade.

Note: The *nrf_power_dcdc_mode** API will be deprecated in a future version. See the "Limitations" section in the release notes for further information.

New functionality

No new functionality in this version.

s110_nrf51822_5.0.0

This section describes how to migrate to s110_nrf51822_5.0.0 from s110_nrf51822_4.0.0

Required changes

Lower stack (Link Layer) interrupts are extended by a “CPU Suspend” state during radio activity to improve link integrity. This means lower stack interrupts will **block application and upper stack (Host) processing during a Radio Event** for a time proportional to the number of packets transferred in the event. The application's interrupt latency (both for App High and App Low interrupts) will therefore also increase by a time proportional to the number of packets transferred in the event due to the lower stack preventing the execution of lower priority contexts. Applications relying on low latency interrupts while the radio is active will therefore have to adapt to avoid timing issues.

The impact of this change and the required modifications to the application will highly depend on the application's nature, requirements and behavior. However, in general terms the application should refer to Table 10 (S110 interrupt latency lower stack) in the S110 SoftDevice Specification v1.1 to analyze how radio traffic will impact interrupt latency and incorporate the numbers in the design and implementation of the application's interrupt handlers.
A typical example of the timing potentially being off is using a TIMER or RTC peripheral running with a period shorter than the interrupt latencies described in the table above. Such short period configurations may cause the Interrupt Service Routine to miss ticks if the implementation keeps time by counting on its execution at a fixed frequency. A different timekeeping strategy needs to be used in this case, such as increasing the period (lowering the frequency) or accounting for missed Interrupt Service Routine execution instances by using the relevant information in registers provided by those peripherals.

The following **GATT Characteristic Presentation Format and Namespace definitions:**

- `BLE_GATT_CPF_FORMAT_*`
- `BLE_GATT_CPF_NAMESPACE_*`

have been moved from `ble_types.h` to `ble_gatt.h`.

Two additional definitions have been added to `ble_gatts.h`:

- `BLE_GATTS_FIX_ATTR_LEN_MAX`
- `BLE_GATTS_VAR_ATTR_LEN_MAX`

These macros define the maximum attribute value length for fixed and variable length attributes respectively.

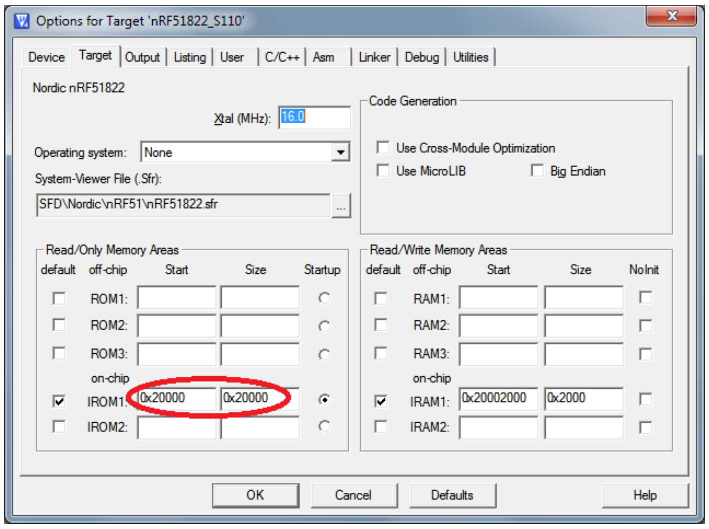
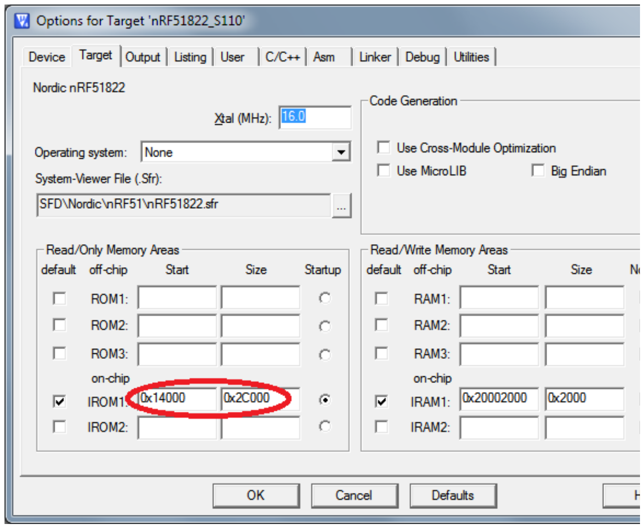
s110_nrf51822_4.0.0

This section describes how to migrate to s110_nrf51822_4.0.0 from s110_nrf51822_3.0.0

Required changes

The flash requirements for the S110 SoftDevice have changed from 128kB to 80kB as of this version. All future releases of the **S110** SoftDevice will require 80kB of flash memory (i.e. CODE_R1_BASE = 0x00014000). To adapt to this new size, applications will need to redefine their base addresses from ~~`0x00020000`~~ to `0x00014000.`

In Keil, you can achieve this by changing the IROM1 value in the target options:

From:	To:
	

Options for Target 'nRF51822_S110'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Nordic nRF51822

Xtal (MHz): 16.0

Operating system: None

System-Viewer File (.Sfr): SFD\Nordic\nRF51\nRF51822.sfr

Code Generation: Use Cross-Module Optimization ☐, Use MicroLIB ☐, Big Endian ☐

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
✓	IROM1:	0x20000	0x20000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x20002000	0x2000	☐
☐	IRAM2:			☐

OK | Cancel | Defaults | Help

Options for Target 'nRF51822_S110'

Nordic nRF51822

Xtal (MHz): 16.0

Operating system: None

System-Viewer File (.Sfr): SFD\Nordic\nRF51\nRF51822.sfr

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
✓	IROM1:	0x14000	0x2C000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size
☐	RAM1:		
☐	RAM2:		
☐	RAM3:		
	on-chip		
✓	IRAM1:	0x20002000	0x2000
☐	IRAM2:		

OK | Cancel | Defaults

All SuperVisor Calls have been renamed in the 4.0.0 SoftDevice according to the following rules:

- `nrf_*` calls are now `sd_*`
- `ble_*` calls are now `sd_ble_*`
- SVC_* SVC IDs are now SD_*

Application source files that invoke SuperVisor Calls will have to be modified accordingly to match the new naming convention.
When debugging an application the presence of an "`sd_`" prefix now signals the jump to a SoftDevice function in the form of an SVC.

