Easy Bootloader Library BL2 File Format Specification

The EZBL bootloader library, wherever possible, represents firmware images in a binary format that is appreciably more compact than the Intel .hex file format. BL2 files generally contain identical information to a .hex file, but contain a few derived fields of additional information to aid in compact and efficient bootloader decoding and communications.

When encoded optimally, .bl2 files will generally be negligibly larger than the total amount of programmable data they contain. This nearly always means a .bl2 file will be smaller than a .hex file. If all data is contiguous and only one data record is used in the .bl2 file, total file size is exactly 108 bytes larger than the programmable data.

BL2 files contain one file header, plus zero or more record headers interleaved with record data, and finally a footer consisting of a SHA-256 has and CRC32 checksum at file end.

General Rules

1. Unless otherwise stated, all header fields and record data are stored using little-endian byte ordering.
2. Unlike .hex files for PIC24 and dsPIC devices which implement address doubling and insertion of 'phantom 0x00 bytes' at every 4th address (i.e. address % 4 == 3), .blob files do not contain phantom bytes. They also do not use doubled addresses. Natural program memory addresses are used instead.
3. Data records do not have to end on a perfect address boundary. For example, PIC24 and dsPIC device .blob's may have 1 or 2 bytes in the last instruction location of a given data record, not a full 3 bytes.
4. Data records generally should be stored in sequentially ascending address order. However, in cases where it is advantageous to present certain data out-of-order, this rule may not apply. When presenting data out of the natural order, the data must be properly aligned for the target bootloader to be able to erase and write the applicable page, row, or flash word size without interfering with subsequent data records.
5. Any two data records generally should not contain data in overlapping address ranges. However, if an overlapping address range is given, the correct overlap data must be provided, depending on the device architecture and bootloader requirements. In cases where the target location will be in an erased state, the overlap data should be set according to the chronological programming sequence data requirements. In cases where more than one programming operation is allowed before being erased, the data in the overlapping region must be masked off with all '1's in all but one of the overlapping data records. This is required to prevent programming a '0' to the same Flash cell more than once total per erase. Some programming specifications advise programming the same contents into overlapping bits when programming a Flash word twice, but they generally will still be compatible with '1's on top of pre-existing '0's so long as the expectation is the final Flash contents will still be a '0' after the second programming operation.

BL2 Header

Length: 16 typical bytes + 48 fixed bytes

1. *SYNC*: 16 bytes - "UUUUUUUUMCUPHCME"
   Synchronization/auto-baud/bootloader wake up string. These characters should not be treated as critical and can be omitted or modified in future bootloader variations. They are treated as transient out-of-band set up data not related to the file contents, but meaningful for the target hardware.

The successive 'U' or 0x55 characters permit UART auto-baud, up to 4 nodes deep in a daisy chain, while the characters "MCUPHCME" are treated as a bootloader wake up sequence decoded directly in the communications RX ISR.

2. *FILE_ID*: 4 byte (32-bits) - "BL2B"
"BL2B" string, non-null terminated ("Blob Version 2 File Format"). Characters are stored byte-wise, so if treated as a single 32-bit little endian field, this string encodes as 0x42324C42.

3. *FILE_LEN*: 4 bytes (32-bits) – Byte count to End of File
Sum of all bytes from the start of the very next field (BOOTID_HASH) onwards to the end of the file (after the CRC32 field). Stored little endian. Assuming default SYNC data, the total length of the .bl2 file should be 24 bytes greater than this FILE_LEN.

4. *BOODID_HASH*: 16 bytes (128-bits) – SHA-256 hash, right-most 16 bytes
Right-most 16 bytes of the SHA-256 hash of the BOODID_* Strings concatenated to each other for the target bootloader or node that this file image is intended to be sent to (or was read from). As hashes are normally represented as a giant big endian quantity, in serialized fashion, these 16-bytes would be the second set of 16 bytes of the hash. In printed hex form, these are the bytes on the right. If the BOOTID_HASH feature is disabled, this field should be set to all 0's to indicate no hash.

