# SB3.1 DATA blocks

The data area of all blocks following the header block can be seen as a contiguous sequence of bytes, called the data stream, that has been divided up into equal-sized chunks. Each chunk is then placed into its own block.

For a given block size, the data chunk size is:

chunkSize = blockSize – blockOverhead

blockSize = 256 bytes on niobe4analog.

blockOverhead = 4bytes for block number (uint32\_t) + sha256 or sha384 digest of next block (32 or 48 bytes)

Data stream can be divided into multiple sections. Each section is starting by section header:

struct section\_header {

uint32\_t sectionUid;

uint32\_t sectionType;

uint32\_t length;

uint32\_t \_pad;

};

The length field of the section header can be used to skip over the section when searching for a given section.

Section types:

* 1 = Data range section
  + Contains one or more ranges of data to loaded and the target addresses.

Only one data range section supported on niobe4analog and by current elftosb.

A data range section contains one or more ranges of data to loaded and the target addresses

Data range sections consist of one or more ranges, where each range is begun with a header:

struct range\_header {

uint32\_t tag

uint32\_t startAddress;

uint32\_t length;

uint32\_t cmd;

};

The startAddress and length specify the target memory address range to be written. Tag carries a magic number 0x55aaaa55 as an identifier of data range header, it should be 0x55aaaa55. Cmd field is an enum of various actions to be performed on the data range.

cmd:

enum class CommandType {

kSB3\_COMMAND\_none = 0x0u,

kSB3\_COMMAND\_erase = 0x1u,

kSB3\_COMMAND\_load = 0x2u,

kSB3\_COMMAND\_execute = 0x3u,

kSB3\_COMMAND\_call = 0x4u,

kSB3\_COMMAND\_programFuses = 0x5u,

kSB3\_COMMAND\_programIfr = 0x6u,

kSB3\_COMMAND\_loadCmac = 0x7u,

kSB3\_COMMAND\_copy = 0x8u,

kSB3\_COMMAND\_loadHashlocking = 0x9u,

kSB3\_COMMAND\_loadKeyBlob = 0xAu,

kSB3\_COMMAND\_configureMemory = 0xBu,

kSB3\_COMMAND\_fillMemory = 0xCu,

kSB3\_COMMAND\_fwVersionCheck = 0xDu,

};

* 1 - Erase
  + Performs a flash erase of the given address range. The erase will be rounded up to the sector size.

struct range\_header {

uint32\_t tag 0x55aaaa55

uint32\_t startAddress;

uint32\_t length;

uint32\_t cmd;

};

Followed by:

typedef struct {

uint32\_t memoryId;

uint32\_t \_ pad0; //zeros

uint32\_t \_ pad1; //zeros

uint32\_t \_pad2; //zeros

} nboot\_range\_header\_memory\_data\_t;

* 2 - Load
  + If set, then the data to write immediately follows the range header. The length field contains the actual data length.

struct range\_header {

uint32\_t tag 0x55aaaa55

uint32\_t startAddress;

uint32\_t length;

uint32\_t cmd;

};

Followed by:

typedef struct {

uint32\_t memoryId;

uint32\_t \_ pad0; //zeros

uint32\_t \_ pad1; //zeros

uint32\_t \_pad2; //zeros

} nboot\_range\_header\_memory\_data\_t;

* 3 – Execute
  + The startAddress will be the jump-to address. No further processing of SB after jump, ROM do not expect to return.
* 4 – Call
  + The startAddress will be the address to jump, however, the state machine should expect a return to the next statement to continue processing the sb file
* 5 – programFuses
  + The startAddress will be the address of fuse register, length will be number of fuse words to program. The data to write to the fuse registers will immediately follow the header.
* 6 – programIFR
  + The startAddress will be the address into the IFR region, length will be in number of bytes to write to IFR region. The data to write to IFR region at the given address will immediately follow the header
* 7 – loadCmac
  + If set, then the data to write immediately follows the range header. The length field contains the actual data length. ROM is calculating cmac from loaded data and storing on address known by ROM decided based on startAddress.

