```
================================================================================
MCU Bootloader Tool v1.2.0 User Manual

2016-11-06  Peter S'heeren, Axiris
--------------------------------------------------------------------------------


Overview
--------

The MCU Bootloader Tool hosts devices with a bootloader that conforms to the
MCU Bootloader protocol v1. The program's main task is uploading firmware to
the device. The program can read and write flash memory, read and write EEPROM,
load and save Intel HEX files, and more.
```

```
Command Line
------------

The program accepts numerous command line arguments. Some arguments accept one
or more options.

An argument always begins with a hyphen, e.g. -serial.

An option can many things, like a path, a string, a label, a number. If you
need to include spaces in an option, place quotation marks around the text. For
example:

  -flash "C:\My Documents\AVR Files\bootloader.hex"

A number can be written in decimal, hexadecimal, or binary. The first digit
must be decimal (0..9). An indicator at the end determines the radix. If no
radix is specified, the server assumes a decimal value. Character sequence 0x
or 0X at the beginning indicates a hexadecimal value. The value may contain
underscores to improve readability. Numbers are case-insensitive. Examples:

  Decimal:        1  10d  1_655_432  255D

  Hexadecimal:    0AFh  1234_eee_h  0AbbaH 0xFF 0x1234_5678 0XAA

  Binary:         1100b  11_0011_0010_B

Specify argument -h for a concise overview. You can always specify -h even if
other arguments are already there. This is useful at times when you're entering
a list of arguments and need to browse the overview; enter -h, press enter,
have a look, recall your last entry on the command line and continue editing.
```

```
MCU Bootloader Protocol
-----------------------

The program works with bootloaders that conform to the MCU Bootloader protocol
v1. The protocol is defined in source file aximcubldr.h.

The protocol defines the following commands:

* Read bootloader information.

* Read a page of flash memory.

* Erase a page of flash memory.

* Write a page of flash memory.

* Read one or more words from EEPROM.

* Write one or more words to EEPROM.

* Run the main application in the device.

A bootloader reports a description of the device's memory spaces to the host.
The protocol defines the following memory spaces and their attributes:

* Flash memory: (a.k.a. program memory)

  - Page-oriented: all operations are performed on a single page at a time.
  - Each page is an array of words, the number of words is a power of two.
  - Each words comprises one or more bytes.
  - A flash page can be read.
  - A flash page can be erased. All bits are set to one.
  - A flash page can be written. The page is always erased before being
    written.

* EEPROM:

  - Word-oriented.
  - Each word comprises one or more bytes.
  - The bootloader specifies a buffer size for EEPROM operations. The host can
    perform an operation on one or more words at once limited to the specified
    buffer size.
  - One or more words be be read.
  - One or more words can be written. The words are always erased before being
    written. Erasing means all bits are set to one.
```

Bootloader

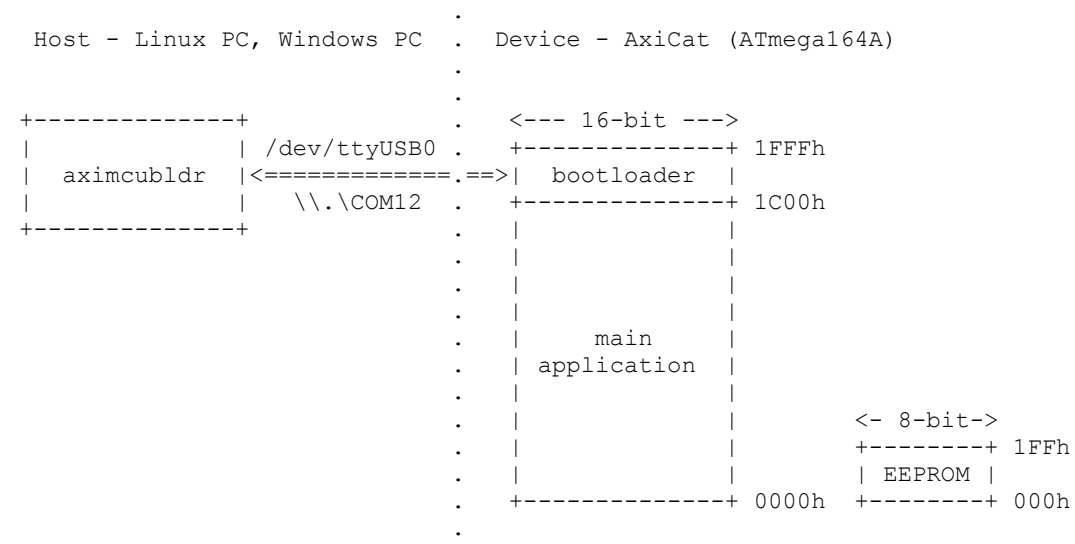
A bootloader is a small program that resides in non-volatile program memory of a device. The bootloader is the first program that's executed when the device is powered on or reset.

