statscraper Documentation

Release 1.0.7

Jens Finnäs and Leo Wallentin, J++; Robin Linderborg

Jun 13, 2019

Contents:

Statscraper provides a common set of guidelines, base classes and standards for writing scrapers for Swedish agencies' websites. Scrapers that comply with these standards provide a unified abstraction layer to the end-user, in terms of both usage and data output.

CHAPTER 1

About Statscraper

Statscraper is a base library for building web scrapers for statistical data, with a helper ontology for (primarily Swedish) statistical data. A set of ready-to-use scrapers are included. With Statscraper comes a unified interface for scraping, and some useful helper methods for scraper authors.

Statscraper is developed by Jens Finnäs and Leo Wallentin from Journalism++, and Robin Linderborg from SVT Nyheter.

The first stable version was released in August 2017. Statscraper is sponsored by Internetfonden/Stiftelsen för internetinfrastruktur and Journalism++ Stockholm.

CHAPTER 2

Using scrapers

Every scraper built on Statscraper shares the same interface towards the user. Here's sample code using one of the included demo scrapers, to fetch the number of cranes spotted at Hornborgarsjön each day from Länsstyrelsen i Västra Götalands län:

```
>>> from statscraper.scrapers import Cranes

>>> scraper = Cranes()
>>> scraper.items  # List available datasets
[<Dataset: Number of cranes>]

>>> dataset = scraper["Number of cranes"]
>>> dataset.dimensions
[<Dimension: date (Day of the month)>, <Dimension: month>, <Dimension: year>]

>>> row = dataset.data[0]  # first row in this dataset
>>> row
<Result: 7 (value)>
>>> row.dict
{'value': '7', u'date': u'7', u'month': u'march', u'year': u'2015'}
>>> row.int
7
>>> row.tuple
('7', {u'date': u'7', u'month': u'march', u'year': u'2015'})

>>> df = dataset.data.pandas  # get this dataset as a Pandas dataframe
```

2.1 Exploring sites

Scrapers act like "cursors" that move around a hierarchy of datasets and collections of datasets. Collections and datasets are refered to as "items".

```
       Collection  Collection  Dataset
ROOT  Collection  Dataset      Dataset
       Collection   Dataset      Dataset
                      Dataset

_____________________________________________

                     items
```

The cursor is moved around the item tree as needed when you access properties or data, but you can also move manually around the items, if you want to be in full control. Some scrapers, e.g. those that need to fill out and post forms, or handle session data, might require that you move the cursor around manually. For most simple scrapers, e.g. those accessing an API, this should not be necessary.

Moving the cursor manually:

```
>>> from statscraper.scrapers import PXWeb

>>> scraper = PXWeb(base_url="http://pxnet2.stat.fi/pxweb/api/v1/sv/StatFin/")
>>> scraper.items
[<Collection: tym (Arbetsmarknaden)>, <Collection: vrm (Befolkning)>, ...]

>>> scraper.move_to("vrm").move_to("synt").move_to("080_synt_tau_203.px")
>>> scraper.current_item
<Dataset: 080_synt_tau_203.px (Befolkningsförändringar efter område 1980 - 2016)>

>>> scraper.move_up()
>>> scraper.current_item
<Collection: synt (Födda)>
>>> scraper.move_to("010_synt_tau_101.px")
>>> scraper.current_item
<Dataset: 010_synt_tau_101.px (Summerat fruktsamhetstal för åren 1776 - 2016)>

>>> scraper.move_to_top()
>>> scraper.move_to(0)  # Moving by index works too
```

The datasets above could also be accessed like this:

```
>>> from statscraper.scrapers import PXWeb

>>> scraper = PXWeb(base_url="http://pxnet2.stat.fi/pxweb/api/v1/sv/StatFin/")

>>> collection = scraper["vrm"]["synt"]
>>> collection
<Collection: synt (Födda)>

>>> dataset_1 = collection["080_synt_tau_203.px"]
>>> dataset_2 = collection["010_synt_tau_101.px"]
```

At any given point, `scraper["foo"]` is shorthand for `scraper.current_item.items["foo"]`.

If you want to loop throuh every available dataset a scraper can offer, there is a `Scraper.descendants` property that will recursively move to every item in the tree. Here is an example, that will find all datasets in the SCB API that has monthly data:

```
>>> from statscraper.scrapers import SCB

>>> scraper = SCB()
>>> for dataset in scraper.descendants:
```

```
>>>     if dataset.dimensions["Tid"].label == u"månad":
>>>         print "Ahoy! Dataset %s has monthly data!" % dataset
```

2.2 Exploring datasets

Much like itemslists (`Collection.items`), datasets are only fetched when you are inspecting or interacting with them.

