hexformat Documentation

Release 0.3.0

Martin Scharrer

Oct 12, 2022

CONTENTS

The **hexformat** Python package allows the processing of several HEX data formats. Supported formats are the Intel-Hex format, the Motorola S-Record format as well as the simple hex dump format. The first two are often used for programming microcontrollers while the last is often used to display binary data in a user readable way.

Files in the mentioned format can be created, modified (e.g. set, delete and fill data) and read from or written to files. A base class *MultiPartBuffer* is provided which implements the handling of multiple data parts where every part is identified by its corresponding start address. This base class allows for the basic operations like reading and writing binary files as well as modifing the binary data. The specific classes *IntelHex*, *SRecord* and *HexDump* are derivated from it which implement the parsing and generating of the corresponding HEX formats as well as implementing file format specific features.

CHAPTER

ONE

MOTIVATION

This package was mainly created to replace the srecord command line tool and its complicated interface with a clean python interface for use with handling microcontroller HEX files. The existing Python library intelhex was used for a while, but then the need for the S-Record format appeared. Also *intelhex.IntelHex* did not provide a suitable interface for the most often used operations.

CHAPTER

TWO

LICENSE

This work is free software under the GPL v3 or any later version:

> Copyright (C) 2015-202 Martin Scharrer <martin.scharrer@web.de>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Contents:

2.1 About

The **hexformat** Python package allows the processing of several HEX data formats. Supported formats are the Intel-Hex format, the Motorola S-Record format as well as the simple hex dump format. The first two are often used for programming microcontrollers while the last is often used to display binary data in a user readable way.

Files in the mentioned format can be created, modified (e.g. set, delete and fill data) and read from or written to files. A base class `MultiPartBuffer` is provided which implements the handling of multiple data parts where every part is identified by its corresponding start address. This base class allows for the basic operations like reading and writing binary files as well as modifing the binary data. The specific classes `IntelHex`, `SRecord` and `HexDump` are derivated from it which implement the parsing and generating of the corresponding HEX formats as well as implementing file format specific features.

2.1.1 Motivation

This package was mainly created to replace the srecord command line tool and its complicated interface with a clean python interface for use with handling microcontroller HEX files. The existing Python library intelhex was used for a while, but then the need for the S-Record format appeared. Also *intelhex.IntelHex* did not provide a suitable interface for the most often used operations.

2.1.2 License

This work is free software under the GPL v3 or any later version:

> Copyright (C) 2015-202 Martin Scharrer <martin.scharrer@web.de>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

2.2 hexformat package

2.2.1 Submodules

2.2.2 hexformat.base module

Provide base class for hexformat classes.

License:

```
Copyright (C) 2015-2022  Martin Scharrer <martin.scharrer@web.de>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

exception hexformat.base.**DecodeError**

Bases: *HexformatError*

Exception is raised if errors during the decoding of a hex file occur.

exception hexformat.base.**EncodeError**

Bases: *HexformatError*

Exception is raised if errors during the encoding of a hex file occur.

class hexformat.base.**HexFormat**

Bases: *MultiPartBuffer*

classmethod fromother(*other*, *shallow_copy=False*)

settings(***settings*)

exception hexformat.base.**HexformatError**

Bases: Exception

General hexformat exception. Base class for all other exceptions of this module.

2.2.3 hexformat.fillpattern module

Provide auto-scaling fill patterns including a random pattern.

License:

```
Copyright (C) 2015-2022  Martin Scharrer <martin.scharrer@web.de>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

class hexformat.fillpattern.**FillPattern**(*pattern=255*, *length=None*)

Bases: object

General fill pattern class which instances contain a underlying pattern with is automatically repeated if required.

The underlying pattern is a bytearray which is repeated on demand to fit into a given official length or a slice of any length. An internal start offset for the underlying pattern is used when an instance is produced as slice with a non-zero start offset.

Parameters

- **pattern** (*byte or iterable of bytes*) – The basic fill pattern which will be repeated. Either a byte sized integer (0..255) or an iterable of such integers (usually a bytearray, str or suitable list or tuple).
- **length** (*None or int, optional*) – Official length of FillPattern. Only used if used with len() or str() etc. If None then the length of the pattern is used instead.

Raises

ValueError – if pattern argument is numeric but outside of the byte range of 0..255.

classmethod fromnumber(*number*, *width=4*, *byteorder='big'*, *length=None*, *signed=False*)

Generate instance from integer number. (Python 3 only)

Parameters

- **number** (*int*) – a numerical value which will be used for the pattern.

- **width** (*int*) – byte width of the number. Usually 1 till 4. If number is narrower than this width it is zero padded.
- **byteorder** (*'big'|'little'*) – If 'big' (default) the number will be turned into a list of bytes from most to least significant byte ("MSB first", "Motorola" style). Otherwise the byte order will be from least to most significant byte ("LSB first", "Intel" style). For any other byte order the method *frompattern()* must be used with a byte list.
- **length** (*None or int, optional*) – Official length of FillPattern. Only used if used with len() or str() etc. If None then the length of the pattern is used instead.
- **signed** (*bool; optional*) – determines if number is represented in two's complement. If False and number is negative an OverflowError is raised.

Returns
New instance of the same class.

classmethod frompattern(*pattern*, *length=None*)

Returns instance by either returning existing one or generate new instance from pattern list. The intended use of this method is as filter of user input, so that an instance or pattern can be passed.

