hexrec

Release 0.4.0

Andrea Zoppi

Mar 08, 2024

CONTENTS

1 Overview **1**
- 1.1 Introduction . . . 1
- 1.2 Documentation . . . 2
- 1.3 Architecture . . . 2
- 1.4 Examples . . . 2
  - 1.4.1 Convert format . . . 2
  - 1.4.2 Merge files . . . 3
  - 1.4.3 Dataset generator . . . 3
  - 1.4.4 Write a CRC . . . 4
  - 1.4.5 Trim for bootloader . . . 4
  - 1.4.6 Export ELF sections . . . 5
- 1.5 Installation . . . 5
- 1.6 Development . . . 6

2 Installation **7**
- 2.1 From *PyPI* . . . 7
- 2.2 From source . . . 7

3 Command Line Interface **9**
- 3.1 hexrec . . . 9
  - 3.1.1 align . . . 9
  - 3.1.2 clear . . . 10
  - 3.1.3 convert . . . 11
  - 3.1.4 crop . . . 12
  - 3.1.5 delete . . . 13
  - 3.1.6 fill . . . 13
  - 3.1.7 flood . . . 14
  - 3.1.8 hd . . . 15
  - 3.1.9 hexdump . . . 17
  - 3.1.10 merge . . . 18
  - 3.1.11 shift . . . 19
  - 3.1.12 srec . . . 20
  - 3.1.13 validate . . . 21
  - 3.1.14 xxd . . . 22

4 Reference **25**
- 4.1 base . . . 25
  - 4.1.1 FILE_TYPES . . . 25
  - 4.1.2 TOKEN_COLOR_CODES . . . 26
  - 4.1.3 colorize_tokens . . . 26

4.1.4 convert . . . 27
4.1.5 guess_format_name . . . 28
4.1.6 guess_format_type . . . 28
4.1.7 load . . . 29
4.1.8 merge . . . 30
4.1.9 BaseFile . . . 31
4.1.10 BaseRecord . . . 66
4.1.11 BaseTag . . . 78
4.2 formats . . . 79
4.3 asciihex . . . 80
4.3.1 AsciiHexFile . . . 80
4.3.2 AsciiHexRecord . . . 114
4.3.3 AsciiHexTag . . . 126
4.4 avr . . . 132
4.4.1 AvrFile . . . 132
4.4.2 AvrRecord . . . 166
4.4.3 AvrTag . . . 177
4.5 ihex . . . 178
4.5.1 IhexFile . . . 178
4.5.2 IhexRecord . . . 214
4.5.3 IhexTag . . . 227
4.6 mos . . . 234
4.6.1 MosFile . . . 234
4.6.2 MosRecord . . . 268
4.6.3 MosTag . . . 279
4.7 raw . . . 285
4.7.1 RawFile . . . 285
4.7.2 RawRecord . . . 319
4.7.3 RawTag . . . 330
4.8 sqtp . . . 331
4.8.1 from_numbers . . . 331
4.8.2 from_strings . . . 334
4.8.3 to_numbers . . . 335
4.8.4 to_strings . . . 338
4.9 srec . . . 339
4.9.1 SIZE_TO_ADDRESS_FORMAT . . . 339
4.9.2 SrecFile . . . 339
4.9.3 SrecRecord . . . 375
4.9.4 SrecTag . . . 388
4.10 titxt . . . 399
4.10.1 TiTxtFile . . . 399
4.10.2 TiTxtRecord . . . 433
4.10.3 TiTxtTag . . . 445
4.11 xtek . . . 451
4.11.1 XtekFile . . . 451
4.11.2 XtekRecord . . . 485
4.11.3 XtekTag . . . 500
4.12 hexdump . . . 505
4.12.1 CHAR_PRINTABLE . . . 506
4.12.2 CHAR_TOKENS . . . 506
4.12.3 DEFAULT_FORMAT_ORDER . . . 507
4.12.4 hexdump_core . . . 507
4.13 utils . . . 509
4.13.1 SUFFIX_SCALE . . . 509

4.13.2 DEFAULT_DELETE . . . 509
4.13.3 chop . . . 509
4.13.4 hexlify . . . 510
4.13.5 parse_int . . . 511
4.13.6 unhexlify . . . 511
4.13.7 SparseMemoryIO . . . 512
4.14 xxd . . . 534
4.14.1 CHAR_ASCII . . . 534
4.14.2 CHAR_EBCDIC . . . 534
4.14.3 parse_seek . . . 534
4.14.4 xxd_core . . . 535

5 Contributing **537**
5.1 Bug reports . . . 537
5.2 Documentation improvements . . . 537
5.3 Feature requests and feedback . . . 537
5.4 Development . . . 538
5.4.1 Pull Request Guidelines . . . 538
5.4.2 Tips . . . 538

6 Authors **539**
6.1 Special thanks . . . 539

7 Changelog **541**
7.1 0.4.0 (TBD) . . . 541
7.2 0.3.1 (2024-01-23) . . . 541
7.3 0.3.0 (2023-02-21) . . . 541
7.4 0.2.3 (2021-12-30) . . . 542
7.5 0.2.2 (2020-11-08) . . . 542
7.6 0.2.1 (2020-03-05) . . . 542
7.7 0.2.0 (2020-02-01) . . . 542
7.8 0.1.0 (2019-08-13) . . . 542
7.9 0.0.4 (2018-12-22) . . . 542
7.10 0.0.3 (2018-12-04) . . . 543
7.11 0.0.2 (2018-08-29) . . . 543
7.12 0.0.1 (2018-06-27) . . . 543

8 Indices and tables **545**

Python Module Index **547**

Index **549**

CHAPTER

ONE

OVERVIEW

docs
tests
package

Library to handle hexadecimal record files

- Free software: BSD 2-Clause License

1.1 Introduction

The purpose of this library is to provide simple but useful methods to load, edit, and save hexadecimal record files.

In the field of embedded systems, hexadecimal record files are the most common way to share binary data to be written to the target non-volatile memory, such as a EEPROM or microcontroller code flash. Such binary data can contain compiled executable code, configuration data, volatile memory dumps, etc.

The most common file formats for hexadecimal record files are *Intel HEX* (.hex) and *Motorola S-record* (.srec). Other common formats for binary data exchange for embedded systems include the *Executable and Linkable Format* (.elf), hex dumps (by *hexdump* or *xxd*), and raw binary files (.bin).

A good thing about hexadecimal record files is that they are almost *de-facto*, so every time a supplier has to give away its binary data it is either in HEX or SREC, although ELF is arguably the most common for debuggable executables.

A bad thing is that their support in embedded software toolsets is sometimes flawed or only one of the formats is supported, while the supplier provides its binary data in the other format.

Another feature is that binary data is split into text record lines (thus their name) protected by some kind of checksum. This is good for data exchange and line-by-line writing to the target memory (in the old days), but this makes in-place editing by humans rather tedious as data should be split, and the checksum and other metadata have to be updated.

All of the above led to the development of this library, which allows to, for example:

- convert between hexadecimal record formats;
- merge/patch multiple hexadecimal record files of different formats;

- access every single record of a hexadecimal record file;
- build records through handy methods;
- edit sparse data in a virtual memory behaving like a `bytearray`;
- extract or update only some parts of the binary data.

1.2 Documentation

For the full documentation, please refer to:

https://hexrec.readthedocs.io/

1.3 Architecture

Within the `hexrec` package itself are the symbols of the most commonly used classes and functions.

As the core of this library are record files, the `hexrec.base` is the first module a user should look up. It provides high-level functions to deal with record files, as well as classes holding record data.

The `hexrec.base` allows to load `bytesparse` virtual memories, which are as easy to use as the native `bytearray`, but with sparse data blocks.

The `hexrec.utils` module provides some miscellaneous utility stuff.

`hexrec.xxd` is an emulation of the `xxd` command line utility delivered by `vim`.

The package can also be run as a command line tool, by running the `hexrec` package itself (`python -m hexrec`), providing some record file utilities. You can also create your own standalone executable, or download a precompiled one from the `pyinstaller` folder.

The codebase is written in a simple fashion, to be easily readable and maintainable, following some naive pythonic *K.I.S.S.* approach by choice.

1.4 Examples

To have a glimpse of the features provided by this library, some simple but common examples are shown in the following.

1.4.1 Convert format

It happens that some software tool only supports some hexadecimal record file formats, or the format given to you is not handled properly, or simply you prefer a format against another (*e.g.* SREC has *linear* addressing, while HEX is in a *segment:offset* fashion).

In this example, a HEX file is converted to SREC.

```
from hexrec import convert

convert('data.hex', 'data.srec')
```

This can also be done by running *hexrec* as a command line tool:

```
$ hexrec convert data.hex data.srec
```

Alternatively, by executing the package itself:

```
$ python -m hexrec convert data.hex data.srec
```

1.4.2 Merge files

It is very common that the board factory prefers to receive a single file to program the microcontroller, because a single file is simpler to manage for them, and might be faster for their workers or machine, where every second counts.

This example shows how to merge a bootloader, an executable, and some configuration data into a single file, in the order they are listed.

```
from hexrec import merge

in_paths = ['bootloader.hex', 'executable.mot', 'configuration.xtek']
out_path = 'merged.srec'
merge(in_paths, out_path)
```

Alternatively, these files can be merged via manual load:

```
from hexrec import load, SrecFile

in_paths = ['bootloader.hex', 'executable.mot', 'configuration.xtek']
in_files = [load(path) for path in in_paths]
out_file = SrecFile().merge(*in_files)
out_file.save('merged.srec')
```

This can also be accomplished by running the *hexrec* package itself as a command line tool:

```
$ hexrec merge bootloader.hex executable.mot configuration.xtek merged.srec
```

1.4.3 Dataset generator

Let us suppose we are early in the development of the embedded system and we need to test the current executable with some data stored in EEPROM. We lack the software tool to generate such data, and even worse we need to test 100 configurations. For the sake of simplicity, the data structure consists of 4096 random values (0 to 1) of `float` type, stored in little-endian at the address `0xDA7A0000`.

```
import struct, random
from hexrec import SrecFile

for index in range(100):
    values = [random.random() for _ in range(4096)]
    data = struct.pack('<4096f', *values)
    file = SrecFile.from_bytes(data, offset=0xDA7A0000)
    file.save(f'dataset_{index:02d}.mot')
```

1.4.4 Write a CRC

Usually, the executable or the configuration data of an embedded system are protected by a CRC, so that their integrity can be self-checked.

Let us suppose that for some reason the compiler does not calculate such CRC the expected way, and we prefer to do it with a script.

This example shows how to load a HEX file, compute a CRC32 from the address `0x1000` to `0x3FFB` (`0x3FFC` exclusive), and write the calculated CRC to `0x3FFC` in big-endian as a SREC file. The rest of the data is left untouched.

```
import binascii, struct
from hexrec import load

file = load('checkme.srec')

with file.view(0x1000, 0x3FFC) as view:
    crc = binascii.crc32(view) & 0xFFFFFFFF  # remove sign

file.write(0x3FFC, struct.pack('>L', crc))
file.save('checkme_crc.srec')
```

1.4.5 Trim for bootloader

When using a bootloader, it is very important that the application being written does not overlap with the bootloader. Sometimes the compiler still generates stuff like a default interrupt table which should reside in the bootloader, and we need to get rid of it, as well as everything outside the address range allocated for the application itself.

This example shows how to trim the application executable record file to the allocated address range `0x8000-0x1FFFF`. Being written to a flash memory, unused memory byte cells default to `0xFF`.

```
from hexrec import load, SrecFile

in_file = load('application.mot')
data = in_file.read(0x8000, 0x1FFFF+1, fill=0xFF)

out_file = SrecFile.from_bytes(data, offset=0x8000)
out_file.save('app_trimmed.mot')
```

This can also be done by running the *hexrec* package as a command line tool:

```
$ hexrec crop -s 0x8000 -e 0x20000 -v 0xFF app_original.hex app_trimmed.srec
```

By contrast, we need to fill the application range within the bootloader image with `0xFF`, so that no existing application will be available again. Also, we need to preserve the address range `0x3F800-0x3FFFF` because it already contains some important data.

```
from hexrec import load

file = load('bootloader.hex')
file.fill(0x8000, 0x1FFFF+1, 0xFF)
file.clear(0x3F800, 0x3FFFF+1)
file.save('boot_fixed.hex')
```

With the command line interface, it can be done via a two-pass processing, first to fill the application range, then to clear the reserved range. Please note that the first command is chained to the second one via standard output/input buffering (the virtual - file path, in `ihex` format as per `boot_original.hex`).

```
$ hexrec fill -s 0x8000 -e 0x20000 -v 0xFF boot_original.hex - | \
  hexrec clear -s 0x3F800 -e 0x40000 -i ihex - boot_fixed.srec
```

(newline continuation is backslash \ for a *Unix-like* shell, caret ^ for a *DOS* prompt).

1.4.6 Export ELF sections

The following example shows how to export *sections* stored within an *Executable and Linkable File* (*ELF*), compiled for a microcontroller. As per the previous example, only data within the range `0x8000-0x1FFFF` are kept, with the rest of the memory filled with the `0xFF` value.

```
from hexrec import SrecFile
from bytesparse import Memory
from elftools.elf.elffile import ELFFile  # "pyelftools" package

with open('appelf.elf', 'rb') as elf_stream:
    elf_file = ELFFile(elf_stream)

    memory = Memory(start=0x8000, endex=0x1FFFF+1)  # bounds set
    memory.fill(pattern=0xFF)  # between bounds

    for section in elf_file.iter_sections():
        if (section.header.sh_flags & 3) == 3:  # SHF_WRITE | SHF_ALLOC
            address = section.header.sh_addr
            data = section.data()
            memory.write(address, data)

out_file = SrecFile.from_memory(memory, header=b'Source: appelf.elf\0')
out_file.save('appelf.srec')
```

1.5 Installation

From PyPI (might not be the latest version found on *github*):

```
$ pip install hexrec
```

From the source code root directory:

```
$ pip install .
```

1.6 Development

To run the all the tests:

```
$ pip install tox
$ tox
```

CHAPTER

TWO

INSTALLATION

2.1 From *PyPI*

At the command line:

```
$ pip install hexrec
```

The package found on *PyPI* might be outdated with respect to the source repository.

2.2 From source

At the command line, at the root of the source directory:

```
$ pip install .
```

CHAPTER

THREE

COMMAND LINE INTERFACE

3.1 hexrec

A set of command line utilities for common operations with record files.

Being built with Click, all the commands follow POSIX-like syntax rules, as well as reserving the virtual file path - for command chaining via standard output/input buffering.

```
hexrec [OPTIONS] COMMAND [ARGS]...
```

3.1.1 align

Pads blocks to align their boundaries.

`INFILE` is the path of the input file. Set to - to read from standard input; input format required.

`OUTFILE` is the path of the output file. Set to - to write to standard output. Leave empty to overwrite `INFILE`.

```
hexrec align [OPTIONS] [INFILE] [OUTFILE]
```

Options

-i, --input-format <input_format>

Forces the input file format. Required for the standard input.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-o, --output-format <output_format>

Forces the output file format. By default it is that of the input file.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-m, --modulo <modulo>

Alignment modulo.

Default

4

-s, --start <start>

Inclusive start address. Negative values are referred to the end of the data. By default it applies from the start of the data contents.

-e, --endex <endex>

Exclusive end address. Negative values are referred to the end of the data. By default it applies till the end of the data contents.

-v, --value <value>

Byte value used to flood alignment padding.

Default
0

-w, --width <width>

Sets the length of the record data field, in bytes. By default it is that of the input file.

Arguments

INFILE

Optional argument

OUTFILE

Optional argument

3.1.2 clear

Clears an address range.

INFILE is the path of the input file. Set to - to read from standard input; input format required.

OUTFILE is the path of the output file. Set to - to write to standard output. Leave empty to overwrite INFILE.

```
hexrec clear [OPTIONS] [INFILE] [OUTFILE]
```

Options

-i, --input-format <input_format>

Forces the input file format. Required for the standard input.

Options
asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-o, --output-format <output_format>

Forces the output file format. By default it is that of the input file.

Options
asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-s, --start <start>

Inclusive start address. Negative values are referred to the end of the data. By default it applies from the start of the data contents.

-e, --endex <endex>

Exclusive end address. Negative values are referred to the end of the data. By default it applies till the end of the data contents.

-w, --width <width>

Sets the length of the record data field, in bytes. By default it is that of the input file.

Arguments

INFILE

Optional argument

OUTFILE

Optional argument

3.1.3 convert

Converts a file to another format.

`INFILE` is the list of paths of the input files. Set to `-` to read from standard input; input format required.

`OUTFILE` is the path of the output file. Set to `-` to write to standard output. Leave empty to overwrite `INFILE`.

```
hexrec convert [OPTIONS] [INFILE] [OUTFILE]
```

Options

-i, --input-format <input_format>

Forces the input file format. Required for the standard input.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-o, --output-format <output_format>

Forces the output file format. By default it is that of the input file.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-w, --width <width>

Sets the length of the record data field, in bytes. By default it is that of the input file.

Arguments

INFILE

Optional argument

OUTFILE

Optional argument

3.1.4 crop

Selects data from an address range.

`INFILE` is the path of the input file. Set to `-` to read from standard input; input format required.

`OUTFILE` is the path of the output file. Set to `-` to write to standard output. Leave empty to overwrite `INFILE`.

```
hexrec crop [OPTIONS] [INFILE] [OUTFILE]
```

Options

-i, --input-format <input_format>

Forces the input file format. Required for the standard input.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-o, --output-format <output_format>

Forces the output file format. By default it is that of the input file.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-s, --start <start>

Inclusive start address. Negative values are referred to the end of the data. By default it applies from the start of the data contents.

-e, --endex <endex>

Exclusive end address. Negative values are referred to the end of the data. By default it applies till the end of the data contents.

-v, --value <value>

Byte value used to flood the address range. By default, no flood is performed.

-w, --width <width>

Sets the length of the record data field, in bytes. By default it is that of the input file.

Arguments

INFILE

Optional argument

OUTFILE

Optional argument

3.1.5 delete

Deletes an address range.

INFILE is the path of the input file. Set to - to read from standard input; input format required.

OUTFILE is the path of the output file. Set to - to write to standard output. Leave empty to overwrite INFILE.

```
hexrec delete [OPTIONS] [INFILE] [OUTFILE]
```

Options

-i, --input-format <input_format>

Forces the input file format. Required for the standard input.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-o, --output-format <output_format>

Forces the output file format. By default it is that of the input file.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-s, --start <start>

Inclusive start address. Negative values are referred to the end of the data. By default it applies from the start of the data contents.

-e, --endex <endex>

Exclusive end address. Negative values are referred to the end of the data. By default it applies till the end of the data contents.

-w, --width <width>

Sets the length of the record data field, in bytes. By default it is that of the input file.

Arguments

INFILE

Optional argument

OUTFILE

Optional argument

3.1.6 fill

Fills an address range with a byte value.

INFILE is the path of the input file. Set to - to read from standard input; input format required.

OUTFILE is the path of the output file. Set to - to write to standard output. Leave empty to overwrite INFILE.

```
hexrec fill [OPTIONS] [INFILE] [OUTFILE]
```

Options

-i, --input-format <input_format>

Forces the input file format. Required for the standard input.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-o, --output-format <output_format>

Forces the output file format. By default it is that of the input file.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-v, --value <value>

Byte value used to fill the address range.

Default

0

-s, --start <start>

Inclusive start address. Negative values are referred to the end of the data. By default it applies from the start of the data contents.

-e, --endex <endex>

Exclusive end address. Negative values are referred to the end of the data. By default it applies till the end of the data contents.

-w, --width <width>

Sets the length of the record data field, in bytes. By default it is that of the input file.

Arguments

INFILE

Optional argument

OUTFILE

Optional argument

3.1.7 flood

Fills emptiness of an address range with a byte value.

INFILE is the path of the input file. Set to - to read from standard input; input format required.

OUTFILE is the path of the output file. Set to - to write to standard output. Leave empty to overwrite INFILE.

```
hexrec flood [OPTIONS] [INFILE] [OUTFILE]
```

Options

-i, --input-format <input_format>

Forces the input file format. Required for the standard input.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-o, --output-format <output_format>

Forces the output file format. By default it is that of the input file.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-v, --value <value>

Byte value used to flood the address range.

Default

0

-s, --start <start>

Inclusive start address. Negative values are referred to the end of the data. By default it applies from the start of the data contents.

-e, --endex <endex>

Exclusive end address. Negative values are referred to the end of the data. By default it applies till the end of the data contents.

-w, --width <width>

Sets the length of the record data field, in bytes. By default it is that of the input file.

Arguments

INFILE

Optional argument

OUTFILE

Optional argument

3.1.8 hd

Display file contents in hexadecimal, decimal, octal, or ascii.

The hexdump utility is a filter which displays the specified files, or standard input if no files are specified, in a user-specified format.

Below, the length and offset arguments may be followed by the multiplicative suffixes KiB (=1024), MiB (=1024*1024), and so on for GiB, TiB, PiB, EiB, ZiB and YiB (the "iB" is optional, e.g., "K" has the same meaning as "KiB"), or the suffixes KB (=1000), MB (=1000*1000), and so on for GB, TB, PB, EB, ZB and YB.

For each input file, hexdump sequentially copies the input to standard output, transforming the data according to the format strings specified by the -e and -f options, in the order that they were specified.

```
hexrec hd [OPTIONS] [INFILE]
```

Options

-b, --one-byte-octal

One-byte octal display. Display the input offset in hexadecimal, followed by sixteen space-separated, three-column, zero-filled bytes of input data, in octal, per line.

-X, --one-byte-hex

One-byte hexadecimal display. Display the input offset in hexadecimal, followed by sixteen space-separated, two-column, zero-filled bytes of input data, in hexadecimal, per line.

-c, --one-byte-char

One-byte character display. Display the input offset in hexadecimal, followed by sixteen space-separated, three-column, space-filled characters of input data per line.

-d, --two-bytes-decimal

Two-byte decimal display. Display the input offset in hexadecimal, followed by eight space-separated, five-column, zero-filled, two-byte units of input data, in unsigned decimal, per line.

-o, --two-bytes-octal

Two-byte octal display. Display the input offset in hexadecimal, followed by eight space-separated, six-column, zero-filled, two-byte quantities of input data, in octal, per line.

-x, --two-bytes-hex

Two-byte hexadecimal display. Display the input offset in hexadecimal, followed by eight space-separated, four-column, zero-filled, two-byte quantities of input data, in hexadecimal, per line.

-n, --length <length>

Interpret only length bytes of input.

-s, --skip <skip>

Skip offset bytes from the beginning of the input.

-v, --no_squeezing

The -v option causes hexdump to display all input data. Without the -v option, any number of groups of output lines which would be identical to the immediately preceding group of output lines (except for the input offsets), are replaced with a line comprised of a single asterisk.

-U, --upper

Uses upper case hex letters on address and data.

-I, --input-format <input_format>

Forces the input file format.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-V, --version

Print version and exit.

Arguments

INFILE
Optional argument

3.1.9 hexdump

Display file contents in hexadecimal, decimal, octal, or ascii.

The hexdump utility is a filter which displays the specified files, or standard input if no files are specified, in a user-specified format.

Below, the length and offset arguments may be followed by the multiplicative suffixes KiB (=1024), MiB (=1024*1024), and so on for GiB, TiB, PiB, EiB, ZiB and YiB (the "iB" is optional, e.g., "K" has the same meaning as "KiB"), or the suffixes KB (=1000), MB (=1000*1000), and so on for GB, TB, PB, EB, ZB and YB.

For each input file, hexdump sequentially copies the input to standard output, transforming the data according to the format strings specified by the -e and -f options, in the order that they were specified.

```
hexrec hexdump [OPTIONS] [INFILE]
```

Options

-b, --one-byte-octal
One-byte octal display. Display the input offset in hexadecimal, followed by sixteen space-separated, three-column, zero-filled bytes of input data, in octal, per line.

-X, --one-byte-hex
One-byte hexadecimal display. Display the input offset in hexadecimal, followed by sixteen space-separated, two-column, zero-filled bytes of input data, in hexadecimal, per line.

-c, --one-byte-char
One-byte character display. Display the input offset in hexadecimal, followed by sixteen space-separated, three-column, space-filled characters of input data per line.

-C, --canonical
Canonical hex+ASCII display. Display the input offset in hexadecimal, followed by sixteen space-separated, two-column, hexadecimal bytes, followed by the same sixteen bytes in %_p format enclosed in | characters. Invoking the program as hd implies this option.

-d, --two-bytes-decimal
Two-byte decimal display. Display the input offset in hexadecimal, followed by eight space-separated, five-column, zero-filled, two-byte units of input data, in unsigned decimal, per line.

-o, --two-bytes-octal
Two-byte octal display. Display the input offset in hexadecimal, followed by eight space-separated, six-column, zero-filled, two-byte quantities of input data, in octal, per line.

-x, --two-bytes-hex
Two-byte hexadecimal display. Display the input offset in hexadecimal, followed by eight space-separated, four-column, zero-filled, two-byte quantities of input data, in hexadecimal, per line.

-n, --length <length>
Interpret only length bytes of input.

-s, --skip <skip>
: Skip offset bytes from the beginning of the input.

-v, --no_squeezing
: The -v option causes hexdump to display all input data. Without the -v option, any number of groups of output lines which would be identical to the immediately preceding group of output lines (except for the input offsets), are replaced with a line comprised of a single asterisk.

-U, --upper
: Uses upper case hex letters on address and data.

-I, --input-format <input_format>
: Forces the input file format.

  Options
  asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-V, --version
: Print version and exit.

Arguments

INFILE
: Optional argument

3.1.10 merge

Merges multiple files.

`INFILES` is the list of paths of the input files. Set any to `-` or none to read from standard input; input format required.

`OUTFILE` is the path of the output file. Set to `-` to write to standard output.

Every file of `INFILES` will overwrite data of previous files of the list where addresses overlap.

```
hexrec merge [OPTIONS] [INFILES]... OUTFILE
```

Options

-i, --input-format <input_format>
: Forces the input file format for all input files. Required for the standard input.

  Options
  asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-o, --output-format <output_format>
: Forces the output file format. By default it is that of the input file.

  Options
  asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-w, --width <width>
: Sets the length of the record data field, in bytes. By default it is that of the input file.

--clear-holes
: Merges memory holes, clearing data at their place.

Arguments

INFILES
Optional argument(s)

OUTFILE
Required argument

3.1.11 shift

Shifts data addresses.

INFILE is the path of the input file. Set to - to read from standard input; input format required.

OUTFILE is the path of the output file. Set to - to write to standard output. Leave empty to overwrite INFILE.

```
hexrec shift [OPTIONS] [INFILE] [OUTFILE]
```

Options

-i, --input-format <input_format>
Forces the input file format. Required for the standard input.

Options
asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-o, --output-format <output_format>
Forces the output file format. By default it is that of the input file.

Options
asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-n, --amount <amount>
Address shift to apply.

-w, --width <width>
Sets the length of the record data field, in bytes. By default it is that of the input file.

Arguments

INFILE
Optional argument

OUTFILE
Optional argument

3.1.12 srec

Motorola SREC specific

```
hexrec srec [OPTIONS] COMMAND [ARGS]...
```

del-header

Deletes the header data record.

The header record is expected to be the first. All other records are kept as-is. No file-wise validation occurs.

`INFILE` is the path of the input file; 'srec' record type. Set to - to read from standard input.

`OUTFILE` is the path of the output file. Set to - to write to standard output. Leave empty to overwrite `INFILE`.

```
hexrec srec del-header [OPTIONS] [INFILE] [OUTFILE]
```

Arguments

INFILE

Optional argument

OUTFILE

Optional argument

get-header

Gets the header data.

`INFILE` is the path of the input file; 'srec' record type. Set to - to read from standard input.

```
hexrec srec get-header [OPTIONS] [INFILE]
```

Options

-f, --format <format>

Header data format.

Default

`ascii`

Options

ascii | hex | HEX | hex. | HEX. | hex- | HEX- | hex: | HEX: | hex_ | HEX_ | hex | HEX

Arguments

INFILE
Optional argument

set-header

Sets the header data record.

The header record is expected to be the first. All other records are kept as-is. No file-wise validation occurs.

`INFILE` is the path of the input file; 'srec' record type. Set to `-` to read from standard input.

`OUTFILE` is the path of the output file. Set to `-` to write to standard output. Leave empty to overwrite `INFILE`.

```
hexrec srec set-header [OPTIONS] HEADER [INFILE] [OUTFILE]
```

Options

-f, --format <format>
Header data format.

Default
`ascii`

Options
ascii | hex | HEX | hex. | HEX. | hex- | HEX- | hex: | HEX: | **hex_** | **HEX_** | hex | HEX

Arguments

HEADER
Required argument

INFILE
Optional argument

OUTFILE
Optional argument

3.1.13 validate

Validates a record file.

`INFILE` is the path of the input file. Set to `-` to read from standard input; input format required.

```
hexrec validate [OPTIONS] [INFILE]
```

Options

-i, --input-format <input_format>

Forces the input file format. Required for the standard input.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

Arguments

INFILE

Optional argument

3.1.14 xxd

Emulates the xxd command.

Please refer to the xxd manual page to know its features and caveats.

Some parameters were changed to satisfy the POSIX-like command line parser.

```
hexrec xxd [OPTIONS] [INFILE] [OUTFILE]
```

Options

-a, --autoskip

Toggles autoskip.

A single '*' replaces null lines.

-b, --bits

Binary digits.

Switches to bits (binary digits) dump, rather than hexdump. This option writes octets as eight digits of '1' and '0' instead of a normal hexadecimal dump. Each line is preceded by a line number in hexadecimal and followed by an ASCII (or EBCDIC) representation. The argument switches -r, -p, -i do not work with this mode.

-c, --cols <cols>

Formats <cols> octets per line. Max 256.

Defaults: normal 16, -i 12, -p 30, -b 6.

-E, --ebcdic, --EBCDIC

Uses EBCDIC charset.

Changes the character encoding in the right-hand column from ASCII to EBCDIC. This does not change the hexadecimal representation. The option is meaningless in combinations with -r, -p or -i.

-e, --endian

Switches to little-endian hexdump.

This option treats byte groups as words in little-endian byte order. The default grouping of 4 bytes may be changed using -g. This option only applies to hexdump, leaving the ASCII (or EBCDIC) representation unchanged. The switches -r, -p, -i do not work with this mode.

-g, --groupsize <groupsize>

Byte group size.

Separates the output of every <groupsize> bytes (two hex characters or eight bit-digits each) by a whitespace. Specify <groupsize> 0 to suppress grouping. <groupsize> defaults to 2 in normal mode, 4 in little-endian mode and 1 in bits mode. Grouping does not apply to -p or -i.

-i, --include

Output in C include file style.

A complete static array definition is written (named after the input file), unless reading from standard input.

-l, --length, --len <length>

Stops after writing <length> octets.

-o, --offset <offset>

Adds <offset> to the displayed file position.

-p, --postscript, --plain, --ps

Outputs in postscript continuous hexdump style.

Also known as plain hexdump style.

-q, --quadword

Uses 64-bit addressing.

-r, --revert

Reverse operation.

Convert (or patch) hexdump into binary. If not writing to standard output, it writes into its output file without truncating it. Use the combination -r and -p to read plain hexadecimal dumps without line number information and without a particular column layout. Additional Whitespace and line breaks are allowed anywhere.

-k, --seek <oseek>

Output seeking.

When used after -r reverts with -o added to file positions found in hexdump.

-s <iseek>

Input seeking.

Starts at <s> bytes absolute (or relative) input offset. Without -s option, it starts at the current file position. The prefix is used to compute the offset. '+' indicates that the seek is relative to the current input position. '-' indicates that the seek should be that many characters from the end of the input. '+-' indicates that the seek should be that many characters before the current stdin file position.

-U, --upper-all

Uses upper case hex letters on address and data.

-u, --upper

Uses upper case hex letters on data only.

-I, --input-format <input_format>

Forces the input file format.

Options

asciihex | avr | ihex | mos | raw | srec | titxt | xtek

-O, --output-format <output_format>

Forces the output file format.

Options
asciihex | avr | ihex | mos | raw | srec | titxt | xtek

--seek-zeroes, --no-seek-zeroes

Output seeking writes zeroes while seeking.

Default
True

-v, --version

Prints the package version number.

Arguments

INFILE

Optional argument

OUTFILE

Optional argument

CHAPTER

FOUR

REFERENCE

hexrec.base	Base types and classes.
hexrec.formats	Record formats.
hexrec.formats.asciihex	ASCII-hex format.
hexrec.formats.avr	Atmel Generic format.
hexrec.formats.ihex	Intel HEX format.
hexrec.formats.mos	MOS Technology format.
hexrec.formats.raw	Binary format.
hexrec.formats.sqtp	Microchip Serial Quick Time Programming format.
hexrec.formats.srec	Motorola S-record format.
hexrec.formats.titxt	Texas Instruments TI-TXT format.
hexrec.formats.xtek	Tektronix extended HEX format.
hexrec.hexdump	Emulation of the hexdump utility.
hexrec.utils	Generic utility functions.
hexrec.xxd	Emulation of the xxd utility.

4.1 base

Base types and classes.

Attributes

FILE_TYPES	Registered record file types.
TOKEN_COLOR_CODES	ANSI color codes for each possible token type.

4.1.1 FILE_TYPES

hexrec.base.**FILE_TYPES: MutableMapping[str, Type[*BaseFile*]] = {'asciihex': <class 'hexrec.formats.asciihex.AsciiHexFile'>, 'avr': <class 'hexrec.formats.avr.AvrFile'>, 'ihex': <class 'hexrec.formats.ihex.IhexFile'>, 'mos': <class 'hexrec.formats.mos.MosFile'>, 'raw': <class 'hexrec.formats.raw.RawFile'>, 'srec': <class 'hexrec.formats.srec.SrecFile'>, 'titxt': <class 'hexrec.formats.titxt.TiTxtFile'>, 'xtek': <class 'hexrec.formats.xtek.XtekFile'>}**

Registered record file types.

4.1.2 TOKEN_COLOR_CODES

hexrec.base.**TOKEN_COLOR_CODES: Mapping[str, bytes] = {'': b'\x1b[0m', '<': b'\x1b[0m', '>': b'\x1b[0m', 'address': b'\x1b[31m', 'addrlen': b'\x1b[33m', 'after': b'\x1b[0m', 'before': b'\x1b[0m', 'begin': b'\x1b[33m', 'checksum': b'\x1b[35m', 'count': b'\x1b[34m', 'data': b'\x1b[36m', 'dataalt': b'\x1b[96m', 'end': b'\x1b[0m', 'tag': b'\x1b[32m'}**

ANSI color codes for each possible token type.

Functions

colorize_tokens	Prepends ANSI color codes to record field tokens.
convert	Converts a file into another format.
guess_format_name	Guesses the record format name.
guess_format_type	Guesses the record format object type.
load	Loads a file.
merge	Merges multiple files.

4.1.3 colorize_tokens

hexrec.base.**colorize_tokens**(*tokens*, *altdata=True*)

Prepends ANSI color codes to record field tokens.

For each token within *tokens*, its key is used to look up the ANSI color code from *TOKEN_COLOR_CODES*. The retrieved code (byte string) is prepended to the token. All the modified tokens are then collected and returned.

Parameters

- **tokens** (*dict*) – A mapping of each token key name to token byte string.
- **altdata** (*bool*) – If true, it alternates each byte (two hex digits) between the ANSI color codes mapped with keys `data` (even byte index) and `dataalt` (odd byte index). If false, only the `data` code is prepended.

Returns

dict – *tokens* with prepended ANSI color codes.

Examples

```
>>> from hexrec.base import colorize_tokens
>>> from hexrec import IhexFile
>>> from pprint import pprint
```

```
>>> record = IhexFile.Record.create_end_of_file()
>>> tokens = record.to_tokens()
>>> pprint(tokens)
{'address': b'0000',
 'after': b'',
 'before': b'',
 'begin': b':',
 'checksum': b'FF',
```

(continues on next page)

(continued from previous page)

```
 'count': b'00',
 'data': b'',
 'end': b'\r\n',
 'tag': b'01'}
>>> colorized = colorize_tokens(tokens)
>>> pprint(colorized)
{'<': b'\x1b[0m',
 '>': b'\x1b[0m',
 'address': b'\x1b[31m0000',
 'begin': b'\x1b[33m:',
 'checksum': b'\x1b[35mFF',
 'count': b'\x1b[34m00',
 'end': b'\x1b[0m\r\n',
 'tag': b'\x1b[32m01'}
```

4.1.4 convert

hexrec.base.**convert**(*in_path*, *out_path*, *in_format=None*, *out_format=None*)

Converts a file into another format.

This is a simple helper function for basic conversion of a file on the filesystem into another record format.

The function returns the input and output file objects, for further processing by the user.

Parameters

- **in_path** (*str*) – Input file path.
- **out_path** (*str*) – Output file path. It can be the same as *in_path*.
- **in_format** (*str*) – Name of the input format, within *FILE_TYPES*. If None, it is guessed via *guess_format_name()*.
- **out_format** (*str*) – Name of the output format, within *FILE_TYPES*. If None, it is guessed via *guess_format_name()*.

Returns

(in_file, out_file) – Input and output file objects used internally.

See also:

FILE_TYPES *BaseFile.convert()* *BaseFile.load()* *BaseFile.save()*

Examples

```
>>> from hexrec import convert
>>> convert('simple.hex', 'simple.srec')
>>> convert('simple.hex', 'simple.srec', in_format='ihex', out_format='srec')
```

4.1.5 guess_format_name

hexrec.base.**guess_format_name**(*file_path*)

Guesses the record format name.

It analyzes the file extension by *file_path* against all the record formats registered into *FILE_TYPES*. The first record format to match the extension within its own *BaseFile.FILE_EXT* is returned.

If no extension matches, the file is parsed until a *BaseFile.parse()* succeds (no exception raised).

Parameters
file_path (*str*) – File path to analyze.

Returns
str – Record format registered within *FILE_TYPES*.

Raises
ValueError – Cannot guess record file format.

See also:

FILE_TYPES

Examples

```
>>> from hexrec import guess_format_name
>>> guess_format_name('simple.hex')
'ihex'
>>> guess_format_name('simple.srec')
'srec'
>>> guess_format_name('simple.s19')
'srec'
>>> guess_format_name('simple.mot')
'srec'
>>> guess_format_name('data.dat')
'raw'
```

4.1.6 guess_format_type

hexrec.base.**guess_format_type**(*file_path*)

Guesses the record format object type.

It calls *guess_format_name()* to return the registered object type within *FILE_TYPES*.

Parameters
file_path (*str*) – File path to analyze.

Returns
str – Record object type registered within *FILE_TYPES*.

Raises
ValueError – Cannot guess record file format.

See also:

FILE_TYPES

Examples

```
>>> from hexrec import guess_format_type
>>> guess_format_type('simple.hex')
<class 'hexrec.formats.ihex.IhexFile'>
>>> guess_format_type('simple.srec')
<class 'hexrec.formats.srec.SrecFile'>
>>> guess_format_type('simple.s19')
<class 'hexrec.formats.srec.SrecFile'>
>>> guess_format_type('simple.mot')
<class 'hexrec.formats.srec.SrecFile'>
>>> guess_format_type('data.dat')
<class 'hexrec.formats.raw.RawFile'>
```

4.1.7 load

hexrec.base.**load**(*in_path*, **load_args*, *in_format=None*, ***load_kwargs*)

Loads a file.

This is a simple helper function to load a record file from the filesystem.

All the custom *load_args* and *load_kwargs* are forwarded to the actual underlying call to *BaseFile.load()*.

Parameters

- **in_path** (*str*) – Input file path.
- **in_format** (*str*) – Name of the input format, within *FILE_TYPES*. If None, it is guessed via *guess_format_name()*.

Returns

BaseFile – The loaded record file object.

See also:

FILE_TYPES *guess_format_name()* *BaseFile.load()*

Examples

```
>>> from hexrec import load
>>> load('simple.hex')
<hexrec.formats.ihex.IhexFile object at ...>
>>> load('simple.hex', in_format='ihex')
<hexrec.formats.ihex.IhexFile object at ...>
>>> load('simple.hex', ignore_errors=True)
<hexrec.formats.ihex.IhexFile object at ...>
```

4.1.8 merge

hexrec.base.**merge**(*in_paths*, *out_path=None*, *in_formats=None*, *out_format=None*)

Merges multiple files.

This is a simple helper function to load multiple files from the filesystem and merge into a new one.

The function returns the list of input file objects and the output file object, for further processing by the user.

Parameters

- **in_paths** (*str list*) – Sequence of input file paths, in merging order.
- **out_path** (*str*) – Output file path. It can be the same one of *in_paths*.
- **in_formats** (*str list*) – Name of the input formats, within *FILE_TYPES*. If the sequence or an item is None, that is guessed via *guess_format_name()*.
- **out_format** (*str*) – Name of the output format, within *FILE_TYPES*. If None, it is guessed via *guess_format_name()*.

Returns

(in_files, out_file) – The record file objects used internally.

See also:

FILE_TYPES *guess_format_name()* *BaseFile.load()* *BaseFile.merge()* *BaseFile.save()*

Examples

```
>>> from hexrec import merge
>>> merge(['data.dat', 'simple.hex'], 'merge.xtek')
([<hexrec.formats.raw.RawFile object at ...>,
  <hexrec.formats.ihex.IhexFile object at ...>],
 <hexrec.formats.xtek.XtekFile object at ...>)
>>> merge(['data.dat', 'simple.hex'], 'merge.xtek',
...       in_formats=['raw', None], out_format='xtek')
([<hexrec.formats.raw.RawFile object at ...>,
  <hexrec.formats.ihex.IhexFile object at ...>],
 <hexrec.formats.xtek.XtekFile object at ...>)
```

Classes

BaseFile	Record file.
BaseRecord	Record.
BaseTag	Record tag.

4.1.9 BaseFile

class hexrec.base.**BaseFile**

Record file.

A *record file* contains a sequence of *records* (*BaseRecord*), which can be serialized to transfer some *binary* data across systems, usually an executable program of some configuration data for an *embedded system*.

The *BaseFile* class provides a useful abstraction of a record file, to create, edit, or convert across file *formats*.

A *BaseFile* instance has a dual role:

- *records role*: to host the sequence of *records* for parsing and serialization;
- *memory role*: to host the equivalent *memory* image and *meta* information.

Both roles can be active at the same time only when both representations are coherent.

The *records* role is useful to *load()* or *save()* the instance against the filesystem. The individual records can be edited via the *records* attribute. This is considered an advanced feature, for debug or experimentation.

The *memory* role is for editing the equivalent *memory image*. The *BaseFile* provides common methods to *read()*, *write()*, *cut()*, *crop()*, *clear()*, *delete()*, *fill()*, *flood()*, and more. Furthermore, this role also abstracts *meta* information, like the maximum *data* size (*maxdatalen*), or the *start address* (if available for the specific file *format*). Advanced editing can be performed via the underlying *memory* attribute, thanks to the powerful `bytesparse.Memory` class (from the bytesparse Python package), the *get_meta()* and *set_meta()* methods, or their equivalent Python property wrappers. The *memory* also provides additional methods and properties for in-depth analysis (min/max address, gaps, spans, find, etc.).

A *BaseFile* instance can impersonate each role via:

- *apply_records()* to mirror the *records* sequence into the equivalent *memory* and *meta*.
- *update_records()* to mirror the *memory* and *meta* into an actual *records* sequence.

Please note that reading from the *records* or *memory* properties automatically activates the corresponding role if inactive (i.e. *update_records()* if `_records` is `None`, *apply_records()* if `_memory` is `None`).

Beware that any editing done within a role may invalidate the other role, if both *records* and *memory* + *meta* are instantiated. For example, setting the *maxdatalen* property means that records must be updated to mirror the new maximum *data* field length. Records must also be invalidated whenever the *memory* is altered.

Usually, *meta* property setters automatically invalidate *records* on change (via *discard_records()*). Instead, only the explicit *memory* methods of *BaseFile()* do it automatically; any operations on the *memory* itself require *update_records()* be called to mirror any changes.

Instantiation of a new *BaseFile* instance should be performed via:

- *BaseFile* for an empty file (only!); *memory* role.
- *from_bytes()*, *from_blocks()*, *from_memory()* to create from existing byte-like, blocks, or `bytesparse.Memory`; *memory* role.
- *copy()*, *convert()* to clone an existing *BaseFile*; *memory* role.
- *from_records()* to create from existing records; *records* role.
- *load()* to load records from the filesystem (via `open()`); *records* role.
- *parse()* to load records from a byte stream; *records* role.

Validation of *records* is performed via `validate()`.

Most methods return *self*, and as such they can be *chained*, instead of being forced to call each method in a separate statement. This comes handy to write some one-liners or generally shorter scripts.

Please note that within the examples of this documentation *chaining* is rarely used, for easier reading. Any places where interactive console commands would return *self*, the returned value is suppressed by assigning _, not to disrupt the output (e.g.: `_ = file.print()` outputs only record content to *stdout*).

Attributes

DEFAULT_DATALEN	Default data attribute length.
FILE_EXT	Supported filename extensions.
META_KEYS	Meta information key names.
Record	Record object type.
maxdatalen	Maximum byte size of the data field.
memory	Memory object stored by records role.
records	Records stored by records role.

Methods

__init__	
align	Pads blocks to align their boundaries.
append	Appends a byte.
apply_records	Applies records to memory and meta.
clear	Clears data within a range.
convert	Converts a file object to another format.
copy	Copies within a range.
crop	Clears data outside a range.
cut	Cuts data within a range.
delete	Deletes data within a range.
discard_memory	Discards underlying memory.
discard_records	Discards underlying records.
extend	Concatenates data.
fill	Fills a range.
find	Finds a substring.
flood	Floods a range.
from_blocks	Creates a file object from a memory object.
from_bytes	Creates a file object from a byte string.
from_memory	Creates a file object from a memory object.
from_records	Creates a file object from records.
get_address_max	Maximum address within memory.
get_address_min	Minimum address within memory.
get_holes	List of memory holes.
get_meta	Meta information.
get_spans	List of memory block spans.
index	Finds a substring.
load	Loads a file object from the filesystem.
merge	Merges data onto the file.
parse	Parses records from a byte stream.
print	Prints record content to stdout.
read	Extracts a substring.
save	Saves a file object into the filesystem.

continues on next page

Table 1 – continued from previous page

serialize	Serializes records onto a byte stream.
set_meta	Sets meta information.
shift	Shifts data addresses by an offset.
split	Splits into parts.
update_records	Applies memory and meta to records.
validate_records	Validates records.
view	Memory view.
write	Writes data into the file.

DEFAULT_DATALEN: int = 16

Default data attribute length.

Default value for the *maxdatalen* *meta*, which sets the maximum size of `BaseRecord.data` field values.

FILE_EXT: Sequence[str] = []

Supported filename extensions.

Sequence of file name extension substrings (e.g. `.hex`). This list is used by functions like *guess_format_name()* to manage mapping of file *formats*.

META_KEYS: Sequence[str] = ['maxdatalen']

Meta information key names.

Sequence of key strings listing the supported *meta* information of this file *format*.

Record: Type[*BaseRecord*] = None

Record object type.

This class attribute indicates the *BaseRecord* class used by this *BaseFile* class.

__add__(*other*)

Concatenates with another file.

Equivalent to *copy()* then *extend()*.

Parameters
other (*BaseFile* or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – Concatenation of *self* and *other*.

See also:

copy() *extend()*

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 + file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
```

(continues on next page)

(continued from previous page)

```
>>> file4 = file3 + b'789'
>>> file4.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__bool__()

bool: Has data records or memory.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> bool(file)
False
>>> _ = file.append(0)
>>> bool(file)
True
```

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> file = IhexFile.from_records([IhexRecord.create_end_of_file()])
>>> bool(file)
False
>>> file.records.insert(0, IhexRecord.create_data(0, b'\0'))
>>> bool(file)
True
```

__delitem__(*key*)

Deletes a range.

Parameters
key (*slice or int*) – Range to delete.

See also:

bytesparse.base.MutableMemory.__delitem__()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> del file[457]
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xz']]
>>> del file[125:457]
>>> file.memory.to_blocks()
[[123, b'abz']]
```

__eq__(*other*)

Equality test.

The file objects *self* and *other* are considered *equal* if the inequality tests of *__ne__()* result false.

Returns

bool – *self* and *other* are *equal*.

See Also

__ne__()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 == file2
True
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 == file3
False
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 == file4
False
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file == ihex_file
False
>>> srec_file.memory == ihex_file.memory
True
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__getitem__(*key*)

Extracts a range.

Parameters

key (*slice or int*) – Range to extract.

Raises

ValueError – invalid range.

See also:

bytesparse.base.ImmutableMemory.__getitem__()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> chr(file[457])
'y'
>>> repr(file[333])
'None'
>>> file[123:125]
b'ab'
>>> file[125:457]
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

__hash__ = None

__iadd__(*other*)

Concatenates data.

Equivalent to *extend()*.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (*BaseFile* or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – *self*.

See also:

memory *extend()* *discard_records()* bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 += file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file1 += b'789'
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__init__()

__ior__(*other*)

Merges with another file.

Equivalent to *merge()*.

Any stored *records* are discarded upon return.

Parameters
other (*BaseFile* or bytes) – Other file or bytes to merge.

Returns
BaseFile – *self*.

See also:

merge() discard_records()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 |= file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file1 |= b'789'
>>> file1.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__ne__(*other*)

Inequality test.

The file objects *self* and *other* are considered *unequal* if any of the following tests result true:

- Both have *memory role* (i.e. *memory*), resulting unequal;
- Both have *records role* (i.e. *records*), resulting unequal;
- *other* does not have a *meta* listed by *META_KEYS*;
- A *meta* value (among those of *META_KEYS*) is different.

Returns
bool – *self* and *other* are *unequal*.

See also:

__eq__() memory records META_KEYS

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 != file2
False
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 != file3
True
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 != file4
True
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file != ihex_file
True
>>> srec_file.memory != ihex_file.memory
False
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__or__(*other*)

Merges with another file.

Equivalent to *copy()* then *merge()*.

Parameters
other (*BaseFile* or bytes) – Other file or bytes to merge.

Returns
BaseFile – *self* merged with *other*.

See also:

copy() *merge()*

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 | file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
```

(continues on next page)

(continued from previous page)

```
>>> file4 = file3 | b'789'
>>> file4.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__setitem__(*key*, *value*)

Sets a range.

Parameters

- **key** (*slice or int*) – Range to set.
- **value** (bytes, `bytesparse.base.ImmutableMemory`, `None`) – Value(s) to set. `None` acts like *clear()*.

Raises

ValueError – invalid range.

See also:

`bytesparse.base.MutableMemory.__setitem__()` *clear()*

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file[124] = b'?'
>>> file.memory.to_blocks()
[[123, b'a?c'], [456, b'xyz']]
>>> file[:125] = None
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'xyz']]
>>> file[457:458] = b'789'
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'x789z']]
```

__weakref__

list of weak references to the object (if defined)

classmethod _is_line_empty(*line*)

Empty line check.

Tells whether a *line* has no meaningful content (e.g. all whitespace). The check itself depends on the implementing file *format*. It may be used internally to skip empty lines, e.g. by *parse()*.

Parameters

line (*bytes*) – A line, byte string.

Returns

bool: The *line* is empty.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexFile._is_line_empty(b'')
True
>>> IhexFile._is_line_empty(b' \t\v\r\n')
True
>>> IhexFile._is_line_empty(b':00000001FF\r\n')
False
```

align(*modulo*, *start=None*, *endex=None*, *pattern=0*)

Pads blocks to align their boundaries.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*, so that memory blocks are aligned to the required *modulo*.

Any stored *records* are discarded upon return.

Parameters

- **modulo** (*int*) – Alignment modulo.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

BaseFile – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.align()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [134, b'xyz']])
>>> _ = file.align(4, pattern=b'.')
>>> file.memory.to_blocks()
[[120, b'...abc..'], [132, b'..xyz...']]
```

append(*item*)

Appends a byte.

It appends the *item* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters

item (*byte or int*) – Byte to append.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.append()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123)
>>> _ = file.append(b'.')
>>> _ = file.append(0)
>>> file.memory.to_blocks()
[[123, b'abc.\x00']]
```

apply_records()

Applies records to memory and meta.

This method processes the stored *records*, converting *data* as *memory*, and special records into their *meta* counterparts.

This effectively converts the *records role* into the *memory role* (keeping both).

The *memory* and *meta* are assigned upon return. Any exceptions being raised should not alter the file object.

Returns
BaseFile – *self*.

Raises
ValueError – *records* attribute not populated.

See also:

records memory get_meta() update_records()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_start_linear_address(456),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records, maxdatalen=16)
>>> _ = file.apply_records()
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
```

clear(*start=None*, *endex=None*)

Clears data within a range.

It clears the specified range of underlying *memory* object, making a memory hole.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

BaseFile – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.clear()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.clear(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

classmethod convert(*source*, *meta=True*)

Converts a file object to another format.

It copies the *memory* and *meta* of the *source* file object, creating a new one of the target *BaseFile* format type.

Parameters

- **source** (*BaseFile*) – Source file object to convert.
- **meta** (*bool*) – Copy *meta* information to the target file object. Only the keys of the target *META_KEYS* are processed.

Returns

BaseFile – Converted copy of *source* to the target format.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> source = IhexFile.from_blocks(blocks, startaddr=789)
>>> target = SrecFile.convert(source)
```

(continues on next page)

(continued from previous page)

```
>>> target.memory is source.memory
False
>>> target.memory == source.memory
True
>>> target.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 789}
```

copy(*start=None*, *endex=None*, *meta=True*)

Copies within a range.

It copied data within the specified range of the file object, creating a new one carrying the inner slice.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

BaseFile – *self*.

See also:

memory *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.copy(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

crop(*start=None*, *endex=None*)

Clears data outside a range.

It clears outside the specified range of underlying *memory* object, timming it.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.crop()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.crop(start=124, endex=132)
>>> file.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
```

cut(*start=None*, *endex=None*, *meta=False*)

Cuts data within a range.

It takes data within the specified range away from the file object, creating a new one carrying the inner slice. The inner slice is cleared from *self*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

BaseFile – *self*.

See also:

memory clear() get_meta() discard_records() bytesparse.base.MutableMemory.cut()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.cut(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

delete(*start=None*, *endex=None*)

Deletes data within a range.

It deletes the specified range of underlying *memory* object, shifting all subsequent data towards the collapsed range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns

BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.delete()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.delete(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'az']]
```

discard_memory()

Discards underlying memory.

The underlying *memory* object is assigned None.

If the underlying *records* object is None, it is assigned a new empty memory object.

Returns

BaseFile – *self*.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_bytes(b'abc', offset=123)
>>> _ = file.update_records()
>>> _ = file.discard_memory()
>>> _ = file.update_records()
Traceback (most recent call last):
    ...
ValueError: memory instance required
```

discard_records()

Discards underlying records.

The underlying *records* object is assigned None.

If the underlying *memory* object is None, it is assigned a new empty memory object.

Returns
BaseFile – *self*.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> _ = file.validate_records()
>>> _ = file.discard_records()
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: records required
```

extend(*other*)

Concatenates data.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (*BaseFile* or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> _ = file1.extend(file2)
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> _ = file1.extend(b'789')
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

fill(*start=None*, *endex=None*, *pattern=0*)

Fills a range.

It writes a *pattern* of bytes onto the underlying *memory* object, overwriting anything within the specified range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for filling.

Returns

BaseFile – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.fill()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.fill(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'a........z']]
```

find(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it returns `-1`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address; `-1` if not found.

See also:

index `bytesparse.base.ImmutableMemory.find()`

Notes

The internal *memory* might allow negative addresses for its stored data. In that case, *index()* would be more appropriate, because it raises an exception when the *item* is not found.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.find(b'yz')
457
>>> file.find(ord('b'))
124
>>> file.find(b'?')
-1
```

flood(*start=None*, *endex=None*, *pattern=0*)

Floods a range.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

BaseFile – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.flood()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.get_holes()
[(126, 130)]
>>> _ = file.flood(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'abc....xyz']]
```

classmethod from_blocks(*blocks*, ***meta*)

Creates a file object from a memory object.

The *blocks* are put into the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **blocks** (*list of blocks*) – Memory blocks to put into *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

BaseFile – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* `bytesparse.base.ImmutableMemory.from_blocks()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_bytes(*data*, *offset=0*, ***meta*)

Creates a file object from a byte string.

The byte string makes a single *data* block, placed at some offset within the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **data** (*bytes*) – A byte string used to make a single data block.
- **offset** (*int*) – Offset of the single data block within *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

BaseFile – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_bytes()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
8
```

classmethod **from_memory**(*memory=None*, ***meta*)

Creates a file object from a memory object.

The *memory* is set as the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **memory** (bytesparse.base.MutableMemory) – Memory object to set as *memory*. If None, an empty memory object is automatically created.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

BaseFile – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS bytesparse.base.MutableMemory

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from bytesparse import Memory
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> memory = Memory.from_blocks(blocks)
>>> from hexrec import SrecFile
>>> file = SrecFile.from_memory(memory, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_records(*records*, *maxdatalen=None*)

Creates a file object from records.

The *records* sequence is set as the `record` attribute of the created file object.

This method creates a file object in *records role*. This means that only its *records* is internally instanced, while the *memory* requires manual or lazy instancing (i.e. either via direct call to *apply_records()*, or any other methods indirectly calling it).

Parameters

- **records** (list of *BaseRecord*) – Record sequence to set as *records*.
- **maxdatalen** (`Optional[int]`) – Maximum record *data* field size. If `None`, the maximum non-zero size of the *data* field from the *records* sequence is used. If all the *records* have zero sized *data* field, the class attribute *DEFAULT_DATALEN* is used.

Returns

BaseFile – The created file object.

Raises

ValueError – invalid *meta* values.

See also:

BaseRecord

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
3
```

get_address_max()

Maximum address within memory.

It returns the maximum address of the underlying *memory* object.

Returns

int – Maximum address.

See also:

`bytesparse.base.ImmutableMemory.endin`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_max()
458
```

get_address_min()

Minimum address within memory.

It returns the minimum address of the underlying *memory* object.

Returns
int – Minimum address.

See also:

`bytesparse.base.ImmutableMemory.start`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_min()
123
```

get_holes()

List of memory holes.

It scans the underlying *memory* and returns the list of memory holes/gaps.

Each hole is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory hole boundaries.

See also:

`bytesparse.base.ImmutableMemory.gaps()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_holes()
[(126, 456), (459, 789)]
```

get_meta()

Meta information.

It builds and returns a dictionary of *meta* information. Meta keys are taken from the *META_KEYS* class attribute.

Returns
dict – Meta information dictionary.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks, header=b'HDR\0')
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 0}
```

get_spans()

List of memory block spans.

It scans the underlying *memory* and returns the list of memory block spans/intervals.

Each span is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory block boundaries.

See also:

`bytesparse.base.ImmutableMemory.intervals()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_spans()
[(123, 126), (456, 459), (789, 791)]
```

index(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it raises `ValueError`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.

- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
int – *item* beginning address.

Raises
ValueError – *item* not found.

See also:

find `bytesparse.base.ImmutableMemory.index()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.index(b'yz')
457
>>> file.index(ord('b'))
124
>>> file.index(b'?')
Traceback (most recent call last):
    ...
ValueError: subsection not found
```

classmethod load(*path*, **args*, ***kwargs*)

Loads a file object from the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *parse()* to load a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdin.buffer` is used.
- **args** – Forwarded to *parse()*.
- **kwargs** – Forwarded to *parse()*.

Returns
BaseFile – Loaded file object.

See also:

save() *parse()* `open()` `sys.stdin.buffer`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.load('data.hex')
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

property maxdatalen: int

Maximum byte size of the data field.

This property sets the maximum byte size of the *data* field of a serialized record.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises

ValueError – Invalid maximum data length.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> file.maxdatalen
16
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212222324252627282 92A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> file.maxdatalen = 8
>>> _ = file.print()
S0030000FC
S10B0000000102030405060 7D8
S10B000808090A0B0C0D0E0F90
S10B001010111213141516 1748
S10B001818191A1B1C1D1E1F00
S10B002020212222324252627B8
S10B002828292A2B2C2D2E2F70
S10B003030313233343536 3728
```

(continues on next page)

(continued from previous page)

```
S10B003838393A3B3C3D3E3FE0
S5030008F4
S9030000FC
>>> file.maxdatalen = 0
Traceback (most recent call last):
    ...
ValueError: invalid maximum data length
```

Type
int

property memory: MutableMemory

Memory object stored by records role.

This readonly property exposes the memory object stored by the file object while in *memory role*.

If this property is accessed while the file object is not in *memory role*, it automatically activates it by an implicit call to *apply_records()*, with default arguments.

For more control activating the *memory role*, please call *apply_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *records role* (i.e. altering content of *records*) would implicitly discard *memory* via *discard_memory()*.

See also:

apply_records() *discard_memory()*

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> _ = file.write(789, b'?!')
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz'], [789, b'?!']]
```

Type
bytesparse.Memory

merge(**files*, *clear=False*)

Merges data onto the file.

It writes the provided *files* onto *self*, in the provided order. Any common address ranges are overwritten.

Any stored *records* are discarded upon return.

Parameters

- **files** (*BaseFile*) – Files to merge.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

BaseFile – *self*.

See also:

clear() discard_records() write()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = SrecFile.from_bytes(b'<<<?????>>>', offset=450)
>>> _ = file3.merge(file1, file2)
>>> file3.memory.to_blocks()
[[123, b'abc'], [450, b'<<<???xyz>>']]
```

classmethod parse(*stream*, *ignore_errors=False*, *ignore_after_termination=True*)

Parses records from a byte stream.

It executes *BaseRecord.parse()* for each line of the incoming *stream*, creating a new file object with the collected records calling *from_records()*.

Lines resulting empty by `_is_empty_line()` are just discarded.

Notes

Please refer to the actual implementation of each record file *format*, because it may be more specialized.

Parameters

- **stream** (*bytes IO or buffer*) – Stream or byte buffer to parse records from.
- **ignore_errors** (*bool*) – Ignore `Exception` raised by *BaseRecord.parse()*.
- **ignore_after_termination** (*bool*) – Ignore anything after the termination record was parsed, if supported (e.g. *End Of File* or *start address* record, depending on the specific file *format*).

Returns

BaseFile – *self*.

See also:

parse() BaseRecord.parse() from_records() `_is_empty_line()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> buffer = b'''
...     :03DA7A0061626383
...     :040000050000CAFE2F
...     :00000001FF
... '''
>>> import io
>>> stream = io.BytesIO(buffer)
>>> file = IhexFile.parse(stream)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
>>> file = IhexFile.parse(buffer)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

print(**args*, *stream=None*, *color=False*, *start=None*, *stop=None*, ***kwargs*)

Prints record content to stdout.

This helper method prints each record of *records* via *BaseRecord.print()*. As such, it also supports colored tokens and streams different from *stdout*.

It is possible to print subset of the records by specifying the record index range.

> **Warning:** This method is **NOT** equivalent to *serialize()*, because it just prints each record from *records*. Please use *serialize()* for an actual serialization of the whole file.

Parameters

- **args** – Forwarded to the underlying call to `to_tokens()`.
- **stream** (*byte stream*) – Stream to print onto. If `None`, *stdout* is used.
- **color** (*bool*) – Colorize record tokens with ANSI color codes.
- **start** (*int*) – Inclusive start record index of the specified range. If `None`, start from the first record.
- **stop** (*int*) – Exclusive end record index of the specified range. If negative, look back from the last index. If `None`, print up to the last record.
- **kwargs** – Forwarded to the underlying call to `to_tokens()`.

Returns

BaseFile – *self*.

See also:

BaseRecord.print()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> _ = file.print(color=True, start=1, stop=-2)
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
```

read(*start=None*, *endex=None*, *fill=0*)

Extracts a substring.

It extracts a byte string from the specified range, filling any memory holes/gaps (without altering *memory*).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **fill** (*bytes or int*) – Byte pattern for filling.

Returns

BaseFile – *self*.

See also:

memory `bytesparse.base.MutableMemory.extract()` `bytesparse.base.MutableMemory.to_bytes()`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.read(start=124, endex=132)
b'bc\x00\x00\x00\x00xy'
>>> file.read(start=124, endex=132, fill=b'.')
b'bc....xy'
```

(continues on next page)

(continued from previous page)

```
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

property records: MutableSequence[*BaseRecord*]

Records stored by records role.

This readonly property exposes the list of records stored by the file object while in *records role*.

If this property is accessed while the file object is not in *records role*, it automatically activates it by an implicit call to *update_records()*, with default arguments.

For more control activating the *records role*, please call *update_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *memory role* (i.e. altering content of *memory*) would implicitly discard *records* via *discard_records()*.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, startaddr=789)
>>> len(file.records)
5
>>> _ = file.print()
S0030000FC
S106007B61626358
S10601C878797AC5
S5030002FA
S9030315E4
>>> _ = file.update_records(data_tag=SrecFile.Record.Tag.DATA_32)
>>> _ = file.print()
S0030000FC
S3080000007B61626356
S308000001C878797AC3
S5030002FA
S70500000315E2
```

Type
list of *BaseRecord*

save(*path*, **args*, ***kwargs*)

Saves a file object into the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *serialize()* to save a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdout.buffer` is used.
- **args** – Forwarded to *serialize()*.
- **kwargs** – Forwarded to *serialize()*.

Returns

BaseFile – *self*.

See also:

load() *serialize()* `open()` `sys.stdout.buffer`

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> _ = file.save('data.hex')
```

serialize(*stream*, **args*, ***kwargs*)

Serializes records onto a byte stream.

It executes *BaseRecord.serialize()* for each of the stored *records*.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **args** – Forwarded to *BaseRecord.serialize()* of each record.
- **kwargs** – Forwarded to *BaseRecord.serialize()* of each record.

Returns

BaseFile – *self*.

See also:

parse() *BaseRecord.serialize()*

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> import sys
>>> _ = file.serialize(sys.stdout.buffer, end=b'\n')
:03DA7A0061626383
:040000050000CAFE2F
:00000001FF
```

set_meta(*meta*, *strict=True*)

Sets meta information.

It sets the provided *kwargs* to their matching *meta* attributes, as listed by *META_KEYS*.

Parameters

- **meta** (*dict*) – Mapping of the *meta* information to set.
- **strict** (*bool*) – All the keys within *meta* must exist within *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *get_meta()*

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 0}
>>> _ = file.set_meta(dict(header=b'HDR\0', startaddr=456))
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 456}
```

shift(*offset*)

Shifts data addresses by an offset.

It shifts addresses of the underlying *memory* object data blocks by the provided *offset* amount.

Any stored *records* are discarded upon return.

Parameters

offset (*int*) – Offset to apply to the underlying data block addresses.

Returns

BaseFile – *self*.

See also:

memory *discard_records()* bytesparse.base.MutableMemory.shift()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> _ = file.shift(1000)
>>> file.memory.to_blocks()
[[1123, b'abc'], [1456, b'xyz']]
```

split(**addresses*, *meta=True*)

Splits into parts.

The provided *addresses* are sorted and used as markers to split *self* into parts.

Each part is the *copy()* of *self* within the range of that part, in *memory role* (i.e., *records* is not populated).

Parameters

- **addresses** (*int*) – Split points.
- **meta** (*bool*) – Each part inherits *meta* from *self*.

Returns

list of *BaseFile* – Parts after splitting.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'Hello, World!', offset=123)
>>> parts = file.split(128, 130)
>>> for part in parts: print(part.memory.to_blocks())
[[123, b'Hello']]
[[128, b', ']]
[[130, b'World!']]
>>> file.memory.to_blocks()
[[123, b'Hello, World!']]
```

abstract update_records()

Applies memory and meta to records.

This method processes the stored *memory* and *meta* information to generate the sequence of *records*.

This effectively converts the *memory role* into the *records role* (keeping both).

The *records* is assigned upon return. Any exceptions being raised should not alter the file object.

Returns

BaseFile – *self*.

Raises

ValueError – *memory* attribute not populated.

See also:

records *memory* *get_meta()* *apply_records()*

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> blocks = [[123, b'abc']]
>>> file = IhexFile.from_blocks(blocks, maxdatalen=16, startaddr=456)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
>>> _ = file.update_records()
>>> len(file.records)
3
>>> _ = file.print()
:03007B006162635C
:04000005000001C82E
:00000001FF
```

abstract validate_records()

Validates records.

It performs consistency checks for the underlying *records*.

Please refer to the record *format* implementation for more details.

Raises

ValueError – Invalid record sequence.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> records = [IhexFile.Record.create_data(123, b'abc')]
>>> file = IhexFile.from_records(records)
>>> file.validate_records()
Traceback (most recent call last):
    ...
ValueError: missing end of file record
```

view(*start=None*, *endex=None*)

Memory view.

It returns a `memoryview` over the specified range, which must cover a *contiguous* data region (i.e. no memory holes within).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
memoryview – View of the specified range.

Raises
ValueError – non-contiguous data within range.

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> bytes(file.view(start=456, endex=458))
b'xy'
>>> bytes(file.view())
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

write(*address*, *data*, *clear=False*)

Writes data into the file.

It writes the provided *data* into the underlying *memory* object.

Any stored *records* are discarded upon return.

Parameters

- **address** (*int*) – Address where *data* has to be written.
- **data** (*bytes or memory*) – Byte data to write.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns
BaseFile – *self*.

See also:

memory *clear()* *discard_records()* bytesparse.base.MutableMemory.write()

Examples

NOTE: These examples are provided by *BaseFile*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> _ = file.write(123, b'abc')
>>> _ = file.write(555, ord('?'))
>>> _ = file.write(1000, SrecFile.from_bytes(b'xyz', offset=456))
>>> file.memory.to_blocks()
[[123, b'abc'], [555, b'?'], [1456, b'xyz']]
```

4.1.10 BaseRecord

class hexrec.base.**BaseRecord**(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

Record.

A *record* is the basic means to transfer data across systems. It is usually a line of text containing some binary data in hexadecimal representation, or some *meta* information (e.g. *start address*, *record count*), often allocated at some *address* into the target system.

Most *formats* also contain information for *consistency checks*, such as the amount of data/characters within the record itself (*count*), and their *checksum*.

The `BaseRecord` class should provide a very generic description of a common record (see *Attributes*). Wherever the *format* requires some additional special information, a child class can provide it.

The *constructor* (`__init__()`) allows direct assignment of attribute values, as well as skipping *validation*. This is considered an advanced feature, typically leveraged for testing or experimental purposes.

Instead, a `BaseRecord` child class should provide factory methods to create records of each specific *nature* (e.g. the mandatory `create_data()` for *data* records).

Variables

- **tag** (*BaseTag*) – The mandatory *tag*, indicating the *nature* of the record.
- **address** (*int*) – The *address* usually tells the position in memory where the provided *data* must be stored. Some *formats* use this attribute to store other *meta* information, such as the *start address* of some program, or the *record count*.
- **data** (*bytes*) – This attribute is most commonly used to store some chunk of binary data to be allocated at the specified *address*. Some *formats* might store some *meta* information within the *data* field of the serialized record, such as the *header string*.
- **count** (*int*) – Most *formats* indicate the number of bytes or characters within the serialized record itself. Some might store other counting information, like the number of characters for the serialized *address* field.
- **checksum** (*int*) – Most *formats* provide some kind of *checksum* to check for consistency of the serialized record itself.
- **before** (*bytes*) – Some *formats* allow to serialize some data before the canonical syntax, like comments or custom/experimental data. This is a non-standard feature; please leave empty if in doubt.
- **after** (*bytes*) – Some *formats* allow to serialize some data after the canonical syntax, like comments or custom/experimental data. This is a non-standard feature; please leave empty if in doubt.
- **coords** (*int couple*) – Some *parsers* may use this attribute to store the coordinates of the parsed record, such as the line number, or the byte offset. This is a non-standard feature, useful for debug only.

Parameters

- **tag** (*BaseTag*) – See `tag` attribute.
- **address** (*int*) – See `address` attribute.
- **data** (*bytes*) – See `data` attribute.
- **count** (*int*) – See `count` attribute. `Ellipsis` initializes `count` via `compute_count()`. `None` assigns `None`, skipping further validation.

- **checksum** (*int*) – See `checksum` attribute. `Ellipsis` initializes `checksum` via *compute_checksum()*. `None` assigns `None`, skipping further validation.
- **before** (*bytes*) – See `before` attribute.
- **after** (*bytes*) – See `after` attribute.
- **coords** (*int couple*) – See `coords` attribute.
- **validate** (*bool*) – If true, *validate()* is called upon initialization.

Attributes

EQUALITY_KEYS	Meta keys for equality checks.
META_KEYS	Meta keys.
Tag	Tag object type.

Methods

__init__	
compute_checksum	Computes the checksum field value.
compute_count	Compute the count field value.
copy	Shallow copy.
create_data	Creates a data record.
data_to_int	Interprets data bytes as integer.
get_meta	Gets meta information.
parse	Parses a record from bytes.
print	Prints a record.
serialize	Serializes onto a stream.
to_bytestr	Converts into a byte string.
to_tokens	Converts into byte string tokens.
update_checksum	Updates the checksum field.
update_count	Updates the count field.
validate	Validates consistency of attribute values.

EQUALITY_KEYS: Sequence[str] = ['address', 'checksum', 'count', 'data', 'tag']

Meta keys for equality checks.

Equality methods (*__eq__()* and *__ne__()*) check against these *meta* keys only. Any other *meta* keys are just ignored.

META_KEYS: Sequence[str] = ['address', 'after', 'before', 'checksum', 'coords', 'count', 'data', 'tag']

Meta keys.

This sequence holds the *meta* keys for copying (see *copy()*).

Tag: Type[*BaseTag*] = None

Tag object type.

This class attribute indicates the *BaseTag* class used by this *BaseRecord* class.

__bytes__()

Serializes the record into bytes.

Returns
bytes – Byte serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> bytes(record)
b':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> bytes(record)
b'abc'
```

__eq__(*other*)

Equality test.