5. APPID_VER: 8 bytes (64-bits) – Application Version Major/Minor/Build
Indicates the application software version being offered/present in this .bl2 file. This 8-byte value is subdivided as:
    - 4-bytes: APPID_VER_BUILD - 32-bit little endian Application version build number long integer.
        Ex: v1.23.xxxx
    - 2-bytes: APPID_VER_MINOR - 16-bit little endian Application version minor revision integer.
        Ex: v1.xx.4567
    - 2-bytes: APPID_VER_MAJOR - 16-bit little endian Application version major revision integer.
        Ex: vx.23.4567

    The subfields are oriented least-significant end first such that the whole 8-byte quantity can be compared with existing APPID_VER data as a 64-bit (long long) integer with greater-than meaning "more recent version" and less than meaning "older firmware". If the image does not contain any Application code (ex: .bl2 file is for a Bootloader only), this field should be set to all 0's.

6. HMAC_SHA_256: 16 bytes (128-bits) – Authenticated hash of prior data + nonce generator
Right-most 16-bytes of a HMAC computed using SHA-256 over the preceding 32 bytes + an encryption secret known only to the Bootloader and vendor writing the Application firmware. If this .bl2 file is not encrypted, all 0's should be stored in this field to indicate a non-encrypted/non-authenticated firmware image.

DATA_RECORD*n* (repeated an arbitrary number of times to store all data)

All data records within the .bl2 are encrypted if encryption is enabled.

DATA_REC_HDR - Record Header

Length: 8 bytes (fixed)

1. *REC_LEN*: 4 byte (32-bit) Record Length, excluding this header, measured in bytes
   32-bit little endian data length in bytes. This length does not include either field of this record header. This field can be zero if no data bytes exist at the target record address.

2. *REC_ADDR*: 4 byte (32-bit) Record start Address
   32-bit little endian target memory address for the data to be stored at. It is assumed that data begins at the specified record address and sequentially increments when the data length is greater than 1, but this is not a hard-and-fast requirement in this file format.

   Conceivably, an implementation can set up virtual target addresses or address ranges which do not naturally exist on the device architecture, but have implicit meaning to the consuming bootloader. For example, a special address could be reserved for Bootloader configuration data, in which no physical destination memory actually exists, nor do subsequent bytes necessarily imply ascending sequential target addresses.

REC_DATA - Record Data

Length: REC_LEN bytes

A series of sequential data bytes to store or be represented in the data record. After REC_LEN bytes have been processed, the next byte of the file is assumed to be another DATA_RECORD<REC_LEN> field unless the FILE_LEN remaining indicates <= 36 remaining bytes in the file. When the file length has reached 36 or less, then the next byte after this data record is a SHA256HASH field instead of a new DATA_RECORD.

FOOTER - Hash/Checksum Footer

Length: 36 bytes (fixed)

1. *FILE_HASH:* 32 bytes (256-bits) SHA-256 Hash
   Full SHA256 hash of file contents, computed starting with the FILE_ID field, proceeding over the subsequent file contents up to the start of this field.

   All data is added to the hash in unencrypted form. If encryption is used, this hash field is encrypted before being stored. This will result in any bit error or tampering of the encrypted byte stream result in a different unencrypted hash computation result which cannot be resaved into the .bl2 file without knowing the encryption secret.

   If encryption is not used, this field can be set to all 0's (such as from device read back), but when feasible to compute it (such as on a PC), this field should be correctly set to the SHA-256 hash over the specified range.

2. *CRC32*: 4 byte (32-bit) Cyclic Redundancy Check code.
   CRC32 (IEEE 802.3/PKZIP polynomial) of the file contents, computed starting with the FILE_ID field, proceeding over all subsequent file contents up to the start of this field. If the file is encrypted, this CRC32 is to be computed over the normal encrypted file contents as it would be stored on an untrusted medium. This field facilitates low-overhead communications-layer data corruption detection, not protection against any type of deliberate data tampering.