Parameters

- **pattern** (*cls, byte or iterable of bytes*) – If pattern is already an instance of the same class it is used, either directly or as a length adjusted copy if the length argument differs from its length. Otherwise it must be a byte or iterable of bytes which is simply passed to the class constructor. If None then the length of the pattern is used instead.
- **length** (*int*) – Official length of pattern. If None the length of the pattern is used. If smaller than the pattern length, only the first *length* bytes are used from the pattern.

Returns
Instance of class based on the given pattern and length.

setlength(*length*)

Set official length.

Parameters
length (*int*) – Official length of FillPattern. Only used if used with len() or str() etc.

class hexformat.fillpattern.**RandomContent**(*length=1*)

Bases: *FillPattern*

Specific FillPattern subclass to produce random content.

Return random content instead any given pattern. Every call produces a different random content. For this the Python `random.randint()` method is used.

Parameters
length (*int*) – Official length. Only used if used with len() etc.

Raises
AttributeError – May be raised by len(pattern) if input is not as requested above.

hexformat.fillpattern.**int_to_bytes**(*self*, */*, *length*, *byteorder*, ***, *signed=False*)

Return an array of bytes representing an integer.

length
Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes.

byteorder
The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder' as the byte order value.

signed
Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

2.2.4 hexformat.hexdump module

Provide class for HexDump content.

License:

class hexformat.hexdump.**HexDump**

Bases: *HexFormat*

Hex dump representation class.

The HexDump class is able to generate and parse hex dumps of binary data.

classmethod fromhexdumpfh(*fh*, *bigendian=True*)

Generates HexDump instance from file handle which must point to hex dump lines.

Creates new instance and calls *loadhexdumpfh()* on it.

Parameters

- **fh** (*file handle or compatible*) – Source of Intel-Hex lines.
- **bigendian** (*bool*) – If True the bytes in a group will be interpreted in big endian (Motorola style, MSB first) order, otherwise in little endian (Intel style, LSB first) order.

Returns
New instance of class with loaded data.

classmethod fromhexdumpfile(*filename*, *bigendian=True*)

Generates HexDump instance from hex dump file.

Opens filename for reading and calls *fromhexdumpfh()* with the file handle.

Parameters

- **filename** (*str*) – Name of file to be loaded.

- **bigendian** (*bool*) – If True the bytes in a group will be interpreted in big endian (Motorola style, MSB first) order, otherwise in little endian (Intel style, LSB first) order.

Returns
New instance of class with loaded data.

loadhexdumpfh(*fh*, *bigendian=True*)

Loads hex dump lines from file handle.

Parameters

- **fh** (*file handle or compatible*) – Source of Intel-Hex lines.
- **bigendian** (*bool*) – If True the bytes in a group will be interpreted in big endian (Motorola style, MSB first) order, otherwise in little endian (Intel style, LSB first) order.

Returns
self

loadhexdumpfile(*filename*, *bigendian=True*)

Loads hex dump lines from named file.

Parameters

- **filename** (*str*) – Name of file to be loaded.
- **bigendian** (*bool*) – If True the bytes in a group will be interpreted in big endian (Motorola style, MSB first) order, otherwise in little endian (Intel style, LSB first) order.

Returns
self

tohexdumpfh(*fh*, *bytesperline=16*, *groupsize=1*, *bigendian=True*, *ascii=True*)

Writes hex dump to file handle.

Parameters

- **fh** (*file handle or compatible*) – File handle to be written to.
- **bytesperline** (*int*) – Number of data bytes per line.
- **groupsize** (*int*) – Number of data bytes to be grouped together.
- **bigendian** (*bool*) – If True the bytes in a group are written in big endian (Motorola style, MSB first) order, otherwise in little endian (Intel style, LSB first) order.
- **ascii** (*bool*) – If True the ASCII representation is written after the hex values.

Returns
self

tohexdumpfile(*filename*, *bytesperline=16*, *groupsize=1*, *bigendian=True*, *ascii=True*)

Writes hex dump to named file.

Opens filename for writing and calls *tohexdumpfh()* on it.

Parameters

- **filename** (*str*) – Name of file to be written.
- **bytesperline** (*int*) – Number of data bytes per line.
- **groupsize** (*int*) – Number of data bytes to be grouped together.
- **bigendian** (*bool*) – If True the bytes in a group are written in big endian (Motorola style, MSB first) order, otherwise in little endian (Intel style, LSB first) order.

- **ascii** (*bool*) – If True the ASCII representation is written after the hex values.

Returns
self

2.2.5 hexformat.intelhex module

Provide class to handle IntelHex content.

