Fileseq Documentation

Release 1.3.0

Matthew Chambers

Mar 30, 2017

Contents

fileseq - A simple python library for parsing file sequence strings commonly used in VFX and Animation applications.

CHAPTER 1

fileseq.exceptions module

exceptions - Exception subclasses relevant to fileseq operations.

exception `fileseq.exceptions.`**`FileSeqException`**

Bases: `exceptions.ValueError`

Thrown for general exceptions handled by FileSeq.

exception `fileseq.exceptions.`**`ParseException`**

Bases: *`fileseq.exceptions.FileSeqException`*

Thrown after a frame range or file sequence parse error.

CHAPTER 2

fileseq.filesequence module

filesequence - A parsing object representing sequential files for fileseq.

class `fileseq.filesequence.`**`FileSequence`**(*sequence*)

Bases: `object`

`FileSequence` represents an ordered sequence of files.

Parameters **`sequence`** (`str`) – (ie: dir/path.1-100#.ext)

`__getitem__`(*idx*)

Allows access via index to the underlying `fileseq.frameset.FrameSet`.

Parameters **`idx`** (`int`) – the desired index

Return type int

`__iter__`()

Allow iteration over the path or paths this `FileSequence` represents.

Return type generator

`__len__`()

The length (number of files) represented by this `FileSequence`.

Return type int

`basename`()

Return the basename of the sequence.

Return type str

`dirname`()

Return the directory name of the sequence.

Return type str

`end`()

Returns the end frame of the sequences `fileseq.frameset.FrameSet`. Will return 0 if the sequence has no frame pattern.

Return type int

extension()

Return the file extension of the sequence, including leading period.

Return type str

static **findSequenceOnDisk**(*pattern*)

Search for a specific sequence on disk.

Example

```
>>> findSequenceOnDisk("seq/bar#.exr") # or any fileseq pattern
```

Parameters **pattern** – the sequence pattern being searched for

Return type str

Raises *fileseq.exceptions.FileSeqException* if no sequence is found on disk

static **findSequencesInList**(*paths*)

Returns the list of discrete sequences within paths. This does not try to determine if the files actually exist on disk, it assumes you already know that.

Parameters **paths** – a list of paths

Return type list

static **findSequencesOnDisk**(*dirpath*, *include_hidden=False*)

Yield the sequences found in the given directory.

Parameters

- **dirpath** – directory to scan
- **include_hidden** (*bool*) – if true, show .hidden files as well

Return type list

format(*template='{basename}{range}{padding}{extension}'*)

Return the file sequence as a formatted string according to the given template.

Utilizes the python string format syntax. Available keys include:

- basename - the basename of the sequence.
- extension - the file extension of the sequence.
- start - the start frame.
- end - the end frame.
- length - the length of the frame range.
- padding - the detecting amount of padding.
- inverted - the inverted frame range. (returns "" if none)
- dirname - the directory name.

Return type str

frame(*frame*)

Return a path go the given frame in the sequence. Integer or string digits are treated as a frame number and padding is applied, all other values are passed though.

Example

```
>>> seq.frame(1)
/foo/bar.0001.exr
>>> seq.frame("#")
/foo/bar.#.exr
```

Parameters **frame** – the desired frame number (int/str) or a char to pass through (ie. #)

Return type str

frameRange()

Returns the string formatted frame range of the sequence. Will return an empty string if the sequence has no frame pattern.

Return type str

frameSet()

Return the *fileseq.frameset.FrameSet* of the sequence if specified, otherwise None.

Return type *fileseq.frameset.FrameSet* or None

static **getPaddingChars**(*num*)

Given a particular amount of padding, return the proper padding characters.

static **getPaddingNum**(*chars*)

Given a supported group of padding characters, return the amount of padding.

Parameters **chars** (*str*) – a supported group of padding characters

Return type int

Raises ValueError if unsupported padding character is detected

index(*idx*)

Return the path to the file at the given index.

Parameters **idx** (*int*) – the desired index

Return type str

invertedFrameRange()

Returns the inverse string formatted frame range of the sequence. Will return an empty string if the sequence has no frame pattern.

Return type str

padding()

Return the the padding characters in the sequence.

Return type str

setBasename(*base*)

Set a new basename for the sequence.

Parameters **base** (*str*) – the new base name

Return type None

setDirname(*dirname*)

Set a new directory name for the sequence.

Parameters **dirname** (*str*) – the new directory name

Return type None

setExtension(*ext*)
Set a new file extension for the sequence.

Note: A leading period will be added if none is provided.

Parameters **ext** – the new file extension

Return type None

setExtention(*ext*)
Deprecated: use `setExtension()`.

setFrameRange(*frange*)
Set a new frame range for the sequence.

