hickleable Documentation

Steven Murray

Feb 01, 2024

CONTENTS

A simple decorator to make your classes hickle-able.

CHAPTER

ONE

WHAT IS THIS?

`hickleable` provides a simple decorator for your classes that will almost always make them serilalize well using the excellent hickle package. By default, custom classes are not supported by `hickle` – instead, they are written to the HDF5 file as a binary dataset that is serialized using the standard Python `pickle`. This obviously negates much of the benefit of `hickle`, for example, the fact that pickle-serialized data is only readable using Python.

`hickle` provides a way to serialize your custom classes using the HDF5 format, via defining a few hooks for loading/dumping. However, it can be a little tricky to implement these hooks, as they are quite general.

`hickleable` provides a "default implementation" of these hooks that should satisfy the requirements of most custom classes, and can be applied as a simple decorator. This makes it a one-liner to transform your class into a well-supported *data format*.

Check out the docs at ReadTheDocs.

1.1 Installation

Simply `pip install hickleable`. Conda-installable dependencies include `h5py`.

1.2 Usage

Simply:

```
from hickleable import hickleable

@hickleable()
class MyClass:
   def __init__(self, a=1, b='foo', c={'a': 'dict'}):
       self.a = a
       self.b = b
       self.c = c
```

Now, `MyClass` can be hickled without any pickling:

```
import hickle

my_obj = MyClass()
hickle.dump(my_obj, 'temporary_file.h5')  # Note: no warnings about having to pickle
new_obj = hickle.load('temporary_file.h5')
```

One super cool thing is that `@cached_property` attributes are respected, and dataclasses are also supported:

```
from dataclass import dataclass
from functools import cached_property

@hickleable()
@dataclass
class CachedClass:
    foo: str
    bar: int

    @cached_property
    def foobar(self) -> str:
        print("Obtaining foobar...")
        return foo*bar

c = CachedClass('baz', 50000)

foobar = c.foobar  # prints "Obtaining foobar..."
foobar = c.foobar  # prints nothing, since it's returning cached value.

hickle.dump(c, 'foobar.h5')
d = hickle.load('foobar.h5')

d_foobar = d.foobar  # prints nothing! The value is cached in the hickle file.
```

One thing to note is that the cached properties are only saved in the hickle file if they have already been evaluated. To force `hickle` to write out all cached properties, use the `evaluate_cached_properties=True` parameter in the call to `hickleable()`.

1.2.1 Customizing Dumping/Loading

While `hickleable` will automatically render most classes hickle-able, there are bound to be corner cases in which constituent attributes are not themselves hickleable, or other concerns that you will want to customize. While all of this is of course totally customizable by using the dumping/loading hooks from `hickle`, the `hickleable` decorator also respects the magic methods `__gethstate__` and `__sethstate__`, which act exactly like `__getstate__` and `__setstate__` do for pickling. In fact, if the latter exist and the former don't, the latter will be used to serialize the object in `hickle`. For instance, let's say you have a class that keeps track of the number of times it is called in its lifecycle:

```
@hickleable()
class Counter:
    def __init__(self, a):
        self.a = a
        self._counts = 0

    def __call__(self, b):
        self._counts += 1
        self.a *= b
```

If we make an instance and call it a few times, the `_counts` attribute is larger than zero. If we save the object to a hickle file and load it back up somewhere else, it will *start* with `_counts > 0`. We can avoid this as follows:

```
def ignore_counts(self, state: dict):
    state['_counts'] = 0
    self.__dict__.update(state)

Counter.__setstate__ = ignore_counts
```

We could also have removed `_counts` entirely from the hickle file:

```
def remove_counts(self) -> dict:
    return {k: v for k, v in self.__dict__.items() if k != '_counts'}

Counter.__gethstate__ = remove_counts
```

Note that since we set `ignore_counts` to be the `__setstate__` method, it will be respected both for `hickle` **and** `pickle`. We set `remove_counts` as the `__gethstate__` method, which means it will only be respected for `hickle`.

1.3 License

1.4 Tutorials

After reading the basic usage, a good place to get a feel for how `hickleable` works is by following some tutorials.

If you've covered the tutorials and still have questions about "how to do stuff" in `hickleable`, consult the FAQs:

1.4.1 Miscellaneous FAQs

Can I eat soup for breakfast?

You should not.

1.5 API Reference