The actual data is stored in a property called data:

```
>>> from statscraper.scrapers import Cranes

>>> scraper = Cranes()
>>> dataset = scraper.items[0]
>>> for row in dataset.data:
>>>     print "%s cranes were spotted on %s" % (row.value, row["date"])
```

The data property will hold a list of result objects. The list can be converted to a few other formats, e.g. a pandas dataframe:

```
>>> from statscraper.scrapers import Cranes

>>> scraper = Cranes()
>>> dataset = scraper.items[0]
>>> df = dataset.data.pandas  # convert to pandas dataframe
```

If you want to querry a site or database for some subset of the available data, you can use the `fetch()` method on the dataset (or on the scraper, to fetch data from the current position, if any):

```
>>> dataset = scraper.items[0]
>>> data = dataset.fetch(query={'year': "2017"})
```

or

```
>>> scraper.move_to(0)
>>> data = scraper.fetch(query={'year': "2017"})
```

Available dimensions can be inspected though the .dimensions property:

```
>>> dataset.dimensions
[<Dimension: date>, <Dimension: year>]
```

Note however that a scraper does not necessarily need to provide dimensions. If `Dataset.dimensions` is None, it could simply mean that the scraper itself is not sure what to expect from the data.

2.3 Dialects

Scraper authors can use the included `Datatypes` module to have a standardised ontology for common statistical dimensions. If a dimensions uses a bulid in datatype, it can be translated to a different dialect. For instance, Swedish municipalities come in the following dialects:

- `short`: `"Ale"`
- `numerical`: `"1440"`

- wikidata: "Q498470"
- brå: "8617"
- scb: "1440 Ale kommun"

By default, Statscraper prefers human readable representations, and municipality values is internally stored like this: u"Borås kommun". The philosophy here is that human readable id's speed up debugging and makes it easy to spot errors during scraping and analysis. Yes, we do use Unicode for id's. It's 2017 after all.

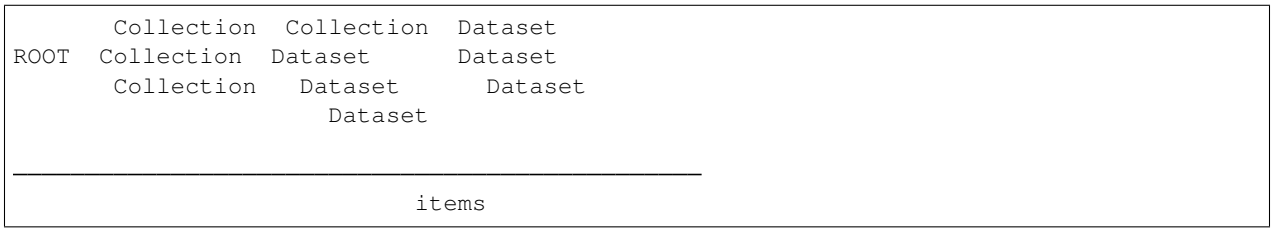
```
>>> from statscraper.scrapers import Cranes

>>> scraper = Cranes()
>>> data = scraper.items[0].data
>>> row = data[0]
>>> row["month"]
<DimensionValue: march (month)>
>>> row["month"].translate("swedish")
u'mars'
```

For available datatypes, domains, values and dialects, see the statscraper-datatypes repo.

CHAPTER 3

Developing scrapers

The scraper can navigate though an hierarchy of collections and datasets. Collections and datasets are refered to as "items".



Scrapers are built by extending the BaseScraper class, or a subclass of it. Every scraper must override the methods `_fetch_itemslist` and `_fetch_data`:

- `_fetch_itemslist(self, item)` must yield items at the current position.
- `_fetch_data(self, dataset, query)` must yield rows from a dataset.

Other methods that a scraper can chose to override are:

- `_fetch_dimensions(self, dataset)` should yield dimensions available on a dataset.
- `_fetch_allowed_values(self, dimension)` should yield allowed values for a dimension.

A number of hooks are avaiable for more advanced scrapers. These are called by adding the on decorator on a method:

```
@BaseScraper.on("up")
def my_method(self):
  # Do something when the cusor moves up one level
```

Check out the statscraper/scrapers directory for some scraper examples.

