ArduinoBootloader

Release 0.0.6

Nov 17, 2020

Contents:

This module encapsulates access for the Arduino bootloader that uses a serial port to connect. It supports the protocols Stk500-v1 and Stk500-v2 for Atmel AVR8 CPUs.

ArduinoBootloader is Open Source software. I have licensed the code base as MIT, which provides almost no restrictions on the use of the code.

Other pages (online)

- project page on GitHub
- Download Page with releases
- This page, when viewed online is at https://arduinobootloader.readthedocs.io/en/latest/

CHAPTER 1

Description

ArduinoBootloader is an open source library in Python for updating the firmware of Arduino boards that use the ATmegaBOOT_168 or Stk500V2 bootloader, for example Arduino Nano or Arduino Uno or Arduino Mega 2560 and many more.

The intention is to have a class that can be imported into any Python project to update the Arduino's through the serial port.

It implements a subset of Atmel's STK-500V1 and STK500V2 protocol, using as reference the protocols implemented by Avrdude in the `arduino.c` and `wiring.c` modules.

For Arduino's using Atmel AVR8 processors there are three versions of the bootoloader available.

1.1 Up to 128 Kbytes - Old Version

For boards that have less than 128 Kbytes of Flash memory, for example Nano using the Atmega328P, etc that are marked in the Arduino IDE as older you have to use:

```
from arduinobootloader import ArduinoBootloader

def update(self):
    ab = ArduinoBootloader()
    prg = ab.select_programmer("Stk500v1")

    if prg.open(speed=57600):
```

1.2 Up to 128 Kbytes - (Optiboot)

And for the new ones (they implement the Optiboot bootloader) you have to use:

```
from arduinobootloader import ArduinoBootloader

def update(self):
    ab = ArduinoBootloader()
    prg = ab.select_programmer("Stk500v1")

    if prg.open(speed=115200):
```

1.3 More than 128 Kbytes

For boards that have processors of more than 128 Kbytes, for example the Mega 2560, use:

```
from arduinobootloader import ArduinoBootloader

def update(self):
    ab = ArduinoBootloader()
    prg = ab.select_programmer("Stk500v2")

    if prg.open(speed=115200):
```

CHAPTER 2

Install

Yo can download or clone the last version from Github, but better use pip or pip3:

```
pip install arduinobootloader
```

CHAPTER 3

Usage

The following paragraph explains the basic flow to update the firmware of an Arduino board from Python.

3.1 Requirements

If the firmware file is in Intel hexadecimal format, use the IntelHex library.

3.2 Select Programmer

Get a instance of the class with

```
ab = ArduinoBootloader()
```

and select the protocol of the programmer with

```
prg = ab.select_protocol("Stk500v1")
```

3.3 Connect

To establish the connection with the Arduino bootloader use

```
if prg.open(speed=115200):
```

that returns `True` when successful.

3.4 CPU information

The library needs the information of the CPU to know the size and count of the flash page, use the method

```
if prg.cpu_signature():
```

that returns `True` when success. The properties have the corresponding information

```
ab.cpu_page_size
ab.cpu_pages
```

3.5 Programmer information

Although the information of the programmer is not important for the update process, because the bootoloader is an emulation of the Atmel programmers, you can do it with the method

```
prg.board_request()
```

that returns `True` when success.

The properties have the information.

```
ab.programmer_name
ab.sw_version
ab.hw_version
```

3.6 Open Firmware File

Open the hexadecimal file with the method. If there are errors in the format or the file path is invalid, exceptions are thrown.

```
try:
    ih.fromfile("filename.hex", format='hex')
except FileNotFoundError:
    print("file not found")
except (AddressOverlapError, HexRecordError):
    print("error, file format")
```

3.7 Parse the Firmware in Pages

To obtain the page of the current address, use the

```
buffer = ih.tobinarray(start=address, size=ab.cpu_page_size)
```

3.8 Write Pages

For write it in the flash memory, use this method which take the buffer and the address as parameters and returns `True` when success.

```
if prg.write_memory(buffer, address):
```

3.9 Read Pages

The read for example to verify, is done in the same way, with the exception that the method returns the memory buffer. When errors returns `None`.

```
read_buffer = prg.read_memory(address, ab.cpu_page_size)
if read_buffer is None:
```

3.10 Save in a File

To save the read firmware to a hexadecimal format file, you need to buffer it in a dictionary where the key is the address of each byte on the page.

```
for i in range(0, ab.cpu_page_size):
    dict_hex[address + i] = read_buffer[i]
```

And when you have finished reading the flash, add the starting address to generate the hexadecimal and to save the file

```
dict_hex["start_addr"] = 0
ih.fromdict(dict_hex)
ih.tofile("read_filename.hex", 'hex')
```

3.11 Execute the Firmware

The bootloader begins the execution of the firmware after a period of time without receiving communication; nevertheless it is convenient to execute the function

```
prg.leave_bootloader()
```

