Modifying the Secure OTA Bootloader to support fetching image from external flash

1. Check if there is a new update available in external flash. Upon boot the bootloader will call nrf_bootloader_init(), which in turn will call dfu_enter_check() from nrf_bootloader.c. Add an if statement to check if there is a new firmware image available in external flash, if yes, return true.

```
/**@brief Function for checking whether to enter DFU mode or not.*/
static bool dfu_enter_check(void)
{
  if (!app_is_valid(crc_on_valid_app_required()))
  {
    NRF_LOG_DEBUG("DFU mode because app is not valid.");
    return true;
  }
  /*
  if(new_firmware_available_in_ext_flash)
  {
    NRF_LOG_DEBUG("Enter DFU mode. New firmware image available in external flash.");
    fw_available_in_external_flash = true;
    return true;
  }
  */
  if (NRF_BL_DFU_ENTER_METHOD_BUTTON &&
    (nrf_gpio_pin_read(NRF_BL_DFU_ENTER_METHOD_BUTTON_PIN) == 0))
  {
    NRF_LOG_DEBUG("DFU mode requested via button.");
    return true;
  }

  if (NRF_BL_DFU_ENTER_METHOD_PINRESET &&
    (NRF_POWER->RESETREAS & POWER_RESETREAS_RESETPIN_Msk))
  {
    NRF_LOG_DEBUG("DFU mode requested via pin-reset.");
    return true;
  }

  if (NRF_BL_DFU_ENTER_METHOD_GPREGRET &&
    (nrf_power_gpregret_get() & BOOTLOADER_DFU_START))
  {
    NRF_LOG_DEBUG("DFU mode requested via GPREGRET.");
    return true;
  }

  if (NRF_BL_DFU_ENTER_METHOD_BUTTONLESS &&
    (s_dfu_settings.enter_buttonless_dfu == 1))
  {
    NRF_LOG_DEBUG("DFU mode requested via bootloader settings.");
    return true;
  }
```

2. If dfu_enter() returns true, you will then enter the following if-statement in nrf_bootloader_init(). Here the nrf_dfu_init_user() function will be called before initializing the DFU Transport layer by calling nrf_dfu_init().

```
if (dfu_enter)
{
    nrf_bootloader_wdt_init();
    scheduler_init();
    dfu_enter_flags_clear();

    // Call user-defined init function if implemented
    ret_val = nrf_dfu_init_user();
    if (ret_val != NRF_SUCCESS)
    {
        return NRF_ERROR_INTERNAL;
    }

    nrf_bootloader_dfu_inactivity_timer_restart(initial_timeout, inactivity_timeout);

    ret_val = nrf_dfu_init(dfu_observer);
    if (ret_val != NRF_SUCCESS)
    {
        return NRF_ERROR_INTERNAL;
    }

    NRF_LOG_DEBUG("Enter main loop");
    loop_forever(); // This function will never return.
    NRF_LOG_ERROR("Unreachable");
}
```

3. The nrf_dfu_init_user() can be modified to perform user-specific initialization, e.g. fetch the init packet(.dat file) from external flash.

```
/**@brief Weak implementation of nrf_dfu_init
 *
 * @note    This function must be overridden in application if
 *          user-specific initialization is needed.
 */
__WEAK uint32_t nrf_dfu_init_user(void)
{
    NRF_LOG_DEBUG("in weak nrf_dfu_init_user");
    return NRF_SUCCESS;
}
```

4. Overriding the nrf_dfu_init_user() function by defining it in another.c file. Note that the nrf_dfu_req_handler_init_user() and send_select_request() functions should only be called if fw_available_in_external_flash flag set in the dfu_enter_check() function.

```
void nrf_dfu_init_user(nrf_dfu_observer_t observer)
{
    uint32_t ret_val;
    NRF_LOG_INFO("In DFU Processor: nrf_dfu_init_user()");

    if(fw_available_in_external_flash)
    {
        NRF_LOG_INFO("Firmware available in external flash.");

        /* Add code to fetch init packet(.dat) from external flash here */

        // Initialize the DFU Request Handler
        ret_val = nrf_dfu_req_handler_init_user(dfu_observer);
        APP_ERROR_CHECK(ret_val);

       /*Submit a Select request to the bootloaders request handler to start the
        *  DFU process. See Message sequence chart in the bootloader docs
        */
       ret_val = send_select_request();
       APP_ERROR_CHECK(ret_val);

        for(;;)
        {
            app_sched_execute();
            NRF_LOG_PROCESS();
        }

        // Should never be reached. The
        NRF_LOG_INFO("After nrf_dfu_init_user");
    }
    else
    {
        NRF_LOG_INFO("No Firmware available in external flash, initializing SD f
or OTA BLE update.");
    }

}
```