This method returns true if *self* is considered equal to *other*.

As inequality is usually easier to check, this method is usually implemented as a trivial `not self != other` (*__ne__()*).

Parameters
other (*BaseRecord*) – Record to compare to.

Returns
bool – *self* equals *other*.

See also:

__ne__()

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 == ihex2
True
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
```

(continues on next page)

(continued from previous page)

```
>>> ihex1 == ihex3
False
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 == raw
False
```

__hash__ = None

__init__(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

__ne__(*other*)

Ineuality test.

This method returns true if *self* is considered unequal to *other*.

Each attribute listed by *EQUALITY_KEYS* is compared between *self* and *other*. This method returns whether any attributes do not match.

Parameters
other (*BaseRecord*) – Record to compare to.

Returns
bool – *self* and *other* are unequal.

See also:

__eq__()

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 != ihex2
False
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 != ihex3
True
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 != raw
True
```

__repr__()

String representation.

It returns a string representation of the record content, for human understanding only.

Returns
str – String representation.

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> repr(record)
"<<class 'hexrec.formats.ihex.IhexRecord'> @...
  address:=0 after:=b'' before:=b'' checksum:=255 coords:=(-1, -1)
  count:=0 data:=b'' tag:=<IhexTag.END_OF_FILE: 1>>"
```

__str__()

Serializes the record into a string.

Returns

str – String serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> str(record)
'abc'
```

__weakref__

list of weak references to the object (if defined)

compute_checksum()

Computes the checksum field value.

It computes and returns the format-specific `checksum` value of a record.

When not specialized, it returns `None` by default.

Returns

int – Computed checksum value.

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_checksum()
215
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_checksum())
'None'
```

compute_count()

Compute the count field value.

It computes and returns the format-specific `count` value of a record.

When not specialized, it returns `None` by default.

Returns
int – Computed checksum value.

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_count()
3
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_count())
'None'
```

copy(*validate=True*)

Shallow copy.

It calls the record constructor, passing *meta* to it.

Parameters
validate (*bool*) – Performs validation on instantiation (*__init__()*).

Returns
BaseRecord – Shallow copy.

See also:

__init__() *get_meta()*

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record1 = IhexFile.Record.create_data(0x1234, b'abc')
>>> record2 = record1.copy()
>>> record1 is record2
False
>>> record1 == record2
True
```

abstract classmethod create_data(*address*, *data*)

Creates a data record.

This is a mandatory class method to instantiate a *data* record.

Parameters

- **address** (*int*) – Record address. If not supported, set zero.
- **data** (*bytes*) – Record byte data.

Returns

BaseRecord – Data record object.

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> str(record)
':0312340061626391\r\n'
```

data_to_int(*byteorder='big'*, *signed=False*)

Interprets data bytes as integer.

It creates an integer from bytes of the `data` field.

Parameters

- **byteorder** (*'big' or 'little'*) – Byte order (endianness): either `'big'` (default) or `'little'`.
- **signed** (*bool*) – Signed integer (2-complement); default false.

Returns

int – Interpreted integer value.

See also:

`int.from_bytes()`

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0xABCD)
>>> record.data
b'\xab\xcd'
>>> addrext = record.data_to_int()
>>> addrext, hex(addrext)
(43981, '0xabcd')
```

get_meta()

Gets meta information.

It returns all the object attributes whose keys are listed by *META_KEYS*.

Returns
dict – Attribute values listed by *META_KEYS*.

See also:

META_KEYS set_meta()

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> record.get_meta()
{'address': 0, 'after': b'', 'before': b'', 'checksum': 255,
 'coords': (-1, -1), 'count': 0, 'data': b'',
 'tag': <IhexTag.END_OF_FILE: 1>}
```

abstract classmethod parse(*line*, *validate=True*)

Parses a record from bytes.

Please refer to the actual implementation provided by the record *format* for more details.

Parameters

- **line** (*bytes*) – String of bytes to parse.
- **validate** (*bool*) – Perform validation checks.

Returns
BaseRecord – Parsed record.

Raises
ValueError – Syntax error.

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.parse(b':00000001FF\r\n')
>>> record.tag
<IhexTag.END_OF_FILE: 1>
>>> IhexFile.Record.parse(b'::00000001FF\r\n')
Traceback (most recent call last):
    ...
ValueError: syntax error
```

print(**args*, *stream=None*, *color=False*, ***kwargs*)

Prints a record.

The record is converted into tokens (eventually colorized) then joined and written onto a byte stream (*stdout* by default).

Parameters

- **args** – Forwarded to the underlying call to *to_tokens()*.
- **stream** (io.BytesIO) – The byte stream where the record tokens are printed. If None, *stdout* is selected.
- **color** (*bool*) – Tokens are colorized before printing.
- **kwargs** – Forwarded to the underlying call to *to_tokens()*.

Returns

BaseRecord – *self*.

See also:

to_tokens() *colorize_tokens()* io.BytesIO

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> _ = record.print()
:0312340061626391
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.print(stream=stream, color=True)
>>> stream.getvalue()
b'\x1b[0m\x1b[33m:\x1b[34m03\x1b[31m1234\x1b[32m00\x1b[36m61\x1b[96m62\
↪x1b[36m63\x1b[35m91\x1b[0m\r\n\x1b[0m'
```

serialize(*stream*, **args*, ***kwargs*)

Serializes onto a stream.

This wraps a call to *to_bytestr()* and stream.write.

Parameters

- **stream** (io.BytesIO) – Stream to write.
- **args** – Forwarded to *to_bytestr()*.
- **kwargs** – Forwarded to *to_bytestr()*.

Returns

BaseRecord – *self*.

See also:

to_bytestr() io.BytesIO

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.serialize(stream, end=b'\n')
>>> stream.getvalue()
b':0312340061626391\n'
```

abstract to_bytestr(**args*, ***kwargs*)

Converts into a byte string.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Byte string representation.

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_bytestr(end=b'\n')
b':0312340061626391\n'
```

abstract to_tokens(**args*, ***kwargs*)

Converts into byte string tokens.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns
bytes – Mapping of token keys to token byte strings.

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_tokens(end=b'\n')
{'before': b'', 'begin': b':', 'count': b'03', 'address': b'1234',
 'tag': b'00', 'data': b'616263', 'checksum': b'91', 'after': b'',
 'end': b'\n'}
```

update_checksum()

Updates the checksum field.

It updates the `checksum` attribute, assigning to it the value returned by *compute_checksum()*.

Returns
BaseRecord – *self*.

See also:

`checksum` *compute_checksum()*

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> record = IhexRecord(IhexRecord.Tag.END_OF_FILE, checksum=None)
>>> record.compute_checksum()
255
>>> record.checksum is None
True
>>> _ = record.update_checksum()
>>> record.checksum
255
```

update_count()

Updates the count field.

It updates the `count` attribute, assigning to it the value returned by *compute_count()*.

Returns
BaseRecord – *self*.

See also:

`count` *compute_count()*

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> Record = IhexFile.Record
>>> Tag = Record.Tag
>>> record = Record(Tag.DATA, data=b'abc', count=None, checksum=None)
>>> record.compute_count()
3
>>> record.count is None
True
>>> _ = record.update_count()
>>> record.count
3
```

validate(*checksum=True*, *count=True*)

Validates consistency of attribute values.

All the record attributes are checked for consistency.

Please refer to the implementation for more details.

Parameters

- **checksum** (*bool*) – Check the consistency of the `checksum` attribute.
- **count** (*bool*) – Check the consistency of the `count` attribute.

Returns

BaseRecord – *self*.

Raises

ValueError – Some targeted attributes are inconsistent.

Examples

NOTE: These examples are provided by *BaseRecord*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> _ = record.validate()
>>> record.data = b'abc'
>>> _ = record.update_count().update_checksum().validate()
Traceback (most recent call last):
    ...
ValueError: unexpcted data
```

4.1.11 BaseTag

class hexrec.base.**BaseTag**

Record tag.

The *record tag* indicates the *nature* of a record. The record tag class usually enumerates all the possible natures of a record within a *record file format*.

The tag is commonly (but not necessarily) an integer, directly written into the *serialized* representation of a record.

Methods

__init__	
is_data	Tells whether this is a data record tag.
is_file_termination	Tells whether this is record tag terminates a record file.

_DATA: Optional[*BaseTag*] = None

Alias to a common data record tag.

This tag is used internally to build a generic data record.

__weakref__

list of weak references to the object (if defined)

abstract is_data()

Tells whether this is a data record tag.

This method returns true if this data record is used for records containing plain data (i.e. without special meaning for the record file format).

Returns
bool – This is a data record tag.

Examples

NOTE: These examples are provided by *BaseTag*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(123, b'abc')
>>> record.tag.is_data()
True
>>> record = IhexFile.Record.create_end_of_file()
>>> record.tag.is_data()
False
```

```
>>> from hexrec import SrecFile
>>> record = SrecFile.Record.create_data(123, b'abc')
>>> record.tag.is_data()
True
```

(continues on next page)

(continued from previous page)

```
>>> record = SrecFile.Record.create_header(b'HDR\0')
>>> record.tag.is_data()
False
```

is_file_termination()

Tells whether this is record tag terminates a record file.

This method returns true if this record is used to terminate a record file.

This is usually the case for *End Of File* or *start address* records, depending on the specific file *format*, if supported.

Returns

bool – This is a file termination tag.

Examples

NOTE: These examples are provided by *BaseTag*. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(123, b'abc')
>>> record.tag.is_file_termination()
False
>>> record = IhexFile.Record.create_end_of_file()
>>> record.tag.is_file_termination()
True
```

```
>>> from hexrec import SrecFile
>>> record = SrecFile.Record.create_data(123, b'abc')
>>> record.tag.is_file_termination()
False
>>> record = SrecFile.Record.create_start()
>>> record.tag.is_file_termination()
True
```

4.2 formats

Record formats.

This is a collection of commonly used hexadecimal record file *formats*.

- *hexrec.formats.asciihex*
- *hexrec.formats.avr*
- *hexrec.formats.ihex*
- *hexrec.formats.mos*
- *hexrec.formats.raw*
- *hexrec.formats.sqtp*
- *hexrec.formats.srec*

- *hexrec.formats.titxt*
- *hexrec.formats.xtek*

Please refer to each submodule for more details.

4.3 asciihex

ASCII-hex format.

See also:

https://srecord.sourceforge.net/man/man5/srec_ascii_hex.5.html

Classes

AsciiHexFile	ASCII-HEX file object.
AsciiHexRecord	ASCII-HEX record object.
AsciiHexTag	ASCII-HEX tag.

4.3.1 AsciiHexFile

class hexrec.formats.asciihex.**AsciiHexFile**

ASCII-HEX file object.

Attributes

DEFAULT_DATALEN	Default data attribute length.
FILE_EXT	Supported filename extensions.
META_KEYS	Meta information key names.
maxdatalen	Maximum byte size of the data field.
memory	Memory object stored by records role.
records	Records stored by records role.

Methods

__init__	
align	Pads blocks to align their boundaries.
append	Appends a byte.
apply_records	Applies records to memory and meta.
clear	Clears data within a range.
convert	Converts a file object to another format.
copy	Copies within a range.
crop	Clears data outside a range.
cut	Cuts data within a range.

continues on next page

Table 2 – continued from previous page

delete	Deletes data within a range.
discard_memory	Discards underlying memory.
discard_records	Discards underlying records.
extend	Concatenates data.
fill	Fills a range.
find	Finds a substring.
flood	Floods a range.
from_blocks	Creates a file object from a memory object.
from_bytes	Creates a file object from a byte string.
from_memory	Creates a file object from a memory object.
from_records	Creates a file object from records.
get_address_max	Maximum address within memory.
get_address_min	Minimum address within memory.
get_holes	List of memory holes.
get_meta	Meta information.
get_spans	List of memory block spans.
index	Finds a substring.
load	Loads a file object from the filesystem.
merge	Merges data onto the file.
parse	Parses records from a byte stream.
print	Prints record content to stdout.
read	Extracts a substring.
save	Saves a file object into the filesystem.
serialize	Serializes records onto a byte stream.
set_meta	Sets meta information.
shift	Shifts data addresses by an offset.
split	Splits into parts.
update_records	Applies memory and meta to records.
validate_records	Validates records.
view	Memory view.
write	Writes data into the file.

DEFAULT_DATALEN: int = 16

Default data attribute length.

Default value for the *maxdatalen* *meta*, which sets the maximum size of `BaseRecord.data` field values.

FILE_EXT: Sequence[str] = []

Supported filename extensions.

Sequence of file name extension substrings (e.g. `.hex`). This list is used by functions like `guess_format_name()` to manage mapping of file *formats*.

META_KEYS: Sequence[str] = ['maxdatalen']

Meta information key names.

Sequence of key strings listing the supported *meta* information of this file *format*.

Record

alias of *AsciiHexRecord*

__add__(*other*)

Concatenates with another file.

Equivalent to *copy()* then *extend()*.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – Concatenation of *self* and *other*.

See also:

copy() extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 + file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file4 = file3 + b'789'
>>> file4.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__bool__()

bool: Has data records or memory.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> bool(file)
False
>>> _ = file.append(0)
>>> bool(file)
True
```

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> file = IhexFile.from_records([IhexRecord.create_end_of_file()])
>>> bool(file)
False
>>> file.records.insert(0, IhexRecord.create_data(0, b'\0'))
>>> bool(file)
True
```

__delitem__(*key*)

Deletes a range.

Parameters
key (*slice or int*) – Range to delete.

See also:

`bytesparse.base.MutableMemory.__delitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> del file[457]
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xz']]
>>> del file[125:457]
>>> file.memory.to_blocks()
[[123, b'abz']]
```

__eq__(*other*)

Equality test.

The file objects *self* and *other* are considered *equal* if the inequality tests of *__ne__()* result false.

Returns
bool – *self* and *other* are *equal*.

See Also
__ne__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 == file2
True
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 == file3
False
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 == file4
False
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
```

(continues on next page)

(continued from previous page)

```
>>> srec_file == ihex_file
False
>>> srec_file.memory == ihex_file.memory
True
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__getitem__(*key*)

Extracts a range.

Parameters
key (*slice or int*) – Range to extract.

Raises
ValueError – invalid range.

See also:

bytesparse.base.ImmutableMemory.__getitem__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> chr(file[457])
'y'
>>> repr(file[333])
'None'
>>> file[123:125]
b'ab'
>>> file[125:457]
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

__hash__ = None

__iadd__(*other*)

Concatenates data.

Equivalent to *extend()*.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – *self*.

See also:

memory *extend()* *discard_records()* bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 += file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file1 += b'789'
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__init__()

__ior__(*other*)

Merges with another file.

Equivalent to *merge()*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self*.

See also:

merge() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 |= file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file1 |= b'789'
>>> file1.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__ne__(*other*)

Inequality test.

The file objects *self* and *other* are considered *unequal* if any of the following tests result true:

- Both have *memory role* (i.e. *memory*), resulting unequal;
- Both have *records role* (i.e. *records*), resulting unequal;

- *other* does not have a *meta* listed by *META_KEYS*;
- A *meta* value (among those of *META_KEYS*) is different.

Returns
bool – *self* and *other* are *unequal*.

See also:

__eq__() memory records META_KEYS

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 != file2
False
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 != file3
True
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 != file4
True
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file != ihex_file
True
>>> srec_file.memory != ihex_file.memory
False
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__or__(*other*)

Merges with another file.

Equivalent to *copy()* then *merge()*.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self* merged with *other*.

See also:

copy() merge()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 | file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file4 = file3 | b'789'
>>> file4.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__setitem__(*key*, *value*)

Sets a range.

Parameters

- **key** (*slice or int*) – Range to set.
- **value** (bytes, `bytesparse.base.ImmutableMemory`, `None`) – Value(s) to set. `None` acts like *clear()*.

Raises

ValueError – invalid range.

See also:

`bytesparse.base.MutableMemory.__setitem__()` *clear()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file[124] = b'?'
>>> file.memory.to_blocks()
[[123, b'a?c'], [456, b'xyz']]
>>> file[:125] = None
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'xyz']]
>>> file[457:458] = b'789'
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'x789z']]
```

__weakref__

list of weak references to the object (if defined)

classmethod _is_line_empty(*line*)

Empty line check.

Tells whether a *line* has no meaningful content (e.g. all whitespace). The check itself depends on the implementing file *format*. It may be used internally to skip empty lines, e.g. by *parse()*.

Parameters
line (*bytes*) – A line, byte string.

Returns
bool: The *line* is empty.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexFile._is_line_empty(b'')
True
>>> IhexFile._is_line_empty(b' \t\v\r\n')
True
>>> IhexFile._is_line_empty(b':00000001FF\r\n')
False
```

align(*modulo*, *start=None*, *endex=None*, *pattern=0*)

Pads blocks to align their boundaries.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*, so that memory blocks are aligned to the required *modulo*.

Any stored *records* are discarded upon return.

Parameters

- **modulo** (*int*) – Alignment modulo.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns
`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.align()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [134, b'xyz']])
>>> _ = file.align(4, pattern=b'.')
>>> file.memory.to_blocks()
[[120, b'...abc..'], [132, b'..xyz...']]
```

append(*item*)

Appends a byte.

It appends the *item* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
item (*byte or int*) – Byte to append.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.append()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123)
>>> _ = file.append(b'.')
>>> _ = file.append(0)
>>> file.memory.to_blocks()
[[123, b'abc.\x00']]
```

apply_records()

Applies records to memory and meta.

This method processes the stored *records*, converting *data* as *memory*, and special records into their *meta* counterparts.

This effectively converts the *records role* into the *memory role* (keeping both).

The *memory* and *meta* are assigned upon return. Any exceptions being raised should not alter the file object.

Returns
BaseFile – *self*.

Raises
ValueError – *records* attribute not populated.

See also:

records memory get_meta() update_records()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_start_linear_address(456),
```

(continues on next page)

(continued from previous page)

```
...             IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records, maxdatalen=16)
>>> _ = file.apply_records()
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
```

clear(*start=None*, *endex=None*)

Clears data within a range.

It clears the specified range of underlying *memory* object, making a memory hole.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.clear()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.clear(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

classmethod convert(*source*, *meta=True*)

Converts a file object to another format.

It copies the *memory* and *meta* of the *source* file object, creating a new one of the target `BaseFile` format type.

Parameters

- **source** (`BaseFile`) – Source file object to convert.
- **meta** (*bool*) – Copy *meta* information to the target file object. Only the keys of the target *META_KEYS* are processed.

Returns

`BaseFile` – Converted copy of *source* to the target format.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> source = IhexFile.from_blocks(blocks, startaddr=789)
>>> target = SrecFile.convert(source)
>>> target.memory is source.memory
False
>>> target.memory == source.memory
True
>>> target.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 789}
```

copy(*start=None*, *endex=None*, *meta=True*)

Copies within a range.

It copied data within the specified range of the file object, creating a new one carrying the inner slice.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

`BaseFile` – *self*.

See also:

memory *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.copy(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

crop(*start=None*, *endex=None*)

Clears data outside a range.

It clears outside the specified range of underlying *memory* object, timming it.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.crop()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.crop(start=124, endex=132)
>>> file.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
```

cut(*start=None*, *endex=None*, *meta=False*)

Cuts data within a range.

It takes data within the specified range away from the file object, creating a new one carrying the inner slice. The inner slice is cleared from *self*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns
`BaseFile` – *self*.

See also:

memory *clear()* *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.cut(start=124, endex=132)
>>> inner.memory.to_blocks()
```

(continues on next page)

(continued from previous page)

```
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

delete(*start=None*, *endex=None*)

Deletes data within a range.

It deletes the specified range of underlying *memory* object, shifting all subsequent data towards the collapsed range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.delete()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.delete(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'az']]
```

discard_memory()

Discards underlying memory.

The underlying *memory* object is assigned `None`.

If the underlying *records* object is `None`, it is assigned a new empty memory object.

Returns

`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_bytes(b'abc', offset=123)
>>> _ = file.update_records()
>>> _ = file.discard_memory()
>>> _ = file.update_records()
Traceback (most recent call last):
    ...
ValueError: memory instance required
```

discard_records()

Discards underlying records.

The underlying *records* object is assigned `None`.

If the underlying *memory* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> _ = file.validate_records()
>>> _ = file.discard_records()
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: records required
```

extend(*other*)

Concatenates data.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.extend()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> _ = file1.extend(file2)
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> _ = file1.extend(b'789')
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

fill(*start=None*, *endex=None*, *pattern=0*)

Fills a range.

It writes a *pattern* of bytes onto the underlying *memory* object, overwriting anything within the specified range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.fill()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.fill(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'a.......z']]
```

find(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it returns `-1`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address; `-1` if not found.

See also:

index `bytesparse.base.ImmutableMemory.find()`

Notes

The internal *memory* might allow negative addresses for its stored data. In that case, *index()* would be more appropriate, because it raises an exception when the *item* is not found.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.find(b'yz')
457
>>> file.find(ord('b'))
124
>>> file.find(b'?')
-1
```

flood(*start=None*, *endex=None*, *pattern=0*)

Floods a range.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.flood()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.get_holes()
[(126, 130)]
>>> _ = file.flood(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'abc....xyz']]
```

classmethod from_blocks(*blocks*, ***meta*)

Creates a file object from a memory object.

The *blocks* are put into the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **blocks** (*list of blocks*) – Memory blocks to put into *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

`BaseFile` – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* `bytesparse.base.ImmutableMemory.from_blocks()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_bytes(*data*, *offset=0*, ***meta*)

Creates a file object from a byte string.

The byte string makes a single *data* block, placed at some offset within the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **data** (*bytes*) – A byte string used to make a single data block.
- **offset** (*int*) – Offset of the single data block within *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_bytes()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
8
```

classmethod **from_memory**(*memory=None*, ***meta*)

Creates a file object from a memory object.

The *memory* is set as the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **memory** (bytesparse.base.MutableMemory) – Memory object to set as *memory*. If None, an empty memory object is automatically created.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS bytesparse.base.MutableMemory

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from bytesparse import Memory
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> memory = Memory.from_blocks(blocks)
>>> from hexrec import SrecFile
>>> file = SrecFile.from_memory(memory, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_records(*records*, *maxdatalen=None*)

Creates a file object from records.

The *records* sequence is set as the `record` attribute of the created file object.

This method creates a file object in *records role*. This means that only its *records* is internally instanced, while the *memory* requires manual or lazy instancing (i.e. either via direct call to *apply_records()*, or any other methods indirectly calling it).

Parameters

- **records** (list of `BaseRecord`) – Record sequence to set as *records*.
- **maxdatalen** (`Optional[int]`) – Maximum record *data* field size. If `None`, the maximum non-zero size of the *data* field from the *records* sequence is used. If all the *records* have zero sized *data* field, the class attribute *DEFAULT_DATALEN* is used.

Returns

`BaseFile` – The created file object.

Raises

ValueError – invalid *meta* values.

See also:

`BaseRecord`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
3
```

get_address_max()

Maximum address within memory.

It returns the maximum address of the underlying *memory* object.

Returns
int – Maximum address.

See also:

`bytesparse.base.ImmutableMemory.endin`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_max()
458
```

get_address_min()

Minimum address within memory.

It returns the minimum address of the underlying *memory* object.

Returns
int – Minimum address.

See also:

`bytesparse.base.ImmutableMemory.start`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_min()
123
```

get_holes()

List of memory holes.

It scans the underlying *memory* and returns the list of memory holes/gaps.

Each hole is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory hole boundaries.

See also:

`bytesparse.base.ImmutableMemory.gaps()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_holes()
[(126, 456), (459, 789)]
```

get_meta()

Meta information.

It builds and returns a dictionary of *meta* information. Meta keys are taken from the *META_KEYS* class attribute.

Returns
dict – Meta information dictionary.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks, header=b'HDR\0')
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 0}
```

get_spans()

List of memory block spans.

It scans the underlying *memory* and returns the list of memory block spans/intervals.

Each span is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory block boundaries.

See also:

`bytesparse.base.ImmutableMemory.intervals()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_spans()
[(123, 126), (456, 459), (789, 791)]
```

index(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it raises `ValueError`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address.

Raises

ValueError – *item* not found.

See also:

find `bytesparse.base.ImmutableMemory.index()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.index(b'yz')
457
>>> file.index(ord('b'))
124
>>> file.index(b'?')
Traceback (most recent call last):
    ...
ValueError: subsection not found
```

classmethod load(*path*, **args*, ***kwargs*)

Loads a file object from the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *parse()* to load a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdin.buffer` is used.
- **args** – Forwarded to *parse()*.
- **kwargs** – Forwarded to *parse()*.

Returns

`BaseFile` – Loaded file object.

See also:

save() *parse()* `open()` `sys.stdin.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.load('data.hex')
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

property maxdatalen: int

Maximum byte size of the data field.

This property sets the maximum byte size of the *data* field of a serialized record.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises

ValueError – Invalid maximum data length.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> file.maxdatalen
16
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> file.maxdatalen = 8
>>> _ = file.print()
S0030000FC
S10B0000000102030405060 7D8
S10B000808090A0B0C0D0E0F90
S10B001010111213141516 1748
S10B001818191A1B1C1D1E1F00
S10B002020212223242526 27B8
S10B002828292A2B2C2D2E2F70
S10B003030313233343536 3728
```

(continues on next page)

(continued from previous page)

```
S10B003838393A3B3C3D3E3FE0
S5030008F4
S9030000FC
>>> file.maxdatalen = 0
Traceback (most recent call last):
    ...
ValueError: invalid maximum data length
```

Type
int

property memory: MutableMemory

Memory object stored by records role.

This readonly property exposes the memory object stored by the file object while in *memory role*.

If this property is accessed while the file object is not in *memory role*, it automatically activates it by an implicit call to *apply_records()*, with default arguments.

For more control activating the *memory role*, please call *apply_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *records role* (i.e. altering content of *records*) would implicitly discard *memory* via *discard_memory()*.

See also:

apply_records() *discard_memory()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> _ = file.write(789, b'?!')
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz'], [789, b'?!']]
```

Type
`bytesparse.Memory`

merge(**files*, *clear=False*)

Merges data onto the file.

It writes the provided *files* onto *self*, in the provided order. Any common address ranges are overwritten.

Any stored *records* are discarded upon return.

Parameters

- **files** (BaseFile) – Files to merge.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

BaseFile – *self*.

See also:

clear() discard_records() write()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = SrecFile.from_bytes(b'<<<?????>>>', offset=450)
>>> _ = file3.merge(file1, file2)
>>> file3.memory.to_blocks()
[[123, b'abc'], [450, b'<<<???xyz>>']]
```

classmethod parse(*stream*, *ignore_errors=False*, *stxetx=True*)

Parses records from a byte stream.

It executes *AsciiHexRecord.parse()* for each line of the incoming *stream*, creating a new file object with the collected records calling *from_records()*.

Lines resulting empty by _is_empty_line() are just discarded.

Notes

Please refer to the actual implementation of each record file *format*, because it may be more specialized.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **ignore_errors** (*bool*) – Ignore Exception raised by *AsciiHexRecord.parse()*.
- **stxetx** (*bool*) – Require record data be enclosed within ASCII STX and ETX bytes.

Returns

AsciiHexFile – *self*.

See also:

parse() AsciiHexRecord.parse() from_records() _is_empty_line()

Examples

```
>>> from hexrec import AsciiHexFile
>>> buffer = b'''
...     \x02
...     $A1234,
...     61 62 63
...     \x03
... '''
>>> import io
>>> stream = io.BytesIO(buffer)
>>> file = AsciiHexFile.parse(stream)
>>> file.memory.to_blocks()
[[4660, b'abc']]
>>> file.get_meta()
{'maxdatalen': 3}
```

print(**args*, *stream=None*, *color=False*, *start=None*, *stop=None*, ***kwargs*)

Prints record content to stdout.

This helper method prints each record of *records* via `BaseRecord.print()`. As such, it also supports colored tokens and streams different from *stdout*.

It is possible to print subset of the records by specifying the record index range.

Warning: This method is **NOT** equivalent to *serialize()*, because it just prints each record from *records*. Please use *serialize()* for an actual serialization of the whole file.

Parameters

- **args** – Forwarded to the underlying call to `to_tokens()`.
- **stream** (*byte stream*) – Stream to print onto. If `None`, *stdout* is used.
- **color** (*bool*) – Colorize record tokens with ANSI color codes.
- **start** (*int*) – Inclusive start record index of the specified range. If `None`, start from the first record.
- **stop** (*int*) – Exclusive end record index of the specified range. If negative, look back from the last index. If `None`, print up to the last record.
- **kwargs** – Forwarded to the underlying call to `to_tokens()`.

Returns

`BaseFile` – *self*.

See also:

`BaseRecord.print()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> _ = file.print(color=True, start=1, stop=-2)
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
```

read(*start=None*, *endex=None*, *fill=0*)

Extracts a substring.

It extracts a byte string from the specified range, filling any memory holes/gaps (without altering *memory*).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **fill** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory `bytesparse.base.MutableMemory.extract()` `bytesparse.base.MutableMemory.to_bytes()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.read(start=124, endex=132)
b'bc\x00\x00\x00\x00xy'
>>> file.read(start=124, endex=132, fill=b'.')
b'bc....xy'
```

(continues on next page)

(continued from previous page)

```
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

property records: MutableSequence[*BaseRecord*]

Records stored by records role.

This readonly property exposes the list of records stored by the file object while in *records role*.

If this property is accessed while the file object is not in *records role*, it automatically activates it by an implicit call to *update_records()*, with default arguments.

For more control activating the *records role*, please call *update_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *memory role* (i.e. altering content of *memory*) would implicitly discard *records* via *discard_records()*.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, startaddr=789)
>>> len(file.records)
5
>>> _ = file.print()
S0030000FC
S106007B61626358
S10601C878797AC5
S5030002FA
S9030315E4
>>> _ = file.update_records(data_tag=SrecFile.Record.Tag.DATA_32)
>>> _ = file.print()
S0030000FC
S3080000007B61626356
S308000001C878797AC3
S5030002FA
S70500000315E2
```

Type
list of `BaseRecord`

save(*path*, **args*, ***kwargs*)

Saves a file object into the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *serialize()* to save a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdout.buffer` is used.
- **args** – Forwarded to *serialize()*.
- **kwargs** – Forwarded to *serialize()*.

Returns

`BaseFile` – *self*.

See also:

load() *serialize()* `open()` `sys.stdout.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> _ = file.save('data.hex')
```

serialize(*stream*, *execharb' '*, *exelast=True*, *dollarend=b','*, *end=b'\\r\\n'*, *stxetx=True*)

Serializes records onto a byte stream.

It executes `MosRecord.serialize()` for each of the stored *records*.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **execharb** (*byte*) – *Execution character* value.
- **exelast** (*bool*) – Append *execution character* also to the last byte of the serialized record.
- **dollarend** (*byte*) – End character of *dollar* records (i.e. *address* and *checksum* records).
- **end** (*bytes*) – End of record termination bytes.
- **stxetx** (*bool*) – Enclose the whole serialized file within ASCII `STX` and `ETX` bytes.

Returns

`MosFile` – *self*.

See also:

parse() `MosRecord.serialize()`

Examples

```
>>> from hexrec import MosFile
>>> file = MosFile.from_blocks([[0xDA7A, b'abc']])
>>> import sys
>>> _ = file.serialize(sys.stdout.buffer, nuls=False, xoff=False)
;03DA7A616263027D
;0000010001
```

set_meta(*meta*, *strict=True*)

Sets meta information.

It sets the provided *kwargs* to their matching *meta* attributes, as listed by *META_KEYS*.

Parameters

- **meta** (*dict*) – Mapping of the *meta* information to set.
- **strict** (*bool*) – All the keys within *meta* must exist within *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *get_meta()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 0}
>>> _ = file.set_meta(dict(header=b'HDR\0', startaddr=456))
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 456}
```

shift(*offset*)

Shifts data addresses by an offset.

It shifts addresses of the underlying *memory* object data blocks by the provided *offset* amount.

Any stored *records* are discarded upon return.

Parameters

offset (*int*) – Offset to apply to the underlying data block addresses.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.shift()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> _ = file.shift(1000)
>>> file.memory.to_blocks()
[[1123, b'abc'], [1456, b'xyz']]
```

split(**addresses*, *meta=True*)

Splits into parts.

The provided *addresses* are sorted and used as markers to split *self* into parts.

Each part is the *copy()* of *self* within the range of that part, in *memory role* (i.e., *records* is not populated).

Parameters

- **addresses** (*int*) – Split points.
- **meta** (*bool*) – Each part inherits *meta* from *self*.

Returns

list of `BaseFile` – Parts after splitting.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'Hello, World!', offset=123)
>>> parts = file.split(128, 130)
>>> for part in parts: print(part.memory.to_blocks())
[[123, b'Hello']]
[[128, b', ']]
[[130, b'World!']]
>>> file.memory.to_blocks()
[[123, b'Hello, World!']]
```

update_records(*align=False*, *checksum=False*, *addrlen=8*)

Applies memory and meta to records.

This method processes the stored *memory* and *meta* information to generate the sequence of *records*.

This effectively converts the *memory role* into the *records role* (keeping both).

The *records* is assigned upon return. Any exceptions being raised should not alter the file object.

Parameters

- **align** (*bool*) – Aligns data record chunk address bounds to *maxdatalen*.
- **checksum** (*bool*) – Generate a final *checksum* record.
- **addrlen** (*int*) – Address length, in *nibbles* (4-bit units).

Returns
AsciiHexFile – *self*.

Raises
ValueError – *memory* attribute not populated.

See also:

records *memory* *get_meta()* *apply_records()*

Examples

```
>>> from hexrec import AsciiHexFile
>>> blocks = [[123, b'abc']]
>>> file = AsciiHexFile.from_blocks(blocks, maxdatalen=16)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'maxdatalen': 16}
>>> _ = file.update_records()
>>> len(file.records)
2
>>> _ = file.print(exelast=False)
$A0000007B,
61 62 63
```

validate_records(*data_ordering=False*, *checksum_values=True*)

Validates records.

It performs consistency checks for the underlying *records*.

Parameters

- **data_ordering** (*bool*) – Checks that the *data* record sequence has monotonically increasing addresses, without any overlapping.
- **checksum_values** (*bool*) – Checks for valid *checksum* values.

Returns
AsciiHexFile – *self*.

Raises
ValueError – Invalid record sequence.

Examples

```
>>> from hexrec import AsciiHexFile
>>> records = [AsciiHexFile.Record.create_data(123, b'abc'),
...            AsciiHexFile.Record.create_checksum(0xFFFF)]
>>> file = AsciiHexFile.from_records(records)
>>> file.validate_records()
Traceback (most recent call last):
    ...
ValueError: wrong checksum
```

view(*start=None*, *endex=None*)

Memory view.

It returns a `memoryview` over the specified range, which must cover a *contiguous* data region (i.e. no memory holes within).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

memoryview – View of the specified range.

Raises

ValueError – non-contiguous data within range.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> bytes(file.view(start=456, endex=458))
b'xy'
>>> bytes(file.view())
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

write(*address*, *data*, *clear=False*)

Writes data into the file.

It writes the provided *data* into the underlying *memory* object.

Any stored *records* are discarded upon return.

Parameters

- **address** (*int*) – Address where *data* has to be written.
- **data** (*bytes or memory*) – Byte data to write.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

`BaseFile` – *self*.

See also:

memory *clear()* *discard_records()* `bytesparse.base.MutableMemory.write()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> _ = file.write(123, b'abc')
>>> _ = file.write(555, ord('?'))
>>> _ = file.write(1000, SrecFile.from_bytes(b'xyz', offset=456))
>>> file.memory.to_blocks()
[[123, b'abc'], [555, b'?'], [1456, b'xyz']]
```

4.3.2 AsciiHexRecord

class hexrec.formats.asciihex.**AsciiHexRecord**(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

ASCII-HEX record object.

Attributes

DATA_EXECHARS	Supported execution characters.
EQUALITY_KEYS	Meta keys for equality checks.
LINE_REGEX	Line parser regex.
META_KEYS	Meta keys.

Methods

__init__	
compute_checksum	Computes the checksum field value.
compute_count	Compute the count field value.
copy	Shallow copy.
create_address	Creates an address record.
create_checksum	Creates a checksum record.
create_data	Creates a data record.
data_to_int	Interprets data bytes as integer.
get_meta	Gets meta information.
parse	Parses a record from bytes.
print	Prints a record.
serialize	Serializes onto a stream.
to_bytestr	Converts into a byte string.
to_tokens	Converts into byte string tokens.
update_checksum	Updates the checksum field.
update_count	Updates the count field.
validate	Validates consistency of attribute values.

DATA_EXECHARS: bytes = b" \t\x0b\x0c\r%',"

Supported execution characters.

EQUALITY_KEYS: Sequence[str] = ['address', 'checksum', 'count', 'data', 'tag']

Meta keys for equality checks.

Equality methods (*__eq__()* and *__ne__()*) check against these *meta* keys only. Any other *meta* keys are just ignored.

LINE_REGEX = re.compile(b"\\s*((?P<data>([0-9A-Fa-f]{2}[\t\x0b\x0c\r%',]?)+)|(\\ $[Aa](?P<address>[0-9A-Fa-f]+)[,.])|(\\$[Ss](?P<checksum>[0-9A-Fa-f]+)[,.]))\\s*")

Line parser regex.

META_KEYS: Sequence[str] = ['address', 'after', 'before', 'checksum', 'coords', 'count', 'data', 'tag']

Meta keys.

This sequence holds the *meta* keys for copying (see *copy()*).

Tag

alias of *AsciiHexTag*

__bytes__()

Serializes the record into bytes.

Returns
bytes – Byte serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> bytes(record)
b':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> bytes(record)
b'abc'
```

__eq__(*other*)

Equality test.

This method returns true if *self* is considered equal to *other*.

As inequality is usually easier to check, this method is usually implemented as a trivial `not self != other` (*__ne__()*).

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* equals *other*.

See also:

__ne__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 == ihex2
True
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 == ihex3
False
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 == raw
False
```

__hash__ = None

__init__(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

__ne__(*other*)

Ineuality test.

This method returns true if *self* is considered unequal to *other*.

Each attribute listed by *EQUALITY_KEYS* is compared between *self* and *other*. This method returns whether any attributes do not match.

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* and *other* are unequal.

See also:

__eq__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 != ihex2
False
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 != ihex3
True
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 != raw
True
```

__repr__()

String representation.

It returns a string representation of the record content, for human understanding only.

Returns

str – String representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> repr(record)
"<<class 'hexrec.formats.ihex.IhexRecord'> @...
  address:=0 after:=b'' before:=b'' checksum:=255 coords:=(-1, -1)
  count:=0 data:=b'' tag:=<IhexTag.END_OF_FILE: 1>>"
```

__str__()

Serializes the record into a string.

Returns

str – String serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> str(record)
'abc'
```

__weakref__

list of weak references to the object (if defined)

compute_checksum()

Computes the checksum field value.

It computes and returns the format-specific `checksum` value of a record.

When not specialized, it returns `None` by default.

Returns

int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_checksum()
215
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_checksum())
'None'
```

compute_count()

Compute the count field value.

It computes and returns the format-specific `count` value of a record.

When not specialized, it returns `None` by default.

Returns

int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_count()
3
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_count())
'None'
```

copy(*validate=True*)

Shallow copy.

It calls the record constructor, passing *meta* to it.

Parameters
validate (*bool*) – Performs validation on instantiation (*__init__()*).

Returns
`BaseRecord` – Shallow copy.

See also:

__init__() *get_meta()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record1 = IhexFile.Record.create_data(0x1234, b'abc')
>>> record2 = record1.copy()
>>> record1 is record2
False
>>> record1 == record2
True
```

classmethod create_address(*address*, *addrlen=8*)

Creates an address record.

Parameters

- **address** (*int*) – Address value.
- **addrlen** (*int*) – Address length, in *nibbles* (4-bit units).

Returns
AsciiHexRecord – Address record object.

Raises
ValueError – invalid parameter.

Examples

```
>>> from hexrec import AsciiHexFile
>>> record = AsciiHexFile.Record.create_address(0x1234, addrlen=4)
>>> str(record)
'$A1234,\r\n'
```

classmethod create_checksum(*checksum*)

Creates a checksum record.

Parameters
checksum (*int*) – 16-bit checksum value.

Returns
AsciiHexRecord – Checksum record object.

Raises
ValueError – invalid parameter.

Examples

```
>>> from hexrec import AsciiHexFile
>>> record = AsciiHexFile.Record.create_checksum(0x1234)
>>> str(record)
'$S1234,\r\n'
```

classmethod create_data(*address*, *data*)

Creates a data record.

Parameters

- **address** (*int*) – Ignored; please provide zero.
- **data** (*bytes*) – Record byte data.

Returns
AsciiHexRecord – Data record object.

Raises
ValueError – invalid parameter.

Examples

```
>>> from hexrec import AsciiHexFile
>>> record = AsciiHexFile.Record.create_data(0, b'abc')
>>> str(record)
'61 62 63 \r\n'
```

data_to_int(*byteorder='big'*, *signed=False*)

Interprets data bytes as integer.

It creates an integer from bytes of the `data` field.

Parameters

- **byteorder** (*'big' or 'little'*) – Byte order (endianness): either `'big'` (default) or `'little'`.

- **signed** (*bool*) – Signed integer (2-complement); default false.

Returns
int – Interpreted integer value.

See also:

`int.from_bytes()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0xABCD)
>>> record.data
b'\xab\xcd'
>>> addrext = record.data_to_int()
>>> addrext, hex(addrext)
(43981, '0xabcd')
```

get_meta()

Gets meta information.

It returns all the object attributes whose keys are listed by *META_KEYS*.

Returns
dict – Attribute values listed by *META_KEYS*.

See also:

META_KEYS `set_meta()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> record.get_meta()
{'address': 0, 'after': b'', 'before': b'', 'checksum': 255,
 'coords': (-1, -1), 'count': 0, 'data': b'',
 'tag': <IhexTag.END_OF_FILE: 1>}
```

classmethod parse(*line*, *address=0*, *validate=True*)

Parses a record from bytes.

Parameters

- **line** (*bytes*) – String of bytes to parse.
- **address** (*int*) – Default record address for *data* records.
- **validate** (*bool*) – Perform validation checks.

Returns
`BaseRecord` – Parsed record.

Raises
ValueError – Syntax error.

Examples

```
>>> from hexrec import AsciiHexFile
>>> record = AsciiHexFile.Record.parse(b'$A1234,\r\n')
>>> record.tag
<AsciiHexTag.ADDRESS: 1>
>>> record = AsciiHexFile.Record.parse(b'61 62 63\r\n', address=123)
>>> record.address, record.data
(123, b'abc')
>>> AsciiHexFile.Record.parse(b'@ABCD\r\n')
Traceback (most recent call last):
    ...
ValueError: syntax error
```

print(**args*, *stream=None*, *color=False*, ***kwargs*)

Prints a record.

The record is converted into tokens (eventually colorized) then joined and written onto a byte stream (*stdout* by default).

Parameters

- **args** – Forwarded to the underlying call to *to_tokens()*.
- **stream** (io.BytesIO) – The byte stream where the record tokens are printed. If None, *stdout* is selected.
- **color** (*bool*) – Tokens are colorized before printing.
- **kwargs** – Forwarded to the underlying call to *to_tokens()*.

Returns
BaseRecord – *self*.

See also:

to_tokens() colorize_tokens() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> _ = record.print()
:0312340061626391
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.print(stream=stream, color=True)
>>> stream.getvalue()
b'\x1b[0m\x1b[33m:\x1b[34m03\x1b[31m1234\x1b[32m00\x1b[36m61\x1b[96m62\
↪x1b[36m63\x1b[35m91\x1b[0m\r\n\x1b[0m'
```

serialize(*stream*, **args*, ***kwargs*)

Serializes onto a stream.

This wraps a call to *to_bytestr()* and `stream.write`.

Parameters

- **stream** (`io.BytesIO`) – Stream to write.
- **args** – Forwarded to *to_bytestr()*.
- **kwargs** – Forwarded to *to_bytestr()*.

Returns

`BaseRecord` – *self*.

See also:

to_bytestr() `io.BytesIO`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.serialize(stream, end=b'\n')
>>> stream.getvalue()
b':0312340061626391\n'
```

to_bytestr(*exechar=b' '*, *exelast=True*, *dollarend=b','*, *end=b'\\r\\n'*)

Converts into a byte string.

Parameters

- **exechar** (*byte*) – *Execution character* value.
- **exelast** (*bool*) – Append *execution character* also to the last byte of the serialized record.
- **dollarend** (*byte*) – End character of *dollar* records (i.e. *address* and *checksum* records).
- **end** (*bytes*) – End of record termination bytes.

Returns

bytes – Byte string representation.

Examples

```
>>> from hexrec import AsciiHexFile
>>> record = AsciiHexFile.Record.create_data(0, b'abc')
>>> record.to_bytestr(exechar=b"'", exelast=False, end=b'\n')
b'61.62.63\n'
>>> record = AsciiHexFile.Record.create_address(0x1234)
>>> record.to_bytestr(dollarend=b'.')
b'$A00001234.\r\n'
```

to_tokens(*exechar=b' '*, *exelast=True*, *dollarend=b','*, *end=b'\\r\\n'*)

Converts into byte string tokens.

Parameters

- **exechar** (*byte*) – *Execution character* value.
- **exelast** (*bool*) – Append *execution character* also to the last byte of the serialized record.
- **dollarend** (*byte*) – End character of *dollar* records (i.e. *address* and *checksum* records).
- **end** (*bytes*) – End of record termination bytes.

Returns

bytes – Mapping of token keys to token byte strings.

Examples

```
>>> from hexrec import AsciiHexFile
>>> record = AsciiHexFile.Record.create_data(0, b'abc')
>>> record.to_tokens(exechar=b"'", exelast=False, end=b'\n')
{'before': b'', 'address': b'', 'data': b"61'62'63",
 'checksum': b'', 'after': b'', 'end': b'\n'}
>>> record = AsciiHexFile.Record.create_address(0x1234)
>>> record.to_tokens(dollarend=b'.')
{'before': b'', 'address': b'$A00001234.', 'data': b'',
 'checksum': b'', 'after': b'', 'end': b'\r\n'}
```

update_checksum()

Updates the checksum field.

It updates the checksum attribute, assigning to it the value returned by *compute_checksum()*.

Returns

BaseRecord – *self*.

See also:

checksum *compute_checksum()*

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> record = IhexRecord(IhexRecord.Tag.END_OF_FILE, checksum=None)
>>> record.compute_checksum()
255
>>> record.checksum is None
True
>>> _ = record.update_checksum()
>>> record.checksum
255
```

update_count()

Updates the count field.

It updates the count attribute, assigning to it the value returned by *compute_count()*.

Returns
BaseRecord – *self*.

See also:

count *compute_count()*

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> Record = IhexFile.Record
>>> Tag = Record.Tag
>>> record = Record(Tag.DATA, data=b'abc', count=None, checksum=None)
>>> record.compute_count()
3
>>> record.count is None
True
>>> _ = record.update_count()
>>> record.count
3
```

validate(*checksum=True*, *count=True*)

Validates consistency of attribute values.

All the record attributes are checked for consistency.

Please refer to the implementation for more details.

Parameters

- **checksum** (*bool*) – Check the consistency of the checksum attribute.
- **count** (*bool*) – Check the consistency of the count attribute.

Returns
BaseRecord – *self*.

Raises
ValueError – Some targeted attributes are inconsistent.

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> _ = record.validate()
>>> record.data = b'abc'
```

(continues on next page)

(continued from previous page)

```
>>> _ = record.update_count().update_checksum().validate()
Traceback (most recent call last):
    ...
ValueError: unexpcted data
```

4.3.3 AsciiHexTag

class hexrec.formats.asciihex.**AsciiHexTag**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

ASCII-HEX tag.

Attributes

DATA	Data.
ADDRESS	Address.
CHECKSUM	Checksum.
denominator	the denominator of a rational number in lowest terms
imag	the imaginary part of a complex number
numerator	the numerator of a rational number in lowest terms
real	the real part of a complex number

Methods

is_address	Tells whether this is an address record.
is_checksum	Tells whether this is a checksum record.
__init__	
as_integer_ratio	Return integer ratio.
bit_count	Number of ones in the binary representation of the absolute value of self.
bit_length	Number of bits necessary to represent self in binary.
conjugate	Returns self, the complex conjugate of any int.
from_bytes	Return the integer represented by the given array of bytes.
to_bytes	Return an array of bytes representing an integer.

ADDRESS = 1

Address.

CHECKSUM = 2

Checksum.

DATA = 0

Data.

_DATA: Optional[*BaseTag*] = 0

Alias to a common data record tag.

This tag is used internally to build a generic data record.

__abs__()

abs(self)

__add__(*value*, /)

Return self+value.

__and__(*value*, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

classmethod __contains__(*member*)

Return True if member is a member of this enum raises TypeError if member is not an enum member

note: in 3.12 TypeError will no longer be raised, and True will also be returned if member is the value of a member in this enum

__dir__()

Returns all members and all public methods

__divmod__(*value*, /)

Return divmod(self, value).

__eq__(*value*, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(*value*, /)

Return self//value.

__format__(*format_spec*, /)

Default object formatter.

__ge__(*value*, /)

Return self>=value.

__getattribute__(*name*, /)

Return getattr(self, name).

classmethod __getitem__(*name*)

Return the member matching *name*.

__gt__(*value*, /)

Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__init__(*args, **kwds)

__int__()
int(self)

__invert__()
~self

classmethod __iter__()
Return members in definition order.

__le__(value, /)
Return self<=value.

classmethod __len__()
Return the number of members (no aliases)

__lshift__(value, /)
Return self<<value.

__lt__(value, /)
Return self<value.

__mod__(value, /)
Return self%value.

__mul__(value, /)
Return self*value.

__ne__(value, /)
Return self!=value.

__neg__()
-self

__new__(value)

__or__(value, /)
Return self|value.

__pos__()
+self

__pow__(value, mod=None, /)
Return pow(self, value, mod).

__radd__(value, /)
Return value+self.

__rand__(value, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__reduce_ex__(*proto*)
Helper for pickle.

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod=None*, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(*value*, /)
Return self-value.

__truediv__(*value*, /)
Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(*value*, /)

Return self^value.

_generate_next_value_(*start*, *count*, *last_values*)

Generate the next value when not given.

name: the name of the member start: the initial start value or None count: the number of existing members last_values: the list of values assigned

_member_type_

alias of `int`

_new_member_(***kwargs*)

Create and return a new object. See help(type) for accurate signature.

_value_repr_()

Return repr(self).

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big'*, ***, *signed=False*)

Return the integer represented by the given array of bytes.

bytes
Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

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. Default is to use 'big'.

signed
Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_address()

Tells whether this is an address record.

This method returns true if this record tag is used for *address* records.

Returns
bool – This is an address record tag.

Examples

```
>>> from hexrec import AsciiHexFile
>>> AsciiHexTag = AsciiHexFile.Record.Tag
>>> AsciiHexTag.ADDRESS.is_address()
True
>>> AsciiHexTag.DATA.is_address()
False
```

is_checksum()

Tells whether this is a checksum record.

This method returns true if this record tag is used for *checksum* records.

Returns
bool – This is a checksum record tag.

Examples

```
>>> from hexrec import AsciiHexFile
>>> AsciiHexTag = AsciiHexFile.Record.Tag
>>> AsciiHexTag.CHECKSUM.is_checksum()
True
>>> AsciiHexTag.DATA.is_checksum()
False
```

numerator
the numerator of a rational number in lowest terms

real
the real part of a complex number

to_bytes(*length=1*, *byteorder='big'*, ***, *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. Default is length 1.

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. Default is to use 'big'.

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.

4.4 avr

Atmel Generic format.

See also:

https://srecord.sourceforge.net/man/man5/srec_atmel_generic.5.html

Classes

AvrFile	Atmel Generic file object.
AvrRecord	Atmel Generic record object.
AvrTag	Atmel Generic tag.

4.4.1 AvrFile

class hexrec.formats.avr.**AvrFile**
Atmel Generic file object.

Attributes

DEFAULT_DATALEN	Default data attribute length.
FILE_EXT	Supported filename extensions.
META_KEYS	Meta information key names.
maxdatalen	Maximum byte size of the data field.
memory	Memory object stored by records role.
records	Records stored by records role.

Methods

__init__	
align	Pads blocks to align their boundaries.
append	Appends a byte.
apply_records	Applies records to memory and meta.
clear	Clears data within a range.
convert	Converts a file object to another format.
copy	Copies within a range.
crop	Clears data outside a range.
cut	Cuts data within a range.
delete	Deletes data within a range.
discard_memory	Discards underlying memory.
discard_records	Discards underlying records.
extend	Concatenates data.
fill	Fills a range.
find	Finds a substring.
flood	Floods a range.
from_blocks	Creates a file object from a memory object.
from_bytes	Creates a file object from a byte string.
from_memory	Creates a file object from a memory object.
from_records	Creates a file object from records.
get_address_max	Maximum address within memory.
get_address_min	Minimum address within memory.
get_holes	List of memory holes.
get_meta	Meta information.
get_spans	List of memory block spans.
index	Finds a substring.
load	Loads a file object from the filesystem.
merge	Merges data onto the file.
parse	Parses records from a byte stream.
print	Prints record content to stdout.
read	Extracts a substring.
save	Saves a file object into the filesystem.
serialize	Serializes records onto a byte stream.
set_meta	Sets meta information.
shift	Shifts data addresses by an offset.
split	Splits into parts.
update_records	Applies memory and meta to records.
validate_records	Validates records.

continues on next page

Table 3 – continued from previous page

view	Memory view.
write	Writes data into the file.

DEFAULT_DATALEN: int = 2

Default data attribute length.

Default value for the *maxdatalen* *meta*, which sets the maximum size of `BaseRecord.data` field values.

FILE_EXT: Sequence[str] = ['.rom']

Supported filename extensions.

Sequence of file name extension substrings (e.g. `.hex`). This list is used by functions like `guess_format_name()` to manage mapping of file *formats*.

META_KEYS: Sequence[str] = ['maxdatalen']

Meta information key names.

Sequence of key strings listing the supported *meta* information of this file *format*.

Record

alias of *AvrRecord*

__add__(*other*)

Concatenates with another file.

Equivalent to *copy()* then *extend()*.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – Concatenation of *self* and *other*.

See also:

copy() *extend()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 + file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file4 = file3 + b'789'
>>> file4.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__bool__()

bool: Has data records or memory.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> bool(file)
False
>>> _ = file.append(0)
>>> bool(file)
True
```

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> file = IhexFile.from_records([IhexRecord.create_end_of_file()])
>>> bool(file)
False
>>> file.records.insert(0, IhexRecord.create_data(0, b'\0'))
>>> bool(file)
True
```

__delitem__(*key*)

Deletes a range.

Parameters
key (*slice or int*) – Range to delete.

See also:

`bytesparse.base.MutableMemory.__delitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> del file[457]
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xz']]
>>> del file[125:457]
>>> file.memory.to_blocks()
[[123, b'abz']]
```

__eq__(*other*)

Equality test.

The file objects *self* and *other* are considered *equal* if the inequality tests of *__ne__()* result false.

Returns
bool – *self* and *other* are *equal*.

See Also
__ne__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 == file2
True
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 == file3
False
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 == file4
False
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file == ihex_file
False
>>> srec_file.memory == ihex_file.memory
True
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__getitem__(*key*)

Extracts a range.

Parameters
key (*slice or int*) – Range to extract.

Raises
ValueError – invalid range.

See also:

`bytesparse.base.ImmutableMemory.__getitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> chr(file[457])
'y'
>>> repr(file[333])
'None'
>>> file[123:125]
```

(continues on next page)

(continued from previous page)

```
b'ab'
>>> file[125:457]
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

__hash__ = None

__iadd__(*other*)

Concatenates data.

Equivalent to *extend()*.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – *self*.

See also:

memory *extend()* *discard_records()* bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 += file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file1 += b'789'
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__init__()

__ior__(*other*)

Merges with another file.

Equivalent to *merge()*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to merge.

Returns
BaseFile – *self*.

See also:

merge() discard_records()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 |= file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file1 |= b'789'
>>> file1.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__ne__(*other*)

Inequality test.

The file objects *self* and *other* are considered *unequal* if any of the following tests result true:

- Both have *memory role* (i.e. *memory*), resulting unequal;
- Both have *records role* (i.e. *records*), resulting unequal;
- *other* does not have a *meta* listed by *META_KEYS*;
- A *meta* value (among those of *META_KEYS*) is different.

Returns
bool – *self* and *other* are *unequal*.

See also:

__eq__() memory records META_KEYS

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 != file2
False
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 != file3
True
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
```

(continues on next page)

(continued from previous page)

```
>>> file1 != file4
True
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file != ihex_file
True
>>> srec_file.memory != ihex_file.memory
False
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__or__(*other*)

Merges with another file.

Equivalent to *copy()* then *merge()*.

Parameters
other (BaseFile or bytes) – Other file or bytes to merge.

Returns
BaseFile – *self* merged with *other*.

See also:

copy() *merge()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 | file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file4 = file3 | b'789'
>>> file4.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__setitem__(*key*, *value*)

Sets a range.

Parameters

- **key** (*slice or int*) – Range to set.
- **value** (bytes, bytesparse.base.ImmutableMemory, None) – Value(s) to set. None acts like *clear()*.

Raises
ValueError – invalid range.

See also:

bytesparse.base.MutableMemory.__setitem__() *clear()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file[124] = b'?'
>>> file.memory.to_blocks()
[[123, b'a?c'], [456, b'xyz']]
>>> file[:125] = None
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'xyz']]
>>> file[457:458] = b'789'
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'x789z']]
```

__weakref__

list of weak references to the object (if defined)

classmethod _is_line_empty(*line*)

Empty line check.

Tells whether a *line* has no meaningful content (e.g. all whitespace). The check itself depends on the implementing file *format*. It may be used internally to skip empty lines, e.g. by *parse()*.

Parameters
line (*bytes*) – A line, byte string.

Returns
bool: The *line* is empty.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexFile._is_line_empty(b'')
True
>>> IhexFile._is_line_empty(b' \t\v\r\n')
True
>>> IhexFile._is_line_empty(b':00000001FF\r\n')
False
```

align(*modulo*, *start=None*, *endex=None*, *pattern=0*)

Pads blocks to align their boundaries.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*, so that memory blocks are aligned to the required *modulo*.

Any stored *records* are discarded upon return.

Parameters

- **modulo** (*int*) – Alignment modulo.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.align()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [134, b'xyz']])
>>> _ = file.align(4, pattern=b'.')
>>> file.memory.to_blocks()
[[120, b'...abc..'], [132, b'..xyz...']]
```

append(*item*)

Appends a byte.

It appends the *item* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters

item (*byte or int*) – Byte to append.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.append()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123)
>>> _ = file.append(b'.')
>>> _ = file.append(0)
>>> file.memory.to_blocks()
[[123, b'abc.\x00']]
```

apply_records()

Applies records to memory and meta.

This method processes the stored *records*, converting *data* as *memory*, and special records into their *meta* counterparts.

This effectively converts the *records role* into the *memory role* (keeping both).

The *memory* and *meta* are assigned upon return. Any exceptions being raised should not alter the file object.

Returns
BaseFile – *self*.

Raises
ValueError – *records* attribute not populated.

See also:

records *memory* *get_meta()* *update_records()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_start_linear_address(456),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records, maxdatalen=16)
>>> _ = file.apply_records()
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
```

clear(*start=None*, *endex=None*)

Clears data within a range.

It clears the specified range of underlying *memory* object, making a memory hole.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
BaseFile – *self*.

See also:

memory *discard_records()* bytesparse.base.MutableMemory.clear()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.clear(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

classmethod convert(*source*, *meta=True*)

Converts a file object to another format.

It copies the *memory* and *meta* of the *source* file object, creating a new one of the target `BaseFile` format type.

Parameters

- **source** (`BaseFile`) – Source file object to convert.
- **meta** (*bool*) – Copy *meta* information to the target file object. Only the keys of the target *META_KEYS* are processed.

Returns

`BaseFile` – Converted copy of *source* to the target format.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> source = IhexFile.from_blocks(blocks, startaddr=789)
>>> target = SrecFile.convert(source)
>>> target.memory is source.memory
False
>>> target.memory == source.memory
True
>>> target.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 789}
```

copy(*start=None*, *endex=None*, *meta=True*)

Copies within a range.

It copied data within the specified range of the file object, creating a new one carrying the inner slice.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns
BaseFile – *self*.

See also:

memory *get_meta()* *discard_records()* bytesparse.base.MutableMemory.cut()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.copy(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

crop(*start=None*, *endex=None*)

Clears data outside a range.

It clears outside the specified range of underlying *memory* object, timming it.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
BaseFile – *self*.

See also:

memory *discard_records()* bytesparse.base.MutableMemory.crop()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.crop(start=124, endex=132)
>>> file.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
```

cut(*start=None*, *endex=None*, *meta=False*)

Cuts data within a range.

It takes data within the specified range away from the file object, creating a new one carrying the inner slice. The inner slice is cleared from *self*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

`BaseFile` – *self*.

See also:

memory *clear()* *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.cut(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

delete(*start=None*, *endex=None*)

Deletes data within a range.

It deletes the specified range of underlying *memory* object, shifting all subsequent data towards the collapsed range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.delete()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.delete(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'az']]
```

discard_memory()

Discards underlying memory.

The underlying *memory* object is assigned `None`.

If the underlying *records* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_bytes(b'abc', offset=123)
>>> _ = file.update_records()
>>> _ = file.discard_memory()
>>> _ = file.update_records()
Traceback (most recent call last):
    ...
ValueError: memory instance required
```

discard_records()

Discards underlying records.

The underlying *records* object is assigned `None`.

If the underlying *memory* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
```

(continues on next page)

(continued from previous page)

```
>>> _ = file.validate_records()
>>> _ = file.discard_records()
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: records required
```

extend(*other*)

Concatenates data.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> _ = file1.extend(file2)
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> _ = file1.extend(b'789')
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

fill(*start=None*, *endex=None*, *pattern=0*)

Fills a range.

It writes a *pattern* of bytes onto the underlying *memory* object, overwriting anything within the specified range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for filling.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.fill()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.fill(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'a........z']]
```

find(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it returns -1.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
int – *item* beginning address; -1 if not found.

See also:

index bytesparse.base.ImmutableMemory.find()

Notes

The internal *memory* might allow negative addresses for its stored data. In that case, *index()* would be more appropriate, because it raises an exception when the *item* is not found.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.find(b'yz')
457
>>> file.find(ord('b'))
```

(continues on next page)

(continued from previous page)

```
124
>>> file.find(b'?')
-1
```

flood(*start=None*, *endex=None*, *pattern=0*)

Floods a range.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.flood()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.get_holes()
[(126, 130)]
>>> _ = file.flood(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'abc....xyz']]
```

classmethod from_blocks(*blocks*, ***meta*)

Creates a file object from a memory object.

The *blocks* are put into the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **blocks** (*list of blocks*) – Memory blocks to put into *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

`BaseFile` – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_blocks()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_bytes(*data*, *offset=0*, ***meta*)

Creates a file object from a byte string.

The byte string makes a single *data* block, placed at some offset within the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **data** (*bytes*) – A byte string used to make a single data block.
- **offset** (*int*) – Offset of the single data block within *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
`BaseFile` – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_bytes()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
8
```

classmethod from_memory(*memory=None*, ***meta*)

Creates a file object from a memory object.

The *memory* is set as the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **memory** (bytesparse.base.MutableMemory) – Memory object to set as *memory*. If None, an empty memory object is automatically created.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

BaseFile – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS bytesparse.base.MutableMemory

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from bytesparse import Memory
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> memory = Memory.from_blocks(blocks)
>>> from hexrec import SrecFile
>>> file = SrecFile.from_memory(memory, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_records(*records*, *maxdatalen=None*)

Creates a file object from records.

The *records* sequence is set as the record attribute of the created file object.

This method creates a file object in *records role*. This means that only its *records* is internally instanced, while the *memory* requires manual or lazy instancing (i.e. either via direct call to *apply_records()*, or any other methods indirectly calling it).

Parameters

- **records** (list of BaseRecord) – Record sequence to set as *records*.
- **maxdatalen** (Optional[int]) – Maximum record *data* field size. If None, the maximum non-zero size of the *data* field from the *records* sequence is used. If all the *records* have zero sized *data* field, the class attribute *DEFAULT_DATALEN* is used.

Returns

BaseFile – The created file object.

Raises
ValueError – invalid *meta* values.

See also:

`BaseRecord`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
3
```

get_address_max()

Maximum address within memory.

It returns the maximum address of the underlying *memory* object.

Returns
int – Maximum address.

See also:

`bytesparse.base.ImmutableMemory.endin`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_max()
458
```

get_address_min()

Minimum address within memory.

It returns the minimum address of the underlying *memory* object.

Returns
int – Minimum address.

See also:

`bytesparse.base.ImmutableMemory.start`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_min()
123
```

get_holes()

List of memory holes.

It scans the underlying *memory* and returns the list of memory holes/gaps.

Each hole is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory hole boundaries.

See also:

`bytesparse.base.ImmutableMemory.gaps()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_holes()
[(126, 456), (459, 789)]
```

get_meta()

Meta information.

It builds and returns a dictionary of *meta* information. Meta keys are taken from the *META_KEYS* class attribute.

Returns
dict – Meta information dictionary.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks, header=b'HDR\0')
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 0}
```

get_spans()

List of memory block spans.

It scans the underlying *memory* and returns the list of memory block spans/intervals.

Each span is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory block boundaries.

See also:

`bytesparse.base.ImmutableMemory.intervals()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_spans()
[(123, 126), (456, 459), (789, 791)]
```

index(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it raises `ValueError`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
int – *item* beginning address.

Raises
ValueError – *item* not found.

See also:

find `bytesparse.base.ImmutableMemory.index()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.index(b'yz')
457
>>> file.index(ord('b'))
124
>>> file.index(b'?')
Traceback (most recent call last):
    ...
ValueError: subsection not found
```

classmethod load(*path*, **args*, ***kwargs*)

Loads a file object from the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *parse()* to load a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdin.buffer` is used.
- **args** – Forwarded to *parse()*.
- **kwargs** – Forwarded to *parse()*.

Returns

`BaseFile` – Loaded file object.

See also:

save() *parse()* `open()` `sys.stdin.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.load('data.hex')
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

property maxdatalen: int

Maximum byte size of the data field.

This property sets the maximum byte size of the *data* field of a serialized record.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises

ValueError – Invalid maximum data length.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> file.maxdatalen
16
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> file.maxdatalen = 8
>>> _ = file.print()
S0030000FC
S10B0000000102030405060 7D8
S10B000808090A0B0C0D0E0F90
S10B00101011121314151617 48
S10B001818191A1B1C1D1E1F00
S10B00202021222324252627 B8
S10B002828292A2B2C2D2E2F70
S10B00303031323334353637 28
S10B003838393A3B3C3D3E3FE0
S5030008F4
S9030000FC
>>> file.maxdatalen = 0
Traceback (most recent call last):
    ...
ValueError: invalid maximum data length
```

Type
int

property memory: MutableMemory

Memory object stored by records role.

This readonly property exposes the memory object stored by the file object while in *memory role*.

If this property is accessed while the file object is not in *memory role*, it automatically activates it by an implicit call to *apply_records()*, with default arguments.

For more control activating the *memory role*, please call *apply_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *records role* (i.e. altering content of *records*) would implicitly discard *memory* via *discard_memory()*.

See also:

apply_records() *discard_memory()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> _ = file.write(789, b'?!')
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz'], [789, b'?!']]
```

Type
`bytesparse.Memory`

merge(**files*, *clear=False*)

Merges data onto the file.

It writes the provided *files* onto *self*, in the provided order. Any common address ranges are overwritten.

Any stored *records* are discarded upon return.

Parameters

- **files** (`BaseFile`) – Files to merge.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns
`BaseFile` – *self*.

See also:

clear() *discard_records()* *write()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = SrecFile.from_bytes(b'<<<?????>>>', offset=450)
>>> _ = file3.merge(file1, file2)
```

(continues on next page)

(continued from previous page)

```
>>> file3.memory.to_blocks()
[[123, b'abc'], [450, b'<<<???xyz>>']]
```

classmethod parse(*stream*, *ignore_errors=False*, *ignore_after_termination=True*)

Parses records from a byte stream.

It executes `BaseRecord.parse()` for each line of the incoming *stream*, creating a new file object with the collected records calling *from_records()*.

Lines resulting empty by `_is_empty_line()` are just discarded.

Notes

Please refer to the actual implementation of each record file *format*, because it may be more specialized.

Parameters

- **stream** (*bytes IO or buffer*) – Stream or byte buffer to parse records from.
- **ignore_errors** (*bool*) – Ignore `Exception` raised by `BaseRecord.parse()`.
- **ignore_after_termination** (*bool*) – Ignore anything after the termination record was parsed, if supported (e.g. *End Of File* or *start address* record, depending on the specific file *format*).

Returns

`BaseFile` – *self*.

See also:

parse() `BaseRecord.parse()` *from_records()* `_is_empty_line()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> buffer = b'''
...     :03DA7A0061626383
...     :040000050000CAFE2F
...     :00000001FF
... '''
>>> import io
>>> stream = io.BytesIO(buffer)
>>> file = IhexFile.parse(stream)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
>>> file = IhexFile.parse(buffer)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

print(**args*, *stream=None*, *color=False*, *start=None*, *stop=None*, ***kwargs*)

Prints record content to stdout.

This helper method prints each record of *records* via `BaseRecord.print()`. As such, it also supports colored tokens and streams different from *stdout*.

It is possible to print subset of the records by specifying the record index range.

> **Warning:** This method is **NOT** equivalent to *serialize()*, because it just prints each record from *records*. Please use *serialize()* for an actual serialization of the whole file.

Parameters

- **args** – Forwarded to the underlying call to `to_tokens()`.
- **stream** (*byte stream*) – Stream to print onto. If `None`, *stdout* is used.
- **color** (*bool*) – Colorize record tokens with ANSI color codes.
- **start** (*int*) – Inclusive start record index of the specified range. If `None`, start from the first record.
- **stop** (*int*) – Exclusive end record index of the specified range. If negative, look back from the last index. If `None`, print up to the last record.
- **kwargs** – Forwarded to the underlying call to `to_tokens()`.

Returns

`BaseFile` – *self*.

See also:

`BaseRecord.print()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212222324252627282 92A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> _ = file.print(color=True, start=1, stop=-2)
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212222324252627282 92A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
```

read(*start=None*, *endex=None*, *fill=0*)

Extracts a substring.

It extracts a byte string from the specified range, filling any memory holes/gaps (without altering *memory*).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **fill** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory `bytesparse.base.MutableMemory.extract()` `bytesparse.base.MutableMemory.to_bytes()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.read(start=124, endex=132)
b'bc\x00\x00\x00\x00xy'
>>> file.read(start=124, endex=132, fill=b'.')
b'bc....xy'
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

property records: MutableSequence[*BaseRecord*]

Records stored by records role.

This readonly property exposes the list of records stored by the file object while in *records role*.

If this property is accessed while the file object is not in *records role*, it automatically activates it by an implicit call to *update_records()*, with default arguments.

For more control activating the *records role*, please call *update_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *memory role* (i.e. altering content of *memory*) would implicitly discard *records* via *discard_records()*.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, startaddr=789)
>>> len(file.records)
5
>>> _ = file.print()
S0030000FC
S106007B61626358
S10601C878797AC5
S5030002FA
S9030315E4
>>> _ = file.update_records(data_tag=SrecFile.Record.Tag.DATA_32)
>>> _ = file.print()
S0030000FC
S3080000007B61626356
S308000001C878797AC3
S5030002FA
S70500000315E2
```

Type
list of `BaseRecord`

save(*path*, **args*, ***kwargs*)

Saves a file object into the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *serialize()* to save a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdout.buffer` is used.
- **args** – Forwarded to *serialize()*.
- **kwargs** – Forwarded to *serialize()*.

Returns
`BaseFile` – *self*.

See also:

load() *serialize()* `open()` `sys.stdout.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> _ = file.save('data.hex')
```

serialize(*stream*, **args*, ***kwargs*)

Serializes records onto a byte stream.

It executes `BaseRecord.serialize()` for each of the stored *records*.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **args** – Forwarded to `BaseRecord.serialize()` of each record.
- **kwargs** – Forwarded to `BaseRecord.serialize()` of each record.

Returns

`BaseFile` – *self*.

See also:

parse() `BaseRecord.serialize()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> import sys
>>> _ = file.serialize(sys.stdout.buffer, end=b'\n')
:03DA7A0061626383
:040000050000CAFE2F
:00000001FF
```

set_meta(*meta*, *strict=True*)

Sets meta information.

