fixedrec Documentation

Release 1.0.3

Jun 15, 2017

Contents

Documentation is available at: http://fixedrec.readthedocs.org/

CHAPTER 1

Installing from PyPI

This is what you want if you just want to use fixedrec:

```
pip install fixedrec
```

CHAPTER 2

As a source package

This is what you want if you are developing fixedrec or want to make local changes to the source code.

```
pip install -e <path>
```

CHAPTER 3

Developing fixedrec

Note: if you're using this as a template for new projects, remember to *python setup.py register <projectname>* before you upload to PyPi.

Uploading to PyPI

- only source distribution:

```
python setup.py sdist upload
```

- source and windows installer:

```
python setup.py sdist bdist_wininst upload
```

- source, windows, and wheel installer:

```
python setup.py sdist bdist_wininst bdist_wheel upload
```

- create a documentation bundle to upload to PyPi:

```
cd build/sphinx/html && zip -r ../../../pypi-docs.zip *
```

Running tests

One of:

```
python setup.py test
py.test fixedrec
python runtests.py
```

with coverage (one of):

```
py.test --cov=.
python runtests.py --cov=.
coverage run runtests.py && coverage report
```

Building documentation

```
python setup.py build_sphinx
```

CHAPTER 4

API documentation

RecordFile

class fixedrec.fixedrec.**RecordFile**(*fname*, *blocksize=4*, *overwrite=False*)
Low level fixed-size record file. Record numbers are zero-based.

close()
Close file and write statusrec.

count()
Return number of records in the fils.

create_new_file(*fname*)
Create a new keystore file (overwriting/deleting any existing file.

Args: fname (str): Filename to use.

Returns: (file): The created file.

curpos
Return the current position.

goto_first_record()
Go to start of first record.

goto_last_record()
Go to the starting position of the last record.

goto_recnum(*n*)
Position the file at record *n*, if *n* == -1, then go to the end of the file (writing then becomes appending). It is possible to go past the end of the file and write a record, causing the file to grow.

goto_recnum_relative(*n*)
Advance forward or backward (negative *n*) *n* records.

open(*fname*, *overwrite*)
Open or create the file.

open_existing_file(*fname*)
Open the file for read and write if possible.

read()
Read a block at the current position.

swap(*a*, *b*)
Swap records at positions *a* and *b*.

truncate(*recnum=None*)
Truncate the file at *recnum* (ie. *recnum* will be gone).

write(*data*, *flush=True*)
Write data to file at current position.

exception fixedrec.fixedrec.**RecordFileError**
Base Exception for block files.

Layout

This module contains enough knowledge about *struct* format strings to figure out size and position of fields.

class fixedrec.layout.**Field**(***kw*)
Representation of a field defined in a *struct* format string. (should only be created from *Layout*.

get_value(*data*)
Get this field's value from *data*.

set_value(*data*, *value*)
Set this field to *value* in *data*.

class fixedrec.layout.**Layout**(*layout*, **names*, ***kw*)
Record layout.

Usage:

```
record_layout = Layout(
     '=12x?3sQ16s16s68s128sHcc',
     'pad',
     'local',
     'rectype',
     'timestamp',
     'salt',
     'digest',
     'key',
     'data',
     'chksum',
     'cr',
     'nl',
     name="Record"
 )
```

layoutre = <_sre.SRE_Pattern object>
regex matching one field in a struct format string

record_prefix = {'!': 'network-endian', '@': 'native aligned', '=': 'native', '<': 'little-endian', '>': 'big-endian'}
legal struct format string record prefixes

split(*data*)
Split the byte string *data* into a list of substrings holding the data for each field.

struct_field_sizes = {'Q': 8, 'c': 1, 'b': 1, 'd': 8, 'P': 4, 'f': 4, 'i': 4, 'h': 2, 'l': 4, 'p': 1, 'L': 4, 'q': 8, 'I': 4, 'N': 4, '
byte sizes corresponding to struct character codes

struct_field_types = {'Q': 'unsigned long long', 'c': 'char', 'b': 'signed char', 'd': 'double', 'P': 'void *', 'f': 'float
types corresponding to struct character codes

Record

class fixedrec.record.**Record**(*layout*, *data=None*, ***kw*)
Record base class, providing attribute access and pretty printing.

Usage:

```
@register_record
class StatusRecord(Record):
    RECTYPE = 'ver'
    layout = Layout(
        '=4sQ10sH12xHcc',
        'rectype',
        'timestamp',
        'version',
        'reclen',
        'pad'
        'chksum',
        'cr',
        'nl',
        name="StatusRecord"
    )

    def __init__(self, data=None, **kw):
        super(StatusRecord, self).__init__(
            StatusRecord.layout, data, **kw
        )
        if data is None:
            self.rectype = StatusRecord.RECTYPE
            self.version = '1.0.0'
            self.reclen = len(StatusRecord.layout)
            self.cr = b'\r'
            self.nl = b'\n'
        self.set_checksum()

    def set_checksum(self):
        # checksum of all preceeding fields.
        cksm_field = self._layout['chksum']
        cksm = utils.bsd_checksum(self._data[:cksm_field.position])
        cksm_field.set_value(self._data, cksm)
```

as_dict()
Or at least as close to a dict as we can get and still preserve field order.

parts()
Return data for each field of the layout.

pretty_parts()
Make binary data more readable by converting padding bytes in string fields to underscore (_) and padding fields to star (*).

fixedrec.rectypes.**record_type**(*rtype*)
Return the constructor for the record type.

fixedrec.rectypes.**register_record**(*cls*)
Decorator that add the client record class *cls* to the record type registry (as a constructor).

fixedrec.rectypes.**valid_rectype**(*rtype*)
Is the record type registered with a constructor?

Utility functions

Utility functions.

fixedrec.utils.**n_**(*s*, *replacement='_'*)
Make binary fields more readable.

fixedrec.utils.**pad**(*data*, *size*, *padchar=' '*)
Pad the *data* to exactly length = *size*.

class fixedrec.utils.**pset**(**args*, ***kwds*)
A property set is an OrderedDict with prettier string display (useful when working with record lengths that are wider than your terminal).

fixedrec.utils.**split_fields**(*s*, *sizes*)
Split a string into fields based on field *sizes*.

fixedrec.utils.**split_string**(*s*, **ndxs*)
String sub-class with a split() method that splits a given indexes.

Usage:

```
>>> print split_string('D2008022002', 1, 5, 7, 9)
['D', '2008', '02', '20', '02']
```

This is a Python implementation of the bsd 16 bit checksum algorithm.

The original code and license is listed at the end of this file, and was fetched from revision 225736.

Listed in its own file to preserve original copyright statement and conditions.

fixedrec.bsd_checksum.**bsd_checksum**(*data*)
Implementation of the bsd 16-bit checksum algorithm.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index