bincopy Documentation

Release 16.9.1

Erik Moqvist

Apr 24, 2020

Contents

CHAPTER 1

About

Mangling of various file formats that conveys binary information (Motorola S-Record, Intel HEX, TI-TXT and binary files).

Project homepage: https://github.com/eerimoq/bincopy

Documentation: https://bincopy.readthedocs.io

CHAPTER 2

Installation

```
pip install bincopy
```

CHAPTER 3

Example usage

3.1 Scripting

A basic example converting from Intel HEX to Intel HEX, SREC, binary, array and hexdump formats:

```
>>> import bincopy
>>> f = bincopy.BinFile("tests/files/in.hex")
>>> print(f.as_ihex())
:20010000214601360121470136007EFE09D219012146017E17C20001FF5F16002148011979
:20012000194E79234623965778239EDA3F01B2CA3F0156702B5E712B722B7321460134219F
:00000001FF

>>> print(f.as_srec())
S32500000100214601360121470136007EFE09D219012146017E17C20001FF5F16002148011973
S32500000120194E79234623965778239EDA3F01B2CA3F0156702B5E712B722B73214601342199
S5030002FA

>>> print(f.as_ti_txt())
@0100
21 46 01 36 01 21 47 01 36 00 7E FE 09 D2 19 01
21 46 01 7E 17 C2 00 01 FF 5F 16 00 21 48 01 19
19 4E 79 23 46 23 96 57 78 23 9E DA 3F 01 B2 CA
3F 01 56 70 2B 5E 71 2B 72 2B 73 21 46 01 34 21
q

>>> f.as_binary()
bytearray(b'!F\x016\x01!G\x016\x00~\xfe\t\xd2\x19\x01!F\x01~\x17\xc2\x00\x01
\xff_\x16\x00!H\x01\x19\x19Ny#F#\x96Wx#\x9e\xda?\x01\xb2\xca?\x01Vp+^q+r+s!
F\x014!')
>>> list(f.segments)
[Segment(address=256, data=bytearray(b'!F\x016\x01!G\x016\x00~\xfe\t\xd2\x19\x01
!F\x01~\x17\xc2\x00\x01\xff_\x16\x00!H\x01\x19\x19Ny#F#\x96Wx#\x9e\xda?\x01
\xb2\xca?\x01Vp+^q+r+s!F\x014!'))]
>>> f.minimum_address
```

(continues on next page)

(continued from previous page)

```
256
>>> f.maximum_address
320
>>> len(f)
64
>>> f[f.minimum_address]
33
>>> f[f.minimum_address:f.minimum_address + 1]
bytearray(b'!')
```

See the test suite for additional examples.

3.2 Command line tool

3.2.1 The info subcommand

Print general information about given binary format file(s).

```
$ bincopy info tests/files/in.hex
File:                   tests/files/in.hex
Data ranges:

    0x00000100 - 0x00000140 (64 bytes)
```

3.2.2 The convert subcommand

Convert file(s) from one format to another.

```
$ bincopy convert -i ihex -o srec tests/files/in.hex -
S3250000010021460136012147013600 7EFE09D219012146017E17C20001FF5F16002148011973
S32500000120194E79234623965778239EDA3F01B2CA3F0156702B5E712B722B73214601342199
S5030002FA
$ bincopy convert -i binary -o hexdump tests/files/in.hex -
00000000  3a 32 30 30 31 30 30 30  30 32 31 34 36 30 31 33  |:200100002146013|
00000010  36 30 31 32 31 34 37 30  31 33 36 30 30 37 45 46  |601214701360 07EF|
00000020  45 30 39 44 32 31 39 30  31 32 31 34 36 30 31 37  |E09D219012146017|
00000030  45 31 37 43 32 30 30 30  31 46 46 35 46 31 36 30  |E17C20001FF5F160|
00000040  30 32 31 34 38 30 31 31  39 37 39 0a 3a 32 30 30  |02148011979.:200|
00000050  31 32 30 30 30 31 39 34  45 37 39 32 33 34 36 32  |12000194E7923462|
00000060  33 39 36 35 37 37 38 32  33 39 45 44 41 33 46 30  |3965778239EDA3F0|
00000070  31 42 32 43 41 33 46 30  31 35 36 37 30 32 42 35  |1B2CA3F0156702B5|
00000080  45 37 31 32 42 37 32 32  42 37 33 32 31 34 36 30  |E712B722B7321460|
00000090  31 33 34 32 31 39 46 0a  3a 30 30 30 30 30 30 30  |134219F.:0000000|
000000a0  31 46 46 0a                                       |1FF.            |
```