It sets the provided *kwargs* to their matching *meta* attributes, as listed by *META_KEYS*.

Parameters

- **meta** (*dict*) – Mapping of the *meta* information to set.
- **strict** (*bool*) – All the keys within *meta* must exist within *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *get_meta()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 0}
>>> _ = file.set_meta(dict(header=b'HDR\0', startaddr=456))
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 456}
```

shift(*offset*)

Shifts data addresses by an offset.

It shifts addresses of the underlying *memory* object data blocks by the provided *offset* amount.

Any stored *records* are discarded upon return.

Parameters

offset (*int*) – Offset to apply to the underlying data block addresses.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.shift()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> _ = file.shift(1000)
>>> file.memory.to_blocks()
[[1123, b'abc'], [1456, b'xyz']]
```

split(**addresses*, *meta=True*)

Splits into parts.

The provided *addresses* are sorted and used as markers to split *self* into parts.

Each part is the *copy()* of *self* within the range of that part, in *memory role* (i.e., *records* is not populated).

Parameters

- **addresses** (*int*) – Split points.
- **meta** (*bool*) – Each part inherits *meta* from *self*.

Returns
list of `BaseFile` – Parts after splitting.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'Hello, World!', offset=123)
>>> parts = file.split(128, 130)
>>> for part in parts: print(part.memory.to_blocks())
[[123, b'Hello']]
[[128, b', ']]
[[130, b'World!']]
>>> file.memory.to_blocks()
[[123, b'Hello, World!']]
```

`update_records`()

Applies memory and meta to records.

This method processes the stored *memory* and *meta* information to generate the sequence of *records*.

This effectively converts the *memory role* into the *records role* (keeping both).

The *records* is assigned upon return. Any exceptions being raised should not alter the file object.

Returns
AvrFile – *self*.

Raises
ValueError – *memory* attribute not populated.

See also:

records *memory* *get_meta()* *apply_records()*

Examples

```
>>> from hexrec import AvrFile
>>> blocks = [[124, b'abcd']]
>>> file = AvrFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[124, b'abcd']]
>>> file.get_meta()
{'maxdatalen': 2}
>>> _ = file.update_records()
>>> len(file.records)
2
>>> _ = file.print()
00003E:6162
00003F:6364
```

validate_records(*data_ordering=False*)

Validates records.

It performs consistency checks for the underlying *records*.

Parameters

data_ordering (*bool*) – Checks that the *data* record sequence has monotonically increasing addresses, without any overlapping.

Returns

AvrFile – *self*.

Raises

ValueError – Invalid record sequence.

Examples

```
>>> from hexrec import AvrFile
>>> records = [AvrFile.Record.create_data(62, b'ab'),
...            AvrFile.Record.create_data(61, b'cd')]
>>> file = AvrFile.from_records(records)
>>> _ = file.validate_records(data_ordering=True)
Traceback (most recent call last):
    ...
ValueError: unordered data record
```

view(*start=None*, *endex=None*)

Memory view.

It returns a `memoryview` over the specified range, which must cover a *contiguous* data region (i.e. no memory holes within).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

memoryview – View of the specified range.

Raises

ValueError – non-contiguous data within range.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> bytes(file.view(start=456, endex=458))
b'xy'
>>> bytes(file.view())
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

write(*address*, *data*, *clear=False*)

Writes data into the file.

It writes the provided *data* into the underlying *memory* object.

Any stored *records* are discarded upon return.

Parameters

- **address** (*int*) – Address where *data* has to be written.
- **data** (*bytes or memory*) – Byte data to write.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

BaseFile – *self*.

See also:

memory *clear()* *discard_records()* bytesparse.base.MutableMemory.write()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> _ = file.write(123, b'abc')
>>> _ = file.write(555, ord('?'))
>>> _ = file.write(1000, SrecFile.from_bytes(b'xyz', offset=456))
>>> file.memory.to_blocks()
[[123, b'abc'], [555, b'?'], [1456, b'xyz']]
```

4.4.2 AvrRecord

class hexrec.formats.avr.**AvrRecord**(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

Atmel Generic record object.

Attributes

EQUALITY_KEYS	Meta keys for equality checks.
LINE_REGEX	Line parser regex.
META_KEYS	Meta keys.

Methods

__init__	
compute_checksum	Computes the checksum field value.
compute_count	Compute the count field value.
copy	Shallow copy.
create_data	Creates a data record.
data_to_int	Interprets data bytes as integer.
get_meta	Gets meta information.
parse	Parses a record from bytes.
print	Prints a record.
serialize	Serializes onto a stream.
to_bytestr	Converts into a byte string.
to_tokens	Converts into byte string tokens.
update_checksum	Updates the checksum field.
update_count	Updates the count field.
validate	Validates consistency of attribute values.

EQUALITY_KEYS: Sequence[str] = ['address', 'checksum', 'count', 'data', 'tag']

Meta keys for equality checks.

Equality methods (*__eq__()* and *__ne__()*) check against these *meta* keys only. Any other *meta* keys are just ignored.

LINE_REGEX = re.compile(b'^(?P<before>[\\t]*)(?P<address>[0-9A-Fa-f]{6})[\\t]*:[\\t]*(?P<data>([0-9A-Fa-f]{4}))(?P<after>[\\t]*)\\r?\\n?$')

Line parser regex.

META_KEYS: Sequence[str] = ['address', 'after', 'before', 'checksum', 'coords', 'count', 'data', 'tag']

Meta keys.

This sequence holds the *meta* keys for copying (see *copy()*).

Tag

alias of *AvrTag*

__bytes__()

Serializes the record into bytes.

Returns
bytes – Byte serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> bytes(record)
b':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> bytes(record)
b'abc'
```

__eq__(*other*)

Equality test.

This method returns true if *self* is considered equal to *other*.

As inequality is usually easier to check, this method is usually implemented as a trivial `not self != other` (*__ne__()*).

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* equals *other*.

See also:

__ne__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 == ihex2
True
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 == ihex3
False
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 == raw
False
```

__hash__ = None

__init__(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

__ne__(*other*)

Ineuality test.

This method returns true if *self* is considered unequal to *other*.

Each attribute listed by *EQUALITY_KEYS* is compared between *self* and *other*. This method returns whether any attributes do not match.

Parameters
other (BaseRecord) – Record to compare to.

Returns
bool – *self* and *other* are unequal.

See also:

__eq__()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 != ihex2
False
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 != ihex3
True
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 != raw
True
```

__repr__()

String representation.

It returns a string representation of the record content, for human understanding only.

Returns
str – String representation.

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> repr(record)
"<<class 'hexrec.formats.ihex.IhexRecord'> @...
  address:=0 after:=b'' before:=b'' checksum:=255 coords:=(-1, -1)
  count:=0 data:=b'' tag:=<IhexTag.END_OF_FILE: 1>>"
```

__str__()

Serializes the record into a string.

Returns
str – String serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> str(record)
'abc'
```

__weakref__

list of weak references to the object (if defined)

compute_checksum()

Computes the checksum field value.

It computes and returns the format-specific `checksum` value of a record.

When not specialized, it returns `None` by default.

Returns
int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_checksum()
215
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_checksum())
'None'
```

compute_count()

Compute the count field value.

It computes and returns the format-specific `count` value of a record.

When not specialized, it returns `None` by default.

Returns
int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_count()
3
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_count())
'None'
```

copy(*validate=True*)

Shallow copy.

It calls the record constructor, passing *meta* to it.

Parameters
validate (*bool*) – Performs validation on instantiation (*__init__()*).

Returns
`BaseRecord` – Shallow copy.

See also:

__init__() *get_meta()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record1 = IhexFile.Record.create_data(0x1234, b'abc')
>>> record2 = record1.copy()
>>> record1 is record2
False
>>> record1 == record2
True
```

classmethod create_data(*address*, *data*)

Creates a data record.

This is a mandatory class method to instantiate a *data* record.

Parameters

- **address** (*int*) – Record address. If not supported, set zero.
- **data** (*bytes*) – Record byte data.

Returns

BaseRecord – Data record object.

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> str(record)
':0312340061626391\r\n'
```

data_to_int(*byteorder='big'*, *signed=False*)

Interprets data bytes as integer.

It creates an integer from bytes of the data field.

Parameters

- **byteorder** (*'big' or 'little'*) – Byte order (endianness): either 'big' (default) or 'little'.
- **signed** (*bool*) – Signed integer (2-complement); default false.

Returns

int – Interpreted integer value.

See also:

int.from_bytes()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0xABCD)
>>> record.data
b'\xab\xcd'
>>> addrext = record.data_to_int()
>>> addrext, hex(addrext)
(43981, '0xabcd')
```

get_meta()

Gets meta information.

It returns all the object attributes whose keys are listed by *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

See also:

META_KEYS set_meta()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> record.get_meta()
{'address': 0, 'after': b'', 'before': b'', 'checksum': 255,
 'coords': (-1, -1), 'count': 0, 'data': b'',
 'tag': <IhexTag.END_OF_FILE: 1>}
```

classmethod parse(*line*, *validate=True*)

Parses a record from bytes.

Parameters

- **line** (*bytes*) – String of bytes to parse.
- **validate** (*bool*) – Perform validation checks.

Returns

BaseRecord – Parsed record.

Raises

ValueError – Syntax error.

Examples

```
>>> from hexrec import AvrFile
>>> record = AvrFile.Record.parse(b'000080:4865\r\n')
>>> record.tag
<AvrTag.EOF: 1>
>>> AvrFile.Record.parse(b'000080::4865\r\n')
Traceback (most recent call last):
    ...
ValueError: syntax error
```

print(**args*, *stream=None*, *color=False*, ***kwargs*)

Prints a record.

The record is converted into tokens (eventually colorized) then joined and written onto a byte stream (*stdout* by default).

Parameters

- **args** – Forwarded to the underlying call to *to_tokens()*.
- **stream** (io.BytesIO) – The byte stream where the record tokens are printed. If None, *stdout* is selected.
- **color** (*bool*) – Tokens are colorized before printing.
- **kwargs** – Forwarded to the underlying call to *to_tokens()*.

Returns
BaseRecord – *self*.

See also:

to_tokens() colorize_tokens() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> _ = record.print()
:0312340061626391
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.print(stream=stream, color=True)
>>> stream.getvalue()
b'\x1b[0m\x1b[33m:\x1b[34m03\x1b[31m1234\x1b[32m00\x1b[36m61\x1b[96m62\
↪x1b[36m63\x1b[35m91\x1b[0m\r\n\x1b[0m'
```

serialize(*stream*, **args*, ***kwargs*)

Serializes onto a stream.

This wraps a call to *to_bytestr()* and stream.write.

Parameters

- **stream** (io.BytesIO) – Stream to write.
- **args** – Forwarded to *to_bytestr()*.
- **kwargs** – Forwarded to *to_bytestr()*.

Returns
BaseRecord – *self*.

See also:

to_bytestr() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.serialize(stream, end=b'\n')
>>> stream.getvalue()
b':0312340061626391\n'
```

to_bytestr(*end=b'\\r\\n'*)

Converts into a byte string.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Byte string representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_bytestr(end=b'\n')
b':0312340061626391\n'
```

to_tokens(*end=b'\\r\\n'*)

Converts into byte string tokens.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Mapping of token keys to token byte strings.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_tokens(end=b'\n')
{'before': b'', 'begin': b':', 'count': b'03', 'address': b'1234',
 'tag': b'00', 'data': b'616263', 'checksum': b'91', 'after': b'',
 'end': b'\n'}
```

update_checksum()

Updates the checksum field.

It updates the `checksum` attribute, assigning to it the value returned by *compute_checksum()*.

Returns

`BaseRecord` – *self*.

See also:

`checksum` *compute_checksum()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> record = IhexRecord(IhexRecord.Tag.END_OF_FILE, checksum=None)
>>> record.compute_checksum()
255
>>> record.checksum is None
True
>>> _ = record.update_checksum()
>>> record.checksum
255
```

`update_count()`

Updates the count field.

It updates the `count` attribute, assigning to it the value returned by *compute_count()*.

Returns
`BaseRecord` – *self*.

See also:

`count` *compute_count()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> Record = IhexFile.Record
>>> Tag = Record.Tag
>>> record = Record(Tag.DATA, data=b'abc', count=None, checksum=None)
>>> record.compute_count()
3
>>> record.count is None
True
>>> _ = record.update_count()
>>> record.count
3
```

`validate`(*checksum=True*, *count=True*)

Validates consistency of attribute values.

All the record attributes are checked for consistency.

Please refer to the implementation for more details.

Parameters

- **checksum** (*bool*) – Check the consistency of the `checksum` attribute.
- **count** (*bool*) – Check the consistency of the `count` attribute.

Returns
BaseRecord – *self*.

Raises
ValueError – Some targeted attributes are inconsistent.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> _ = record.validate()
>>> record.data = b'abc'
>>> _ = record.update_count().update_checksum().validate()
Traceback (most recent call last):
    ...
ValueError: unexpcted data
```

4.4.3 AvrTag

class hexrec.formats.avr.**AvrTag**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Atmel Generic tag.

Attributes

DATA	Data.

DATA = Ellipsis
Data.

_DATA: Optional['BaseTag'] = Ellipsis
Alias to a common data record tag.

This tag is used internally to build a generic data record.

classmethod __contains__(*member*)
Return True if member is a member of this enum raises TypeError if member is not an enum member

note: in 3.12 TypeError will no longer be raised, and True will also be returned if member is the value of a member in this enum

classmethod __getitem__(*name*)
Return the member matching *name*.

classmethod __iter__()
Return members in definition order.

classmethod __len__()
Return the number of members (no aliases)

__new__(*value*)

_generate_next_value_(*start*, *count*, *last_values*)

Generate the next value when not given.

name: the name of the member start: the initial start value or None count: the number of existing members last_values: the list of values assigned

_member_type_

alias of `object`

_new_member_(***kwargs*)

Create and return a new object. See help(type) for accurate signature.

4.5 ihex

Intel HEX format.

See also:

https://en.wikipedia.org/wiki/Intel_HEX

Classes

IhexFile	Intel HEX file object.
IhexRecord	Intel HEX record object.
IhexTag	Intel HEX tag.

4.5.1 IhexFile

class hexrec.formats.ihex.**IhexFile**

Intel HEX file object.

Attributes

DEFAULT_DATALEN	Default data attribute length.
FILE_EXT	Supported filename extensions.
META_KEYS	Meta information key names.
linear	Linear addressing.
maxdatalen	Maximum byte size of the data field.
memory	Memory object stored by records role.
records	Records stored by records role.
startaddr	Start address.

Methods

__init__	
align	Pads blocks to align their boundaries.
append	Appends a byte.
apply_records	Applies records to memory and meta.
clear	Clears data within a range.
convert	Converts a file object to another format.
copy	Copies within a range.
crop	Clears data outside a range.
cut	Cuts data within a range.
delete	Deletes data within a range.
discard_memory	Discards underlying memory.
discard_records	Discards underlying records.
extend	Concatenates data.
fill	Fills a range.
find	Finds a substring.
flood	Floods a range.
from_blocks	Creates a file object from a memory object.
from_bytes	Creates a file object from a byte string.
from_memory	Creates a file object from a memory object.
from_records	Creates a file object from records.
get_address_max	Maximum address within memory.
get_address_min	Minimum address within memory.
get_holes	List of memory holes.
get_meta	Meta information.
get_spans	List of memory block spans.
index	Finds a substring.
load	Loads a file object from the filesystem.
merge	Merges data onto the file.
parse	Parses records from a byte stream.
print	Prints record content to stdout.
read	Extracts a substring.
save	Saves a file object into the filesystem.
serialize	Serializes records onto a byte stream.
set_meta	Sets meta information.
shift	Shifts data addresses by an offset.
split	Splits into parts.
update_records	Applies memory and meta to records.
validate_records	Validates records.
view	Memory view.
write	Writes data into the file.

DEFAULT_DATALEN: int = 16

Default data attribute length.

Default value for the *maxdatalen* *meta*, which sets the maximum size of `BaseRecord.data` field values.

FILE_EXT: Sequence[str] = ['.hex', '.mcs', '.int', '.ihex', '.ihe', '.ihx', '.h80', '.h86', '.a43', '.a90', '.obj', '.obl', '.obh', '.rom', '.eep', '.num']

Supported filename extensions.

Sequence of file name extension substrings (e.g. `.hex`). This list is used by functions like `guess_format_name()` to manage mapping of file *formats*.

META_KEYS: Sequence[str] = ['linear', 'maxdatalen', 'startaddr']

Meta information key names.

Sequence of key strings listing the supported *meta* information of this file *format*.

Record

alias of *IhexRecord*

__add__(*other*)

Concatenates with another file.

Equivalent to *copy()* then *extend()*.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – Concatenation of *self* and *other*.

See also:

copy() extend()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 + file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file4 = file3 + b'789'
>>> file4.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__bool__()

bool: Has data records or memory.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> bool(file)
False
>>> _ = file.append(0)
>>> bool(file)
True
```

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> file = IhexFile.from_records([IhexRecord.create_end_of_file()])
>>> bool(file)
False
>>> file.records.insert(0, IhexRecord.create_data(0, b'\0'))
>>> bool(file)
True
```

__delitem__(*key*)

Deletes a range.

Parameters
key (*slice or int*) – Range to delete.

See also:

`bytesparse.base.MutableMemory.__delitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> del file[457]
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xz']]
>>> del file[125:457]
>>> file.memory.to_blocks()
[[123, b'abz']]
```

__eq__(*other*)

Equality test.

The file objects *self* and *other* are considered *equal* if the inequality tests of *__ne__()* result false.

Returns
bool – *self* and *other* are *equal*.

See Also
__ne__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
```

(continues on next page)

(continued from previous page)

```
>>> file1 == file2
True
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 == file3
False
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 == file4
False
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file == ihex_file
False
>>> srec_file.memory == ihex_file.memory
True
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__getitem__(*key*)

Extracts a range.

Parameters
key (*slice or int*) – Range to extract.

Raises
ValueError – invalid range.

See also:

`bytesparse.base.ImmutableMemory.__getitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> chr(file[457])
'y'
>>> repr(file[333])
'None'
>>> file[123:125]
b'ab'
>>> file[125:457]
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

__hash__ = None

__iadd__(*other*)

Concatenates data.

Equivalent to *extend()*.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – *self*.

See also:

memory *extend()* *discard_records()* bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 += file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file1 += b'789'
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__init__()

__ior__(*other*)

Merges with another file.

Equivalent to *merge()*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to merge.

Returns
BaseFile – *self*.

See also:

merge() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 |= file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file1 |= b'789'
>>> file1.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__ne__(*other*)

Inequality test.

The file objects *self* and *other* are considered *unequal* if any of the following tests result true:

- Both have *memory role* (i.e. *memory*), resulting unequal;
- Both have *records role* (i.e. *records*), resulting unequal;
- *other* does not have a *meta* listed by *META_KEYS*;
- A *meta* value (among those of *META_KEYS*) is different.

Returns

bool – *self* and *other* are *unequal*.

See also:

__eq__() memory records META_KEYS

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 != file2
False
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 != file3
True
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 != file4
True
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file != ihex_file
True
>>> srec_file.memory != ihex_file.memory
False
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__or__(*other*)

Merges with another file.

Equivalent to *copy()* then *merge()*.

Parameters
other (BaseFile or bytes) – Other file or bytes to merge.

Returns
BaseFile – *self* merged with *other*.

See also:

copy() *merge()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 | file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file4 = file3 | b'789'
>>> file4.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__setitem__(*key*, *value*)

Sets a range.

Parameters

- **key** (*slice or int*) – Range to set.
- **value** (bytes, bytesparse.base.ImmutableMemory, None) – Value(s) to set. None acts like *clear()*.

Raises
ValueError – invalid range.

See also:

bytesparse.base.MutableMemory.__setitem__() *clear()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file[124] = b'?'
>>> file.memory.to_blocks()
[[123, b'a?c'], [456, b'xyz']]
>>> file[:125] = None
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'xyz']]
>>> file[457:458] = b'789'
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'x789z']]
```

__weakref__

list of weak references to the object (if defined)

classmethod _is_line_empty(*line*)

Empty line check.

Tells whether a *line* has no meaningful content (e.g. all whitespace). The check itself depends on the implementing file *format*. It may be used internally to skip empty lines, e.g. by *parse()*.

Parameters
line (*bytes*) – A line, byte string.

Returns
bool: The *line* is empty.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexFile._is_line_empty(b'')
True
>>> IhexFile._is_line_empty(b' \t\v\r\n')
True
>>> IhexFile._is_line_empty(b':00000001FF\r\n')
False
```

align(*modulo*, *start=None*, *endex=None*, *pattern=0*)

Pads blocks to align their boundaries.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*, so that memory blocks are aligned to the required *modulo*.

Any stored *records* are discarded upon return.

Parameters

- **modulo** (*int*) – Alignment modulo.

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns
`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.align()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [134, b'xyz']])
>>> _ = file.align(4, pattern=b'.')
>>> file.memory.to_blocks()
[[120, b'...abc..'], [132, b'..xyz...']]
```

append(*item*)

Appends a byte.

It appends the *item* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
item (*byte or int*) – Byte to append.

Returns
`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.append()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123)
>>> _ = file.append(b'.')
>>> _ = file.append(0)
>>> file.memory.to_blocks()
[[123, b'abc.\x00']]
```

apply_records()

Applies records to memory and meta.

This method processes the stored *records*, converting *data* as *memory*, and special records into their *meta* counterparts.

This effectively converts the *records role* into the *memory role* (keeping both).

The *memory* and *meta* are assigned upon return. Any exceptions being raised should not alter the file object.

Returns
BaseFile – *self*.

Raises
ValueError – *records* attribute not populated.

See also:

records *memory* *get_meta()* *update_records()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_start_linear_address(456),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records, maxdatalen=16)
>>> _ = file.apply_records()
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
```

clear(*start=None*, *endex=None*)

Clears data within a range.

It clears the specified range of underlying *memory* object, making a memory hole.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
BaseFile – *self*.

See also:

memory *discard_records()* bytesparse.base.MutableMemory.clear()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.clear(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

classmethod convert(*source*, *meta=True*)

Converts a file object to another format.

It copies the *memory* and *meta* of the *source* file object, creating a new one of the target `BaseFile` format type.

Parameters

- **source** (`BaseFile`) – Source file object to convert.
- **meta** (*bool*) – Copy *meta* information to the target file object. Only the keys of the target *META_KEYS* are processed.

Returns

`BaseFile` – Converted copy of *source* to the target format.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> source = IhexFile.from_blocks(blocks, startaddr=789)
>>> target = SrecFile.convert(source)
>>> target.memory is source.memory
False
>>> target.memory == source.memory
True
>>> target.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 789}
```

copy(*start=None*, *endex=None*, *meta=True*)

Copies within a range.

It copied data within the specified range of the file object, creating a new one carrying the inner slice.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns
BaseFile – *self*.

See also:

memory *get_meta()* *discard_records()* bytesparse.base.MutableMemory.cut()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.copy(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

crop(*start=None*, *endex=None*)

Clears data outside a range.

It clears outside the specified range of underlying *memory* object, timming it.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
BaseFile – *self*.

See also:

memory *discard_records()* bytesparse.base.MutableMemory.crop()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.crop(start=124, endex=132)
>>> file.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
```

cut(*start=None*, *endex=None*, *meta=False*)

Cuts data within a range.

It takes data within the specified range away from the file object, creating a new one carrying the inner slice. The inner slice is cleared from *self*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

`BaseFile` – *self*.

See also:

memory *clear()* *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.cut(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

delete(*start=None*, *endex=None*)

Deletes data within a range.

It deletes the specified range of underlying *memory* object, shifting all subsequent data towards the collapsed range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.delete()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.delete(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'az']]
```

discard_memory()

Discards underlying memory.

The underlying *memory* object is assigned `None`.

If the underlying *records* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_bytes(b'abc', offset=123)
>>> _ = file.update_records()
>>> _ = file.discard_memory()
>>> _ = file.update_records()
Traceback (most recent call last):
    ...
ValueError: memory instance required
```

discard_records()

Discards underlying records.

The underlying *records* object is assigned `None`.

If the underlying *memory* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
```

(continues on next page)

(continued from previous page)

```
>>> _ = file.validate_records()
>>> _ = file.discard_records()
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: records required
```

extend(*other*)

Concatenates data.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> _ = file1.extend(file2)
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> _ = file1.extend(b'789')
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

fill(*start=None*, *endex=None*, *pattern=0*)

Fills a range.

It writes a *pattern* of bytes onto the underlying *memory* object, overwriting anything within the specified range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for filling.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.fill()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.fill(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'a........z']]
```

find(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it returns -1.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
int – *item* beginning address; -1 if not found.

See also:

index bytesparse.base.ImmutableMemory.find()

Notes

The internal *memory* might allow negative addresses for its stored data. In that case, *index()* would be more appropriate, because it raises an exception when the *item* is not found.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.find(b'yz')
457
>>> file.find(ord('b'))
```

(continues on next page)

(continued from previous page)

```
124
>>> file.find(b'?')
-1
```

flood(*start=None*, *endex=None*, *pattern=0*)

Floods a range.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.flood()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.get_holes()
[(126, 130)]
>>> _ = file.flood(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'abc....xyz']]
```

classmethod from_blocks(*blocks*, ***meta*)

Creates a file object from a memory object.

The *blocks* are put into the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **blocks** (*list of blocks*) – Memory blocks to put into *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

`BaseFile` – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_blocks()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_bytes(*data*, *offset=0*, ***meta*)

Creates a file object from a byte string.

The byte string makes a single *data* block, placed at some offset within the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **data** (*bytes*) – A byte string used to make a single data block.
- **offset** (*int*) – Offset of the single data block within *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
`BaseFile` – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_bytes()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
8
```

classmethod from_memory(*memory=None*, ***meta*)

Creates a file object from a memory object.

The *memory* is set as the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **memory** (bytesparse.base.MutableMemory) – Memory object to set as *memory*. If None, an empty memory object is automatically created.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

BaseFile – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS bytesparse.base.MutableMemory

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from bytesparse import Memory
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> memory = Memory.from_blocks(blocks)
>>> from hexrec import SrecFile
>>> file = SrecFile.from_memory(memory, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_records(*records*, *maxdatalen=None*)

Creates a file object from records.

The *records* sequence is set as the record attribute of the created file object.

This method creates a file object in *records role*. This means that only its *records* is internally instanced, while the *memory* requires manual or lazy instancing (i.e. either via direct call to *apply_records()*, or any other methods indirectly calling it).

Parameters

- **records** (list of BaseRecord) – Record sequence to set as *records*.
- **maxdatalen** (Optional[int]) – Maximum record *data* field size. If None, the maximum non-zero size of the *data* field from the *records* sequence is used. If all the *records* have zero sized *data* field, the class attribute *DEFAULT_DATALEN* is used.

Returns

BaseFile – The created file object.

Raises
ValueError – invalid *meta* values.

See also:

`BaseRecord`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
3
```

get_address_max()

Maximum address within memory.

It returns the maximum address of the underlying *memory* object.

Returns
int – Maximum address.

See also:

`bytesparse.base.ImmutableMemory.endin`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_max()
458
```

get_address_min()

Minimum address within memory.

It returns the minimum address of the underlying *memory* object.

Returns
int – Minimum address.

See also:

`bytesparse.base.ImmutableMemory.start`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_min()
123
```

get_holes()

List of memory holes.

It scans the underlying *memory* and returns the list of memory holes/gaps.

Each hole is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory hole boundaries.

See also:

`bytesparse.base.ImmutableMemory.gaps()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_holes()
[(126, 456), (459, 789)]
```

get_meta()

Meta information.

It builds and returns a dictionary of *meta* information. Meta keys are taken from the *META_KEYS* class attribute.

Returns
dict – Meta information dictionary.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks, header=b'HDR\0')
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 0}
```

`get_spans`()

List of memory block spans.

It scans the underlying *memory* and returns the list of memory block spans/intervals.

Each span is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory block boundaries.

See also:

`bytesparse.base.ImmutableMemory.intervals()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_spans()
[(123, 126), (456, 459), (789, 791)]
```

`index`(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it raises `ValueError`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
int – *item* beginning address.

Raises
ValueError – *item* not found.

See also:

find `bytesparse.base.ImmutableMemory.index()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.index(b'yz')
457
>>> file.index(ord('b'))
124
>>> file.index(b'?')
Traceback (most recent call last):
    ...
ValueError: subsection not found
```

property linear: bool

Linear addressing.

This property sets the linear addressing mode (the default).

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

When true, *records* are generated to support *linear addressing*, i.e. full 32-bit addressing. Each *data* record holds the 16-bit *offset*, which is the lower part of a 32-bit address, while *Extended Linear Address* records set the upper 16-bit *segment*.

When false, *records* are generated to support *segment addressing*, i.e. 20-bit addressing. Each *data* record holds the 16-bit *offset*, which is the lower part of a 20-bit address, while *Extended Segment Address* records set a 16-bit added value, as bits *20:4* of the address.

Examples

```
>>> from hexrec import IhexFile
>>> blocks = [[0x1234, b'abc'], [0x000F4321, b'xyz']]
>>> file = IhexFile.from_blocks(blocks)
>>> file.linear
True
>>> _ = file.print()
:0312340061626391
:02000004000FEB
:0343210078797A2E
:00000001FF
>>> file.linear = False
>>> _ = file.print()
:0312340061626391
:02000002F0000C
:0343210078797A2E
:00000001FF
```

Type
bool

classmethod load(*path*, **args*, ***kwargs*)

Loads a file object from the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *parse()* to load a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdin.buffer` is used.
- **args** – Forwarded to *parse()*.
- **kwargs** – Forwarded to *parse()*.

Returns

`BaseFile` – Loaded file object.

See also:

save() *parse()* `open()` `sys.stdin.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.load('data.hex')
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

property maxdatalen: int

Maximum byte size of the data field.

This property sets the maximum byte size of the *data* field of a serialized record.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises

ValueError – Invalid maximum data length.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> file.maxdatalen
16
>>> _ = file.print()
```

(continues on next page)

(continued from previous page)

```
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> file.maxdatalen = 8
>>> _ = file.print()
S0030000FC
S10B000000010203040506 07D8
S10B000808090A0B0C0D0E 0F90
S10B001010111213141516 1748
S10B001818191A1B1C1D1E 1F00
S10B002020212223242526 27B8
S10B002828292A2B2C2D2E 2F70
S10B003030313233343536 3728
S10B003838393A3B3C3D3E 3FE0
S5030008F4
S9030000FC
>>> file.maxdatalen = 0
Traceback (most recent call last):
    ...
ValueError: invalid maximum data length
```

Type
int

property memory: MutableMemory

Memory object stored by records role.

This readonly property exposes the memory object stored by the file object while in *memory role*.

If this property is accessed while the file object is not in *memory role*, it automatically activates it by an implicit call to *apply_records()*, with default arguments.

For more control activating the *memory role*, please call *apply_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *records role* (i.e. altering content of *records*) would implicitly discard *memory* via *discard_memory()*.

See also:

apply_records() *discard_memory()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> _ = file.write(789, b'?!')
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz'], [789, b'?!']]
```

Type
`bytesparse.Memory`

merge(**files*, *clear=False*)

Merges data onto the file.

It writes the provided *files* onto *self*, in the provided order. Any common address ranges are overwritten.

Any stored *records* are discarded upon return.

Parameters

- **files** (`BaseFile`) – Files to merge.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns
`BaseFile` – *self*.

See also:

clear() discard_records() write()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = SrecFile.from_bytes(b'<<<?????>>>', offset=450)
>>> _ = file3.merge(file1, file2)
>>> file3.memory.to_blocks()
[[123, b'abc'], [450, b'<<<???xyz>>']]
```

classmethod parse(*stream*, *ignore_errors=False*, *ignore_after_termination=True*)

Parses records from a byte stream.

It executes `BaseRecord.parse()` for each line of the incoming *stream*, creating a new file object with the collected records calling *from_records()*.

Lines resulting empty by `_is_empty_line()` are just discarded.

Notes

Please refer to the actual implementation of each record file *format*, because it may be more specialized.

Parameters

- **stream** (*bytes IO or buffer*) – Stream or byte buffer to parse records from.
- **ignore_errors** (*bool*) – Ignore `Exception` raised by `BaseRecord.parse()`.
- **ignore_after_termination** (*bool*) – Ignore anything after the termination record was parsed, if supported (e.g. *End Of File* or *start address* record, depending on the specific file *format*).

Returns

`BaseFile` – *self*.

See also:

parse() `BaseRecord.parse()` *from_records()* `_is_empty_line()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> buffer = b'''
...     :03DA7A0061626383
...     :040000050000CAFE2F
...     :00000001FF
... '''
>>> import io
>>> stream = io.BytesIO(buffer)
>>> file = IhexFile.parse(stream)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
>>> file = IhexFile.parse(buffer)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

print(**args*, *stream=None*, *color=False*, *start=None*, *stop=None*, ***kwargs*)

Prints record content to stdout.

This helper method prints each record of *records* via `BaseRecord.print()`. As such, it also supports colored tokens and streams different from *stdout*.

It is possible to print subset of the records by specifying the record index range.

> **Warning:** This method is **NOT** equivalent to *serialize()*, because it just prints each record from *records*. Please use *serialize()* for an actual serialization of the whole file.

Parameters

- **args** – Forwarded to the underlying call to `to_tokens()`.
- **stream** (*byte stream*) – Stream to print onto. If `None`, *stdout* is used.
- **color** (*bool*) – Colorize record tokens with ANSI color codes.
- **start** (*int*) – Inclusive start record index of the specified range. If `None`, start from the first record.
- **stop** (*int*) – Exclusive end record index of the specified range. If negative, look back from the last index. If `None`, print up to the last record.
- **kwargs** – Forwarded to the underlying call to `to_tokens()`.

Returns

`BaseFile` – *self*.

See also:

`BaseRecord.print()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> _ = file.print(color=True, start=1, stop=-2)
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
```

read(*start=None*, *endex=None*, *fill=0*)

Extracts a substring.

It extracts a byte string from the specified range, filling any memory holes/gaps (without altering *memory*).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **fill** (*bytes or int*) – Byte pattern for filling.

Returns
BaseFile – *self*.

See also:

memory bytesparse.base.MutableMemory.extract() bytesparse.base.MutableMemory.to_bytes()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.read(start=124, endex=132)
b'bc\x00\x00\x00\x00xy'
>>> file.read(start=124, endex=132, fill=b'.')
b'bc....xy'
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

property records: MutableSequence[*BaseRecord*]

Records stored by records role.

This readonly property exposes the list of records stored by the file object while in *records role*.

If this property is accessed while the file object is not in *records role*, it automatically activates it by an implicit call to *update_records()*, with default arguments.

For more control activating the *records role*, please call *update_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *memory role* (i.e. altering content of *memory*) would implicitly discard *records* via *discard_records()*.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, startaddr=789)
>>> len(file.records)
5
>>> _ = file.print()
S0030000FC
S106007B61626358
```

(continues on next page)

(continued from previous page)

```
S10601C878797AC5
S5030002FA
S9030315E4
>>> _ = file.update_records(data_tag=SrecFile.Record.Tag.DATA_32)
>>> _ = file.print()
S0030000FC
S3080000007B61626356
S308000001C878797AC3
S5030002FA
S70500000315E2
```

Type
list of `BaseRecord`

`save`(*path*, **args*, ***kwargs*)

Saves a file object into the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *`serialize()`* to save a file object.

Parameters

- **`path`** (*`str`*) – Path of the file within the filesystem. If `None`, `sys.stdout.buffer` is used.
- **`args`** – Forwarded to *`serialize()`*.
- **`kwargs`** – Forwarded to *`serialize()`*.

Returns
`BaseFile` – *self*.

See also:

`load()` *`serialize()`* `open()` `sys.stdout.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> _ = file.save('data.hex')
```

`serialize`(*stream*, **args*, ***kwargs*)

Serializes records onto a byte stream.

It executes `BaseRecord.serialize()` for each of the stored *`records`*.

Parameters

- **`stream`** (*`bytes IO`*) – Stream to serialize records onto.
- **`args`** – Forwarded to `BaseRecord.serialize()` of each record.
- **`kwargs`** – Forwarded to `BaseRecord.serialize()` of each record.

Returns
`BaseFile` – *self*.

See also:

parse() BaseRecord.serialize()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> import sys
>>> _ = file.serialize(sys.stdout.buffer, end=b'\n')
:03DA7A0061626383
:040000050000CAFE2F
:00000001FF
```

set_meta(*meta*, *strict=True*)

Sets meta information.

It sets the provided *kwargs* to their matching *meta* attributes, as listed by *META_KEYS*.

Parameters

- **meta** (*dict*) – Mapping of the *meta* information to set.
- **strict** (*bool*) – All the keys within *meta* must exist within *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *get_meta()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 0}
>>> _ = file.set_meta(dict(header=b'HDR\0', startaddr=456))
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 456}
```

shift(*offset*)

Shifts data addresses by an offset.

It shifts addresses of the underlying *memory* object data blocks by the provided *offset* amount.

Any stored *records* are discarded upon return.

Parameters
offset (*int*) – Offset to apply to the underlying data block addresses.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.shift()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> _ = file.shift(1000)
>>> file.memory.to_blocks()
[[1123, b'abc'], [1456, b'xyz']]
```

split(**addresses*, *meta=True*)

Splits into parts.

The provided *addresses* are sorted and used as markers to split *self* into parts.

Each part is the *copy()* of *self* within the range of that part, in *memory role* (i.e., *records* is not populated).

Parameters

- **addresses** (*int*) – Split points.
- **meta** (*bool*) – Each part inherits *meta* from *self*.

Returns
list of BaseFile – Parts after splitting.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'Hello, World!', offset=123)
>>> parts = file.split(128, 130)
>>> for part in parts: print(part.memory.to_blocks())
[[123, b'Hello']]
[[128, b', ']]
[[130, b'World!']]
>>> file.memory.to_blocks()
[[123, b'Hello, World!']]
```

property startaddr: int | None

Start address.

This property sets the *start address* of the serialized record file.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

If provided, the start address is stated either by a *Start Linear Address* record when *linear* is true, or by a *Start Segment Address* record when *linear* is false.

If None (the default), no start address record is generated.

Examples

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> file.startaddr is None
True
>>> _ = file.print()
:00000001FF
>>> file.startaddr = 0x87654321
>>> _ = file.print()
:040000587654321A7
:00000001FF
>>> file.linear = False
>>> _ = file.print()
:040000387654321A9
:00000001FF
```

update_records(*align=False*, *start=True*)

Applies memory and meta to records.

This method processes the stored *memory* and *meta* information to generate the sequence of *records*.

This effectively converts the *memory role* into the *records role* (keeping both).

The *records* is assigned upon return. Any exceptions being raised should not alter the file object.

Parameters

- **align** (*bool*) – Aligns data record chunk address bounds to *maxdatalen*.
- **start** (*bool*) – Generates the *start address* record, if *startaddr* is not None.

Returns

IhexFile – *self*.

Raises

ValueError – *memory* attribute not populated.

See also:

records *memory* *get_meta()* *apply_records()*

Examples

```
>>> from hexrec import IhexFile
>>> blocks = [[123, b'abc']]
>>> file = IhexFile.from_blocks(blocks, maxdatalen=16, startaddr=456)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
>>> _ = file.update_records()
>>> len(file.records)
3
>>> _ = file.print()
:03007B006162635C
:04000005000001C82E
:00000001FF
```

validate_records(*data_ordering=False*, *start_required=False*, *start_penultimate=True*, *start_within_data=False*)

Validates records.

It performs consistency checks for the underlying *records*.

Parameters

- **data_ordering** (*bool*) – Checks that the *data* record sequence has monotonically increasing addresses, without any overlapping.
- **start_required** (*bool*) – Requires the *start address* record be present.
- **start_penultimate** (*bool*) – Requires the *start address* record be the penultimate one.
- **start_within_data** (*bool*) – Requires *start address* fall within data carried by some *data* record.

Returns

IhexFile – *self*.

Raises

ValueError – Invalid record sequence.

Examples

```
>>> from hexrec import IhexFile
>>> records = [IhexFile.Record.create_data(123, b'abc')]
>>> file = IhexFile.from_records(records)
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: missing end of file record
```

view(*start=None*, *endex=None*)

Memory view.

It returns a `memoryview` over the specified range, which must cover a *contiguous* data region (i.e. no memory holes within).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

memoryview – View of the specified range.

Raises

ValueError – non-contiguous data within range.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> bytes(file.view(start=456, endex=458))
b'xy'
>>> bytes(file.view())
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

write(*address*, *data*, *clear=False*)

Writes data into the file.

It writes the provided *data* into the underlying *memory* object.

Any stored *records* are discarded upon return.

Parameters

- **address** (*int*) – Address where *data* has to be written.
- **data** (*bytes or memory*) – Byte data to write.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

`BaseFile` – *self*.

See also:

memory *clear()* *discard_records()* `bytesparse.base.MutableMemory.write()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> _ = file.write(123, b'abc')
>>> _ = file.write(555, ord('?'))
>>> _ = file.write(1000, SrecFile.from_bytes(b'xyz', offset=456))
>>> file.memory.to_blocks()
[[123, b'abc'], [555, b'?'], [1456, b'xyz']]
```

4.5.2 IhexRecord

class hexrec.formats.ihex.**IhexRecord**(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

Intel HEX record object.

Attributes

EQUALITY_KEYS	Meta keys for equality checks.
LINE_REGEX	Line parser regex.
META_KEYS	Meta keys.

Methods

__init__	
compute_checksum	Computes the checksum field value.
compute_count	Compute the count field value.
copy	Shallow copy.
create_data	Creates a data record.
create_end_of_file	Creates an End Of File record.
create_extended_linear_address	Creates an Extended Linear Address record.
create_extended_segment_address	Creates an Extended Segment Address record.
create_start_linear_address	Creates a Start Linear Address record.
create_start_segment_address	Creates a Start Segment Address record.
data_to_int	Interprets data bytes as integer.
get_meta	Gets meta information.
parse	Parses a record from bytes.
print	Prints a record.
serialize	Serializes onto a stream.
to_bytestr	Converts into a byte string.
to_tokens	Converts into byte string tokens.
update_checksum	Updates the checksum field.
update_count	Updates the count field.
validate	Validates consistency of attribute values.

EQUALITY_KEYS: Sequence[str] = ['address', 'checksum', 'count', 'data', 'tag']

Meta keys for equality checks.

Equality methods (*__eq__()* and *__ne__()*) check against these *meta* keys only. Any other *meta* keys are just ignored.

LINE_REGEX = re.compile(b'^(?P<before>[^:]*):(?P<count>[0-9A-Fa-f]{2})(?P<address>[0-9A-Fa-f]{4})(?P<tag>[0-9A-Fa-f]{2})(?P<data>([0-9A-Fa-f]{2}){,255})(?P<checksum>[0-9A-Fa-f]{2})(?P<after>[^\\r\\n]*)\\r?\\n?$')

Line parser regex.

META_KEYS: Sequence[str] = ['address', 'after', 'before', 'checksum', 'coords', 'count', 'data', 'tag']

Meta keys.

This sequence holds the *meta* keys for copying (see *copy()*).

Tag

alias of *IhexTag*

__bytes__()

Serializes the record into bytes.

Returns
bytes – Byte serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> bytes(record)
b':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> bytes(record)
b'abc'
```

__eq__(*other*)

Equality test.

This method returns true if *self* is considered equal to *other*.

As inequality is usually easier to check, this method is usually implemented as a trivial `not self != other` (*__ne__()*).

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* equals *other*.

See also:

__ne__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 == ihex2
True
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 == ihex3
False
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 == raw
False
```

__hash__ = None

__init__(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

__ne__(*other*)

Ineuality test.

This method returns true if *self* is considered unequal to *other*.

Each attribute listed by *EQUALITY_KEYS* is compared between *self* and *other*. This method returns whether any attributes do not match.

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* and *other* are unequal.

See also:

__eq__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
```

(continues on next page)

(continued from previous page)

```
>>> ihex1 != ihex2
False
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 != ihex3
True
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 != raw
True
```

__repr__()

String representation.

It returns a string representation of the record content, for human understanding only.

Returns

str – String representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> repr(record)
"<<class 'hexrec.formats.ihex.IhexRecord'> @...
  address:=0 after:=b'' before:=b'' checksum:=255 coords:=(-1, -1)
  count:=0 data:=b'' tag:=<IhexTag.END_OF_FILE: 1>>"
```

__str__()

Serializes the record into a string.

Returns

str – String serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> str(record)
'abc'
```

__weakref__

list of weak references to the object (if defined)

compute_checksum()

Computes the checksum field value.

It computes and returns the format-specific `checksum` value of a record.

When not specialized, it returns `None` by default.

Returns

int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_checksum()
215
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_checksum())
'None'
```

compute_count()

Compute the count field value.

It computes and returns the format-specific `count` value of a record.

When not specialized, it returns `None` by default.

Returns

int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_count()
3
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_count())
'None'
```

copy(*validate=True*)

Shallow copy.

It calls the record constructor, passing *meta* to it.

Parameters
validate (*bool*) – Performs validation on instantiation (*__init__()*).

Returns
BaseRecord – Shallow copy.

See also:

__init__() *get_meta()*

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record1 = IhexFile.Record.create_data(0x1234, b'abc')
>>> record2 = record1.copy()
>>> record1 is record2
False
>>> record1 == record2
True
```

classmethod create_data(*address*, *data*)

Creates a data record.

This is a mandatory class method to instantiate a *data* record.

Parameters

- **address** (*int*) – Record address. If not supported, set zero.
- **data** (*bytes*) – Record byte data.

Returns
BaseRecord – Data record object.

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> str(record)
':0312340061626391\r\n'
```

classmethod create_end_of_file()

Creates an End Of File record.

Returns
IhexRecord – End Of File record object.

Examples

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

classmethod create_extended_linear_address(*extension*)

Creates an Extended Linear Address record.

Parameters
extension (*int*) – Address extension value.

Returns
IhexRecord – Extended Linear Address record object.

Examples

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0x1234)
>>> str(record)
':020000041234B4\r\n'
```

classmethod create_extended_segment_address(*extension*)

Creates an Extended Segment Address record.

Parameters
extension (*int*) – Address extension value.

Returns
IhexRecord – Extended Segment Address record object.

Examples

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_segment_address(0x1234)
>>> str(record)
':020000021234B6\r\n'
```

classmethod create_start_linear_address(*address*)

Creates a Start Linear Address record.

Parameters
address (*int*) – Start address.

Returns
IhexRecord – Start Linear Address record object.

Examples

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_start_linear_address(0x12345678)
>>> str(record)
':0400000512345678E3\r\n'
```

classmethod create_start_segment_address(*address*)

Creates a Start Segment Address record.

Parameters
address (*int*) – Start address.

Returns
IhexRecord – Start Segment Address record object.

Examples

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_start_segment_address(0x12345678)
>>> str(record)
':0400000312345678E5\r\n'
```

data_to_int(*byteorder='big'*, *signed=False*)

Interprets data bytes as integer.

It creates an integer from bytes of the `data` field.

Parameters

- **byteorder** (*'big' or 'little'*) – Byte order (endianness): either `'big'` (default) or `'little'`.
- **signed** (*bool*) – Signed integer (2-complement); default false.

Returns
int – Interpreted integer value.

See also:

`int.from_bytes()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0xABCD)
>>> record.data
b'\xab\xcd'
>>> addrext = record.data_to_int()
>>> addrext, hex(addrext)
(43981, '0xabcd')
```

get_meta()

Gets meta information.

It returns all the object attributes whose keys are listed by *META_KEYS*.

Returns
dict – Attribute values listed by *META_KEYS*.

See also:

META_KEYS set_meta()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> record.get_meta()
{'address': 0, 'after': b'', 'before': b'', 'checksum': 255,
 'coords': (-1, -1), 'count': 0, 'data': b'',
 'tag': <IhexTag.END_OF_FILE: 1>}
```

classmethod parse(*line*, *validate=True*)

Parses a record from bytes.

Please refer to the actual implementation provided by the record *format* for more details.

Parameters

- **line** (*bytes*) – String of bytes to parse.
- **validate** (*bool*) – Perform validation checks.

Returns
BaseRecord – Parsed record.

Raises
ValueError – Syntax error.

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.parse(b':00000001FF\r\n')
>>> record.tag
<IhexTag.END_OF_FILE: 1>
>>> IhexFile.Record.parse(b'::00000001FF\r\n')
Traceback (most recent call last):
    ...
ValueError: syntax error
```

print(**args*, *stream=None*, *color=False*, ***kwargs*)

Prints a record.

The record is converted into tokens (eventually colorized) then joined and written onto a byte stream (*stdout* by default).

Parameters

- **args** – Forwarded to the underlying call to *to_tokens()*.
- **stream** (io.BytesIO) – The byte stream where the record tokens are printed. If None, *stdout* is selected.
- **color** (*bool*) – Tokens are colorized before printing.
- **kwargs** – Forwarded to the underlying call to *to_tokens()*.

Returns

BaseRecord – *self*.

See also:

to_tokens() colorize_tokens() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> _ = record.print()
:0312340061626391
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.print(stream=stream, color=True)
>>> stream.getvalue()
b'\x1b[0m\x1b[33m:\x1b[34m03\x1b[31m1234\x1b[32m00\x1b[36m61\x1b[96m62\
↪x1b[36m63\x1b[35m91\x1b[0m\r\n\x1b[0m'
```

serialize(*stream*, **args*, ***kwargs*)

Serializes onto a stream.

This wraps a call to *to_bytestr()* and stream.write.

Parameters

- **stream** (io.BytesIO) – Stream to write.
- **args** – Forwarded to *to_bytestr()*.
- **kwargs** – Forwarded to *to_bytestr()*.

Returns

BaseRecord – *self*.

See also:

to_bytestr() io.BytesIO

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.serialize(stream, end=b'\n')
>>> stream.getvalue()
b':0312340061626391\n'
```

`to_bytestr`(*end=b'\\r\\n'*)

Converts into a byte string.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Byte string representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_bytestr(end=b'\n')
b':0312340061626391\n'
```

`to_tokens`(*end=b'\\r\\n'*)

Converts into byte string tokens.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Mapping of token keys to token byte strings.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_tokens(end=b'\n')
{'before': b'', 'begin': b':', 'count': b'03', 'address': b'1234',
 'tag': b'00', 'data': b'616263', 'checksum': b'91', 'after': b'',
 'end': b'\n'}
```

update_checksum()

Updates the checksum field.

It updates the `checksum` attribute, assigning to it the value returned by *compute_checksum()*.

Returns
`BaseRecord` – *self*.

See also:

`checksum` *compute_checksum()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> record = IhexRecord(IhexRecord.Tag.END_OF_FILE, checksum=None)
>>> record.compute_checksum()
255
>>> record.checksum is None
True
>>> _ = record.update_checksum()
>>> record.checksum
255
```

update_count()

Updates the count field.

It updates the `count` attribute, assigning to it the value returned by *compute_count()*.

Returns
`BaseRecord` – *self*.

See also:

`count` *compute_count()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> Record = IhexFile.Record
>>> Tag = Record.Tag
>>> record = Record(Tag.DATA, data=b'abc', count=None, checksum=None)
>>> record.compute_count()
3
>>> record.count is None
True
>>> _ = record.update_count()
>>> record.count
3
```

validate(*checksum=True*, *count=True*)

Validates consistency of attribute values.

All the record attributes are checked for consistency.

Please refer to the implementation for more details.

Parameters

- **checksum** (*bool*) – Check the consistency of the `checksum` attribute.
- **count** (*bool*) – Check the consistency of the `count` attribute.

Returns

`BaseRecord` – *self*.

Raises

ValueError – Some targeted attributes are inconsistent.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> _ = record.validate()
>>> record.data = b'abc'
>>> _ = record.update_count().update_checksum().validate()
Traceback (most recent call last):
    ...
ValueError: unexpcted data
```

4.5.3 IhexTag

class hexrec.formats.ihex.**IhexTag**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Intel HEX tag.

Attributes

DATA	Binary data.
END_OF_FILE	End Of File.
EXTENDED_SEGMENT_ADDRESS	Extended Segment Address.
START_SEGMENT_ADDRESS	Start Segment Address.
EXTENDED_LINEAR_ADDRESS	Extended Linear Address.
START_LINEAR_ADDRESS	Start Linear Address.
denominator	the denominator of a rational number in lowest terms
imag	the imaginary part of a complex number
numerator	the numerator of a rational number in lowest terms
real	the real part of a complex number

Methods

is_eof	Tells whether this is an End Of File record tag.
is_extension	Tells whether this is an Extended Address record tag.
is_start	Tells whether this is a Start Address record tag.
__init__	
as_integer_ratio	Return integer ratio.
bit_count	Number of ones in the binary representation of the absolute value of self.
bit_length	Number of bits necessary to represent self in binary.
conjugate	Returns self, the complex conjugate of any int.
from_bytes	Return the integer represented by the given array of bytes.
to_bytes	Return an array of bytes representing an integer.

DATA = 0

Binary data.

END_OF_FILE = 1

End Of File.

EXTENDED_LINEAR_ADDRESS = 4

Extended Linear Address.

EXTENDED_SEGMENT_ADDRESS = 2

Extended Segment Address.

START_LINEAR_ADDRESS = 5

Start Linear Address.

START_SEGMENT_ADDRESS = 3

Start Segment Address.

_DATA: Optional[*BaseTag*] = 0

Alias to a common data record tag.

This tag is used internally to build a generic data record.

__abs__()

abs(self)

__add__(*value*, /)

Return self+value.

__and__(*value*, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

classmethod __contains__(*member*)

Return True if member is a member of this enum raises TypeError if member is not an enum member

note: in 3.12 TypeError will no longer be raised, and True will also be returned if member is the value of a member in this enum

__dir__()

Returns all members and all public methods

__divmod__(*value*, /)

Return divmod(self, value).

__eq__(*value*, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(*value*, /)

Return self//value.

__format__(*format_spec*, /)

Default object formatter.

__ge__(*value*, /)

Return self>=value.

__getattribute__(*name*, /)

Return getattr(self, name).

classmethod __getitem__(*name*)

Return the member matching *name*.

__gt__(*value*, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__init__(**args*, ***kwds*)

__int__()
int(self)

__invert__()
~self

classmethod __iter__()
Return members in definition order.

__le__(*value*, /)
Return self<=value.

classmethod __len__()
Return the number of members (no aliases)

__lshift__(*value*, /)
Return self<<value.

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

__mul__(*value*, /)
Return self*value.

__ne__(*value*, /)
Return self!=value.

__neg__()
-self

__new__(*value*)

__or__(*value*, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod=None*, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__reduce_ex__(*proto*)
Helper for pickle.

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod=None*, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(*value*, /)
Return self-value.

__truediv__(*value*, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(*value*, /)

Return self^value.

_generate_next_value_(*start*, *count*, *last_values*)

Generate the next value when not given.

name: the name of the member start: the initial start value or None count: the number of existing members last_values: the list of values assigned

_member_type_

alias of `int`

_new_member_(***kwargs*)

Create and return a new object. See help(type) for accurate signature.

_value_repr_()

Return repr(self).

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big'*, ***, *signed=False*)

Return the integer represented by the given array of bytes.

bytes
Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

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. Default is to use 'big'.

signed
Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_eof()

Tells whether this is an End Of File record tag.

This method returns true if this record tag is used for *End Of File* records.

Returns
bool – This is an End Of File record tag.

Examples

```
>>> from hexrec import IhexFile
>>> IhexTag = IhexFile.Record.Tag
>>> IhexTag.END_OF_FILE.is_eof()
True
>>> IhexTag.DATA.is_eof()
False
```

is_extension()

Tells whether this is an Extended Address record tag.

This method returns true if this record tag is used for *address extension* records.

Returns
bool – This is an Extended Address record tag.

Examples

```
>>> from hexrec import IhexFile
>>> IhexTag = IhexFile.Record.Tag
>>> IhexTag.EXTENDED_LINEAR_ADDRESS.is_extension()
True
>>> IhexTag.EXTENDED_SEGMENT_ADDRESS.is_extension()
True
>>> IhexTag.DATA.is_extension()
False
```

is_start()

Tells whether this is a Start Address record tag.

This method returns true if this record tag is used for *start address* records.

Returns

bool – This is a Start Address record tag.

Examples

```
>>> from hexrec import IhexFile
>>> IhexTag = IhexFile.Record.Tag
>>> IhexTag.START_LINEAR_ADDRESS.is_start()
True
>>> IhexTag.START_SEGMENT_ADDRESS.is_start()
True
>>> IhexTag.DATA.is_extension()
False
```

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1*, *byteorder='big'*, ***, *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. Default is length 1.

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. Default is to use 'big'.

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.

4.6 mos

MOS Technology format.

See also:

https://srecord.sourceforge.net/man/man5/srec_mos_tech.5.html

Classes

MosFile	MOS Technology file object.
MosRecord	MOS Technology record object.
MosTag	MOS Technology tag.

4.6.1 MosFile

class hexrec.formats.mos.**MosFile**

MOS Technology file object.

Attributes

DEFAULT_DATALEN	Default data attribute length.
FILE_EXT	Supported filename extensions.
META_KEYS	Meta information key names.
maxdatalen	Maximum byte size of the data field.
memory	Memory object stored by records role.
records	Records stored by records role.

Methods

__init__	
align	Pads blocks to align their boundaries.
append	Appends a byte.
apply_records	Applies records to memory and meta.
clear	Clears data within a range.
convert	Converts a file object to another format.
copy	Copies within a range.
crop	Clears data outside a range.
cut	Cuts data within a range.
delete	Deletes data within a range.
discard_memory	Discards underlying memory.
discard_records	Discards underlying records.
extend	Concatenates data.
fill	Fills a range.
find	Finds a substring.

continues on next page

Table 5 – continued from previous page

flood	Floods a range.
from_blocks	Creates a file object from a memory object.
from_bytes	Creates a file object from a byte string.
from_memory	Creates a file object from a memory object.
from_records	Creates a file object from records.
get_address_max	Maximum address within memory.
get_address_min	Minimum address within memory.
get_holes	List of memory holes.
get_meta	Meta information.
get_spans	List of memory block spans.
index	Finds a substring.
load	Loads a file object from the filesystem.
merge	Merges data onto the file.
parse	Parses records from a byte stream.
print	Prints record content to stdout.
read	Extracts a substring.
save	Saves a file object into the filesystem.
serialize	Serializes records onto a byte stream.
set_meta	Sets meta information.
shift	Shifts data addresses by an offset.
split	Splits into parts.
update_records	Applies memory and meta to records.
validate_records	Validates records.
view	Memory view.
write	Writes data into the file.

DEFAULT_DATALEN: int = 24

Default data attribute length.

Default value for the *maxdatalen* *meta*, which sets the maximum size of `BaseRecord.data` field values.

FILE_EXT: Sequence[str] = []

Supported filename extensions.

Sequence of file name extension substrings (e.g. `.hex`). This list is used by functions like `guess_format_name()` to manage mapping of file *formats*.

META_KEYS: Sequence[str] = ['maxdatalen']

Meta information key names.

Sequence of key strings listing the supported *meta* information of this file *format*.

Record

alias of *MosRecord*

__add__(*other*)

Concatenates with another file.

Equivalent to *copy()* then *extend()*.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – Concatenation of *self* and *other*.

See also:

copy() extend()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 + file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file4 = file3 + b'789'
>>> file4.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__bool__()

bool: Has data records or memory.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> bool(file)
False
>>> _ = file.append(0)
>>> bool(file)
True
```

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> file = IhexFile.from_records([IhexRecord.create_end_of_file()])
>>> bool(file)
False
>>> file.records.insert(0, IhexRecord.create_data(0, b'\0'))
>>> bool(file)
True
```

__delitem__(*key*)

Deletes a range.

Parameters

key (*slice or int*) – Range to delete.

See also:

`bytesparse.base.MutableMemory.__delitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> del file[457]
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xz']]
>>> del file[125:457]
>>> file.memory.to_blocks()
[[123, b'abz']]
```

__eq__(*other*)

Equality test.

The file objects *self* and *other* are considered *equal* if the inequality tests of *__ne__()* result false.

Returns

bool – *self* and *other* are *equal*.

See Also

__ne__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 == file2
True
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 == file3
False
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 == file4
False
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file == ihex_file
False
>>> srec_file.memory == ihex_file.memory
True
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__getitem__(*key*)

Extracts a range.

Parameters
key (*slice or int*) – Range to extract.

Raises
ValueError – invalid range.

See also:

bytesparse.base.ImmutableMemory.__getitem__()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> chr(file[457])
'y'
>>> repr(file[333])
'None'
>>> file[123:125]
b'ab'
>>> file[125:457]
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

__hash__ = None

__iadd__(*other*)

Concatenates data.

Equivalent to *extend()*.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – *self*.

See also:

memory *extend()* *discard_records()* bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 += file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file1 += b'789'
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__init__()

__ior__(*other*)

Merges with another file.

Equivalent to *merge()*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self*.

See also:

merge() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 |= file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file1 |= b'789'
>>> file1.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__ne__(*other*)

Inequality test.

The file objects *self* and *other* are considered *unequal* if any of the following tests result true:

- Both have *memory role* (i.e. *memory*), resulting unequal;
- Both have *records role* (i.e. *records*), resulting unequal;

- *other* does not have a *meta* listed by *META_KEYS*;
- A *meta* value (among those of *META_KEYS*) is different.

Returns
bool – *self* and *other* are *unequal*.

See also:

__eq__() memory records META_KEYS

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 != file2
False
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 != file3
True
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 != file4
True
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file != ihex_file
True
>>> srec_file.memory != ihex_file.memory
False
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__or__(*other*)

Merges with another file.

Equivalent to *copy()* then *merge()*.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self* merged with *other*.

See also:

copy() merge()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 | file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file4 = file3 | b'789'
>>> file4.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__setitem__(*key*, *value*)

Sets a range.

Parameters

- **key** (*slice or int*) – Range to set.
- **value** (bytes, `bytesparse.base.ImmutableMemory`, `None`) – Value(s) to set. `None` acts like *clear()*.

Raises

ValueError – invalid range.

See also:

`bytesparse.base.MutableMemory.__setitem__()` *clear()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file[124] = b'?'
>>> file.memory.to_blocks()
[[123, b'a?c'], [456, b'xyz']]
>>> file[:125] = None
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'xyz']]
>>> file[457:458] = b'789'
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'x789z']]
```

__weakref__

list of weak references to the object (if defined)

classmethod _is_line_empty(*line*)

Empty line check.

Tells whether a *line* has no meaningful content (e.g. all whitespace). The check itself depends on the implementing file *format*. It may be used internally to skip empty lines, e.g. by *parse()*.

Parameters
line (*bytes*) – A line, byte string.

Returns
bool: The *line* is empty.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexFile._is_line_empty(b'')
True
>>> IhexFile._is_line_empty(b' \t\v\r\n')
True
>>> IhexFile._is_line_empty(b':00000001FF\r\n')
False
```

align(*modulo*, *start=None*, *endex=None*, *pattern=0*)

Pads blocks to align their boundaries.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*, so that memory blocks are aligned to the required *modulo*.

Any stored *records* are discarded upon return.

Parameters

- **modulo** (*int*) – Alignment modulo.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns
`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.align()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [134, b'xyz']])
>>> _ = file.align(4, pattern=b'.')
>>> file.memory.to_blocks()
[[120, b'...abc..'], [132, b'..xyz...']]
```

append(*item*)

Appends a byte.

It appends the *item* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
item (*byte or int*) – Byte to append.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.append()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123)
>>> _ = file.append(b'.')
>>> _ = file.append(0)
>>> file.memory.to_blocks()
[[123, b'abc.\x00']]
```

apply_records()

Applies records to memory and meta.

This method processes the stored *records*, converting *data* as *memory*, and special records into their *meta* counterparts.

This effectively converts the *records role* into the *memory role* (keeping both).

The *memory* and *meta* are assigned upon return. Any exceptions being raised should not alter the file object.

Returns
BaseFile – *self*.

Raises
ValueError – *records* attribute not populated.

See also:

records memory get_meta() update_records()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_start_linear_address(456),
```

(continues on next page)

(continued from previous page)

```
...             IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records, maxdatalen=16)
>>> _ = file.apply_records()
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
```

clear(*start=None*, *endex=None*)

Clears data within a range.

It clears the specified range of underlying *memory* object, making a memory hole.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.clear()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.clear(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

classmethod convert(*source*, *meta=True*)

Converts a file object to another format.

It copies the *memory* and *meta* of the *source* file object, creating a new one of the target `BaseFile` format type.

Parameters

- **source** (`BaseFile`) – Source file object to convert.
- **meta** (*bool*) – Copy *meta* information to the target file object. Only the keys of the target *META_KEYS* are processed.

Returns

`BaseFile` – Converted copy of *source* to the target format.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> source = IhexFile.from_blocks(blocks, startaddr=789)
>>> target = SrecFile.convert(source)
>>> target.memory is source.memory
False
>>> target.memory == source.memory
True
>>> target.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 789}
```

copy(*start=None*, *endex=None*, *meta=True*)

Copies within a range.

It copied data within the specified range of the file object, creating a new one carrying the inner slice.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

`BaseFile` – *self*.

See also:

memory *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.copy(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

crop(*start=None*, *endex=None*)

Clears data outside a range.

It clears outside the specified range of underlying *memory* object, timming it.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.crop()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.crop(start=124, endex=132)
>>> file.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
```

cut(*start=None*, *endex=None*, *meta=False*)

Cuts data within a range.

It takes data within the specified range away from the file object, creating a new one carrying the inner slice. The inner slice is cleared from *self*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns
`BaseFile` – *self*.

See also:

memory *clear()* *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.cut(start=124, endex=132)
>>> inner.memory.to_blocks()
```

(continues on next page)

(continued from previous page)

```
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

delete(*start=None*, *endex=None*)

Deletes data within a range.

It deletes the specified range of underlying *memory* object, shifting all subsequent data towards the collapsed range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.delete()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.delete(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'az']]
```

discard_memory()

Discards underlying memory.

The underlying *memory* object is assigned `None`.

If the underlying *records* object is `None`, it is assigned a new empty memory object.

Returns

`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_bytes(b'abc', offset=123)
>>> _ = file.update_records()
>>> _ = file.discard_memory()
>>> _ = file.update_records()
Traceback (most recent call last):
    ...
ValueError: memory instance required
```

discard_records()

Discards underlying records.

The underlying *records* object is assigned `None`.

If the underlying *memory* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> _ = file.validate_records()
>>> _ = file.discard_records()
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: records required
```

extend(*other*)

Concatenates data.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.extend()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> _ = file1.extend(file2)
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> _ = file1.extend(b'789')
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

fill(*start=None*, *endex=None*, *pattern=0*)

Fills a range.

It writes a *pattern* of bytes onto the underlying *memory* object, overwriting anything within the specified range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.fill()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.fill(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'a.......z']]
```

find(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it returns `-1`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address; `-1` if not found.

See also:

index `bytesparse.base.ImmutableMemory.find()`

Notes

The internal *memory* might allow negative addresses for its stored data. In that case, *index()* would be more appropriate, because it raises an exception when the *item* is not found.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.find(b'yz')
457
>>> file.find(ord('b'))
124
>>> file.find(b'?')
-1
```

flood(*start=None*, *endex=None*, *pattern=0*)

Floods a range.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.flood()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.get_holes()
[(126, 130)]
>>> _ = file.flood(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'abc....xyz']]
```

classmethod from_blocks(*blocks*, ***meta*)

Creates a file object from a memory object.

The *blocks* are put into the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **blocks** (*list of blocks*) – Memory blocks to put into *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

`BaseFile` – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* `bytesparse.base.ImmutableMemory.from_blocks()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_bytes(*data*, *offset=0*, ***meta*)

Creates a file object from a byte string.

The byte string makes a single *data* block, placed at some offset within the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **data** (*bytes*) – A byte string used to make a single data block.
- **offset** (*int*) – Offset of the single data block within *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_bytes()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
8
```

classmethod **from_memory**(*memory=None*, ***meta*)

Creates a file object from a memory object.

The *memory* is set as the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **memory** (bytesparse.base.MutableMemory) – Memory object to set as *memory*. If None, an empty memory object is automatically created.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS bytesparse.base.MutableMemory

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from bytesparse import Memory
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> memory = Memory.from_blocks(blocks)
>>> from hexrec import SrecFile
>>> file = SrecFile.from_memory(memory, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_records(*records*, *maxdatalen=None*)

Creates a file object from records.

The *records* sequence is set as the `record` attribute of the created file object.

This method creates a file object in *records role*. This means that only its *records* is internally instanced, while the *memory* requires manual or lazy instancing (i.e. either via direct call to *apply_records()*, or any other methods indirectly calling it).

Parameters

- **records** (list of `BaseRecord`) – Record sequence to set as *records*.
- **maxdatalen** (`Optional[int]`) – Maximum record *data* field size. If `None`, the maximum non-zero size of the *data* field from the *records* sequence is used. If all the *records* have zero sized *data* field, the class attribute *DEFAULT_DATALEN* is used.

Returns

`BaseFile` – The created file object.

Raises

ValueError – invalid *meta* values.

See also:

`BaseRecord`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
3
```

get_address_max()

Maximum address within memory.

It returns the maximum address of the underlying *memory* object.

Returns
int – Maximum address.

See also:

`bytesparse.base.ImmutableMemory.endin`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_max()
458
```

get_address_min()

Minimum address within memory.

It returns the minimum address of the underlying *memory* object.

Returns
int – Minimum address.

See also:

`bytesparse.base.ImmutableMemory.start`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_min()
123
```

get_holes()

List of memory holes.

It scans the underlying *memory* and returns the list of memory holes/gaps.

Each hole is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory hole boundaries.

See also:

`bytesparse.base.ImmutableMemory.gaps()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_holes()
[(126, 456), (459, 789)]
```

get_meta()

Meta information.

It builds and returns a dictionary of *meta* information. Meta keys are taken from the *META_KEYS* class attribute.

Returns
dict – Meta information dictionary.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks, header=b'HDR\0')
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 0}
```

get_spans()

List of memory block spans.

It scans the underlying *memory* and returns the list of memory block spans/intervals.

Each span is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory block boundaries.

See also:

`bytesparse.base.ImmutableMemory.intervals()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_spans()
[(123, 126), (456, 459), (789, 791)]
```

index(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it raises `ValueError`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address.

Raises

ValueError – *item* not found.

See also:

find `bytesparse.base.ImmutableMemory.index()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.index(b'yz')
457
>>> file.index(ord('b'))
124
>>> file.index(b'?')
Traceback (most recent call last):
    ...
ValueError: subsection not found
```

classmethod load(*path*, **args*, ***kwargs*)

Loads a file object from the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *parse()* to load a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdin.buffer` is used.
- **args** – Forwarded to *parse()*.
- **kwargs** – Forwarded to *parse()*.

Returns

`BaseFile` – Loaded file object.

See also:

save() *parse()* `open()` `sys.stdin.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.load('data.hex')
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

property maxdatalen: int

Maximum byte size of the data field.

This property sets the maximum byte size of the *data* field of a serialized record.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises

ValueError – Invalid maximum data length.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> file.maxdatalen
16
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> file.maxdatalen = 8
>>> _ = file.print()
S0030000FC
S10B0000000102030405060 7D8
S10B000808090A0B0C0D0E0F90
S10B001010111213141516 1748
S10B001818191A1B1C1D1E1F00
S10B002020212223242526 27B8
S10B002828292A2B2C2D2E2F70
S10B003030313233343536 3728
```

(continues on next page)

(continued from previous page)

```
S10B003838393A3B3C3D3E3FE0
S5030008F4
S9030000FC
>>> file.maxdatalen = 0
Traceback (most recent call last):
    ...
ValueError: invalid maximum data length
```

Type
int

property memory: MutableMemory

Memory object stored by records role.

This readonly property exposes the memory object stored by the file object while in *memory role*.

If this property is accessed while the file object is not in *memory role*, it automatically activates it by an implicit call to *apply_records()*, with default arguments.

For more control activating the *memory role*, please call *apply_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *records role* (i.e. altering content of *records*) would implicitly discard *memory* via *discard_memory()*.

See also:

apply_records() *discard_memory()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> _ = file.write(789, b'?!')
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz'], [789, b'?!']]
```

Type
`bytesparse.Memory`

merge(**files*, *clear=False*)

Merges data onto the file.

It writes the provided *files* onto *self*, in the provided order. Any common address ranges are overwritten.

Any stored *records* are discarded upon return.

Parameters

- **files** (BaseFile) – Files to merge.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

BaseFile – *self*.

See also:

clear() discard_records() write()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = SrecFile.from_bytes(b'<<<?????>>>', offset=450)
>>> _ = file3.merge(file1, file2)
>>> file3.memory.to_blocks()
[[123, b'abc'], [450, b'<<<???xyz>>']]
```

classmethod parse(*stream*, *ignore_errors=False*, *ignore_after_termination=True*, *eof_record=True*)

Parses records from a byte stream.