License:

```
Copyright (C) 2015-2022  Martin Scharrer <martin.scharrer@web.de>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

hexformat.intelhex.**RT_DATA**

Intel-Hex record type number for data record.

Type
int constant=0

hexformat.intelhex.**RT_END_OF_FILE**

Intel-Hex record type number for end of file record.

Type
int constant=1

hexformat.intelhex.**RT_EXTENDED_SEGMENT_ADDRESS**

Intel-Hex record type number for extend segment address record.

Type
int constant=2

hexformat.intelhex.**RT_START_SEGMENT_ADDRESS**

Intel-Hex record type number for start segment address record.

Type
int constant=3

hexformat.intelhex.**RT_EXTENDED_LINEAR_ADDRESS**

Intel-Hex record type number for extended linear address record.

Type
int constant=4

hexformat.intelhex.**RT_START_LINEAR_ADDRESS**

Intel-Hex record type number for start linear address record.

Type
int constant=5

class hexformat.intelhex.**IntelHex**(***settings*)

Bases: *HexFormat*

Intel-Hex file representation class.

The IntelHex class is able to parse and generate binary data in the Intel-Hex representation.

Parameters

- **bytesperline** (*int*) – Number of bytes per line.
- **variant** – Variant of Intel-Hex format. Must be one out of ('I08HEX', 'I8HEX', 'I16HEX', 'I32HEX', 8, 16, 32).
- **cs_ip** (*int, 32-bit*) – Value of CS:IP starting address used for I16HEX variant.
- **eip** (*int, 32-bit*) – Value of EIP starting address used for I32HEX variant.

_DATALENGTH

Data length according to record type.

Type
tuple

_VARIANTS

Mapping dict from variant name to number.

Type
dict

_DEFAULT_BYTES_PER_LINE

Default number of bytes per line.

Type
int

_DEFAULT_VARIANT

Default variant.

_bytesperline

Number of bytes per line of this instance. If None the default value will be used.

Type
None or int

_cs_ip

CS:IP address for I16HEX variant. If None no CS:IP address will be written.

Type
None or int

_eip

EIP address for I32HEX variant. If None no CS:IP address will be written.

Type
None or int

_variant

Numeric file format variant. If None the default variant is used.

Type
None or 8, 16 or 32

property bytesperline

property cs_ip

property eip

classmethod fromihexfh(*fh*, *ignore_checksum_errors=False*)

Generates IntelHex instance from file handle which must point to Intel-Hex lines.

Creates new instance and calls *loadihexfh()* on it.

Parameters

- **fh** (*file handle or compatible*) – Source of Intel-Hex lines.
- **ignore_checksum_errors** (*bool*) – If True no error is raised on checksum failures.

Returns
New instance of class with loaded data.

classmethod fromihexfile(*filename*, *ignore_checksum_errors=False*)

Generates IntelHex instance from Intel-Hex file.

Opens filename for reading and calls *fromihexfh()* with the file handle.

Parameters

- **filename** (*str*) – input filename
- **ignore_checksum_errors** (*bool*) – If True no error is raised on checksum failures

Returns
New instance of class with loaded data.

loadihexfh(*fh*, *ignore_checksum_errors=False*)

Loads Intel-Hex lines from file handle.

Parameters

- **fh** (*file handle or compatible*) – Source of Intel-Hex lines.
- **ignore_checksum_errors** (*bool*) – If True no error is raised on checksum failures.

Returns
self

Raises

- ***DecodeError*** – on checksum mismatch if ignore_checksum_errors is False.
- ***DecodeError*** – on unsupported record type.

loadihexfile(*filename*, *ignore_checksum_errors=False*)

Loads Intel-Hex lines from named file.

Creates new instance and calls *loadihexfh()* on it.

Parameters

- **filename** (*str*) – Name of Intel-Hex file.

- **ignore_checksum_errors** (*bool*) – If True no error is raised on checksum failures.

Returns
self

toihexfh(*fh*, ***settings*)

Writes content as Intel-Hex file to given file handle.

Parameters

- **fh** (*file handle or compatible*) – Destination of S-Record lines.
- **bytesperline** (*int*) – Number of bytes per line.
- **variant** (*'I08HEX', 'I8HEX', 'I16HEX', 'I32HEX', 8, 16, 32*) – Variant of Intel-Hex format.
- **cs_ip** (*int, 32-bit*) – Value of CS:IP starting address used for I16HEX variant.
- **eip** (*int, 32-bit*) – Value of EIP starting address used for I32HEX variant.

Returns
self

Raises
EncodeError – if selected address length is not wide enough to fit all addresses.

toihexfile(*filename*, ***settings*)

Writes content as Intel-Hex file to given file name.

Opens filename for writing and calls *toihexfh()* with the file handle and all arguments. See *toihexfh()* for description of the arguments.

Parameters

- **filename** (*str*) – Input file name.
- **bytesperline** (*int*) – Number of bytes per line.
- **variant** (*'I08HEX', 'I8HEX', 'I16HEX', 'I32HEX', 8, 16, 32*) – Variant of Intel-Hex format.
- **cs_ip** (*int, 32-bit*) – Value of CS:IP starting address used for I16HEX variant.
- **eip** (*int, 32-bit*) – Value of EIP starting address used for I32HEX variant.

Returns
self

property variant

2.2.6 hexformat.multipartbuffer module

Provide class to handle a data buffer with multiple disconnected parts.

License:

```
Copyright (C) 2015-2022  Martin Scharrer <martin.scharrer@web.de>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
```

(continues on next page)

(continued from previous page)

```
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

class hexformat.multipartbuffer.**MultiPartBuffer**

Bases: object

Class to handle disconnected binary data.

Each segment (simply called "part") is identified by its starting address and its content (a Buffer instance).

_STANDARD_FORMAT

The standard format used by *fromfh()* and *fromfile()* if no format was given.

Type
str

_padding

Standard fill pattern.

Type
int, iterable or *FillPattern*

add(*other*, *overwrite=True*)

Add content of other instance to itself, overwriting or keeping existing data if parts overlap.