Parameters **frange** – a properly formatted frame range, as per `fileseq.frameset.FrameSet`

Return type None

setFrameSet(*frameSet*)
Set a new `fileseq.frameset.FrameSet` for the sequence.

Parameters **frameSet** – the new `fileseq.frameset.FrameSet` object

Return type None

setPadding(*padding*)
Set new padding characters for the sequence. i.e. "#" or "@@@" or '%04d', or an empty string to disable range formatting.

Return type None

split()
Split the `FileSequence` into contiguous pieces and return them as a list of `FileSequence` instances.

Return type list

start()
Returns the start frame of the sequence's `fileseq.frameset.FrameSet`. Will return 0 if the sequence has no frame pattern.

Return type int

static yield_sequences_in_list(*paths*)
Yield the discrete sequences within paths. This does not try to determine if the files actually exist on disk, it assumes you already know that.

Parameters **paths** – a list of paths

Return type generator

zfill()
Returns the zfill depth (ie the number of zeroes to pad with).

Return type int

CHAPTER 3

fileseq.frameset module

frameset - A set-like object representing a frame range for fileseq.

class `fileseq.frameset.`**`FrameSet`**(*frange*)

Bases: `_abcoll.Set`

A `FrameSet` is an immutable representation of the ordered, unique set of frames in a given frame range.

The frame range can be expressed in the following ways:

- 1-5
- 1-5,10-20
- 1-100x5 (every fifth frame)
- 1-100y5 (opposite of above, fills in missing frames)
- 1-100:4 (same as 1-100x4,1-100x3,1-100x2,1-100)

A `FrameSet` is effectively an ordered frozenset, with FrameSet-returning versions of frozenset methods:

```
>>> FrameSet('1-5').union(FrameSet('5-10'))
FrameSet('1-10')
>>> FrameSet('1-5').intersection(FrameSet('5-10'))
FrameSet('5')
```

Because a FrameSet is hashable, it can be used as the key to a dictionary:

```
>>> {FrameSet('1-20'): 'good'}
```

Caveats:

1. All frozenset operations return a normalized `FrameSet`: internal frames are in numerically increasing order.
2. Equality is based on the contents and order, NOT the frame range string (there are a finite, but potentially extremely large, number of strings that can represent any given range, only a "best guess" can be made).

3. Human-created frame ranges (ie 1-100x5) will be reduced to the actual internal frames (ie 1-96x5).
4. The "null" `Frameset` (`FrameSet('')`) is now a valid thing to create, it is required by set operations, but may cause confusion as both its start and end methods will raise IndexError. The `is_null()` property has been added to allow you to guard against this.

Parameters **frange** (*str*) – the frame range as a string (ie "1-100x5")

Return type None

Raises `fileseq.exceptions.ParseException` if the frame range (or a portion of it) could not be parsed

__and__(*other*)

Overloads the `&` operator. Returns a new `FrameSet` that holds only the frames *self* and *other* have in common.

Note: The order of operations is irrelevant: `(self & other) == (other & self)`

Return type `FrameSet`, or `NotImplemented` if :param: other fails to convert to a `FrameSet`

__contains__(*item*)

Check if item is a member of this `FrameSet`.

Parameters **item** (*int*) – the frame number to check for

Return type bool

__eq__(*other*)

Check if *self* == *other* via a comparison of the hash of their contents. If *other* is not a `FrameSet`, but is a set, frozenset, or is iterable, it will be cast to a `FrameSet`.

Parameters **other** (`FrameSet`) – Also accepts an object that can be cast to a `FrameSet`

Return type bool, or `NotImplemented` if *other* fails to convert to a `FrameSet`

__ge__(*other*)

Check if *self* >= *other* via a comparison of the contents. If *other* is not a `FrameSet`, but is a set, frozenset, or is iterable, it will be cast to a `FrameSet`.

Parameters **other** (`FrameSet`) – Also accepts an object that can be cast to one a `FrameSet`

Return type bool, or `NotImplemented` if *other* fails to convert to a `FrameSet`

__getitem__(*index*)

Allows indexing into the ordered frames of this `FrameSet`.

Parameters **index** – the index to retrieve

Return type int

Raises `IndexError` if index is out of bounds

__getstate__()

Allows for serialization to a pickled `FrameSet`.

Return type tuple (frame range string,)

__gt__(*other*)

Check if *self* > *other* via a comparison of the contents. If *other* is not a `FrameSet`, but is a set, frozenset, or is iterable, it will be cast to a `FrameSet`.