Concatename two or more files.

```
$ bincopy convert -o srec tests/files/in.s19 tests/files/convert.s19 -
S00F000068656C6C6F202020202000003C
S325000000007C0802A6900100049421FFF07C6C1B787C8C23783C600000386300004BFFFFE5F2
S3250000002039800000 7D83637880010014382100107C0803A64E80002048656C6C6F20776F13
S30B00000040726C642E0A003A
```

(continues on next page)

(continued from previous page)

```
S3250000010021460136012147013600 7EFE09D219012146017E17C20001FF5F16002148011973
S32500000120194E79234623965778239EDA3F01B2CA3F0156702B5E712B722B73214601342199
S5030005F7
S70500000000FA
```

3.2.3 The pretty subcommand

Easy to read Motorola S-Record, Intel HEX and TI TXT files with the pretty subcommand.

```
> bincopy pretty tests/files/in.s19
S00F000068656C6C6F202020202000003C (header)
S11F00007C0802A6900100049421FFF07C6C1B787C8C23783C6000003863000026 (data)
S11F001C4BFFFFE5398000007D836378800100143821001 07C0803A64E800020E9 (data)
S1110038 48656C6C6F20776F726C642E0A0042 (data)
S5030003F9 (count)
S9030000FC (start address)
```

```
> bincopy pretty tests/files/in.hex
:20010000214601360121470136007EFE09D219012146017E17C20001FF5F16002148011979 (data)
:20012000194E79234623965778239EDA3F01B2CA3F0156702B5E712B722B7321460134219F (data)
:00000001FF (end of file)
```

```
> bincopy pretty tests/files/in.hex.txt
@0100 (segment address)
21 46 01 36 01 21 47 01 36 00 7E FE 09 D2 19 01 (data)
21 46 01 7E 17 C2 00 01 FF 5F 16 00 21 48 01 19 (data)
19 4E 79 23 46 23 96 57 78 23 9E DA 3F 01 B2 CA (data)
3F 01 56 70 2B 5E 71 2B 72 2B 73 21 46 01 34 21 (data)
q (end of file)
```

CHAPTER 4

Contributing

1. Fork the repository.
2. Install prerequisites.

```
pip install -r requirements.txt
```

3. Implement the new feature or bug fix.
4. Implement test case(s) to ensure that future changes do not break legacy.
5. Run the tests.

```
make test
```

6. Create a pull request.

CHAPTER 5

Functions and classes

class bincopy.**BinFile**(*filenames=None*, *overwrite=False*, *word_size_bits=8*, *header_encoding='utf-8'*)

A binary file.

filenames may be a single file or a list of files. Each file is opened and its data added, given that the format is Motorola S-Records, Intel HEX or TI-TXT.

Set *overwrite* to `True` to allow already added data to be overwritten.

word_size_bits is the number of bits per word.

header_encoding is the encoding used to encode and decode the file header (if any). Give as `None` to disable encoding, leaving the header as an untouched bytes object.

add(*data*, *overwrite=False*)

Add given data string by guessing its format. The format must be Motorola S-Records, Intel HEX or TI-TXT. Set *overwrite* to `True` to allow already added data to be overwritten.

add_binary(*data*, *address=0*, *overwrite=False*)

Add given data at given address. Set *overwrite* to `True` to allow already added data to be overwritten.

add_binary_file(*filename*, *address=0*, *overwrite=False*)

Open given binary file and add its contents. Set *overwrite* to `True` to allow already added data to be overwritten.

add_file(*filename*, *overwrite=False*)