It executes *MosRecord.parse()* for each line of the incoming *stream*, creating a new file object with the collected records calling *from_records()*.

Lines resulting empty by _is_empty_line() are just discarded.

Notes

Please refer to the actual implementation of each record file *format*, because it may be more specialized.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **ignore_errors** (*bool*) – Ignore Exception raised by *MosRecord.parse()*.
- **ignore_after_termination** (*bool*) – Ignore anything after the *End Of File* record was parsed.
- **eof_record** (*bool*) – Interpret the last record as the *End Of File* record.

Returns

MosFile – *self*.

See also:

parse() MosRecord.parse() from_records() _is_empty_line()

Examples

```
>>> from hexrec import MosFile
>>> buffer = b'''
...     ;031234616263016F
...     ;0000010001
... '''
>>> import io
>>> stream = io.BytesIO(buffer)
>>> file = MosFile.parse(stream)
>>> file.memory.to_blocks()
[[4660, b'abc']]
>>> file.get_meta()
{'maxdatalen': 3}
```

print(**args*, *stream=None*, *color=False*, *start=None*, *stop=None*, ***kwargs*)

Prints record content to stdout.

This helper method prints each record of *records* via `BaseRecord.print()`. As such, it also supports colored tokens and streams different from *stdout*.

It is possible to print subset of the records by specifying the record index range.

Warning: This method is **NOT** equivalent to *serialize()*, because it just prints each record from *records*. Please use *serialize()* for an actual serialization of the whole file.

Parameters

- **args** – Forwarded to the underlying call to `to_tokens()`.
- **stream** (*byte stream*) – Stream to print onto. If `None`, *stdout* is used.
- **color** (*bool*) – Colorize record tokens with ANSI color codes.
- **start** (*int*) – Inclusive start record index of the specified range. If `None`, start from the first record.
- **stop** (*int*) – Exclusive end record index of the specified range. If negative, look back from the last index. If `None`, print up to the last record.
- **kwargs** – Forwarded to the underlying call to `to_tokens()`.

Returns

`BaseFile` – *self*.

See also:

`BaseRecord.print()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> _ = file.print(color=True, start=1, stop=-2)
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
```

read(*start=None*, *endex=None*, *fill=0*)

Extracts a substring.

It extracts a byte string from the specified range, filling any memory holes/gaps (without altering *memory*).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **fill** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory `bytesparse.base.MutableMemory.extract()` `bytesparse.base.MutableMemory.to_bytes()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.read(start=124, endex=132)
b'bc\x00\x00\x00\x00xy'
>>> file.read(start=124, endex=132, fill=b'.')
b'bc....xy'
```

(continues on next page)

(continued from previous page)

```
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

property records: MutableSequence[*BaseRecord*]

Records stored by records role.

This readonly property exposes the list of records stored by the file object while in *records role*.

If this property is accessed while the file object is not in *records role*, it automatically activates it by an implicit call to *update_records()*, with default arguments.

For more control activating the *records role*, please call *update_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *memory role* (i.e. altering content of *memory*) would implicitly discard *records* via *discard_records()*.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, startaddr=789)
>>> len(file.records)
5
>>> _ = file.print()
S0030000FC
S106007B61626358
S10601C878797AC5
S5030002FA
S9030315E4
>>> _ = file.update_records(data_tag=SrecFile.Record.Tag.DATA_32)
>>> _ = file.print()
S0030000FC
S3080000007B61626356
S308000001C878797AC3
S5030002FA
S70500000315E2
```

Type
list of `BaseRecord`

save(*path*, **args*, ***kwargs*)

Saves a file object into the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *serialize()* to save a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdout.buffer` is used.
- **args** – Forwarded to *serialize()*.
- **kwargs** – Forwarded to *serialize()*.

Returns

`BaseFile` – *self*.

See also:

load() *serialize()* `open()` `sys.stdout.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> _ = file.save('data.hex')
```

serialize(*stream*, *end=b'\\r\\n'*, *nuls=True*, *xoff=True*)

Serializes records onto a byte stream.

It executes *MosRecord.serialize()* for each of the stored *records*.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **end** (*bytes*) – Line ending suffix bytes.
- **nuls** (*bool*) – Append six ASCII `NUL` (zero) bytes after each line, as prescribed by the original specifications.
- **xoff** (*bool*) – Append the ASCII `XOFF` byte at the end of the whole serialization.

Returns

MosFile – *self*.

See also:

parse() *MosRecord.serialize()*

Examples

```
>>> from hexrec import MosFile
>>> file = MosFile.from_blocks([[0xDA7A, b'abc']])
>>> import sys
>>> _ = file.serialize(sys.stdout.buffer, nuls=False, xoff=False)
;03DA7A616263027D
;0000010001
```

set_meta(*meta*, *strict=True*)

Sets meta information.

It sets the provided *kwargs* to their matching *meta* attributes, as listed by *META_KEYS*.

Parameters

- **meta** (*dict*) – Mapping of the *meta* information to set.
- **strict** (*bool*) – All the keys within *meta* must exist within *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *get_meta()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 0}
>>> _ = file.set_meta(dict(header=b'HDR\0', startaddr=456))
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 456}
```

shift(*offset*)

Shifts data addresses by an offset.

It shifts addresses of the underlying *memory* object data blocks by the provided *offset* amount.

Any stored *records* are discarded upon return.

Parameters

offset (*int*) – Offset to apply to the underlying data block addresses.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.shift()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> _ = file.shift(1000)
>>> file.memory.to_blocks()
[[1123, b'abc'], [1456, b'xyz']]
```

split(**addresses*, *meta=True*)

Splits into parts.

The provided *addresses* are sorted and used as markers to split *self* into parts.

Each part is the *copy()* of *self* within the range of that part, in *memory role* (i.e., *records* is not populated).

Parameters

- **addresses** (*int*) – Split points.
- **meta** (*bool*) – Each part inherits *meta* from *self*.

Returns

list of BaseFile – Parts after splitting.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'Hello, World!', offset=123)
>>> parts = file.split(128, 130)
>>> for part in parts: print(part.memory.to_blocks())
[[123, b'Hello']]
[[128, b', ']]
[[130, b'World!']]
>>> file.memory.to_blocks()
[[123, b'Hello, World!']]
```

update_records(*align=False*)

Applies memory and meta to records.

This method processes the stored *memory* and *meta* information to generate the sequence of *records*.

This effectively converts the *memory role* into the *records role* (keeping both).

The *records* is assigned upon return. Any exceptions being raised should not alter the file object.

Parameters

align (*bool*) – Aligns data record chunk address bounds to *maxdatalen*.

Returns

MosFile – *self*.

Raises

ValueError – *memory* attribute not populated.

See also:

records *memory* *get_meta()* *apply_records()*

Examples

```
>>> from hexrec import MosFile
>>> blocks = [[123, b'abc']]
>>> file = MosFile.from_blocks(blocks, maxdatalen=16)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'maxdatalen': 16}
>>> _ = file.update_records()
>>> len(file.records)
2
>>> _ = file.print(nuls=False)
;03007B61626301A4
;0000010001
```

validate_records(*data_ordering=False*, *eof_record_required=True*)

Validates records.

It performs consistency checks for the underlying *records*.

Parameters

- **data_ordering** (*bool*) – Checks that the *data* record sequence has monotonically increasing addresses, without any overlapping.
- **eof_record_required** (*bool*) – Requires the *End Of File* record be present.

Returns

MosFile – *self*.

Raises

ValueError – Invalid record sequence.

Examples

```
>>> from hexrec import MosFile
>>> records = [MosFile.Record.create_data(123, b'abc')]
>>> file = MosFile.from_records(records)
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: missing end of file record
```

view(*start=None*, *endex=None*)

Memory view.

It returns a `memoryview` over the specified range, which must cover a *contiguous* data region (i.e. no memory holes within).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
memoryview – View of the specified range.

Raises
ValueError – non-contiguous data within range.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> bytes(file.view(start=456, endex=458))
b'xy'
>>> bytes(file.view())
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

write(*address*, *data*, *clear=False*)

Writes data into the file.

It writes the provided *data* into the underlying *memory* object.

Any stored *records* are discarded upon return.

Parameters

- **address** (*int*) – Address where *data* has to be written.
- **data** (*bytes or memory*) – Byte data to write.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns
`BaseFile` – *self*.

See also:

memory *clear()* *discard_records()* `bytesparse.base.MutableMemory.write()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> _ = file.write(123, b'abc')
>>> _ = file.write(555, ord('?'))
>>> _ = file.write(1000, SrecFile.from_bytes(b'xyz', offset=456))
>>> file.memory.to_blocks()
[[123, b'abc'], [555, b'?'], [1456, b'xyz']]
```

4.6.2 MosRecord

class hexrec.formats.mos.**MosRecord**(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

MOS Technology record object.

Attributes

EQUALITY_KEYS	Meta keys for equality checks.
LINE_REGEX	Line parser regex.
META_KEYS	Meta keys.

Methods

__init__	
compute_checksum	Computes the checksum field value.
compute_count	Compute the count field value.
copy	Shallow copy.
create_data	Creates a data record.
create_eof	Creates an End Of File record.
data_to_int	Interprets data bytes as integer.
get_meta	Gets meta information.
parse	Parses a record from bytes.
print	Prints a record.
serialize	Serializes onto a stream.
to_bytestr	Converts into a byte string.
to_tokens	Converts into byte string tokens.
update_checksum	Updates the checksum field.
update_count	Updates the count field.
validate	Validates consistency of attribute values.

EQUALITY_KEYS: Sequence[str] = ['address', 'checksum', 'count', 'data', 'tag']

Meta keys for equality checks.

Equality methods (*__eq__()* and *__ne__()*) check against these *meta* keys only. Any other *meta* keys are just ignored.

LINE_REGEX = re.compile(b'^\x00*(?P<before>[^;]*);(?P<count>[0-9A-Fa-f]{2})(?P<address>[0-9A-Fa-f]{4})(?P<data>([0-9A-Fa-f]{2}){,255})(?P<checksum>[0-9A-Fa-f]{4})(?P<after>[^\\r\\n]*)\\r?\\n?\x00*$')

Line parser regex.

META_KEYS: Sequence[str] = ['address', 'after', 'before', 'checksum', 'coords', 'count', 'data', 'tag']

Meta keys.

This sequence holds the *meta* keys for copying (see *copy()*).

Tag

alias of *MosTag*

__bytes__()

Serializes the record into bytes.

Returns
bytes – Byte serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> bytes(record)
b':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> bytes(record)
b'abc'
```

__eq__(*other*)

Equality test.

This method returns true if *self* is considered equal to *other*.

As inequality is usually easier to check, this method is usually implemented as a trivial `not self != other` (*__ne__()*).

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* equals *other*.

See also:

__ne__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
```

(continues on next page)

(continued from previous page)

```
>>> ihex1 == ihex2
True
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 == ihex3
False
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 == raw
False
```

__hash__ = None

__init__(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

__ne__(*other*)

Ineuality test.

This method returns true if *self* is considered unequal to *other*.

Each attribute listed by *EQUALITY_KEYS* is compared between *self* and *other*. This method returns whether any attributes do not match.

Parameters
other (BaseRecord) – Record to compare to.

Returns
bool – *self* and *other* are unequal.

See also:

__eq__()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 != ihex2
False
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 != ihex3
True
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 != raw
True
```

__repr__()

String representation.

It returns a string representation of the record content, for human understanding only.

Returns
str – String representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> repr(record)
"<<class 'hexrec.formats.ihex.IhexRecord'> @...
  address:=0 after:=b'' before:=b'' checksum:=255 coords:=(-1, -1)
  count:=0 data:=b'' tag:=<IhexTag.END_OF_FILE: 1>>"
```

__str__()

Serializes the record into a string.

Returns
str – String serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> str(record)
'abc'
```

__weakref__

list of weak references to the object (if defined)

compute_checksum()

Computes the checksum field value.

It computes and returns the format-specific `checksum` value of a record.

When not specialized, it returns `None` by default.

Returns
int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_checksum()
215
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_checksum())
'None'
```

compute_count()

Compute the count field value.

It computes and returns the format-specific `count` value of a record.

When not specialized, it returns `None` by default.

Returns
int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_count()
3
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_count())
'None'
```

copy(*validate=True*)

Shallow copy.

It calls the record constructor, passing *meta* to it.

Parameters
validate (*bool*) – Performs validation on instantiation (*__init__()*).

Returns
`BaseRecord` – Shallow copy.

See also:

__init__() *get_meta()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record1 = IhexFile.Record.create_data(0x1234, b'abc')
>>> record2 = record1.copy()
>>> record1 is record2
False
>>> record1 == record2
True
```

classmethod create_data(*address*, *data*)

Creates a data record.

This is a mandatory class method to instantiate a *data* record.

Parameters

- **address** (*int*) – Record address. If not supported, set zero.
- **data** (*bytes*) – Record byte data.

Returns

`BaseRecord` – Data record object.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> str(record)
':0312340061626391\r\n'
```

classmethod create_eof(*record_count*)

Creates an End Of File record.

The End Of File record also carries the *record count*.

Parameters

record_count (*int*) – Number of preceding records.

Returns

MosRecord – End Of File record object.

Examples

```
>>> from hexrec import MosFile
>>> record = MosFile.Record.create_eof(123)
>>> str(record)
';00007B007B\r\n\x00\x00\x00\x00\x00\x00'
```

data_to_int(*byteorder='big'*, *signed=False*)

Interprets data bytes as integer.

It creates an integer from bytes of the `data` field.

Parameters

- **byteorder** (*'big' or 'little'*) – Byte order (endianness): either `'big'` (default) or `'little'`.
- **signed** (*bool*) – Signed integer (2-complement); default false.

Returns

int – Interpreted integer value.

See also:

`int.from_bytes()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0xABCD)
>>> record.data
b'\xab\xcd'
>>> addrext = record.data_to_int()
>>> addrext, hex(addrext)
(43981, '0xabcd')
```

get_meta()

Gets meta information.

It returns all the object attributes whose keys are listed by *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

See also:

META_KEYS `set_meta()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> record.get_meta()
{'address': 0, 'after': b'', 'before': b'', 'checksum': 255,
 'coords': (-1, -1), 'count': 0, 'data': b'',
 'tag': <IhexTag.END_OF_FILE: 1>}
```

classmethod parse(*line*, *eof=False*, *validate=True*)

Parses a record from bytes.

Please refer to the actual implementation provided by the record *format* for more details.

Parameters

- **line** (*bytes*) – String of bytes to parse.
- **eof** (*bool*) – Parsing an *End Of File* record.
- **validate** (*bool*) – Perform validation checks.

Returns

`BaseRecord` – Parsed record.

Raises

ValueError – Syntax error.

Examples

```
>>> from hexrec import MosFile
>>> record = MosFile.Record.parse(b';0000010001\r\n', eof=True)
>>> record.tag
<MosTag.EOF: 1>
>>> MosFile.Record.parse(b';;0000010001\r\n', eof=True)
Traceback (most recent call last):
    ...
ValueError: syntax error
```

print(**args*, *stream=None*, *color=False*, ***kwargs*)

Prints a record.

The record is converted into tokens (eventually colorized) then joined and written onto a byte stream (*stdout* by default).

Parameters

- **args** – Forwarded to the underlying call to *to_tokens()*.
- **stream** (`io.BytesIO`) – The byte stream where the record tokens are printed. If `None`, *stdout* is selected.
- **color** (*bool*) – Tokens are colorized before printing.
- **kwargs** – Forwarded to the underlying call to *to_tokens()*.

Returns
BaseRecord – *self*.

See also:

to_tokens() colorize_tokens() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> _ = record.print()
:0312340061626391
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.print(stream=stream, color=True)
>>> stream.getvalue()
b'\x1b[0m\x1b[33m:\x1b[34m03\x1b[31m1234\x1b[32m00\x1b[36m61\x1b[96m62\
↪x1b[36m63\x1b[35m91\x1b[0m\r\n\x1b[0m'
```

serialize(*stream*, **args*, ***kwargs*)

Serializes onto a stream.

This wraps a call to *to_bytestr()* and stream.write.

Parameters

- **stream** (io.BytesIO) – Stream to write.
- **args** – Forwarded to *to_bytestr()*.
- **kwargs** – Forwarded to *to_bytestr()*.

Returns
BaseRecord – *self*.

See also:

to_bytestr() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.serialize(stream, end=b'\n')
>>> stream.getvalue()
b':0312340061626391\n'
```

to_bytestr(*end=b'\\r\\n'*, *nuls=True*)

Converts into a byte string.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Byte string representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_bytestr(end=b'\n')
b':0312340061626391\n'
```

to_tokens(*end=b'\\r\\n'*, *nuls=True*)

Converts into byte string tokens.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Mapping of token keys to token byte strings.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_tokens(end=b'\n')
{'before': b'', 'begin': b':', 'count': b'03', 'address': b'1234',
 'tag': b'00', 'data': b'616263', 'checksum': b'91', 'after': b'',
 'end': b'\n'}
```

update_checksum()

Updates the checksum field.

It updates the `checksum` attribute, assigning to it the value returned by *compute_checksum()*.

Returns

`BaseRecord` – *self*.

See also:

`checksum` *compute_checksum()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> record = IhexRecord(IhexRecord.Tag.END_OF_FILE, checksum=None)
>>> record.compute_checksum()
255
>>> record.checksum is None
True
>>> _ = record.update_checksum()
>>> record.checksum
255
```

`update_count()`

Updates the count field.

It updates the `count` attribute, assigning to it the value returned by *compute_count()*.

Returns
`BaseRecord` – *self*.

See also:

`count` *compute_count()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> Record = IhexFile.Record
>>> Tag = Record.Tag
>>> record = Record(Tag.DATA, data=b'abc', count=None, checksum=None)
>>> record.compute_count()
3
>>> record.count is None
True
>>> _ = record.update_count()
>>> record.count
3
```

`validate`(*checksum=True*, *count=True*)

Validates consistency of attribute values.

All the record attributes are checked for consistency.

Please refer to the implementation for more details.

Parameters

- **checksum** (*bool*) – Check the consistency of the `checksum` attribute.
- **count** (*bool*) – Check the consistency of the `count` attribute.

Returns
BaseRecord – *self*.

Raises
ValueError – Some targeted attributes are inconsistent.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> _ = record.validate()
>>> record.data = b'abc'
>>> _ = record.update_count().update_checksum().validate()
Traceback (most recent call last):
    ...
ValueError: unexpcted data
```

4.6.3 MosTag

class hexrec.formats.mos.**MosTag**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

MOS Technology tag.

Attributes

DATA	Data.
EOF	End Of File.
denominator	the denominator of a rational number in lowest terms
imag	the imaginary part of a complex number
numerator	the numerator of a rational number in lowest terms
real	the real part of a complex number

Methods

is_eof	Tells whether this is an End Of File record tag.
__init__	
as_integer_ratio	Return integer ratio.
bit_count	Number of ones in the binary representation of the absolute value of self.
bit_length	Number of bits necessary to represent self in binary.
conjugate	Returns self, the complex conjugate of any int.
from_bytes	Return the integer represented by the given array of bytes.
to_bytes	Return an array of bytes representing an integer.

DATA = 0

Data.

EOF = 1

End Of File.

_DATA: Optional['BaseTag'] = 0

Alias to a common data record tag.

This tag is used internally to build a generic data record.

__abs__()

abs(self)

__add__(*value*, /)

Return self+value.

__and__(*value*, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

classmethod __contains__(*member*)

Return True if member is a member of this enum raises TypeError if member is not an enum member

note: in 3.12 TypeError will no longer be raised, and True will also be returned if member is the value of a member in this enum

__dir__()

Returns all members and all public methods

__divmod__(*value*, /)

Return divmod(self, value).

__eq__(*value*, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(*value*, /)

Return self//value.

__format__(*format_spec*, /)

Default object formatter.

__ge__(*value*, /)

Return self>=value.

__getattribute__(*name*, /)

Return getattr(self, name).

classmethod __getitem__(*name*)
Return the member matching *name*.

__gt__(*value*, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__init__(**args*, ***kwds*)

__int__()
int(self)

__invert__()
~self

classmethod __iter__()
Return members in definition order.

__le__(*value*, /)
Return self<=value.

classmethod __len__()
Return the number of members (no aliases)

__lshift__(*value*, /)
Return self<<value.

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

__mul__(*value*, /)
Return self*value.

__ne__(*value*, /)
Return self!=value.

__neg__()
-self

__new__(*value*)

__or__(*value*, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod=None*, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__reduce_ex__(*proto*)
Helper for pickle.

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod=None*, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(*value*, /)

Return self-value.

__truediv__(*value*, /)

Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(*value*, /)

Return self^value.

_generate_next_value_(*start*, *count*, *last_values*)

Generate the next value when not given.

name: the name of the member start: the initial start value or None count: the number of existing members last_values: the list of values assigned

_member_type_

alias of `int`

_new_member_(***kwargs*)

Create and return a new object. See help(type) for accurate signature.

_value_repr_()

Return repr(self).

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

`conjugate()`

Returns self, the complex conjugate of any int.

`denominator`

the denominator of a rational number in lowest terms

`from_bytes`(*byteorder='big'*, ***, *signed=False*)

Return the integer represented by the given array of bytes.

bytes
Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

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. Default is to use 'big'.

signed
Indicates whether two's complement is used to represent the integer.

`imag`

the imaginary part of a complex number

`is_eof()`

Tells whether this is an End Of File record tag.

This method returns true if this record tag is used for *End Of File* records.

Returns
bool – This is an End Of File record tag.

Examples

```
>>> from hexrec import MosFile
>>> MosTag = MosFile.Record.Tag
>>> MosTag.EOF.is_eof()
True
>>> MosTag.DATA.is_eof()
False
```

`numerator`

the numerator of a rational number in lowest terms

`real`

the real part of a complex number

`to_bytes`(*length=1*, *byteorder='big'*, ***, *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. Default is length 1.

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. Default is to use 'big'.

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.

4.7 raw

Binary format.

This format is actually used to hold binary chunks of raw data (*bytes*).

Classes

RawFile	Raw binary file object.
RawRecord	Raw binary record object.
RawTag	Raw binary tag.

4.7.1 RawFile

class hexrec.formats.raw.**RawFile**
Raw binary file object.

Attributes

DEFAULT_DATALEN	Default data attribute length.
FILE_EXT	Supported filename extensions.
META_KEYS	Meta information key names.
maxdatalen	Maximum byte size of the data field.
memory	Memory object stored by records role.
records	Records stored by records role.

Methods

__init__	
align	Pads blocks to align their boundaries.
append	Appends a byte.
apply_records	Applies records to memory and meta.
clear	Clears data within a range.
convert	Converts a file object to another format.

continues on next page

Table 6 – continued from previous page

copy	Copies within a range.
crop	Clears data outside a range.
cut	Cuts data within a range.
delete	Deletes data within a range.
discard_memory	Discards underlying memory.
discard_records	Discards underlying records.
extend	Concatenates data.
fill	Fills a range.
find	Finds a substring.
flood	Floods a range.
from_blocks	Creates a file object from a memory object.
from_bytes	Creates a file object from a byte string.
from_memory	Creates a file object from a memory object.
from_records	Creates a file object from records.
get_address_max	Maximum address within memory.
get_address_min	Minimum address within memory.
get_holes	List of memory holes.
get_meta	Meta information.
get_spans	List of memory block spans.
index	Finds a substring.
load	Loads a file object from the filesystem.
merge	Merges data onto the file.
parse	Parses records from a byte stream.
print	Prints record content to stdout.
read	Extracts a substring.
save	Saves a file object into the filesystem.
serialize	Serializes records onto a byte stream.
set_meta	Sets meta information.
shift	Shifts data addresses by an offset.
split	Splits into parts.
update_records	Applies memory and meta to records.
validate_records	Validates records.
view	Memory view.
write	Writes data into the file.

DEFAULT_DATALEN: int = 9223372036854775807

Default data attribute length.

Default value for the *maxdatalen* *meta*, which sets the maximum size of `BaseRecord.data` field values.

FILE_EXT: Sequence[str] = ['.bin', '.dat', '.eep', '.raw']

Supported filename extensions.

Sequence of file name extension substrings (e.g. `.hex`). This list is used by functions like `guess_format_name()` to manage mapping of file *formats*.

META_KEYS: Sequence[str] = ['maxdatalen']

Meta information key names.

Sequence of key strings listing the supported *meta* information of this file *format*.

Record

alias of *RawRecord*

__add__(*other*)

Concatenates with another file.

Equivalent to *copy()* then *extend()*.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – Concatenation of *self* and *other*.

See also:

copy() extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 + file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file4 = file3 + b'789'
>>> file4.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__bool__()

bool: Has data records or memory.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> bool(file)
False
>>> _ = file.append(0)
>>> bool(file)
True
```

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> file = IhexFile.from_records([IhexRecord.create_end_of_file()])
>>> bool(file)
False
>>> file.records.insert(0, IhexRecord.create_data(0, b'\0'))
>>> bool(file)
True
```

__delitem__(*key*)

Deletes a range.

Parameters
key (*slice or int*) – Range to delete.

See also:

`bytesparse.base.MutableMemory.__delitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> del file[457]
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xz']]
>>> del file[125:457]
>>> file.memory.to_blocks()
[[123, b'abz']]
```

__eq__(*other*)

Equality test.

The file objects *self* and *other* are considered *equal* if the inequality tests of *__ne__()* result false.

Returns
bool – *self* and *other* are *equal*.

See Also
__ne__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 == file2
True
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 == file3
False
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 == file4
False
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file == ihex_file
False
>>> srec_file.memory == ihex_file.memory
True
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__getitem__(*key*)

Extracts a range.

Parameters
key (*slice or int*) – Range to extract.

Raises
ValueError – invalid range.

See also:

`bytesparse.base.ImmutableMemory.__getitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> chr(file[457])
'y'
>>> repr(file[333])
'None'
>>> file[123:125]
b'ab'
>>> file[125:457]
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

__hash__ = None

__iadd__(*other*)

Concatenates data.

Equivalent to *extend()*.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – *self*.

See also:

memory extend() discard_records() bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 += file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file1 += b'789'
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__init__()

__ior__(*other*)

Merges with another file.

Equivalent to *merge()*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self*.

See also:

merge() discard_records()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 |= file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file1 |= b'789'
>>> file1.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__ne__(*other*)

Inequality test.

The file objects *self* and *other* are considered *unequal* if any of the following tests result true:

- Both have *memory role* (i.e. *memory*), resulting unequal;
- Both have *records role* (i.e. *records*), resulting unequal;
- *other* does not have a *meta* listed by *META_KEYS*;
- A *meta* value (among those of *META_KEYS*) is different.

Returns
bool – *self* and *other* are *unequal*.

See also:

__eq__() memory records META_KEYS

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 != file2
False
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 != file3
True
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 != file4
True
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file != ihex_file
True
>>> srec_file.memory != ihex_file.memory
False
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__or__(*other*)

Merges with another file.

Equivalent to *copy()* then *merge()*.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self* merged with *other*.

See also:

copy() merge()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 | file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file4 = file3 | b'789'
>>> file4.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__setitem__(*key*, *value*)

Sets a range.

Parameters

- **key** (*slice or int*) – Range to set.
- **value** (bytes, `bytesparse.base.ImmutableMemory`, `None`) – Value(s) to set. `None` acts like *clear()*.

Raises

ValueError – invalid range.

See also:

`bytesparse.base.MutableMemory.__setitem__()` *clear()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file[124] = b'?'
>>> file.memory.to_blocks()
[[123, b'a?c'], [456, b'xyz']]
>>> file[:125] = None
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'xyz']]
>>> file[457:458] = b'789'
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'x789z']]
```

__weakref__

list of weak references to the object (if defined)

classmethod _is_line_empty(*line*)

Empty line check.

Tells whether a *line* has no meaningful content (e.g. all whitespace). The check itself depends on the implementing file *format*. It may be used internally to skip empty lines, e.g. by *parse()*.

Parameters
line (*bytes*) – A line, byte string.

Returns
bool: The *line* is empty.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexFile._is_line_empty(b'')
True
>>> IhexFile._is_line_empty(b' \t\v\r\n')
True
>>> IhexFile._is_line_empty(b':00000001FF\r\n')
False
```

align(*modulo*, *start=None*, *endex=None*, *pattern=0*)

Pads blocks to align their boundaries.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*, so that memory blocks are aligned to the required *modulo*.

Any stored *records* are discarded upon return.

Parameters

- **modulo** (*int*) – Alignment modulo.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns
`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.align()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [134, b'xyz']])
>>> _ = file.align(4, pattern=b'.')
>>> file.memory.to_blocks()
[[120, b'...abc..'], [132, b'..xyz...']]
```

append(*item*)

Appends a byte.

It appends the *item* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
item (*byte or int*) – Byte to append.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.append()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123)
>>> _ = file.append(b'.')
>>> _ = file.append(0)
>>> file.memory.to_blocks()
[[123, b'abc.\x00']]
```

apply_records()

Applies records to memory and meta.

This method processes the stored *records*, converting *data* as *memory*, and special records into their *meta* counterparts.

This effectively converts the *records role* into the *memory role* (keeping both).

The *memory* and *meta* are assigned upon return. Any exceptions being raised should not alter the file object.

Returns
BaseFile – *self*.

Raises
ValueError – *records* attribute not populated.

See also:

records memory get_meta() update_records()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_start_linear_address(456),
```

(continues on next page)

(continued from previous page)

```
...             IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records, maxdatalen=16)
>>> _ = file.apply_records()
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
```

clear(*start=None*, *endex=None*)

Clears data within a range.

It clears the specified range of underlying *memory* object, making a memory hole.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.clear()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.clear(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

classmethod convert(*source*, *meta=True*)

Converts a file object to another format.

It copies the *memory* and *meta* of the *source* file object, creating a new one of the target `BaseFile` format type.

Parameters

- **source** (`BaseFile`) – Source file object to convert.
- **meta** (*bool*) – Copy *meta* information to the target file object. Only the keys of the target *META_KEYS* are processed.

Returns

`BaseFile` – Converted copy of *source* to the target format.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> source = IhexFile.from_blocks(blocks, startaddr=789)
>>> target = SrecFile.convert(source)
>>> target.memory is source.memory
False
>>> target.memory == source.memory
True
>>> target.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 789}
```

copy(*start=None*, *endex=None*, *meta=True*)

Copies within a range.

It copied data within the specified range of the file object, creating a new one carrying the inner slice.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

`BaseFile` – *self*.

See also:

memory *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.copy(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

crop(*start=None*, *endex=None*)

Clears data outside a range.

It clears outside the specified range of underlying *memory* object, timming it.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.crop()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.crop(start=124, endex=132)
>>> file.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
```

cut(*start=None*, *endex=None*, *meta=False*)

Cuts data within a range.

It takes data within the specified range away from the file object, creating a new one carrying the inner slice. The inner slice is cleared from *self*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns
`BaseFile` – *self*.

See also:

memory *clear()* *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.cut(start=124, endex=132)
>>> inner.memory.to_blocks()
```

(continues on next page)

(continued from previous page)

```
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

delete(*start=None*, *endex=None*)

Deletes data within a range.

It deletes the specified range of underlying *memory* object, shifting all subsequent data towards the collapsed range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.delete()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.delete(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'az']]
```

discard_memory()

Discards underlying memory.

The underlying *memory* object is assigned `None`.

If the underlying *records* object is `None`, it is assigned a new empty memory object.

Returns

`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_bytes(b'abc', offset=123)
>>> _ = file.update_records()
>>> _ = file.discard_memory()
>>> _ = file.update_records()
Traceback (most recent call last):
    ...
ValueError: memory instance required
```

discard_records()

Discards underlying records.

The underlying *records* object is assigned `None`.

If the underlying *memory* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> _ = file.validate_records()
>>> _ = file.discard_records()
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: records required
```

extend(*other*)

Concatenates data.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.extend()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> _ = file1.extend(file2)
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> _ = file1.extend(b'789')
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

fill(*start=None*, *endex=None*, *pattern=0*)

Fills a range.

It writes a *pattern* of bytes onto the underlying *memory* object, overwriting anything within the specified range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.fill()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.fill(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'a.......z']]
```

find(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it returns `-1`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address; `-1` if not found.

See also:

index `bytesparse.base.ImmutableMemory.find()`

Notes

The internal *memory* might allow negative addresses for its stored data. In that case, *index()* would be more appropriate, because it raises an exception when the *item* is not found.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.find(b'yz')
457
>>> file.find(ord('b'))
124
>>> file.find(b'?')
-1
```

flood(*start=None*, *endex=None*, *pattern=0*)

Floods a range.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.flood()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.get_holes()
[(126, 130)]
>>> _ = file.flood(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'abc....xyz']]
```

classmethod from_blocks(*blocks*, ***meta*)

Creates a file object from a memory object.

The *blocks* are put into the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **blocks** (*list of blocks*) – Memory blocks to put into *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

`BaseFile` – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* `bytesparse.base.ImmutableMemory.from_blocks()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_bytes(*data*, *offset=0*, ***meta*)

Creates a file object from a byte string.

The byte string makes a single *data* block, placed at some offset within the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **data** (*bytes*) – A byte string used to make a single data block.
- **offset** (*int*) – Offset of the single data block within *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_bytes()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
8
```

classmethod **from_memory**(*memory=None*, ***meta*)

Creates a file object from a memory object.

The *memory* is set as the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **memory** (bytesparse.base.MutableMemory) – Memory object to set as *memory*. If None, an empty memory object is automatically created.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS bytesparse.base.MutableMemory

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from bytesparse import Memory
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> memory = Memory.from_blocks(blocks)
>>> from hexrec import SrecFile
>>> file = SrecFile.from_memory(memory, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_records(*records*, *maxdatalen=None*)

Creates a file object from records.

The *records* sequence is set as the `record` attribute of the created file object.

This method creates a file object in *records role*. This means that only its *records* is internally instanced, while the *memory* requires manual or lazy instancing (i.e. either via direct call to *apply_records()*, or any other methods indirectly calling it).

Parameters

- **records** (list of `BaseRecord`) – Record sequence to set as *records*.
- **maxdatalen** (`Optional[int]`) – Maximum record *data* field size. If `None`, the maximum non-zero size of the *data* field from the *records* sequence is used. If all the *records* have zero sized *data* field, the class attribute *DEFAULT_DATALEN* is used.

Returns

`BaseFile` – The created file object.

Raises

ValueError – invalid *meta* values.

See also:

`BaseRecord`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
3
```

get_address_max()

Maximum address within memory.

It returns the maximum address of the underlying *memory* object.

Returns
int – Maximum address.

See also:

`bytesparse.base.ImmutableMemory.endin`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_max()
458
```

get_address_min()

Minimum address within memory.

It returns the minimum address of the underlying *memory* object.

Returns
int – Minimum address.

See also:

`bytesparse.base.ImmutableMemory.start`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_min()
123
```

get_holes()

List of memory holes.

It scans the underlying *memory* and returns the list of memory holes/gaps.

Each hole is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory hole boundaries.

See also:

`bytesparse.base.ImmutableMemory.gaps()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_holes()
[(126, 456), (459, 789)]
```

get_meta()

Meta information.

It builds and returns a dictionary of *meta* information. Meta keys are taken from the *META_KEYS* class attribute.

Returns
dict – Meta information dictionary.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks, header=b'HDR\0')
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 0}
```

get_spans()

List of memory block spans.

It scans the underlying *memory* and returns the list of memory block spans/intervals.

Each span is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory block boundaries.

See also:

`bytesparse.base.ImmutableMemory.intervals()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_spans()
[(123, 126), (456, 459), (789, 791)]
```

index(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it raises `ValueError`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address.

Raises

ValueError – *item* not found.

See also:

find `bytesparse.base.ImmutableMemory.index()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.index(b'yz')
457
>>> file.index(ord('b'))
124
>>> file.index(b'?')
Traceback (most recent call last):
    ...
ValueError: subsection not found
```

classmethod load(*path*, **args*, ***kwargs*)

Loads a file object from the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *parse()* to load a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdin.buffer` is used.
- **args** – Forwarded to *parse()*.
- **kwargs** – Forwarded to *parse()*.

Returns

`BaseFile` – Loaded file object.

See also:

save() *parse()* `open()` `sys.stdin.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.load('data.hex')
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

property maxdatalen: int

Maximum byte size of the data field.

This property sets the maximum byte size of the *data* field of a serialized record.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises

ValueError – Invalid maximum data length.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> file.maxdatalen
16
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> file.maxdatalen = 8
>>> _ = file.print()
S0030000FC
S10B0000000102030405060 7D8
S10B000808090A0B0C0D0E0F90
S10B001010111213141516 1748
S10B001818191A1B1C1D1E1F00
S10B002020212223242526 27B8
S10B002828292A2B2C2D2E2F70
S10B003030313233343536 3728
```

(continues on next page)

(continued from previous page)

```
S10B003838393A3B3C3D3E3FE0
S5030008F4
S9030000FC
>>> file.maxdatalen = 0
Traceback (most recent call last):
    ...
ValueError: invalid maximum data length
```

Type
int

property memory: MutableMemory

Memory object stored by records role.

This readonly property exposes the memory object stored by the file object while in *memory role*.

If this property is accessed while the file object is not in *memory role*, it automatically activates it by an implicit call to *apply_records()*, with default arguments.

For more control activating the *memory role*, please call *apply_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *records role* (i.e. altering content of *records*) would implicitly discard *memory* via *discard_memory()*.

See also:

apply_records() *discard_memory()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> _ = file.write(789, b'?!')
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz'], [789, b'?!']]
```

Type
`bytesparse.Memory`

merge(**files*, *clear=False*)

Merges data onto the file.

It writes the provided *files* onto *self*, in the provided order. Any common address ranges are overwritten.

Any stored *records* are discarded upon return.

Parameters

- **files** (BaseFile) – Files to merge.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

BaseFile – *self*.

See also:

clear() *discard_records()* *write()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = SrecFile.from_bytes(b'<<<?????>>>', offset=450)
>>> _ = file3.merge(file1, file2)
>>> file3.memory.to_blocks()
[[123, b'abc'], [450, b'<<<???xyz>>']]
```

classmethod parse(*stream*, *ignore_errors=False*, *maxdatalen=9223372036854775807*, *address=0*)

Parses records from a byte stream.

It executes *RawRecord.parse()* for each line of the incoming *stream*, creating a new file object with the collected records calling *from_records()*.

Notes

Please refer to the actual implementation of each record file *format*, because it may be more specialized.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **ignore_errors** (*bool*) – Ignore Exception raised by *RawRecord.parse()*.
- **maxdatalen** (*int*) – Maximum *data* record data size, to chop the incoming stream.
- **address** (*int*) – Initial address.

Returns

RawFile – *self*.

See also:

parse() BaseRecord.parse() *from_records()*

Examples

```
>>> from hexrec import RawFile
>>> import io
>>> stream = io.BytesIO(b'Hello, World!')
>>> file = RawFile.parse(stream, maxdatalen=5, address=1000)
>>> for record in file.records:
...     print(f'{record.address}: {record.data!r}')
1000: b'Hello'
1005: b', Wor'
1010: b'ld!'
>>> file.get_meta()
{'maxdatalen': 5}
```

print(**args*, *stream=None*, *color=False*, *start=None*, *stop=None*, ***kwargs*)

Prints record content to stdout.

This helper method prints each record of *records* via `BaseRecord.print()`. As such, it also supports colored tokens and streams different from *stdout*.

It is possible to print subset of the records by specifying the record index range.

Warning: This method is **NOT** equivalent to *serialize()*, because it just prints each record from *records*. Please use *serialize()* for an actual serialization of the whole file.

Parameters

- **args** – Forwarded to the underlying call to `to_tokens()`.
- **stream** (*byte stream*) – Stream to print onto. If `None`, *stdout* is used.
- **color** (*bool*) – Colorize record tokens with ANSI color codes.
- **start** (*int*) – Inclusive start record index of the specified range. If `None`, start from the first record.
- **stop** (*int*) – Exclusive end record index of the specified range. If negative, look back from the last index. If `None`, print up to the last record.
- **kwargs** – Forwarded to the underlying call to `to_tokens()`.

Returns

`BaseFile` – *self*.

See also:

`BaseRecord.print()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> _ = file.print(color=True, start=1, stop=-2)
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
```

read(*start=None*, *endex=None*, *fill=0*)

Extracts a substring.

It extracts a byte string from the specified range, filling any memory holes/gaps (without altering *memory*).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **fill** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory `bytesparse.base.MutableMemory.extract()` `bytesparse.base.MutableMemory.to_bytes()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.read(start=124, endex=132)
b'bc\x00\x00\x00\x00xy'
>>> file.read(start=124, endex=132, fill=b'.')
b'bc....xy'
```

(continues on next page)

(continued from previous page)

```
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

property records: MutableSequence[*BaseRecord*]

Records stored by records role.

This readonly property exposes the list of records stored by the file object while in *records role*.

If this property is accessed while the file object is not in *records role*, it automatically activates it by an implicit call to *update_records()*, with default arguments.

For more control activating the *records role*, please call *update_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *memory role* (i.e. altering content of *memory*) would implicitly discard *records* via *discard_records()*.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, startaddr=789)
>>> len(file.records)
5
>>> _ = file.print()
S0030000FC
S106007B61626358
S10601C878797AC5
S5030002FA
S9030315E4
>>> _ = file.update_records(data_tag=SrecFile.Record.Tag.DATA_32)
>>> _ = file.print()
S0030000FC
S3080000007B61626356
S308000001C878797AC3
S5030002FA
S70500000315E2
```

Type
list of `BaseRecord`

save(*path*, **args*, ***kwargs*)

Saves a file object into the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *serialize()* to save a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdout.buffer` is used.
- **args** – Forwarded to *serialize()*.
- **kwargs** – Forwarded to *serialize()*.

Returns

`BaseFile` – *self*.

See also:

load() *serialize()* `open()` `sys.stdout.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> _ = file.save('data.hex')
```

serialize(*stream*, **args*, ***kwargs*)

Serializes records onto a byte stream.

It executes `BaseRecord.serialize()` for each of the stored *records*.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **args** – Forwarded to `BaseRecord.serialize()` of each record.
- **kwargs** – Forwarded to `BaseRecord.serialize()` of each record.

Returns

`BaseFile` – *self*.

See also:

parse() `BaseRecord.serialize()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> import sys
>>> _ = file.serialize(sys.stdout.buffer, end=b'\n')
:03DA7A0061626383
:040000050000CAFE2F
:00000001FF
```

set_meta(*meta*, *strict=True*)

Sets meta information.

It sets the provided *kwargs* to their matching *meta* attributes, as listed by *META_KEYS*.

Parameters

- **meta** (*dict*) – Mapping of the *meta* information to set.
- **strict** (*bool*) – All the keys within *meta* must exist within *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *get_meta()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 0}
>>> _ = file.set_meta(dict(header=b'HDR\0', startaddr=456))
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 456}
```

shift(*offset*)

Shifts data addresses by an offset.

It shifts addresses of the underlying *memory* object data blocks by the provided *offset* amount.

Any stored *records* are discarded upon return.

Parameters

offset (*int*) – Offset to apply to the underlying data block addresses.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.shift()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> _ = file.shift(1000)
>>> file.memory.to_blocks()
[[1123, b'abc'], [1456, b'xyz']]
```

split(**addresses*, *meta=True*)

Splits into parts.

The provided *addresses* are sorted and used as markers to split *self* into parts.

Each part is the *copy()* of *self* within the range of that part, in *memory role* (i.e., *records* is not populated).

Parameters

- **addresses** (*int*) – Split points.
- **meta** (*bool*) – Each part inherits *meta* from *self*.

Returns

list of `BaseFile` – Parts after splitting.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'Hello, World!', offset=123)
>>> parts = file.split(128, 130)
>>> for part in parts: print(part.memory.to_blocks())
[[123, b'Hello']]
[[128, b', ']]
[[130, b'World!']]
>>> file.memory.to_blocks()
[[123, b'Hello, World!']]
```

update_records(*align=False*)

Applies memory and meta to records.

This method processes the stored *memory* and *meta* information to generate the sequence of *records*.

This effectively converts the *memory role* into the *records role* (keeping both).

The *records* is assigned upon return. Any exceptions being raised should not alter the file object.

Parameters

align (*bool*) – Aligns data record chunk address bounds to *maxdatalen*.

Returns

RawFile – *self*.

Raises

ValueError – *memory* attribute not populated.

See also:

records *memory* *get_meta()* *apply_records()*

Examples

```
>>> from hexrec import RawFile
>>> blocks = [[123, b'abc']]
>>> file = RawFile.from_blocks(blocks, maxdatalen=16)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'maxdatalen': 16}
>>> _ = file.update_records()
>>> len(file.records)
1
>>> _ = file.print()
abc
```

validate_records(*data_start=True*, *data_contiguity=True*, *data_ordering=True*)

Validates records.

It performs consistency checks for the underlying *records*.

Parameters

- **data_start** (*bool*) – Data records must start from address zero.
- **data_contiguity** (*bool*) – Requires *data* records be ordered and contiguous.
- **data_ordering** (*bool*) – Checks that the *data* record sequence has monotonically increasing addresses, without any overlapping.

Returns

RawFile – *self*.

Raises

ValueError – Invalid record sequence.

Examples

```
>>> from hexrec import RawFile
>>> records = [RawFile.Record.create_data(123, b'abc')]
>>> file = RawFile.from_records(records)
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: first record address not zero
```

view(*start=None*, *endex=None*)

Memory view.

It returns a `memoryview` over the specified range, which must cover a *contiguous* data region (i.e. no memory holes within).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
memoryview – View of the specified range.

Raises
ValueError – non-contiguous data within range.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> bytes(file.view(start=456, endex=458))
b'xy'
>>> bytes(file.view())
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

write(*address*, *data*, *clear=False*)

Writes data into the file.

It writes the provided *data* into the underlying *memory* object.

Any stored *records* are discarded upon return.

Parameters

- **address** (*int*) – Address where *data* has to be written.
- **data** (*bytes or memory*) – Byte data to write.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns
`BaseFile` – *self*.

See also:

memory *clear()* *discard_records()* `bytesparse.base.MutableMemory.write()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> _ = file.write(123, b'abc')
>>> _ = file.write(555, ord('?'))
>>> _ = file.write(1000, SrecFile.from_bytes(b'xyz', offset=456))
```

(continues on next page)

(continued from previous page)

```
>>> file.memory.to_blocks()
[[123, b'abc'], [555, b'?'], [1456, b'xyz']]
```

4.7.2 RawRecord

class hexrec.formats.raw.**RawRecord**(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

Raw binary record object.

Attributes

EQUALITY_KEYS	Meta keys for equality checks.
META_KEYS	Meta keys.

Methods

__init__	
compute_checksum	Computes the checksum field value.
compute_count	Compute the count field value.
copy	Shallow copy.
create_data	Creates a data record.
data_to_int	Interprets data bytes as integer.
get_meta	Gets meta information.
parse	Parses a record from bytes.
print	Prints a record.
serialize	Serializes onto a stream.
to_bytestr	Converts into a byte string.
to_tokens	Converts into byte string tokens.
update_checksum	Updates the checksum field.
update_count	Updates the count field.
validate	Validates consistency of attribute values.

EQUALITY_KEYS: Sequence[str] = ['address', 'checksum', 'count', 'data', 'tag']

Meta keys for equality checks.

Equality methods (*__eq__()* and *__ne__()*) check against these *meta* keys only. Any other *meta* keys are just ignored.

META_KEYS: Sequence[str] = ['address', 'after', 'before', 'checksum', 'coords', 'count', 'data', 'tag']

Meta keys.

This sequence holds the *meta* keys for copying (see *copy()*).

Tag

alias of *RawTag*

__bytes__()

Serializes the record into bytes.

Returns
bytes – Byte serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> bytes(record)
b':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> bytes(record)
b'abc'
```

__eq__(*other*)

Equality test.

This method returns true if *self* is considered equal to *other*.

As inequality is usually easier to check, this method is usually implemented as a trivial `not self != other` (*__ne__()*).

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* equals *other*.

See also:

__ne__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 == ihex2
True
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
```

(continues on next page)

(continued from previous page)

```
>>> ihex1 == ihex3
False
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 == raw
False
```

__hash__ = None

__init__(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

__ne__(*other*)

Ineuality test.

This method returns true if *self* is considered unequal to *other*.

Each attribute listed by *EQUALITY_KEYS* is compared between *self* and *other*. This method returns whether any attributes do not match.

Parameters
other (BaseRecord) – Record to compare to.

Returns
bool – *self* and *other* are unequal.

See also:

__eq__()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 != ihex2
False
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 != ihex3
True
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 != raw
True
```

__repr__()

String representation.

It returns a string representation of the record content, for human understanding only.

Returns
str – String representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> repr(record)
"<<class 'hexrec.formats.ihex.IhexRecord'> @...
  address:=0 after:=b'' before:=b'' checksum:=255 coords:=(-1, -1)
  count:=0 data:=b'' tag:=<IhexTag.END_OF_FILE: 1>>"
```

__str__()

Serializes the record into a string.

Returns

str – String serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> str(record)
'abc'
```

__weakref__

list of weak references to the object (if defined)

compute_checksum()

Computes the checksum field value.

It computes and returns the format-specific `checksum` value of a record.

When not specialized, it returns `None` by default.

Returns

int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_checksum()
215
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_checksum())
'None'
```

compute_count()

Compute the count field value.

It computes and returns the format-specific `count` value of a record.

When not specialized, it returns `None` by default.

Returns
int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_count()
3
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_count())
'None'
```

copy(*validate=True*)

Shallow copy.

It calls the record constructor, passing *meta* to it.

Parameters
validate (*bool*) – Performs validation on instantiation (*__init__()*).

Returns
`BaseRecord` – Shallow copy.

See also:

__init__() *get_meta()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record1 = IhexFile.Record.create_data(0x1234, b'abc')
>>> record2 = record1.copy()
>>> record1 is record2
False
>>> record1 == record2
True
```

classmethod create_data(*address*, *data*)

Creates a data record.

This is a mandatory class method to instantiate a *data* record.

Parameters

- **address** (*int*) – Record address. If not supported, set zero.
- **data** (*bytes*) – Record byte data.

Returns

`BaseRecord` – Data record object.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> str(record)
':0312340061626391\r\n'
```

data_to_int(*byteorder='big'*, *signed=False*)

Interprets data bytes as integer.

It creates an integer from bytes of the `data` field.

Parameters

- **byteorder** (*'big' or 'little'*) – Byte order (endianness): either `'big'` (default) or `'little'`.
- **signed** (*bool*) – Signed integer (2-complement); default false.

Returns

int – Interpreted integer value.

See also:

`int.from_bytes()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0xABCD)
>>> record.data
b'\xab\xcd'
>>> addrext = record.data_to_int()
>>> addrext, hex(addrext)
(43981, '0xabcd')
```

get_meta()

Gets meta information.

It returns all the object attributes whose keys are listed by *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

See also:

META_KEYS `set_meta()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> record.get_meta()
{'address': 0, 'after': b'', 'before': b'', 'checksum': 255,
 'coords': (-1, -1), 'count': 0, 'data': b'',
 'tag': <IhexTag.END_OF_FILE: 1>}
```

classmethod parse(*line*, *address=0*, *validate=True*)

Parses a record from bytes.

Please refer to the actual implementation provided by the record *format* for more details.

Parameters

- **line** (*bytes*) – String of bytes to parse.
- **address** (*int*) – Record address.
- **validate** (*bool*) – Perform validation checks.

Returns

RawRecord – Parsed record.

Raises

ValueError – Syntax error.

Examples

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.parse(b'abc', address=123)
>>> record.address
123
>>> record.data
b'abc'
```

print(**args*, *stream=None*, *color=False*, ***kwargs*)

Prints a record.

The record is converted into tokens (eventually colorized) then joined and written onto a byte stream (*stdout* by default).

Parameters

- **args** – Forwarded to the underlying call to *to_tokens()*.
- **stream** (io.BytesIO) – The byte stream where the record tokens are printed. If None, *stdout* is selected.
- **color** (*bool*) – Tokens are colorized before printing.
- **kwargs** – Forwarded to the underlying call to *to_tokens()*.

Returns

BaseRecord – *self*.

See also:

to_tokens() colorize_tokens() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> _ = record.print()
:0312340061626391
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.print(stream=stream, color=True)
>>> stream.getvalue()
b'\x1b[0m\x1b[33m:\x1b[34m03\x1b[31m1234\x1b[32m00\x1b[36m61\x1b[96m62\
↪x1b[36m63\x1b[35m91\x1b[0m\r\n\x1b[0m'
```

serialize(*stream*, **args*, ***kwargs*)

Serializes onto a stream.

This wraps a call to *to_bytestr()* and stream.write.

Parameters

- **stream** (io.BytesIO) – Stream to write.
- **args** – Forwarded to *to_bytestr()*.

- **kwargs** – Forwarded to *to_bytestr()*.

Returns
BaseRecord – *self*.

See also:

to_bytestr() io.BytesIO

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.serialize(stream, end=b'\n')
>>> stream.getvalue()
b':0312340061626391\n'
```

`to_bytestr()`

Converts into a byte string.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns
bytes – Byte string representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_bytestr(end=b'\n')
b':0312340061626391\n'
```

`to_tokens()`

Converts into byte string tokens.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns
bytes – Mapping of token keys to token byte strings.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_tokens(end=b'\n')
{'before': b'', 'begin': b':', 'count': b'03', 'address': b'1234',
 'tag': b'00', 'data': b'616263', 'checksum': b'91', 'after': b'',
 'end': b'\n'}
```

update_checksum()

Updates the checksum field.

It updates the `checksum` attribute, assigning to it the value returned by *compute_checksum()*.

Returns
`BaseRecord` – *self*.

See also:

`checksum` *compute_checksum()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> record = IhexRecord(IhexRecord.Tag.END_OF_FILE, checksum=None)
>>> record.compute_checksum()
255
>>> record.checksum is None
True
>>> _ = record.update_checksum()
>>> record.checksum
255
```

update_count()

Updates the count field.

It updates the `count` attribute, assigning to it the value returned by *compute_count()*.

Returns
`BaseRecord` – *self*.

See also:

`count` *compute_count()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> Record = IhexFile.Record
>>> Tag = Record.Tag
>>> record = Record(Tag.DATA, data=b'abc', count=None, checksum=None)
>>> record.compute_count()
3
>>> record.count is None
True
>>> _ = record.update_count()
>>> record.count
3
```

validate(*checksum=True*, *count=True*)

Validates consistency of attribute values.

All the record attributes are checked for consistency.

Please refer to the implementation for more details.

Parameters

- **checksum** (*bool*) – Check the consistency of the `checksum` attribute.
- **count** (*bool*) – Check the consistency of the `count` attribute.

Returns

`BaseRecord` – *self*.

Raises

ValueError – Some targeted attributes are inconsistent.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> _ = record.validate()
>>> record.data = b'abc'
>>> _ = record.update_count().update_checksum().validate()
Traceback (most recent call last):
    ...
ValueError: unexpcted data
```

4.7.3 RawTag

class hexrec.formats.raw.**RawTag**(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Raw binary tag.

Attributes

DATA	Data.

DATA = Ellipsis

Data.

_DATA: Optional['BaseTag'] = Ellipsis

Alias to a common data record tag.

This tag is used internally to build a generic data record.

classmethod __contains__(*member*)

Return True if member is a member of this enum raises TypeError if member is not an enum member

note: in 3.12 TypeError will no longer be raised, and True will also be returned if member is the value of a member in this enum

classmethod __getitem__(*name*)

Return the member matching *name*.

classmethod __iter__()

Return members in definition order.

classmethod __len__()

Return the number of members (no aliases)

__new__(*value*)

_generate_next_value_(*start*, *count*, *last_values*)

Generate the next value when not given.

name: the name of the member start: the initial start value or None count: the number of existing members last_values: the list of values assigned

_member_type_

alias of `object`

_new_member_(***kwargs*)

Create and return a new object. See help(type) for accurate signature.

4.8 sqtp

Microchip Serial Quick Time Programming format.

See also:

https://developerhelp.microchip.com/xwiki/bin/view/software-tools/ipe/sqtp-file-format-specification/

Functions

from_numbers	Creates a file from numbers.
from_strings	Creates a file from byte strings.
to_numbers	Extracts numbers from a file.
to_strings	Extracts byte strings from a file.

4.8.1 from_numbers

hexrec.formats.sqtp.**from_numbers**(*numbers*, *length=2*, *start=0*, *retlw=None*, *wordsize=2*, *byteorder='little'*, *forcedela=False*)

Creates a file from numbers.

Given a sequence of numbers, it creates an *Intel HEX* file with the special record sequence and addressing of *Microchip SQTP*.

Parameters

- **numbers** (*list of int*) – Sequence of serial numbers.
- **length** (*int*) – Serial number byte size.
- **start** (*int*) – Start word address of a serial number within the target memory.
- **retlw** (*int*) – If `None`, this has no effect. If a byte integer is given, it must be the equivalent of the `RETLW` *opcode* of the target processor. The `RETLW` byte is put after each byte of the serial number.
- **wordsize** (*int*) – Memory word size (2 or 4 bytes).
- **byteorder** (*str*) – By default, *Microchip SQTP* uses `little` endian. Provide `big` for the alternative integer byte order.
- **forcedela** (*bool*) – Forces *Extended Linear Address* generation.

Returns

`IhexFile` – *Microchip SQTP* file as special *Intel HEX* format.

Examples

https://developerhelp.microchip.com/xwiki/bin/view/software-tools/ipe/sqtp-file-format-specification/examples/.

```
>>> from hexrec.formats.sqtp import from_numbers
```

```
>>> # Program Memory - PIC18F1220
>>> file = from_numbers(range(5), retlw=0x0C)
>>> _ = file.print()
:04000000000C000CE4
:04000000010C000CE3
:04000000020C000CE2
:04000000030C000CE1
:04000000040C000CE0
:00000001FF
```

```
>>> # Program Memory - PIC32MX360F512L
>>> file = from_numbers(range(5), length=4, start=0x1D000000, wordsize=4)
>>> _ = file.print()
:02000004740086
:040000000000000FC
:040000001000000FB
:040000002000000FA
:040000003000000F9
:040000004000000F8
:00000001FF
```

```
>>> # User ID - PIC12F1501 (with correct checksums)
>>> numbers = [0xCF7E, 0xC590, 0x110B, 0xF3F2, 0x681C]
>>> file = from_numbers(numbers, start=0x8000, retlw=0x34)
>>> _ = file.print()
:020000040001F9
:040000007E34CF3447
:040000009034C5343F
:04000000B34113478
:04000000F234F334AF
:04000001C34683410
:00000001FF
>>> file = from_numbers(numbers, start=0x8000, retlw=0x34, byteorder='big')
>>> _ = file.print()
:020000040001F9
:04000000CF347E3447
:04000000C53490343F
:04000001134 0B3478
:04000000F334F234AF
:0400000068341C3410
:00000001FF
```

```
>>> # User ID - PIC32MX360F512L
>>> file = from_numbers(range(5), length=4, start=0x1FC02FF0, wordsize=4)
>>> _ = file.print()
```

(continues on next page)

(continued from previous page)

```
:020000047F007B
:04BFC000000000007D
:04BFC000010000007C
:04BFC000020000007B
:04BFC000030000007A
:04BFC0000400000079
:00000001FF
```

```
>>> # Auxiliary Memory - dsPIC33EP256MU806
>>> file = from_numbers(range(5), length=4, start=0x7FC000, wordsize=4)
>>> _ = file.print()
:0200000401FFFA
:040000000000000FC
:040000001000000FB
:040000002000000FA
:040000003000000F9
:040000004000000F8
:00000001FF
```

```
>>> # Boot Memory - PIC32MX110F016B
>>> numbers = [0xC78E2639, 0xE277B71F, 0x3D7E1E03, 0xE2646FD5, 0xA7C293F9]
>>> file = from_numbers(numbers, length=4, start=0x1FC00000, wordsize=4)
>>> _ = file.print()
:020000047F007B
:0400000039268EC748
:040000001FB777E2CD
:04000000031E7E3D20
:04000000D56F64E272
:04000000F993C2A707
:00000001FF
>>> file = from_numbers(numbers, length=4, start=0x1FC00000, wordsize=4, byteorder=
↪'big')
>>> _ = file.print()
:020000047F007B
:04000000C78E263948
:04000000E277B71FCD
:040000003D7E1E0320
:04000000E2646FD572
:04000000A7C293F907
:00000001FF
```

```
>>> # EEPROM - PIC12F1840
>>> file = from_numbers(range(5), forcedela=True)
>>> _ = file.print()
:020000040000FA
:020000000000FE
:020000000100FD
:020000000200FC
:020000000300FB
:020000000400FA
:00000001FF
```

```
>>> # EEPROM - PIC18F1220 (with actual start address 0x00780000)
>>> file = from_numbers(range(5), start=0x00780000, forcedela=True)
>>> _ = file.print()
:0200000400F00A
:020000000000FE
:020000000100FD
:020000000200FC
:020000000300FB
:020000000400FA
:00000001FF
```

4.8.2 from_strings

hexrec.formats.sqtp.**from_strings**(*strings*, *start=0*, *retlw=None*, *wordsize=2*, *forcedela=False*)

Creates a file from byte strings.

Given a sequence of byte strings, it creates an *Intel HEX* file with the special record sequence and addressing of *Microchip SQTP*.

All the *strings* must be the same length, between the minimum word size of the processor (minimum 2) and 256.

Parameters

- **strings** (*list of bytes*) – Sequence of byte strings.
- **start** (*int*) – Start word address of a byte string within the target memory.
- **retlw** (*int*) – If None, this has no effect. If a byte integer is given, it must be the equivalent of the RETLW *opcode* of the target processor. The RETLW byte is put after each byte of the byte string.
- **wordsize** (*int*) – Memory word size (2 or 4 bytes).
- **forcedela** (*bool*) – Forces *Extended Linear Address* generation.

Returns

IhexFile – *Microchip SQTP* file as special *Intel HEX* format.

Examples

```
>>> from hexrec.formats.sqtp import from_strings
>>> strings = [b'abcdefghijklm', b'nopqrstuvwxyz', b'ABCDEFGHIJKLM', b'NOPQRSTUVWXYZ
↪']
```

```
>>> file = from_strings(strings, forcedela=True)
>>> _ = file.print()
:020000040000FA
:0D0000006162636465666768696A6B6C6DB8
:0D0000006E6F707172737475767778797A0F
:0D00000041424344454647484 94A4B4C4D58
:0D0000004E4F505152535455565758595AAF
:00000001FF
```

```
>>> file = from_strings(strings, start=0x1FC02FF0, wordsize=4)
>>> _ = file.print()
:020000047F007B
:0DBFC000616263646566768696A6B6C6D39
:0DBFC0006E6F707172737475767778797A90
:0DBFC000414243444546474849 4A4B4C4DD9
:0DBFC0004E4F505152535455565758595A30
:00000001FF
```

```
>>> file = from_strings(strings, start=0x8000, retlw=0x34)
>>> _ = file.print()
:020000040001F9
:1A000000613462346334643465346634673468346934 6A346B346C346D3407
:1A0000006E346F347034713472347334743475347634773478347934 7A345E
:1A000000413442344334443445344634473448344934 4A344B344C344D34A7
:1A0000004E344F345034513452345334543455345634573458345934 5A34FE
:00000001FF
```

4.8.3 to_numbers

hexrec.formats.sqtp.**to_numbers**(*file*, *retlw=False*, *byteorder='little'*)

Extracts numbers from a file.

Given a *Microchip SQTP* file (as special *Intel HEX* format), it extracts numbers from *data* records.

> **Warning:** This algorithm ignores addressing. It just takes *data* records and converts them into numbers. Please provide valid *Microchip SQTP* files only.

Parameters

- **file** (IhexFile) – *Microchip SQTP* file as special *Intel HEX* format.
- **retlw** (*bool*) – The RETLW byte is put after each byte of the byte string. If true, it ignores the RETLW bytes.
- **byteorder** (*str*) – By default, *Microchip SQTP* uses little endian. Provide big for the alternative integer byte order.

Returns

list of int – Sequence of serial numbers.

Examples

https://developerhelp.microchip.com/xwiki/bin/view/software-tools/ipe/sqtp-file-format-specification/examples/.

```
>>> from hexrec import IhexFile
>>> from hexrec.formats.sqtp import to_numbers
```

```
>>> # Program Memory - PIC18F1220
>>> file = IhexFile.parse(b'''
...     :04000000000C000CE4
...     :04000000010C000CE3
...     :04000000020C000CE2
...     :04000000030C000CE1
...     :04000000040C000CE0
...     :00000001FF
... ''')
>>> to_numbers(file, retlw=True)
[0, 1, 2, 3, 4]
```

```
>>> # Program Memory - PIC32MX360F512L
>>> file = IhexFile.parse(b'''
...     :02000004740086
...     :0400000000000000FC
...     :0400000001000000FB
...     :0400000002000000FA
...     :0400000003000000F9
...     :0400000004000000F8
...     :00000001FF
... ''')
>>> to_numbers(file)
[0, 1, 2, 3, 4]
```

```
>>> # User ID - PIC12F1501 (with correct checksums)
>>> file = IhexFile.parse(b'''
...     :020000040001F9
...     :040000007E34CF3447
...     :040000009034C5343F
...     :040000000B34113478
...     :04000000F234F334AF
...     :040000001C34683410
...     :00000001FF
... ''')
>>> to_numbers(file, retlw=True)
[53118, 50576, 4363, 62450, 26652]
>>> to_numbers(file, retlw=True, byteorder='big')
[32463, 37061, 2833, 62195, 7272]
```

```
>>> # User ID - PIC32MX360F512L
>>> file = IhexFile.parse(b'''
...     :020000047F007B
...     :04BFC000000000007D
...     :04BFC000010000007C
...     :04BFC000020000007B
...     :04BFC000030000007A
...     :04BFC00004000000079
...     :00000001FF
... ''')
>>> to_numbers(file)
[0, 1, 2, 3, 4]
```

```
>>> # Auxiliary Memory - dsPIC33EP256MU806
>>> file = IhexFile.parse(b'''
...     :0200000401FFFA
...     :04000000000000FC
...     :04000000010000FB
...     :04000000020000FA
...     :04000000030000F9
...     :04000000040000F8
...     :00000001FF
... ''')
>>> to_numbers(file)
[0, 1, 2, 3, 4]
```

```
>>> # Boot Memory - PIC32MX110F016B
>>> file = IhexFile.parse(b'''
...     :020000047F007B
...     :04000000039268EC748
...     :040000001FB777E2CD
...     :04000000031E7E3D20
...     :04000000D56F64E272
...     :04000000F993C2A707
...     :00000001FF
... ''')
>>> to_numbers(file)
[3347981881, 3799496479, 1031675395, 3798233045, 2814546937]
>>> to_numbers(file, byteorder='big')
[958828231, 532117474, 52330045, 3580847330, 4187210407]
```

```
>>> # EEPROM - PIC12F1840
>>> file = IhexFile.parse(b'''
...     :020000040000FA
...     :020000000000FE
...     :020000000100FD
...     :020000000200FC
...     :020000000300FB
...     :020000000400FA
...     :00000001FF
... ''')
>>> to_numbers(file)
[0, 1, 2, 3, 4]
```

```
>>> # EEPROM - PIC18F1220 (with actual start address 0x00780000)
>>> file = IhexFile.parse(b'''
...     :0200000400F00A
...     :020000000000FE
...     :020000000100FD
...     :020000000200FC
...     :020000000300FB
...     :020000000400FA
...     :00000001FF
... ''')
>>> to_numbers(file)
```

(continues on next page)

(continued from previous page)

```
[0, 1, 2, 3, 4]
```

4.8.4 to_strings

hexrec.formats.sqtp.**to_strings**(*file*, *retlw=False*)

Extracts byte strings from a file.

Given a *Microchip SQTP* file (as special *Intel HEX* format), it extracts byte strings from *data* records.

Warning: This algorithm ignores addressing. It just takes *data* records. Please provide valid *Microchip SQTP* files only.

Parameters

- **file** (IhexFile) – *Microchip SQTP* file as special *Intel HEX* format.
- **retlw** (*bool*) – The RETLW byte is put after each byte of the byte string. If true, it ignores the RETLW bytes.

Returns

list of int – Sequence of byte strings.

Examples

```
>>> from hexrec import IhexFile
>>> from hexrec.formats.sqtp import to_strings
```

```
>>> file = IhexFile.parse(b'''
...     :020000040000FA
...     :0D00000061626364656666768696A6B6C6DB8
...     :0D0000006E6F70717273747475767778797A0F
...     :0D00000041424344454647484944A4B4C4D58
...     :0D0000004E4F505152535455565758595AAF
...     :00000001FF
... ''')
>>> to_strings(file)
[b'abcdefghijklm', b'nopqrstuvwxyz', b'ABCDEFGHIJKLM', b'NOPQRSTUVWXYZ']
```

```
>>> file = IhexFile.parse(b'''
...     :020000047F007B
...     :0DBFC000061626364656666768696A6B6C6D39
...     :0DBFC0006E6F70717273747475767778797A90
...     :0DBFC00041424344454647484944A4B4C4DD9
...     :0DBFC0004E4F505152535455565758595A30
...     :00000001FF
... ''')
>>> to_strings(file)
[b'abcdefghijklm', b'nopqrstuvwxyz', b'ABCDEFGHIJKLM', b'NOPQRSTUVWXYZ']
```

```
>>> file = IhexFile.parse(b'''
...     :020000040001F9
...     :1A000000613462346334643465346634673468346934 6A346B346C346D3407
...     :1A0000006E346F3470347134723473347334753476347734783479347A345E
...     :1A000000413442344334443445344634473448344934 4A344B344C344D34A7
...     :1A0000004E344F3450345134523453345434553456345734583459345A34FE
...     :00000001FF
... ''')
>>> to_strings(file, retlw=True)
[b'abcdefghijklm', b'nopqrstuvwxyz', b'ABCDEFGHIJKLM', b'NOPQRSTUVWXYZ']
```

4.9 srec

Motorola S-record format.

See also:

https://en.wikipedia.org/wiki/SREC_(file_format)

Attributes

SIZE_TO_ADDRESS_FORMAT	Format byte string for each supported address size.

4.9.1 SIZE_TO_ADDRESS_FORMAT

hexrec.formats.srec.**SIZE_TO_ADDRESS_FORMAT: Mapping[int, bytes] = {2: b'%04X', 3: b'%06X', 4: b'%08X'}**

Format byte string for each supported address size.

Classes

SrecFile	Motorola S-record file object.
SrecRecord	Motorola S-record record object.
SrecTag	Motorola S-record tag.

4.9.2 SrecFile

class hexrec.formats.srec.**SrecFile**

Motorola S-record file object.

Attributes

DEFAULT_DATALEN	Default data attribute length.
FILE_EXT	Supported filename extensions.
META_KEYS	Meta information key names.
header	Header byte string.
maxdatalen	Maximum byte size of the data field.
memory	Memory object stored by records role.
records	Records stored by records role.
startaddr	Start address.

Methods

__init__	
align	Pads blocks to align their boundaries.
append	Appends a byte.
apply_records	Applies records to memory and meta.
clear	Clears data within a range.
convert	Converts a file object to another format.
copy	Copies within a range.
crop	Clears data outside a range.
cut	Cuts data within a range.
delete	Deletes data within a range.
discard_memory	Discards underlying memory.
discard_records	Discards underlying records.
extend	Concatenates data.
fill	Fills a range.
find	Finds a substring.
flood	Floods a range.
from_blocks	Creates a file object from a memory object.
from_bytes	Creates a file object from a byte string.
from_memory	Creates a file object from a memory object.
from_records	Creates a file object from records.
get_address_max	Maximum address within memory.
get_address_min	Minimum address within memory.
get_holes	List of memory holes.
get_meta	Meta information.
get_spans	List of memory block spans.
index	Finds a substring.
load	Loads a file object from the filesystem.
merge	Merges data onto the file.
parse	Parses records from a byte stream.
print	Prints record content to stdout.
read	Extracts a substring.
save	Saves a file object into the filesystem.
serialize	Serializes records onto a byte stream.
set_meta	Sets meta information.
shift	Shifts data addresses by an offset.
split	Splits into parts.

continues on next page

Table 7 – continued from previous page

update_records	Applies memory and meta to records.
validate_records	Validates records.
view	Memory view.
write	Writes data into the file.

DEFAULT_DATALEN: int = 16

Default data attribute length.

Default value for the *maxdatalen* *meta*, which sets the maximum size of `BaseRecord.data` field values.

FILE_EXT: Sequence[str] = ['.s19', '.s28', '.s37', '.s', '.s1', '.s2', '.s3', '.sx', '.srec', '.exo', '.mot', '.mxt']

Supported filename extensions.

Sequence of file name extension substrings (e.g. `.hex`). This list is used by functions like `guess_format_name()` to manage mapping of file *formats*.

META_KEYS: Sequence[str] = ['header', 'maxdatalen', 'startaddr']

Meta information key names.

Sequence of key strings listing the supported *meta* information of this file *format*.

Record

alias of *SrecRecord*

__add__(*other*)

Concatenates with another file.

Equivalent to *copy()* then *extend()*.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – Concatenation of *self* and *other*.

See also:

copy() extend()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 + file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file4 = file3 + b'789'
>>> file4.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__bool__()

bool: Has data records or memory.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> bool(file)
False
>>> _ = file.append(0)
>>> bool(file)
True
```

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> file = IhexFile.from_records([IhexRecord.create_end_of_file()])
>>> bool(file)
False
>>> file.records.insert(0, IhexRecord.create_data(0, b'\0'))
>>> bool(file)
True
```

__delitem__(*key*)

Deletes a range.

Parameters
key (*slice or int*) – Range to delete.

See also:

`bytesparse.base.MutableMemory.__delitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> del file[457]
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xz']]
>>> del file[125:457]
>>> file.memory.to_blocks()
[[123, b'abz']]
```

__eq__(*other*)

Equality test.