3.12 Close Communication

Call the method to release the serial port

```
prg.close()
```

CHAPTER 4

Usage Example

Note: For UNO and Nano use “Stk500v1”, for Mega 2560 “Stk500v2”.

```
from intelhex import IntelHex
from arduinobootloader import ArduinoBootloader

def update(self):
    ih = IntelHex()
    ab = ArduinoBootloader()
    prg = ab.select_programmer("Stk500v1")

    if prg.open(speed=115200):
        if not prg.board_request():
            prg.close()
            return

        print("botloader name: {} version: {} hardware: {}".format(ab.programmer_name,
                                                                   ab.sw_version,
                                                                   ab.hw_version))

        if not prg.cpu_signature():
            prg.close()
            return

        print("cpu name: {}".format(ab.cpu_name) )

        try:
            ih.fromfile("filename.hex", format='hex')
        except (FileNotFoundError, AddressOverlapError, HexRecordError):
            return

        for address in range(0, ih.maxaddr(), ab.cpu_page_size):
            buffer = ih.tobinarray(start=address, size=ab.cpu_page_size)
```

(continues on next page)

(continued from previous page)

```
            if not prg.write_memory(buffer, address):
                print("Write error")
                prg.leave_bootloader()
                prg.close()
                return

        for address in range(0, ih.maxaddr(), ab.cpu_page_size):
            buffer = ih.tobinarray(start=address, size=ab.cpu_page_size)
            read_buffer = prg.read_memory(address, ab.cpu_page_size)
            if read_buffer is None:
                print("Read error")
                break

            if buffer != read_buffer:
                print("File not match")
                break

        prg.leave_bootloader()
        prg.close()
```

CHAPTER 5

Kivy App Example

As an example of use, there is an APP in KivyMd and Kivy that exposes through a GUI all the methods required to update and verify the firmware.

The first example shows the upgrade of a Nano board that have the OptiBoot bootolader. Select STK500-V1 at 115200 baud.

The second example shows the upgrade of a Mega board with 2560 processor. STK500-V2 must be selected at 115200 baud.

CHAPTER 6

Script Example

The Script folder contains arduinoflash.py file that allows update or read the firmware of Arduino boards.

One of the purposes is to show the use of the PyArduinoBootloader library in conjunction with the IntelHex library to process hexadecimal files.

Use the argparse library, to read the command line (file and options).

And to indicate the progress the progressbar2 library.

The following capture shows the reading of the flash memory of an Arduino Nano board.

And the next shows the firmware update of an Arduino Nano board.

The following capture shows the reading of the flash memory of an Arduino Mega 2560 board.

And the next shows the firmware update of an Arduino Mega 2560 board.

CHAPTER 7

API

Is a Python Class for updating the firmware of Arduino boards that use Atmel AVR CPUs. For example Arduino Nano, Uno, Mega and more.

The module implements the essential parts that Avrdude uses for the arduino and wiring protocols. In turn, they are a subset of the STK500 V1 and V2 protocols respectively.

arduinobootloader.**AVR_ATMEL_CPUS = {2003718: ['ATmega8515', 64, 128], 2003719: ['ATmega8**
Dictionary with the list of Atmel AVR 8 CPUs used by Arduino boards. Contains the size in bytes and the number of pages in flash memory. The key is the processor signature which is made up of SIG1, SIG2 and SIG3.

class arduinobootloader.**ArduinoBootloader**(**args*, ***kwargs*)
Bases: object

Contains the two inner classes that support the Stk500 V1 and V2 protocols for comunicate with arduino bootloaders.

class Stk500v1(*ab*)
Bases: object

It encapsulates the communication protocol that Arduino uses for the first versions of bootoloader, which can write up to 128 K bytes of flash memory. For example: Nano, Uno, etc. The older version (ATmegaBOOT_168.c) works at 57600 baudios, the new version (OptiBoot) at 115200

board_request()
Get the firmware and hardware version of the bootloader.
Returns True when success.
Return type bool

close()
Close the communication port.

cpu_signature()
Get CPU information: name, size and count of the flash memory pages
Returns True when success.
Return type bool

get_sync()
Send the sync command whose function is to discard the reception buffers of both serial units. Set the receive unit timeout to 500mS and send the sync command up to 5 times to eliminate noise from the line.
Returns True when success.
Return type bool

leave_bootloader()
Leave programming mode and start executing the stored firmware
Returns True when success.
Return type bool

open(*port=None*, *speed=57600*)
Find and open the communication port where the Arduino is connected. Generate the reset sequence with the DTR / RTS pins. Send the sync command to verify that there is a valid bootloader.
Parameters
- **port** (*str*) – serial port identifier (example: ttyUSB0 or COM1). None for automatic board search.
- **speed** (*int*) – comunication baurate, for older bootloader use 57600.

Returns True when the serial port was opened and the connection to the board was established.
Return type bool

read_memory(*address*, *count*, *flash=True*)
Read the memory from requested address.
Parameters
- **address** (*int*) – memory address of the first byte to read. (16 bits).
- **count** (*int*) – bytes to read.
- **flash** (*bool*) – eeprom supported only by the older version of bootloader.