Parameters

- **other** (MultiPartBuffer, *dict or iterable*) – Second summand. If a dict then the keys must be address and the values a buffer. If an iterable it must yield (address, data) combinations.
- **overwrite** (*bool*) – If True existing data will be overwritten if parts overlap.

copy()

Return a deep copy of the instance.

crop(*address*, *size=None*)

Crop content to range <address>+<size> by deleting all other content.

delete(*address*, *size=None*)

Deletes <size> bytes starting from <address>. Does nothing if <size> is non-positive.

end()

Get end address, i.e. the address one after the very last byte of data. An empty buffer will return 0.

extract(*address*, *size=None*, *keep=True*)

Extract given range and return it as new instance. Gaps in the range are preserved. The <keep> argument controls if the range is kept in the original instance or deleted.

fill(*address=None*, *size=None*, *fillpattern=None*, *overwrite=False*)

Fill with <fillpattern> from <address> for <size> bytes. Filling pattern can be a single byte (0..255) or list-like object. If <overwrite> is False (default) then only gaps are filled. If <address> is None the first existing address is used. If <size> is None the remaining size is used.

fillend(*endaddress*, *fillpattern=None*)

Fill the data range after the buffer up to the given address.

fillfront(*startaddress=0*, *fillpattern=None*)

Fill the data range in front starting from the given address (0 by default) to the beginning of the buffer.

fillgaps(*fillpattern=None*)

Fill all gaps with given fillpattern.

filter(*filterfunc*, *address=None*, *size=None*, *fillpattern=None*)

Call filterfunc(bufferaddr, buffer, bufferstartindex, buffersize) on all parts matching <address> and <size>. If <address> is None the first existing address is used. If <size> is None the remaining size is used. If fillpattern is NOT None the given range is filled and the filter function will be called with the resulting single part.

classmethod frombinfh(*fh*, *address=0*, *size=-1*, *offset=0*)

classmethod frombinfile(*filename*, *address=0*, *size=-1*, *offset=0*)

classmethod fromfh(*fh*, *fformat=None*, **args*, ***kvargs*)

classmethod fromfile(*filename*, *fformat=None*, **args*, ***kvargs*)

gaps()

Return a list with (address,length) tuples for all gaps between the existing parts.

get(*address*, *size*, *fillpattern=None*)

Get <size> bytes from <address>. Fill missing bytes with <fillpattern>.

Where <fillpattern> can be:

1. A byte-sized integer, i.e. in range 0..255.
2. **A list-like object which has all of the following properties:**
   - can be converted to a list of bytes
   - is indexable
   - should be multipliable
3. A type which produces a list-like object as mentioned in 2) if instanciated with no argument or with the requested size as only argument.
4. An exception class or instance. If given then no filling is performed but the exception is raised if filling would be required.

includesgaps(*address=None*, *size=None*)

loadbinfh(*fh*, *address=0*, *size=-1*, *offset=0*)

loadbinfile(*filename*, *address=0*, *size=-1*, *offset=0*)

loaddict(*d*, *overwrite=True*)

Load data from dictionary where each key must be numeric and represent an address and the corresponding value the byte value.

loadfh(*fh*, *fformat=None*, **args*, ***kvargs*)

loadfile(*filename*, *fformat=None*, **args*, ***kvargs*)

offset(*offset*)

Add an offset to all addresses. A ValueError is raised if offset < -start, as this would lead to negative addresses. If offset is None, the negative start address is used, i.e. the first byte is moved to address 0.

parts()

Return a list with (address,length) tuples for all parts.

range()

Get range of content as (start address, size) tuple. The range may contain unfilled gaps. An empty buffer with return (0, 0).

relocate(*newaddress*, *address=None*, *size=None*, *overwrite=True*)

Relocate given range to new address. If address is None the start address is used. If size is None the remaining size from address to the endaddress is used. The <overwrite> argument determines if existing data in the new range is overwritten or if the overlapping bytes of the relocated data are discarded.

set(*address*, *newdata*, *datasize=None*, *dataoffset=0*, *overwrite=True*)

Add <newdata> starting at <address>. The data size can be given explicitly, otherwise it is taken as len(newdata). Additionally a data offset can be specified to read the data starting from this index from <newdata>.

setint(*address*, *intvalue*, *datasize*, *byteorder='big'*, *signed=False*, *overwrite=True*)

Set integer value at given address.

start()

Get start address. An empty buffer with return 0.

tobinfh(*fh*, *address=None*, *size=None*, *fillpattern=None*)

tobinfile(*filename*, *address=None*, *size=None*, *fillpattern=None*)

todict()

Return a dictionary with a numeric key for all used bytes like intelhex.IntelHex does it.

tofh(*fh*, *fformat=None*, **args*, ***kvargs*)

tofile(*filename*, *fformat=None*, **args*, ***kvargs*)

unfill(*address=None*, *size=None*, *unfillpattern=None*, *mingapsize=16*)

Removes <unfillpattern> and leaves a gap, as long a resulting new gap would be least <mingapsize> large.

usedsize()

Returns used data size, i.e. without the size of any gaps

hexformat.multipartbuffer.**ensurebuffer**(*buforint*)

hexformat.multipartbuffer.**int_to_bytes**(*self*, */*, *length*, *byteorder*, ***, *signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An OverflowError is raised if the integer is not representable with the given number of bytes.

byteorder
The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder' as the byte order value.

signed
Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

2.2.7 hexformat.srecord module

Provide class to handle Motorola SRecord content.