The file objects *self* and *other* are considered *equal* if the inequality tests of *__ne__()* result false.

Returns
bool – *self* and *other* are *equal*.

See Also
__ne__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 == file2
True
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 == file3
False
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 == file4
False
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file == ihex_file
False
>>> srec_file.memory == ihex_file.memory
True
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__getitem__(*key*)

Extracts a range.

Parameters
key (*slice or int*) – Range to extract.

Raises
ValueError – invalid range.

See also:

`bytesparse.base.ImmutableMemory.__getitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> chr(file[457])
'y'
>>> repr(file[333])
'None'
>>> file[123:125]
```

(continues on next page)

(continued from previous page)

```
b'ab'
>>> file[125:457]
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

__hash__ = None

__iadd__(*other*)

Concatenates data.

Equivalent to *extend()*.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – *self*.

See also:

memory *extend()* *discard_records()* bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 += file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file1 += b'789'
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__init__()

__ior__(*other*)

Merges with another file.

Equivalent to *merge()*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to merge.

Returns
BaseFile – *self*.

See also:

merge() discard_records()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 |= file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file1 |= b'789'
>>> file1.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__ne__(*other*)

Inequality test.

The file objects *self* and *other* are considered *unequal* if any of the following tests result true:

- Both have *memory role* (i.e. *memory*), resulting unequal;
- Both have *records role* (i.e. *records*), resulting unequal;
- *other* does not have a *meta* listed by *META_KEYS*;
- A *meta* value (among those of *META_KEYS*) is different.

Returns
bool – *self* and *other* are *unequal*.

See also:

__eq__() memory records META_KEYS

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 != file2
False
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 != file3
True
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
```

(continues on next page)

(continued from previous page)

```
>>> file1 != file4
True
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file != ihex_file
True
>>> srec_file.memory != ihex_file.memory
False
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__or__(*other*)

Merges with another file.

Equivalent to *copy()* then *merge()*.

Parameters
other (BaseFile or bytes) – Other file or bytes to merge.

Returns
BaseFile – *self* merged with *other*.

See also:

copy() *merge()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 | file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file4 = file3 | b'789'
>>> file4.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__setitem__(*key*, *value*)

Sets a range.

Parameters

- **key** (*slice or int*) – Range to set.
- **value** (bytes, bytesparse.base.ImmutableMemory, None) – Value(s) to set. None acts like *clear()*.

Raises
ValueError – invalid range.

See also:

`bytesparse.base.MutableMemory.__setitem__()` *clear()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file[124] = b'?'
>>> file.memory.to_blocks()
[[123, b'a?c'], [456, b'xyz']]
>>> file[:125] = None
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'xyz']]
>>> file[457:458] = b'789'
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'x789z']]
```

__weakref__

list of weak references to the object (if defined)

classmethod _is_line_empty(*line*)

Empty line check.

Tells whether a *line* has no meaningful content (e.g. all whitespace). The check itself depends on the implementing file *format*. It may be used internally to skip empty lines, e.g. by *parse()*.

Parameters
line (*bytes*) – A line, byte string.

Returns
bool: The *line* is empty.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexFile._is_line_empty(b'')
True
>>> IhexFile._is_line_empty(b' \t\v\r\n')
True
>>> IhexFile._is_line_empty(b':00000001FF\r\n')
False
```

align(*modulo*, *start=None*, *endex=None*, *pattern=0*)

Pads blocks to align their boundaries.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*, so that memory blocks are aligned to the required *modulo*.

Any stored *records* are discarded upon return.

Parameters

- **modulo** (*int*) – Alignment modulo.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.align()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [134, b'xyz']])
>>> _ = file.align(4, pattern=b'.')
>>> file.memory.to_blocks()
[[120, b'...abc..'], [132, b'..xyz...']]
```

append(*item*)

Appends a byte.

It appends the *item* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters

item (*byte or int*) – Byte to append.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.append()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123)
>>> _ = file.append(b'.')
>>> _ = file.append(0)
>>> file.memory.to_blocks()
[[123, b'abc.\x00']]
```

apply_records()

Applies records to memory and meta.

This method processes the stored *records*, converting *data* as *memory*, and special records into their *meta* counterparts.

This effectively converts the *records role* into the *memory role* (keeping both).

The *memory* and *meta* are assigned upon return. Any exceptions being raised should not alter the file object.

Returns
BaseFile – *self*.

Raises
ValueError – *records* attribute not populated.

See also:

records *memory* *get_meta()* *update_records()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_start_linear_address(456),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records, maxdatalen=16)
>>> _ = file.apply_records()
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
```

clear(*start=None*, *endex=None*)

Clears data within a range.

It clears the specified range of underlying *memory* object, making a memory hole.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
BaseFile – *self*.

See also:

memory *discard_records()* bytesparse.base.MutableMemory.clear()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.clear(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

classmethod convert(*source*, *meta=True*)

Converts a file object to another format.

It copies the *memory* and *meta* of the *source* file object, creating a new one of the target `BaseFile` format type.

Parameters

- **source** (`BaseFile`) – Source file object to convert.
- **meta** (*bool*) – Copy *meta* information to the target file object. Only the keys of the target *META_KEYS* are processed.

Returns

`BaseFile` – Converted copy of *source* to the target format.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> source = IhexFile.from_blocks(blocks, startaddr=789)
>>> target = SrecFile.convert(source)
>>> target.memory is source.memory
False
>>> target.memory == source.memory
True
>>> target.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 789}
```

copy(*start=None*, *endex=None*, *meta=True*)

Copies within a range.

It copied data within the specified range of the file object, creating a new one carrying the inner slice.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns
BaseFile – *self*.

See also:

memory *get_meta()* *discard_records()* bytesparse.base.MutableMemory.cut()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.copy(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

crop(*start=None*, *endex=None*)

Clears data outside a range.

It clears outside the specified range of underlying *memory* object, timming it.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
BaseFile – *self*.

See also:

memory *discard_records()* bytesparse.base.MutableMemory.crop()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.crop(start=124, endex=132)
>>> file.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
```

cut(*start=None*, *endex=None*, *meta=False*)

Cuts data within a range.

It takes data within the specified range away from the file object, creating a new one carrying the inner slice. The inner slice is cleared from *self*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

`BaseFile` – *self*.

See also:

memory *clear()* *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.cut(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

delete(*start=None*, *endex=None*)

Deletes data within a range.

It deletes the specified range of underlying *memory* object, shifting all subsequent data towards the collapsed range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.delete()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.delete(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'az']]
```

discard_memory()

Discards underlying memory.

The underlying *memory* object is assigned `None`.

If the underlying *records* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_bytes(b'abc', offset=123)
>>> _ = file.update_records()
>>> _ = file.discard_memory()
>>> _ = file.update_records()
Traceback (most recent call last):
    ...
ValueError: memory instance required
```

discard_records()

Discards underlying records.

The underlying *records* object is assigned `None`.

If the underlying *memory* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
```

(continues on next page)

(continued from previous page)

```
>>> _ = file.validate_records()
>>> _ = file.discard_records()
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: records required
```

extend(*other*)

Concatenates data.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> _ = file1.extend(file2)
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> _ = file1.extend(b'789')
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

fill(*start=None*, *endex=None*, *pattern=0*)

Fills a range.

It writes a *pattern* of bytes onto the underlying *memory* object, overwriting anything within the specified range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for filling.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.fill()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.fill(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'a........z']]
```

find(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it returns -1.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
int – *item* beginning address; -1 if not found.

See also:

index bytesparse.base.ImmutableMemory.find()

Notes

The internal *memory* might allow negative addresses for its stored data. In that case, *index()* would be more appropriate, because it raises an exception when the *item* is not found.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.find(b'yz')
457
>>> file.find(ord('b'))
```

(continues on next page)

(continued from previous page)

```
124
>>> file.find(b'?')
-1
```

flood(*start=None*, *endex=None*, *pattern=0*)

Floods a range.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.flood()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.get_holes()
[(126, 130)]
>>> _ = file.flood(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'abc....xyz']]
```

classmethod from_blocks(*blocks*, ***meta*)

Creates a file object from a memory object.

The *blocks* are put into the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **blocks** (*list of blocks*) – Memory blocks to put into *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

`BaseFile` – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_blocks()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_bytes(*data*, *offset=0*, ***meta*)

Creates a file object from a byte string.

The byte string makes a single *data* block, placed at some offset within the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **data** (*bytes*) – A byte string used to make a single data block.
- **offset** (*int*) – Offset of the single data block within *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
`BaseFile` – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_bytes()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
8
```

classmethod from_memory(*memory=None*, ***meta*)

Creates a file object from a memory object.

The *memory* is set as the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **memory** (bytesparse.base.MutableMemory) – Memory object to set as *memory*. If None, an empty memory object is automatically created.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

BaseFile – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS bytesparse.base.MutableMemory

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from bytesparse import Memory
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> memory = Memory.from_blocks(blocks)
>>> from hexrec import SrecFile
>>> file = SrecFile.from_memory(memory, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_records(*records*, *maxdatalen=None*)

Creates a file object from records.

The *records* sequence is set as the record attribute of the created file object.

This method creates a file object in *records role*. This means that only its *records* is internally instanced, while the *memory* requires manual or lazy instancing (i.e. either via direct call to *apply_records()*, or any other methods indirectly calling it).

Parameters

- **records** (list of BaseRecord) – Record sequence to set as *records*.
- **maxdatalen** (Optional[int]) – Maximum record *data* field size. If None, the maximum non-zero size of the *data* field from the *records* sequence is used. If all the *records* have zero sized *data* field, the class attribute *DEFAULT_DATALEN* is used.

Returns

BaseFile – The created file object.

Raises
ValueError – invalid *meta* values.

See also:

`BaseRecord`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
3
```

get_address_max()

Maximum address within memory.

It returns the maximum address of the underlying *memory* object.

Returns
int – Maximum address.

See also:

`bytesparse.base.ImmutableMemory.endin`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_max()
458
```

get_address_min()

Minimum address within memory.

It returns the minimum address of the underlying *memory* object.

Returns
int – Minimum address.

See also:

`bytesparse.base.ImmutableMemory.start`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_min()
123
```

get_holes()

List of memory holes.

It scans the underlying *memory* and returns the list of memory holes/gaps.

Each hole is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory hole boundaries.

See also:

`bytesparse.base.ImmutableMemory.gaps()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_holes()
[(126, 456), (459, 789)]
```

get_meta()

Meta information.

It builds and returns a dictionary of *meta* information. Meta keys are taken from the *META_KEYS* class attribute.

Returns
dict – Meta information dictionary.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks, header=b'HDR\0')
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 0}
```

get_spans()

List of memory block spans.

It scans the underlying *memory* and returns the list of memory block spans/intervals.

Each span is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory block boundaries.

See also:

`bytesparse.base.ImmutableMemory.intervals()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_spans()
[(123, 126), (456, 459), (789, 791)]
```

property header: bytes | bytearray | memoryview | None

Header byte string.

This property sets the file *header* byte string; `None` to disable.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises
ValueError – data size overflow.

See also:

update_records() *discard_records()*

Examples

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> file.header
b''
>>> _ = file.print()
S0030000FC
S5030000FC
S9030000FC
>>> file.header = b'HDR\0'
>>> _ = file.print()
S0070000484452001A
S5030000FC
S9030000FC
>>> file.header = None
```

(continues on next page)

(continued from previous page)

```
>>> _ = file.print()
S5030000FC
S9030000FC
```

Type
bytes

index(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it raises `ValueError`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
int – *item* beginning address.

Raises
ValueError – *item* not found.

See also:

find `bytesparse.base.ImmutableMemory.index()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.index(b'yz')
457
>>> file.index(ord('b'))
124
>>> file.index(b'?')
Traceback (most recent call last):
    ...
ValueError: subsection not found
```

classmethod load(*path*, **args*, ***kwargs*)

Loads a file object from the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *parse()* to load a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdin.buffer` is used.

- **args** – Forwarded to *parse()*.
- **kwargs** – Forwarded to *parse()*.

Returns
BaseFile – Loaded file object.

See also:

save() *parse()* open() sys.stdin.buffer

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.load('data.hex')
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

property maxdatalen: int

Maximum byte size of the data field.

This property sets the maximum byte size of the *data* field of a serialized record.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises
ValueError – Invalid maximum data length.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> file.maxdatalen
16
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212222324252627282 92A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
```

(continues on next page)

(continued from previous page)

```
>>> file.maxdatalen = 8
>>> _ = file.print()
S0030000FC
S10B0000000102030405060708
S10B000808090A0B0C0D0E0F90
S10B001010111213141516174B
S10B001818191A1B1C1D1E1F00
S10B00202021222324252627B8
S10B002828292A2B2C2D2E2F70
S10B00303031323334353637E8
S10B003838393A3B3C3D3E3FE0
S5030008F4
S9030000FC
>>> file.maxdatalen = 0
Traceback (most recent call last):
    ...
ValueError: invalid maximum data length
```

Type
int

property memory: MutableMemory

Memory object stored by records role.

This readonly property exposes the memory object stored by the file object while in *memory role*.

If this property is accessed while the file object is not in *memory role*, it automatically activates it by an implicit call to *apply_records()*, with default arguments.

For more control activating the *memory role*, please call *apply_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *records role* (i.e. altering content of *records*) would implicitly discard *memory* via *discard_memory()*.

See also:

apply_records() *discard_memory()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> _ = file.write(789, b'?!')
```

(continues on next page)

(continued from previous page)

```
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz'], [789, b'?!']]
```

Type
bytesparse.Memory

merge(**files*, *clear=False*)

Merges data onto the file.

It writes the provided *files* onto *self*, in the provided order. Any common address ranges are overwritten.

Any stored *records* are discarded upon return.

Parameters

- **files** (BaseFile) – Files to merge.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns
BaseFile – *self*.

See also:

clear() discard_records() write()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = SrecFile.from_bytes(b'<<<?????>>>', offset=450)
>>> _ = file3.merge(file1, file2)
>>> file3.memory.to_blocks()
[[123, b'abc'], [450, b'<<<???xyz>>']]
```

classmethod parse(*stream*, *ignore_errors=False*, *ignore_after_termination=True*)

Parses records from a byte stream.

It executes BaseRecord.parse() for each line of the incoming *stream*, creating a new file object with the collected records calling *from_records()*.

Lines resulting empty by _is_empty_line() are just discarded.

Notes

Please refer to the actual implementation of each record file *format*, because it may be more specialized.

Parameters

- **stream** (*bytes IO or buffer*) – Stream or byte buffer to parse records from.
- **ignore_errors** (*bool*) – Ignore `Exception` raised by `BaseRecord.parse()`.
- **ignore_after_termination** (*bool*) – Ignore anything after the termination record was parsed, if supported (e.g. *End Of File* or *start address* record, depending on the specific file *format*).

Returns

`BaseFile` – *self*.

See also:

parse() `BaseRecord.parse()` *from_records()* `_is_empty_line()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> buffer = b'''
...     :03DA7A0061626383
...     :040000050000CAFE2F
...     :00000001FF
... '''
>>> import io
>>> stream = io.BytesIO(buffer)
>>> file = IhexFile.parse(stream)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
>>> file = IhexFile.parse(buffer)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

print(**args*, *stream=None*, *color=False*, *start=None*, *stop=None*, ***kwargs*)

Prints record content to stdout.

This helper method prints each record of *records* via `BaseRecord.print()`. As such, it also supports colored tokens and streams different from *stdout*.

It is possible to print subset of the records by specifying the record index range.

> **Warning:** This method is **NOT** equivalent to *serialize()*, because it just prints each record from *records*. Please use *serialize()* for an actual serialization of the whole file.

Parameters

- **args** – Forwarded to the underlying call to `to_tokens()`.
- **stream** (*byte stream*) – Stream to print onto. If `None`, *stdout* is used.
- **color** (*bool*) – Colorize record tokens with ANSI color codes.
- **start** (*int*) – Inclusive start record index of the specified range. If `None`, start from the first record.
- **stop** (*int*) – Exclusive end record index of the specified range. If negative, look back from the last index. If `None`, print up to the last record.
- **kwargs** – Forwarded to the underlying call to `to_tokens()`.

Returns

`BaseFile` – *self*.

See also:

`BaseRecord.print()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> _ = file.print(color=True, start=1, stop=-2)
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
```

read(*start=None*, *endex=None*, *fill=0*)

Extracts a substring.

It extracts a byte string from the specified range, filling any memory holes/gaps (without altering *memory*).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **fill** (*bytes or int*) – Byte pattern for filling.

Returns
BaseFile – *self*.

See also:

memory bytesparse.base.MutableMemory.extract() bytesparse.base.MutableMemory.to_bytes()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.read(start=124, endex=132)
b'bc\x00\x00\x00\x00xy'
>>> file.read(start=124, endex=132, fill=b'.')
b'bc....xy'
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

property records: MutableSequence[*BaseRecord*]

Records stored by records role.

This readonly property exposes the list of records stored by the file object while in *records role*.

If this property is accessed while the file object is not in *records role*, it automatically activates it by an implicit call to *update_records()*, with default arguments.

For more control activating the *records role*, please call *update_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *memory role* (i.e. altering content of *memory*) would implicitly discard *records* via *discard_records()*.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, startaddr=789)
>>> len(file.records)
5
>>> _ = file.print()
S0030000FC
S106007B61626358
```

(continues on next page)

(continued from previous page)

```
S10601C878797AC5
S5030002FA
S9030315E4
>>> _ = file.update_records(data_tag=SrecFile.Record.Tag.DATA_32)
>>> _ = file.print()
S0030000FC
S3080000007B61626356
S308000001C878797AC3
S5030002FA
S70500000315E2
```

Type
list of `BaseRecord`

save(*path*, **args*, ***kwargs*)

Saves a file object into the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *serialize()* to save a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdout.buffer` is used.
- **args** – Forwarded to *serialize()*.
- **kwargs** – Forwarded to *serialize()*.

Returns
`BaseFile` – *self*.

See also:

load() *serialize()* `open()` `sys.stdout.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> _ = file.save('data.hex')
```

serialize(*stream*, **args*, ***kwargs*)

Serializes records onto a byte stream.

It executes `BaseRecord.serialize()` for each of the stored *records*.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **args** – Forwarded to `BaseRecord.serialize()` of each record.
- **kwargs** – Forwarded to `BaseRecord.serialize()` of each record.

Returns
`BaseFile` – *self*.

See also:

parse() BaseRecord.serialize()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> import sys
>>> _ = file.serialize(sys.stdout.buffer, end=b'\n')
:03DA7A0061626383
:040000050000CAFE2F
:00000001FF
```

set_meta(*meta*, *strict=True*)

Sets meta information.

It sets the provided *kwargs* to their matching *meta* attributes, as listed by *META_KEYS*.

Parameters

- **meta** (*dict*) – Mapping of the *meta* information to set.
- **strict** (*bool*) – All the keys within *meta* must exist within *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *get_meta()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 0}
>>> _ = file.set_meta(dict(header=b'HDR\0', startaddr=456))
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 456}
```

shift(*offset*)

Shifts data addresses by an offset.

It shifts addresses of the underlying *memory* object data blocks by the provided *offset* amount.

Any stored *records* are discarded upon return.

Parameters
offset (*int*) – Offset to apply to the underlying data block addresses.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.shift()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> _ = file.shift(1000)
>>> file.memory.to_blocks()
[[1123, b'abc'], [1456, b'xyz']]
```

split(**addresses*, *meta=True*)

Splits into parts.

The provided *addresses* are sorted and used as markers to split *self* into parts.

Each part is the *copy()* of *self* within the range of that part, in *memory role* (i.e., *records* is not populated).

Parameters

- **addresses** (*int*) – Split points.
- **meta** (*bool*) – Each part inherits *meta* from *self*.

Returns
list of BaseFile – Parts after splitting.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'Hello, World!', offset=123)
>>> parts = file.split(128, 130)
>>> for part in parts: print(part.memory.to_blocks())
[[123, b'Hello']]
[[128, b', ']]
[[130, b'World!']]
>>> file.memory.to_blocks()
[[123, b'Hello, World!']]
```

property startaddr: int

Start address.

This property sets the *start address* of the serialized record file.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Examples

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> file.startaddr
0
>>> _ = file.print()
S0030000FC
S5030000FC
S9030000FC
>>> file.startaddr = 0x87654321
>>> _ = file.print()
S0030000FC
S5030000FC
S70587654321AA
```

update_records(*align=False*, *header=True*, *data=False*, *count=True*, *start=True*, *data_tag=None*, *count_tag=None*)

Applies memory and meta to records.

This method processes the stored *memory* and *meta* information to generate the sequence of *records*.

This effectively converts the *memory role* into the *records role* (keeping both).

The *records* is assigned upon return. Any exceptions being raised should not alter the file object.

Parameters

- **align** (*bool*) – Aligns data record chunk address bounds to *maxdatalen*.
- **header** (*bool*) – Generates the *header* record if *header*.
- **data** (*bool*) – Requires at least one *data* record be present, even if empty.
- **count** (*bool*) – Generates the *count* record.
- **start** (*bool*) – Generates the *start address* record.
- **data_tag** (*SrecTag*) – Specific *data* record tag to use.
- **count_tag** (:class:`SrecTag`git) – Specific *count* record tag to use.

Returns

SrecFile – *self*.

Raises

ValueError – *memory* attribute not populated.

See also:

records *memory* *get_meta()* *apply_records()*

Examples

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=16, startaddr=456)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 456}
>>> _ = file.update_records()
>>> len(file.records)
4
>>> _ = file.print()
S0030000FC
S106007B61626358
S5030001FB
S90301C833
```

validate_records(*header_required=False*, *header_first=True*, *data_ordering=False*, *data_uniform=True*, *count_required=False*, *count_penultimate=True*, *start_last=True*, *start_within_data=False*)

Validates records.

It performs consistency checks for the underlying *records*.

Parameters

- **header_required** (*bool*) – Requires the *header* record be present.
- **header_first** (*bool*) – Requires the *header* record be the first of the sequence.
- **data_ordering** (*bool*) – Checks that the *data* record sequence has monotonically increasing addresses, without any overlapping.
- **data_uniform** (*bool*) – Requires *data* records have the same tag.
- **count_required** (*bool*) – Requires the *count* record be present.
- **count_penultimate** (*bool*) – Requires the *start address* record be the penultimate one.
- **start_last** (*bool*) – Requires the *start address* record be the last of the sequence.
- **start_within_data** (*bool*) – Requires *start address* fall within data carried by some *data* record.

Returns

SrecFile – *self*.

Raises

ValueError – Invalid record sequence.

Examples

```
>>> from hexrec import SrecFile
>>> records = [SrecFile.Record.create_data(123, b'abc')]
>>> file = SrecFile.from_records(records)
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: missing start record
```

view(*start=None*, *endex=None*)

Memory view.

It returns a `memoryview` over the specified range, which must cover a *contiguous* data region (i.e. no memory holes within).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

memoryview – View of the specified range.

Raises

ValueError – non-contiguous data within range.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> bytes(file.view(start=456, endex=458))
b'xy'
>>> bytes(file.view())
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

write(*address*, *data*, *clear=False*)

Writes data into the file.

It writes the provided *data* into the underlying *memory* object.

Any stored *records* are discarded upon return.

Parameters

- **address** (*int*) – Address where *data* has to be written.
- **data** (*bytes or memory*) – Byte data to write.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns
BaseFile – *self*.

See also:

memory clear() discard_records() bytesparse.base.MutableMemory.write()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> _ = file.write(123, b'abc')
>>> _ = file.write(555, ord('?'))
>>> _ = file.write(1000, SrecFile.from_bytes(b'xyz', offset=456))
>>> file.memory.to_blocks()
[[123, b'abc'], [555, b'?'], [1456, b'xyz']]
```

4.9.3 SrecRecord

class hexrec.formats.srec.**SrecRecord**(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

Motorola S-record record object.

Attributes

EQUALITY_KEYS	Meta keys for equality checks.
LINE1_REGEX	Line parser regex, part 1.
LINE2_REGEX	Line parser regex, part 2.
LINE3_REGEX	Line parser regex, part 3.
META_KEYS	Meta keys.

Methods

__init__	
compute_checksum	Computes the checksum field value.
compute_count	Compute the count field value.
copy	Shallow copy.
create_count	Creates a record count record.
create_data	Creates a data record.
create_header	Creates a header record.
create_start	Creates a start address record.
data_to_int	Interprets data bytes as integer.
get_meta	Gets meta information.
parse	Parses a record from bytes.
print	Prints a record.
serialize	Serializes onto a stream.
to_bytestr	Converts into a byte string.
to_tokens	Converts into byte string tokens.
update_checksum	Updates the checksum field.
update_count	Updates the count field.
validate	Validates consistency of attribute values.

EQUALITY_KEYS: Sequence[str] = ['address', 'checksum', 'count', 'data', 'tag']

Meta keys for equality checks.

Equality methods (*__eq__()* and *__ne__()*) check against these *meta* keys only. Any other *meta* keys are just ignored.

LINE1_REGEX =
re.compile(b'^(?P<before>\\s*)[Ss](?P<tag>[0-9A-Fa-f])(?P<count>[0-9A-Fa-f]{2})')

Line parser regex, part 1.

LINE2_REGEX = [re.compile(b'^(?P<address>[0-9A-Fa-f]{4})'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{6})'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{8})')]

Line parser regex, part 2.

LINE3_REGEX = re.compile(b'^(?P<data>([0-9A-Fa-f]{2})*)(?
P<checksum>[0-9A-Fa-f]{2})(?P<after>[^\\r\\n]*)\\r?\\n?$')

Line parser regex, part 3.

META_KEYS: Sequence[str] = ['address', 'after', 'before', 'checksum', 'coords',
'count', 'data', 'tag']

Meta keys.

This sequence holds the *meta* keys for copying (see *copy()*).

Tag

alias of *SrecTag*

__bytes__()

Serializes the record into bytes.

Returns
bytes – Byte serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> bytes(record)
b':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> bytes(record)
b'abc'
```

__eq__(*other*)

Equality test.

This method returns true if *self* is considered equal to *other*.

As inequality is usually easier to check, this method is usually implemented as a trivial `not self != other` (*__ne__()*).

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* equals *other*.

See also:

__ne__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 == ihex2
True
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 == ihex3
False
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 == raw
False
```

__hash__ = None

__init__(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

__ne__(*other*)

Ineuality test.

This method returns true if *self* is considered unequal to *other*.

Each attribute listed by *EQUALITY_KEYS* is compared between *self* and *other*. This method returns whether any attributes do not match.

Parameters
other (BaseRecord) – Record to compare to.

Returns
bool – *self* and *other* are unequal.

See also:

__eq__()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 != ihex2
False
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 != ihex3
True
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 != raw
True
```

__repr__()

String representation.

It returns a string representation of the record content, for human understanding only.

Returns
str – String representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> repr(record)
"<<class 'hexrec.formats.ihex.IhexRecord'> @...
  address:=0 after:=b'' before:=b'' checksum:=255 coords:=(-1, -1)
  count:=0 data:=b'' tag:=<IhexTag.END_OF_FILE: 1>>"
```

__str__()

Serializes the record into a string.

Returns

str – String serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> str(record)
'abc'
```

__weakref__

list of weak references to the object (if defined)

compute_checksum()

Computes the checksum field value.

It computes and returns the format-specific `checksum` value of a record.

When not specialized, it returns `None` by default.

Returns

int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_checksum()
215
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_checksum())
'None'
```

compute_count()

Compute the count field value.

It computes and returns the format-specific `count` value of a record.

When not specialized, it returns `None` by default.

Returns
int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_count()
3
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_count())
'None'
```

copy(*validate=True*)

Shallow copy.

It calls the record constructor, passing *meta* to it.

Parameters
validate (*bool*) – Performs validation on instantiation (*__init__()*).

Returns
`BaseRecord` – Shallow copy.

See also:

__init__() *get_meta()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record1 = IhexFile.Record.create_data(0x1234, b'abc')
>>> record2 = record1.copy()
>>> record1 is record2
False
>>> record1 == record2
True
```

classmethod create_count(*count*, *tag=None*)

Creates a record count record.

This is method instantiates a *record count* record, optionally choosing the desired count size.

Parameters

- **count** (*int*) – Number of preceding *data* records.
- **tag** (*SrecTag*) – Chosen *record count* tag. If `None`, it uses the one returned by *SrecTag.fit_count_tag()*.

Returns

SrecRecord – Record count record object.

Examples

```
>>> from hexrec import SrecFile
>>> record = SrecFile.Record.create_count(0x1234)
>>> str(record)
'S5031234B6\r\n'
>>> tag = SrecFile.Record.Tag.COUNT_24
>>> record = SrecFile.Record.create_count(0x1234, tag=tag)
>>> str(record)
'S604001234B5\r\n'
```

classmethod create_data(*address*, *data*, *tag=None*)

Creates a data record.

This is method instantiates a *data* record, optionally choosing the desired address size.

Parameters

- **address** (*int*) – Record address.
- **data** (*bytes*) – Record byte data.
- **tag** (*SrecTag*) – Chosen *data* tag. If `None`, it uses the one returned by *SrecTag.fit_data_tag()*.

Returns

SrecRecord – Data record object.

Examples

```
>>> from hexrec import SrecFile
>>> record = SrecFile.Record.create_data(0x1234, b'abc')
>>> str(record)
'S10612346162638D\r\n'
>>> tag = SrecFile.Record.Tag.DATA_32
>>> record = SrecFile.Record.create_data(0x1234, b'abc', tag=tag)
>>> str(record)
'S308000012346162638B\r\n'
```

classmethod create_header(*data=b''*)

Creates a header record.

Parameters
data (*bytes*) – Header byte data.

Returns
IhexRecord – Header record.

Raises
ValueError – data size overflow.

Examples

```
>>> from hexrec import SrecFile
>>> record = SrecFile.Record.create_header()
>>> str(record)
'S0030000FC\r\n'
>>> record = SrecFile.Record.create_header(b'HDR\0')
>>> str(record)
'S00700004844452001A\r\n'
```

classmethod create_start(*address=0*, *tag=None*)

Creates a start address record.

This is method instantiates a *start address* record, optionally choosing the desired address size.

Parameters

- **address** (*int*) – Start address.
- **tag** (*SrecTag*) – Chosen *start* tag. If None, it uses the one matching *SrecTag.fit_start_tag()*.

Returns
SrecRecord – Start address record object.

Examples

```
>>> from hexrec import SrecFile
>>> record = SrecFile.Record.create_start(0x1234)
>>> str(record)
'S9031234B6\r\n'
>>> tag = SrecFile.Record.Tag.START_32
>>> record = SrecFile.Record.create_start(0x1234, tag=tag)
>>> str(record)
'S70500001234B4\r\n'
```

data_to_int(*byteorder='big'*, *signed=False*)

Interprets data bytes as integer.

It creates an integer from bytes of the `data` field.

Parameters

- **byteorder** (*'big' or 'little'*) – Byte order (endianness): either `'big'` (default) or `'little'`.
- **signed** (*bool*) – Signed integer (2-complement); default false.

Returns

int – Interpreted integer value.

See also:

`int.from_bytes()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0xABCD)
>>> record.data
b'\xab\xcd'
>>> addrext = record.data_to_int()
>>> addrext, hex(addrext)
(43981, '0xabcd')
```

get_meta()

Gets meta information.

It returns all the object attributes whose keys are listed by *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

See also:

META_KEYS `set_meta()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> record.get_meta()
{'address': 0, 'after': b'', 'before': b'', 'checksum': 255,
 'coords': (-1, -1), 'count': 0, 'data': b'',
 'tag': <IhexTag.END_OF_FILE: 1>}
```

classmethod parse(*line*, *validate=True*)

Parses a record from bytes.

Please refer to the actual implementation provided by the record *format* for more details.

Parameters

- **line** (*bytes*) – String of bytes to parse.
- **validate** (*bool*) – Perform validation checks.

Returns

`BaseRecord` – Parsed record.

Raises

ValueError – Syntax error.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.parse(b':00000001FF\r\n')
>>> record.tag
<IhexTag.END_OF_FILE: 1>
>>> IhexFile.Record.parse(b'::00000001FF\r\n')
Traceback (most recent call last):
    ...
ValueError: syntax error
```

print(**args*, *stream=None*, *color=False*, ***kwargs*)

Prints a record.

The record is converted into tokens (eventually colorized) then joined and written onto a byte stream (*stdout* by default).

Parameters

- **args** – Forwarded to the underlying call to *to_tokens()*.
- **stream** (`io.BytesIO`) – The byte stream where the record tokens are printed. If `None`, *stdout* is selected.
- **color** (*bool*) – Tokens are colorized before printing.
- **kwargs** – Forwarded to the underlying call to *to_tokens()*.

Returns
BaseRecord – *self*.

See also:

to_tokens() colorize_tokens() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> _ = record.print()
:0312340061626391
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.print(stream=stream, color=True)
>>> stream.getvalue()
b'\x1b[0m\x1b[33m:\x1b[34m03\x1b[31m1234\x1b[32m00\x1b[36m61\x1b[96m62\
↪x1b[36m63\x1b[35m91\x1b[0m\r\n\x1b[0m'
```

serialize(*stream*, **args*, ***kwargs*)

Serializes onto a stream.

This wraps a call to *to_bytestr()* and stream.write.

Parameters

- **stream** (io.BytesIO) – Stream to write.
- **args** – Forwarded to *to_bytestr()*.
- **kwargs** – Forwarded to *to_bytestr()*.

Returns
BaseRecord – *self*.

See also:

to_bytestr() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.serialize(stream, end=b'\n')
>>> stream.getvalue()
b':0312340061626391\n'
```

to_bytestr(*end=b'\\r\\n'*)

Converts into a byte string.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Byte string representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_bytestr(end=b'\n')
b':0312340061626391\n'
```

to_tokens(*end=b'\\r\\n'*)

Converts into byte string tokens.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Mapping of token keys to token byte strings.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_tokens(end=b'\n')
{'before': b'', 'begin': b':', 'count': b'03', 'address': b'1234',
 'tag': b'00', 'data': b'616263', 'checksum': b'91', 'after': b'',
 'end': b'\n'}
```

update_checksum()

Updates the checksum field.

It updates the `checksum` attribute, assigning to it the value returned by *compute_checksum()*.

Returns

`BaseRecord` – *self*.

See also:

`checksum` *compute_checksum()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> record = IhexRecord(IhexRecord.Tag.END_OF_FILE, checksum=None)
>>> record.compute_checksum()
255
>>> record.checksum is None
True
>>> _ = record.update_checksum()
>>> record.checksum
255
```

update_count()

Updates the count field.

It updates the `count` attribute, assigning to it the value returned by *compute_count()*.

Returns

`BaseRecord` – *self*.

See also:

`count` *compute_count()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> Record = IhexFile.Record
>>> Tag = Record.Tag
>>> record = Record(Tag.DATA, data=b'abc', count=None, checksum=None)
>>> record.compute_count()
3
>>> record.count is None
True
>>> _ = record.update_count()
>>> record.count
3
```

validate(*checksum=True*, *count=True*)

Validates consistency of attribute values.

All the record attributes are checked for consistency.

Please refer to the implementation for more details.

Parameters

- **checksum** (*bool*) – Check the consistency of the `checksum` attribute.
- **count** (*bool*) – Check the consistency of the `count` attribute.

Returns
BaseRecord – *self*.

Raises
ValueError – Some targeted attributes are inconsistent.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> _ = record.validate()
>>> record.data = b'abc'
>>> _ = record.update_count().update_checksum().validate()
Traceback (most recent call last):
    ...
ValueError: unexpcted data
```

4.9.4 SrecTag

class hexrec.formats.srec.**SrecTag**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Motorola S-record tag.

Attributes

HEADER	Header string.
DATA_16	16-bit address data record.
DATA_24	24-bit address data record.
DATA_32	32-bit address data record.
RESERVED	Reserved tag.
COUNT_16	16-bit record count.
COUNT_24	24-bit record count.
START_32	32-bit start address.
START_24	24-bit start address.
START_16	16-bit start address.
denominator	the denominator of a rational number in lowest terms
imag	the imaginary part of a complex number
numerator	the numerator of a rational number in lowest terms
real	the real part of a complex number

Methods

fit_count_tag	Fits count record tag.
fit_data_tag	Fits data record tag.
fit_start_tag	Fits data record tag.
get_address_max	Calculates the maximum address.
get_address_size	Calculates the maximum address size.
get_data_max	Calculates the maximum data size.
get_tag_match	Calculates the matching tag.
is_count	Tells whether this is a record count tag.
is_header	Tells whether this is a header record tag.
is_start	Tells whether this is a start address record tag.
__init__	
as_integer_ratio	Return integer ratio.
bit_count	Number of ones in the binary representation of the absolute value of self.
bit_length	Number of bits necessary to represent self in binary.
conjugate	Returns self, the complex conjugate of any int.
from_bytes	Return the integer represented by the given array of bytes.
to_bytes	Return an array of bytes representing an integer.

COUNT_16 = 5

16-bit record count. Optional.

COUNT_24 = 6

24-bit record count. Optional.

DATA_16 = 1

16-bit address data record.

DATA_24 = 2

24-bit address data record.

DATA_32 = 3

32-bit address data record.

HEADER = 0

Header string. Optional.

RESERVED = 4

Reserved tag.

START_16 = 9

16-bit start address. Terminates *DATA_16*.

START_24 = 8

24-bit start address. Terminates *DATA_24*.

START_32 = 7

32-bit start address. Terminates *DATA_32*.

_DATA: Optional[*BaseTag*] = 1

Alias to a common data record tag.

This tag is used internally to build a generic data record.

__abs__()

abs(self)

__add__(*value*, /)

Return self+value.

__and__(*value*, /)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

classmethod __contains__(*member*)

Return True if member is a member of this enum raises TypeError if member is not an enum member

note: in 3.12 TypeError will no longer be raised, and True will also be returned if member is the value of a member in this enum

__dir__()

Returns all members and all public methods

__divmod__(*value*, /)

Return divmod(self, value).

__eq__(*value*, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(*value*, /)

Return self//value.

__format__(*format_spec*, /)

Default object formatter.

__ge__(*value*, /)

Return self>=value.

__getattribute__(*name*, /)

Return getattr(self, name).

classmethod __getitem__(*name*)

Return the member matching *name*.

__gt__(*value*, /)

Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__init__(**args*, ***kwds*)

__int__()
int(self)

__invert__()
~self

classmethod __iter__()
Return members in definition order.

__le__(*value*, /)
Return self<=value.

classmethod __len__()
Return the number of members (no aliases)

__lshift__(*value*, /)
Return self<<value.

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

__mul__(*value*, /)
Return self*value.

__ne__(*value*, /)
Return self!=value.

__neg__()
-self

__new__(*value*)

__or__(*value*, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod=None*, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__reduce_ex__(*proto*)
Helper for pickle.

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod=None*, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(*value*, /)
Return self-value.

__truediv__(*value*, /)
Return self/value.

__trunc__()

Truncating an Integral returns itself.

__xor__(*value*, */*)

Return self^value.

_generate_next_value_(*start*, *count*, *last_values*)

Generate the next value when not given.

name: the name of the member start: the initial start value or None count: the number of existing members last_values: the list of values assigned

_member_type_

alias of `int`

_new_member_(***kwargs*)

Create and return a new object. See help(type) for accurate signature.

_value_repr_()

Return repr(self).

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

classmethod fit_count_tag(*count*)

Fits count record tag.

Given the record sequence count, it fits the most compact *count* tag.

Parameters
count (*int*) – Record sequence *count*.

Returns
SrecTag – *Count* record tag.

Raises
ValueError – invalid *count*.

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> SrecTag.fit_count_tag(0xFFFF)
<SrecTag.COUNT_16: 5>
>>> SrecTag.fit_count_tag(0xFFFFFF)
<SrecTag.COUNT_24: 6>
>>> SrecTag.fit_count_tag(0x1000000)
Traceback (most recent call last):
    ...
ValueError: count overflow
```

classmethod fit_data_tag(*address_max*)

Fits data record tag.

Given the maximum *address* of the involved *data* records, it fits the most compact *data* tag.

Parameters
address_max (*int*) – Maximum *address* of the involved *data* records.

Returns
SrecTag – *Data* record tag.

Raises
ValueError – invalid *address_max*.

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> SrecTag.fit_data_tag(0xFFFF)
<SrecTag.DATA_16: 1>
>>> SrecTag.fit_data_tag(0xFFFFFF)
<SrecTag.DATA_24: 2>
>>> SrecTag.fit_data_tag(0xFFFFFFFF)
<SrecTag.DATA_32: 3>
>>> SrecTag.fit_data_tag(0x100000000)
Traceback (most recent call last):
    ...
ValueError: address overflow
```

classmethod fit_start_tag(*address*)

Fits data record tag.

Given the *start address*, it fits the most compact *start address* tag.

Parameters
address (*int*) – Start address.

Returns
SrecTag – *Start address* record tag.

Raises
ValueError – invalid *address*.

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> SrecTag.fit_start_tag(0xFFFF)
<SrecTag.START_16: 9>
>>> SrecTag.fit_start_tag(0xFFFFFF)
<SrecTag.START_24: 8>
>>> SrecTag.fit_start_tag(0xFFFFFFFF)
<SrecTag.START_32: 7>
>>> SrecTag.fit_start_tag(0x100000000)
Traceback (most recent call last):
    ...
ValueError: address overflow
```

from_bytes(*byteorder='big'*, *, *signed=False*)

Return the integer represented by the given array of bytes.

bytes
Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

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. Default is to use 'big'.

signed
Indicates whether two's complement is used to represent the integer.

get_address_max()

Calculates the maximum address.

It calculates the maximum *address* field for the calling tag. If the *address* field is not supported, it returns `None`.

Returns
int – Maximum *address* value, or `None`.

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> hex(SrecTag.DATA_32.get_address_max())
'0xffffffff'
>>> hex(SrecTag.START_32.get_address_max())
'0xffffffff'
>>> hex(SrecTag.COUNT_24.get_address_max())
'0xffffff'
>>> SrecTag.RESERVED.get_address_max()
0
```

get_address_size()

Calculates the maximum address size.

It calculates the maximum *address* field size for the calling tag. If the *address* field is not supported, it returns zero.

Returns

int – Maximum *address* size, or None.

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> SrecTag.DATA_32.get_address_size()
4
>>> SrecTag.START_32.get_address_size()
4
>>> SrecTag.COUNT_24.get_address_size()
3
>>> SrecTag.RESERVED.get_address_size()
0
```

get_data_max()

Calculates the maximum data size.

It calculates the maximum *data* field size for the calling tag. If the *data* field is not supported, it returns None.

Returns

int – Maximum *data* size, or None.

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> SrecTag.DATA_16.get_data_max()
252
>>> SrecTag.DATA_32.get_data_max()
250
>>> SrecTag.START_32.get_data_max()
0
```

get_tag_match()

Calculates the matching tag.

Given *data* or *start address* records, it returns the matching tag.

Returns

SrecTag – Matching tag for *self*, or None

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> SrecTag.DATA_16.get_tag_match()
<SrecTag.START_16: 9>
>>> SrecTag.START_32.get_tag_match()
<SrecTag.DATA_32: 3>
>>> SrecTag.HEADER.get_tag_match() is None
True
```

imag

the imaginary part of a complex number

is_count()

Tells whether this is a record count tag.

This method returns true if this record tag is used for *record count* records.

Returns

bool – This is a record count tag.

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> SrecTag.COUNT_16.is_count()
True
>>> SrecTag.COUNT_24.is_count()
True
>>> SrecTag.DATA_16.is_count()
False
```

`is_header`()

Tells whether this is a header record tag.

This method returns true if this record tag is used for *header* records.

Returns
bool – This is a header record tag.

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> SrecTag.HEADER.is_header()
True
>>> SrecTag.DATA_16.is_header()
False
```

`is_start`()

Tells whether this is a start address record tag.

This method returns true if this record tag is used for *start address* records.

Returns
bool – This is a start address record tag.

Examples

```
>>> from hexrec import SrecFile
>>> SrecTag = SrecFile.Record.Tag
>>> SrecTag.START_16.is_start()
True
>>> SrecTag.START_32.is_start()
True
>>> SrecTag.DATA_16.is_start()
False
```

`numerator`

the numerator of a rational number in lowest terms

`real`

the real part of a complex number

`to_bytes`(*length=1*, *byteorder='big'*, ***, *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. Default is length 1.

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. Default is to use 'big'.

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.

4.10 titxt

Texas Instruments TI-TXT format.

See also:

https://downloads.ti.com/docs/esd/SPNU118/ti-txt-hex-format-ti-txt-option-stdz0795656.html

Classes

TiTxtFile	Texas Instruments TI-TXT file object.
TiTxtRecord	Texas Instruments TI-TXT record object.
TiTxtTag	Texas Instruments TI-TXT tag.

4.10.1 TiTxtFile

class hexrec.formats.titxt.**TiTxtFile**

Texas Instruments TI-TXT file object.

Attributes

DEFAULT_DATALEN	Default data attribute length.
FILE_EXT	Supported filename extensions.
META_KEYS	Meta information key names.
maxdatalen	Maximum byte size of the data field.
memory	Memory object stored by records role.
records	Records stored by records role.

Methods

__init__	
align	Pads blocks to align their boundaries.
append	Appends a byte.
apply_records	Applies records to memory and meta.
clear	Clears data within a range.
convert	Converts a file object to another format.
copy	Copies within a range.
crop	Clears data outside a range.
cut	Cuts data within a range.
delete	Deletes data within a range.

continues on next page

Table 8 – continued from previous page

discard_memory	Discards underlying memory.
discard_records	Discards underlying records.
extend	Concatenates data.
fill	Fills a range.
find	Finds a substring.
flood	Floods a range.
from_blocks	Creates a file object from a memory object.
from_bytes	Creates a file object from a byte string.
from_memory	Creates a file object from a memory object.
from_records	Creates a file object from records.
get_address_max	Maximum address within memory.
get_address_min	Minimum address within memory.
get_holes	List of memory holes.
get_meta	Meta information.
get_spans	List of memory block spans.
index	Finds a substring.
load	Loads a file object from the filesystem.
merge	Merges data onto the file.
parse	Parses records from a byte stream.
print	Prints record content to stdout.
read	Extracts a substring.
save	Saves a file object into the filesystem.
serialize	Serializes records onto a byte stream.
set_meta	Sets meta information.
shift	Shifts data addresses by an offset.
split	Splits into parts.
update_records	Applies memory and meta to records.
validate_records	Validates records.
view	Memory view.
write	Writes data into the file.

DEFAULT_DATALEN: int = 16

Default data attribute length.

Default value for the *maxdatalen* *meta*, which sets the maximum size of `BaseRecord.data` field values.

FILE_EXT: Sequence[str] = ['.txt']

Supported filename extensions.

Sequence of file name extension substrings (e.g. `.hex`). This list is used by functions like `guess_format_name()` to manage mapping of file *formats*.

META_KEYS: Sequence[str] = ['maxdatalen']

Meta information key names.

Sequence of key strings listing the supported *meta* information of this file *format*.

Record

alias of *TiTxtRecord*

__add__(*other*)

Concatenates with another file.

Equivalent to *copy()* then *extend()*.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – Concatenation of *self* and *other*.

See also:

copy() extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 + file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file4 = file3 + b'789'
>>> file4.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__bool__()

bool: Has data records or memory.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> bool(file)
False
>>> _ = file.append(0)
>>> bool(file)
True
```

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> file = IhexFile.from_records([IhexRecord.create_end_of_file()])
>>> bool(file)
False
>>> file.records.insert(0, IhexRecord.create_data(0, b'\0'))
>>> bool(file)
True
```

__delitem__(*key*)

Deletes a range.

Parameters
key (*slice or int*) – Range to delete.

See also:

`bytesparse.base.MutableMemory.__delitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> del file[457]
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xz']]
>>> del file[125:457]
>>> file.memory.to_blocks()
[[123, b'abz']]
```

__eq__(*other*)

Equality test.

The file objects *self* and *other* are considered *equal* if the inequality tests of *__ne__()* result false.

Returns
bool – *self* and *other* are *equal*.

See Also
__ne__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 == file2
True
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 == file3
False
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 == file4
False
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
```

(continues on next page)

(continued from previous page)

```
>>> srec_file == ihex_file
False
>>> srec_file.memory == ihex_file.memory
True
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__getitem__(*key*)

Extracts a range.

Parameters
key (*slice or int*) – Range to extract.

Raises
ValueError – invalid range.

See also:

bytesparse.base.ImmutableMemory.__getitem__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> chr(file[457])
'y'
>>> repr(file[333])
'None'
>>> file[123:125]
b'ab'
>>> file[125:457]
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

__hash__ = None

__iadd__(*other*)

Concatenates data.

Equivalent to *extend()*.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – *self*.

See also:

memory *extend()* *discard_records()* bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 += file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file1 += b'789'
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__init__()

__ior__(*other*)

Merges with another file.

Equivalent to *merge()*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self*.

See also:

merge() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 |= file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file1 |= b'789'
>>> file1.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__ne__(*other*)

Inequality test.

The file objects *self* and *other* are considered *unequal* if any of the following tests result true:

- Both have *memory role* (i.e. *memory*), resulting unequal;
- Both have *records role* (i.e. *records*), resulting unequal;

- *other* does not have a *meta* listed by *META_KEYS*;
- A *meta* value (among those of *META_KEYS*) is different.

Returns
bool – *self* and *other* are *unequal*.

See also:

__eq__() memory records META_KEYS

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 != file2
False
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 != file3
True
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 != file4
True
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file != ihex_file
True
>>> srec_file.memory != ihex_file.memory
False
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__or__(*other*)

Merges with another file.

Equivalent to *copy()* then *merge()*.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self* merged with *other*.

See also:

copy() merge()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 | file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file4 = file3 | b'789'
>>> file4.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__setitem__(*key*, *value*)

Sets a range.

Parameters

- **key** (*slice or int*) – Range to set.
- **value** (bytes, `bytesparse.base.ImmutableMemory`, `None`) – Value(s) to set. `None` acts like *clear()*.

Raises

ValueError – invalid range.

See also:

`bytesparse.base.MutableMemory.__setitem__()` *clear()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file[124] = b'?'
>>> file.memory.to_blocks()
[[123, b'a?c'], [456, b'xyz']]
>>> file[:125] = None
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'xyz']]
>>> file[457:458] = b'789'
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'x789z']]
```

__weakref__

list of weak references to the object (if defined)

classmethod _is_line_empty(*line*)

Empty line check.

Tells whether a *line* has no meaningful content (e.g. all whitespace). The check itself depends on the implementing file *format*. It may be used internally to skip empty lines, e.g. by *parse()*.

Parameters
line (*bytes*) – A line, byte string.

Returns
bool: The *line* is empty.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexFile._is_line_empty(b'')
True
>>> IhexFile._is_line_empty(b' \t\v\r\n')
True
>>> IhexFile._is_line_empty(b':00000001FF\r\n')
False
```

align(*modulo*, *start=None*, *endex=None*, *pattern=0*)

Pads blocks to align their boundaries.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*, so that memory blocks are aligned to the required *modulo*.

Any stored *records* are discarded upon return.

Parameters

- **modulo** (*int*) – Alignment modulo.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns
`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.align()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [134, b'xyz']])
>>> _ = file.align(4, pattern=b'.')
>>> file.memory.to_blocks()
[[120, b'...abc..'], [132, b'..xyz...']]
```

append(*item*)

Appends a byte.

It appends the *item* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
item (*byte or int*) – Byte to append.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.append()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123)
>>> _ = file.append(b'.')
>>> _ = file.append(0)
>>> file.memory.to_blocks()
[[123, b'abc.\x00']]
```

apply_records()

Applies records to memory and meta.

This method processes the stored *records*, converting *data* as *memory*, and special records into their *meta* counterparts.

This effectively converts the *records role* into the *memory role* (keeping both).

The *memory* and *meta* are assigned upon return. Any exceptions being raised should not alter the file object.

Returns
BaseFile – *self*.

Raises
ValueError – *records* attribute not populated.

See also:

records memory get_meta() update_records()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_start_linear_address(456),
```

(continues on next page)

(continued from previous page)

```
...             IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records, maxdatalen=16)
>>> _ = file.apply_records()
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
```

clear(*start=None*, *endex=None*)

Clears data within a range.

It clears the specified range of underlying *memory* object, making a memory hole.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.clear()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.clear(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

classmethod convert(*source*, *meta=True*)

Converts a file object to another format.

It copies the *memory* and *meta* of the *source* file object, creating a new one of the target `BaseFile` format type.

Parameters

- **source** (`BaseFile`) – Source file object to convert.
- **meta** (*bool*) – Copy *meta* information to the target file object. Only the keys of the target *META_KEYS* are processed.

Returns

`BaseFile` – Converted copy of *source* to the target format.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> source = IhexFile.from_blocks(blocks, startaddr=789)
>>> target = SrecFile.convert(source)
>>> target.memory is source.memory
False
>>> target.memory == source.memory
True
>>> target.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 789}
```

copy(*start=None*, *endex=None*, *meta=True*)

Copies within a range.

It copied data within the specified range of the file object, creating a new one carrying the inner slice.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

`BaseFile` – *self*.

See also:

memory *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.copy(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

crop(*start=None*, *endex=None*)

Clears data outside a range.

It clears outside the specified range of underlying *memory* object, timming it.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.crop()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.crop(start=124, endex=132)
>>> file.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
```

cut(*start=None*, *endex=None*, *meta=False*)

Cuts data within a range.

It takes data within the specified range away from the file object, creating a new one carrying the inner slice. The inner slice is cleared from *self*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns
`BaseFile` – *self*.

See also:

memory *clear()* *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.cut(start=124, endex=132)
>>> inner.memory.to_blocks()
```

(continues on next page)

(continued from previous page)

```
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

delete(*start=None*, *endex=None*)

Deletes data within a range.

It deletes the specified range of underlying *memory* object, shifting all subsequent data towards the collapsed range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.delete()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.delete(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'az']]
```

discard_memory()

Discards underlying memory.

The underlying *memory* object is assigned `None`.

If the underlying *records* object is `None`, it is assigned a new empty memory object.

Returns

`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_bytes(b'abc', offset=123)
>>> _ = file.update_records()
>>> _ = file.discard_memory()
>>> _ = file.update_records()
Traceback (most recent call last):
    ...
ValueError: memory instance required
```

discard_records()

Discards underlying records.

The underlying *records* object is assigned `None`.

If the underlying *memory* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> _ = file.validate_records()
>>> _ = file.discard_records()
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: records required
```

extend(*other*)

Concatenates data.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.extend()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> _ = file1.extend(file2)
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> _ = file1.extend(b'789')
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

fill(*start=None*, *endex=None*, *pattern=0*)

Fills a range.

It writes a *pattern* of bytes onto the underlying *memory* object, overwriting anything within the specified range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.fill()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.fill(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'a.......z']]
```

find(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it returns `-1`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address; `-1` if not found.

See also:

index `bytesparse.base.ImmutableMemory.find()`

Notes

The internal *memory* might allow negative addresses for its stored data. In that case, *index()* would be more appropriate, because it raises an exception when the *item* is not found.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.find(b'yz')
457
>>> file.find(ord('b'))
124
>>> file.find(b'?')
-1
```

flood(*start=None*, *endex=None*, *pattern=0*)

Floods a range.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.flood()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.get_holes()
[(126, 130)]
>>> _ = file.flood(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'abc....xyz']]
```

classmethod from_blocks(*blocks*, ***meta*)

Creates a file object from a memory object.

The *blocks* are put into the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **blocks** (*list of blocks*) – Memory blocks to put into *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

`BaseFile` – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* `bytesparse.base.ImmutableMemory.from_blocks()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_bytes(*data*, *offset=0*, ***meta*)

Creates a file object from a byte string.

The byte string makes a single *data* block, placed at some offset within the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **data** (*bytes*) – A byte string used to make a single data block.
- **offset** (*int*) – Offset of the single data block within *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_bytes()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
8
```

classmethod **from_memory**(*memory=None*, ***meta*)

Creates a file object from a memory object.

The *memory* is set as the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **memory** (bytesparse.base.MutableMemory) – Memory object to set as *memory*. If None, an empty memory object is automatically created.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS bytesparse.base.MutableMemory

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from bytesparse import Memory
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> memory = Memory.from_blocks(blocks)
>>> from hexrec import SrecFile
>>> file = SrecFile.from_memory(memory, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_records(*records*, *maxdatalen=None*)

Creates a file object from records.

The *records* sequence is set as the `record` attribute of the created file object.

This method creates a file object in *records role*. This means that only its *records* is internally instanced, while the *memory* requires manual or lazy instancing (i.e. either via direct call to *apply_records()*, or any other methods indirectly calling it).

Parameters

- **records** (list of `BaseRecord`) – Record sequence to set as *records*.
- **maxdatalen** (`Optional[int]`) – Maximum record *data* field size. If `None`, the maximum non-zero size of the *data* field from the *records* sequence is used. If all the *records* have zero sized *data* field, the class attribute *DEFAULT_DATALEN* is used.

Returns

`BaseFile` – The created file object.

Raises

ValueError – invalid *meta* values.

See also:

`BaseRecord`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
3
```

get_address_max()

Maximum address within memory.

It returns the maximum address of the underlying *memory* object.

Returns
int – Maximum address.

See also:

`bytesparse.base.ImmutableMemory.endin`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_max()
458
```

get_address_min()

Minimum address within memory.

It returns the minimum address of the underlying *memory* object.

Returns
int – Minimum address.

See also:

`bytesparse.base.ImmutableMemory.start`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_min()
123
```

get_holes()

List of memory holes.

It scans the underlying *memory* and returns the list of memory holes/gaps.

Each hole is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory hole boundaries.

See also:

`bytesparse.base.ImmutableMemory.gaps()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_holes()
[(126, 456), (459, 789)]
```

`get_meta()`

Meta information.

It builds and returns a dictionary of *meta* information. Meta keys are taken from the *META_KEYS* class attribute.

Returns
dict – Meta information dictionary.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks, header=b'HDR\0')
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 0}
```

`get_spans()`

List of memory block spans.

It scans the underlying *memory* and returns the list of memory block spans/intervals.

Each span is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory block boundaries.

See also:

`bytesparse.base.ImmutableMemory.intervals()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_spans()
[(123, 126), (456, 459), (789, 791)]
```

index(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it raises `ValueError`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address.

Raises

ValueError – *item* not found.

See also:

find `bytesparse.base.ImmutableMemory.index()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.index(b'yz')
457
>>> file.index(ord('b'))
124
>>> file.index(b'?')
Traceback (most recent call last):
    ...
ValueError: subsection not found
```

classmethod load(*path*, **args*, ***kwargs*)

Loads a file object from the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *parse()* to load a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdin.buffer` is used.
- **args** – Forwarded to *parse()*.
- **kwargs** – Forwarded to *parse()*.

Returns

`BaseFile` – Loaded file object.

See also:

save() *parse()* `open()` `sys.stdin.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.load('data.hex')
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

property maxdatalen: int

Maximum byte size of the data field.

This property sets the maximum byte size of the *data* field of a serialized record.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises

ValueError – Invalid maximum data length.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> file.maxdatalen
16
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> file.maxdatalen = 8
>>> _ = file.print()
S0030000FC
S10B0000000102030405060 7D8
S10B000808090A0B0C0D0E0F90
S10B001010111213141516 1748
S10B001818191A1B1C1D1E1F00
S10B002020212223242526 27B8
S10B002828292A2B2C2D2E2F70
S10B003030313233343536 3728
```

(continues on next page)

(continued from previous page)

```
S10B003838393A3B3C3D3E3FE0
S5030008F4
S9030000FC
>>> file.maxdatalen = 0
Traceback (most recent call last):
    ...
ValueError: invalid maximum data length
```

Type
int

property memory: MutableMemory

Memory object stored by records role.

This readonly property exposes the memory object stored by the file object while in *memory role*.

If this property is accessed while the file object is not in *memory role*, it automatically activates it by an implicit call to *apply_records()*, with default arguments.

For more control activating the *memory role*, please call *apply_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *records role* (i.e. altering content of *records*) would implicitly discard *memory* via *discard_memory()*.

See also:

apply_records() *discard_memory()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> _ = file.write(789, b'?!')
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz'], [789, b'?!']]
```

Type
`bytesparse.Memory`

merge(**files*, *clear=False*)

Merges data onto the file.

It writes the provided *files* onto *self*, in the provided order. Any common address ranges are overwritten.

Any stored *records* are discarded upon return.

Parameters

- **files** (BaseFile) – Files to merge.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

BaseFile – *self*.

See also:

clear() *discard_records()* *write()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = SrecFile.from_bytes(b'<<<?????>>>', offset=450)
>>> _ = file3.merge(file1, file2)
>>> file3.memory.to_blocks()
[[123, b'abc'], [450, b'<<<???xyz>>']]
```

classmethod parse(*stream*, *ignore_errors=False*, *ignore_after_termination=True*)

Parses records from a byte stream.

It executes BaseRecord.parse() for each line of the incoming *stream*, creating a new file object with the collected records calling *from_records()*.

Lines resulting empty by _is_empty_line() are just discarded.

Notes

Please refer to the actual implementation of each record file *format*, because it may be more specialized.

Parameters

- **stream** (*bytes IO or buffer*) – Stream or byte buffer to parse records from.
- **ignore_errors** (*bool*) – Ignore Exception raised by BaseRecord.parse().
- **ignore_after_termination** (*bool*) – Ignore anything after the termination record was parsed, if supported (e.g. *End Of File* or *start address* record, depending on the specific file *format*).

Returns

BaseFile – *self*.

See also:

parse() BaseRecord.parse() *from_records()* _is_empty_line()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> buffer = b'''
...     :03DA7A0061626383
...     :040000050000CAFE2F
...     :00000001FF
... '''
>>> import io
>>> stream = io.BytesIO(buffer)
>>> file = IhexFile.parse(stream)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
>>> file = IhexFile.parse(buffer)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

print(**args*, *stream=None*, *color=False*, *start=None*, *stop=None*, ***kwargs*)

Prints record content to stdout.

This helper method prints each record of *records* via `BaseRecord.print()`. As such, it also supports colored tokens and streams different from *stdout*.

It is possible to print subset of the records by specifying the record index range.

> **Warning:** This method is **NOT** equivalent to *serialize()*, because it just prints each record from *records*. Please use *serialize()* for an actual serialization of the whole file.

Parameters

- **args** – Forwarded to the underlying call to `to_tokens()`.
- **stream** (*byte stream*) – Stream to print onto. If `None`, *stdout* is used.
- **color** (*bool*) – Colorize record tokens with ANSI color codes.
- **start** (*int*) – Inclusive start record index of the specified range. If `None`, start from the first record.
- **stop** (*int*) – Exclusive end record index of the specified range. If negative, look back from the last index. If `None`, print up to the last record.
- **kwargs** – Forwarded to the underlying call to `to_tokens()`.

Returns

`BaseFile` – *self*.

See also:

`BaseRecord.print()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212222324252627282 92A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> _ = file.print(color=True, start=1, stop=-2)
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212222324252627282 92A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
```

read(*start=None*, *endex=None*, *fill=0*)

Extracts a substring.

It extracts a byte string from the specified range, filling any memory holes/gaps (without altering *memory*).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **fill** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory `bytesparse.base.MutableMemory.extract()` `bytesparse.base.MutableMemory.to_bytes()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.read(start=124, endex=132)
b'bc\x00\x00\x00\x00xy'
>>> file.read(start=124, endex=132, fill=b'.')
b'bc....xy'
```

(continues on next page)

(continued from previous page)

```
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

property records: MutableSequence[*BaseRecord*]

Records stored by records role.

This readonly property exposes the list of records stored by the file object while in *records role*.

If this property is accessed while the file object is not in *records role*, it automatically activates it by an implicit call to *update_records()*, with default arguments.

For more control activating the *records role*, please call *update_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *memory role* (i.e. altering content of *memory*) would implicitly discard *records* via *discard_records()*.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, startaddr=789)
>>> len(file.records)
5
>>> _ = file.print()
S0030000FC
S106007B61626358
S10601C878797AC5
S5030002FA
S9030315E4
>>> _ = file.update_records(data_tag=SrecFile.Record.Tag.DATA_32)
>>> _ = file.print()
S0030000FC
S3080000007B61626356
S308000001C878797AC3
S5030002FA
S70500000315E2
```

Type
list of `BaseRecord`

save(*path*, **args*, ***kwargs*)

Saves a file object into the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *serialize()* to save a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdout.buffer` is used.
- **args** – Forwarded to *serialize()*.
- **kwargs** – Forwarded to *serialize()*.

Returns

`BaseFile` – *self*.

See also:

load() *serialize()* `open()` `sys.stdout.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> _ = file.save('data.hex')
```

serialize(*stream*, **args*, ***kwargs*)

Serializes records onto a byte stream.

It executes `BaseRecord.serialize()` for each of the stored *records*.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **args** – Forwarded to `BaseRecord.serialize()` of each record.
- **kwargs** – Forwarded to `BaseRecord.serialize()` of each record.

Returns

`BaseFile` – *self*.

See also:

parse() `BaseRecord.serialize()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> import sys
>>> _ = file.serialize(sys.stdout.buffer, end=b'\n')
:03DA7A0061626383
:040000050000CAFE2F
:00000001FF
```

set_meta(*meta*, *strict=True*)

Sets meta information.

It sets the provided *kwargs* to their matching *meta* attributes, as listed by *META_KEYS*.

Parameters

- **meta** (*dict*) – Mapping of the *meta* information to set.
- **strict** (*bool*) – All the keys within *meta* must exist within *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *get_meta()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 0}
>>> _ = file.set_meta(dict(header=b'HDR\0', startaddr=456))
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 456}
```

shift(*offset*)

Shifts data addresses by an offset.

It shifts addresses of the underlying *memory* object data blocks by the provided *offset* amount.

Any stored *records* are discarded upon return.

Parameters

offset (*int*) – Offset to apply to the underlying data block addresses.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.shift()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> _ = file.shift(1000)
>>> file.memory.to_blocks()
[[1123, b'abc'], [1456, b'xyz']]
```

split(**addresses*, *meta=True*)

Splits into parts.

The provided *addresses* are sorted and used as markers to split *self* into parts.

Each part is the *copy()* of *self* within the range of that part, in *memory role* (i.e., *records* is not populated).

Parameters

- **addresses** (*int*) – Split points.
- **meta** (*bool*) – Each part inherits *meta* from *self*.

Returns

list of `BaseFile` – Parts after splitting.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'Hello, World!', offset=123)
>>> parts = file.split(128, 130)
>>> for part in parts: print(part.memory.to_blocks())
[[123, b'Hello']]
[[128, b', ']]
[[130, b'World!']]
>>> file.memory.to_blocks()
[[123, b'Hello, World!']]
```

update_records(*align=False*, *addrlen=4*)

Applies memory and meta to records.

This method processes the stored *memory* and *meta* information to generate the sequence of *records*.

This effectively converts the *memory role* into the *records role* (keeping both).

The *records* is assigned upon return. Any exceptions being raised should not alter the file object.

Parameters

- **align** (*bool*) – Aligns data record chunk address bounds to *maxdatalen*.
- **addrlen** (*int*) – Address length, in *nibbles* (4-bit units).

Returns

TiTxtFile – *self*.

Raises
ValueError – *memory* attribute not populated.

See also:

records *memory* *get_meta()* *apply_records()*

Examples

```
>>> from hexrec import TiTxtFile
>>> blocks = [[456, b'abc']]
>>> file = TiTxtFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[456, b'abc']]
>>> file.get_meta()
{'maxdatalen': 8}
>>> _ = file.update_records()
>>> len(file.records)
3
>>> _ = file.print()
@01C8
61 62 63
q
```

validate_records(*data_ordering=False*, *address_even=True*)

Validates records.

It performs consistency checks for the underlying *records*.

Parameters

- **data_ordering** (*bool*) – Checks that the *data* record sequence has monotonically increasing addresses, without any overlapping.
- **address_even** (*bool*) – Addresses must be even.

Returns
TiTxtFile – *self*.

Raises
ValueError – Invalid record sequence.

Examples

```
>>> from hexrec import TiTxtFile
>>> records = [TiTxtFile.Record.create_data(456, b'abc')]
>>> file = TiTxtFile.from_records(records)
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: missing end of file record
```

view(*start=None*, *endex=None*)

Memory view.

It returns a `memoryview` over the specified range, which must cover a *contiguous* data region (i.e. no memory holes within).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

memoryview – View of the specified range.

Raises

ValueError – non-contiguous data within range.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> bytes(file.view(start=456, endex=458))
b'xy'
>>> bytes(file.view())
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

write(*address*, *data*, *clear=False*)

Writes data into the file.

It writes the provided *data* into the underlying *memory* object.

Any stored *records* are discarded upon return.

Parameters

- **address** (*int*) – Address where *data* has to be written.
- **data** (*bytes or memory*) – Byte data to write.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

`BaseFile` – *self*.

See also:

memory *clear()* *discard_records()* `bytesparse.base.MutableMemory.write()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> _ = file.write(123, b'abc')
>>> _ = file.write(555, ord('?'))
>>> _ = file.write(1000, SrecFile.from_bytes(b'xyz', offset=456))
>>> file.memory.to_blocks()
[[123, b'abc'], [555, b'?'], [1456, b'xyz']]
```

4.10.2 TiTxtRecord

class hexrec.formats.titxt.**TiTxtRecord**(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

Texas Instruments TI-TXT record object.

Attributes

EQUALITY_KEYS	Meta keys for equality checks.
LINE_REGEX	Line parser regex.
META_KEYS	Meta keys.

Methods

__init__	
compute_checksum	Computes the checksum field value.
compute_count	Compute the count field value.
copy	Shallow copy.
create_address	Creates an address record.
create_data	Creates a data record.
create_eof	Creates an End Of File record.
data_to_int	Interprets data bytes as integer.
get_meta	Gets meta information.
parse	Parses a record from bytes.
print	Prints a record.
serialize	Serializes onto a stream.
to_bytestr	Converts into a byte string.
to_tokens	Converts into byte string tokens.
update_checksum	Updates the checksum field.
update_count	Updates the count field.
validate	Validates consistency of attribute values.

EQUALITY_KEYS: Sequence[str] = ['address', 'checksum', 'count', 'data', 'tag']

Meta keys for equality checks.

Equality methods (*__eq__()* and *__ne__()*) check against these *meta* keys only. Any other *meta* keys are just ignored.

LINE_REGEX = re.compile(b'^\\s*((?P<data>([0-9A-Fa-f]{2}[\\t]?)+)|(@(?P<address>[0-9A-Fa-f]+))|(?P<eof>q))\\s*\\r?\\n?$')

Line parser regex.

META_KEYS: Sequence[str] = ['address', 'after', 'before', 'checksum', 'coords', 'count', 'data', 'tag']

Meta keys.

This sequence holds the *meta* keys for copying (see *copy()*).

Tag

alias of *TiTxtTag*

__bytes__()

Serializes the record into bytes.

Returns
bytes – Byte serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> bytes(record)
b':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> bytes(record)
b'abc'
```

__eq__(*other*)

Equality test.

This method returns true if *self* is considered equal to *other*.

As inequality is usually easier to check, this method is usually implemented as a trivial `not self != other` (*__ne__()*).

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* equals *other*.

See also:

__ne__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 == ihex2
True
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 == ihex3
False
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 == raw
False
```

__hash__ = None

__init__(*tag*, *address=0*, *data=b''*, *count=Ellipsis*, *checksum=Ellipsis*, *before=b''*, *after=b''*, *coords=(-1, -1)*, *validate=True*)

__ne__(*other*)

Ineuality test.

This method returns true if *self* is considered unequal to *other*.

Each attribute listed by *EQUALITY_KEYS* is compared between *self* and *other*. This method returns whether any attributes do not match.

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* and *other* are unequal.

See also:

__eq__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
```

(continues on next page)

(continued from previous page)

```
>>> ihex1 != ihex2
False
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 != ihex3
True
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 != raw
True
```

__repr__()

String representation.

It returns a string representation of the record content, for human understanding only.

Returns

str – String representation.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> repr(record)
"<<class 'hexrec.formats.ihex.IhexRecord'> @...
  address:=0 after:=b'' before:=b'' checksum:=255 coords:=(-1, -1)
  count:=0 data:=b'' tag:=<IhexTag.END_OF_FILE: 1>>"
```

__str__()

Serializes the record into a string.

Returns

str – String serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> str(record)
'abc'
```

__weakref__

list of weak references to the object (if defined)

compute_checksum()

Computes the checksum field value.