Below if the full code for the CranesScraper scraper used in the chapter Using Scrapers:

```
# encoding: utf-8
""" A scraper to fetch daily cranes sightings at Hornborgasjön
    from http://web05.lansstyrelsen.se/transtat_O/transtat.asp
```

```
    This is intended to be a minimal example of a scraper
    using Beautiful Soup.
"""
import requests
from bs4 import BeautifulSoup
from statscraper import BaseScraper, Dataset, Dimension, Result


class Cranes(BaseScraper):

    def _fetch_itemslist(self, item):
        """ There is only one dataset. """
        yield Dataset("Number of cranes")

    def _fetch_dimensions(self, dataset):
        """ Declaring available dimensions like this is not mandatory,
         but nice, especially if they differ from dataset to dataset.

         If you are using a built in datatype, you can specify the dialect
         you are expecting, to have values normalized. This scraper will
         look for Swedish month names (e.g. 'Januari'), but return them
         according to the Statscraper standard ('january').
        """
        yield Dimension(u"date", label="Day of the month")
        yield Dimension(u"month", datatype="month", dialect="swedish")
        yield Dimension(u"year", datatype="year")

    def _fetch_data(self, dataset, query=None):
        html = requests.get("http://web05.lansstyrelsen.se/transtat_O/transtat.asp").
↪text
        soup = BeautifulSoup(html, 'html.parser')
        table = soup.find("table", "line").find_all("table")[2].findNext("table")
        rows = table.find_all("tr")
        column_headers = rows.pop(0).find_all("td", recursive=False)
        years = [x.text for x in column_headers[2:]]
        for row in rows:
            cells = row.find_all("td")
            date = cells.pop(0).text
            month = cells.pop(0).text
            i = 0
            for value in cells:
                # Each column from here is a year.
                if value.text:
                    yield Result(value.text.encode("utf-8"), {
                        "date": date,
                        "month": month,
                        "year": years[i],
                    })
                i += 1
```

3.1 Hooks

Some scrapers might need to execute certains tasks as the user moves around the items tree. There are a number of hooks, that can be used to run code as a respons to an event. A scraper class method is attached to a hook by using the `BaseScraper.on` decorator, with the name of the hook as the only argument. Here is an example of a hook in a

Selenium based browser, used to refresh the browser each time the end user navigates to the top-most collection.

```
@BaseScraper.on("top")
def refresh_browser(self):
    """ Refresh browser, to reset all forms """
    self.browser.refresh()
```

Available hooks are:

- init: Called when initiating the class
- up: Called when trying to go up one level (even if the scraper failed moving up)
- top: Called when moving to top level
- select: Called when trying to move to a specific Collection or Dataset. The target item will be provided as an artgument to the function.

CHAPTER 4

API Documentation

Documentation of statscraper's public API.

4.1 Main Interface

class statscraper.**BaseScraper**(**args*, ***kwargs*)
The base class for scrapers.

children
Former, misleading name for descendants.

descendants
Recursively return every dataset below current item.

fetch(*query=None*, ***kwargs*)
Let the current item fetch it's data.

items
ItemList of collections or datasets at the current position.

None will be returned in case of no further levels

move_to(*id_*)
Select a child item by id (str), reference or index.

move_to_top()
Move to root item.

move_up()
Move up one level in the hierarchy, unless already on top.

classmethod on(*hook*)
Hook decorator.

parent
Return the item above the current, if any.

path
All named collections above, including the current, but not root.

class statscraper.**BaseScraperList**
Lists of dimensions, values, etc all inherit this class for some common convenience methods, such as get_by_label()

get(*key*)
Provide alias for bracket notation.

class statscraper.**BaseScraperObject**
Objects like items, dimensions, values etc all inherit this class. BaseScraperObjects are typicalliy stored in a BaseScraperList.

class statscraper.**Collection**(*id_*, *label=None*, *blob=None*)
A collection can contain collection of datasets.

Lorem ipsum lorem lorem ipsum lorem. Dummy text.

Basic Usage:

```
>>> from statscraper import Collection
>>> c = Collection()
<class 'statscraper.base_scraper.Collection'>
```

class statscraper.**Dataset**(*id_*, *label=None*, *blob=None*)
A dataset. Can be empty.

class statscraper.**Dimension**(*id_=None*, *label=None*, *allowed_values=None*, *datatype=None*, *dialect=None*, *domain=None*)
A dimension in a dataset.

class statscraper.**DimensionList**
A one dimensional list of dimensions.

class statscraper.**DimensionValue**(*value*, *dimension*, *label=None*)
The value for a dimension inside a Resultset.

class statscraper.**Item**(*id_*, *label=None*, *blob=None*)
Common base class for collections and datasets.

class statscraper.**Result**(*value*, *dimensions={}*)
A "row" in a result.

A result contains a numerical value, and optionally a set of dimensions with values.

class statscraper.**ResultSet**
The result of a dataset query.

This is essentially a list of Result objects.

class statscraper.**ValueList**
A list of dimension values.

allowed_values uses this class, to allow checking membership.

4.2 Exceptions

class statscraper.exceptions.**DatasetNotInView**
Tried to operate on a dataset that is not visible.

This can be raised by a scraper if the cursor needs to move before inspecting an item.

class statscraper.exceptions.**InvalidData**
The scraper encountered some invalid data.

class statscraper.exceptions.**InvalidID**
This string is not allowed as an id at this point. Note: Inherits from Exception instead of StandardError for Python3.x compatibility reasons.

class statscraper.exceptions.**NoSuchDatatype**
No datatype with that id.

class statscraper.exceptions.**NoSuchItem**
No such Collection or Dataset.

Index