To ensure normal operation of the device, the bootloader simply passes control to the main application that is also stored in program memory. That is, unless the bootloader is activated.

When the bootloader is activated, it retains control of the device and waits for commands that are sent over an interface. The party that sends commands is called the host. The interface between host and bootloader typically is a serial bus.

Each bootloader implements a mechanism for activating the bootloader when the device is powered on or reset. Examples include holding down a button, shorting two pads on the board, sending a command within a defined period of time after the device is powered on or reset.

A bootloader is tailor-made for the device it serves. This is logical, as the device provides the means for interfacing with the host and for activating the bootloader, and the bootloader has to be aware of the memory spaces that can be programmed.



The block diagram visualizes the program connected with an AxiCat. The AxiCat incorporates an FT245R USB FIFO chip that is enumerated as a serial port on the host.

A bootloader can access the entire EEPROM and the area of program memory intended for the main application. A bootloader can't overwrite itself.

```
Connection with Bootloader
--------------------------

Since the bootloader protocol only requires a TX and RX channel, various
hardware and software interfaces can be used by a bootloader. Currently, the
program supports the following interface types:

* The bootloader is accessible by means of a serial path:

  -serial PATH

    PATH: Serial path, e.g. /dev/ttyUSB0 in Linux, \\.\COM12 in Windows.

* The bootloader is linked with AxiCat UART #0 or UART #1:

  -axicat PATH U

    PATH: Serial path, e.g. /dev/ttyUSB0 in Linux, \\.\COM12 in Windows.
    U:    Specify u0 for UART #0, u1 for UART #1.

* Fake bootloader. If you want to run the program without an actual device, you
  can define a fake bootloader:

  -fake FLP FLWP FLBW EEW EEBW EEWB

    FLP:  Flash memory pages (1..65536).
    FLWP: Flash memory words per page, power of 2 (1, 2, 4, ..., 32768).
    FLBW: Flash memory bytes per word (1..16).
    EEW:  EEPROM words (1..65536).
    EEBW: EEPROM bytes per word (1..16).
    EEWB: EEPROM words per buffer (1..65536).

The program uses a default serial speed of 115200 baudrate. You can override
the speed with -baud.

Note that -baud only has meaning for serial interfaces that are based on an
actual UART. E.g. specifying -baud with AxiCat has no effect since AxiCat uses
an FT245R USB FIFO chip. As such there's no serial bus, even though the host
accesses AxiCat using a serial path.
```

```
Data Buffers for Flash and EEPROM
---------------------------------

Argument -write instructs the program to write data to flash memory or EEPROM.
The data bytes to be written to the device are first stored in data buffers:

* Data buffer for flash memory.

* Data buffer for EEPROM.

The program always stores bytes in the data buffers, nomatter the actual bytes-
per-word the device implements. E.g. if the EEPROM has a word size of four
bytes, you can still write a single byte to the data buffer for EEPROM. When
-write is specified, the program applies an algorithm to make sure this byte
will be written to the correct offset in the target word without modifying the
other bytes in the word.

The bytes can come from various sources:

* An Intel HEX file.

* A raw file.

* Bytes specified on the command line.

* The device itself.

Arguments -flash and -eeprom store bytes in their corresponding data buffer.
Options are:

  -flash ...            Select data buffer for flash memory:

     FILE                  Load IHEX file into data buffer.

     ihex FILE             Load IHEX file into data buffer.

     raw ...               Load file contents into data buffer:

        AD FILE               Starting at address AD.

        page P FILE           Starting at page P.

        AD page P FILE        Starting at page P plus byte offset AD.

     bytes ...             Store an array of bytes in data buffer:

        AD BYTES              Starting at address AD.

        page P BYTES          Starting at page P.

        AD page P BYTES       Starting at page P plus byte offset AD.

     dev ...               Read from device into data buffer:

        all                   Entire memory space.

        AD to AD2             AD2-AD+1 bytes starting at address AD.

        AD cnt N              N bytes starting at address AD.

        AD to AD2 page P      AD2-AD+1 bytes starting at page P plus byte
                              offset AD.

        AD cnt N page P       N bytes starting at page P plus byte offset AD.
```

```
        page P                Page P.
        page P to P2          Pages P..P2.
        page P cnt N          N pages starting at page P.
        AD page P             One page starting at page P plus byte offset AD.
        AD page P to P2       P2-P+1 pages starting at page P plus byte offset AD.
        AD page P cnt N       N pages starting at page P plus byte offset AD.
  -eeprom ...           Select data buffer for EEPROM:
     FILE                  Load IHEX file into data buffer.
     ihex FILE             Load IHEX file into data buffer.
     raw AD FILE           Load file contents into data buffer starting at
                           address AD.
     bytes AD BYTES        Store an array of bytes in data buffer starting at
                           address AD.
     dev ...               Read from device into data buffer:
        all                   Entire memory space.
        AD to AD2             AD2-AD+1 bytes starting at address AD.
        AD cnt N              N bytes starting at address AD.

Examples:

  -flash ihex program.hex

  -flash raw 0A400h image.bin

  -flash raw page 20 image.bin

  -flash bytes 5 page 10 65 66 66 65 0

  -flash dev all

  -flash dev page 4 to 7

  -eeprom settings.eep

  -eeprom bytes 0000h 120 35 6 80

  -eeprom dev 0000h cnt 40

You can specify these arguments multiple times. Each argument is added to a
list. There's a list for -flash arguments and a list for -eeprom arguments.

After the program has processed the command line, it iterates through each list
and adds the data from the specified source to the corresponding data buffer.
```

