JSONStreams Documentation

Release 0.4.1

Dylan Baker

January 18, 2017

Contents

CHAPTER 1

Description

JSONstreams is a package that attempts to making writing JSON in a streaming format easier. In contrast to the core json module, this package doesn't require building a complete tree of dicts and lists before writing, instead it provides a straightforward way to to write a JSON document **without** building the whole data structure ahead of time.

JSONstreams considers there to be two basic types, the JSON array and the JSON object, which correspond to Python's list and dict respectively, and can encode any types that the json.JSONEncoder can, or can use an subclass to handle additional types.

The interface is designed to be context manger centric. The Stream class, and the Array and Object classes returned by the subarray and subobject methods (respectively), can be used as context managers or not, but use as context managers are recommended to ensure that each container is closed properly.

Contents:

1.1 Examples

These are examples of how you might use this library

1.1.1 Basic

As an object with a filename:

```python
import jsonstreams

with jsonstreams.Stream(jsonstreams.Type.object, filename='foo') as f:
    f.write('foo', 1)
    with f.subobject('bar') as b:
        b.iterwrite((str(s), s) for s in range(5))
    with f.subarray('oink') as b:
        b.write('foo')
        b.write('bar')
        b.write('oink')
```

As an array with an fd:

```python
import bz2

import jsonstreams

with bz2.open('foo') as f:
    with jsonstreams.Stream(jsonstreams.Type.array, fd=f) as s:
        s.write('foo')
        s.write('bar')
        with s.subobject() as b:
            b.write('foo', 'bar')
        with s.subarray() as b:
            b.write('x')
            b.write('y')
            b.write('z')
        s.write('oink')
```

1.1.2 Customizing the encoder

The encoder can be customized to allow complex types to be passed in without having to convert them into types that json.JSONEncoder can natively understand. It can be done by subclassing the JSONEncoder, but this isn't recommended by simplejson, instead it is better to pass a function to the `json.JSONencoder()` 's default parameter. This is easily achieved by using a `functools.partial()`.

Warning: It is critical that you do not pass a value for indent, as the *Stream* class sets this value internally.

```
from functools import partial
from json import JSONEncoder

def my_encoder(self, obj):
    # Turn sets into lists so they can be encoded
    if isinstance(obj, set):
        return list(obj)
    return obj

with jsonstreams.Stream(jsonstreams.Type.object, filename='foo',
                        encoder=partial(JSONEncoder, default=my_encoder)):
    s.write('foo', {'foo', 'bar'})
```

1.2 Public API

1.2.1 Overview

The main component is `jsonstreams.Stream`, which provides the interface for either an array or an object. The interfaces for this class depend on whether it was initialized as an array or an object.

1.2.2 Exceptions

exception **`JsonStreamsError`**(*message*)

A base exception class for other JSONstreams errors.

Parameters **`message`** (*str (python 3) or unicode (python 2)*) – the message to be displayed with the exception is raised

exception **`ModifyWrongStreamError`**(*message*)

An exception raised when trying to modify on object within the stream which is not in focus.

Because JSON is so strictly defined, and this module writes out all data into the stream immediately without building any intermediate data structures, it is impossible to write into a parent while a sub-stream is opened. This exception will be raised in that case.

It is not advised to handle this exception, it is almost certainly a programming error.

Parameters **`message`** (*`str`*) – the message to be displayed with the exception is raised

```
with jsonstreams.Stream(Type.object, filename='foo') as s:
    with s.subobject('bar') as b:
        s.write('foo', 'bar')
ModifyWrongStreamError
```

exception **`InvalidTypeError`**(*message*)

An exception that is raised when an invalid type is passed for an argument. Primarily this will be raised from the *`Object.write()`* and *`Object.iterwrite()`* methods.

JSON is pretty particularly about what kinds of values can be used as keys for objects, only text type is allowed, not lists, objects, null or numeric types. JSONstreams does not attempt to coerce values for the developer, instead it raises this exception.

It is not advised to handle this exception, it is almost certainly a programming error.

Parameters **`message`** (*`str (python 3) or unicode (python 2)`*) – the message to be displayed with the exception is raised

```
with jsonstreams.Stream(Type.object, filename='foo') as s:
    with s.subobject(1) as b:
InvalidTypeError
```

exception **`StreamClosedError`**(*message*)

An exception that is raised when trying to write into an *`Object`* or *`Array`* after the close() method has already been called.