Open given file and add its data by guessing its format. The format must be Motorola S-Records, Intel HEX or TI-TXT. Set *overwrite* to `True` to allow already added data to be overwritten.

add_ihex(*records*, *overwrite=False*)

Add given Intel HEX records string. Set *overwrite* to `True` to allow already added data to be overwritten.

add_ihex_file(*filename*, *overwrite=False*)

Open given Intel HEX file and add its records. Set *overwrite* to `True` to allow already added data to be overwritten.

add_srec(*records*, *overwrite=False*)

Add given Motorola S-Records string. Set *overwrite* to `True` to allow already added data to be overwritten.

add_srec_file(*filename*, *overwrite=False*)
Open given Motorola S-Records file and add its records. Set *overwrite* to `True` to allow already added data to be overwritten.

add_ti_txt(*lines*, *overwrite=False*)
Add given TI-TXT string *lines*. Set *overwrite* to `True` to allow already added data to be overwritten.

add_ti_txt_file(*filename*, *overwrite=False*)
Open given TI-TXT file and add its contents. Set *overwrite* to `True` to allow already added data to be overwritten.

as_array(*minimum_address=None*, *padding=None*, *separator=', '*)
Format the binary file as a string values separated by given separator *separator*. This function can be used to generate array initialization code for C and other languages.

minimum_address is the absolute minimum address of the resulting binary data. By default this is the minimum address in the binary.

padding is the word value of the padding between non-adjacent segments. Give as a bytes object of length 1 when the word size is 8 bits, length 2 when the word size is 16 bits, and so on. By default the padding is `b'\xff' * word_size_bytes`.

```
>>> binfile.as_array()
'0x21, 0x46, 0x01, 0x36, 0x01, 0x21, 0x47, 0x01, 0x36, 0x00, 0x7e,
 0xfe, 0x09, 0xd2, 0x19, 0x01, 0x21, 0x46, 0x01, 0x7e, 0x17, 0xc2,
 0x00, 0x01, 0xff, 0x5f, 0x16, 0x00, 0x21, 0x48, 0x01, 0x19, 0x19,
 0x4e, 0x79, 0x23, 0x46, 0x23, 0x96, 0x57, 0x78, 0x23, 0x9e, 0xda,
 0x3f, 0x01, 0xb2, 0xca, 0x3f, 0x01, 0x56, 0x70, 0x2b, 0x5e, 0x71,
 0x2b, 0x72, 0x2b, 0x73, 0x21, 0x46, 0x01, 0x34, 0x21'
```

as_binary(*minimum_address=None*, *maximum_address=None*, *padding=None*)
Return a byte string of all data within given address range.

minimum_address is the absolute minimum address of the resulting binary data (including). By default this is the minimum address in the binary.

maximum_address is the absolute maximum address of the resulting binary data (excluding). By default this is the maximum address in the binary plus one.

padding is the word value of the padding between non-adjacent segments. Give as a bytes object of length 1 when the word size is 8 bits, length 2 when the word size is 16 bits, and so on. By default the padding is `b'\xff' * word_size_bytes`.

```
>>> binfile.as_binary()
bytearray(b'!F\x016\x01!G\x016\x00~\xfe\t\xd2\x19\x01!F\x01~\x17\xc2\x00\x01
\xff_\x16\x00!H\x01\x19\x19Ny#F#\x96Wx#\x9e\xda?\x01\xb2\xca?\x01Vp+^q+r+s!
F\x014!')
```

as_hexdump()
Format the binary file as a hexdump and return it as a string.

```
>>> print(binfile.as_hexdump())
00000100  21 46 01 36 01 21 47 01  36 00 7e fe 09 d2 19 01  |!F.6.!G.6.~.....|
00000110  21 46 01 7e 17 c2 00 01  ff 5f 16 00 21 48 01 19  |!F.~....._..!H..|
00000120  19 4e 79 23 46 23 96 57  78 23 9e da 3f 01 b2 ca  |.Ny#F#.Wx#..?...|
00000130  3f 01 56 70 2b 5e 71 2b  72 2b 73 21 46 01 34 21  |?.Vp+^q+r+s!F.4!|
```

as_ihex(*number_of_data_bytes=32*, *address_length_bits=32*)
Format the binary file as Intel HEX records and return them as a string.