Note: A `FrameSet` is greater than *other* if the set of its contents are greater, OR if the contents are equal but the order is greater.

Listing 3.1: Same contents, but (1,2,3,4,5) sorts below (5,4,3,2,1)

```
>>> FrameSet("1-5") > FrameSet("5-1")
False
```

Parameters **other** (`FrameSet`) – Also accepts an object that can be cast to a `FrameSet`

Return type bool, or `NotImplemented` if :param: other fails to convert to a `FrameSet`

__hash__()

Builds the hash of this `FrameSet` for equality checking and to allow use as a dictionary key.

Return type int

__iter__()

Allows for iteration over the ordered frames of this `FrameSet`.

Return type generator

__le__(*other*)

Check if *self* <= *other* via a comparison of the contents. If *other* is not a `FrameSet`, but is a set, frozenset, or is iterable, it will be cast to a `FrameSet`.

Parameters **other** (`FrameSet`) – Also accepts an object that can be cast to a `FrameSet`

Return type bool, or `NotImplemented` if *other* fails to convert to a `FrameSet`

__len__()

Returns the length of the ordered frames of this `FrameSet`.

Return type int

__lt__(*other*)

Check if self < other via a comparison of the contents. If other is not a `FrameSet`, but is a set, frozenset, or is iterable, it will be cast to a `FrameSet`.

Note: A `FrameSet` is less than other if the set of its contents are less, OR if the contents are equal but the order of the items is less.

Listing 3.2: Same contents, but (1,2,3,4,5) sorts below (5,4,3,2,1)

```
>>> FrameSet("1-5") < FrameSet("5-1")
True
```

Parameters **other** (`FrameSet`) – Can also be an object that can be cast to a `FrameSet`

Return type bool, or `NotImplemented` if *other* fails to convert to a `FrameSet`

__ne__(*other*)

Check if *self* != *other* via a comparison of the hash of their contents. If *other* is not a `FrameSet`, but is a set, frozenset, or is iterable, it will be cast to a `FrameSet`.

Parameters **other** (`FrameSet`) – Also accepts an object that can be cast to a `FrameSet`

Return type bool, or `NotImplemented` if *other* fails to convert to a `FrameSet`

__or__(*other*)

Overloads the `|` operator. Returns a new `FrameSet` that holds all the frames in *self, other,* or both.

Note: The order of operations is irrelevant: `(self | other) == (other | self)`

Return type `FrameSet`, or `NotImplemented` if *other* fails to convert to a `FrameSet`

__rand__(*other*)

Overloads the `&` operator. Returns a new `FrameSet` that holds only the frames *self* and *other* have in common.

Note: The order of operations is irrelevant: `(self & other) == (other & self)`

Return type `FrameSet`, or `NotImplemented` if :param: other fails to convert to a `FrameSet`

__repr__()

Returns a long-form representation of this `FrameSet`.

Return type str

__reversed__()

Allows for reversed iteration over the ordered frames of this `FrameSet`.

Return type generator

__ror__(*other*)

Overloads the `|` operator. Returns a new `FrameSet` that holds all the frames in *self, other,* or both.

Note: The order of operations is irrelevant: `(self | other) == (other | self)`

Return type `FrameSet`, or `NotImplemented` if *other* fails to convert to a `FrameSet`

__rsub__(*other*)

Overloads the `-` operator. Returns a new `FrameSet` that holds only the frames of *other* that are not in *self.*

Note: This is for right-hand subtraction (`other - self`).

Return type `FrameSet`, or `NotImplemented` if *other* fails to convert to a `FrameSet`

__rxor__(*other*)

Overloads the ^ operator. Returns a new `FrameSet` that holds all the frames in *self* or *other* but not both.

Note: The order of operations is irrelevant: `(self ^ other) == (other ^ self)`

Return type `FrameSet`, or `NotImplemented` if *other* fails to convert to a `FrameSet`.

__setstate__(*state*)

Allows for de-serialization from a pickled `FrameSet`.

Parameters **state** (*tuple, str, or dict*) – A string/dict can be used for backwards compatibility

Return type None

Raises `ValueError` if state is not an appropriate type

__str__()

Returns the frame range string of this `FrameSet`.

Return type str

__sub__(*other*)

Overloads the - operator. Returns a new `FrameSet` that holds only the frames of *self* that are not in *other*.

Note: This is for left-hand subtraction (`self - other`).

Return type `FrameSet`, or `NotImplemented` if *other* fails to convert to a `FrameSet`

__xor__(*other*)

Overloads the ^ operator. Returns a new `FrameSet` that holds all the frames in *self* or *other* but not both.