It computes and returns the format-specific `checksum` value of a record.

When not specialized, it returns `None` by default.

Returns

int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_checksum()
215
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_checksum())
'None'
```

compute_count()

Compute the count field value.

It computes and returns the format-specific `count` value of a record.

When not specialized, it returns `None` by default.

Returns

int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_count()
3
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_count())
'None'
```

copy(*validate=True*)

Shallow copy.

It calls the record constructor, passing *meta* to it.

Parameters
validate (*bool*) – Performs validation on instantiation (*__init__()*).

Returns
BaseRecord – Shallow copy.

See also:

__init__() get_meta()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record1 = IhexFile.Record.create_data(0x1234, b'abc')
>>> record2 = record1.copy()
>>> record1 is record2
False
>>> record1 == record2
True
```

classmethod create_address(*address*, *addrlen=4*)

Creates an address record.

Parameters

- **address** (*int*) – Address value.
- **addrlen** (*int*) – Address length, in *nibbles* (4-bit units).

Returns
TiTxtRecord – Address record object.

Raises
ValueError – invalid parameter.

Examples

```
>>> from hexrec import TiTxtFile
>>> record = TiTxtFile.Record.create_address(0x1234)
>>> str(record)
'@1234\r\n'
```

classmethod create_data(*address*, *data*)

Creates a data record.

Parameters

- **address** (*int*) – Ignored; please provide zero.
- **data** (*bytes*) – Record byte data.

Returns
TiTxtRecord – Data record object.

Raises
ValueError – invalid parameter.

Examples

```
>>> from hexrec import TiTxtFile
>>> record = TiTxtFile.Record.create_data(0, b'abc')
>>> str(record)
'61 62 63\r\n'
```

classmethod create_eof()

Creates an End Of File record.

Returns
TiTxtRecord – End Of File record object.

Raises
ValueError – invalid parameter.

Examples

```
>>> from hexrec import TiTxtFile
>>> record = TiTxtFile.Record.create_eof()
>>> str(record)
'q\r\n'
```

data_to_int(*byteorder='big'*, *signed=False*)

Interprets data bytes as integer.

It creates an integer from bytes of the `data` field.

Parameters

- **byteorder** (*'big' or 'little'*) – Byte order (endianness): either `'big'` (default) or `'little'`.
- **signed** (*bool*) – Signed integer (2-complement); default false.

Returns
int – Interpreted integer value.

See also:

`int.from_bytes()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0xABCD)
>>> record.data
b'\xab\xcd'
>>> addrext = record.data_to_int()
>>> addrext, hex(addrext)
(43981, '0xabcd')
```

get_meta()

Gets meta information.

It returns all the object attributes whose keys are listed by *META_KEYS*.

Returns
dict – Attribute values listed by *META_KEYS*.

See also:

META_KEYS `set_meta()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> record.get_meta()
{'address': 0, 'after': b'', 'before': b'', 'checksum': 255,
 'coords': (-1, -1), 'count': 0, 'data': b'',
 'tag': <IhexTag.END_OF_FILE: 1>}
```

classmethod parse(*line*, *address=0*, *validate=True*)

Parses a record from bytes.

Parameters

- **line** (*bytes*) – String of bytes to parse.
- **address** (*int*) – Default record address for *data* records.
- **validate** (*bool*) – Perform validation checks.

Returns
`BaseRecord` – Parsed record.

Raises
ValueError – Syntax error.

Examples

```
>>> from hexrec import TiTxtFile
>>> record = TiTxtFile.Record.parse(b'@ABCD\r\n')
>>> record.tag
<TiTxtTag.ADDRESS: 1>
>>> record = TiTxtFile.Record.parse(b'61 62 63\r\n', address=123)
>>> record.address, record.data
(123, b'abc')
>>> TiTxtFile.Record.parse(b':ABCD\r\n')
Traceback (most recent call last):
    ...
ValueError: syntax error
```

print(**args*, *stream=None*, *color=False*, ***kwargs*)

Prints a record.

The record is converted into tokens (eventually colorized) then joined and written onto a byte stream (*stdout* by default).

Parameters

- **args** – Forwarded to the underlying call to *to_tokens()*.
- **stream** (io.BytesIO) – The byte stream where the record tokens are printed. If None, *stdout* is selected.
- **color** (*bool*) – Tokens are colorized before printing.
- **kwargs** – Forwarded to the underlying call to *to_tokens()*.

Returns

BaseRecord – *self*.

See also:

to_tokens() colorize_tokens() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> _ = record.print()
:0312340061626391
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.print(stream=stream, color=True)
>>> stream.getvalue()
b'\x1b[0m\x1b[33m:\x1b[34m03\x1b[31m1234\x1b[32m00\x1b[36m61\x1b[96m62\
↪x1b[36m63\x1b[35m91\x1b[0m\r\n\x1b[0m'
```

serialize(*stream*, **args*, ***kwargs*)

Serializes onto a stream.

This wraps a call to *to_bytestr()* and stream.write.

Parameters

- **stream** (io.BytesIO) – Stream to write.
- **args** – Forwarded to *to_bytestr()*.
- **kwargs** – Forwarded to *to_bytestr()*.

Returns

BaseRecord – *self*.

See also:

to_bytestr() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.serialize(stream, end=b'\n')
>>> stream.getvalue()
b':0312340061626391\n'
```

to_bytestr(*end=b'\\r\\n'*)

Converts into a byte string.

Parameters

end (*bytes*) – End of record termination bytes.

Returns

bytes – Byte string representation.

Examples

```
>>> from hexrec import TiTxtFile
>>> record = TiTxtFile.Record.create_data(0, b'abc')
>>> record.to_bytestr(end=b'\n')
b'61 62 63\n'
```

to_tokens(*end=b'\\r\\n'*)

Converts into byte string tokens.

Parameters

end (*bytes*) – End of record termination bytes.

Returns

bytes – Mapping of token keys to token byte strings.

Examples

```
>>> from hexrec import TiTxtFile
>>> record = TiTxtFile.Record.create_data(0, b'abc')
>>> record.to_tokens(end=b'\n')
{'before': b'', 'begin': b'', 'address': b'', 'data': b'61 62 63',
 'after': b'', 'end': b'\n'}
```

update_checksum()

Updates the checksum field.

It updates the checksum attribute, assigning to it the value returned by *compute_checksum()*.

Returns
BaseRecord – *self*.

See also:

checksum *compute_checksum()*

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> record = IhexRecord(IhexRecord.Tag.END_OF_FILE, checksum=None)
>>> record.compute_checksum()
255
>>> record.checksum is None
True
>>> _ = record.update_checksum()
>>> record.checksum
255
```

update_count()

Updates the count field.

It updates the count attribute, assigning to it the value returned by *compute_count()*.

Returns
BaseRecord – *self*.

See also:

count *compute_count()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> Record = IhexFile.Record
>>> Tag = Record.Tag
>>> record = Record(Tag.DATA, data=b'abc', count=None, checksum=None)
>>> record.compute_count()
3
>>> record.count is None
True
>>> _ = record.update_count()
>>> record.count
3
```

validate(*checksum=True*, *count=True*)

Validates consistency of attribute values.

All the record attributes are checked for consistency.

Please refer to the implementation for more details.

Parameters

- **checksum** (*bool*) – Check the consistency of the `checksum` attribute.
- **count** (*bool*) – Check the consistency of the `count` attribute.

Returns

`BaseRecord` – *self*.

Raises

ValueError – Some targeted attributes are inconsistent.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> _ = record.validate()
>>> record.data = b'abc'
>>> _ = record.update_count().update_checksum().validate()
Traceback (most recent call last):
    ...
ValueError: unexpcted data
```

4.10.3 TiTxtTag

class hexrec.formats.titxt.**TiTxtTag**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Texas Instruments TI-TXT tag.

Attributes

DATA	Data.
ADDRESS	Address.
EOF	End Of File.
denominator	the denominator of a rational number in lowest terms
imag	the imaginary part of a complex number
numerator	the numerator of a rational number in lowest terms
real	the real part of a complex number

Methods

is_address	Tells whether this is an address record.
is_eof	Tells whether this is an End Of File record.
__init__	
as_integer_ratio	Return integer ratio.
bit_count	Number of ones in the binary representation of the absolute value of self.
bit_length	Number of bits necessary to represent self in binary.
conjugate	Returns self, the complex conjugate of any int.
from_bytes	Return the integer represented by the given array of bytes.
to_bytes	Return an array of bytes representing an integer.

ADDRESS = 1

Address.

DATA = 0

Data.

EOF = 2

End Of File.

_DATA: Optional[*BaseTag*] = 0

Alias to a common data record tag.

This tag is used internally to build a generic data record.

__abs__()

abs(self)

__add__(*value*, */*)

Return self+value.

__and__(*value*, /)
Return self&value.

__bool__()
True if self else False

__ceil__()
Ceiling of an Integral returns itself.

classmethod __contains__(*member*)
Return True if member is a member of this enum raises TypeError if member is not an enum member

note: in 3.12 TypeError will no longer be raised, and True will also be returned if member is the value of a member in this enum

__dir__()
Returns all members and all public methods

__divmod__(*value*, /)
Return divmod(self, value).

__eq__(*value*, /)
Return self==value.

__float__()
float(self)

__floor__()
Flooring an Integral returns itself.

__floordiv__(*value*, /)
Return self//value.

__format__(*format_spec*, /)
Default object formatter.

__ge__(*value*, /)
Return self>=value.

__getattribute__(*name*, /)
Return getattr(self, name).

classmethod __getitem__(*name*)
Return the member matching *name*.

__gt__(*value*, /)
Return self>value.

__hash__()
Return hash(self).

__index__()
Return self converted to an integer, if self is suitable for use as an index into a list.

__init__(**args*, ***kwds*)

__int__()
int(self)

__invert__()
~self

classmethod __iter__()
Return members in definition order.

__le__(*value*, /)
Return self<=value.

classmethod __len__()
Return the number of members (no aliases)

__lshift__(*value*, /)
Return self<<value.

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

__mul__(*value*, /)
Return self*value.

__ne__(*value*, /)
Return self!=value.

__neg__()
-self

__new__(*value*)

__or__(*value*, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod=None*, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__reduce_ex__(*proto*)
Helper for pickle.

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod=None*, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(*value*, /)
Return self-value.

__truediv__(*value*, /)
Return self/value.

__trunc__()
Truncating an Integral returns itself.

__xor__(*value*, /)
Return self^value.

_generate_next_value_(*start*, *count*, *last_values*)
Generate the next value when not given.

name: the name of the member start: the initial start value or None count: the number of existing members last_values: the list of values assigned

_member_type_
alias of int

_new_member_(***kwargs*)
Create and return a new object. See help(type) for accurate signature.

_value_repr_()
Return repr(self).

as_integer_ratio()
Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()
Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()
Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()
Returns self, the complex conjugate of any int.

denominator
the denominator of a rational number in lowest terms

from_bytes(*byteorder='big'*, ***, *signed=False*)
Return the integer represented by the given array of bytes.

bytes
Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

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. Default is to use 'big'.

signed
Indicates whether two's complement is used to represent the integer.

`imag`

the imaginary part of a complex number

`is_address`()

Tells whether this is an address record.

This method returns true if this record tag is used for *address* records.

Returns
bool – This is an address record tag.

Examples

```
>>> from hexrec import TiTxtFile
>>> TiTxtTag = TiTxtFile.Record.Tag
>>> TiTxtTag.ADDRESS.is_address()
True
>>> TiTxtTag.DATA.is_address()
False
```

`is_eof`()

Tells whether this is an End Of File record.

This method returns true if this record tag is used for *End Of File* records.

Returns
bool – This is an End Of File record tag.

Examples

```
>>> from hexrec import TiTxtFile
>>> TiTxtTag = TiTxtFile.Record.Tag
>>> TiTxtTag.EOF.is_eof()
True
>>> TiTxtTag.DATA.is_eof()
False
```

`numerator`

the numerator of a rational number in lowest terms

`real`

the real part of a complex number

`to_bytes`(*length=1*, *byteorder='big'*, ***, *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. Default is length 1.

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. Default is to use 'big'.

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.

4.11 xtek

Tektronix extended HEX format.

See also:

https://en.wikipedia.org/wiki/Tektronix_extended_HEX

Classes

XtekFile	Tektronix Extended file object.
XtekRecord	Tektronix Extended record object.
XtekTag	Tektronix Extended tag.

4.11.1 XtekFile

class hexrec.formats.xtek.**XtekFile**

Tektronix Extended file object.

Attributes

DEFAULT_DATALEN	Default data attribute length.
FILE_EXT	Supported filename extensions.
META_KEYS	Meta information key names.
maxdatalen	Maximum byte size of the data field.
memory	Memory object stored by records role.
records	Records stored by records role.
startaddr	Start address.

Methods

__init__	
align	Pads blocks to align their boundaries.
append	Appends a byte.
apply_records	Applies records to memory and meta.
clear	Clears data within a range.
convert	Converts a file object to another format.

continues on next page

Table 9 – continued from previous page

copy	Copies within a range.
crop	Clears data outside a range.
cut	Cuts data within a range.
delete	Deletes data within a range.
discard_memory	Discards underlying memory.
discard_records	Discards underlying records.
extend	Concatenates data.
fill	Fills a range.
find	Finds a substring.
flood	Floods a range.
from_blocks	Creates a file object from a memory object.
from_bytes	Creates a file object from a byte string.
from_memory	Creates a file object from a memory object.
from_records	Creates a file object from records.
get_address_max	Maximum address within memory.
get_address_min	Minimum address within memory.
get_holes	List of memory holes.
get_meta	Meta information.
get_spans	List of memory block spans.
index	Finds a substring.
load	Loads a file object from the filesystem.
merge	Merges data onto the file.
parse	Parses records from a byte stream.
print	Prints record content to stdout.
read	Extracts a substring.
save	Saves a file object into the filesystem.
serialize	Serializes records onto a byte stream.
set_meta	Sets meta information.
shift	Shifts data addresses by an offset.
split	Splits into parts.
update_records	Applies memory and meta to records.
validate_records	Validates records.
view	Memory view.
write	Writes data into the file.

DEFAULT_DATALEN: int = 16

Default data attribute length.

Default value for the *maxdatalen* *meta*, which sets the maximum size of `BaseRecord.data` field values.

FILE_EXT: Sequence[int] = ['.tek', '.xtek']

Supported filename extensions.

Sequence of file name extension substrings (e.g. `.hex`). This list is used by functions like `guess_format_name()` to manage mapping of file *formats*.

META_KEYS: Sequence[str] = ['maxdatalen', 'startaddr']

Meta information key names.

Sequence of key strings listing the supported *meta* information of this file *format*.

Record

alias of *XtekRecord*

__add__(*other*)

Concatenates with another file.

Equivalent to *copy()* then *extend()*.

Parameters
other (BaseFile or bytes) – Other file or bytes to concatenate.

Returns
BaseFile – Concatenation of *self* and *other*.

See also:

copy() extend()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 + file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file4 = file3 + b'789'
>>> file4.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__bool__()

bool: Has data records or memory.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile()
>>> bool(file)
False
>>> _ = file.append(0)
>>> bool(file)
True
```

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> file = IhexFile.from_records([IhexRecord.create_end_of_file()])
>>> bool(file)
False
>>> file.records.insert(0, IhexRecord.create_data(0, b'\0'))
>>> bool(file)
True
```

__delitem__(*key*)

Deletes a range.

Parameters
key (*slice or int*) – Range to delete.

See also:

`bytesparse.base.MutableMemory.__delitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> del file[457]
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xz']]
>>> del file[125:457]
>>> file.memory.to_blocks()
[[123, b'abz']]
```

__eq__(*other*)

Equality test.

The file objects *self* and *other* are considered *equal* if the inequality tests of *__ne__()* result false.

Returns
bool – *self* and *other* are *equal*.

See Also
__ne__()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 == file2
True
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 == file3
False
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 == file4
False
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file == ihex_file
False
>>> srec_file.memory == ihex_file.memory
True
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__getitem__(*key*)

Extracts a range.

Parameters
key (*slice or int*) – Range to extract.

Raises
ValueError – invalid range.

See also:

`bytesparse.base.ImmutableMemory.__getitem__()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> chr(file[457])
'y'
>>> repr(file[333])
'None'
>>> file[123:125]
b'ab'
>>> file[125:457]
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

__hash__ = None

__iadd__(*other*)

Concatenates data.

Equivalent to *extend()*.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – *self*.

See also:

memory extend() discard_records() bytesparse.base.MutableMemory.extend()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 += file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> file1 += b'789'
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

__init__()

__ior__(*other*)

Merges with another file.

Equivalent to *merge()*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self*.

See also:

merge() discard_records()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file1 |= file2
>>> file1.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file1 |= b'789'
>>> file1.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__ne__(*other*)

Inequality test.

The file objects *self* and *other* are considered *unequal* if any of the following tests result true:

- Both have *memory role* (i.e. *memory*), resulting unequal;
- Both have *records role* (i.e. *records*), resulting unequal;
- *other* does not have a *meta* listed by *META_KEYS*;
- A *meta* value (among those of *META_KEYS*) is different.

Returns
bool – *self* and *other* are *unequal*.

See also:

__eq__() memory records META_KEYS

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'abc', offset=123)
>>> file1 is file2
False
>>> file1 != file2
False
>>> file3 = SrecFile.from_bytes(b'xyz', offset=123)
>>> file1 != file3
True
>>> file4 = SrecFile.from_bytes(b'abc', offset=456)
>>> file1 != file4
True
```

```
>>> from hexrec import SrecFile, IhexFile
>>> srec_file = SrecFile.from_bytes(b'abc', offset=123)
>>> ihex_file = IhexFile.from_bytes(b'abc', offset=123)
>>> srec_file != ihex_file
True
>>> srec_file.memory != ihex_file.memory
False
>>> set(srec_file.META_KEYS) - set(ihex_file.META_KEYS)
{'header'}
```

__or__(*other*)

Merges with another file.

Equivalent to *copy()* then *merge()*.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to merge.

Returns
`BaseFile` – *self* merged with *other*.

See also:

copy() merge()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = file1 | file2
>>> file3.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file4 = file3 | b'789'
>>> file4.memory.to_blocks()
[[0, b'789'], [123, b'abc'], [456, b'xyz']]
```

__setitem__(*key*, *value*)

Sets a range.

Parameters

- **key** (*slice or int*) – Range to set.
- **value** (bytes, `bytesparse.base.ImmutableMemory`, `None`) – Value(s) to set. `None` acts like *clear()*.

Raises

ValueError – invalid range.

See also:

`bytesparse.base.MutableMemory.__setitem__()` *clear()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file[124] = b'?'
>>> file.memory.to_blocks()
[[123, b'a?c'], [456, b'xyz']]
>>> file[:125] = None
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'xyz']]
>>> file[457:458] = b'789'
>>> file.memory.to_blocks()
[[125, b'c'], [456, b'x789z']]
```

__weakref__

list of weak references to the object (if defined)

classmethod _is_line_empty(*line*)

Empty line check.

Tells whether a *line* has no meaningful content (e.g. all whitespace). The check itself depends on the implementing file *format*. It may be used internally to skip empty lines, e.g. by *parse()*.

Parameters
line (*bytes*) – A line, byte string.

Returns
bool: The *line* is empty.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexFile._is_line_empty(b'')
True
>>> IhexFile._is_line_empty(b' \t\v\r\n')
True
>>> IhexFile._is_line_empty(b':00000001FF\r\n')
False
```

align(*modulo*, *start=None*, *endex=None*, *pattern=0*)

Pads blocks to align their boundaries.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*, so that memory blocks are aligned to the required *modulo*.

Any stored *records* are discarded upon return.

Parameters

- **modulo** (*int*) – Alignment modulo.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns
`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.align()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [134, b'xyz']])
>>> _ = file.align(4, pattern=b'.')
>>> file.memory.to_blocks()
[[120, b'...abc..'], [132, b'..xyz...']]
```

append(*item*)

Appends a byte.

It appends the *item* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
item (*byte or int*) – Byte to append.

Returns
BaseFile – *self*.

See also:

memory discard_records() bytesparse.base.MutableMemory.append()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123)
>>> _ = file.append(b'.')
>>> _ = file.append(0)
>>> file.memory.to_blocks()
[[123, b'abc.\x00']]
```

apply_records()

Applies records to memory and meta.

This method processes the stored *records*, converting *data* as *memory*, and special records into their *meta* counterparts.

This effectively converts the *records role* into the *memory role* (keeping both).

The *memory* and *meta* are assigned upon return. Any exceptions being raised should not alter the file object.

Returns
BaseFile – *self*.

Raises
ValueError – *records* attribute not populated.

See also:

records memory get_meta() update_records()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_start_linear_address(456),
```

(continues on next page)

(continued from previous page)

```
...             IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records, maxdatalen=16)
>>> _ = file.apply_records()
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 16, 'startaddr': 456}
```

clear(*start=None*, *endex=None*)

Clears data within a range.

It clears the specified range of underlying *memory* object, making a memory hole.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.clear()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.clear(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

classmethod convert(*source*, *meta=True*)

Converts a file object to another format.

It copies the *memory* and *meta* of the *source* file object, creating a new one of the target `BaseFile` format type.

Parameters

- **source** (`BaseFile`) – Source file object to convert.
- **meta** (*bool*) – Copy *meta* information to the target file object. Only the keys of the target *META_KEYS* are processed.

Returns

`BaseFile` – Converted copy of *source* to the target format.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> source = IhexFile.from_blocks(blocks, startaddr=789)
>>> target = SrecFile.convert(source)
>>> target.memory is source.memory
False
>>> target.memory == source.memory
True
>>> target.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 789}
```

copy(*start=None*, *endex=None*, *meta=True*)

Copies within a range.

It copied data within the specified range of the file object, creating a new one carrying the inner slice.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns

`BaseFile` – *self*.

See also:

memory *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.copy(start=124, endex=132)
>>> inner.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

crop(*start=None*, *endex=None*)

Clears data outside a range.

It clears outside the specified range of underlying *memory* object, timming it.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.crop()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.crop(start=124, endex=132)
>>> file.memory.to_blocks()
[[124, b'bc'], [130, b'xy']]
```

cut(*start=None*, *endex=None*, *meta=False*)

Cuts data within a range.

It takes data within the specified range away from the file object, creating a new one carrying the inner slice. The inner slice is cleared from *self*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **meta** (*bool*) – Copy *meta* information to the created file object.

Returns
`BaseFile` – *self*.

See also:

memory *clear()* *get_meta()* *discard_records()* `bytesparse.base.MutableMemory.cut()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> inner = file.cut(start=124, endex=132)
>>> inner.memory.to_blocks()
```

(continues on next page)

(continued from previous page)

```
[[124, b'bc'], [130, b'xy']]
>>> file.memory.to_blocks()
[[123, b'a'], [132, b'z']]
```

delete(*start=None*, *endex=None*)

Deletes data within a range.

It deletes the specified range of underlying *memory* object, shifting all subsequent data towards the collapsed range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.delete()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.delete(start=124, endex=132)
>>> file.memory.to_blocks()
[[123, b'az']]
```

discard_memory()

Discards underlying memory.

The underlying *memory* object is assigned `None`.

If the underlying *records* object is `None`, it is assigned a new empty memory object.

Returns

`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_bytes(b'abc', offset=123)
>>> _ = file.update_records()
>>> _ = file.discard_memory()
>>> _ = file.update_records()
Traceback (most recent call last):
    ...
ValueError: memory instance required
```

discard_records()

Discards underlying records.

The underlying *records* object is assigned `None`.

If the underlying *memory* object is `None`, it is assigned a new empty memory object.

Returns
`BaseFile` – *self*.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> _ = file.validate_records()
>>> _ = file.discard_records()
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: records required
```

extend(*other*)

Concatenates data.

It concatenates *other* to the underlyng *memory*.

Any stored *records* are discarded upon return.

Parameters
other (`BaseFile` or bytes) – Other file or bytes to concatenate.

Returns
`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.extend()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> _ = file1.extend(file2)
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz']]
>>> _ = file1.extend(b'789')
>>> file1.memory.to_blocks()
[[123, b'abc'], [582, b'xyz789']]
```

fill(*start=None*, *endex=None*, *pattern=0*)

Fills a range.

It writes a *pattern* of bytes onto the underlying *memory* object, overwriting anything within the specified range.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.fill()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> _ = file.fill(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'a.......z']]
```

find(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it returns `-1`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address; `-1` if not found.

See also:

index `bytesparse.base.ImmutableMemory.find()`

Notes

The internal *memory* might allow negative addresses for its stored data. In that case, *index()* would be more appropriate, because it raises an exception when the *item* is not found.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.find(b'yz')
457
>>> file.find(ord('b'))
124
>>> file.find(b'?')
-1
```

flood(*start=None*, *endex=None*, *pattern=0*)

Floods a range.

It fills memory holes of the underlying *memory* within the specified range with a *pattern*.

Any stored *records* are discarded upon return.

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **pattern** (*bytes or int*) – Byte pattern for flooding.

Returns

`BaseFile` – *self*.

See also:

memory discard_records() `bytesparse.base.MutableMemory.flood()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.get_holes()
[(126, 130)]
>>> _ = file.flood(start=124, endex=132, pattern=b'.')
>>> file.memory.to_blocks()
[[123, b'abc....xyz']]
```

classmethod from_blocks(*blocks*, ***meta*)

Creates a file object from a memory object.

The *blocks* are put into the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **blocks** (*list of blocks*) – Memory blocks to put into *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns

`BaseFile` – The created file object.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* `bytesparse.base.ImmutableMemory.from_blocks()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_bytes(*data*, *offset=0*, ***meta*)

Creates a file object from a byte string.

The byte string makes a single *data* block, placed at some offset within the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **data** (*bytes*) – A byte string used to make a single data block.
- **offset** (*int*) – Offset of the single data block within *memory*.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS *from_memory()* bytesparse.base.ImmutableMemory.from_bytes()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'abc', offset=123, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
8
```

classmethod **from_memory**(*memory=None*, ***meta*)

Creates a file object from a memory object.

The *memory* is set as the *memory* of the created file object.

This method creates a file object in *memory role*. This means that only its *memory* is internally instanced, while the *records* requires manual or lazy instancing (i.e. either via direct call to *update_records()*, or any other methods indirectly calling it).

Parameters

- **memory** (bytesparse.base.MutableMemory) – Memory object to set as *memory*. If None, an empty memory object is automatically created.
- **meta** – *Meta* attributes to set, among *META_KEYS*.

Returns
BaseFile – The created file object.

Raises
KeyError – invalid *meta* key.

See also:

META_KEYS bytesparse.base.MutableMemory

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from bytesparse import Memory
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> memory = Memory.from_blocks(blocks)
>>> from hexrec import SrecFile
>>> file = SrecFile.from_memory(memory, maxdatalen=8)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> file.maxdatalen
8
```

classmethod from_records(*records*, *maxdatalen=None*)

Creates a file object from records.

The *records* sequence is set as the `record` attribute of the created file object.

This method creates a file object in *records role*. This means that only its *records* is internally instanced, while the *memory* requires manual or lazy instancing (i.e. either via direct call to *apply_records()*, or any other methods indirectly calling it).

Parameters

- **records** (list of `BaseRecord`) – Record sequence to set as *records*.
- **maxdatalen** (`Optional[int]`) – Maximum record *data* field size. If `None`, the maximum non-zero size of the *data* field from the *records* sequence is used. If all the *records* have zero sized *data* field, the class attribute *DEFAULT_DATALEN* is used.

Returns

`BaseFile` – The created file object.

Raises

ValueError – invalid *meta* values.

See also:

`BaseRecord`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> records = [IhexRecord.create_data(123, b'abc'),
...            IhexRecord.create_end_of_file()]
>>> file = IhexFile.from_records(records)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.maxdatalen
3
```

get_address_max()

Maximum address within memory.

It returns the maximum address of the underlying *memory* object.

Returns
int – Maximum address.

See also:

`bytesparse.base.ImmutableMemory.endin`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_max()
458
```

get_address_min()

Minimum address within memory.

It returns the minimum address of the underlying *memory* object.

Returns
int – Minimum address.

See also:

`bytesparse.base.ImmutableMemory.start`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.get_address_min()
123
```

get_holes()

List of memory holes.

It scans the underlying *memory* and returns the list of memory holes/gaps.

Each hole is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory hole boundaries.

See also:

`bytesparse.base.ImmutableMemory.gaps()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_holes()
[(126, 456), (459, 789)]
```

`get_meta()`

Meta information.

It builds and returns a dictionary of *meta* information. Meta keys are taken from the *META_KEYS* class attribute.

Returns
dict – Meta information dictionary.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks, header=b'HDR\0')
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 0}
```

`get_spans()`

List of memory block spans.

It scans the underlying *memory* and returns the list of memory block spans/intervals.

Each span is a couple of `(start, stop)` addresses (as per `slice` or `range()`).

Returns
list of couples – List of memory block boundaries.

See also:

`bytesparse.base.ImmutableMemory.intervals()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_spans()
[(123, 126), (456, 459), (789, 791)]
```

index(*item*, *start=None*, *endex=None*)

Finds a substring.

It searches the provided *item* within the specified address range, returning the first matching address.

If not found, it raises `ValueError`.

Parameters

- **item** (*bytes or int*) – Byte pattern to find.
- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.

Returns

int – *item* beginning address.

Raises

ValueError – *item* not found.

See also:

find `bytesparse.base.ImmutableMemory.index()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> file.index(b'yz')
457
>>> file.index(ord('b'))
124
>>> file.index(b'?')
Traceback (most recent call last):
    ...
ValueError: subsection not found
```

classmethod load(*path*, **args*, ***kwargs*)

Loads a file object from the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *parse()* to load a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdin.buffer` is used.
- **args** – Forwarded to *parse()*.
- **kwargs** – Forwarded to *parse()*.

Returns

`BaseFile` – Loaded file object.

See also:

save() *parse()* `open()` `sys.stdin.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.load('data.hex')
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

property maxdatalen: int

Maximum byte size of the data field.

This property sets the maximum byte size of the *data* field of a serialized record.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Raises

ValueError – Invalid maximum data length.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> file.maxdatalen
16
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212223242526272829 2A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> file.maxdatalen = 8
>>> _ = file.print()
S0030000FC
S10B0000000102030405060 7D8
S10B000808090A0B0C0D0E0F90
S10B001010111213141516 1748
S10B001818191A1B1C1D1E1F00
S10B002020212223242526 27B8
S10B002828292A2B2C2D2E2F70
S10B003030313233343536 3728
```

(continues on next page)

(continued from previous page)

```
S10B003838393A3B3C3D3E3FE0
S5030008F4
S9030000FC
>>> file.maxdatalen = 0
Traceback (most recent call last):
    ...
ValueError: invalid maximum data length
```

Type
int

property memory: MutableMemory

Memory object stored by records role.

This readonly property exposes the memory object stored by the file object while in *memory role*.

If this property is accessed while the file object is not in *memory role*, it automatically activates it by an implicit call to *apply_records()*, with default arguments.

For more control activating the *memory role*, please call *apply_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *records role* (i.e. altering content of *records*) would implicitly discard *memory* via *discard_memory()*.

See also:

apply_records() *discard_memory()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz']]
>>> _ = file.write(789, b'?!')
>>> file.memory.to_blocks()
[[123, b'abc'], [456, b'xyz'], [789, b'?!']]
```

Type
`bytesparse.Memory`

merge(**files*, *clear=False*)

Merges data onto the file.

It writes the provided *files* onto *self*, in the provided order. Any common address ranges are overwritten.

Any stored *records* are discarded upon return.

Parameters

- **files** (BaseFile) – Files to merge.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

BaseFile – *self*.

See also:

clear() *discard_records()* *write()*

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file1 = SrecFile.from_bytes(b'abc', offset=123)
>>> file2 = SrecFile.from_bytes(b'xyz', offset=456)
>>> file3 = SrecFile.from_bytes(b'<<<?????>>>', offset=450)
>>> _ = file3.merge(file1, file2)
>>> file3.memory.to_blocks()
[[123, b'abc'], [450, b'<<<???xyz>>']]
```

classmethod parse(*stream*, *ignore_errors=False*, *ignore_after_termination=True*)

Parses records from a byte stream.

It executes BaseRecord.parse() for each line of the incoming *stream*, creating a new file object with the collected records calling *from_records()*.

Lines resulting empty by _is_empty_line() are just discarded.

Notes

Please refer to the actual implementation of each record file *format*, because it may be more specialized.

Parameters

- **stream** (*bytes IO or buffer*) – Stream or byte buffer to parse records from.
- **ignore_errors** (*bool*) – Ignore Exception raised by BaseRecord.parse().
- **ignore_after_termination** (*bool*) – Ignore anything after the termination record was parsed, if supported (e.g. *End Of File* or *start address* record, depending on the specific file *format*).

Returns

BaseFile – *self*.

See also:

parse() BaseRecord.parse() *from_records()* _is_empty_line()

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> buffer = b'''
...     :03DA7A0061626383
...     :040000050000CAFE2F
...     :00000001FF
... '''
>>> import io
>>> stream = io.BytesIO(buffer)
>>> file = IhexFile.parse(stream)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
>>> file = IhexFile.parse(buffer)
>>> file.memory.to_blocks()
[[55930, b'abc']]
>>> file.get_meta()
{'linear': True, 'maxdatalen': 3, 'startaddr': 51966}
```

print(**args*, *stream=None*, *color=False*, *start=None*, *stop=None*, ***kwargs*)

Prints record content to stdout.

This helper method prints each record of *records* via `BaseRecord.print()`. As such, it also supports colored tokens and streams different from *stdout*.

It is possible to print subset of the records by specifying the record index range.

Warning: This method is **NOT** equivalent to *serialize()*, because it just prints each record from *records*. Please use *serialize()* for an actual serialization of the whole file.

Parameters

- **args** – Forwarded to the underlying call to `to_tokens()`.
- **stream** (*byte stream*) – Stream to print onto. If `None`, *stdout* is used.
- **color** (*bool*) – Colorize record tokens with ANSI color codes.
- **start** (*int*) – Inclusive start record index of the specified range. If `None`, start from the first record.
- **stop** (*int*) – Exclusive end record index of the specified range. If negative, look back from the last index. If `None`, print up to the last record.
- **kwargs** – Forwarded to the underlying call to `to_tokens()`.

Returns

`BaseFile` – *self*.

See also:

`BaseRecord.print()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> buffer = bytes(range(64))
>>> file = SrecFile.from_bytes(buffer)
>>> _ = file.print()
S0030000FC
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212222324252627282 92A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
S5030004F8
S9030000FC
>>> _ = file.print(color=True, start=1, stop=-2)
S113000000010203040506070809 0A0B0C0D0E0F74
S113001010111213141516171819 1A1B1C1D1E1F64
S113002020212222324252627282 92A2B2C2D2E2F54
S113003030313233343536373839 3A3B3C3D3E3F44
```

read(*start=None*, *endex=None*, *fill=0*)

Extracts a substring.

It extracts a byte string from the specified range, filling any memory holes/gaps (without altering *memory*).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If `None`, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If `None`, extend after the end of the *memory*.
- **fill** (*bytes or int*) – Byte pattern for filling.

Returns

`BaseFile` – *self*.

See also:

memory `bytesparse.base.MutableMemory.extract()` `bytesparse.base.MutableMemory.to_bytes()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [130, b'xyz']])
>>> file.read(start=124, endex=132)
b'bc\x00\x00\x00\x00xy'
>>> file.read(start=124, endex=132, fill=b'.')
b'bc....xy'
```

(continues on next page)

(continued from previous page)

```
>>> file.memory.to_blocks()
[[123, b'abc'], [130, b'xyz']]
```

property records: MutableSequence[*BaseRecord*]

Records stored by records role.

This readonly property exposes the list of records stored by the file object while in *records role*.

If this property is accessed while the file object is not in *records role*, it automatically activates it by an implicit call to *update_records()*, with default arguments.

For more control activating the *records role*, please call *update_records()* manually, providing the desired arguments.

Notes

Most methods acting on the *memory role* (i.e. altering content of *memory*) would implicitly discard *records* via *discard_records()*.

See also:

update_records() *discard_records()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz']]
>>> file = SrecFile.from_blocks(blocks, startaddr=789)
>>> len(file.records)
5
>>> _ = file.print()
S0030000FC
S106007B61626358
S10601C878797AC5
S5030002FA
S9030315E4
>>> _ = file.update_records(data_tag=SrecFile.Record.Tag.DATA_32)
>>> _ = file.print()
S0030000FC
S3080000007B61626356
S308000001C878797AC3
S5030002FA
S70500000315E2
```

Type
list of `BaseRecord`

save(*path*, **args*, ***kwargs*)

Saves a file object into the filesystem.

The `open()` function creates a *stream* from the filesystem, allowing *serialize()* to save a file object.

Parameters

- **path** (*str*) – Path of the file within the filesystem. If `None`, `sys.stdout.buffer` is used.
- **args** – Forwarded to *serialize()*.
- **kwargs** – Forwarded to *serialize()*.

Returns

`BaseFile` – *self*.

See also:

load() *serialize()* `open()` `sys.stdout.buffer`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> _ = file.save('data.hex')
```

serialize(*stream*, **args*, ***kwargs*)

Serializes records onto a byte stream.

It executes `BaseRecord.serialize()` for each of the stored *records*.

Parameters

- **stream** (*bytes IO*) – Stream to serialize records onto.
- **args** – Forwarded to `BaseRecord.serialize()` of each record.
- **kwargs** – Forwarded to `BaseRecord.serialize()` of each record.

Returns

`BaseFile` – *self*.

See also:

parse() `BaseRecord.serialize()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> file = IhexFile.from_blocks([[0xDA7A, b'abc']], startaddr=0xCAFE)
>>> import sys
>>> _ = file.serialize(sys.stdout.buffer, end=b'\n')
:03DA7A0061626383
:040000050000CAFE2F
:00000001FF
```

set_meta(*meta*, *strict=True*)

Sets meta information.

It sets the provided *kwargs* to their matching *meta* attributes, as listed by *META_KEYS*.

Parameters

- **meta** (*dict*) – Mapping of the *meta* information to set.
- **strict** (*bool*) – All the keys within *meta* must exist within *META_KEYS*.

Returns

dict – Attribute values listed by *META_KEYS*.

Raises

KeyError – invalid *meta* key.

See also:

META_KEYS *get_meta()*

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> blocks = [[123, b'abc'], [456, b'xyz'], [789, b'?!']]
>>> file = SrecFile.from_blocks(blocks)
>>> file.get_meta()
{'header': b'', 'maxdatalen': 16, 'startaddr': 0}
>>> _ = file.set_meta(dict(header=b'HDR\0', startaddr=456))
>>> file.get_meta()
{'header': b'HDR\x00', 'maxdatalen': 16, 'startaddr': 456}
```

shift(*offset*)

Shifts data addresses by an offset.

It shifts addresses of the underlying *memory* object data blocks by the provided *offset* amount.

Any stored *records* are discarded upon return.

Parameters

offset (*int*) – Offset to apply to the underlying data block addresses.

Returns

`BaseFile` – *self*.

See also:

memory *discard_records()* `bytesparse.base.MutableMemory.shift()`

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> _ = file.shift(1000)
>>> file.memory.to_blocks()
[[1123, b'abc'], [1456, b'xyz']]
```

split(**addresses*, *meta=True*)

Splits into parts.

The provided *addresses* are sorted and used as markers to split *self* into parts.

Each part is the *copy()* of *self* within the range of that part, in *memory role* (i.e., *records* is not populated).

Parameters

- **addresses** (*int*) – Split points.
- **meta** (*bool*) – Each part inherits *meta* from *self*.

Returns

list of `BaseFile` – Parts after splitting.

Examples

NOTE: These examples are provided by `BaseFile`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_bytes(b'Hello, World!', offset=123)
>>> parts = file.split(128, 130)
>>> for part in parts: print(part.memory.to_blocks())
[[123, b'Hello']]
[[128, b', ']]
[[130, b'World!']]
>>> file.memory.to_blocks()
[[123, b'Hello, World!']]
```

property startaddr: int

Start address.

This property sets the *start address* of the serialized record file.

This is usually taken into account by *update_records()* while splitting *memory* into *records*.

Setting a different value triggers *discard_records()*.

Examples

```
>>> from hexrec import XtekFile
>>> file = XtekFile()
>>> file.startaddr
0
>>> _ = file.print()
%0E81E800000000
>>> file.startaddr = 0x87654321
>>> _ = file.print()
%0E842887654321
```

update_records(*align=False*, *addrlen=8*)

Applies memory and meta to records.

This method processes the stored *memory* and *meta* information to generate the sequence of *records*.

This effectively converts the *memory role* into the *records role* (keeping both).

The *records* is assigned upon return. Any exceptions being raised should not alter the file object.

Parameters

- **align** (*bool*) – Aligns data record chunk address bounds to *maxdatalen*.
- **addrlen** (*int*) – Address length, in *nibbles* (4-bit units).

Returns

XtekFile – *self*.

Raises

ValueError – *memory* attribute not populated.

See also:

records *memory* *get_meta()* *apply_records()*

Examples

```
>>> from hexrec import XtekFile
>>> blocks = [[123, b'abc']]
>>> file = XtekFile.from_blocks(blocks, maxdatalen=16, startaddr=456)
>>> file.memory.to_blocks()
[[123, b'abc']]
>>> file.get_meta()
{'maxdatalen': 16, 'startaddr': 456}
>>> _ = file.update_records()
>>> len(file.records)
2
>>> _ = file.print()
%1463D80000007B616263
%0E8338000001C8
```

validate_records(*data_ordering=False*, *start_within_data=False*)

Validates records.

It performs consistency checks for the underlying *records*.

Parameters

- **data_ordering** (*bool*) – Checks that the *data* record sequence has monotonically increasing addresses, without any overlapping.
- **start_within_data** (*bool*) – Requires *start address* fall within data carried by some *data* record.

Returns
SrecFile – *self*.

Raises
ValueError – Invalid record sequence.

Examples

```
>>> from hexrec import XtekFile
>>> records = [XtekFile.Record.create_data(123, b'abc')]
>>> file = XtekFile.from_records(records)
>>> _ = file.validate_records()
Traceback (most recent call last):
    ...
ValueError: missing end of file record
```

view(*start=None*, *endex=None*)

Memory view.

It returns a memoryview over the specified range, which must cover a *contiguous* data region (i.e. no memory holes within).

Parameters

- **start** (*int*) – Inclusive start address of the specified range. If None, start from the beginning of the *memory*.
- **endex** (*int*) – Exclusive end address of the specified range. If None, extend after the end of the *memory*.

Returns
memoryview – View of the specified range.

Raises
ValueError – non-contiguous data within range.

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile.from_blocks([[123, b'abc'], [456, b'xyz']])
>>> bytes(file.view(start=456, endex=458))
b'xy'
>>> bytes(file.view())
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

write(*address*, *data*, *clear=False*)

Writes data into the file.

It writes the provided *data* into the underlying *memory* object.

Any stored *records* are discarded upon return.

Parameters

- **address** (*int*) – Address where *data* has to be written.
- **data** (*bytes or memory*) – Byte data to write.
- **clear** (*bool*) – *clear()* the target address range before writing.

Returns

BaseFile – *self*.

See also:

memory *clear()* *discard_records()* bytesparse.base.MutableMemory.write()

Examples

NOTE: These examples are provided by BaseFile. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import SrecFile
>>> file = SrecFile()
>>> _ = file.write(123, b'abc')
>>> _ = file.write(555, ord('?'))
>>> _ = file.write(1000, SrecFile.from_bytes(b'xyz', offset=456))
>>> file.memory.to_blocks()
[[123, b'abc'], [555, b'?'], [1456, b'xyz']]
```

4.11.2 XtekRecord

class hexrec.formats.xtek.**XtekRecord**(**super_init_args*, *addrlen=8*, *validate=True*, ***super_init_kwargs*)

Tektronix Extended record object.

Attributes

EQUALITY_KEYS	Meta keys for equality checks.
LINE1_REGEX	Line parser regex, part 1.
LINE2_REGEX	Line parser regex, part 2.
META_KEYS	Meta keys.

Methods

__init__	
compute_address_max	Calculates the maximum address.
compute_checksum	Computes the checksum field value.
compute_count	Compute the count field value.
compute_data_max	Calculates the maximum data size.
copy	Shallow copy.
create_data	Creates a data record.
create_eof	Creates an End Of File record.
data_to_int	Interprets data bytes as integer.
get_address_max	Calculates the maximum address.
get_data_max	Calculates the maximum data size.
get_meta	Gets meta information.
parse	Parses a record from bytes.
print	Prints a record.
serialize	Serializes onto a stream.
to_bytestr	Converts into a byte string.
to_tokens	Converts into byte string tokens.
update_checksum	Updates the checksum field.
update_count	Updates the count field.
validate	Validates consistency of attribute values.

EQUALITY_KEYS: Sequence[str] = ['address', 'checksum', 'count', 'data', 'tag', 'addrlen']

Meta keys for equality checks.

Equality methods (*__eq__()* and *__ne__()*) check against these *meta* keys only. Any other *meta* keys are just ignored.

LINE1_REGEX = re.compile(b'^(?P<before>[^%]*)%(?P<count>[0-9A-Fa-f]{2})(?P<tag>[68])(?P<checksum>[0-9A-Fa-f]{2})(?P<addrlen>[1-9A-Fa-f])')

Line parser regex, part 1.

```
LINE2_REGEX = [re.compile(b'^(?P<address>[0-9A-Fa-f]{1})(?P<data>([0-9A-Fa-f]{2}){,
124})(?P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{2})(?P<data>([0-9A-Fa-f]{2}){,123})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{3})(?P<data>([0-9A-Fa-f]{2}){,123})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{4})(?P<data>([0-9A-Fa-f]{2}){,122})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{5})(?P<data>([0-9A-Fa-f]{2}){,122})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{6})(?P<data>([0-9A-Fa-f]{2}){,121})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{7})(?P<data>([0-9A-Fa-f]{2}){,121})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{8})(?P<data>([0-9A-Fa-f]{2}){,120})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{9})(?P<data>([0-9A-Fa-f]{2}){,120})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{10})(?P<data>([0-9A-Fa-f]{2}){,119})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{11})(?P<data>([0-9A-Fa-f]{2}){,119})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{12})(?P<data>([0-9A-Fa-f]{2}){,118})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{13})(?P<data>([0-9A-Fa-f]{2}){,118})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{14})(?P<data>([0-9A-Fa-f]{2}){,117})(?
P<after>[^\\r\\n]*)\\r?\\n$'),
re.compile(b'^(?P<address>[0-9A-Fa-f]{15})(?P<data>([0-9A-Fa-f]{2}){,117})(?
P<after>[^\\r\\n]*)\\r?\\n$')]
```

Line parser regex, part 2.

```
META_KEYS: Sequence[str] = ['address', 'after', 'before', 'checksum', 'coords',
'count', 'data', 'tag', 'addrlen']
```

Meta keys.

This sequence holds the *meta* keys for copying (see *copy()*).

Tag

alias of *XtekTag*

__bytes__()

Serializes the record into bytes.

Returns
bytes – Byte serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> bytes(record)
b':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> bytes(record)
b'abc'
```

__eq__(*other*)

Equality test.

This method returns true if *self* is considered equal to *other*.

As inequality is usually easier to check, this method is usually implemented as a trivial `not self != other` (*__ne__()*).

Parameters
other (`BaseRecord`) – Record to compare to.

Returns
bool – *self* equals *other*.

See also:

__ne__()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 == ihex2
True
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 == ihex3
False
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 == raw
False
```

__hash__ = None

__init__(**super_init_args*, *addrlen=8*, *validate=True*, ***super_init_kwargs*)

__ne__(*other*)

Ineuality test.

This method returns true if *self* is considered unequal to *other*.

Each attribute listed by *EQUALITY_KEYS* is compared between *self* and *other*. This method returns whether any attributes do not match.

Parameters
other (BaseRecord) – Record to compare to.

Returns
bool – *self* and *other* are unequal.

See also:

__eq__()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile, RawFile
>>> ihex1 = IhexFile.Record.create_data(0, b'abc')
>>> ihex2 = IhexFile.Record.create_data(0, b'abc')
>>> ihex1 is ihex2
False
>>> ihex1 != ihex2
False
>>> ihex3 = IhexFile.Record.create_data(0, b'xyz')
>>> ihex1 != ihex3
True
>>> raw = RawFile.Record.create_data(0, b'abc')
>>> ihex1 != raw
True
```

__repr__()

String representation.

It returns a string representation of the record content, for human understanding only.

Returns
str – String representation.

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> repr(record)
"<<class 'hexrec.formats.ihex.IhexRecord'> @...
  address:=0 after:=b'' before:=b'' checksum:=255 coords:=(-1, -1)
  count:=0 data:=b'' tag:=<IhexTag.END_OF_FILE: 1>>"
```

__str__()

Serializes the record into a string.

Returns
str – String serialization.

See also:

to_bytestr()

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> str(record)
':00000001FF\r\n'
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> str(record)
'abc'
```

__weakref__

list of weak references to the object (if defined)

classmethod compute_address_max(*addrlen*)

Calculates the maximum address.

It calculates the maximum *address* field given the number of *nibbles*.

Parameters
addrlen (*int*) – Address length, in *nibbles* (4-bit units).

Returns
int – Maximum *address* value.

Raises
ValueError – invalid *addrlen*.

Examples

```
>>> from hexrec import XtekFile
>>> XtekRecord = XtekFile.Record
>>> hex(XtekRecord.compute_address_max(4))
'0xffff'
>>> hex(XtekRecord.compute_address_max(6))
'0xffffff'
>>> hex(XtekRecord.compute_address_max(8))
'0xffffffff'
```

compute_checksum()

Computes the checksum field value.

It computes and returns the format-specific `checksum` value of a record.

When not specialized, it returns `None` by default.

Returns
int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_checksum()
215
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_checksum())
'None'
```

compute_count()

Compute the count field value.

It computes and returns the format-specific `count` value of a record.

When not specialized, it returns `None` by default.

Returns
int – Computed checksum value.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0, b'abc')
>>> record.compute_count()
3
```

```
>>> from hexrec import RawFile
>>> record = RawFile.Record.create_data(0, b'abc')
>>> repr(record.compute_count())
'None'
```

classmethod compute_data_max(*addrlen*)

Calculates the maximum data size.

It calculates the maximum *data* field size given the number of *nibbles*

Parameters
addrlen (*int*) – Address length, in *nibbles* (4-bit units).

Returns
int – Maximum *data* size.

Raises
ValueError – invalid *addrlen*.

Examples

```
>>> from hexrec import XtekFile
>>> XtekRecord = XtekFile.Record
>>> XtekRecord.compute_data_max(4)
122
>>> XtekRecord.compute_data_max(6)
121
>>> XtekRecord.compute_data_max(8)
120
```

copy(*validate=True*)

Shallow copy.

It calls the record constructor, passing *meta* to it.

Parameters
validate (*bool*) – Performs validation on instantiation (*__init__()*).

Returns
BaseRecord – Shallow copy.

See also:

__init__() *get_meta()*

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record1 = IhexFile.Record.create_data(0x1234, b'abc')
>>> record2 = record1.copy()
>>> record1 is record2
False
>>> record1 == record2
True
```

classmethod create_data(*address*, *data*, *addrlen=8*)

Creates a data record.

This is a mandatory class method to instantiate a *data* record.

Parameters

- **address** (*int*) – Record address. If not supported, set zero.
- **data** (*bytes*) – Record byte data.

- **addrlen** (*int*) – Address length, in *nibbles* (4-bit units).

Returns
XtekRecord – Data record object.

Raises
ValueError – invalid parameter.

Examples

```
>>> from hexrec import XtekFile
>>> record = XtekFile.Record.create_data(0x1234, b'abc')
>>> str(record)
'%1463580000123461626 3\r\n'
```

classmethod create_eof(*start=0*, *addrlen=8*)

Creates an End Of File record.

The End Of File record also carries the *start address*.

Parameters

- **start** (*int*) – Start address.
- **addrlen** (*int*) – Address length, in *nibbles* (4-bit units).

Returns
XtekRecord – End Of File record object.

Examples

```
>>> from hexrec import XtekFile
>>> record = XtekFile.Record.create_eof(start=0x12345678)
>>> str(record)
'%0E842812345678\r\n'
```

data_to_int(*byteorder='big'*, *signed=False*)

Interprets data bytes as integer.

It creates an integer from bytes of the `data` field.

Parameters

- **byteorder** (*'big' or 'little'*) – Byte order (endianness): either `'big'` (default) or `'little'`.
- **signed** (*bool*) – Signed integer (2-complement); default false.

Returns
int – Interpreted integer value.

See also:

`int.from_bytes()`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_extended_linear_address(0xABCD)
>>> record.data
b'\xab\xcd'
>>> addrext = record.data_to_int()
>>> addrext, hex(addrext)
(43981, '0xabcd')
```

get_address_max()

Calculates the maximum address.

It calculates the maximum *address* field for the calling tag. If the *address* field is not supported, it returns `None`.

Returns

int – Maximum *address* value, or `None`.

Examples

```
>>> from hexrec import XtekFile
>>> XtekRecord = XtekFile.Record
>>> record = XtekRecord.create_data(0xFFFF, b'abc', addrlen=4)
>>> hex(record.get_address_max())
'0xffff'
>>> record = XtekRecord.create_data(0xFFFF, b'abc', addrlen=6)
>>> hex(record.get_address_max())
'0xffffff'
>>> record = XtekRecord.create_data(0xFFFF, b'abc', addrlen=8)
>>> hex(record.get_address_max())
'0xffffffff'
```

get_data_max()

Calculates the maximum data size.

It calculates the maximum *data* field size given the number of *nibbles*

Returns

int – Maximum *data* size.

Raises

ValueError – invalid *addrlen*.

Examples

```
>>> from hexrec import XtekFile
>>> XtekRecord = XtekFile.Record
>>> record = XtekRecord.create_data(0xFFFF, b'abc', addrlen=4)
>>> record.get_data_max()
122
>>> record = XtekRecord.create_data(0xFFFF, b'abc', addrlen=6)
>>> record.get_data_max()
121
>>> record = XtekRecord.create_data(0xFFFF, b'abc', addrlen=8)
>>> record.get_data_max()
120
```

get_meta()

Gets meta information.

It returns all the object attributes whose keys are listed by *META_KEYS*.

Returns
dict – Attribute values listed by *META_KEYS*.

See also:

META_KEYS set_meta()

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> record.get_meta()
{'address': 0, 'after': b'', 'before': b'', 'checksum': 255,
 'coords': (-1, -1), 'count': 0, 'data': b'',
 'tag': <IhexTag.END_OF_FILE: 1>}
```

classmethod parse(*line*, *validate=True*)

Parses a record from bytes.

Please refer to the actual implementation provided by the record *format* for more details.

Parameters

- **line** (*bytes*) – String of bytes to parse.
- **validate** (*bool*) – Perform validation checks.

Returns
BaseRecord – Parsed record.

Raises
ValueError – Syntax error.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.parse(b':00000001FF\r\n')
>>> record.tag
<IhexTag.END_OF_FILE: 1>
>>> IhexFile.Record.parse(b'::00000001FF\r\n')
Traceback (most recent call last):
    ...
ValueError: syntax error
```

print(**args*, *stream=None*, *color=False*, ***kwargs*)

Prints a record.

The record is converted into tokens (eventually colorized) then joined and written onto a byte stream (*stdout* by default).

Parameters

- **args** – Forwarded to the underlying call to *to_tokens()*.
- **stream** (`io.BytesIO`) – The byte stream where the record tokens are printed. If `None`, *stdout* is selected.
- **color** (*bool*) – Tokens are colorized before printing.
- **kwargs** – Forwarded to the underlying call to *to_tokens()*.

Returns

`BaseRecord` – *self*.

See also:

to_tokens() `colorize_tokens()` `io.BytesIO`

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> _ = record.print()
:0312340061626391
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.print(stream=stream, color=True)
>>> stream.getvalue()
b'\x1b[0m\x1b[33m:\x1b[34m03\x1b[31m1234\x1b[32m00\x1b[36m61\x1b[96m62\
↪x1b[36m63\x1b[35m91\x1b[0m\r\n\x1b[0m'
```

serialize(*stream*, **args*, ***kwargs*)

Serializes onto a stream.

This wraps a call to *to_bytestr()* and `stream.write`.

Parameters

- **stream** (io.BytesIO) – Stream to write.
- **args** – Forwarded to *to_bytestr()*.
- **kwargs** – Forwarded to *to_bytestr()*.

Returns

BaseRecord – *self*.

See also:

to_bytestr() io.BytesIO

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> import io
>>> stream = io.BytesIO()
>>> _ = record.serialize(stream, end=b'\n')
>>> stream.getvalue()
b':0312340061626391\n'
```

to_bytestr(*end=b'\\r\\n'*)

Converts into a byte string.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns

bytes – Byte string representation.

Examples

NOTE: These examples are provided by BaseRecord. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_bytestr(end=b'\n')
b':0312340061626391\n'
```

to_tokens(*end=b'\\r\\n'*)

Converts into byte string tokens.

Parameters

- **args** – Implementation specific.
- **kwargs** – Implementation specific.

Returns
bytes – Mapping of token keys to token byte strings.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_data(0x1234, b'abc')
>>> record.to_tokens(end=b'\n')
{'before': b'', 'begin': b':', 'count': b'03', 'address': b'1234',
 'tag': b'00', 'data': b'616263', 'checksum': b'91', 'after': b'',
 'end': b'\n'}
```

update_checksum()

Updates the checksum field.

It updates the `checksum` attribute, assigning to it the value returned by *compute_checksum()*.

Returns
`BaseRecord` – *self*.

See also:

`checksum` *compute_checksum()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> IhexRecord = IhexFile.Record
>>> record = IhexRecord(IhexRecord.Tag.END_OF_FILE, checksum=None)
>>> record.compute_checksum()
255
>>> record.checksum is None
True
>>> _ = record.update_checksum()
>>> record.checksum
255
```

update_count()

Updates the count field.

It updates the `count` attribute, assigning to it the value returned by *compute_count()*.

Returns
`BaseRecord` – *self*.

See also:

`count` *compute_count()*

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> Record = IhexFile.Record
>>> Tag = Record.Tag
>>> record = Record(Tag.DATA, data=b'abc', count=None, checksum=None)
>>> record.compute_count()
3
>>> record.count is None
True
>>> _ = record.update_count()
>>> record.count
3
```

validate(*checksum=True*, *count=True*)

Validates consistency of attribute values.

All the record attributes are checked for consistency.

Please refer to the implementation for more details.

Parameters

- **checksum** (*bool*) – Check the consistency of the `checksum` attribute.
- **count** (*bool*) – Check the consistency of the `count` attribute.

Returns

`BaseRecord` – *self*.

Raises

ValueError – Some targeted attributes are inconsistent.

Examples

NOTE: These examples are provided by `BaseRecord`. Inherited classes for specific *formats* may require an adaptation.

```
>>> from hexrec import IhexFile
>>> record = IhexFile.Record.create_end_of_file()
>>> _ = record.validate()
>>> record.data = b'abc'
>>> _ = record.update_count().update_checksum().validate()
Traceback (most recent call last):
    ...
ValueError: unexpcted data
```

4.11.3 XtekTag

class hexrec.formats.xtek.**XtekTag**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Tektronix Extended tag.

Attributes

DATA	Data.
EOF	End Of File.
denominator	the denominator of a rational number in lowest terms
imag	the imaginary part of a complex number
numerator	the numerator of a rational number in lowest terms
real	the real part of a complex number

Methods

is_eof	Tells whether this is an End Of File record tag.
__init__	
as_integer_ratio	Return integer ratio.
bit_count	Number of ones in the binary representation of the absolute value of self.
bit_length	Number of bits necessary to represent self in binary.
conjugate	Returns self, the complex conjugate of any int.
from_bytes	Return the integer represented by the given array of bytes.
to_bytes	Return an array of bytes representing an integer.

DATA = 6

Data.

EOF = 8

End Of File.

_DATA: Optional['BaseTag'] = 6

Alias to a common data record tag.

This tag is used internally to build a generic data record.

__abs__()

abs(self)

__add__(*value*, */*)

Return self+value.

__and__(*value*, */*)

Return self&value.

__bool__()

True if self else False

__ceil__()

Ceiling of an Integral returns itself.

classmethod __contains__(*member*)

Return True if member is a member of this enum raises TypeError if member is not an enum member

note: in 3.12 TypeError will no longer be raised, and True will also be returned if member is the value of a member in this enum

__dir__()

Returns all members and all public methods

__divmod__(*value*, /)

Return divmod(self, value).

__eq__(*value*, /)

Return self==value.

__float__()

float(self)

__floor__()

Flooring an Integral returns itself.

__floordiv__(*value*, /)

Return self//value.

__format__(*format_spec*, /)

Default object formatter.

__ge__(*value*, /)

Return self>=value.

__getattribute__(*name*, /)

Return getattr(self, name).

classmethod __getitem__(*name*)

Return the member matching *name*.

__gt__(*value*, /)

Return self>value.

__hash__()

Return hash(self).

__index__()

Return self converted to an integer, if self is suitable for use as an index into a list.

__init__(**args*, ***kwds*)

__int__()

int(self)

__invert__()

~self

classmethod __iter__()

Return members in definition order.

__le__(*value*, /)
Return self<=value.

classmethod __len__()
Return the number of members (no aliases)

__lshift__(*value*, /)
Return self<<value.

__lt__(*value*, /)
Return self<value.

__mod__(*value*, /)
Return self%value.

__mul__(*value*, /)
Return self*value.

__ne__(*value*, /)
Return self!=value.

__neg__()
-self

__new__(*value*)

__or__(*value*, /)
Return self|value.

__pos__()
+self

__pow__(*value*, *mod=None*, /)
Return pow(self, value, mod).

__radd__(*value*, /)
Return value+self.

__rand__(*value*, /)
Return value&self.

__rdivmod__(*value*, /)
Return divmod(value, self).

__reduce_ex__(*proto*)
Helper for pickle.

__repr__()
Return repr(self).

__rfloordiv__(*value*, /)
Return value//self.

__rlshift__(*value*, /)
Return value<<self.

__rmod__(*value*, /)
Return value%self.

__rmul__(*value*, /)
Return value*self.

__ror__(*value*, /)
Return value|self.

__round__()
Rounding an Integral returns itself.

Rounding with an ndigits argument also returns an integer.

__rpow__(*value*, *mod=None*, /)
Return pow(value, self, mod).

__rrshift__(*value*, /)
Return value>>self.

__rshift__(*value*, /)
Return self>>value.

__rsub__(*value*, /)
Return value-self.

__rtruediv__(*value*, /)
Return value/self.

__rxor__(*value*, /)
Return value^self.

__sizeof__()
Returns size in memory, in bytes.

__str__()
Return repr(self).

__sub__(*value*, /)
Return self-value.

__truediv__(*value*, /)
Return self/value.

__trunc__()
Truncating an Integral returns itself.

__xor__(*value*, /)
Return self^value.

_generate_next_value_(*start*, *count*, *last_values*)
Generate the next value when not given.

name: the name of the member start: the initial start value or None count: the number of existing members last_values: the list of values assigned

_member_type_
alias of `int`

_new_member_(***kwargs*)
Create and return a new object. See help(type) for accurate signature.

_value_repr_()

Return repr(self).

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big'*, ***, *signed=False*)

Return the integer represented by the given array of bytes.

bytes
: Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

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. Default is to use 'big'.

signed
: Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

is_eof()

Tells whether this is an End Of File record tag.

This method returns true if this record tag is used for *End Of File* records.

Returns

bool – This is an End Of File record tag.

Examples

```
>>> from hexrec import XtekFile
>>> XtekTag = XtekFile.Record.Tag
>>> XtekTag.EOF.is_eof()
True
>>> XtekTag.DATA.is_eof()
False
```

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1*, *byteorder='big'*, ***, *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. Default is length 1.

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. Default is to use 'big'.

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.

4.12 hexdump

Emulation of the hexdump utility.

Attributes

CHAR_PRINTABLE	Printable characters lookup table.
CHAR_TOKENS	Character tokens lookup table.
DEFAULT_FORMAT_ORDER	Default order of display options.

4.12.1 CHAR_PRINTABLE

```
hexrec.hexdump.CHAR_PRINTABLE: Sequence[bytes] = [b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b' ', b'!', b'"', b'#',
b'$', b'%', b'&', b"'", b'(', b')', b'*', b'+', b',', b'-', b'.', b'/', b'0', b'1', b'2',
b'3', b'4', b'5', b'6', b'7', b'8', b'9', b':',  b';', b'<', b'=', b'>', b'?', b'@',
b'A', b'B', b'C', b'D', b'E', b'F', b'G', b'H', b'I', b'J', b'K', b'L', b'M', b'N', b'O',
b'P', b'Q', b'R', b'S', b'T', b'U', b'V', b'W', b'X', b'Y', b'Z', b'[', b'\\', b']',
b'^', b'_', b'`', b'a', b'b', b'c', b'd', b'e', b'f', b'g', b'h', b'i', b'j', b'k', b'l',
b'm', b'n', b'o', b'p', b'q', b'r', b's', b't', b'u', b'v', b'w', b'x', b'y', b'z', b'{',
b'|', b'}', b'~', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.',
b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b'.', b' ', b'>', b'<']
```

Printable characters lookup table.

4.12.2 CHAR_TOKENS

hexrec.hexdump.**CHAR_TOKENS: Sequence[bytes] = [b' \\0', b' 001', b' 002', b' 003', b' 004', b' 005', b' 006', b' \\a', b' \\b', b' \\t', b' \\n', b' \\v', b' \\f', b' \\r', b' 016', b' 017', b' 020', b' 021', b' 022', b' 023', b' 024', b' 025', b' 026', b' 027', b' 030', b' 031', b' 032', b' 033', b' 034', b' 035', b' 036', b' 037', b' ', b' !', b' "', b' #', b' $', b' %', b' &', b" '", b' (', b')', b' *', b' +', b' ,', b' -', b' .', b' /', b' 0', b' 1', b' 2', b' 3', b' 4', b' 5', b' 6', b' 7', b' 8', b' 9', b' :', b' ;', b' <', b' =', b' >', b' ?', b' @', b' A', b' B', b' C', b' D', b' E', b' F', b' G', b' H', b' I', b' J', b' K', b' L', b' M', b' N', b' O', b' P', b' Q', b' R', b' S', b' T', b' U', b' V', b' W', b' X', b' Y', b' Z', b' [', b' \\', b']', b' ^', b' _', b' `', b' a', b' b', b' c', b' d', b' e', b' f', b' g', b' h', b' i', b' j', b' k', b' l', b' m', b' n', b' o', b' p', b' q', b' r', b' s', b' t', b' u', b' v', b' w', b' x', b' y', b' z', b' {', b' |', b' }', b' ~', b' 177', b' 200', b' 201', b' 202', b' 203', b' 204', b' 205', b' 206', b' 207', b' 210', b' 211', b' 212', b' 213', b' 214', b' 215', b' 216', b' 217', b' 220', b' 221', b' 222', b' 223', b' 224', b' 225', b' 226', b' 227', b' 230', b' 231', b' 232', b' 233', b' 234', b' 235', b' 236', b' 237', b' 240', b' 241', b' 242', b' 243', b' 244', b' 245', b' 246', b' 247', b' 250', b' 251', b' 252', b' 253', b' 254', b' 255', b' 256', b' 257', b' 260', b' 261', b' 262', b' 263', b' 264', b' 265', b' 266', b' 267', b' 270', b' 271', b' 272', b' 273', b' 274', b' 275', b' 276', b' 277', b' 300', b' 301', b' 302', b' 303', b' 304', b' 305', b' 306', b' 307', b' 310', b' 311', b' 312', b' 313', b' 314', b' 315', b' 316', b' 317', b' 320', b' 321', b' 322', b' 323', b' 324', b' 325', b' 326', b' 327', b' 330', b' 331', b' 332', b' 333', b' 334', b' 335', b' 336', b' 337', b' 340', b' 341', b' 342', b' 343', b' 344', b' 345', b' 346', b' 347', b' 350', b' 351', b' 352', b' 353', b' 354', b' 355', b' 356', b' 357', b' 360', b' 361', b' 362', b' 363', b' 364', b' 365', b' 366', b' 367', b' 370', b' 371', b' 372', b' 373', b' 374', b' 375', b' 376', b' 377', b' ---', b' >>>', b' <<<']**

Character tokens lookup table.

4.12.3 DEFAULT_FORMAT_ORDER

hexrec.hexdump.**DEFAULT_FORMAT_ORDER: Sequence[str] = ['one_byte_octal', 'one_byte_hex', 'one_byte_char', 'canonical', 'two_bytes_decimal', 'two_bytes_octal', 'two_bytes_hex']**

Default order of display options.

Functions

hexdump_core	Emulation of the *hexdump* utility core.

4.12.4 hexdump_core

hexrec.hexdump.**hexdump_core**(*infile=None*, *outfile=None*, *one_byte_octal=False*, *one_byte_hex=False*, *one_byte_char=False*, *canonical=False*, *two_bytes_decimal=False*, *two_bytes_octal=False*, *two_bytes_hex=False*, *color=None*, *format=None*, *format_file=None*, *length=None*, *skip=None*, *no_squeezing=False*, *upper=False*, *width=16*, *linesep=None*, *format_order=None*)

Emulation of the *hexdump* utility core.

Parameters

- **infile** (*str or bytes*) – Input data. If `str`, it is considered as the input file path. If `bytes`, it is the input byte chunk. If `None`, it reads from the standard input.

- **outfile** (*str or bytes*) – Output data. If `str`, it is considered as the output file path. If `bytes`, it is the output byte chunk. If `None`, it writes to the standard output.
- **one_byte_octal** (*bool*) – One-byte octal display. Display the input offset in hexadecimal, followed by sixteen space-separated, three-column, zero-filled bytes of input data, in octal, per line.
- **one_byte_hex** (*bool*) – One-byte hexadecimal display. Display the input offset in hexadecimal, followed by sixteen space-separated, two-column, zero-filled bytes of input data, in hexadecimal, per line.
- **one_byte_char** (*bool*) – One-byte character display. Display the input offset in hexadecimal, followed by sixteen space-separated, three-column, space-filled characters of input data per line.
- **canonical** (*bool*) – Canonical hex+ASCII display. Display the input offset in hexadecimal, followed by sixteen space-separated, two-column, hexadecimal bytes, followed by the same sixteen bytes in %_p format enclosed in | characters. Invoking the program as hd implies this option.
- **two_bytes_decimal** (*bool*) – Two-byte decimal display. Display the input offset in hexadecimal, followed by eight space-separated, five-column, zero-filled, two-byte units of input data, in unsigned decimal, per line.
- **two_bytes_octal** (*bool*) – Two-byte octal display. Display the input offset in hexadecimal, followed by eight space-separated, six-column, zero-filled, two-byte quantities of input data, in octal, per line.
- **two_bytes_hex** (*bool*) – Two-byte hexadecimal display. Display the input offset in hexadecimal, followed by eight space-separated, four-column, zero-filled, two-byte quantities of input data, in hexadecimal, per line.
- **color** (*str*) – *CURRENTLY NOT SUPPORTED*. Please provide `None`.
- **format** (*str*) – *CURRENTLY NOT SUPPORTED*. Please provide `None`.
- **format_file** (*str*) – *CURRENTLY NOT SUPPORTED*. Please provide `None`.
- **length** (*int*) – Interpret only length bytes of input.
- **skip** (*int*) – Skip offset bytes from the beginning of the input.
- **no_squeezing** (*bool*) – The -v option causes hexdump to display all input data. Without the -v option, any number of groups of output lines which would be identical to the immediately preceding group of output lines (except for the input offsets), are replaced with a line comprised of a single asterisk.
- **upper** (*bool*) – Uses upper case hex letters on address and data.
- **width** (*int*) – Number of bytes per line.
- **linesep** (*bytes*) – Line separator bytes.
- **format_order** (*list of str*) – If not `None`, it indicates the order of display options (`one_byte_octal`, `one_byte_hex`, `one_byte_char`, `canonical`, `two_bytes_decimal`, `two_bytes_octal`, `two_bytes_hex`). Duplicates are allowed. Only those with the corresponding boolean argument true are used.

Returns

stream – The handle to the output stream.

4.13 utils

Generic utility functions.

Attributes

SUFFIX_SCALE	Integer suffix to scale factor.
DEFAULT_DELETE	Delete from hex strings.

4.13.1 SUFFIX_SCALE

hexrec.utils.**SUFFIX_SCALE: Mapping[str, int] = {'e': 1152921504606846976, 'eb': 1000000000000000000, 'eib': 1152921504606846976, 'g': 1073741824, 'gb': 1000000000, 'gib': 1073741824, 'k': 1024, 'kb': 1000, 'kib': 1024, 'm': 1048576, 'mb': 1000000, 'mib': 1048576, 'p': 1125899906842624, 'pb': 1000000000000000, 'pib': 1125899906842624, 't': 1099511627776, 'tb': 1000000000000, 'tib': 1099511627776, 'y': 1208925819614629174706176, 'yb': 1000000000000000000000000, 'yib': 1208925819614629174706176, 'z': 1180591620717411303424, 'zb': 1000000000000000000000, 'zib': 1180591620717411303424}**

Integer suffix to scale factor.

4.13.2 DEFAULT_DELETE

hexrec.utils.**DEFAULT_DELETE: bytes = b' \t.-:\r\n'**

Delete from hex strings.

Default values to delete from hexadecimal strings via *unhexlify()*. These are commonly used as byte separators or whitespace in hex strings.

Functions

chop	Chops a vector.
hexlify	Converts raw bytes into a hexadecimal byte string.
parse_int	Parses an integer.
unhexlify	Converts a hexadecimal byte string into raw bytes.

4.13.3 chop

hexrec.utils.**chop**(*vector*, *window*, *align_base=0*)

Chops a vector.

Iterates through the vector grouping its items into windows.

Parameters

- **vector** (*items*) – Vector to chop.
- **window** (*int*) – Window length.

- **align_base** (*int*) – Offset of the first window.

Yields
list or items – *vector* slices of up to *window* elements.

Examples

```
>>> list(chop(b'ABCDEFG', 2))
['AB', 'CD', 'EF', 'G']
```

```
>>> b':'.join(chop(b'ABCDEFG', 2))
b'AB:CD:EF:G'
```

```
>>> list(chop(b'ABCDEFG', 4, 3))
[b'A', b'BCDE', b'FG']
```

4.13.4 hexlify

hexrec.utils.**hexlify**(*bytestr*, *sep=None*, *upper=True*)

Converts raw bytes into a hexadecimal byte string.

Parameters

- **bytestr** (*bytes*) – Source byte string.
- **sep** (*bytes*) – Optional byte separator.
- **upper** (*bool*) – Uppercase hexadecimal string.

Returns
bytes – Hexadecimal byte string.

Examples

```
>>> from hexrec.utils import hexlify
>>> hexlify(b'\xAA\xBB\xCC')
b'AABBCC'
>>> hexlify(b'\xAA\xBB\xCC', sep=b' ')
b'AA BB CC'
>>> hexlify(b'\xAA\xBB\xCC', sep=b'-')
b'AA-BB-CC'
>>> hexlify(b'\xAA\xBB\xCC', upper=False)
b'aabbcc'
```

4.13.5 parse_int

hexrec.utils.**parse_int**(*value*)

Parses an integer.

Parameters

value (Union[str, Any]) – A generic object to convert to integer. In case *value* is a str (case-insensitive), it can be either prefixed with 0x or postfixed with h to convert from a hexadecimal representation, or prefixed with 0b from binary; a prefix of only 0 converts from octal. A further suffix applies a scale factor as per *SUFFIX_SCALE*. A None value evaluates as None. Any other object class will call the standard int().

Returns

int – None if *value* is None, its integer conversion otherwise.

Examples

```
>>> parse_int('-0xABk')
-175104
```

```
>>> parse_int(None) is None
True
```

```
>>> parse_int(123)
123
```

```
>>> parse_int(135.7)
135
```

4.13.6 unhexlify

hexrec.utils.**unhexlify**(*hexstr*, *delete=None*)

Converts a hexadecimal byte string into raw bytes.

If *delete*, its byte values are deleted from *hexstr* before evaluation. Useful to remove whitespace and separators.

Parameters

- **hexstr** (*bytes*) – Source hexadecimal byte string.
- **delete** (*bytes*) – If empty or None, no deletion occurs. If Ellipsis, *DEFAULT_DELETE* is used.

Returns

bytes – Raw byte string.

Examples

```
>>> from hexrec.utils import unhexlify
>>> unhexlify(b'AABBCC')
b'\xaa\xbb\xcc'
>>> unhexlify(b'AA BB CC', delete=...)
b'\xaa\xbb\xcc'
>>> unhexlify(b'AA-BB-CC', delete=...)
b'\xaa\xbb\xcc'
>>> unhexlify(b'AA/BB/CC', delete=b'/')
b'\xaa\xbb\xcc'
```

Classes

SparseMemoryIO	Sparse memory I/O wrapper.

4.13.7 SparseMemoryIO

class hexrec.utils.**SparseMemoryIO**(*memory=None*, *seek=None*)

Sparse memory I/O wrapper.

With respect to the parent class `bytesparse.io.MemoryIO`, it allows reading and writing memory *holes*.

Such holes are marked by the following integer values (instead of `None`):

- **`0x100` = hole byte within memory span**
  (`bytesparse.base.ImmutableMemory.span`);
- **`0x101` = hole byte before memory start address**
  (`bytesparse.base.ImmutableMemory.start`);
- **`0x102` = hole byte after memory end address**
  (`bytesparse.base.ImmutableMemory.endex`);

These special values allow displaying dedicated stuff when dumping memory data to standard output.

See also:

`bytesparse.io.MemoryIO` `bytesparse.base.ImmutableMemory.span` `bytesparse.base.ImmutableMemory.start` `bytesparse.base.ImmutableMemory.endex`

Attributes

closed	Closed stream.
memory	Underlying memory object.

Methods

__init__	
close	Closes the stream.
detach	Detaches the underlying raw stream.
fileno	File descriptor identifier.
flush	Flushes buffered data into the underlying raw steam.
getbuffer	Memory view of the underlying memory object.
getvalue	Byte string copy of the underlying memory object.
isatty	Interactive console stream.
peek	Previews the next chunk of bytes.
read	Reads a chunk of bytes.
read1	Reads a chunk of bytes.
readable	Stream is readable.
readinto	Reads data into a byte buffer.
readinto1	Reads data into a byte buffer.
readline	Reads a line.
readlines	Reads a list of lines.
seek	Changes the current stream position.
seekable	Stream is seekable.
skip_data	Skips a data block.
skip_hole	Skips a memory hole.
tell	Current stream position.
truncate	Truncates stream.
writable	Stream is writable.
write	Writes data into the stream.
writelines	Writes lines to the stream.

__del__()

Prepares the object for destruction.

It makes sure the stream is closed upon object destruction.

__enter__()

Context manager enter function.

Returns

MemoryIO – The stream object itself.

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> with MemoryIO(Memory.from_bytes(b'Hello, World!')) as stream:
...     data = stream.read()
>>> data
b'Hello, World!'
```

__exit__(*exc_type*, *exc_val*, *exc_tb*)

Context manager exit function.