number_of_data_bytes is the number of data bytes in each record.

address_length_bits is the number of address bits in each record.

```
>>> print(binfile.as_ihex())
:20010000214601360121470136007EFE09D219012146017E17C20001FF5F16002148011979
:20012000194E79234623965778239EDA3F01B2CA3F0156702B5E712B722B7321460134219F
:00000001FF
```

as_srec(*number_of_data_bytes=32*, *address_length_bits=32*)
Format the binary file as Motorola S-Records records and return them as a string.

number_of_data_bytes is the number of data bytes in each record.

address_length_bits is the number of address bits in each record.

```
>>> print(binfile.as_srec())
S32500000100214601360121470136007EFE09D219012146017E17C20001FF5F16002148011973
S32500000120194E79234623965778239EDA3F01B2CA3F0156702B5E712B722B73214601342199
S5030002FA
```

as_ti_txt()
Format the binary file as a TI-TXT file and return it as a string.

```
>>> print(binfile.as_ti_txt())
@0100
21 46 01 36 01 21 47 01 36 00 7E FE 09 D2 19 01
21 46 01 7E 17 C2 00 01 FF 5F 16 00 21 48 01 19
19 4E 79 23 46 23 96 57 78 23 9E DA 3F 01 B2 CA
3F 01 56 70 2B 5E 71 2B 72 2B 73 21 46 01 34 21
q
```

crop(*minimum_address*, *maximum_address*)
Keep given range and discard the rest.

minimum_address is the first word address to keep (including).

maximum_address is the last word address to keep (excluding).

exclude(*minimum_address*, *maximum_address*)
Exclude given range and keep the rest.

minimum_address is the first word address to exclude (including).

maximum_address is the last word address to exclude (excluding).

execution_start_address
The execution start address, or `None` if missing.

fill(*value=None*, *max_words=None*)
Fill all empty space between segments.

value is the value which is used to fill the empty space. By default the value is `b'\xff' * word_size_bytes`.

max_words is the maximum number of words to fill between the segments. Empty space which larger than this is not touched. If `None`, all empty space is filled.

header
The binary file header, or `None` if missing. See *BinFile's header_encoding* argument for encoding options.

info()
Return a string of human readable information about the binary file.

```
>>> print(binfile.info())
Data ranges:

    0x00000100 - 0x00000140 (64 bytes)
```

maximum_address
The maximum address of the data plus one, or `None` if the file is empty.

minimum_address
The minimum address of the data, or `None` if the file is empty.

segments
The segments object. Can be used to iterate over all segments in the binary.

Below is an example iterating over all segments, two in this case, and printing them.

```
>>> for segment in binfile.segments:
...     print(segment)
...
Segment(address=0, data=bytearray(b'\x00\x01\x02'))
Segment(address=10, data=bytearray(b'\x03\x04\x05'))
```

All segments can be split into smaller pieces using the *chunks(size=32, alignment=1)* method.

```
>>> for chunk in binfile.segments.chunks(2):
...     print(chunk)
...
Chunk(address=0, data=bytearray(b'\x00\x01'))
Chunk(address=2, data=bytearray(b'\x02'))
Chunk(address=10, data=bytearray(b'\x03\x04'))
Chunk(address=12, data=bytearray(b'\x05'))
```

Each segment can be split into smaller pieces using the *chunks(size=32, alignment=1)* method on a single segment.

```
>>> for segment in binfile.segments:
...     print(segment)
...     for chunk in segment.chunks(2):
...         print(chunk)
...
Segment(address=0, data=bytearray(b'\x00\x01\x02'))
Chunk(address=0, data=bytearray(b'\x00\x01'))
Chunk(address=2, data=bytearray(b'\x02'))
Segment(address=10, data=bytearray(b'\x03\x04\x05'))
Chunk(address=10, data=bytearray(b'\x03\x04'))
Chunk(address=12, data=bytearray(b'\x05'))
```

Index