The `sd_power_pof_enable()` function no longer uses a callback. Instead, use `sd_power_pof_enable(true)` to enable power failure detection; the `NRF_EVENT_POWER_FAILURE_WARNING` event will be returned by `sd_event_get()`if a power failure occurs.

UUID conversion functions are now in the SoftDevice. All UUIDs are represented in the S110 SoftDevice as a structure named **`ble_uuid_t`** containing 2 fields (`uuid` and `type`). This representation achieves better speed and memory efficiency than standard 2 and 16 byte arrays and makes it easier to interface with APIs using a unified UUID type. Starting from 4.0.0, the encoding and decoding functions are now inside the BLE stack instead of being in the SDK, and the table population function has changed. To be able to use the new functionality, the application will have to first add all the required 128-bit UUIDs so that they can be referenced later by the ble_uuid_t instances, calling the population function as many times as necessary.

This used to be achieved with the `ble_uuid_vs_assign()` function, which has now been removed in favor of `sd_ble_uuid_vs_add()`:

```
const ble_uuid128_t vs_uuids[] = {{0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC,
0x00, 0x00, 0xFF, 0x17},
{0xA1, 0xB2, 0xC3, 0xD4, 0xE5, 0xF6, 0x07, 0x18, 0x29, 0x3A, 0x4B, 0x5C, 0x00, 0x00, 0x6F, 0x77}};

uint8_t type;
uint32_t i, errcode;
```

~~`errcode = ble_uuid_base_set(NUMELTS(vs_uuids), vs_uuids);`~~

or

~~`errcode = ble_uuid_vs_assign(NUMELTS(vs_uuids), vs_uuids);`~~

```
for(i = 0; i < NUMELTS(vs_uuids); i++)
{
  errcode = sd_ble_uuid_vs_add((ble_uuid128_t const *) &vs_uuids[i], &type);
  ASSERT(errcode == NRF_SUCCESS);
  ASSERT(type == BLE_UUID_TYPE_VENDOR_BEGIN + i);
}
```

Later the code can call the encoding and decoding functions to convert 16 or 128-bit UUIDs into or from ble_uuid_t structures:

```
const ble_uuid128_t raw_sig_uuid = {0xFB, 0x34, 0x9B, 0x5F, 0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00,
0x08, 0x82, 0x00, 0x00};
const ble_uuid128_t raw_vs_uuid = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC,
0x34, 0x12, 0xFF, 0x17};

uint8_t raw_output[16];
ble_uuid_t ble_uuid;
uint8_t uuid_len;
```

~~`errcode = ble_uuid_decode(16, raw_sig_uuid.uuid128, &ble_uuid);`~~
`errcode = sd_ble_uuid_decode(16, raw_sig_uuid.uuid128, &ble_uuid);`

~~`errcode = ble_uuid_encode(&ble_uuid, &uuid_len, raw_output);`~~
`errcode = sd_ble_uuid_encode(&ble_uuid, &uuid_len, raw_output);`

~~`errcode = ble_uuid_decode(16, raw_vs_uuid.uuid128, &ble_uuid);`~~

```
errcode = sd_ble_uuid_decode(16, raw_vs_uuid.uuid128, &ble_uuid);
```

~~errcode = ble_uuid_encode(&ble_uuid, &uuid_len, raw_output);~~

```
errcode = sd_ble_uuid_encode(&ble_uuid, &uuid_len, raw_output);
```

Applications can now check directly for characteristic properties in the corresponding bitfield included in the characteristic discovery response:

```
ble_evt_t ble_evt;
```

~~if((ble_evt.evt.gattc_evt.params.char_disc_rsp.chars[0].properties & 0x4)~~
~~&& (ble_evt.evt.gattc_evt.params.char_disc_rsp.chars[0].properties & 0x10))~~
~~{...}~~

```
if(ble_evt.evt.gattc_evt.params.char_disc_rsp.chars[0].char_props.write_wo_resp
&& ble_evt.evt.gattc_evt.params.char_disc_rsp.chars[0].char_props.notify)
{...}
```

The characteristic properties types are now shared between client and server and so GATTS service population has changed slightly:

```
ble_gatts_char_md_t char_md;
```

~~char_md.char_properties.notify = 1;~~
~~char_md.char_properties.broadcast = 1;~~
~~char_md.char_properties.wr_aux = 1;~~

```
char_md.char_props.notify = 1;
char_md.char_props.broadcast = 1;
char_md.char_ext_props.wr_aux = 1;
```

New functionality

It is now possible to clear the advertising data and/or the scan response data, and the combinations stand today as shown below:

```
sd_ble_gap_adv_data_set(uint8_t const * const p_data, uint8_t dlen, uint8_t const * const p_sr_data,
uint8_t srdlen));


p_data == NULL && dlen == 0      /* adv data unchanged */
p_sr_data == NULL && srdlen == 0 /* scan response data unchanged */
p_data != NULL && dlen == 0      /* adv data cleared (0-length packet) */
p_sr_data != NULL && srdlen == 0 /* scan response data cleared (0-length packet) */
```