Parameters **`message`** (*`str (python 3) or unicode (python 2)`*) – the message to be displayed with the exception is raised

```
with jsonstreams.Stream(Type.object, filename='foo') as s:
    with s.subobject(1) as b:
        b.write('foo', 'bar)
    b.write('foo', 'bar)
StreamClosedError
```

1.2.3 Classes

class **`Type`**

This is an enum that provides valid types for the Stream class.

`object`

A JSON object

`array`

A JSON array

class **`Stream`**(*jtype*, *filename=none*, *fd=none*, *indent=0*, *pretty=false*, *encoder=json.JSONencoder*)

The stream class is the basic entry point for using JSONstreams, and is the only class meant to be instantiated directly. When initialized this class will add the methods of *`Object`* or *`Array`*, as matches the value of jtype.

It can be initialized with either a filename, which it will open via `open()`, or a file-like object already opened for write, but not both.

It also takes and indent argument, which will cause the writer to add the appropriate white space to the output. For especially large documents this may help decode, as some parsers have a limit on the number of characters per line.

A pretty flag can be passed, which will further cause indents to be consistently written even for complex objects, which would normally not be set at the same base indent level as other objects. This can have a negative effect on performance.

This class can also be used as a context manager (used with the with statement), which will automatically call the *Stream.close()* method when exiting the context.

```
with jsonwriter.Stream(jsonstreams.Type.array, filename='foo') as s:
    s.write('foo')
```

Parameters

- **jtype** (Type) – A value of the *Type* enum.
- **filename** (*str or None*) – If set this will be opened and the stream written into it.
- **fd** (*file*) – A file-like object defining a write and close method.
- **indent** (*int*) – The number of spaces before each level in the JSON document.
- **pretty** (*bool*) – Whether or not to indent complex objects.
- **encoder** (*json.JSONEncoder*) – A callable that will create a json.JSONEncoder instance.

write()

This method will differ in signature depending on whether jtype is Type.array or Type.object.

If Type.array then this method is an alias for *Array.write()*. If Type.'object then this method is an alias for *Object.write()*.

iterwrite()

This method will differ in signature depending on whether jtype is Type.object or Type.array.

If Type.array then this method is an alias for *Array.iterwrite()*. If Type.object then this method is an alias for *Object.iterwrite()*.

close()

This method will close the root object by calling either *Object.close()* or *Array.close()*, and will also close the file.

subobject()

This method will differ in signature depending on whether jtype is Type.object or Type.array.

This method will open a new object in the stream by calling either *Object.subobject()* or *Array.subobject()*

subarray()

This method will differ in signature depending on whether jtype is Type.object or Type.array.

This method will open a new array in the stream by calling either *Object.subarray()* or *Array.subarray()*

class **Object**

The Object constructor is not considered a public API, and is not documented here because it is not guaranteed according to the Semantic Versioning standard. All other public methods, however are considered public API.

This class represents an object in a JSON document. It provides as public API all of the methods necessary to write into the stream and to close it. Like the `Stream` it provides a context manager, and can be used as a context manager, including when called from the `Object.subobject()` or `Array.subobject()`.

subobject(*key*)

Open a new sub-object within the current object stream.

Parameters **key** (`str`) – When written this will be the key and the new object will be the value

Returns The sub-object instance.

Return type `Object`

Raises

- ***InvalidTypeError*** – if the key is not a str
- ***ModifyWrongStreamError*** – if this stream is not the top of the stack
- ***StreamClosedError*** – if `Object.close()` has been called

subarray(*key*)

Open a new sub-array within the current object stream.

Parameters **key** (`str`) – When written this will be the key and the new Array will be the value

Returns The sub-array instance.

Return type `Array`

Raises

- ***InvalidTypeError*** – if the key is not a str
- ***ModifyWrongStreamError*** – if this stream is not the top of the stack
- ***StreamClosedError*** – if `Object.close()` has been called

write(*key*, *value*)

Write a key:value pair into the object stream.

Parameters

- **key** (`str`) – The key value.
- **value** (Any type that can be encoded by the encoder argument of `Stream`) – The value to be written.