License:

```
Copyright (C) 2015-2022  Martin Scharrer <martin.scharrer@web.de>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

class hexformat.srecord.**RECORD_TYPE**

Bases: object

COUNT_16 = 5

COUNT_24 = 6

DATA_16 = 1

DATA_24 = 2

DATA_32 = 3

FOOTER_16 = 9

FOOTER_24 = 8

FOOTER_32 = 7

HEADER = 0

S0 = 0

S1 = 1

S2 = 2

S3 = 3

S5 = 5

S6 = 6

S7 = 7

S8 = 8

S9 = 9

class hexformat.srecord.**SRecord**(***settings*)

Bases: *HexFormat*

Motorola S-Record hex file representation class.

The SRecord class is able to parse and generate binary data in the S-Record representation.

_SRECORD_ADDRESSLENGTH

Address length in bytes for each record type.

Type
tuple

_STANDARD_FORMAT

The standard format used by *fromfh()* and *fromfile()* if no format was given.

Type
str

_startaddress

Starting execution location. This tells the programmer which address contains the start routine. Default: 0.

Type
int

_header

Header data written using record type 0 if not None. The content is application specific.

Type
data buffer or None

property addresslength

property bytesperline

classmethod fromsrecfh(*fh*, *raise_error_on_miscount=True*)

Generates SRecord instance from file handle which must point to S-Record lines.

Creates new instance and calls *loadsrecfh()* on it.

Parameters

- **fh** (*file handle or compatible*) – Source of S-Record lines.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns
New instance of class with loaded data.

classmethod fromsrecfile(*filename*, *raise_error_on_miscount=True*)

Generates SRecord instance from S-Record file.

Opens filename for reading and calls *fromsrecfh()* with the file handle.

Parameters

- **filename** (*str*) – Name of S-Record file.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns

New instance of class with loaded data.

property header

loadsrecfh(*fh*, *overwrite_metadata=False*, *overwrite_data=True*, *raise_error_on_miscount=True*)

Loads data from S-Record file over file handle.

Parses every source line using `_parsesrecline()` and processes the decoded elements according to the record type.

Parameters

- **fh** (*file handle or compatible*) – Source of S-Record lines.
- **overwrite_metadata** (*bool*) – If True existing metadata will be overwritten.
- **overwrite_data** (*bool*) – If True existing data will be overwritten.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns

self

Raises

- ***DecodeError*** – If decoded record type is outside of range 0..9.
- ***DecodeError*** – If raise_error_on_miscount is True and number of records read differ from stored number of records.

loadsrecfile(*filename*, *overwrite_metadata=False*, *overwrite_data=True*, *raise_error_on_miscount=True*)

Loads S-Record lines from named file.

Creates new instance and calls *loadsrecfh()* on it.

Parameters

- **filename** (*str*) – Name of S-Record file.
- **overwrite_metadata** (*bool*) – If True existing metadata will be overwritten.
- **overwrite_data** (*bool*) – If True existing data will be overwritten.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns

self

property startaddress

tosrecfh(*fh*, ***settings*)

Writes content as S-Record file to given file handle.

Parameters

- **fh** (*file handle or compatible*) – Destination of S-Record lines.
- **bytesperline** (*int*) – Number of data bytes per line.
- **addresslength** (*None or int in range 2..4*) – Address length in bytes. This determines the used file format variant. If None then the shortest possible address length large enough to encode the highest address present is used.
- **write_number_of_records** (*bool*) – If True then the number of data records is written as a record type 5 or 6. This adds an additional verification method if the S-Record file is consistent.

Returns
self

tosrecfile(*filename*, ***settings*)

Writes content as S-Record file to given file name.

Opens filename for writing and calls *tosrecfh()* with the file handle and all arguments. See `tosecfh()` for description of the arguments.

Returns
self

property write_number_of_records

2.2.8 hexformat.tektronix module

Provide class to handle Tektronix Extended Hex content.

License:

```
Copyright (C) 2015-2022  Martin Scharrer <martin.scharrer@web.de>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

class hexformat.tektronix.**TektronixExtHex**(***settings*)

Bases: *HexFormat*

Tektronix Extended Hex file representation class.

The TektronixExtHex class is able to parse and generate binary data in the Tektronix Extended Hex representation.

Attributes:

property addresslength

property bytesperline

classmethod fromtekfh(*fh*)

Generates instance from file handle which must point to Tektronix Extended Hex lines.

Creates new instance and calls *loadtekfh()* on it.

Parameters
fh (*file handle or compatible*) – Source of Tektronix Extended Hex lines.

Returns
New instance of class with loaded data.

classmethod fromtekfile(*filename*)

Generates instance from Tektronix Extended Hex file.

Opens filename for reading and calls *fromtekfh()* with the file handle.

Parameters
filename (*str*) – Name of Tektronix Extended Hex file.

Returns
New instance of class with loaded data.

loadtekfh(*fh*, *overwrite_metadata=False*, *overwrite_data=True*, *raise_error_on_miscount=True*)

Loads data from Tektronix Extended Hex file over file handle.

Parses every source line using `_parsetekline()` and processes the decoded elements according to the record type.