It makes sure the stream is closed upon context exit.

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> with MemoryIO(Memory.from_bytes(b'Hello, World!')) as stream:
...     print(stream.closed)
False
>>> print(stream.closed)
True
```

__init__(*memory=None*, *seek=None*)

__iter__()

Iterates over lines.

Repeatedly calls *readline()*, as long as it returns byte strings. Yields the values returned by such calls.

Yields
bytes – Single line; terminator included.

See also:

readline()

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello\nWorld!'], [20, b'Bye\n'], [28, b'Bye!']]
>>> with MemoryIO(Memory.from_blocks(blocks)) as stream:
...     lines = [line for line in stream]
>>> lines
[b'Hello\n', b'World!', b'Bye\n', b'Bye!']
```

__new__(***kwargs*)

__next__()

Next iterated line.

Calls *readline()* once, returning the value.

Returns
bytes or int – Line read from the stream, or the negative gap size.

See also:

readline()

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello\nWorld!'], [20, b'Bye\n'], [28, b'Bye!']]
>>> with MemoryIO(Memory.from_blocks(blocks), seek=9) as stream:
...     print(next(stream))
b'World!'
```

_check_closed()

Checks if the stream is closed.

In case the stream is *closed*, it raises `ValueError`.

Raises

ValueError – The stream is closed.

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> with MemoryIO(Memory.from_bytes(b'ABC')) as stream:
...     stream._check_closed()
>>> stream._check_closed()
Traceback (most recent call last):
    ...
ValueError: I/O operation on closed stream.
```

close()

Closes the stream.

Any subsequent operations on the closed stream may fail, and some properties may change state.

The stream no more links to an underlying memory object.

See also:

closed

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> stream = MemoryIO(Memory.from_bytes(b'ABC'))
>>> stream.closed
False
>>> stream.memory is None
False
>>> stream.readable()
True
>>> stream.close()
>>> stream.closed
```

(continues on next page)

(continued from previous page)

```
True
>>> stream.memory is None
True
>>> stream.readable()
Traceback (most recent call last):
    ...
ValueError: I/O operation on closed stream.
```

property closed: bool

Closed stream.

Returns

bool – Closed stream.

See also:

close()

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> stream = MemoryIO(Memory.from_bytes(b'ABC'))
>>> stream.closed
False
>>> stream.close()
>>> stream.closed
True
```

```
>>> with MemoryIO(Memory.from_bytes(b'ABC')) as stream:
...     print(stream.closed)
False
>>> print(stream.closed)
True
```

detach()

Detaches the underlying raw stream.

Warning: It always raises `io.UnsupportedOperation`. This method is present only for API compatibility. No actual underlying stream is present for this object.

Raises

io.UnsupportedOperation – No underlying raw stream.

fileno()

File descriptor identifier.

Warning: It always raises `io.UnsupportedOperation`. This method is present only for API compatibility. No actual file descriptor is associated to this object.

Raises
OSError – Not a file stream.

flush()
Flushes buffered data into the underlying raw steam.

Notes

Since no underlying stream is associated, this method does nothing.

getbuffer()
Memory view of the underlying memory object.

> **Warning:** This method may fail when the underlying memory object has gaps within data.

Returns
memoryview – Memory view over the underlying memory object.

See also:

`ImmutableMemory.view()`

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> with MemoryIO(Memory.from_bytes(b'Hello, World!')) as stream:
...     with stream.getbuffer() as buffer:
...         print(type(buffer), '=', bytes(buffer))
<class 'memoryview'> = b'Hello, World!'
```

```
>>> blocks = [[3, b'Hello'], [10, b'World!']]
>>> with MemoryIO(Memory.from_blocks(blocks)) as stream:
...     stream.getbuffer()
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

getvalue()
Byte string copy of the underlying memory object.

> **Warning:** This method may fail when the underlying memory object has gaps within data.

Returns
bytes – Byte string copy of the underlying memory object.

See also:

`ImmutableMemory.to_bytes()`

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> with MemoryIO(Memory.from_bytes(b'Hello, World!')) as stream:
...     value = stream.getvalue()
...     print(type(value), '=', bytes(value))
<class 'bytes'> = b'Hello, World!'
```

```
>>> blocks = [[3, b'Hello'], [10, b'World!']]
>>> with MemoryIO(Memory.from_blocks(blocks)) as stream:
...     stream.getvalue()
Traceback (most recent call last):
    ...
ValueError: non-contiguous data within range
```

isatty()

Interactive console stream.

Returns

bool – `False`, not an interactive console stream.

property memory: ImmutableMemory | None

Underlying memory object.

`None` when *closed*.

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> memory = Memory.from_bytes(b'Hello, World!')
>>> with MemoryIO(memory) as stream:
...     print(stream.memory is memory)
True
>>> print(stream.memory is memory)
False
>>> print(stream.memory is None)
True
```

Type

`ImmutableMemory`

peek(*size=0*, *asmemview=False*)

Previews the next chunk of bytes.

Similar to *read()*, without moving the stream position instead. This method can be used to preview the next chunk of bytes, without affecting the stream itself.

The number of returned bytes may be different from *size*, which acts as a mere hint.

If the current stream position lies within a memory gap, this method returns the negative amount of bytes to reach the next data block.

If the current stream position is after the end of memory data, this method returns an empty byte string.

Parameters

- **size** (*int*) – Number of bytes to read. If negative or `None`, read as many bytes as possible.
- **asmemview** (*bool*) – Return a `memoryview` instead of `bytes`.

Returns

bytes – Chunk of bytes.

See also:

read()

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello'], [10, b'World!']]
>>> memory = Memory.from_blocks(blocks)
>>> stream = MemoryIO(memory, seek=4)
>>> stream.peek()
b''
>>> stream.peek(1)
b'e'
>>> stream.peek(11)
b'ello'
>>> stream.peek(None)
b'ello'
>>> stream.tell()
4
>>> memview = stream.peek(-1, asmemview=True)
>>> type(memview)
<class 'memoryview'>
>>> bytes(memview)
b'ello'
>>> stream.seek(8)
8
>>> stream.peek()
-2
```

read(*size=-1*, *asmemview=False*)

Reads a chunk of bytes.

Starting from the current stream position, this method tries to read up to *size* bytes (or as much as possible if negative or `None`).

The number of bytes can be less than *size* in the case a memory hole or the end are encountered.

If the current stream position lies within a memory gap, this method returns the negative amount of bytes to reach the next data block.

If the current stream position is after the end of memory data, this method returns an empty byte string.

Parameters

- **size** (*int*) – Number of bytes to read. If negative or `None`, read as many bytes as possible.

- **asmemview** (*bool*) – Return a `memoryview` instead of `bytes`.

Returns
bytes – Chunk of up to *size* bytes.

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello'], [10, b'World!']]
>>> memory = Memory.from_blocks(blocks)
>>> stream = MemoryIO(memory, seek=4)
>>> stream.read(1)
b'e'
>>> stream.tell()
5
>>> stream.read(99)
b'llo'
>>> stream.tell()
8
>>> stream.read()
-2
>>> stream.tell()
10
>>> memview = stream.read(None, asmemview=True)
>>> type(memview)
<class 'memoryview'>
>>> bytes(memview)
b'World!'
>>> stream.tell()
16
>>> stream.read()
b''
>>> stream.tell()
16
```

read1(*size=-1*, *asmemview=False*)

Reads a chunk of bytes.

Starting from the current stream position, this method tries to read up to *size* bytes (or as much as possible if negative or `None`).

The number of bytes can be less than *size* in the case a memory hole or the end are encountered.

If the current stream position lies within a memory gap, this method returns the negative amount of bytes to reach the next data block.

If the current stream position is after the end of memory data, this method returns an empty byte string.

Parameters

- **size** (*int*) – Number of bytes to read. If negative or `None`, read as many bytes as possible.
- **asmemview** (*bool*) – Return a `memoryview` instead of `bytes`.

Returns
bytes – Chunk of up to *size* bytes.

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello'], [10, b'World!']]
>>> memory = Memory.from_blocks(blocks)
>>> stream = MemoryIO(memory, seek=4)
>>> stream.read(1)
b'e'
>>> stream.tell()
5
>>> stream.read(99)
b'llo'
>>> stream.tell()
8
>>> stream.read()
-2
>>> stream.tell()
10
>>> memview = stream.read(None, asmemview=True)
>>> type(memview)
<class 'memoryview'>
>>> bytes(memview)
b'World!'
>>> stream.tell()
16
>>> stream.read()
b''
>>> stream.tell()
16
```

readable()

Stream is readable.

Returns

bool – `True`, stream is always readable.

readinto(*buffer*, *skipgaps=True*)

Reads data into a byte buffer.

If the stream is pointing after the memory end, no bytes are read.

If pointing within a memory hole (gap), the negative number of bytes until the next data block is returned. The stream is always positioned after the gap.

If a memory hole (gap) is encountered after reading some bytes, the reading stops there, and the number of bytes read is returned. The stream is always positioned after the gap.

Standard operation reads data until *buffer* is full, or encountering the memory end. It returns the number of bytes read.

Parameters

- **buffer** (*bytearray*) – A pre-allocated byte array to fill with bytes read from the stream.
- **skipgaps** (*bool*) – If false, it stops reading when a memory hole (gap) is encountered.

Returns

int – Number of bytes read, or the negative gap size.

Examples

```
>>> from bytesparse import Memory, MemoryIO
>>> blocks = [[3, b'Hello'], [10, b'World!']]
>>> memory = Memory.from_blocks(blocks)
```

```
>>> stream = MemoryIO(memory, seek=4)
>>> buffer = bytearray(b'.' * 8)
>>> stream.readinto(buffer, skipgaps=True)
8
>>> buffer
bytearray(b'elloWorl')
>>> stream.tell()
14
>>> stream.readinto(buffer, skipgaps=True)
2
>>> buffer
bytearray(b'd!loWorl')
>>> stream.tell()
16
>>> stream.readinto(buffer, skipgaps=True)
0
>>> buffer
bytearray(b'd!loWorl')
>>> stream.tell()
16
```

```
>>> stream = MemoryIO(memory, seek=4)
>>> buffer = bytearray(b'.' * 8)
>>> stream.readinto(buffer, skipgaps=False)
4
>>> buffer
bytearray(b'ello....')
>>> stream.tell()
8
>>> stream.readinto(buffer, skipgaps=False)
-2
>>> stream.tell()
10
>>> stream.readinto(buffer, skipgaps=False)
6
>>> buffer
bytearray(b'World!..')
>>> stream.tell()
16
>>> stream.readinto(buffer, skipgaps=False)
0
>>> buffer
bytearray(b'World!..')
```

(continues on next page)

(continued from previous page)

```
>>> stream.tell()
16
```

readinto1(*buffer*, *skipgaps=True*)

Reads data into a byte buffer.

If the stream is pointing after the memory end, no bytes are read.

If pointing within a memory hole (gap), the negative number of bytes until the next data block is returned. The stream is always positioned after the gap.

If a memory hole (gap) is encountered after reading some bytes, the reading stops there, and the number of bytes read is returned. The stream is always positioned after the gap.

Standard operation reads data until *buffer* is full, or encountering the memory end. It returns the number of bytes read.

Parameters

- **buffer** (*bytearray*) – A pre-allocated byte array to fill with bytes read from the stream.
- **skipgaps** (*bool*) – If false, it stops reading when a memory hole (gap) is encountered.

Returns

int – Number of bytes read, or the negative gap size.

Examples

```
>>> from bytesparse import Memory, MemoryIO
>>> blocks = [[3, b'Hello'], [10, b'World!']]
>>> memory = Memory.from_blocks(blocks)
```

```
>>> stream = MemoryIO(memory, seek=4)
>>> buffer = bytearray(b'.' * 8)
>>> stream.readinto(buffer, skipgaps=True)
8
>>> buffer
bytearray(b'elloWorl')
>>> stream.tell()
14
>>> stream.readinto(buffer, skipgaps=True)
2
>>> buffer
bytearray(b'd!loWorl')
>>> stream.tell()
16
>>> stream.readinto(buffer, skipgaps=True)
0
>>> buffer
bytearray(b'd!loWorl')
>>> stream.tell()
16
```

```
>>> stream = MemoryIO(memory, seek=4)
>>> buffer = bytearray(b'.' * 8)
>>> stream.readinto(buffer, skipgaps=False)
4
>>> buffer
bytearray(b'ello....')
>>> stream.tell()
8
>>> stream.readinto(buffer, skipgaps=False)
-2
>>> stream.tell()
10
>>> stream.readinto(buffer, skipgaps=False)
6
>>> buffer
bytearray(b'World!..')
>>> stream.tell()
16
>>> stream.readinto(buffer, skipgaps=False)
0
>>> buffer
bytearray(b'World!..')
>>> stream.tell()
16
```

readline(*size=-1*, *skipgaps=True*, *asmemview=False*)

Reads a line.

A standard line is a sequence of bytes terminating with a `b'\n'` newline character.

If *size* is provided (not `None` nor negative), the current line ends there, without a trailing newline character.

If the stream is pointing after the memory end, an empty byte string is returned.

If a memory hole (gap) is encountered, the current line ends there without a trailing newline character. The stream is always positioned after the gap.

If the stream points within a memory hole, it returns the negative number of bytes until the next data block. The stream is always positioned after the gap.

Parameters

- **size** (*int*) – Maximum number of bytes for the line to read. If `None` or negative, no limit is set.
- **skipgaps** (*bool*) – If false, the negative size of the pointed memory hole.
- **asmemview** (*bool*) – If true, the returned object is a `memoryview` instead of `bytes`.

Returns

bytes or int – Line read from the stream, or the negative gap size.

See also:

read() readlines()

Examples

```
>>> from bytesparse import Memory, MemoryIO
>>> blocks = [[3, b'Hello\nWorld!'], [20, b'Bye\n'], [28, b'Bye!']]
```

```
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.readline()
b'Hello\n'
>>> stream.tell()
9
>>> stream.readline(None)
b'World!'
>>> stream.tell()
15
>>> stream.readline(99)
b'Bye\n'
>>> stream.tell()
24
>>> stream.readline(99)
b'Bye!'
>>> stream.tell()
32
>>> stream.readline()
b''
>>> stream.tell()
32
```

```
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.readline(4)
b'Hell'
>>> stream.tell()
7
>>> stream.readline(4)
b'o\n'
>>> stream.tell()
9
```

```
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> view = stream.readline(asmemview=True)
>>> type(view) is memoryview
True
>>> bytes(view)
b'Hello\n'
```

```
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> # Emulating stream.readlines(skipgaps=False)
>>> lines = []
>>> line = True
>>> while line:
...     line = stream.readline(skipgaps=False)
...     lines.append(line)
>>> lines
```

(continues on next page)

(continued from previous page)

```
[-3, b'Hello\n', b'World!', -5, b'Bye\n', -4, b'Bye!']
>>> stream.tell()
32
>>> stream.readline(skipgaps=False)
b''
>>> stream.tell()
32
```

readlines(*hint=-1*, *skipgaps=True*, *asmemview=False*)

Reads a list of lines.

It repeatedly calls *readline()*, collecting the returned values into a list, until the total number of bytes read reaches *hint*.

If a memory hole (gap) is encountered, the current line ends there without a trailing newline character, and the stream is positioned after the gap.

If *skipgaps* is false, the list is appended the negative size of each encountered memory hole.

Parameters

- **hint** (*int*) – Number of bytes after which line reading stops. If `None` or negative, no limit is set.
- **skipgaps** (*bool*) – If false, the list hosts the negative size of each memory hole.
- **asmemview** (*bool*) – If true, the returned objects are memory views instead of byte strings.

Returns

list of bytes or int – List of lines and gaps read from the stream.

See also:

__iter__() read() readline()

Examples

```
>>> from bytesparse import Memory, MemoryIO
>>> blocks = [[3, b'Hello\nWorld!'], [20, b'Bye\n'], [28, b'Bye!']]
```

```
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.readlines()
[b'Hello\n', b'World!', b'Bye\n', b'Bye!']
>>> stream.tell()
32
>>> stream.readlines()
[]
>>> stream.tell()
32
```

```
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.readlines(hint=10)
[b'Hello\n', b'World!']
>>> stream.tell()
15
```

```
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> views = stream.readlines(asmemview=True)
>>> all(type(view) is memoryview for view in views)
True
>>> [bytes(view) for view in views]
[b'Hello\n', b'World!', b'Bye\n', b'Bye!']
```

```
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.readlines(skipgaps=False)
[-3, b'Hello\n', b'World!', -5, b'Bye\n', -4, b'Bye!']
>>> stream.tell()
32
>>> stream.readlines(skipgaps=False)
[]
>>> stream.tell()
32
```

seek(*offset*, *whence=0*)

Changes the current stream position.

It performs the classic `seek()` I/O operation.

The *whence* can be any of:

- **SEEK_SET (0 or None):**
  referring to the absolute address 0.
- **SEEK_CUR (1):**
  referring to the current stream position (*tell()*).
- **SEEK_END (2):**
  referring to the memory end (`ImmutableMemory.endex`).
- **SEEK_DATA (3):**
  if the current stream position lies within a memory hole, it moves to the beginning of the next data block; no operation is performed otherwise.
- **SEEK_HOLE (4):**
  if the current stream position lies within a data block, it moves to the beginning of the next memory hole (note: the end of the stream is considered as a memory hole); no operation is performed otherwise.

Parameters

- **offset** (*int*) – Position offset to apply.
- **whence** (*int*) – Where the offset is referred. It can be any of the standard `SEEK_*` values. By default, it refers to the beginning of the stream.

Returns

int – The updated stream position.

Notes

Stream position is just a number, not related to memory ranges.

Examples

```
>>> from bytesparse import *
```

```
>>> blocks = [[3, b'Hello'], [12, b'World!']]
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.seek(5)
5
>>> stream.seek(-3, SEEK_END)
15
>>> stream.seek(2, SEEK_CUR)
17
>>> stream.seek(1, SEEK_SET)
1
>>> stream.seek(stream.tell(), SEEK_HOLE)
1
>>> stream.seek(stream.tell(), SEEK_DATA)
3
>>> stream.seek(stream.tell(), SEEK_HOLE)
8
>>> stream.seek(stream.tell(), SEEK_DATA)
12
>>> stream.seek(stream.tell(), SEEK_HOLE)  # EOF
18
>>> stream.seek(stream.tell(), SEEK_DATA)  # EOF
18
>>> stream.seek(22)  # after
22
>>> stream.seek(0)  # before
0
```

seekable()

Stream is seekable.

Returns
bool – `True`, stream is always seekable.

skip_data()

Skips a data block.

It moves the current stream position after the end of the currently pointed data block.

No action is performed if the current stream position lies within a memory hole (gap).

Returns
int – Updated stream position.

See also:

seek()

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello'], [12, b'World!']]
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.skip_data()
0
>>> stream.seek(6)
6
>>> stream.skip_data()
8
>>> stream.skip_data()
8
>>> stream.seek(12)
12
>>> stream.skip_data()
18
>>> stream.skip_data()
18
>>> stream.seek(20)
20
>>> stream.skip_data()
20
```

skip_hole()

Skips a memory hole.

It moves the current stream position after the end of the currently pointed memory hole (gap).

No action is performed if the current stream position lies within a data block.

Returns

int – Updated stream position.

See also:

seek()

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello'], [12, b'World!']]
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.skip_hole()
3
>>> stream.skip_hole()
3
>>> stream.seek(9)
9
>>> stream.skip_hole()
12
>>> stream.skip_hole()
```

(continues on next page)

(continued from previous page)

```
12
>>> stream.seek(20)
20
>>> stream.skip_hole()
20
```

tell()

Current stream position.

Returns
int – Current stream position.

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello'], [12, b'World!']]
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.tell()
0
>>> stream.skip_hole()
3
>>> stream.tell()
3
>>> stream.read(5)
b'Hello'
>>> stream.tell()
8
>>> stream.skip_hole()
12
>>> stream.read()
b'World!'
>>> stream.tell()
18
>>> stream.seek(20)
20
>>> stream.tell()
20
```

truncate(*size=None*)

Truncates stream.

If *size* is provided, it moves the current stream position to it.

Any data after the updated stream position are deleted from the underlying memory object.

The updated stream position can lie outside the actual memory bounds (i.e. extending after the memory). No filling is performed, only the stream position is moved there.

Parameters
size (*int*) – If not None, the stream is positioned there.

Returns
int – Updated stream position.

Raises
io.UnsupportedOperation – Stream not writable.

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello'], [12, b'World!']]
>>> stream = MemoryIO(Memory.from_blocks(blocks))
>>> stream.seek(7)
7
>>> stream.truncate()
7
>>> stream.tell()
7
>>> stream.memory.to_blocks()
[[3, b'Hell']]
>>> stream.truncate(10)
10
>>> stream.tell()
10
```

```
>>> memory = Memory.from_bytes(b'Hello, World!')
>>> setattr(memory, 'write', None)  # exception on write()
>>> stream = MemoryIO(memory)
>>> stream.seek(7)
7
>>> stream.truncate()
Traceback (most recent call last):
    ...
io.UnsupportedOperation: truncate
```

writable()

Stream is writable.

Returns
bool – Stream is writable.

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> memory = Memory.from_bytes(b'Hello, World!')
>>> with MemoryIO(memory) as stream:
...     print(stream.writable())
True
>>> setattr(memory, 'write', None)  # exception on write()
>>> with MemoryIO(memory) as stream:
...     print(stream.writable())
False
```

write(*buffer*)

Writes data into the stream.

The behaviour depends on the nature of *buffer*: byte-like or integer.

Byte-like data are written into the underlying memory object via its `bytesparse.base.MutableMemory.write()` method, at the current stream position (i.e. *tell()*). The stream position is always incremented by the size of *buffer*, regardless of the actual number of bytes written into the underlying memory object (e.g. when cropped by existing `bytesparse.base.MutableMemory.bounds_span` settings).

If *buffer* is a positive integer, that is the amount of bytes to `bytesparse.base.MutableMemory.clear()` from the current stream position onwards. The stream position is incremented by *buffer* bytes. It returns *buffer* as a positive number.

If *buffer* is a negative integer, that is the amount of bytes to `bytesparse.base.MutableMemory.delete()` from the current stream position onwards. The stream position is not changed. It returns *buffer* as a positive number.

Notes

buffer is considered an integer if the execution of `buffer.__index__()` does not raise an `Exception`.

Parameters
buffer (*bytes*) – Byte data to write at the current stream position.

Returns
int – Size of the written *buffer*.

Raises
io.UnsupportedOperation – Stream not writable.

See also:

`bytesparse.base.MutableMemory.clear()` `bytesparse.base.MutableMemory.delete()` `bytesparse.base.MutableMemory.write()`

Examples

```
>>> from bytesparse import Memory, MemoryIO
```

```
>>> blocks = [[3, b'Hello'], [10, b'World!']]
>>> memory = Memory.from_blocks(blocks)
>>> stream = MemoryIO(memory, seek=10)
>>> stream.write(b'Human')
5
>>> memory.to_blocks()
[[3, b'Hello'], [10, b'Human!']]
>>> stream.tell()
15
>>> stream.seek(7)
7
>>> stream.write(5)  # clear 5 bytes
5
>>> memory.to_blocks()
[[3, b'Hell'], [12, b'man!']]
```

(continues on next page)

(continued from previous page)

```
>>> stream.tell()
12
>>> stream.seek(7)
7
>>> stream.write(-5)  # delete 5 bytes
5
>>> memory.to_blocks()
[[3, b'Hellman!']]
>>> stream.tell()
7
```

```
>>> memory = Memory.from_bytes(b'Hello, World!')
>>> setattr(memory, 'write', None)  # exception on write()
>>> stream = MemoryIO(memory, seek=7)
>>> stream.write(b'Human')
Traceback (most recent call last):
    ...
io.UnsupportedOperation: not writable
```

writelines(*lines*)

Writes lines to the stream.

Line separators are not added, so it is usual for each of the lines provided to have a line separator at the end.

If a *line* is an integer, its behavior is as per *write()* (positive: clear, negative: delete).

Parameters

lines (*list of bytes*) – List of byte strings to write.

See also:

bytesparse.base.MutableMemory.clear() bytesparse.base.MutableMemory.delete() *write()*

Examples

```
>>> from bytesparse import Memory, MemoryIO
>>> lines = [3, b'Hello\n', b'World!', 5, b'Bye\n', 4, b'Bye!']
>>> stream = MemoryIO()
>>> stream.writelines(lines)
>>> stream.memory.to_blocks()
[[3, b'Hello\nWorld!'], [20, b'Bye\n'], [28, b'Bye!']]
```

4.14 xxd

Emulation of the xxd utility.

Attributes

CHAR_ASCII	Mapping from integer to ASCII characters.
CHAR_EBCDIC	Mapping from integer to EBCDIC characters.

4.14.1 CHAR_ASCII

hexrec.xxd.**CHAR_ASCII** = b'.....
!"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\
]^_`abcdefghijklmnopqrstuvwxyz{|}~.....
..... ><'

Mapping from integer to ASCII characters.

4.14.2 CHAR_EBCDIC

hexrec.xxd.**CHAR_EBCDIC** =
b'..... .....<(+|&.....
...!$*);~-/.....,%_>?.....`:#@\'=".abcdefghi.....jklmnopqr^.....stuvwxyz...[.
.....]..{ABCDEFGHI.....}JKLMNOPQR.....\\.STUVWXYZ.....0123456789..... ><'

Mapping from integer to EBCDIC characters.

Functions

parse_seek	Parses the seek option string.
xxd_core	Emulation of the *xxd* utility core.

4.14.3 parse_seek

hexrec.xxd.**parse_seek**(*value*)

Parses the seek option string.

Argument:

value (str):
The value to convert. It is converted to `str` before processing. `None` equals zero.

Returns
tuple – `(sign_string, unsigned_value)`.

4.14.4 xxd_core

hexrec.xxd.**xxd_core**(*infile=None*, *outfile=None*, *autoskip=False*, *bits=None*, *cols=None*, *ebcdic=False*, *endian=False*, *groupsize=None*, *include=False*, *length=None*, *linesep=None*, *offset=None*, *postscript=False*, *quadword=False*, *revert=False*, *oseek=None*, *iseek=None*, *upper_all=False*, *upper=False*, *oseek_zeroes=True*)

Emulation of the *xxd* utility core.

Parameters

- **infile** (*str or bytes*) – Input data. If `str`, it is considered as the input file path. If `bytes`, it is the input byte chunk. If `None`, it reads from the standard input.
- **outfile** (*str or bytes*) – Output data. If `str`, it is considered as the output file path. If `bytes`, it is the output byte chunk. If `None`, it writes to the standard output.
- **autoskip** (*bool*) – Toggles autoskip. A single `'*'` replaces null lines.
- **bits** (*bool*) – Switches to bits (binary digits) dump, rather than hexdump. This option writes octets as eight digits of '1' and '0' instead of a normal hexadecimal dump. Each line is preceded by a line number in hexadecimal and followed by an ASCII (or EBCDIC) representation. The argument switches `revert`, `postscript`, `include` do not work with this mode.
- **cols** (*int*) – Formats `cols` octets per line. Max 256. Defaults: normal 16, `include` 12, `postscript` 30, `bits` 6.
- **ebcdic** (*bool*) – Changes the character encoding in the right-hand column from ASCII to EBCDIC. This does not change the hexadecimal representation. The option is meaningless in combinations with `revert`, `postscript` or `include`.
- **endian** (*bool*) – Switches to little-endian hexdump. This option treats byte groups as words in little-endian byte order. The default grouping of 4 bytes may be changed using `groupsize`. This option only applies to hexdump, leaving the ASCII (or EBCDIC) representation unchanged. The switches `revert`, `postscript`, `include` do not work with this mode.
- **groupsize** (*int*) – Separates the output of every `groupsize` bytes (two hex characters or eight bit-digits each) by a whitespace. Specify `groupsize` 0 to suppress grouping. `groupsize` defaults to 2 in normal mode, 4 in little-endian mode and 1 in bits mode. Grouping does not apply to `postscript` or `include`.
- **include** (*bool*) – Output in C include file style. A complete static array definition is written (named after the input file), unless reading from standard input.
- **length** (*int*) – Stops after writing `length` octets.
- **linesep** (*bytes*) – Line separator characters. If `None`, it defaults to `os.linesep.encode()`.
- **offset** (*int*) – Adds `offset` to the displayed file position.
- **postscript** (*bool*) – Outputs in postscript continuous hexdump style. Also known as plain hexdump style.
- **quadword** (*bool*) – Uses 64-bit addressing.
- **revert** (*bool*) – Reverse operation: convert (or patch) hexdump into binary. If not writing to standard output, it writes into its output file without truncating it. Use the combination `revert` and `postscript` to read plain hexadecimal dumps without line number information and without a particular column layout. Additional Whitespace and line breaks are allowed anywhere.

- **oseek** (*int*) – When used after `revert` reverts with `offset` added to file positions found in hexdump.
- **iseek** (*int or str*) – Starts at `iseej` bytes absolute (or relative) input offset. Without `iseek` option, it starts at the current file position. The prefix is used to compute the offset. `+` indicates that the seek is relative to the current input position. `-` indicates that the seek should be that many characters from the end of the input. `+-` indicates that the seek should be that many characters before the current stdin file position.
- **upper_all** (*bool*) – Uses upper case hex letters on address and data.
- **upper** (*bool*) – Uses upper case hex letters on data only.
- **oseek_zeroes** (*bool*) – Output seeking fills with zeros. Only affects *outfile* of `bytesparse.base.MutableMemory`.

Returns

stream – The handle to the output stream.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

hexrec could always use more documentation, whether as part of the official hexrec docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/TexZK/hexrec/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *hexrec* for local development:

1. Fork hexrec (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/hexrec.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`).
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

CHAPTER

SIX

AUTHORS

- Andrea Zoppi - main developer - https://github.com/TexZK

6.1 Special thanks

- Scott Finneran - main developer of the SRecord project - https://srecord.sourceforge.net/

CHAPTER

SEVEN

CHANGELOG

7.1 0.4.0 (TBD)

- Library rewritten from scratch (not backwards compatible).
- Added new object oriented API, hopefully more user friendly.
- Added *Texas Instruments TI-TXT* file format.
- Improved docs and examples.

7.2 0.3.1 (2024-01-23)

- Added support for Python 3.12.
- Added Motorola header editing.
- Minor fixes and changes.

7.3 0.3.0 (2023-02-21)

- Added support for Python 3.11, removed 3.6.
- Deprecated `hexrec.blocks` module entirely.
- Using `bytesparse` for virtual memory management.
- Improved repository layout.
- Improved testing and packaging workflow.
- Minor fixes and changes.

7.4 0.2.3 (2021-12-30)

- Removed dependency of legacy pathlib package; using Python's own module instead.
- Added support for Python 3.10.
- Fixed maximum SREC length.
- Changed pattern offset behavior.
- Some alignment to the `bytesparse.Memory` API; deprecated code marked as such.

7.5 0.2.2 (2020-11-08)

- Added workaround to register default record types.
- Added support for Python 3.9.
- Fixed insertion bug.
- Added empty space reservation.

7.6 0.2.1 (2020-03-05)

- Fixed flood with empty span.

7.7 0.2.0 (2020-02-01)

- Added support for current Python versions (3.8, PyPy 3).
- Removed support for old Python versions (< 3.6, PyPy 2).
- Major refactoring to allow an easier integration of new record formats.

7.8 0.1.0 (2019-08-13)

- Added support for Python 3.7.

7.9 0.0.4 (2018-12-22)

- New command line interface made with Click.
- More testing and fixing.
- Some refactoring.
- More documentation.

7.10 0.0.3 (2018-12-04)

- Much testing and fixing.
- Some refactoring.
- More documentation.

7.11 0.0.2 (2018-08-29)

- Major refactoring.
- Added most of the documentation.
- Added first drafts to manage blocks of data.
- Added first test suites.

7.12 0.0.1 (2018-06-27)

- First release on PyPI.
- Added first drafts to manage record files.
- Added first emulation of xxd.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

hexrec.base, 25
hexrec.formats, 79
hexrec.formats.asciihex, 80
hexrec.formats.avr, 132
hexrec.formats.ihex, 178
hexrec.formats.mos, 234
hexrec.formats.raw, 285
hexrec.formats.sqtp, 331
hexrec.formats.srec, 339
hexrec.formats.titxt, 399
hexrec.formats.xtek, 451
hexrec.hexdump, 505
hexrec.utils, 509
hexrec.xxd, 534

INDEX

Symbols

_DATA (*hexrec.base.BaseTag attribute*), 78
_DATA (*hexrec.formats.asciihex.AsciiHexTag attribute*), 126
_DATA (*hexrec.formats.avr.AvrTag attribute*), 177
_DATA (*hexrec.formats.ihex.IhexTag attribute*), 228
_DATA (*hexrec.formats.mos.MosTag attribute*), 280
_DATA (*hexrec.formats.raw.RawTag attribute*), 330
_DATA (*hexrec.formats.srec.SrecTag attribute*), 389
_DATA (*hexrec.formats.titxt.TiTxtTag attribute*), 445
_DATA (*hexrec.formats.xtek.XtekTag attribute*), 500
__abs__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__abs__() (*hexrec.formats.ihex.IhexTag method*), 228
__abs__() (*hexrec.formats.mos.MosTag method*), 280
__abs__() (*hexrec.formats.srec.SrecTag method*), 390
__abs__() (*hexrec.formats.titxt.TiTxtTag method*), 445
__abs__() (*hexrec.formats.xtek.XtekTag method*), 500
__add__() (*hexrec.base.BaseFile method*), 33
__add__() (*hexrec.formats.asciihex.AsciiHexFile method*), 81
__add__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__add__() (*hexrec.formats.avr.AvrFile method*), 134
__add__() (*hexrec.formats.ihex.IhexFile method*), 180
__add__() (*hexrec.formats.ihex.IhexTag method*), 228
__add__() (*hexrec.formats.mos.MosFile method*), 235
__add__() (*hexrec.formats.mos.MosTag method*), 280
__add__() (*hexrec.formats.raw.RawFile method*), 286
__add__() (*hexrec.formats.srec.SrecFile method*), 341
__add__() (*hexrec.formats.srec.SrecTag method*), 390
__add__() (*hexrec.formats.titxt.TiTxtFile method*), 400
__add__() (*hexrec.formats.titxt.TiTxtTag method*), 445
__add__() (*hexrec.formats.xtek.XtekFile method*), 452
__add__() (*hexrec.formats.xtek.XtekTag method*), 500
__and__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__and__() (*hexrec.formats.ihex.IhexTag method*), 228
__and__() (*hexrec.formats.mos.MosTag method*), 280
__and__() (*hexrec.formats.srec.SrecTag method*), 390
__and__() (*hexrec.formats.titxt.TiTxtTag method*), 445
__and__() (*hexrec.formats.xtek.XtekTag method*), 500
__bool__() (*hexrec.base.BaseFile method*), 34
__bool__() (*hexrec.formats.asciihex.AsciiHexFile method*), 82
__bool__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__bool__() (*hexrec.formats.avr.AvrFile method*), 134
__bool__() (*hexrec.formats.ihex.IhexFile method*), 180
__bool__() (*hexrec.formats.ihex.IhexTag method*), 228
__bool__() (*hexrec.formats.mos.MosFile method*), 236
__bool__() (*hexrec.formats.mos.MosTag method*), 280
__bool__() (*hexrec.formats.raw.RawFile method*), 287
__bool__() (*hexrec.formats.srec.SrecFile method*), 341
__bool__() (*hexrec.formats.srec.SrecTag method*), 390
__bool__() (*hexrec.formats.titxt.TiTxtFile method*), 401
__bool__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__bool__() (*hexrec.formats.xtek.XtekFile method*), 453
__bool__() (*hexrec.formats.xtek.XtekTag method*), 500
__bytes__() (*hexrec.base.BaseRecord method*), 67
__bytes__() (*hexrec.formats.asciihex.AsciiHexRecord method*), 115
__bytes__() (*hexrec.formats.avr.AvrRecord method*), 167
__bytes__() (*hexrec.formats.ihex.IhexRecord method*), 215
__bytes__() (*hexrec.formats.mos.MosRecord method*), 269
__bytes__() (*hexrec.formats.raw.RawRecord method*), 319
__bytes__() (*hexrec.formats.srec.SrecRecord method*), 376
__bytes__() (*hexrec.formats.titxt.TiTxtRecord method*), 434
__bytes__() (*hexrec.formats.xtek.XtekRecord method*), 487
__ceil__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__ceil__() (*hexrec.formats.ihex.IhexTag method*), 228
__ceil__() (*hexrec.formats.mos.MosTag method*), 280
__ceil__() (*hexrec.formats.srec.SrecTag method*), 390
__ceil__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__ceil__() (*hexrec.formats.xtek.XtekTag method*), 500
__contains__() (*hexrec.formats.asciihex.AsciiHexTag*

class method), 127
__contains__() (*hexrec.formats.avr.AvrTag class method*), 177
__contains__() (*hexrec.formats.ihex.IhexTag class method*), 228
__contains__() (*hexrec.formats.mos.MosTag class method*), 280
__contains__() (*hexrec.formats.raw.RawTag class method*), 330
__contains__() (*hexrec.formats.srec.SrecTag class method*), 390
__contains__() (*hexrec.formats.titxt.TiTxtTag class method*), 446
__contains__() (*hexrec.formats.xtek.XtekTag class method*), 501
__del__() (*hexrec.utils.SparseMemoryIO method*), 513
__delitem__() (*hexrec.base.BaseFile method*), 34
__delitem__() (*hexrec.formats.asciihex.AsciiHexFile method*), 82
__delitem__() (*hexrec.formats.avr.AvrFile method*), 135
__delitem__() (*hexrec.formats.ihex.IhexFile method*), 181
__delitem__() (*hexrec.formats.mos.MosFile method*), 236
__delitem__() (*hexrec.formats.raw.RawFile method*), 287
__delitem__() (*hexrec.formats.srec.SrecFile method*), 342
__delitem__() (*hexrec.formats.titxt.TiTxtFile method*), 401
__delitem__() (*hexrec.formats.xtek.XtekFile method*), 453
__dir__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__dir__() (*hexrec.formats.ihex.IhexTag method*), 228
__dir__() (*hexrec.formats.mos.MosTag method*), 280
__dir__() (*hexrec.formats.srec.SrecTag method*), 390
__dir__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__dir__() (*hexrec.formats.xtek.XtekTag method*), 501
__divmod__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__divmod__() (*hexrec.formats.ihex.IhexTag method*), 228
__divmod__() (*hexrec.formats.mos.MosTag method*), 280
__divmod__() (*hexrec.formats.srec.SrecTag method*), 390
__divmod__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__divmod__() (*hexrec.formats.xtek.XtekTag method*), 501
__enter__() (*hexrec.utils.SparseMemoryIO method*), 513
__eq__() (*hexrec.base.BaseFile method*), 34
__eq__() (*hexrec.base.BaseRecord method*), 68
__eq__() (*hexrec.formats.asciihex.AsciiHexFile method*), 83
__eq__() (*hexrec.formats.asciihex.AsciiHexRecord method*), 115
__eq__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__eq__() (*hexrec.formats.avr.AvrFile method*), 135
__eq__() (*hexrec.formats.avr.AvrRecord method*), 168
__eq__() (*hexrec.formats.ihex.IhexFile method*), 181
__eq__() (*hexrec.formats.ihex.IhexRecord method*), 215
__eq__() (*hexrec.formats.ihex.IhexTag method*), 228
__eq__() (*hexrec.formats.mos.MosFile method*), 237
__eq__() (*hexrec.formats.mos.MosRecord method*), 269
__eq__() (*hexrec.formats.mos.MosTag method*), 280
__eq__() (*hexrec.formats.raw.RawFile method*), 288
__eq__() (*hexrec.formats.raw.RawRecord method*), 320
__eq__() (*hexrec.formats.srec.SrecFile method*), 342
__eq__() (*hexrec.formats.srec.SrecRecord method*), 377
__eq__() (*hexrec.formats.srec.SrecTag method*), 390
__eq__() (*hexrec.formats.titxt.TiTxtFile method*), 402
__eq__() (*hexrec.formats.titxt.TiTxtRecord method*), 434
__eq__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__eq__() (*hexrec.formats.xtek.XtekFile method*), 454
__eq__() (*hexrec.formats.xtek.XtekRecord method*), 488
__eq__() (*hexrec.formats.xtek.XtekTag method*), 501
__exit__() (*hexrec.utils.SparseMemoryIO method*), 513
__float__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__float__() (*hexrec.formats.ihex.IhexTag method*), 228
__float__() (*hexrec.formats.mos.MosTag method*), 280
__float__() (*hexrec.formats.srec.SrecTag method*), 390
__float__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__float__() (*hexrec.formats.xtek.XtekTag method*), 501
__floor__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__floor__() (*hexrec.formats.ihex.IhexTag method*), 228
__floor__() (*hexrec.formats.mos.MosTag method*), 280
__floor__() (*hexrec.formats.srec.SrecTag method*), 390
__floor__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__floor__() (*hexrec.formats.xtek.XtekTag method*), 501
__floordiv__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__floordiv__() (*hexrec.formats.ihex.IhexTag method*), 228
__floordiv__() (*hexrec.formats.mos.MosTag method*), 280
__floordiv__() (*hexrec.formats.srec.SrecTag method*), 390

__floordiv__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__floordiv__() (*hexrec.formats.xtek.XtekTag method*), 501
__format__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__format__() (*hexrec.formats.ihex.IhexTag method*), 228
__format__() (*hexrec.formats.mos.MosTag method*), 280
__format__() (*hexrec.formats.srec.SrecTag method*), 390
__format__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__format__() (*hexrec.formats.xtek.XtekTag method*), 501
__ge__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__ge__() (*hexrec.formats.ihex.IhexTag method*), 228
__ge__() (*hexrec.formats.mos.MosTag method*), 280
__ge__() (*hexrec.formats.srec.SrecTag method*), 390
__ge__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__ge__() (*hexrec.formats.xtek.XtekTag method*), 501
__getattribute__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__getattribute__() (*hexrec.formats.ihex.IhexTag method*), 228
__getattribute__() (*hexrec.formats.mos.MosTag method*), 280
__getattribute__() (*hexrec.formats.srec.SrecTag method*), 390
__getattribute__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__getattribute__() (*hexrec.formats.xtek.XtekTag method*), 501
__getitem__() (*hexrec.base.BaseFile method*), 35
__getitem__() (*hexrec.formats.asciihex.AsciiHexFile method*), 84
__getitem__() (*hexrec.formats.asciihex.AsciiHexTag class method*), 127
__getitem__() (*hexrec.formats.avr.AvrFile method*), 136
__getitem__() (*hexrec.formats.avr.AvrTag class method*), 177
__getitem__() (*hexrec.formats.ihex.IhexFile method*), 182
__getitem__() (*hexrec.formats.ihex.IhexTag class method*), 228
__getitem__() (*hexrec.formats.mos.MosFile method*), 237
__getitem__() (*hexrec.formats.mos.MosTag class method*), 280
__getitem__() (*hexrec.formats.raw.RawFile method*), 289
__getitem__() (*hexrec.formats.raw.RawTag class method*), 330
__getitem__() (*hexrec.formats.srec.SrecFile method*), 343
__getitem__() (*hexrec.formats.srec.SrecTag class method*), 390
__getitem__() (*hexrec.formats.titxt.TiTxtFile method*), 403
__getitem__() (*hexrec.formats.titxt.TiTxtTag class method*), 446
__getitem__() (*hexrec.formats.xtek.XtekFile method*), 455
__getitem__() (*hexrec.formats.xtek.XtekTag class method*), 501
__gt__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__gt__() (*hexrec.formats.ihex.IhexTag method*), 228
__gt__() (*hexrec.formats.mos.MosTag method*), 281
__gt__() (*hexrec.formats.srec.SrecTag method*), 390
__gt__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__gt__() (*hexrec.formats.xtek.XtekTag method*), 501
__hash__ (*hexrec.base.BaseFile attribute*), 36
__hash__ (*hexrec.base.BaseRecord attribute*), 69
__hash__ (*hexrec.formats.asciihex.AsciiHexFile attribute*), 84
__hash__ (*hexrec.formats.asciihex.AsciiHexRecord attribute*), 116
__hash__ (*hexrec.formats.avr.AvrFile attribute*), 137
__hash__ (*hexrec.formats.avr.AvrRecord attribute*), 168
__hash__ (*hexrec.formats.ihex.IhexFile attribute*), 182
__hash__ (*hexrec.formats.ihex.IhexRecord attribute*), 216
__hash__ (*hexrec.formats.mos.MosFile attribute*), 238
__hash__ (*hexrec.formats.mos.MosRecord attribute*), 270
__hash__ (*hexrec.formats.raw.RawFile attribute*), 289
__hash__ (*hexrec.formats.raw.RawRecord attribute*), 321
__hash__ (*hexrec.formats.srec.SrecFile attribute*), 344
__hash__ (*hexrec.formats.srec.SrecRecord attribute*), 377
__hash__ (*hexrec.formats.titxt.TiTxtFile attribute*), 403
__hash__ (*hexrec.formats.titxt.TiTxtRecord attribute*), 435
__hash__ (*hexrec.formats.xtek.XtekFile attribute*), 455
__hash__ (*hexrec.formats.xtek.XtekRecord attribute*), 488
__hash__() (*hexrec.formats.asciihex.AsciiHexTag method*), 127
__hash__() (*hexrec.formats.ihex.IhexTag method*), 229
__hash__() (*hexrec.formats.mos.MosTag method*), 281
__hash__() (*hexrec.formats.srec.SrecTag method*), 390
__hash__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__hash__() (*hexrec.formats.xtek.XtekTag method*), 501

__iadd__() (*hexrec.base.BaseFile method*), 36
__iadd__() (*hexrec.formats.asciihex.AsciiHexFile method*), 84
__iadd__() (*hexrec.formats.avr.AvrFile method*), 137
__iadd__() (*hexrec.formats.ihex.IhexFile method*), 182
__iadd__() (*hexrec.formats.mos.MosFile method*), 238
__iadd__() (*hexrec.formats.raw.RawFile method*), 289
__iadd__() (*hexrec.formats.srec.SrecFile method*), 344
__iadd__() (*hexrec.formats.titxt.TiTxtFile method*), 403
__iadd__() (*hexrec.formats.xtek.XtekFile method*), 455
__index__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__index__() (*hexrec.formats.ihex.IhexTag method*), 229
__index__() (*hexrec.formats.mos.MosTag method*), 281
__index__() (*hexrec.formats.srec.SrecTag method*), 391
__index__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__index__() (*hexrec.formats.xtek.XtekTag method*), 501
__init__() (*hexrec.base.BaseFile method*), 36
__init__() (*hexrec.base.BaseRecord method*), 69
__init__() (*hexrec.formats.asciihex.AsciiHexFile method*), 85
__init__() (*hexrec.formats.asciihex.AsciiHexRecord method*), 116
__init__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__init__() (*hexrec.formats.avr.AvrFile method*), 137
__init__() (*hexrec.formats.avr.AvrRecord method*), 168
__init__() (*hexrec.formats.ihex.IhexFile method*), 183
__init__() (*hexrec.formats.ihex.IhexRecord method*), 216
__init__() (*hexrec.formats.ihex.IhexTag method*), 229
__init__() (*hexrec.formats.mos.MosFile method*), 239
__init__() (*hexrec.formats.mos.MosRecord method*), 270
__init__() (*hexrec.formats.mos.MosTag method*), 281
__init__() (*hexrec.formats.raw.RawFile method*), 290
__init__() (*hexrec.formats.raw.RawRecord method*), 321
__init__() (*hexrec.formats.srec.SrecFile method*), 344
__init__() (*hexrec.formats.srec.SrecRecord method*), 378
__init__() (*hexrec.formats.srec.SrecTag method*), 391
__init__() (*hexrec.formats.titxt.TiTxtFile method*), 404
__init__() (*hexrec.formats.titxt.TiTxtRecord method*), 435
__init__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__init__() (*hexrec.formats.xtek.XtekFile method*), 456
__init__() (*hexrec.formats.xtek.XtekRecord method*), 488
__init__() (*hexrec.formats.xtek.XtekTag method*), 501
__init__() (*hexrec.utils.SparseMemoryIO method*), 514
__int__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__int__() (*hexrec.formats.ihex.IhexTag method*), 229
__int__() (*hexrec.formats.mos.MosTag method*), 281
__int__() (*hexrec.formats.srec.SrecTag method*), 391
__int__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__int__() (*hexrec.formats.xtek.XtekTag method*), 501
__invert__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__invert__() (*hexrec.formats.ihex.IhexTag method*), 229
__invert__() (*hexrec.formats.mos.MosTag method*), 281
__invert__() (*hexrec.formats.srec.SrecTag method*), 391
__invert__() (*hexrec.formats.titxt.TiTxtTag method*), 446
__invert__() (*hexrec.formats.xtek.XtekTag method*), 501
__ior__() (*hexrec.base.BaseFile method*), 36
__ior__() (*hexrec.formats.asciihex.AsciiHexFile method*), 85
__ior__() (*hexrec.formats.avr.AvrFile method*), 137
__ior__() (*hexrec.formats.ihex.IhexFile method*), 183
__ior__() (*hexrec.formats.mos.MosFile method*), 239
__ior__() (*hexrec.formats.raw.RawFile method*), 290
__ior__() (*hexrec.formats.srec.SrecFile method*), 344
__ior__() (*hexrec.formats.titxt.TiTxtFile method*), 404
__ior__() (*hexrec.formats.xtek.XtekFile method*), 456
__iter__() (*hexrec.formats.asciihex.AsciiHexTag class method*), 128
__iter__() (*hexrec.formats.avr.AvrTag class method*), 177
__iter__() (*hexrec.formats.ihex.IhexTag class method*), 229
__iter__() (*hexrec.formats.mos.MosTag class method*), 281
__iter__() (*hexrec.formats.raw.RawTag class method*), 330
__iter__() (*hexrec.formats.srec.SrecTag class method*), 391
__iter__() (*hexrec.formats.titxt.TiTxtTag class method*), 447
__iter__() (*hexrec.formats.xtek.XtekTag class method*), 501
__iter__() (*hexrec.utils.SparseMemoryIO method*), 514
__le__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__le__() (*hexrec.formats.ihex.IhexTag method*), 229
__le__() (*hexrec.formats.mos.MosTag method*), 281
__le__() (*hexrec.formats.srec.SrecTag method*), 391
__le__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__le__() (*hexrec.formats.xtek.XtekTag method*), 501

__len__() (*hexrec.formats.asciihex.AsciiHexTag class method*), 128
__len__() (*hexrec.formats.avr.AvrTag class method*), 177
__len__() (*hexrec.formats.ihex.IhexTag class method*), 229
__len__() (*hexrec.formats.mos.MosTag class method*), 281
__len__() (*hexrec.formats.raw.RawTag class method*), 330
__len__() (*hexrec.formats.srec.SrecTag class method*), 391
__len__() (*hexrec.formats.titxt.TiTxtTag class method*), 447
__len__() (*hexrec.formats.xtek.XtekTag class method*), 502
__lshift__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__lshift__() (*hexrec.formats.ihex.IhexTag method*), 229
__lshift__() (*hexrec.formats.mos.MosTag method*), 281
__lshift__() (*hexrec.formats.srec.SrecTag method*), 391
__lshift__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__lshift__() (*hexrec.formats.xtek.XtekTag method*), 502
__lt__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__lt__() (*hexrec.formats.ihex.IhexTag method*), 229
__lt__() (*hexrec.formats.mos.MosTag method*), 281
__lt__() (*hexrec.formats.srec.SrecTag method*), 391
__lt__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__lt__() (*hexrec.formats.xtek.XtekTag method*), 502
__mod__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__mod__() (*hexrec.formats.ihex.IhexTag method*), 229
__mod__() (*hexrec.formats.mos.MosTag method*), 281
__mod__() (*hexrec.formats.srec.SrecTag method*), 391
__mod__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__mod__() (*hexrec.formats.xtek.XtekTag method*), 502
__mul__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__mul__() (*hexrec.formats.ihex.IhexTag method*), 229
__mul__() (*hexrec.formats.mos.MosTag method*), 281
__mul__() (*hexrec.formats.srec.SrecTag method*), 391
__mul__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__mul__() (*hexrec.formats.xtek.XtekTag method*), 502
__ne__() (*hexrec.base.BaseFile method*), 37
__ne__() (*hexrec.base.BaseRecord method*), 69
__ne__() (*hexrec.formats.asciihex.AsciiHexFile method*), 85
__ne__() (*hexrec.formats.asciihex.AsciiHexRecord method*), 116
__ne__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__ne__() (*hexrec.formats.avr.AvrFile method*), 138
__ne__() (*hexrec.formats.avr.AvrRecord method*), 168
__ne__() (*hexrec.formats.ihex.IhexFile method*), 184
__ne__() (*hexrec.formats.ihex.IhexRecord method*), 216
__ne__() (*hexrec.formats.ihex.IhexTag method*), 229
__ne__() (*hexrec.formats.mos.MosFile method*), 239
__ne__() (*hexrec.formats.mos.MosRecord method*), 270
__ne__() (*hexrec.formats.mos.MosTag method*), 281
__ne__() (*hexrec.formats.raw.RawFile method*), 290
__ne__() (*hexrec.formats.raw.RawRecord method*), 321
__ne__() (*hexrec.formats.srec.SrecFile method*), 345
__ne__() (*hexrec.formats.srec.SrecRecord method*), 378
__ne__() (*hexrec.formats.srec.SrecTag method*), 391
__ne__() (*hexrec.formats.titxt.TiTxtFile method*), 404
__ne__() (*hexrec.formats.titxt.TiTxtRecord method*), 435
__ne__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__ne__() (*hexrec.formats.xtek.XtekFile method*), 456
__ne__() (*hexrec.formats.xtek.XtekRecord method*), 488
__ne__() (*hexrec.formats.xtek.XtekTag method*), 502
__neg__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__neg__() (*hexrec.formats.ihex.IhexTag method*), 229
__neg__() (*hexrec.formats.mos.MosTag method*), 281
__neg__() (*hexrec.formats.srec.SrecTag method*), 391
__neg__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__neg__() (*hexrec.formats.xtek.XtekTag method*), 502
__new__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__new__() (*hexrec.formats.avr.AvrTag method*), 177
__new__() (*hexrec.formats.ihex.IhexTag method*), 229
__new__() (*hexrec.formats.mos.MosTag method*), 281
__new__() (*hexrec.formats.raw.RawTag method*), 330
__new__() (*hexrec.formats.srec.SrecTag method*), 391
__new__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__new__() (*hexrec.formats.xtek.XtekTag method*), 502
__new__() (*hexrec.utils.SparseMemoryIO method*), 514
__next__() (*hexrec.utils.SparseMemoryIO method*), 514
__or__() (*hexrec.base.BaseFile method*), 38
__or__() (*hexrec.formats.asciihex.AsciiHexFile method*), 86
__or__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__or__() (*hexrec.formats.avr.AvrFile method*), 139
__or__() (*hexrec.formats.ihex.IhexFile method*), 185
__or__() (*hexrec.formats.ihex.IhexTag method*), 229
__or__() (*hexrec.formats.mos.MosFile method*), 240
__or__() (*hexrec.formats.mos.MosTag method*), 281
__or__() (*hexrec.formats.raw.RawFile method*), 291
__or__() (*hexrec.formats.srec.SrecFile method*), 346

__or__() (*hexrec.formats.srec.SrecTag method*), 391
__or__() (*hexrec.formats.titxt.TiTxtFile method*), 405
__or__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__or__() (*hexrec.formats.xtek.XtekFile method*), 457
__or__() (*hexrec.formats.xtek.XtekTag method*), 502
__pos__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__pos__() (*hexrec.formats.ihex.IhexTag method*), 229
__pos__() (*hexrec.formats.mos.MosTag method*), 281
__pos__() (*hexrec.formats.srec.SrecTag method*), 391
__pos__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__pos__() (*hexrec.formats.xtek.XtekTag method*), 502
__pow__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__pow__() (*hexrec.formats.ihex.IhexTag method*), 229
__pow__() (*hexrec.formats.mos.MosTag method*), 281
__pow__() (*hexrec.formats.srec.SrecTag method*), 391
__pow__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__pow__() (*hexrec.formats.xtek.XtekTag method*), 502
__radd__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__radd__() (*hexrec.formats.ihex.IhexTag method*), 229
__radd__() (*hexrec.formats.mos.MosTag method*), 281
__radd__() (*hexrec.formats.srec.SrecTag method*), 391
__radd__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__radd__() (*hexrec.formats.xtek.XtekTag method*), 502
__rand__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__rand__() (*hexrec.formats.ihex.IhexTag method*), 229
__rand__() (*hexrec.formats.mos.MosTag method*), 282
__rand__() (*hexrec.formats.srec.SrecTag method*), 391
__rand__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__rand__() (*hexrec.formats.xtek.XtekTag method*), 502
__rdivmod__() (*hexrec.formats.asciihex.AsciiHexTag method*), 128
__rdivmod__() (*hexrec.formats.ihex.IhexTag method*), 230
__rdivmod__() (*hexrec.formats.mos.MosTag method*), 282
__rdivmod__() (*hexrec.formats.srec.SrecTag method*), 391
__rdivmod__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__rdivmod__() (*hexrec.formats.xtek.XtekTag method*), 502
__reduce_ex__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__reduce_ex__() (*hexrec.formats.ihex.IhexTag method*), 230
__reduce_ex__() (*hexrec.formats.mos.MosTag method*), 282
__reduce_ex__() (*hexrec.formats.srec.SrecTag method*), 392
__reduce_ex__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__reduce_ex__() (*hexrec.formats.xtek.XtekTag method*), 502
__repr__() (*hexrec.base.BaseRecord method*), 69
__repr__() (*hexrec.formats.asciihex.AsciiHexRecord method*), 117
__repr__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__repr__() (*hexrec.formats.avr.AvrRecord method*), 169
__repr__() (*hexrec.formats.ihex.IhexRecord method*), 217
__repr__() (*hexrec.formats.ihex.IhexTag method*), 230
__repr__() (*hexrec.formats.mos.MosRecord method*), 270
__repr__() (*hexrec.formats.mos.MosTag method*), 282
__repr__() (*hexrec.formats.raw.RawRecord method*), 321
__repr__() (*hexrec.formats.srec.SrecRecord method*), 378
__repr__() (*hexrec.formats.srec.SrecTag method*), 392
__repr__() (*hexrec.formats.titxt.TiTxtRecord method*), 436
__repr__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__repr__() (*hexrec.formats.xtek.XtekRecord method*), 489
__repr__() (*hexrec.formats.xtek.XtekTag method*), 502
__rfloordiv__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__rfloordiv__() (*hexrec.formats.ihex.IhexTag method*), 230
__rfloordiv__() (*hexrec.formats.mos.MosTag method*), 282
__rfloordiv__() (*hexrec.formats.srec.SrecTag method*), 392
__rfloordiv__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__rfloordiv__() (*hexrec.formats.xtek.XtekTag method*), 502
__rlshift__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__rlshift__() (*hexrec.formats.ihex.IhexTag method*), 230
__rlshift__() (*hexrec.formats.mos.MosTag method*), 282
__rlshift__() (*hexrec.formats.srec.SrecTag method*), 392
__rlshift__() (*hexrec.formats.titxt.TiTxtTag method*), 447
__rlshift__() (*hexrec.formats.xtek.XtekTag method*), 502
__rmod__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129

__rmod__() (*hexrec.formats.ihex.IhexTag method*), 230
__rmod__() (*hexrec.formats.mos.MosTag method*), 282
__rmod__() (*hexrec.formats.srec.SrecTag method*), 392
__rmod__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__rmod__() (*hexrec.formats.xtek.XtekTag method*), 502
__rmul__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__rmul__() (*hexrec.formats.ihex.IhexTag method*), 230
__rmul__() (*hexrec.formats.mos.MosTag method*), 282
__rmul__() (*hexrec.formats.srec.SrecTag method*), 392
__rmul__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__rmul__() (*hexrec.formats.xtek.XtekTag method*), 502
__ror__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__ror__() (*hexrec.formats.ihex.IhexTag method*), 230
__ror__() (*hexrec.formats.mos.MosTag method*), 282
__ror__() (*hexrec.formats.srec.SrecTag method*), 392
__ror__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__ror__() (*hexrec.formats.xtek.XtekTag method*), 503
__round__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__round__() (*hexrec.formats.ihex.IhexTag method*), 230
__round__() (*hexrec.formats.mos.MosTag method*), 282
__round__() (*hexrec.formats.srec.SrecTag method*), 392
__round__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__round__() (*hexrec.formats.xtek.XtekTag method*), 503
__rpow__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__rpow__() (*hexrec.formats.ihex.IhexTag method*), 230
__rpow__() (*hexrec.formats.mos.MosTag method*), 282
__rpow__() (*hexrec.formats.srec.SrecTag method*), 392
__rpow__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__rpow__() (*hexrec.formats.xtek.XtekTag method*), 503
__rrshift__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__rrshift__() (*hexrec.formats.ihex.IhexTag method*), 230
__rrshift__() (*hexrec.formats.mos.MosTag method*), 282
__rrshift__() (*hexrec.formats.srec.SrecTag method*), 392
__rrshift__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__rrshift__() (*hexrec.formats.xtek.XtekTag method*), 503
__rshift__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__rshift__() (*hexrec.formats.ihex.IhexTag method*), 230
__rshift__() (*hexrec.formats.mos.MosTag method*), 282
__rshift__() (*hexrec.formats.srec.SrecTag method*), 392
__rshift__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__rshift__() (*hexrec.formats.xtek.XtekTag method*), 503
__rsub__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__rsub__() (*hexrec.formats.ihex.IhexTag method*), 230
__rsub__() (*hexrec.formats.mos.MosTag method*), 282
__rsub__() (*hexrec.formats.srec.SrecTag method*), 392
__rsub__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__rsub__() (*hexrec.formats.xtek.XtekTag method*), 503
__rtruediv__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__rtruediv__() (*hexrec.formats.ihex.IhexTag method*), 230
__rtruediv__() (*hexrec.formats.mos.MosTag method*), 282
__rtruediv__() (*hexrec.formats.srec.SrecTag method*), 392
__rtruediv__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__rtruediv__() (*hexrec.formats.xtek.XtekTag method*), 503
__rxor__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__rxor__() (*hexrec.formats.ihex.IhexTag method*), 230
__rxor__() (*hexrec.formats.mos.MosTag method*), 282
__rxor__() (*hexrec.formats.srec.SrecTag method*), 392
__rxor__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__rxor__() (*hexrec.formats.xtek.XtekTag method*), 503
__setitem__() (*hexrec.base.BaseFile method*), 39
__setitem__() (*hexrec.formats.asciihex.AsciiHexFile method*), 87
__setitem__() (*hexrec.formats.avr.AvrFile method*), 139
__setitem__() (*hexrec.formats.ihex.IhexFile method*), 185
__setitem__() (*hexrec.formats.mos.MosFile method*), 241
__setitem__() (*hexrec.formats.raw.RawFile method*), 292
__setitem__() (*hexrec.formats.srec.SrecFile method*), 346
__setitem__() (*hexrec.formats.titxt.TiTxtFile method*), 406
__setitem__() (*hexrec.formats.xtek.XtekFile method*), 458
__sizeof__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__sizeof__() (*hexrec.formats.ihex.IhexTag method*), 230
__sizeof__() (*hexrec.formats.mos.MosTag method*), 282
__sizeof__() (*hexrec.formats.srec.SrecTag method*),

392
__sizeof__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__sizeof__() (*hexrec.formats.xtek.XtekTag method*), 503
__str__() (*hexrec.base.BaseRecord method*), 70
__str__() (*hexrec.formats.asciihex.AsciiHexRecord method*), 117
__str__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__str__() (*hexrec.formats.avr.AvrRecord method*), 169
__str__() (*hexrec.formats.ihex.IhexRecord method*), 217
__str__() (*hexrec.formats.ihex.IhexTag method*), 230
__str__() (*hexrec.formats.mos.MosRecord method*), 271
__str__() (*hexrec.formats.mos.MosTag method*), 282
__str__() (*hexrec.formats.raw.RawRecord method*), 322
__str__() (*hexrec.formats.srec.SrecRecord method*), 379
__str__() (*hexrec.formats.srec.SrecTag method*), 392
__str__() (*hexrec.formats.titxt.TiTxtRecord method*), 436
__str__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__str__() (*hexrec.formats.xtek.XtekRecord method*), 489
__str__() (*hexrec.formats.xtek.XtekTag method*), 503
__sub__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__sub__() (*hexrec.formats.ihex.IhexTag method*), 230
__sub__() (*hexrec.formats.mos.MosTag method*), 282
__sub__() (*hexrec.formats.srec.SrecTag method*), 392
__sub__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__sub__() (*hexrec.formats.xtek.XtekTag method*), 503
__truediv__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__truediv__() (*hexrec.formats.ihex.IhexTag method*), 230
__truediv__() (*hexrec.formats.mos.MosTag method*), 283
__truediv__() (*hexrec.formats.srec.SrecTag method*), 392
__truediv__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__truediv__() (*hexrec.formats.xtek.XtekTag method*), 503
__trunc__() (*hexrec.formats.asciihex.AsciiHexTag method*), 129
__trunc__() (*hexrec.formats.ihex.IhexTag method*), 231
__trunc__() (*hexrec.formats.mos.MosTag method*), 283
__trunc__() (*hexrec.formats.srec.SrecTag method*), 392
__trunc__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__trunc__() (*hexrec.formats.xtek.XtekTag method*), 503
__weakref__ (*hexrec.base.BaseFile attribute*), 39
__weakref__ (*hexrec.base.BaseRecord attribute*), 70
__weakref__ (*hexrec.base.BaseTag attribute*), 78
__weakref__ (*hexrec.formats.asciihex.AsciiHexFile attribute*), 87
__weakref__ (*hexrec.formats.asciihex.AsciiHexRecord attribute*), 118
__weakref__ (*hexrec.formats.avr.AvrFile attribute*), 140
__weakref__ (*hexrec.formats.avr.AvrRecord attribute*), 170
__weakref__ (*hexrec.formats.ihex.IhexFile attribute*), 186
__weakref__ (*hexrec.formats.ihex.IhexRecord attribute*), 217
__weakref__ (*hexrec.formats.mos.MosFile attribute*), 241
__weakref__ (*hexrec.formats.mos.MosRecord attribute*), 271
__weakref__ (*hexrec.formats.raw.RawFile attribute*), 292
__weakref__ (*hexrec.formats.raw.RawRecord attribute*), 322
__weakref__ (*hexrec.formats.srec.SrecFile attribute*), 347
__weakref__ (*hexrec.formats.srec.SrecRecord attribute*), 379
__weakref__ (*hexrec.formats.titxt.TiTxtFile attribute*), 406
__weakref__ (*hexrec.formats.titxt.TiTxtRecord attribute*), 436
__weakref__ (*hexrec.formats.xtek.XtekFile attribute*), 458
__weakref__ (*hexrec.formats.xtek.XtekRecord attribute*), 490
__xor__() (*hexrec.formats.asciihex.AsciiHexTag method*), 130
__xor__() (*hexrec.formats.ihex.IhexTag method*), 231
__xor__() (*hexrec.formats.mos.MosTag method*), 283
__xor__() (*hexrec.formats.srec.SrecTag method*), 393
__xor__() (*hexrec.formats.titxt.TiTxtTag method*), 448
__xor__() (*hexrec.formats.xtek.XtekTag method*), 503
_check_closed() (*hexrec.utils.SparseMemoryIO method*), 515
_generate_next_value_() (*hexrec.formats.asciihex.AsciiHexTag method*), 130
_generate_next_value_() (*hexrec.formats.avr.AvrTag method*), 178
_generate_next_value_() (*hexrec.formats.ihex.IhexTag method*), 231
_generate_next_value_() (*hexrec.formats.mos.MosTag method*), 283
_generate_next_value_()

