2.2 Intel HEX file Specification

Hexmate reads and processes HEX files, in particular, those files conforming to the Hexadecimal Object File Format developed by Intel®. Other forms of HEX files and other file formats cannot be processed by Hexmate. The information in this section describes the HEX file specification, as interpreted by Hexmate when processing input files.

Intel HEX files store binary program data using ASCII characters. One character is used to store each nibble of the binary data, so, for example, the byte value 0xFE would be stored in a HEX file as the ASCII byte 0x46 (representing the most significant nibble of the value, `F`) followed by 0x45 (`E`). If you view a HEX file with a text editor, it will show these two bytes as the characters `FE`, since it assumes the data is ASCII and maps it accordingly for display. Upper case hexadecimal letters are typically used in HEX files, but Hexmate will accept either case.

A HEX file consists of a number of records, with at most one record per line.

A record begins with a Record Mark, that being a colon (`:`) character. Any other characters encountered at the beginning of a line are interpreted by Hexmate as a comment until either a colon or new-line character is encountered. A line in a HEX file is valid if it is:

- entirely a valid record (described below)
- blank (only contains a new-line character)
- entirely a comment (contains any characters excluding those that define a valid record)
- a comment followed by a valid record

Once a Record Mark has been encountered in a line, it and all the following characters on that line are assumed to be part of a record. The record will be checked for syntactic correctness and any errors will be reported, but with one exception: Should a record following a comment be malformed, then the whole line will be treated as a comment and no errors will be reported.

The general form of a record, showing the different fields, is as follows.

Record Mark	Data Length	Address Offset	Record Type	Data/Argument	Checksum
`:`	*`LL`*	*`OOOO`*	*`TT`*	*`DDDD...DDDD`*	*`CC`*

The Data Length field follows the Record Mark and is a one-byte (two ASCII character) value indicating the length of the Data/Argument field in the record. This specification is required, since the length of the Data/Argument field is not fixed, and it can vary from one record to the next.

A two-byte (four character) Address Offset field follows. If a record is one that contains program data, the value in this field (and potentially information contained in other records) indicates the address at which to start programming the data in the device; otherwise, it is not used and contains the character sequence `0000`. The correct Address Offset must be specified with each record that holds data; the location for data cannot “flow on” from any previous record in the file. The mandatory inclusion of an address with each record allows records to be placed in the file in any order, subject to some limitations, described later.

Note that the Address Offset field, as its name implies, specifies an address *offset*, not an absolute address. The base address to which the offset applies is specified by special record types, discussed later in this section; however Hexmate assumes that this base address is 0 if no such special records have been previously encountered in the file.

A single byte (two characters) Record Type value follows the Address Offset and is used to indicate what sort of information the record contains. There are six types of record, tabulated below.

Record Type Characters	Record Name	Purpose
`00`	Data Record	Contains data bytes to be programmed into the device.
`01`	End-of-file Record	Indicates the final record in the file.
`02`	Extended Segment Address Record	Contains the Upper Segment Base Address.
`03`	Start Segment Address Record	Contains the execution start address when using Extended Segment Address Record data.
`04`	Extended Linear Address Record	Contains the Linear Base Address.
`05`	Start Linear Address Record	Contains the execution start address when using Linear Address Record data.

A multipurpose, variable-length Data/Argument field follows the Record Type field for most record types. This field is used to store the data to be programmed for type 0 records, store a base address for type 2 and 4 records, and store a start address for type 3 and 5 records. This field is not present for type 1 records. For type 0 Data Records, the length of the Data/Argument field is permitted to range from 1 to 255 bytes (2 to 510 characters), and this length can vary from one Data Record to the next. The bytes of data in this field assume consecutive addresses, indexed from the first byte, which has the Address Offset specified in the record.

A Checksum field makes up the last byte (2 characters) in the record for all record types and this value is used by Hexmate to minimize the chance of corrupted data being programmed. Record checksums are calculated as an 8-bit two's complement of the summation of each byte (in its binary form) in that record, starting from the Record Length byte to the last byte in the Data/Argument Field.

The following line (formatted to highlight the different fields) is an example of a valid record that might be contained in a HEX file.

```
:0400000000FEEFFFF020
```

This line begins with a colon (`:`), signaling the start of a record. There is no comment present on the line. The next two characters (`04`) indicate the length of the Data/Argument field in this record to be 4 bytes (8 characters). The following 4 bolded characters (**`0000`**) form the Address Offset, that being the value 0. This is followed by two characters (`00`) being the record type, indicating that this is a Data Record containing bytes to be programmed. The following 8 characters of data, `FEEFFFF0`, are underlined. These represent the bytes of data 0xFE, 0xEF, 0xFF, and 0xF0. These bytes have address offsets of 0 through 3, respectively. Finally, there are two characters representing the checksum value, `20`. This value is obtained by first summing the bytes 0x04, 0x00, 0x00, 0x00, 0xFE, 0xEF, 0xFF, and 0xF0 contained in the record, which yields the value 0x3E0. The 8-bit two's complement of this value is 0x20.

Another example including a comment, two Data Records, and one End-of-file Record, forming a complete HEX file is shown below. For illustrative purposes, the Address Offset in each record has been bolded and the Data/Argument field (where present) is underlined.

```
;configured for basic mode
:0400000000FDEF0FF011
:0C1FF400B88100EF00F00001FAEF0FF0E0
:00000001FF
```

Note that HEX files always use byte addresses, that is, each byte of data in each record has a unique address. Some devices use word addressing, where each unique device address might contain more than one byte of data. The program memory addressing size of Microchip device families is shown in the following table.

Device family	Program memory addressing size
8-bit Baseline, Mid-range, and Enhanced Mid-range PIC® MCUs	2 bytes (word)
PIC18 MCUs	1 byte
8-bit AVR® MCUs	2 bytes (word)
PIC24 MCUs	2 bytes (word)
dsPIC® DSCs with 24-bit instruction sets	2 bytes (word)
dsPIC DSCs with 32-bit instruction sets	1 byte
PIC32 and SAM MCUs and MPUs	1 byte

For those Microchip devices that use word addressing, two bytes of data (with two unique HEX file addresses) are needed to program one address location on the device, thus, the HEX file address for a particular byte of data will not be the same as the address at which that byte will be programmed in the device. Always be mindful that this mapping might need to be performed if you are searching a HEX file for data at a particular device address. Hexmate's `-addressing` option (see Addressing Hexmate Option) can assist with this mapping when specifying options.

The following table shows an example of how word-addressed device memory will be programmed from a HEX file record (assuming the base address is 0):

```
:0401FC00E040FEFFE2
```

Table 2-1. Mapping from HEX file address to device address for word-addressed devices

HEX file address	HEX file data	Device Address	Device Data
0x1FC	0xE0	0xFE	0x40E0
0x1FD	0x40		
0x1FE	0xFE	0xFF	0xFFFE
0x1FF	0xFF		