Parameters

- **fh** (*file handle or compatible*) – Source of Tektronix Extended Hex lines.
- **overwrite_metadata** (*bool*) – If True existing metadata will be overwritten.
- **overwrite_data** (*bool*) – If True existing data will be overwritten.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns
self

Raises

- ***DecodeError*** – If decoded record type is outside of range 0..9.
- ***DecodeError*** – If raise_error_on_miscount is True and number of records read differ from stored number of records.

loadtekfile(*filename*, *overwrite_metadata=False*, *overwrite_data=True*, *raise_error_on_miscount=True*)

Loads Tektronix Extended Hex lines from named file.

Creates new instance and calls *loadtekfh()* on it.

Parameters

- **filename** (*str*) – Name of Tektronix Extended Hex file.
- **overwrite_metadata** (*bool*) – If True existing metadata will be overwritten.

- **overwrite_data** (*bool*) – If True existing data will be overwritten.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns
self

property startaddress

totekfh(*fh*, ***settings*)

Writes content as Tektronix Extended Hex file to given file handle.

Parameters

- **fh** (*file handle or compatible*) – Destination of Tektronix Extended Hex lines.
- **settings** –

Returns
self

totekfile(*filename*, ***settings*)

Writes content as Tektronix Extended Hex file to given file name.

Opens filename for writing and calls *totekfh()* with the file handle and all arguments. See *totekfh()* for description of the arguments.

Returns
self

2.2.9 Module contents

Provide classes for popular hex formats as well as auxiliary classes to generate fill patterns.