Flash memory Details

Flash memory, a.k.a. program memory, is the place where program code is stored. The processor executes instructions stored in this memory. Flash memory also contains constant data, ASCII strings for example.

From the instruction viewpoint, program memory is read-only. It's not possible to directly write data to flash. Nonetheless, the processor implements dedicated instructions for erasing and writing pages of flash memory. These instructions indirectly modify flash memory by controlling specific hardware. These instructions are the building blocks for constructing the bootloader.

The program's data buffer for flash memory is page-oriented as well. The data buffer is organized as a page table. The size of the pages is the same as the page size of the device, although it stores bytes rather than th actual bytes-per-word.

The data buffer is initially empty, no pages are present in the page table. As data is added (see -flash argument), the program adds the necessary pages for the targetted address ranges and stores the bytes in these pages. If a page for an address range already exists, the data bytes will be overwritten.

A new page is always initialized to one-bits before the data bytes are stored. This is consistent with the state of erased flash memory where all bits are one.

When -write is specified, the program will only write the pages that are present in the page table.

Example: ATtiny25

The ATtiny25 microcontroller has 1024 words or 2048 bytes of flash memory. A page consists of 16 words.

Now let's see how the program builds up the page table of the data buffer for various example -flash arguments.

```
-flash bytes 0004h 41h 42h 43h 44h

+-[PT]-+
|      | Page table
+------+
   |
   |  +-[PAGE]-------------------------------------------------+
   +--|+00h: FF FF FF FF 41 42 43 44 FF FF FF FF FF FF FF FF | ad. 0000h
      |+10h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | page 0
      +-------------------------------------------------------+
```

The program adds page 0 and stores the data bytes at 0004h..0007h.

```
-flash bytes 0090h 1 2 3 4 5 -flash bytes 0092h 6 7

+-[PT]-+
|      | Page table
+------+
   |
   |  +-[PAGE]-----------------------------------------------+
   +--|+00h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | ad. 0080h
      |+10h: 01 02 06 07 05 FF FF FF FF FF FF FF FF FF FF FF | page 4
      +-------------------------------------------------------+
```

The first -flash argument provides data bytes for addresses 0090h..0094h. The program adds page 4 and stores the data bytes.