5. The first select request is used to start the DFU process, see the Transfer of an init packet message sequence chart in the Bootloader documentation. The select command will select the last object that was created. This is done to check if an object has been create previously, but the DFU process was interrupted, e.g. a reset or power failure. This is to allow the bootloader to resume the process instead of starting all over again.

```
uint32_t send_select_request(){

     uint32_t ret_val;

    // Prepare Request
    nrf_dfu_request_t dfu_request;
    dfu_request.request            = NRF_DFU_OP_OBJECT_SELECT;
    dfu_request.callback.response  = on_select_response;
    dfu_request.select.object_type = NRF_DFU_OBJ_TYPE_COMMAND;

    // Schedule the request
    ret_val = nrf_dfu_req_handler_on_req(p_req);

    if (ret_val != NRF_SUCCESS)
    {
        NRF_LOG_WARNING("Scheduler ran out of space!");
    }
    return ret_val;
}
```

6. The DFU request handler will call the callback in the nrf_dfu_request_t structure, i.e. to return the response, in this case on_select_response.
7. The values passed with the on_select_response callback will tell you if an init packet has been transferred before(partially or in full) or not. If it's a partial transfer, you may resume the transfer where you left of. If an entire init packet has been transferred, then you may jumpdirectly to executing it. If no init packet has been transferred you then create a new object and then transfer the init packet

8. If no init packet has been transferred, then you can send a create request to the request handler, e.g.

```
uint32_t send_create_request(){

     uint32_t ret_val;

    // Prepare Request
    nrf_dfu_request_t dfu_request;
    dfu_request.request            = NRF_DFU_OP_OBJECT_CREATE;
    dfu_request.callback.response  = on_create_response;
    dfu_request.create.object_type = NRF_DFU_OBJ_TYPE_COMMAND;
    dfu_request.create.object_size = init_packet_size; // Add Object Size

    // Schedule the request
    ret_val = nrf_dfu_req_handler_on_req(p_req);

    if (ret_val != NRF_SUCCESS)
    {
        NRF_LOG_WARNING("Scheduler ran out of space!");
    }
    return ret_val;
}
```

9. As with the select request, the respons will be provided in the on_create_callback. The nrf_dfu_response_t will contain the size of the create object, that you can pass on to the fuction that triggers the next request, which is the write request that actually transfers the data.

```
void on_create_response(nrf_dfu_response_t * p_res, void * p_context)
{
    //Trigger next request, i.e. the write request
}
```

10. The write request is the request that contains the actual data, either the init packet or a chunk of the firmware image. So you can fetch the init packet or firmware image chunk before scheduling the request.

```
uint32_t send_write_request(){

    uint32_t ret_val;

    // Fetch init packet or firmware image chunk from external flash

    // Prepare Request
    nrf_dfu_request_t dfu_request;
    dfu_request.request            = NRF_DFU_OP_OBJECT_WRITE;
    dfu_request.callback.response  = on_write_response;
    dfu_request.callback.write     = (nrf_dfu_flash_callback_t)on_write_op_complete;
    dfu_request.write.p_data       = pointer_to_data;
    dfu_request.write.len          = size_of_chunk;

    // Schedule the request
    ret_val = nrf_dfu_req_handler_on_req(p_req);

    if (ret_val != NRF_SUCCESS)
    {
        NRF_LOG_WARNING("Scheduler ran out of space!");
    }
    return ret_val;
}
```

11. The respons to the write request is passed with the on_write_response and the dfu_request.callback.write function, in this case, on_write_op_complete, will be called when the flash operation has finished. You can then use this callback to trigger the next write request. In the case of the init packet you will be able to fit the entire packet inside one object.
12. After the content of the init packet has been written, then you execute the init packet by sending a execute request. This will start the validation of the init packet.

```
uint32_t send_execute_request(){

    uint32_t ret;

    // Prepare Request
    nrf_dfu_request_t dfu_request;
    dfu_request.request            = NRF_DFU_OP_OBJECT_EXECUTE;
    dfu_request.callback.response  = on_execute_response;

    // Schedule the request
    ret = nrf_dfu_req_handler_on_req(p_req);

    if (ret != NRF_SUCCESS)
    {
        NRF_LOG_WARNING("Scheduler ran out of space!");
    }
    return ret;
}
```

13. The result of the validation will be passed with the on_execute_response callback, see the possible responses under Execute request in the bootloader documentation. If the execution is a success, i.e. the init packet is valid, you may start sending the firmware image according to the Transfer of a firmware image message sequence chart.
14. The Transfer of a firmware image follows the same approach as the transfer of the init packet, i.e. Select, Create, Write and finally execute.
15. When the firmware image is sent using write requests, then the bootloader will write the chunks to bank 1.
16. The execution of the firmware image will trigger the actual update, i.e. write the content in bank 1 to bank 0.

Entering the bootloader from the application

The bootloader can be entered by writing a specific value to one of the GPREGRET registers

```
uint32_t reset_to_bootloader(void)
{
    uint32_t err_code;

   err_code = sd_power_gpregret_clr(0, 0xffffffff);
   VERIFY_SUCCESS(err_code);

   err_code = sd_power_gpregret_set(0, BOOTLOADER_DFU_START); // BOOTLOADER_DFU_START == 0xB1
   VERIFY_SUCCESS(err_code);

   NVIC_SystemReset();
}
```