(*hexrec.formats.raw.RawTag method*), 330
_generate_next_value_() (*hexrec.formats.srec.SrecTag method*), 393
_generate_next_value_() (*hexrec.formats.titxt.TiTxtTag method*), 448
_generate_next_value_() (*hexrec.formats.xtek.XtekTag method*), 503
_is_line_empty() (*hexrec.base.BaseFile class method*), 39
_is_line_empty() (*hexrec.formats.asciihex.AsciiHexFile class method*), 87
_is_line_empty() (*hexrec.formats.avr.AvrFile class method*), 140
_is_line_empty() (*hexrec.formats.ihex.IhexFile class method*), 186
_is_line_empty() (*hexrec.formats.mos.MosFile class method*), 241
_is_line_empty() (*hexrec.formats.raw.RawFile class method*), 292
_is_line_empty() (*hexrec.formats.srec.SrecFile class method*), 347
_is_line_empty() (*hexrec.formats.titxt.TiTxtFile class method*), 406
_is_line_empty() (*hexrec.formats.xtek.XtekFile class method*), 458
_member_type_ (*hexrec.formats.asciihex.AsciiHexTag attribute*), 130
_member_type_ (*hexrec.formats.avr.AvrTag attribute*), 178
_member_type_ (*hexrec.formats.ihex.IhexTag attribute*), 231
_member_type_ (*hexrec.formats.mos.MosTag attribute*), 283
_member_type_ (*hexrec.formats.raw.RawTag attribute*), 330
_member_type_ (*hexrec.formats.srec.SrecTag attribute*), 393
_member_type_ (*hexrec.formats.titxt.TiTxtTag attribute*), 448
_member_type_ (*hexrec.formats.xtek.XtekTag attribute*), 503
_new_member_() (*hexrec.formats.asciihex.AsciiHexTag method*), 130
_new_member_() (*hexrec.formats.avr.AvrTag method*), 178
_new_member_() (*hexrec.formats.ihex.IhexTag method*), 231
_new_member_() (*hexrec.formats.mos.MosTag method*), 283
_new_member_() (*hexrec.formats.raw.RawTag method*), 330
_new_member_() (*hexrec.formats.srec.SrecTag method*), 393
_new_member_() (*hexrec.formats.titxt.TiTxtTag method*), 449
_new_member_() (*hexrec.formats.xtek.XtekTag method*), 503
_value_repr_() (*hexrec.formats.asciihex.AsciiHexTag method*), 130
_value_repr_() (*hexrec.formats.ihex.IhexTag method*), 231
_value_repr_() (*hexrec.formats.mos.MosTag method*), 283
_value_repr_() (*hexrec.formats.srec.SrecTag method*), 393
_value_repr_() (*hexrec.formats.titxt.TiTxtTag method*), 449
_value_repr_() (*hexrec.formats.xtek.XtekTag method*), 503
-C
    hexrec-hexdump command line option, 17
-E
    hexrec-xxd command line option, 22
-I
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 18
    hexrec-xxd command line option, 23
-O
    hexrec-xxd command line option, 23
-U
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 18
    hexrec-xxd command line option, 23
-V
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 18
-X
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 17
--EBCDIC
    hexrec-xxd command line option, 22
--amount
    hexrec-shift command line option, 19
--autoskip
    hexrec-xxd command line option, 22
--bits
    hexrec-xxd command line option, 22
--canonical
    hexrec-hexdump command line option, 17
--clear-holes
    hexrec-merge command line option, 18
--cols
    hexrec-xxd command line option, 22
--ebcdic
    hexrec-xxd command line option, 22
--endex
    hexrec-align command line option, 10
    hexrec-clear command line option, 10

hexrec-crop command line option, 12
hexrec-delete command line option, 13
hexrec-fill command line option, 14
hexrec-flood command line option, 15
--endian
hexrec-xxd command line option, 22
--format
hexrec-srec-get-header command line option, 20
hexrec-srec-set-header command line option, 21
--groupsize
hexrec-xxd command line option, 22
--include
hexrec-xxd command line option, 23
--input-format
hexrec-align command line option, 9
hexrec-clear command line option, 10
hexrec-convert command line option, 11
hexrec-crop command line option, 12
hexrec-delete command line option, 13
hexrec-fill command line option, 14
hexrec-flood command line option, 15
hexrec-hd command line option, 16
hexrec-hexdump command line option, 18
hexrec-merge command line option, 18
hexrec-shift command line option, 19
hexrec-validate command line option, 22
hexrec-xxd command line option, 23
--len
hexrec-xxd command line option, 23
--length
hexrec-hd command line option, 16
hexrec-hexdump command line option, 17
hexrec-xxd command line option, 23
--modulo
hexrec-align command line option, 9
--no_squeezing
hexrec-hd command line option, 16
hexrec-hexdump command line option, 18
--no-seek-zeroes
hexrec-xxd command line option, 24
--offset
hexrec-xxd command line option, 23
--one-byte-char
hexrec-hd command line option, 16
hexrec-hexdump command line option, 17
--one-byte-hex
hexrec-hd command line option, 16
hexrec-hexdump command line option, 17
--one-byte-octal
hexrec-hd command line option, 16
hexrec-hexdump command line option, 17
--output-format
hexrec-align command line option, 9
hexrec-clear command line option, 10
hexrec-convert command line option, 11
hexrec-crop command line option, 12
hexrec-delete command line option, 13
hexrec-fill command line option, 14
hexrec-flood command line option, 15
hexrec-merge command line option, 18
hexrec-shift command line option, 19
hexrec-xxd command line option, 23
--plain
hexrec-xxd command line option, 23
--postscript
hexrec-xxd command line option, 23
--ps
hexrec-xxd command line option, 23
--quadword
hexrec-xxd command line option, 23
--revert
hexrec-xxd command line option, 23
--seek
hexrec-xxd command line option, 23
--seek-zeroes
hexrec-xxd command line option, 24
--skip
hexrec-hd command line option, 16
hexrec-hexdump command line option, 17
--start
hexrec-align command line option, 9
hexrec-clear command line option, 10
hexrec-crop command line option, 12
hexrec-delete command line option, 13
hexrec-fill command line option, 14
hexrec-flood command line option, 15
--two-bytes-decimal
hexrec-hd command line option, 16
hexrec-hexdump command line option, 17
--two-bytes-hex
hexrec-hd command line option, 16
hexrec-hexdump command line option, 17
--two-bytes-octal
hexrec-hd command line option, 16
hexrec-hexdump command line option, 17
--upper
hexrec-hd command line option, 16
hexrec-hexdump command line option, 18
hexrec-xxd command line option, 23
--upper-all
hexrec-xxd command line option, 23
--value
hexrec-align command line option, 10
hexrec-crop command line option, 12
hexrec-fill command line option, 14
hexrec-flood command line option, 15

--version
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 18
    hexrec-xxd command line option, 24
--width
    hexrec-align command line option, 10
    hexrec-clear command line option, 11
    hexrec-convert command line option, 11
    hexrec-crop command line option, 12
    hexrec-delete command line option, 13
    hexrec-fill command line option, 14
    hexrec-flood command line option, 15
    hexrec-merge command line option, 18
    hexrec-shift command line option, 19
-a
    hexrec-xxd command line option, 22
-b
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 17
    hexrec-xxd command line option, 22
-c
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 17
    hexrec-xxd command line option, 22
-d
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 17
-e
    hexrec-align command line option, 10
    hexrec-clear command line option, 10
    hexrec-crop command line option, 12
    hexrec-delete command line option, 13
    hexrec-fill command line option, 14
    hexrec-flood command line option, 15
    hexrec-xxd command line option, 22
-f
    hexrec-srec-get-header command line option, 20
    hexrec-srec-set-header command line option, 21
-g
    hexrec-xxd command line option, 22
-i
    hexrec-align command line option, 9
    hexrec-clear command line option, 10
    hexrec-convert command line option, 11
    hexrec-crop command line option, 12
    hexrec-delete command line option, 13
    hexrec-fill command line option, 14
    hexrec-flood command line option, 15
    hexrec-merge command line option, 18
    hexrec-shift command line option, 19
    hexrec-validate command line option, 22
    hexrec-xxd command line option, 23
-k
    hexrec-xxd command line option, 23
-l
    hexrec-xxd command line option, 23
-m
    hexrec-align command line option, 9
-n
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 17
    hexrec-shift command line option, 19
-o
    hexrec-align command line option, 9
    hexrec-clear command line option, 10
    hexrec-convert command line option, 11
    hexrec-crop command line option, 12
    hexrec-delete command line option, 13
    hexrec-fill command line option, 14
    hexrec-flood command line option, 15
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 17
    hexrec-merge command line option, 18
    hexrec-shift command line option, 19
    hexrec-xxd command line option, 23
-p
    hexrec-xxd command line option, 23
-q
    hexrec-xxd command line option, 23
-r
    hexrec-xxd command line option, 23
-s
    hexrec-align command line option, 9
    hexrec-clear command line option, 10
    hexrec-crop command line option, 12
    hexrec-delete command line option, 13
    hexrec-fill command line option, 14
    hexrec-flood command line option, 15
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 17
    hexrec-xxd command line option, 23
-u
    hexrec-xxd command line option, 23
-v
    hexrec-align command line option, 10
    hexrec-crop command line option, 12
    hexrec-fill command line option, 14
    hexrec-flood command line option, 15
    hexrec-hd command line option, 16
    hexrec-hexdump command line option, 18
    hexrec-xxd command line option, 24
-w
    hexrec-align command line option, 10
    hexrec-clear command line option, 11
    hexrec-convert command line option, 11
    hexrec-crop command line option, 12

hexrec-delete command line option, 13
hexrec-fill command line option, 14
hexrec-flood command line option, 15
hexrec-merge command line option, 18
hexrec-shift command line option, 19
-x
hexrec-hd command line option, 16
hexrec-hexdump command line option, 17

A

ADDRESS (*hexrec.formats.asciihex.AsciiHexTag attribute*), 126
ADDRESS (*hexrec.formats.titxt.TiTxtTag attribute*), 445
align() (*hexrec.base.BaseFile method*), 40
align() (*hexrec.formats.asciihex.AsciiHexFile method*), 88
align() (*hexrec.formats.avr.AvrFile method*), 140
align() (*hexrec.formats.ihex.IhexFile method*), 186
align() (*hexrec.formats.mos.MosFile method*), 242
align() (*hexrec.formats.raw.RawFile method*), 293
align() (*hexrec.formats.srec.SrecFile method*), 347
align() (*hexrec.formats.titxt.TiTxtFile method*), 407
align() (*hexrec.formats.xtek.XtekFile method*), 459
append() (*hexrec.base.BaseFile method*), 40
append() (*hexrec.formats.asciihex.AsciiHexFile method*), 88
append() (*hexrec.formats.avr.AvrFile method*), 141
append() (*hexrec.formats.ihex.IhexFile method*), 187
append() (*hexrec.formats.mos.MosFile method*), 242
append() (*hexrec.formats.raw.RawFile method*), 293
append() (*hexrec.formats.srec.SrecFile method*), 348
append() (*hexrec.formats.titxt.TiTxtFile method*), 407
append() (*hexrec.formats.xtek.XtekFile method*), 459
apply_records() (*hexrec.base.BaseFile method*), 41
apply_records() (*hexrec.formats.asciihex.AsciiHexFile method*), 89
apply_records() (*hexrec.formats.avr.AvrFile method*), 141
apply_records() (*hexrec.formats.ihex.IhexFile method*), 187
apply_records() (*hexrec.formats.mos.MosFile method*), 243
apply_records() (*hexrec.formats.raw.RawFile method*), 294
apply_records() (*hexrec.formats.srec.SrecFile method*), 348
apply_records() (*hexrec.formats.titxt.TiTxtFile method*), 408
apply_records() (*hexrec.formats.xtek.XtekFile method*), 460
as_integer_ratio() (*hexrec.formats.asciihex.AsciiHexTag method*), 130
as_integer_ratio() (*hexrec.formats.ihex.IhexTag method*), 231
as_integer_ratio() (*hexrec.formats.mos.MosTag method*), 283
as_integer_ratio() (*hexrec.formats.srec.SrecTag method*), 393
as_integer_ratio() (*hexrec.formats.titxt.TiTxtTag method*), 449
as_integer_ratio() (*hexrec.formats.xtek.XtekTag method*), 504
AsciiHexFile (*class in hexrec.formats.asciihex*), 80
AsciiHexRecord (*class in hexrec.formats.asciihex*), 114
AsciiHexTag (*class in hexrec.formats.asciihex*), 126
AvrFile (*class in hexrec.formats.avr*), 132
AvrRecord (*class in hexrec.formats.avr*), 166
AvrTag (*class in hexrec.formats.avr*), 177

B

BaseFile (*class in hexrec.base*), 31
BaseRecord (*class in hexrec.base*), 66
BaseTag (*class in hexrec.base*), 78
bit_count() (*hexrec.formats.asciihex.AsciiHexTag method*), 130
bit_count() (*hexrec.formats.ihex.IhexTag method*), 231
bit_count() (*hexrec.formats.mos.MosTag method*), 283
bit_count() (*hexrec.formats.srec.SrecTag method*), 393
bit_count() (*hexrec.formats.titxt.TiTxtTag method*), 449
bit_count() (*hexrec.formats.xtek.XtekTag method*), 504
bit_length() (*hexrec.formats.asciihex.AsciiHexTag method*), 130
bit_length() (*hexrec.formats.ihex.IhexTag method*), 231
bit_length() (*hexrec.formats.mos.MosTag method*), 283
bit_length() (*hexrec.formats.srec.SrecTag method*), 393
bit_length() (*hexrec.formats.titxt.TiTxtTag method*), 449
bit_length() (*hexrec.formats.xtek.XtekTag method*), 504

C

CHAR_ASCII (*in module hexrec.xxd*), 534
CHAR_EBCDIC (*in module hexrec.xxd*), 534
CHAR_PRINTABLE (*in module hexrec.hexdump*), 506
CHAR_TOKENS (*in module hexrec.hexdump*), 506
CHECKSUM (*hexrec.formats.asciihex.AsciiHexTag attribute*), 126
chop() (*in module hexrec.utils*), 509
clear() (*hexrec.base.BaseFile method*), 41
clear() (*hexrec.formats.asciihex.AsciiHexFile method*), 90
clear() (*hexrec.formats.avr.AvrFile method*), 142
clear() (*hexrec.formats.ihex.IhexFile method*), 188
clear() (*hexrec.formats.mos.MosFile method*), 244

clear() (*hexrec.formats.raw.RawFile method*), 295
clear() (*hexrec.formats.srec.SrecFile method*), 349
clear() (*hexrec.formats.titxt.TiTxtFile method*), 409
clear() (*hexrec.formats.xtek.XtekFile method*), 461
close() (*hexrec.utils.SparseMemoryIO method*), 515
closed (*hexrec.utils.SparseMemoryIO property*), 516
colorize_tokens() (*in module hexrec.base*), 26
compute_address_max() (*hexrec.formats.xtek.XtekRecord class method*), 490
compute_checksum() (*hexrec.base.BaseRecord method*), 70
compute_checksum() (*hexrec.formats.asciihex.AsciiHexRecord method*), 118
compute_checksum() (*hexrec.formats.avr.AvrRecord method*), 170
compute_checksum() (*hexrec.formats.ihex.IhexRecord method*), 218
compute_checksum() (*hexrec.formats.mos.MosRecord method*), 271
compute_checksum() (*hexrec.formats.raw.RawRecord method*), 322
compute_checksum() (*hexrec.formats.srec.SrecRecord method*), 379
compute_checksum() (*hexrec.formats.titxt.TiTxtRecord method*), 437
compute_checksum() (*hexrec.formats.xtek.XtekRecord method*), 490
compute_count() (*hexrec.base.BaseRecord method*), 71
compute_count() (*hexrec.formats.asciihex.AsciiHexRecord method*), 118
compute_count() (*hexrec.formats.avr.AvrRecord method*), 170
compute_count() (*hexrec.formats.ihex.IhexRecord method*), 218
compute_count() (*hexrec.formats.mos.MosRecord method*), 272
compute_count() (*hexrec.formats.raw.RawRecord method*), 323
compute_count() (*hexrec.formats.srec.SrecRecord method*), 380
compute_count() (*hexrec.formats.titxt.TiTxtRecord method*), 437
compute_count() (*hexrec.formats.xtek.XtekRecord method*), 491
compute_data_max() (*hexrec.formats.xtek.XtekRecord class method*), 491
conjugate() (*hexrec.formats.asciihex.AsciiHexTag method*), 130
conjugate() (*hexrec.formats.ihex.IhexTag method*), 231
conjugate() (*hexrec.formats.mos.MosTag method*), 283
conjugate() (*hexrec.formats.srec.SrecTag method*), 393
conjugate() (*hexrec.formats.titxt.TiTxtTag method*), 449
conjugate() (*hexrec.formats.xtek.XtekTag method*), 504
convert() (*hexrec.base.BaseFile class method*), 42
convert() (*hexrec.formats.asciihex.AsciiHexFile class method*), 90
convert() (*hexrec.formats.avr.AvrFile class method*), 143
convert() (*hexrec.formats.ihex.IhexFile class method*), 189
convert() (*hexrec.formats.mos.MosFile class method*), 244
convert() (*hexrec.formats.raw.RawFile class method*), 295
convert() (*hexrec.formats.srec.SrecFile class method*), 350
convert() (*hexrec.formats.titxt.TiTxtFile class method*), 409
convert() (*hexrec.formats.xtek.XtekFile class method*), 461
convert() (*in module hexrec.base*), 27
copy() (*hexrec.base.BaseFile method*), 43
copy() (*hexrec.base.BaseRecord method*), 71
copy() (*hexrec.formats.asciihex.AsciiHexFile method*), 91
copy() (*hexrec.formats.asciihex.AsciiHexRecord method*), 119
copy() (*hexrec.formats.avr.AvrFile method*), 143
copy() (*hexrec.formats.avr.AvrRecord method*), 171
copy() (*hexrec.formats.ihex.IhexFile method*), 189
copy() (*hexrec.formats.ihex.IhexRecord method*), 218
copy() (*hexrec.formats.mos.MosFile method*), 245
copy() (*hexrec.formats.mos.MosRecord method*), 272
copy() (*hexrec.formats.raw.RawFile method*), 296
copy() (*hexrec.formats.raw.RawRecord method*), 323
copy() (*hexrec.formats.srec.SrecFile method*), 350
copy() (*hexrec.formats.srec.SrecRecord method*), 380
copy() (*hexrec.formats.titxt.TiTxtFile method*), 410
copy() (*hexrec.formats.titxt.TiTxtRecord method*), 437
copy() (*hexrec.formats.xtek.XtekFile method*), 462
copy() (*hexrec.formats.xtek.XtekRecord method*), 492
COUNT_16 (*hexrec.formats.srec.SrecTag attribute*), 389
COUNT_24 (*hexrec.formats.srec.SrecTag attribute*), 389
create_address() (*hexrec.formats.asciihex.AsciiHexRecord class method*), 119
create_address() (*hexrec.formats.titxt.TiTxtRecord class method*), 438
create_checksum() (*hexrec.formats.asciihex.AsciiHexRecord class method*), 120
create_count() (*hexrec.formats.srec.SrecRecord class method*), 381
create_data() (*hexrec.base.BaseRecord class method*), 72
create_data() (*hexrec.formats.asciihex.AsciiHexRecord class method*), 120

create_data() (*hexrec.formats.avr.AvrRecord class method*), 171
create_data() (*hexrec.formats.ihex.IhexRecord class method*), 219
create_data() (*hexrec.formats.mos.MosRecord class method*), 273
create_data() (*hexrec.formats.raw.RawRecord class method*), 324
create_data() (*hexrec.formats.srec.SrecRecord class method*), 381
create_data() (*hexrec.formats.titxt.TiTxtRecord class method*), 438
create_data() (*hexrec.formats.xtek.XtekRecord class method*), 492
create_end_of_file() (*hexrec.formats.ihex.IhexRecord class method*), 219
create_eof() (*hexrec.formats.mos.MosRecord class method*), 273
create_eof() (*hexrec.formats.titxt.TiTxtRecord class method*), 439
create_eof() (*hexrec.formats.xtek.XtekRecord class method*), 493
create_extended_linear_address() (*hexrec.formats.ihex.IhexRecord class method*), 220
create_extended_segment_address() (*hexrec.formats.ihex.IhexRecord class method*), 220
create_header() (*hexrec.formats.srec.SrecRecord class method*), 382
create_start() (*hexrec.formats.srec.SrecRecord class method*), 382
create_start_linear_address() (*hexrec.formats.ihex.IhexRecord class method*), 220
create_start_segment_address() (*hexrec.formats.ihex.IhexRecord class method*), 221
crop() (*hexrec.base.BaseFile method*), 43
crop() (*hexrec.formats.asciihex.AsciiHexFile method*), 91
crop() (*hexrec.formats.avr.AvrFile method*), 144
crop() (*hexrec.formats.ihex.IhexFile method*), 190
crop() (*hexrec.formats.mos.MosFile method*), 245
crop() (*hexrec.formats.raw.RawFile method*), 296
crop() (*hexrec.formats.srec.SrecFile method*), 351
crop() (*hexrec.formats.titxt.TiTxtFile method*), 410
crop() (*hexrec.formats.xtek.XtekFile method*), 462
cut() (*hexrec.base.BaseFile method*), 44
cut() (*hexrec.formats.asciihex.AsciiHexFile method*), 92
cut() (*hexrec.formats.avr.AvrFile method*), 144
cut() (*hexrec.formats.ihex.IhexFile method*), 190
cut() (*hexrec.formats.mos.MosFile method*), 246
cut() (*hexrec.formats.raw.RawFile method*), 297
cut() (*hexrec.formats.srec.SrecFile method*), 351
cut() (*hexrec.formats.titxt.TiTxtFile method*), 411
cut() (*hexrec.formats.xtek.XtekFile method*), 463

D

DATA (*hexrec.formats.asciihex.AsciiHexTag attribute*), 126
DATA (*hexrec.formats.avr.AvrTag attribute*), 177
DATA (*hexrec.formats.ihex.IhexTag attribute*), 227
DATA (*hexrec.formats.mos.MosTag attribute*), 279
DATA (*hexrec.formats.raw.RawTag attribute*), 330
DATA (*hexrec.formats.titxt.TiTxtTag attribute*), 445
DATA (*hexrec.formats.xtek.XtekTag attribute*), 500
DATA_16 (*hexrec.formats.srec.SrecTag attribute*), 389
DATA_24 (*hexrec.formats.srec.SrecTag attribute*), 389
DATA_32 (*hexrec.formats.srec.SrecTag attribute*), 389
DATA_EXECHARS (*hexrec.formats.asciihex.AsciiHexRecord attribute*), 114
data_to_int() (*hexrec.base.BaseRecord method*), 72
data_to_int() (*hexrec.formats.asciihex.AsciiHexRecord method*), 120
data_to_int() (*hexrec.formats.avr.AvrRecord method*), 172
data_to_int() (*hexrec.formats.ihex.IhexRecord method*), 221
data_to_int() (*hexrec.formats.mos.MosRecord method*), 274
data_to_int() (*hexrec.formats.raw.RawRecord method*), 324
data_to_int() (*hexrec.formats.srec.SrecRecord method*), 383
data_to_int() (*hexrec.formats.titxt.TiTxtRecord method*), 439
data_to_int() (*hexrec.formats.xtek.XtekRecord method*), 493
DEFAULT_DATALEN (*hexrec.base.BaseFile attribute*), 33
DEFAULT_DATALEN (*hexrec.formats.asciihex.AsciiHexFile attribute*), 81
DEFAULT_DATALEN (*hexrec.formats.avr.AvrFile attribute*), 134
DEFAULT_DATALEN (*hexrec.formats.ihex.IhexFile attribute*), 179
DEFAULT_DATALEN (*hexrec.formats.mos.MosFile attribute*), 235
DEFAULT_DATALEN (*hexrec.formats.raw.RawFile attribute*), 286
DEFAULT_DATALEN (*hexrec.formats.srec.SrecFile attribute*), 341
DEFAULT_DATALEN (*hexrec.formats.titxt.TiTxtFile attribute*), 400
DEFAULT_DATALEN (*hexrec.formats.xtek.XtekFile attribute*), 452
DEFAULT_DELETE (*in module hexrec.utils*), 509

DEFAULT_FORMAT_ORDER (*in module hexrec.hexdump*), 507
delete() (*hexrec.base.BaseFile method*), 44
delete() (*hexrec.formats.asciihex.AsciiHexFile method*), 93
delete() (*hexrec.formats.avr.AvrFile method*), 145
delete() (*hexrec.formats.ihex.IhexFile method*), 191
delete() (*hexrec.formats.mos.MosFile method*), 247
delete() (*hexrec.formats.raw.RawFile method*), 298
delete() (*hexrec.formats.srec.SrecFile method*), 352
delete() (*hexrec.formats.titxt.TiTxtFile method*), 412
delete() (*hexrec.formats.xtek.XtekFile method*), 464
denominator (*hexrec.formats.asciihex.AsciiHexTag attribute*), 130
denominator (*hexrec.formats.ihex.IhexTag attribute*), 231
denominator (*hexrec.formats.mos.MosTag attribute*), 284
denominator (*hexrec.formats.srec.SrecTag attribute*), 393
denominator (*hexrec.formats.titxt.TiTxtTag attribute*), 449
denominator (*hexrec.formats.xtek.XtekTag attribute*), 504
detach() (*hexrec.utils.SparseMemoryIO method*), 516
discard_memory() (*hexrec.base.BaseFile method*), 45
discard_memory() (*hexrec.formats.asciihex.AsciiHexFile method*), 93
discard_memory() (*hexrec.formats.avr.AvrFile method*), 146
discard_memory() (*hexrec.formats.ihex.IhexFile method*), 192
discard_memory() (*hexrec.formats.mos.MosFile method*), 247
discard_memory() (*hexrec.formats.raw.RawFile method*), 298
discard_memory() (*hexrec.formats.srec.SrecFile method*), 353
discard_memory() (*hexrec.formats.titxt.TiTxtFile method*), 412
discard_memory() (*hexrec.formats.xtek.XtekFile method*), 464
discard_records() (*hexrec.base.BaseFile method*), 45
discard_records() (*hexrec.formats.asciihex.AsciiHexFile method*), 94
discard_records() (*hexrec.formats.avr.AvrFile method*), 146
discard_records() (*hexrec.formats.ihex.IhexFile method*), 192
discard_records() (*hexrec.formats.mos.MosFile method*), 248
discard_records() (*hexrec.formats.raw.RawFile method*), 299
discard_records() (*hexrec.formats.srec.SrecFile method*), 353
discard_records() (*hexrec.formats.titxt.TiTxtFile method*), 413
discard_records() (*hexrec.formats.xtek.XtekFile method*), 465

E

END_OF_FILE (*hexrec.formats.ihex.IhexTag attribute*), 227
EOF (*hexrec.formats.mos.MosTag attribute*), 280
EOF (*hexrec.formats.titxt.TiTxtTag attribute*), 445
EOF (*hexrec.formats.xtek.XtekTag attribute*), 500
EQUALITY_KEYS (*hexrec.base.BaseRecord attribute*), 67
EQUALITY_KEYS (*hexrec.formats.asciihex.AsciiHexRecord attribute*), 115
EQUALITY_KEYS (*hexrec.formats.avr.AvrRecord attribute*), 167
EQUALITY_KEYS (*hexrec.formats.ihex.IhexRecord attribute*), 214
EQUALITY_KEYS (*hexrec.formats.mos.MosRecord attribute*), 268
EQUALITY_KEYS (*hexrec.formats.raw.RawRecord attribute*), 319
EQUALITY_KEYS (*hexrec.formats.srec.SrecRecord attribute*), 376
EQUALITY_KEYS (*hexrec.formats.titxt.TiTxtRecord attribute*), 433
EQUALITY_KEYS (*hexrec.formats.xtek.XtekRecord attribute*), 486
extend() (*hexrec.base.BaseFile method*), 46
extend() (*hexrec.formats.asciihex.AsciiHexFile method*), 94
extend() (*hexrec.formats.avr.AvrFile method*), 147
extend() (*hexrec.formats.ihex.IhexFile method*), 193
extend() (*hexrec.formats.mos.MosFile method*), 248
extend() (*hexrec.formats.raw.RawFile method*), 299
extend() (*hexrec.formats.srec.SrecFile method*), 354
extend() (*hexrec.formats.titxt.TiTxtFile method*), 413
extend() (*hexrec.formats.xtek.XtekFile method*), 465
EXTENDED_LINEAR_ADDRESS (*hexrec.formats.ihex.IhexTag attribute*), 227
EXTENDED_SEGMENT_ADDRESS (*hexrec.formats.ihex.IhexTag attribute*), 227

F

FILE_EXT (*hexrec.base.BaseFile attribute*), 33
FILE_EXT (*hexrec.formats.asciihex.AsciiHexFile attribute*), 81
FILE_EXT (*hexrec.formats.avr.AvrFile attribute*), 134
FILE_EXT (*hexrec.formats.ihex.IhexFile attribute*), 179
FILE_EXT (*hexrec.formats.mos.MosFile attribute*), 235
FILE_EXT (*hexrec.formats.raw.RawFile attribute*), 286
FILE_EXT (*hexrec.formats.srec.SrecFile attribute*), 341
FILE_EXT (*hexrec.formats.titxt.TiTxtFile attribute*), 400

FILE_EXT (*hexrec.formats.xtek.XtekFile attribute*), 452
FILE_TYPES (*in module hexrec.base*), 25
fileno() (*hexrec.utils.SparseMemoryIO method*), 516
fill() (*hexrec.base.BaseFile method*), 46
fill() (*hexrec.formats.asciihex.AsciiHexFile method*), 95
fill() (*hexrec.formats.avr.AvrFile method*), 147
fill() (*hexrec.formats.ihex.IhexFile method*), 193
fill() (*hexrec.formats.mos.MosFile method*), 249
fill() (*hexrec.formats.raw.RawFile method*), 300
fill() (*hexrec.formats.srec.SrecFile method*), 354
fill() (*hexrec.formats.titxt.TiTxtFile method*), 414
fill() (*hexrec.formats.xtek.XtekFile method*), 466
find() (*hexrec.base.BaseFile method*), 47
find() (*hexrec.formats.asciihex.AsciiHexFile method*), 95
find() (*hexrec.formats.avr.AvrFile method*), 148
find() (*hexrec.formats.ihex.IhexFile method*), 194
find() (*hexrec.formats.mos.MosFile method*), 249
find() (*hexrec.formats.raw.RawFile method*), 300
find() (*hexrec.formats.srec.SrecFile method*), 355
find() (*hexrec.formats.titxt.TiTxtFile method*), 414
find() (*hexrec.formats.xtek.XtekFile method*), 466
fit_count_tag() (*hexrec.formats.srec.SrecTag class method*), 393
fit_data_tag() (*hexrec.formats.srec.SrecTag class method*), 394
fit_start_tag() (*hexrec.formats.srec.SrecTag class method*), 394
flood() (*hexrec.base.BaseFile method*), 48
flood() (*hexrec.formats.asciihex.AsciiHexFile method*), 96
flood() (*hexrec.formats.avr.AvrFile method*), 149
flood() (*hexrec.formats.ihex.IhexFile method*), 195
flood() (*hexrec.formats.mos.MosFile method*), 250
flood() (*hexrec.formats.raw.RawFile method*), 301
flood() (*hexrec.formats.srec.SrecFile method*), 356
flood() (*hexrec.formats.titxt.TiTxtFile method*), 415
flood() (*hexrec.formats.xtek.XtekFile method*), 467
flush() (*hexrec.utils.SparseMemoryIO method*), 517
from_blocks() (*hexrec.base.BaseFile class method*), 48
from_blocks() (*hexrec.formats.asciihex.AsciiHexFile class method*), 97
from_blocks() (*hexrec.formats.avr.AvrFile class method*), 149
from_blocks() (*hexrec.formats.ihex.IhexFile class method*), 195
from_blocks() (*hexrec.formats.mos.MosFile class method*), 251
from_blocks() (*hexrec.formats.raw.RawFile class method*), 302
from_blocks() (*hexrec.formats.srec.SrecFile class method*), 356
from_blocks() (*hexrec.formats.titxt.TiTxtFile class method*), 416
from_blocks() (*hexrec.formats.xtek.XtekFile class method*), 468
from_bytes() (*hexrec.base.BaseFile class method*), 49
from_bytes() (*hexrec.formats.asciihex.AsciiHexFile class method*), 97
from_bytes() (*hexrec.formats.asciihex.AsciiHexTag method*), 130
from_bytes() (*hexrec.formats.avr.AvrFile class method*), 150
from_bytes() (*hexrec.formats.ihex.IhexFile class method*), 196
from_bytes() (*hexrec.formats.ihex.IhexTag method*), 232
from_bytes() (*hexrec.formats.mos.MosFile class method*), 251
from_bytes() (*hexrec.formats.mos.MosTag method*), 284
from_bytes() (*hexrec.formats.raw.RawFile class method*), 302
from_bytes() (*hexrec.formats.srec.SrecFile class method*), 357
from_bytes() (*hexrec.formats.srec.SrecTag method*), 395
from_bytes() (*hexrec.formats.titxt.TiTxtFile class method*), 416
from_bytes() (*hexrec.formats.titxt.TiTxtTag method*), 449
from_bytes() (*hexrec.formats.xtek.XtekFile class method*), 468
from_bytes() (*hexrec.formats.xtek.XtekTag method*), 504
from_memory() (*hexrec.base.BaseFile class method*), 50
from_memory() (*hexrec.formats.asciihex.AsciiHexFile class method*), 98
from_memory() (*hexrec.formats.avr.AvrFile class method*), 150
from_memory() (*hexrec.formats.ihex.IhexFile class method*), 196
from_memory() (*hexrec.formats.mos.MosFile class method*), 252
from_memory() (*hexrec.formats.raw.RawFile class method*), 303
from_memory() (*hexrec.formats.srec.SrecFile class method*), 357
from_memory() (*hexrec.formats.titxt.TiTxtFile class method*), 417
from_memory() (*hexrec.formats.xtek.XtekFile class method*), 469
from_numbers() (*in module hexrec.formats.sqtp*), 331
from_records() (*hexrec.base.BaseFile class method*), 50
from_records() (*hexrec.formats.asciihex.AsciiHexFile*

class method), 99
from_records() (*hexrec.formats.avr.AvrFile class method*), 151
from_records() (*hexrec.formats.ihex.IhexFile class method*), 197
from_records() (*hexrec.formats.mos.MosFile class method*), 253
from_records() (*hexrec.formats.raw.RawFile class method*), 304
from_records() (*hexrec.formats.srec.SrecFile class method*), 358
from_records() (*hexrec.formats.titxt.TiTxtFile class method*), 418
from_records() (*hexrec.formats.xtek.XtekFile class method*), 470
from_strings() (*in module hexrec.formats.sqtp*), 334

G

get_address_max() (*hexrec.base.BaseFile method*), 51
get_address_max() (*hexrec.formats.asciihex.AsciiHexFile method*), 99
get_address_max() (*hexrec.formats.avr.AvrFile method*), 152
get_address_max() (*hexrec.formats.ihex.IhexFile method*), 198
get_address_max() (*hexrec.formats.mos.MosFile method*), 253
get_address_max() (*hexrec.formats.raw.RawFile method*), 304
get_address_max() (*hexrec.formats.srec.SrecFile method*), 359
get_address_max() (*hexrec.formats.srec.SrecTag method*), 395
get_address_max() (*hexrec.formats.titxt.TiTxtFile method*), 418
get_address_max() (*hexrec.formats.xtek.XtekFile method*), 470
get_address_max() (*hexrec.formats.xtek.XtekRecord method*), 494
get_address_min() (*hexrec.base.BaseFile method*), 52
get_address_min() (*hexrec.formats.asciihex.AsciiHexFile method*), 100
get_address_min() (*hexrec.formats.avr.AvrFile method*), 152
get_address_min() (*hexrec.formats.ihex.IhexFile method*), 198
get_address_min() (*hexrec.formats.mos.MosFile method*), 254
get_address_min() (*hexrec.formats.raw.RawFile method*), 305
get_address_min() (*hexrec.formats.srec.SrecFile method*), 359
get_address_min() (*hexrec.formats.titxt.TiTxtFile method*), 419
get_address_min() (*hexrec.formats.xtek.XtekFile method*), 471
get_address_size() (*hexrec.formats.srec.SrecTag method*), 396
get_data_max() (*hexrec.formats.srec.SrecTag method*), 396
get_data_max() (*hexrec.formats.xtek.XtekRecord method*), 494
get_holes() (*hexrec.base.BaseFile method*), 52
get_holes() (*hexrec.formats.asciihex.AsciiHexFile method*), 100
get_holes() (*hexrec.formats.avr.AvrFile method*), 153
get_holes() (*hexrec.formats.ihex.IhexFile method*), 199
get_holes() (*hexrec.formats.mos.MosFile method*), 254
get_holes() (*hexrec.formats.raw.RawFile method*), 305
get_holes() (*hexrec.formats.srec.SrecFile method*), 360
get_holes() (*hexrec.formats.titxt.TiTxtFile method*), 419
get_holes() (*hexrec.formats.xtek.XtekFile method*), 471
get_meta() (*hexrec.base.BaseFile method*), 52
get_meta() (*hexrec.base.BaseRecord method*), 73
get_meta() (*hexrec.formats.asciihex.AsciiHexFile method*), 101
get_meta() (*hexrec.formats.asciihex.AsciiHexRecord method*), 121
get_meta() (*hexrec.formats.avr.AvrFile method*), 153
get_meta() (*hexrec.formats.avr.AvrRecord method*), 172
get_meta() (*hexrec.formats.ihex.IhexFile method*), 199
get_meta() (*hexrec.formats.ihex.IhexRecord method*), 221
get_meta() (*hexrec.formats.mos.MosFile method*), 255
get_meta() (*hexrec.formats.mos.MosRecord method*), 274
get_meta() (*hexrec.formats.raw.RawFile method*), 306
get_meta() (*hexrec.formats.raw.RawRecord method*), 325
get_meta() (*hexrec.formats.srec.SrecFile method*), 360
get_meta() (*hexrec.formats.srec.SrecRecord method*), 383
get_meta() (*hexrec.formats.titxt.TiTxtFile method*), 420
get_meta() (*hexrec.formats.titxt.TiTxtRecord method*), 440
get_meta() (*hexrec.formats.xtek.XtekFile method*), 472
get_meta() (*hexrec.formats.xtek.XtekRecord method*), 495
get_spans() (*hexrec.base.BaseFile method*), 53
get_spans() (*hexrec.formats.asciihex.AsciiHexFile method*), 101
get_spans() (*hexrec.formats.avr.AvrFile method*), 153

get_spans() (*hexrec.formats.ihex.IhexFile method*), 199
get_spans() (*hexrec.formats.mos.MosFile method*), 255
get_spans() (*hexrec.formats.raw.RawFile method*), 306
get_spans() (*hexrec.formats.srec.SrecFile method*), 360
get_spans() (*hexrec.formats.titxt.TiTxtFile method*), 420
get_spans() (*hexrec.formats.xtek.XtekFile method*), 472
get_tag_match() (*hexrec.formats.srec.SrecTag method*), 397
getbuffer() (*hexrec.utils.SparseMemoryIO method*), 517
getvalue() (*hexrec.utils.SparseMemoryIO method*), 517
guess_format_name() (*in module hexrec.base*), 28
guess_format_type() (*in module hexrec.base*), 28

H

HEADER
    hexrec-srec-set-header command line option, 21
header (*hexrec.formats.srec.SrecFile property*), 361
HEADER (*hexrec.formats.srec.SrecTag attribute*), 389
hexdump_core() (*in module hexrec.hexdump*), 507
hexlify() (*in module hexrec.utils*), 510
hexrec.base
    module, 25
hexrec.formats
    module, 79
hexrec.formats.asciihex
    module, 80
hexrec.formats.avr
    module, 132
hexrec.formats.ihex
    module, 178
hexrec.formats.mos
    module, 234
hexrec.formats.raw
    module, 285
hexrec.formats.sqtp
    module, 331
hexrec.formats.srec
    module, 339
hexrec.formats.titxt
    module, 399
hexrec.formats.xtek
    module, 451
hexrec.hexdump
    module, 505
hexrec.utils
    module, 509
hexrec.xxd
    module, 534
hexrec-align command line option
    --endex, 10
    --input-format, 9
    --modulo, 9
    --output-format, 9
    --start, 9
    --value, 10
    --width, 10
    -e, 10
    -i, 9
    -m, 9
    -o, 9
    -s, 9
    -v, 10
    -w, 10
    INFILE, 10
    OUTFILE, 10
hexrec-clear command line option
    --endex, 10
    --input-format, 10
    --output-format, 10
    --start, 10
    --width, 11
    -e, 10
    -i, 10
    -o, 10
    -s, 10
    -w, 11
    INFILE, 11
    OUTFILE, 11
hexrec-convert command line option
    --input-format, 11
    --output-format, 11
    --width, 11
    -i, 11
    -o, 11
    -w, 11
    INFILE, 11
    OUTFILE, 11
hexrec-crop command line option
    --endex, 12
    --input-format, 12
    --output-format, 12
    --start, 12
    --value, 12
    --width, 12
    -e, 12
    -i, 12
    -o, 12
    -s, 12
    -v, 12
    -w, 12

INFILE, 12
OUTFILE, 12
hexrec-delete command line option
--endex, 13
--input-format, 13
--output-format, 13
--start, 13
--width, 13
-e, 13
-i, 13
-o, 13
-s, 13
-w, 13
INFILE, 13
OUTFILE, 13
hexrec-fill command line option
--endex, 14
--input-format, 14
--output-format, 14
--start, 14
--value, 14
--width, 14
-e, 14
-i, 14
-o, 14
-s, 14
-v, 14
-w, 14
INFILE, 14
OUTFILE, 14
hexrec-flood command line option
--endex, 15
--input-format, 15
--output-format, 15
--start, 15
--value, 15
--width, 15
-e, 15
-i, 15
-o, 15
-s, 15
-v, 15
-w, 15
INFILE, 15
OUTFILE, 15
hexrec-hd command line option
-I, 16
-U, 16
-V, 16
-X, 16
--input-format, 16
--length, 16
--no_squeezing, 16
--one-byte-char, 16
--one-byte-hex, 16
--one-byte-octal, 16
--skip, 16
--two-bytes-decimal, 16
--two-bytes-hex, 16
--two-bytes-octal, 16
--upper, 16
--version, 16
-b, 16
-c, 16
-d, 16
-n, 16
-o, 16
-s, 16
-v, 16
-x, 16
INFILE, 17
hexrec-hexdump command line option
-C, 17
-I, 18
-U, 18
-V, 18
-X, 17
--canonical, 17
--input-format, 18
--length, 17
--no_squeezing, 18
--one-byte-char, 17
--one-byte-hex, 17
--one-byte-octal, 17
--skip, 17
--two-bytes-decimal, 17
--two-bytes-hex, 17
--two-bytes-octal, 17
--upper, 18
--version, 18
-b, 17
-c, 17
-d, 17
-n, 17
-o, 17
-s, 17
-v, 18
-x, 17
INFILE, 18
hexrec-merge command line option
--clear-holes, 18
--input-format, 18
--output-format, 18
--width, 18
-i, 18
-o, 18
-w, 18
INFILES, 19

OUTFILE, 19
hexrec-shift command line option
--amount, 19
--input-format, 19
--output-format, 19
--width, 19
-i, 19
-n, 19
-o, 19
-w, 19
INFILE, 19
OUTFILE, 19
hexrec-srec-del-header command line option
INFILE, 20
OUTFILE, 20
hexrec-srec-get-header command line option
--format, 20
-f, 20
INFILE, 21
hexrec-srec-set-header command line option
--format, 21
-f, 21
HEADER, 21
INFILE, 21
OUTFILE, 21
hexrec-validate command line option
--input-format, 22
-i, 22
INFILE, 22
hexrec-xxd command line option
-E, 22
-I, 23
-O, 23
-U, 23
--EBCDIC, 22
--autoskip, 22
--bits, 22
--cols, 22
--ebcdic, 22
--endian, 22
--groupsize, 22
--include, 23
--input-format, 23
--len, 23
--length, 23
--no-seek-zeroes, 24
--offset, 23
--output-format, 23
--plain, 23
--postscript, 23
--ps, 23
--quadword, 23
--revert, 23
--seek, 23
--seek-zeroes, 24
--upper, 23
--upper-all, 23
--version, 24
-a, 22
-b, 22
-c, 22
-e, 22
-g, 22
-i, 23
-k, 23
-l, 23
-o, 23
-p, 23
-q, 23
-r, 23
-s, 23
-u, 23
-v, 24
INFILE, 24
OUTFILE, 24

I

IhexFile (*class in hexrec.formats.ihex*), 178
IhexRecord (*class in hexrec.formats.ihex*), 214
IhexTag (*class in hexrec.formats.ihex*), 227
imag (*hexrec.formats.asciihex.AsciiHexTag attribute*), 131
imag (*hexrec.formats.ihex.IhexTag attribute*), 232
imag (*hexrec.formats.mos.MosTag attribute*), 284
imag (*hexrec.formats.srec.SrecTag attribute*), 397
imag (*hexrec.formats.titxt.TiTxtTag attribute*), 450
imag (*hexrec.formats.xtek.XtekTag attribute*), 504
index() (*hexrec.base.BaseFile method*), 53
index() (*hexrec.formats.asciihex.AsciiHexFile method*), 101
index() (*hexrec.formats.avr.AvrFile method*), 154
index() (*hexrec.formats.ihex.IhexFile method*), 200
index() (*hexrec.formats.mos.MosFile method*), 255
index() (*hexrec.formats.raw.RawFile method*), 306
index() (*hexrec.formats.srec.SrecFile method*), 362
index() (*hexrec.formats.titxt.TiTxtFile method*), 420
index() (*hexrec.formats.xtek.XtekFile method*), 472
INFILE
hexrec-align command line option, 10
hexrec-clear command line option, 11
hexrec-convert command line option, 11
hexrec-crop command line option, 12
hexrec-delete command line option, 13
hexrec-fill command line option, 14
hexrec-flood command line option, 15
hexrec-hd command line option, 17
hexrec-hexdump command line option, 18
hexrec-shift command line option, 19

hexrec-srec-del-header command line option, 20
hexrec-srec-get-header command line option, 21
hexrec-srec-set-header command line option, 21
hexrec-validate command line option, 22
hexrec-xxd command line option, 24
INFILES
hexrec-merge command line option, 19
is_address() (*hexrec.formats.asciihex.AsciiHexTag method*), 131
is_address() (*hexrec.formats.titxt.TiTxtTag method*), 450
is_checksum() (*hexrec.formats.asciihex.AsciiHexTag method*), 131
is_count() (*hexrec.formats.srec.SrecTag method*), 397
is_data() (*hexrec.base.BaseTag method*), 78
is_eof() (*hexrec.formats.ihex.IhexTag method*), 232
is_eof() (*hexrec.formats.mos.MosTag method*), 284
is_eof() (*hexrec.formats.titxt.TiTxtTag method*), 450
is_eof() (*hexrec.formats.xtek.XtekTag method*), 505
is_extension() (*hexrec.formats.ihex.IhexTag method*), 232
is_file_termination() (*hexrec.base.BaseTag method*), 79
is_header() (*hexrec.formats.srec.SrecTag method*), 397
is_start() (*hexrec.formats.ihex.IhexTag method*), 233
is_start() (*hexrec.formats.srec.SrecTag method*), 398
isatty() (*hexrec.utils.SparseMemoryIO method*), 518

L

LINE1_REGEX (*hexrec.formats.srec.SrecRecord attribute*), 376
LINE1_REGEX (*hexrec.formats.xtek.XtekRecord attribute*), 486
LINE2_REGEX (*hexrec.formats.srec.SrecRecord attribute*), 376
LINE2_REGEX (*hexrec.formats.xtek.XtekRecord attribute*), 486
LINE3_REGEX (*hexrec.formats.srec.SrecRecord attribute*), 376
LINE_REGEX (*hexrec.formats.asciihex.AsciiHexRecord attribute*), 115
LINE_REGEX (*hexrec.formats.avr.AvrRecord attribute*), 167
LINE_REGEX (*hexrec.formats.ihex.IhexRecord attribute*), 215
LINE_REGEX (*hexrec.formats.mos.MosRecord attribute*), 268
LINE_REGEX (*hexrec.formats.titxt.TiTxtRecord attribute*), 434
linear (*hexrec.formats.ihex.IhexFile property*), 201
load() (*hexrec.base.BaseFile class method*), 54
load() (*hexrec.formats.asciihex.AsciiHexFile class method*), 102
load() (*hexrec.formats.avr.AvrFile class method*), 155
load() (*hexrec.formats.ihex.IhexFile class method*), 201
load() (*hexrec.formats.mos.MosFile class method*), 256
load() (*hexrec.formats.raw.RawFile class method*), 307
load() (*hexrec.formats.srec.SrecFile class method*), 362
load() (*hexrec.formats.titxt.TiTxtFile class method*), 421
load() (*hexrec.formats.xtek.XtekFile class method*), 473
load() (*in module hexrec.base*), 29

M

maxdatalen (*hexrec.base.BaseFile property*), 55
maxdatalen (*hexrec.formats.asciihex.AsciiHexFile property*), 103
maxdatalen (*hexrec.formats.avr.AvrFile property*), 155
maxdatalen (*hexrec.formats.ihex.IhexFile property*), 202
maxdatalen (*hexrec.formats.mos.MosFile property*), 257
maxdatalen (*hexrec.formats.raw.RawFile property*), 308
maxdatalen (*hexrec.formats.srec.SrecFile property*), 363
maxdatalen (*hexrec.formats.titxt.TiTxtFile property*), 422
maxdatalen (*hexrec.formats.xtek.XtekFile property*), 474
memory (*hexrec.base.BaseFile property*), 56
memory (*hexrec.formats.asciihex.AsciiHexFile property*), 104
memory (*hexrec.formats.avr.AvrFile property*), 156
memory (*hexrec.formats.ihex.IhexFile property*), 203
memory (*hexrec.formats.mos.MosFile property*), 258
memory (*hexrec.formats.raw.RawFile property*), 309
memory (*hexrec.formats.srec.SrecFile property*), 364
memory (*hexrec.formats.titxt.TiTxtFile property*), 423
memory (*hexrec.formats.xtek.XtekFile property*), 475
memory (*hexrec.utils.SparseMemoryIO property*), 518
merge() (*hexrec.base.BaseFile method*), 56
merge() (*hexrec.formats.asciihex.AsciiHexFile method*), 104
merge() (*hexrec.formats.avr.AvrFile method*), 157
merge() (*hexrec.formats.ihex.IhexFile method*), 204
merge() (*hexrec.formats.mos.MosFile method*), 258
merge() (*hexrec.formats.raw.RawFile method*), 309
merge() (*hexrec.formats.srec.SrecFile method*), 365
merge() (*hexrec.formats.titxt.TiTxtFile method*), 423
merge() (*hexrec.formats.xtek.XtekFile method*), 475
merge() (*in module hexrec.base*), 30
META_KEYS (*hexrec.base.BaseFile attribute*), 33
META_KEYS (*hexrec.base.BaseRecord attribute*), 67
META_KEYS (*hexrec.formats.asciihex.AsciiHexFile attribute*), 81
META_KEYS (*hexrec.formats.asciihex.AsciiHexRecord attribute*), 115

META_KEYS (*hexrec.formats.avr.AvrFile attribute*), 134
META_KEYS (*hexrec.formats.avr.AvrRecord attribute*), 167
META_KEYS (*hexrec.formats.ihex.IhexFile attribute*), 180
META_KEYS (*hexrec.formats.ihex.IhexRecord attribute*), 215
META_KEYS (*hexrec.formats.mos.MosFile attribute*), 235
META_KEYS (*hexrec.formats.mos.MosRecord attribute*), 268
META_KEYS (*hexrec.formats.raw.RawFile attribute*), 286
META_KEYS (*hexrec.formats.raw.RawRecord attribute*), 319
META_KEYS (*hexrec.formats.srec.SrecFile attribute*), 341
META_KEYS (*hexrec.formats.srec.SrecRecord attribute*), 376
META_KEYS (*hexrec.formats.titxt.TiTxtFile attribute*), 400
META_KEYS (*hexrec.formats.titxt.TiTxtRecord attribute*), 434
META_KEYS (*hexrec.formats.xtek.XtekFile attribute*), 452
META_KEYS (*hexrec.formats.xtek.XtekRecord attribute*), 487
module
    hexrec.base, 25
    hexrec.formats, 79
    hexrec.formats.asciihex, 80
    hexrec.formats.avr, 132
    hexrec.formats.ihex, 178
    hexrec.formats.mos, 234
    hexrec.formats.raw, 285
    hexrec.formats.sqtp, 331
    hexrec.formats.srec, 339
    hexrec.formats.titxt, 399
    hexrec.formats.xtek, 451
    hexrec.hexdump, 505
    hexrec.utils, 509
    hexrec.xxd, 534
MosFile (*class in hexrec.formats.mos*), 234
MosRecord (*class in hexrec.formats.mos*), 268
MosTag (*class in hexrec.formats.mos*), 279

N

numerator (*hexrec.formats.asciihex.AsciiHexTag attribute*), 131
numerator (*hexrec.formats.ihex.IhexTag attribute*), 233
numerator (*hexrec.formats.mos.MosTag attribute*), 284
numerator (*hexrec.formats.srec.SrecTag attribute*), 398
numerator (*hexrec.formats.titxt.TiTxtTag attribute*), 450
numerator (*hexrec.formats.xtek.XtekTag attribute*), 505

O

OUTFILE
    hexrec-align command line option, 10
    hexrec-clear command line option, 11
    hexrec-convert command line option, 11
    hexrec-crop command line option, 12
    hexrec-delete command line option, 13
    hexrec-fill command line option, 14
    hexrec-flood command line option, 15
    hexrec-merge command line option, 19
    hexrec-shift command line option, 19
    hexrec-srec-del-header command line option, 20
    hexrec-srec-set-header command line option, 21
    hexrec-xxd command line option, 24

P

parse() (*hexrec.base.BaseFile class method*), 57
parse() (*hexrec.base.BaseRecord class method*), 73
parse() (*hexrec.formats.asciihex.AsciiHexFile class method*), 105
parse() (*hexrec.formats.asciihex.AsciiHexRecord class method*), 121
parse() (*hexrec.formats.avr.AvrFile class method*), 158
parse() (*hexrec.formats.avr.AvrRecord class method*), 173
parse() (*hexrec.formats.ihex.IhexFile class method*), 204
parse() (*hexrec.formats.ihex.IhexRecord class method*), 222
parse() (*hexrec.formats.mos.MosFile class method*), 259
parse() (*hexrec.formats.mos.MosRecord class method*), 275
parse() (*hexrec.formats.raw.RawFile class method*), 310
parse() (*hexrec.formats.raw.RawRecord class method*), 325
parse() (*hexrec.formats.srec.SrecFile class method*), 365
parse() (*hexrec.formats.srec.SrecRecord class method*), 384
parse() (*hexrec.formats.titxt.TiTxtFile class method*), 424
parse() (*hexrec.formats.titxt.TiTxtRecord class method*), 440
parse() (*hexrec.formats.xtek.XtekFile class method*), 476
parse() (*hexrec.formats.xtek.XtekRecord class method*), 495
parse_int() (*in module hexrec.utils*), 511
parse_seek() (*in module hexrec.xxd*), 534
peek() (*hexrec.utils.SparseMemoryIO method*), 518
print() (*hexrec.base.BaseFile method*), 58
print() (*hexrec.base.BaseRecord method*), 74
print() (*hexrec.formats.asciihex.AsciiHexFile method*), 106

print() (*hexrec.formats.asciihex.AsciiHexRecord method*), 122
print() (*hexrec.formats.avr.AvrFile method*), 158
print() (*hexrec.formats.avr.AvrRecord method*), 173
print() (*hexrec.formats.ihex.IhexFile method*), 205
print() (*hexrec.formats.ihex.IhexRecord method*), 222
print() (*hexrec.formats.mos.MosFile method*), 260
print() (*hexrec.formats.mos.MosRecord method*), 275
print() (*hexrec.formats.raw.RawFile method*), 311
print() (*hexrec.formats.raw.RawRecord method*), 326
print() (*hexrec.formats.srec.SrecFile method*), 366
print() (*hexrec.formats.srec.SrecRecord method*), 384
print() (*hexrec.formats.titxt.TiTxtFile method*), 425
print() (*hexrec.formats.titxt.TiTxtRecord method*), 441
print() (*hexrec.formats.xtek.XtekFile method*), 477
print() (*hexrec.formats.xtek.XtekRecord method*), 496

R

RawFile (*class in hexrec.formats.raw*), 285
RawRecord (*class in hexrec.formats.raw*), 319
RawTag (*class in hexrec.formats.raw*), 330
read() (*hexrec.base.BaseFile method*), 59
read() (*hexrec.formats.asciihex.AsciiHexFile method*), 107
read() (*hexrec.formats.avr.AvrFile method*), 159
read() (*hexrec.formats.ihex.IhexFile method*), 206
read() (*hexrec.formats.mos.MosFile method*), 261
read() (*hexrec.formats.raw.RawFile method*), 312
read() (*hexrec.formats.srec.SrecFile method*), 367
read() (*hexrec.formats.titxt.TiTxtFile method*), 426
read() (*hexrec.formats.xtek.XtekFile method*), 478
read() (*hexrec.utils.SparseMemoryIO method*), 519
read1() (*hexrec.utils.SparseMemoryIO method*), 520
readable() (*hexrec.utils.SparseMemoryIO method*), 521
readinto() (*hexrec.utils.SparseMemoryIO method*), 521
readinto1() (*hexrec.utils.SparseMemoryIO method*), 523
readline() (*hexrec.utils.SparseMemoryIO method*), 524
readlines() (*hexrec.utils.SparseMemoryIO method*), 526
real (*hexrec.formats.asciihex.AsciiHexTag attribute*), 132
real (*hexrec.formats.ihex.IhexTag attribute*), 233
real (*hexrec.formats.mos.MosTag attribute*), 284
real (*hexrec.formats.srec.SrecTag attribute*), 398
real (*hexrec.formats.titxt.TiTxtTag attribute*), 450
real (*hexrec.formats.xtek.XtekTag attribute*), 505
Record (*hexrec.base.BaseFile attribute*), 33
Record (*hexrec.formats.asciihex.AsciiHexFile attribute*), 81
Record (*hexrec.formats.avr.AvrFile attribute*), 134
Record (*hexrec.formats.ihex.IhexFile attribute*), 180
Record (*hexrec.formats.mos.MosFile attribute*), 235
Record (*hexrec.formats.raw.RawFile attribute*), 286
Record (*hexrec.formats.srec.SrecFile attribute*), 341
Record (*hexrec.formats.titxt.TiTxtFile attribute*), 400
Record (*hexrec.formats.xtek.XtekFile attribute*), 452
records (*hexrec.base.BaseFile property*), 60
records (*hexrec.formats.asciihex.AsciiHexFile property*), 108
records (*hexrec.formats.avr.AvrFile property*), 160
records (*hexrec.formats.ihex.IhexFile property*), 207
records (*hexrec.formats.mos.MosFile property*), 262
records (*hexrec.formats.raw.RawFile property*), 313
records (*hexrec.formats.srec.SrecFile property*), 368
records (*hexrec.formats.titxt.TiTxtFile property*), 427
records (*hexrec.formats.xtek.XtekFile property*), 479
RESERVED (*hexrec.formats.srec.SrecTag attribute*), 389

S

save() (*hexrec.base.BaseFile method*), 60
save() (*hexrec.formats.asciihex.AsciiHexFile method*), 108
save() (*hexrec.formats.avr.AvrFile method*), 161
save() (*hexrec.formats.ihex.IhexFile method*), 208
save() (*hexrec.formats.mos.MosFile method*), 262
save() (*hexrec.formats.raw.RawFile method*), 313
save() (*hexrec.formats.srec.SrecFile method*), 369
save() (*hexrec.formats.titxt.TiTxtFile method*), 427
save() (*hexrec.formats.xtek.XtekFile method*), 479
seek() (*hexrec.utils.SparseMemoryIO method*), 527
seekable() (*hexrec.utils.SparseMemoryIO method*), 528
serialize() (*hexrec.base.BaseFile method*), 61
serialize() (*hexrec.base.BaseRecord method*), 74
serialize() (*hexrec.formats.asciihex.AsciiHexFile method*), 109
serialize() (*hexrec.formats.asciihex.AsciiHexRecord method*), 122
serialize() (*hexrec.formats.avr.AvrFile method*), 162
serialize() (*hexrec.formats.avr.AvrRecord method*), 174
serialize() (*hexrec.formats.ihex.IhexFile method*), 208
serialize() (*hexrec.formats.ihex.IhexRecord method*), 223
serialize() (*hexrec.formats.mos.MosFile method*), 263
serialize() (*hexrec.formats.mos.MosRecord method*), 276
serialize() (*hexrec.formats.raw.RawFile method*), 314
serialize() (*hexrec.formats.raw.RawRecord method*), 326
serialize() (*hexrec.formats.srec.SrecFile method*), 369

serialize() (*hexrec.formats.srec.SrecRecord method*), 385
serialize() (*hexrec.formats.titxt.TiTxtFile method*), 428
serialize() (*hexrec.formats.titxt.TiTxtRecord method*), 441
serialize() (*hexrec.formats.xtek.XtekFile method*), 480
serialize() (*hexrec.formats.xtek.XtekRecord method*), 496
set_meta() (*hexrec.base.BaseFile method*), 61
set_meta() (*hexrec.formats.asciihex.AsciiHexFile method*), 109
set_meta() (*hexrec.formats.avr.AvrFile method*), 162
set_meta() (*hexrec.formats.ihex.IhexFile method*), 209
set_meta() (*hexrec.formats.mos.MosFile method*), 263
set_meta() (*hexrec.formats.raw.RawFile method*), 314
set_meta() (*hexrec.formats.srec.SrecFile method*), 370
set_meta() (*hexrec.formats.titxt.TiTxtFile method*), 428
set_meta() (*hexrec.formats.xtek.XtekFile method*), 480
shift() (*hexrec.base.BaseFile method*), 62
shift() (*hexrec.formats.asciihex.AsciiHexFile method*), 110
shift() (*hexrec.formats.avr.AvrFile method*), 163
shift() (*hexrec.formats.ihex.IhexFile method*), 209
shift() (*hexrec.formats.mos.MosFile method*), 264
shift() (*hexrec.formats.raw.RawFile method*), 315
shift() (*hexrec.formats.srec.SrecFile method*), 370
shift() (*hexrec.formats.titxt.TiTxtFile method*), 429
shift() (*hexrec.formats.xtek.XtekFile method*), 481
SIZE_TO_ADDRESS_FORMAT (*in module hexrec.formats.srec*), 339
skip_data() (*hexrec.utils.SparseMemoryIO method*), 528
skip_hole() (*hexrec.utils.SparseMemoryIO method*), 529
SparseMemoryIO (*class in hexrec.utils*), 512
split() (*hexrec.base.BaseFile method*), 63
split() (*hexrec.formats.asciihex.AsciiHexFile method*), 111
split() (*hexrec.formats.avr.AvrFile method*), 163
split() (*hexrec.formats.ihex.IhexFile method*), 210
split() (*hexrec.formats.mos.MosFile method*), 264
split() (*hexrec.formats.raw.RawFile method*), 316
split() (*hexrec.formats.srec.SrecFile method*), 371
split() (*hexrec.formats.titxt.TiTxtFile method*), 430
split() (*hexrec.formats.xtek.XtekFile method*), 482
SrecFile (*class in hexrec.formats.srec*), 339
SrecRecord (*class in hexrec.formats.srec*), 375
SrecTag (*class in hexrec.formats.srec*), 388
START_16 (*hexrec.formats.srec.SrecTag attribute*), 389
START_24 (*hexrec.formats.srec.SrecTag attribute*), 389
START_32 (*hexrec.formats.srec.SrecTag attribute*), 389
START_LINEAR_ADDRESS (*hexrec.formats.ihex.IhexTag attribute*), 227
START_SEGMENT_ADDRESS (*hexrec.formats.ihex.IhexTag attribute*), 227
startaddr (*hexrec.formats.ihex.IhexFile property*), 210
startaddr (*hexrec.formats.srec.SrecFile property*), 371
startaddr (*hexrec.formats.xtek.XtekFile property*), 482
SUFFIX_SCALE (*in module hexrec.utils*), 509

T

Tag (*hexrec.base.BaseRecord attribute*), 67
Tag (*hexrec.formats.asciihex.AsciiHexRecord attribute*), 115
Tag (*hexrec.formats.avr.AvrRecord attribute*), 167
Tag (*hexrec.formats.ihex.IhexRecord attribute*), 215
Tag (*hexrec.formats.mos.MosRecord attribute*), 268
Tag (*hexrec.formats.raw.RawRecord attribute*), 319
Tag (*hexrec.formats.srec.SrecRecord attribute*), 376
Tag (*hexrec.formats.titxt.TiTxtRecord attribute*), 434
Tag (*hexrec.formats.xtek.XtekRecord attribute*), 487
tell() (*hexrec.utils.SparseMemoryIO method*), 530
TiTxtFile (*class in hexrec.formats.titxt*), 399
TiTxtRecord (*class in hexrec.formats.titxt*), 433
TiTxtTag (*class in hexrec.formats.titxt*), 445
to_bytes() (*hexrec.formats.asciihex.AsciiHexTag method*), 132
to_bytes() (*hexrec.formats.ihex.IhexTag method*), 233
to_bytes() (*hexrec.formats.mos.MosTag method*), 284
to_bytes() (*hexrec.formats.srec.SrecTag method*), 398
to_bytes() (*hexrec.formats.titxt.TiTxtTag method*), 450
to_bytes() (*hexrec.formats.xtek.XtekTag method*), 505
to_bytestr() (*hexrec.base.BaseRecord method*), 75
to_bytestr() (*hexrec.formats.asciihex.AsciiHexRecord method*), 123
to_bytestr() (*hexrec.formats.avr.AvrRecord method*), 174
to_bytestr() (*hexrec.formats.ihex.IhexRecord method*), 224
to_bytestr() (*hexrec.formats.mos.MosRecord method*), 276
to_bytestr() (*hexrec.formats.raw.RawRecord method*), 327
to_bytestr() (*hexrec.formats.srec.SrecRecord method*), 385
to_bytestr() (*hexrec.formats.titxt.TiTxtRecord method*), 442
to_bytestr() (*hexrec.formats.xtek.XtekRecord method*), 497
to_numbers() (*in module hexrec.formats.sqtp*), 335
to_strings() (*in module hexrec.formats.sqtp*), 338
to_tokens() (*hexrec.base.BaseRecord method*), 75
to_tokens() (*hexrec.formats.asciihex.AsciiHexRecord method*), 123

to_tokens() (*hexrec.formats.avr.AvrRecord method*), 175
to_tokens() (*hexrec.formats.ihex.IhexRecord method*), 224
to_tokens() (*hexrec.formats.mos.MosRecord method*), 277
to_tokens() (*hexrec.formats.raw.RawRecord method*), 327
to_tokens() (*hexrec.formats.srec.SrecRecord method*), 386
to_tokens() (*hexrec.formats.titxt.TiTxtRecord method*), 442
to_tokens() (*hexrec.formats.xtek.XtekRecord method*), 497
TOKEN_COLOR_CODES (*in module hexrec.base*), 26
truncate() (*hexrec.utils.SparseMemoryIO method*), 530

U

unhexlify() (*in module hexrec.utils*), 511
update_checksum() (*hexrec.base.BaseRecord method*), 76
update_checksum() (*hexrec.formats.asciihex.AsciiHexRecord method*), 124
update_checksum() (*hexrec.formats.avr.AvrRecord method*), 175
update_checksum() (*hexrec.formats.ihex.IhexRecord method*), 225
update_checksum() (*hexrec.formats.mos.MosRecord method*), 277
update_checksum() (*hexrec.formats.raw.RawRecord method*), 328
update_checksum() (*hexrec.formats.srec.SrecRecord method*), 386
update_checksum() (*hexrec.formats.titxt.TiTxtRecord method*), 443
update_checksum() (*hexrec.formats.xtek.XtekRecord method*), 498
update_count() (*hexrec.base.BaseRecord method*), 76
update_count() (*hexrec.formats.asciihex.AsciiHexRecord method*), 124
update_count() (*hexrec.formats.avr.AvrRecord method*), 176
update_count() (*hexrec.formats.ihex.IhexRecord method*), 225
update_count() (*hexrec.formats.mos.MosRecord method*), 278
update_count() (*hexrec.formats.raw.RawRecord method*), 328
update_count() (*hexrec.formats.srec.SrecRecord method*), 387
update_count() (*hexrec.formats.titxt.TiTxtRecord method*), 443
update_count() (*hexrec.formats.xtek.XtekRecord method*), 498
update_records() (*hexrec.base.BaseFile method*), 63
update_records() (*hexrec.formats.asciihex.AsciiHexFile method*), 111
update_records() (*hexrec.formats.avr.AvrFile method*), 164
update_records() (*hexrec.formats.ihex.IhexFile method*), 211
update_records() (*hexrec.formats.mos.MosFile method*), 265
update_records() (*hexrec.formats.raw.RawFile method*), 316
update_records() (*hexrec.formats.srec.SrecFile method*), 372
update_records() (*hexrec.formats.titxt.TiTxtFile method*), 430
update_records() (*hexrec.formats.xtek.XtekFile method*), 483

V

validate() (*hexrec.base.BaseRecord method*), 77
validate() (*hexrec.formats.asciihex.AsciiHexRecord method*), 125
validate() (*hexrec.formats.avr.AvrRecord method*), 176
validate() (*hexrec.formats.ihex.IhexRecord method*), 226
validate() (*hexrec.formats.mos.MosRecord method*), 278
validate() (*hexrec.formats.raw.RawRecord method*), 329
validate() (*hexrec.formats.srec.SrecRecord method*), 387
validate() (*hexrec.formats.titxt.TiTxtRecord method*), 444
validate() (*hexrec.formats.xtek.XtekRecord method*), 499
validate_records() (*hexrec.base.BaseFile method*), 64
validate_records() (*hexrec.formats.asciihex.AsciiHexFile method*), 112
validate_records() (*hexrec.formats.avr.AvrFile method*), 164
validate_records() (*hexrec.formats.ihex.IhexFile method*), 212
validate_records() (*hexrec.formats.mos.MosFile method*), 266
validate_records() (*hexrec.formats.raw.RawFile method*), 317
validate_records() (*hexrec.formats.srec.SrecFile method*), 373
validate_records() (*hexrec.formats.titxt.TiTxtFile method*), 431

validate_records() (*hexrec.formats.xtek.XtekFile method*), 483
view() (*hexrec.base.BaseFile method*), 64
view() (*hexrec.formats.asciihex.AsciiHexFile method*), 112
view() (*hexrec.formats.avr.AvrFile method*), 165
view() (*hexrec.formats.ihex.IhexFile method*), 212
view() (*hexrec.formats.mos.MosFile method*), 266
view() (*hexrec.formats.raw.RawFile method*), 317
view() (*hexrec.formats.srec.SrecFile method*), 374
view() (*hexrec.formats.titxt.TiTxtFile method*), 431
view() (*hexrec.formats.xtek.XtekFile method*), 484

W

writable() (*hexrec.utils.SparseMemoryIO method*), 531
write() (*hexrec.base.BaseFile method*), 65
write() (*hexrec.formats.asciihex.AsciiHexFile method*), 113
write() (*hexrec.formats.avr.AvrFile method*), 166
write() (*hexrec.formats.ihex.IhexFile method*), 213
write() (*hexrec.formats.mos.MosFile method*), 267
write() (*hexrec.formats.raw.RawFile method*), 318
write() (*hexrec.formats.srec.SrecFile method*), 374
write() (*hexrec.formats.titxt.TiTxtFile method*), 432
write() (*hexrec.formats.xtek.XtekFile method*), 484
write() (*hexrec.utils.SparseMemoryIO method*), 531
writelines() (*hexrec.utils.SparseMemoryIO method*), 533

X

XtekFile (*class in hexrec.formats.xtek*), 451
XtekRecord (*class in hexrec.formats.xtek*), 485
XtekTag (*class in hexrec.formats.xtek*), 500
xxd_core() (*in module hexrec.xxd*), 535