The second -flash argument carries data bytes for address range 0092h..0093h. Since the page is already present, the program overwrites the address range with the specified data bytes.

```
-flash bytes 025Eh 61h 62h 63h 64h

+-[PT]-+
|      | Page table
+------+
   |
   |  +-[PAGE]-----------------------------------------------+
   +--|+00h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | ad. 0240h
   |  |+10h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF 61 62 | page 18
   |  +-------------------------------------------------------+
   |
   |  +-[PAGE]-----------------------------------------------+
   +--|+00h: 63 64 FF FF FF FF FF FF FF FF FF FF FF FF FF FF | ad. 0260h
      |+10h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | page 19
      +-------------------------------------------------------+
```

The specified data bytes cross a page boundary, hence the program adds two consecutive pages that span the specified address range (025Eh..0261h).

```
-flash bytes 025Eh 61h 62h 63h 64h -flash bytes 0004h 41h 42h 43h 44h

+-[PT]-+
|      | Page table
+------+
   |
   |  +-[PAGE]-----------------------------------------------+
   +--|+00h: FF FF FF FF 41 42 43 44 FF FF FF FF FF FF FF FF | ad. 0000h
   |  |+10h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | page 0
   |  +-------------------------------------------------------+
   |
   |  +-[PAGE]-----------------------------------------------+
   +--|+00h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | ad. 0240h
   |  |+10h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF 61 62 | page 18
   |  +-------------------------------------------------------+
   |
   |  +-[PAGE]-----------------------------------------------+
   +--|+00h: 63 64 FF FF FF FF FF FF FF FF FF FF FF FF FF FF | ad. 0260h
      |+10h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF | page 19
      +-------------------------------------------------------+
```

The program first adds pages 18 and 19, then page 0.

Note that the pages are added in the order that the -flash arguments appear in the program's command line.

The examples use literal bytes for simplicity. The program manages the page table similarly when you specify other options with the -flash argument.

You can dump the data buffer with argument -dump buf flash. Let's take the last example:

```
$ aximcubldr -v -serial /dev/ttyUSB0 -dump buf flash -flash bytes 025Eh 61h
62h 63h 64h -flash bytes 0004h 41h 42h 43h 44h
```

EEPROM Details

The program's data buffer for EEPROM keeps track of each byte that's being stored. When the program writes to EEPROM, it'll only write those bytes that were marked as stored.

Since the data buffer stores individual bytes, it's possible to write a single byte to EEPROM even if the actual word size of the EEPROM is two or more bytes. The program checks for unaligned writes. If a data word will be overwritten partially, the program first reads the data word, modifies the necessary bytes, and writes back the word to EEPROM.

You can dump the data buffer with argument -dump buf eeprom:

```
$ aximcubldr -v -serial /dev/ttyUSB0 -dump buf eeprom -eeprom bytes 0020h 120
```