Raises

- ***InvalidTypeError*** – If the key is not str
- ***ModifyWrongStreamError*** – if this stream is not the top of the stack
- ***StreamClosedError*** – if `Object.close()` has been called

iterwrite(*args*)

Write key:value pairs from an iterable.

One should not use this for dumping a complete dictionary or list, unless doing transformations. This is intended to have a generator passed into it.

```
with jsonstreams.Stream(Type.object, filename='foo') as s:
    s.iterwrite((str(s), s) for s in range(5))
```

param args An iterator returning key value pairs

type value An iterable of tuples where the key is str and the value is any type that can be encoded by the encoder argument of `Stream`

raises InvalidTypeError If the key is not str

raises ModifyWrongStreamError if this stream is not the top of the stack

raises StreamClosedError if `Object.close()` has been called

close()

Close the current object.

Once this is called any call to `write()`, `iterwrite()`, `subobject()`, or `subarray()` will cause an `StreamClosedError` to be raised.

class **Array**

The Array constructor is not considered a public API, and is not documented here because it is not guaranteed according to the Semantic Versioning standard. All other public methods are considered public API.

This class represents an array in a JSON document. It provides as public API all of the methods necessary to write into the stream and to close it. Like the `Stream` it provides a context manager, and can be used as a context manager, including when called from the `Object.subarray()` or `Array.subarray()`.

subobject()

Open a new sub-object within the current array stream.

Returns The sub-object instance.

Return type `Object`

Raises

- **`ModifyWrongStreamError`** – if this stream is not the top of the stack
- **`StreamClosedError`** – if `Object.close()` has been called

subarray()

Open a new sub-array within the current array stream.

Returns The sub-array instance.

Return type `Array`

Raises

- **`ModifyWrongStreamError`** – if this stream is not the top of the stack
- **`StreamClosedError`** – if `Object.close()` has been called

write(*value*)

Write a value into the array stream.

Parameters **value** (Any type that can be encoded by the encoder argument of `Stream`) – The value to be written.

Raises

- **`ModifyWrongStreamError`** – if this stream is not the top of the stack
- **`StreamClosedError`** – if `Object.close()` has been called

iterwrite(*args*)

Write values into an array from an iterator.

One should not use this for dumping a complete list unless doing transformations. This is intended to have a generator passed into it.

```
with jsonstreams.Stream(Type.object, filename='foo') as s:
    s.iterwrite(range(10, step=2))
```

param args An iterator returning key value pairs

type value An iterable of tuples where the key is str and the value is any type that can be encoded by the encoder argument of `Stream`

raises ModifyWrongStreamError if this stream is not the top of the stack

raises StreamClosedError if `Object.close()` has been called

close()
Close the current object.

Once this is called any call to `write()`, `iterwrite()`, `subobject()`, or `subarray()` will cause an `StreamClosedError` to be raised.

1.3 Changes

1.3.1 0.4.1

New Features

- Bump from alpha to beta quailty. The public API will no longer change in a non-backwards compatible way without a very good reason.
- Add support for Python 3.6

1.3.2 0.4.0

New Features:

- Use an enum rather than a string to set the object type in Stream. For python < 3.4 this adds a new requirement on enum34

Bug Fixes:

- Fix numerous typos and errors in the sphinx documentation

1.3.3 0.3.2

New Features:

- Proper documentation via readthedocs

1.3.4 0.3.1

New Features:

- Add __slots__ to the Writer classes

Bug Fixes:

- Fix a bug with both ObjectWriter and ArrayWriter with pretty printing, in which the comma property wouldn't be properly set. (#12)
- Fix bug with ObjectWriter and pretty printing. (#11)

1.3.5 0.3.0

New features:

- Allow passing a filename or an already opened fd to the Stream class. (#4)
- Add typing stub files. (#6)
- Add iterwrite methods. These allow writing generators and iterators without creating an in memory datastructure. (#8)

1.3.6 0.2.1

Bug Fixes:

- Pass the pretty flag down in the Stream class correctly. This bug was more of an annoyance than anything else. (#7)

1.3.7 0.2.0

New features:

- Added a pretty printer flag. This allows printing complex object with the expected level of indent, but with added overhead. (#3)

Bug Fixes:

- Pass the indent value to the encoder of the writer, which means complex objects get indented. The value may not be what is expected without the pretty flag. (#2)
- Invalid types can no longer be passed as keys to Object.write. (#1)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index