Note: The order of operations is irrelevant: `(self ^ other) == (other ^ self)`

Return type `FrameSet`, or `NotImplemented` if *other* fails to convert to a `FrameSet`.

copy()

Returns a shallow copy of this `FrameSet`.

Return type `FrameSet`

difference(**other*)

Returns a new `FrameSet` with elements in *self* but not in *other*.

Return type `FrameSet`

end()

The last frame in the `FrameSet`.

Return type int

Raises `IndexError` (with the empty `FrameSet`)

frame(*index*)

Return the frame at the given index.

Parameters **index** (*int*) – the index to find the frame for

Return type int

Raises `IndexError` if index is out of bounds

frameRange (*zfill=0*)

Return the frame range used to create this *FrameSet*, padded if desired.

Example

```
>>> FrameSet('1-100').frameRange()
'1-100'
>>> FrameSet('1-100').frameRange(5)
'00001-00100'
```

Parameters **zfill** (*int*) – the width to use to zero-pad the frame range string

Return type str

static **framesToFrameRange** (*frames*, *sort=True*, *zfill=0*, *compress=False*)

Converts an iterator of frames into a `fileseq.framerange.FrameRange`.

Parameters

- **frames** (*iterable*) – sequence of frames to process
- **sort** (*bool*) – sort the sequence before processing
- **zfill** (*int*) – width for zero padding
- **compress** (*bool*) – remove any duplicates before processing

Return type str

static **framesToFrameRanges** (*frames*, *zfill=0*)

Converts a sequence of frames to a series of padded `fileseq.framerange.FrameRange` s.

Parameters

- **frames** (*iterable*) – sequence of frames to process
- **zfill** (*int*) – width for zero padding

Return type generator

frange

Read-only access to the frame range used to create this *FrameSet*.

Return type frozenset

classmethod **from_iterable** (*frames*, *sort=False*)

Build a *FrameSet* from an iterable of frames.

Parameters

- **frames** – an iterable object containing frames as integers
- **sort** – True to sort frames before creation, default is False

Return type *FrameSet*

hasFrame (*frame*)

Check if the *FrameSet* contains the frame.

Parameters **frame** (*int*) – the frame number to search for

Return type bool

index(*frame*)
Return the index of the given frame number within the `FrameSet`.

Parameters **frame** (*int*) – the frame number to find the index for

Return type int

Raises `ValueError` if frame is not in self

intersection(**other*)
Returns a new `FrameSet` with the elements common to *self* and *other*.

Return type `FrameSet`

invertedFrameRange(*zfill=0*)
Return the inverse of the `FrameSet` 's frame range, padded if desired. The inverse is every frame within the full extent of the range.

Example

```
>>> FrameSet('1-100x2').invertedFrameRange()
'2-98x2'
>>> FrameSet('1-100x2').invertedFrameRange(5)
'00002-00098x2'
```

Parameters **zfill** (*int*) – the width to use to zero-pad the frame range string

Return type str

static **isFrameRange**(*frange*)
Return True if the given string is a frame range. Any padding characters, such as '#' and '@' are ignored.

Parameters **frange** (*str*) – a frame range to test

Return type bool

is_null
Read-only access to determine if the `FrameSet` is the null or empty `FrameSet`.

Return type bool

isdisjoint(*other*)
Check if the contents of :class:self has no common intersection with the contents of :class:other.

Return type bool, or `NotImplemented` if *other* fails to convert to a `FrameSet`

issubset(*other*)
Check if the contents of *self* is a subset of the contents of *other*.

Return type bool, or `NotImplemented` if *other* fails to convert to a `FrameSet`

issuperset(*other*)
Check if the contents of *self* is a superset of the contents of *other*.

Return type bool, or `NotImplemented` if *other* fails to convert to a `FrameSet`

items
Read-only access to the unique frames that form this `FrameSet`.

Return type frozenset

normalize()
Returns a new normalized (sorted and compacted) `FrameSet`.

Return type `FrameSet`

order

Read-only access to the ordered frames that form this *FrameSet*.

Return type tuple

static padFrameRange(*frange*, *zfill*)

Return the zero-padded version of the frame range string.

Parameters **frange** (*str*) – a frame range to test

Return type str

start()

The first frame in the *FrameSet*.

Return type int

Raises IndexError (with the empty *FrameSet*)

symmetric_difference(*other*)

Returns a new *FrameSet* that contains all the elements in either *self* or *other*, but not both.

Return type *FrameSet*

union(**other*)

Returns a new *FrameSet* with the elements of *self* and of *other*.

Return type *FrameSet*

Python Module Index

f

Index

Symbols

B

C

D

E

F

G

H

I

N

O

P

S

U

Y

Z