When the program dumps large chunks of EEPROM memory, you may notice a break on 256-byte boundaries. This is nothing to worry about. Internally, the data buffer for EEPROM uses a page table, similar to the data buffer for flash memory. The page size is set to 256 bytes, hence the break. The page size has no influence on the program's EEPROM operations whatsoever. Here's an example of how to observe a break:

```
$ aximcubldr -v -serial /dev/ttyUSB0 -dump buf eeprom -eeprom bytes 00FCh 1 2
  3 4 5 6 7 8
```

```
Erase Flash Memory
------------------

Argument -ef commands the program to erase the flash memory.
```

```
Erase EEPROM
------------

Argument -ee commands the program to erase the EEPROM.
```

```
Write to Device
---------------

Argument -write instructs the program to write the contents of the data buffers
to their respective memory space.

When -write is combined with -ef, the program will erase and write flash memory
in one pass.

When -write is combined with -ee, the program will erase and write EEPROM in
one pass.

Argument -write assumes a verification of the written data. You overrule this
assumption and omit the verification step with -noverify.

Examples:

    -write -ef -flash ihex main.hex

    -write -noverify -flash raw 0F000h image.bin
```

```
Verify Device
-------------

Argument -verify tells the program to verify the contents of the data buffers
with the data stored in the device.

Examples:

    -verify -flash ihex main.hex

    -verify -eeprom bytes 20h 65 66 67 68
```

```
Save Data
---------

The -save argument takes one or more of the following options:

  flash ihex FILE       Save data buffor for flash memory to IHEX file.

  flash FILE            Save data buffor for flash memory to IHEX file.

  eeprom ihex FILE      Save data buffor for EEPROM to IHEX file.

  eeprom FILE           Save data buffor for EEPROM to IHEX file.

Examples:

  -save flash ihex core.hex

  -save flash tv.hex eeprom tv.eep
```

```
Dump Information
----------------

Argument -dump instructs the program to print certain information to standard
output. The argument takes one or more of the following options:

  buf flash          Dump the data buffer for flash memory.

  buf eeprom         Dump the data buffer for EEPROM.

  dev flash          Read the entire flash memory and dump.

  dev eeprom         Read the entire EEPROM and dump.

Examples:

  -dump dev eeprom
```

```
Run
---

Argument -run instructs the bootloader to transfer control to the main
application.
```

```
Examples
--------

The examples presented here are applicable in Linux. It's assumed that the
serial path is /dev/ttyUSB0. If this is not the case, change to the correct
serial path, for example:

  $ aximcubldr -v -serial /dev/ttyUSB4

In case your Linux requires root privileges for accessing the serial path, run
the examples as root.

If you want to run the examples in Windows, make sure the serial path is
prefixed with \\.\ to access the Win32 device namespace. For example:

  > aximcubldr.exe -v -axicat \\.\COM25 u0

Examples:

  $ aximcubldr -v -serial /dev/ttyUSB0

    Display information about the connected device.

  $ aximcubldr -v -axicat /dev/ttyUSB0 u1

    Display information about the device connected to UART #1 of the AxiCat.

  $ aximcubldr -v -fake 50 64 4 1024 1 64

    Display information about the specified fake bootloader.

  $ aximcubldr -serial /dev/ttyUSB0 -dump dev flash dev eeprom

    Dump the device's flash memory and EEPROM.

  $ aximcubldr -v -serial /dev/ttyUSB0 -ef -ee

    Erase flash memory and EEPROM.

  $ aximcubldr -v -serial /dev/ttyUSB0 -write -flash main.hex

    The program will:
    1. Load the IHEX file into the data buffer for flash memory.
    2. Write the bytes in the data buffer to flash memory.
    3. Verify the written data in flash memory.

  $ aximcubldr -v -serial /dev/ttyUSB0 -verify -flash main.hex

    The program will:
    1. Load the IHEX file into the data buffer for flash memory.
    2. Verify the data in flash memory.

  $ aximcubldr -v -serial /dev/ttyUSB0 -eeprom bytes 20h 0 10h 3Ah 1010b
    -write

    The program will:
    1. Store the four bytes at address 20h in the data buffer for EEPROM.
    2. Write the specified data to EEPROM.
    3. Verify the written data in EEPROM.
```

```
$ aximcubldr -v -serial /dev/ttyUSB0 -flash part1.hex -flash part2.hex -dump
  buf flash

  Dump the flash pages that would be written to the device.

  The program will:
  1. Load two IHEX files into the data buffer for flash memory.
  2. Dump the data buffer for flash memory.

$ aximcubldr -v -serial /dev/ttyUSB0 -flash dev all -save flash fw.hex

  Read the entire flash memory and save to Intel HEX file.

$ aximcubldr -v -serial /dev/ttyUSB0 -eeprom dev 0020h to 003Fh -save eeprom
  chunk.eep

  Read a section of EEPROM and save to Intel HEX file.
```

```
Document History
----------------

2016-11-06  Peter S'heeren, Axiris

  * First release.
```