struct range\_header {

uint32\_t tag 0x55aaaa55

uint32\_t startAddress;

uint32\_t length;

uint32\_t cmd;

};

Followed by:

typedef struct {

uint32\_t memoryId;

uint32\_t \_ pad0; //zeros

uint32\_t \_ pad1; //zeros

uint32\_t \_pad2; //zeros

} nboot\_range\_header\_memory\_data\_t;

* 8 – copy
  + Used for copying data from one place to another. 32 bytes fixed size. Structure is:

struct range\_header {

uint32\_t tag 0x55aaaa55

uint32\_t startAddress; //address to copy from

uint32\_t length; //number of bytes to copy

uint32\_t cmd; //0x8 (copy)

};

Followed by:

typedef struct {

uint32\_t destinationAddress; //target address of copy

uint32\_t memoryIdFrom;

uint32\_t memoryIdTo;

uint32\_t \_pad0; //zeros

} nboot\_copy\_command\_data\_t;

* 9 – loadHashLocking
  + If set, then the data to write immediately follows the range header. The length field contains the actual data length. ROM is calculating hash of the data and storing the value in the last 64 bytes of the loaded data, which are reserved for it.

struct range\_header {

uint32\_t tag 0x55aaaa55

uint32\_t startAddress;

uint32\_t length;

uint32\_t cmd;

};

Followed by:

typedef struct {

uint32\_t memoryId;

uint32\_t \_ pad0; //zeros

uint32\_t \_ pad1; //zeros

uint32\_t \_pad2; //zeros

} nboot\_range\_header\_memory\_data\_t;

* 10 – loadKeyBlob
  + Wrapped key blob immediately follows the range key blob header. The length field contains the actual data length.

typedef struct {

const uint32\_t tag = NBOOT\_RANGE\_SECTION\_TAG;

uint16\_t offset; // offset in CMPA

uint16\_t keyWrapId; //16 = NXP\_CUST\_KEK\_INT\_SK, 17 = NXP\_CUST\_KEK\_EXT\_SK

uint32\_t length;

uint32\_t cmd; // 0xA

} nboot\_load\_key\_range\_header\_t;

* 11 – configureMemory
  + Configuring memory.

typedef struct {

const uint32\_t tag = NBOOT\_RANGE\_SECTION\_TAG;

uint32\_t memoryId;

uint32\_t address;

uint32\_t cmd;

} nboot\_config\_memory\_header\_t;

* 12 – fillMemory
  + Used for filling of the memory range by same repeated int32 pattern.

struct range\_header {

uint32\_t tag 0x55aaaa55

uint32\_t startAddress;

uint32\_t length; // number of repeats of pattern

uint32\_t cmd; //0x8 (copy)

};

Followed by:

typedef struct {

uint32\_t pattern;

uint32\_t \_pad0; //zeros

uint32\_t \_pad1; //zeros

uint32\_t \_pad2; //zeros

} nboot\_copy\_command\_data\_t;

* 13 – fwVersionCheck
  + Used to execute check of provided counter value with value stored in specified monotonous counter in device. If values are not same, SB file is rejected.

typedef struct {

const uint32\_t tag = NBOOT\_RANGE\_SECTION\_TAG;

uint32\_t value; //value to compare with counter

uint32\_t counterId; // 1 - nonsecure, 2 – secure, 3

uint32\_t cmd; //0xDu

} nboot\_fw\_version\_check\_range\_header\_t;

enum class CounterId\_t {  
            kNBOOT\_CNT\_none = 0x0u,  
            kNBOOT\_CNT\_nonsecure = 0x1u,   
            kNBOOT\_CNT\_secure = 0x2u,   
            kNBOOT\_CNT\_radio = 0x3u,   
            kNBOOT\_CNT\_snt = 0x4u,   
            kNBOOT\_CNT\_bootloader = 0x5u,   
        };

It is an error to have a cmd field with a value of 0.