fhirdatetime

Mike Mabey

Apr 27, 2021

CONTENTS

1 Installation **3**

2 Usage **5**
- 2.1 Creation 5
- 2.2 Comparison 5
- 2.3 Sorting 6

3 License **7**
- 3.1 `FhirDateTime` API 7

A `datetime`-compatible class for FHIR date/datetime values.

The FHIR specification from HL7 is "a standard for health care data exchange." The FHIR spec includes date and datetime data types that provide more flexibility than the standard Python `date` and `datetime` types. This makes sense when you consider a patient may report to their provider that they have experience a particular symptom since a particular year without knowing the month or day of onset.

CHAPTER

ONE

INSTALLATION

Install `fhirdatetime` using pip:

```
pip install fhirdatetime
```

CHAPTER

TWO

USAGE

2.1 Creation

The `FhirDateTime` class is designed to be used to store date/datetime values from FHIR payloads (which are JSON strings), you can create instances from `str` values:

```
>>> FhirDateTime("2021-03-15")
fhirdatetime.FhirDateTime(2021, 3, 15)
```

You can also convert native `date` and `datetime` objects directly:

```
>>> FhirDateTime(date(2021, 3, 15))
fhirdatetime.FhirDateTime(2021, 3, 15)
>>> FhirDateTime(datetime(2021, 3, 15, 20, 54))
fhirdatetime.FhirDateTime(2021, 3, 15, 20, 54)
```

One purpose of this library is to allow flexibility in granularity without sacrificing the ability to compare (using <, >, ==, etc.) against objects of the same type as well as native `date` and `datetime` objects.

2.2 Comparison

When comparing objects, only the values that are populated for *both* objects are considered. Consider the following examples in which only the years are compared:

```
>>> FhirDateTime(2021) == FhirDateTime(2021, 3, 15)
True
>>> FhirDateTime(2021) == datetime(2021, 3, 15, 23, 56)
True
>>> FhirDateTime(2021) == date(2021, 3, 15)
True
>>> FhirDateTime(2021) < FhirDateTime(2021, 3, 15)
False
>>> FhirDateTime(2021) > FhirDateTime(2021, 3, 15)
False
```

2.3 Sorting

Important: When there is ambiguity due to one `FhirDateTime` object storing less-granular data than another (e.g., `FhirDateTime(2021)` vs. `FhirDateTime(2021, 4)`), objects with missing values will be ordered *before* those with more granular values that would otherwise be considered equivalent when using the == operator.

When you need to sort a sequence of either `FhirDateTime` objects or object that *contain* a `FhirDateTime` object, the `FhirDateTime.sort_key()` function will make it easier to sort the items properly.

There are two ways to use this function. The first is intended for use when sorting a sequence of `FhirDateTime` objects, something like this (notice that `sort_key()` is called with no parameters):

```
>>> sorted(
...     [FhirDateTime(2021, 4), FhirDateTime(2021), FhirDateTime(2021, 4, 12)],
...     key=FhirDateTime.sort_key()
... )
[FhirDateTime(2021), FhirDateTime(2021, 4), FhirDateTime(2021, 4, 12)]
```

The second is for use when sorting a sequence of objects that have `FhirDateTime` objects as attributes. This example sorts the `CarePlan`[1] objects by the care plan's period's start date:

```
>>> sorted(care_plan_list, key=FhirDateTime.sort_key("period.start"))
```

In this example, `sorted()` passes each item in `care_plan_list` to the `sort_key` static method, which first gets the `period` attribute of the item, then gets the `start` attribute of the period. Finally, the year, month, day, and other values are returned to `sorted()`, which does the appropriate sorting on those values.

If neither of these use cases of the `sort_key()` function apply to what you need to do, you can always use a custom lambda to do your sorting. For example, the following is equivalent to the care plan sorting example:

```
>>> sorted(care_plan_list, key=lambda x: FhirDateTime.sort_key(x.period.start))
```

[1] Take a look at the `fhir.resources` definition of a CarePlan here to get a better idea of what is going on in the example.

CHAPTER

THREE

LICENSE

This project is licensed under the MIT license.

3.1 `FhirDateTime` API