License:

```
Copyright (C) 2015-2022  Martin Scharrer <martin.scharrer@web.de>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

exception hexformat.**DecodeError**

Bases: *HexformatError*

Exception is raised if errors during the decoding of a hex file occur.

exception hexformat.**EncodeError**

Bases: *HexformatError*

Exception is raised if errors during the encoding of a hex file occur.

class hexformat.**FillPattern**(*pattern=255*, *length=None*)

Bases: object

General fill pattern class which instances contain a underlying pattern with is automatically repeated if required.

The underlying pattern is a bytearray which is repeated on demand to fit into a given official length or a slice of any length. An internal start offset for the underlying pattern is used when an instance is produced as slice with a non-zero start offset.

Parameters

- **pattern** (*byte or iterable of bytes*) – The basic fill pattern which will be repeated. Either a byte sized integer (0..255) or an iterable of such integers (usually a bytearray, str or suitable list or tuple).
- **length** (*None or int, optional*) – Official length of FillPattern. Only used if used with len() or str() etc. If None then the length of the pattern is used instead.

Raises

ValueError – if pattern argument is numeric but outside of the byte range of 0..255.

classmethod **fromnumber**(*number*, *width=4*, *byteorder='big'*, *length=None*, *signed=False*)

Generate instance from integer number. (Python 3 only)

Parameters

- **number** (*int*) – a numerical value which will be used for the pattern.
- **width** (*int*) – byte width of the number. Usually 1 till 4. If number is narrower than this width it is zero padded.
- **byteorder** (*'big'|'little'*) – If 'big' (default) the number will be turned into a list of bytes from most to least significant byte ("MSB first", "Motorola" style). Otherwise the byte order will be from least to most significant byte ("LSB first", "Intel" style). For any other byte order the method *frompattern()* must be used with a byte list.
- **length** (*None or int, optional*) – Official length of FillPattern. Only used if used with len() or str() etc. If None then the length of the pattern is used instead.
- **signed** (*bool; optional*) – determines if number is represented in two's complement. If False and number is negative an OverflowError is raised.

Returns

New instance of the same class.

classmethod **frompattern**(*pattern*, *length=None*)

Returns instance by either returning existing one or generate new instance from pattern list. The intended use of this method is as filter of user input, so that an instance or pattern can be passed.

Parameters

- **pattern** (*cls, byte or iterable of bytes*) – If pattern is already an instance of the same class it is used, either directly or as a length adjusted copy if the length argument differs from its length. Otherwise it must be a byte or iterable of bytes which is simply passed to the class constructor. If None then the length of the pattern is used instead.
- **length** (*int*) – Official length of pattern. If None the length of the pattern is used. If smaller than the pattern length, only the first *length* bytes are used from the pattern.

Returns
Instance of class based on the given pattern and length.

`setlength`(*length*)

Set official length.

Parameters
length (*int*) – Official length of FillPattern. Only used if used with len() or str() etc.

`class hexformat.IntelHex`(***settings*)

Bases: *HexFormat*

Intel-Hex file representation class.

The IntelHex class is able to parse and generate binary data in the Intel-Hex representation.

Parameters

- **bytesperline** (*int*) – Number of bytes per line.
- **variant** – Variant of Intel-Hex format. Must be one out of ('I08HEX', 'I8HEX', 'I16HEX', 'I32HEX', 8, 16, 32).
- **cs_ip** (*int, 32-bit*) – Value of CS:IP starting address used for I16HEX variant.
- **eip** (*int, 32-bit*) – Value of EIP starting address used for I32HEX variant.

`_DATALENGTH`

Data length according to record type.

Type
tuple

`_VARIANTS`

Mapping dict from variant name to number.

Type
dict

`_DEFAULT_BYTES_PER_LINE`

Default number of bytes per line.

Type
int

`_DEFAULT_VARIANT`

Default variant.

`_bytesperline`

Number of bytes per line of this instance. If None the default value will be used.

Type
None or int

`_cs_ip`

CS:IP address for I16HEX variant. If None no CS:IP address will be written.

Type
None or int

_eip

EIP address for I32HEX variant. If None no CS:IP address will be written.

Type
None or int

_variant

Numeric file format variant. If None the default variant is used.

Type
None or 8, 16 or 32

property bytesperline

property cs_ip

property eip

classmethod fromihexfh(*fh*, *ignore_checksum_errors=False*)

Generates IntelHex instance from file handle which must point to Intel-Hex lines.

Creates new instance and calls *loadihexfh()* on it.

Parameters

- **fh** (*file handle or compatible*) – Source of Intel-Hex lines.
- **ignore_checksum_errors** (*bool*) – If True no error is raised on checksum failures.

Returns
New instance of class with loaded data.

classmethod fromihexfile(*filename*, *ignore_checksum_errors=False*)

Generates IntelHex instance from Intel-Hex file.

Opens filename for reading and calls *fromihexfh()* with the file handle.

Parameters

- **filename** (*str*) – input filename
- **ignore_checksum_errors** (*bool*) – If True no error is raised on checksum failures

Returns
New instance of class with loaded data.

loadihexfh(*fh*, *ignore_checksum_errors=False*)

Loads Intel-Hex lines from file handle.

Parameters

- **fh** (*file handle or compatible*) – Source of Intel-Hex lines.
- **ignore_checksum_errors** (*bool*) – If True no error is raised on checksum failures.

Returns
self

Raises

- *DecodeError* – on checksum mismatch if ignore_checksum_errors is False.
- *DecodeError* – on unsupported record type.

loadihexfile(*filename*, *ignore_checksum_errors=False*)

Loads Intel-Hex lines from named file.

Creates new instance and calls *loadihexfh()* on it.

Parameters

- **filename** (*str*) – Name of Intel-Hex file.
- **ignore_checksum_errors** (*bool*) – If True no error is raised on checksum failures.

Returns
self

toihexfh(*fh*, ***settings*)

Writes content as Intel-Hex file to given file handle.

Parameters

- **fh** (*file handle or compatible*) – Destination of S-Record lines.
- **bytesperline** (*int*) – Number of bytes per line.
- **variant** (*'I08HEX', 'I8HEX', 'I16HEX', 'I32HEX', 8, 16, 32*) – Variant of Intel-Hex format.
- **cs_ip** (*int, 32-bit*) – Value of CS:IP starting address used for I16HEX variant.
- **eip** (*int, 32-bit*) – Value of EIP starting address used for I32HEX variant.

Returns
self

Raises
EncodeError – if selected address length is not wide enough to fit all addresses.

toihexfile(*filename*, ***settings*)

Writes content as Intel-Hex file to given file name.

Opens filename for writing and calls *toihexfh()* with the file handle and all arguments. See *toihexfh()* for description of the arguments.

Parameters

- **filename** (*str*) – Input file name.
- **bytesperline** (*int*) – Number of bytes per line.
- **variant** (*'I08HEX', 'I8HEX', 'I16HEX', 'I32HEX', 8, 16, 32*) – Variant of Intel-Hex format.
- **cs_ip** (*int, 32-bit*) – Value of CS:IP starting address used for I16HEX variant.
- **eip** (*int, 32-bit*) – Value of EIP starting address used for I32HEX variant.

Returns
self

property variant

class hexformat.**RandomContent**(*length=1*)

Bases: *FillPattern*

Specific FillPattern subclass to produce random content.

Return random content instead any given pattern. Every call produces a different random content. For this the Python `random.randint()` method is used.

Parameters
length (*int*) – Official length. Only used if used with len() etc.

Raises
AttributeError – May be raised by len(pattern) if input is not as requested above.

class hexformat.**SRecord**(***settings*)

Bases: *HexFormat*

Motorola S-Record hex file representation class.

The SRecord class is able to parse and generate binary data in the S-Record representation.

_SRECORD_ADDRESSLENGTH

Address length in bytes for each record type.

Type
tuple

_STANDARD_FORMAT

The standard format used by *fromfh()* and *fromfile()* if no format was given.

Type
str

_startaddress

Starting execution location. This tells the programmer which address contains the start routine. Default: 0.

Type
int

_header

Header data written using record type 0 if not None. The content is application specific.

Type
data buffer or None

property addresslength

property bytesperline

classmethod fromsrecfh(*fh*, *raise_error_on_miscount=True*)

Generates SRecord instance from file handle which must point to S-Record lines.

Creates new instance and calls *loadsrecfh()* on it.

Parameters
- **fh** (*file handle or compatible*) – Source of S-Record lines.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns
New instance of class with loaded data.

classmethod fromsrecfile(*filename*, *raise_error_on_miscount=True*)

Generates SRecord instance from S-Record file.

Opens filename for reading and calls *fromsrecfh()* with the file handle.

Parameters

- **filename** (*str*) – Name of S-Record file.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns
New instance of class with loaded data.

property header

loadsrecfh(*fh*, *overwrite_metadata=False*, *overwrite_data=True*, *raise_error_on_miscount=True*)

Loads data from S-Record file over file handle.

Parses every source line using `_parsesrecline()` and processes the decoded elements according to the record type.

Parameters

- **fh** (*file handle or compatible*) – Source of S-Record lines.
- **overwrite_metadata** (*bool*) – If True existing metadata will be overwritten.
- **overwrite_data** (*bool*) – If True existing data will be overwritten.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns
self

Raises

- ***DecodeError*** – If decoded record type is outside of range 0..9.
- ***DecodeError*** – If raise_error_on_miscount is True and number of records read differ from stored number of records.

loadsrecfile(*filename*, *overwrite_metadata=False*, *overwrite_data=True*, *raise_error_on_miscount=True*)

Loads S-Record lines from named file.

Creates new instance and calls *loadsrecfh()* on it.

Parameters

- **filename** (*str*) – Name of S-Record file.
- **overwrite_metadata** (*bool*) – If True existing metadata will be overwritten.
- **overwrite_data** (*bool*) – If True existing data will be overwritten.
- **raise_error_on_miscount** (*bool*) – If True a DecodeError is raised if the number of records read differs from stored number of records.

Returns
self

property startaddress

tosrecfh(*fh*, ***settings*)

Writes content as S-Record file to given file handle.

Parameters

- **fh** (*file handle or compatible*) – Destination of S-Record lines.

- **bytesperline** (*int*) – Number of data bytes per line.
- **addresslength** (*None or int in range 2..4*) – Address length in bytes. This determines the used file format variant. If None then the shortest possible address length large enough to encode the highest address present is used.
- **write_number_of_records** (*bool*) – If True then the number of data records is written as a record type 5 or 6. This adds an additional verification method if the S-Record file is consistent.

Returns
self

tosrecfile(*filename*, ***settings*)

Writes content as S-Record file to given file name.

Opens filename for writing and calls *tosrecfh()* with the file handle and all arguments. See `tosecfh()` for description of the arguments.

Returns
self

property write_number_of_records

2.3 How-to

2.3.1 Basic actions

The following methods are defined in the base class *MultiPartBuffer* and are available with all hexformat classes.

Load an existing file

Methods are provided to read the data from already opened file handles (or compatibles) or by providing the file name only.

A new instance can be directly created from using *fromfh()* and *fromfile()*. These methods will use the default format and call `from<formatname>fh()` and `from<formatname>file()` respectively. All classes support reading from binary files using *frombinfh()* and *frombinfile()*

Additional content can be read to an exising instance using *loadfh()* and *loadfile()*. These methods will use the default format and call `load<formatname>fh()` and `load<formatname>file()` respectively. All classes support reading from binary files using *loadbinfh()* and *loadbinfile()*

Create new instance from binary file

classmethod MultiPartBuffer.**frombinfh**(*fh*, *address=0*, *size=-1*, *offset=0*)

classmethod MultiPartBuffer.**frombinfile**(*filename*, *address=0*, *size=-1*, *offset=0*)

Example:

```
from hexformat.srecord import SRecord
srec = SRecord.frombinfile("somefile.bin")
with open("somefile.bin", "rb") as fh:
    srec2 = SRecord.frombinfh(fh)
```

Load (more) content from a binary file

MultiPartBuffer.**loadbinfh**(*fh*, *address=0*, *size=-1*, *offset=0*)

MultiPartBuffer.**loadbinfile**(*filename*, *address=0*, *size=-1*, *offset=0*)

Example:

```
from hexformat.srecord import SRecord
srec = SRecord()
srec.loadbinfile("somefile.bin")
with open("somefile.bin", "rb") as fh:
    srec = SRecord.loadbinfh(fh)
```

Create new instance from hex file

These methods are using the standard format of the class as long no other is specified using the *fformat* argument.

classmethod MultiPartBuffer.**fromfh**(*fh*, *fformat=None*, **args*, ***kvargs*)

classmethod MultiPartBuffer.**fromfile**(*filename*, *fformat=None*, **args*, ***kvargs*)

Example:

```
from hexformat.srecord import SRecord
srec = SRecord.fromfile("somefile.bin")
with open("somefile.bin", "rb") as fh:
    srec2 = SRecord.fromfh(fh)
```

Load (more) content from a hex file

These methods are using the standard format of the class as long no other is specified using the *fformat* argument.

MultiPartBuffer.**loadfh**(*fh*, *fformat=None*, **args*, ***kvargs*)

MultiPartBuffer.**loadfile**(*filename*, *fformat=None*, **args*, ***kvargs*)

Example:

```
from hexformat.srecord import SRecord
srec = SRecord()
srec.loadfile("somefile.bin")
with open("somefile.bin", "rb") as fh:
    srec = SRecord.loadfh(fh)
```

2.4 Changelog

2.4.1 v0.3.0 - 2022-10-11

- Changed unittests from nose to unittest.

2.4.2 v0.2 - 2016-04-02

- Ensured compatibility with Python 2.7 and Python 3 (tested with 3.3, 3.4 and 3.5).
- SRecord encoding fixes: Last byte was dropped if it would be on a line on its own.
- Several other code fixes.
- Enhanced unit tests.
- Added properties for hexformats.
- Renamed module *main* to *base*.
- Added most used imports to package init file.
- Code cleanup, including argument renaming
- Added FillPattern support for negative indicies.

2.4.3 v0.1 - 2015-09-20

- First version.

2.5 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

O

P

R

S

T

U

V

W