Returns the buffer read or None when there is error.
Return type bytearray

write_memory(*buffer*, *address*, *flash=True*)
Write the buffer to the requested address of memory.
Parameters
- **buffer** (*bytearray*) – data to write.
- **address** (*int*) – memory address of the first byte (16 bits).
- **flash** (*bool*) – for old bootloader version can be flash or eeprom.

Returns True the buffer was successfully written.
Return type bool

class Stk500v2(*ab*)
Bases: object

It encapsulates the communication protocol that Arduino uses in bootloaders with more than 128K bytes of flash memory. For example: Mega 2560 etc

board_request()
Get the firmware and hardware version of the bootloader.
Returns True when success.
Return type bool

close()
Close the communication port.

cpu_signature()
Get CPU information: name, size and count of the flash memory pages
Returns True when success.

Return type bool

get_sync()
Send the sync command
Returns True when success.
Return type bool

leave_bootloader()
Leave programming mode and start executing the stored firmware
Returns True when success.
Return type bool

open(*port=None*, *speed=115200*)
Find and open the communication port where the Arduino is connected. Generate the reset sequence with the DTR / RTS pins. Send the sync command to verify that there is a valid bootloader.
Parameters
- **port** (*str*) – serial port identifier (example: ttyUSB0 or COM1). None for automatic board search.
- **speed** (*int*) – comunication baurate (115200).

Returns True when the serial port was opened and the connection to the board was established.
Return type bool

read_memory(*address*, *count*, *flash=True*)
Read the memory from requested address.
Parameters
- **address** (*int*) – memory address of the first byte to read. (32 bits).
- **count** (*int*) – bytes to read.
- **flash** (*bool*) – stk500v2 version only supports flash.

Returns the buffer read or None when there is error.
Return type bytearray

write_memory(*buffer*, *address*, *flash=True*)
Write the buffer to the requested address of memory.
Parameters
- **buffer** (*bytearray*) – data to write.
- **address** (*int*) – memory address of the first byte (32 bits).
- **flash** (*bool*) – stk500v2 version only supports flash.

Returns True the buffer was successfully written.
Return type bool

close()
Close the serial communication port.

cpu_name
Dictionary cpu name

Setter name

Type str

cpu_page_size
CPU flash page size in bytes, not words.

Setter size

Type int

cpu_pages
CPU flash pages

Setter pages

Type int

hw_version
bootloader hardware version

Setter version

Type int

open(*port=None*, *speed=115200*)
Find and open the communication port where the Arduino is connected. Generate the reset sequence with the DTR / RTS pins. Send the sync command to verify that there is a valid bootloader.

Parameters

- **port** (*str*) – serial port identifier (example: ttyUSB0 or COM1). None for automatic board search.
- **speed** (*int*) – comunication baurate.

Returns True when the serial port was opened and the connection to the board was established.

Return type bool

programmer_name
Name given by Atmel to its programmers, for example (ISP_V2). Optiboot returns an empty string to decrease the footprint of the bootloader.

Setter name

Type str

select_programmer(*protocol*)
Select the communication protocol to connect with the Arduino bootloader.

Parameters **protocol** (*str*) – arduino bootloader can be: Stk500v1 or Stk500v2

Returns None for unknow protocol

Return type object

sw_version
botloader sotware version

Setter version

Type str

arduinobootloader.**CMD_GET_PARAMETER = 3**
Bootloader information of the Stk500v2 Protocol

arduinobootloader.**CMD_LEAVE_PROGMODE_ISP = 17**
Leave the programmer mode of the Stk500v2 Protocol

arduinobootloader.**CMD_LOAD_ADDRESS = 6**
Set the flash adddress of the Stk500v2 Protocol

arduinobootloader.**CMD_PROGRAM_FLASH_ISP = 19**
Write the flash of the Stk500v2 Protocol

arduinobootloader.**CMD_READ_FLASH_ISP = 20**
Read the flash of the Stk500v2 Protocol

arduinobootloader.**CMD_SIGN_ON = 1**
Synchronize the communication of the Stk500v2 Protocol

arduinobootloader.**CMD_SPI_MULTI = 29**
Cpu information of the Stk500v2 Protocol

arduinobootloader.**CPU_SIG1 = 0**
Cpu signature part 1

arduinobootloader.**CPU_SIG2 = 1**
Cpu signature part 2

arduinobootloader.**CPU_SIG3 = 2**
Cpu signature part 3

arduinobootloader.**MESSAGE_START = 27**
Start message of the Stk500v2 header (ESC = 27 decimal)

arduinobootloader.**OPT_HW_VERSION = b'\x90'**
Hardware version of the bootloader

arduinobootloader.**OPT_SW_MAJOR = b'\x91'**
Major software bootloader version

arduinobootloader.**OPT_SW_MINOR = b'\x92'**
Minor software bootloader version

arduinobootloader.**RESP_STK_IN_SYNC = 20**
Start message of the Stk500v1

arduinobootloader.**RESP_STK_OK = 16**
End message of the Stk500v1

arduinobootloader.**STATUS_CMD_OK = 0**
The command was successful

arduinobootloader.**TOKEN = 14**
End message of the Stk500v2 header (ESC = 27 decimal)

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

O

P

R

S

T

W