Saving Scraped Data To JSON With Scrapy Feed Exporters

You've built a spider that will scrape data from a website, now you want to save it somewhere. One of the easiest ways to save scrape data is to save it to a JSON file.

In this guide, we will go through how:

First, let's go over what are **Scrapy Feed Exporters**.

What Are Scrapy Feed Exporters?

The need to save scraped data to a file is a very common requirement for developers, so to make our lives easier the developers behind Scrapy have implemented Feed Exporters.

Feed Exporters are a ready made toolbox of methods we can use to easily save/export our scraped data into:

- JSON & JSON lines file format
- CVS file format
- XML file format
- Pythons pickle format

And save them to:

- The local machine Scrapy is running on
- A remote machine using FTP (file transfer protocall)
- Amazon S3 Storage
- Google Cloud Storage
- Standard output

In this guide, we will walk you through the different ways you can save JSON files from Scrapy.

Saving In JSON & JSON Lines Format

When saving in JSON format, we have two options:

- JSON
- JSON lines

Storing data in JSON format is okay for small amounts of data but it doesn't scale well for large amounts of data, as incremental (aka. stream-mode) parsing is not well supported (if at all) and can result in the entire dataset being stored into memory creating the potential for a memory leak.

JSON data is held memory in an array and new data is appended to it:

```
[
    {"name": "Color TV", "price": "1200"},
    {"name": "DVD player", "price": "200"}
]
```

As a result, it is advised to use JSON lines format if you want to save data in JSON.

```
{"name": "Color TV", "price": "1200"}
{"name": "DVD player", "price": "200"}
```

Using JSON lines allows new data to be incrementally added to a file and can be split into numerous chunks.

Saving JSON Files Via The Command Line

The first and simplest way to create a JSON file of the data you have scraped, is to simply define a output path when starting your spider in the command line.

To save to a JSON file add the flag `-o` to the `scrapy crawl` command along with the file path you want to save the file to.

You can set a relative path like below:

```
scrapy crawl bookspider -o bookspider_data.json

scrapy crawl bookspider -o bookspider_data.jsonl
```

Or you can also set a absolute path like this:

```
scrapy crawl bookspider -o file:///path/to/my/project/bookspider_data.json
```

You have two options when using this command, use are small `-o` or use a capital `-O`.

Flag	Description
`-o`	Appends new data to an existing file.
`-O`	Overwrites any existing file with the same name with the current data.

Telling Scrapy to save the data to a JSON via the command line is okay, but can be a little messy. The other option is setting it in your code, which Scrapy makes very easy.

Saving JSON Files With Feeds Setting

Often the cleanest option is to tell Scrapy to save the data to a JSON via the FEEDS setting.

We can configure it in our `settings.py` file by passing it a dictionary with the path/name of the file and the file format.

For JSON format:

```
# settings.py

FEEDS = {
    'data.json': {'format': 'json'}
}
```

For JSON lines format:

```
# settings.py

FEEDS = {
    'data.jsonl': {'format': 'jsonlines'}
}
```

You can also configure this in each individual spider by setting a `custom_setting` in your spider.

```
# bookspider.py

import scrapy
from proxy_waterfall.items import BookItem

class BookSpider(scrapy.Spider):
        name = 'bookspider'
        start_urls = ["http://books.toscrape.com"]

        custom_settings = {
                'FEEDS': { 'data.jsonl': { 'format': 'jsonlines',}}
                }

        def parse(self, response):

                for article in response.css('article.product_pod'):
                        book_item = BookItem(
                                url = article.css("h3 > a::attr(href)").get(),
                                title = article.css("h3 > a::attr(title)").extract_first(),
                                price = article.css(".price_color::text").extract_first(),
                        )
                        yield book_item
```

The default overwriting behaviour of the **FEEDS** functionality is dependant on where the data is going to be stored. However, you can set it to overwite existing data or not by adding a `overwrite` key to the `FEEDS` dictionary with either **True** or **False**.

```
# settings.py
```

```
FEEDS = {
    'data.jsonl': {'format': 'jsonlines', 'overwrite': True}
}
```

When saving locally, by default `overwrite` is set to **False**. The full set of defaults can be found in the Feeds docs.

1. Setting Dynamic File Paths/Names

Setting a static filepath is okay for development or very small projects, however, when in production you will likely don't want all your data being saved into one big file. So to solve this Scrapy allows you create dynamic file paths/names using spider variables.

For example, here tell create a JSON file for the data in the data folder, followed by the subfolder with the spiders name, and a file name that includes the spider name and date it was scraped.

```
# settings.py

FEEDS = {
    'data/%(name)s/%(name)s_%(time)s.jsonl': {
        'format': 'jsonlines',
        }
}
```

The generated path would look something like this.

```
"data/bookspider/bookspider_2022-05-18T07-47-03.jsonl"
```

Any other named parameter gets replaced by the spider attribute of the same name. For example, `%(site_id)s` would get replaced by the `spider.site_id` attribute the moment the feed is being created.

2. Configuring Extra Functionality

The **Feeds** functionality has other settings that you can configure by passing key/value pairs to the `FEEDS` dictionary you define.

Key	Description
`encoding`	The encoding to be used for the feed. If unset or set to None (default) it uses UTF-8 for everything except JSON output, which uses safe numeric encoding (\uXXXX sequences) for historic reasons.
`fields`	A list of fields to export, allowing you to only save certain fields from your Items.
`item_classes`	A list of item classes to export. If undefined or empty, all items are exported.
`item_filter`	A filter class to filter items to export. `ItemFilter` is used be default.
`indent`	Amount of spaces used to indent the output on each level.
`store_empty`	Whether to export empty feeds (i.e. feeds with no items).
`uri_params`	A string with the import path of a function to set the parameters to apply with printf-style string formatting to the feed URI.
`postprocessing`	List of plugins to use for post-processing.
`batch_item_count`	If assigned an integer number higher than 0, Scrapy generates multiple output files storing up to the specified number of items in each output file. Docs

An example `FEED` setting use multiple of these would be:

```
# settings.py

FEEDS = {
    'data/%(name)s/%(name)s_%(time)s.jsonl': {
        'format': 'jsonlines',
        'encoding': 'utf8',
        'store_empty': False,
        'item_classes': [MyItemClass1, 'myproject.items.MyItemClass2'],
        'fields': None,
        'indent': 4,
        'item_export_kwargs': {
            'export_empty_fields': True,
        },
        }
}
```

Saving Data To Multiple JSON File Batches

Depending on your job, you may want to store the scraped data in numerous file batches instead of in one large JSON lines file to make it more managable. Scrapy makes it very easy to do this with the `batch_item_count` key you can set in your **FEEDS** settings.

Simply set add the `batch_item_count` key to your Feed settings and set the number of Items you would like in each file. This will then start a new JSON file when it reaches this limit.

Note: You will also need to add at least one of the following placeholders in the feed URI to indicate how the different output file names are generated:

Placehold	Description
`%(batch_time)s`	Inserts a timestamp when the batch is being created
`%(batch_id)d`	Inserts a 1-based sequence number of the batch.

For example, these Feed settings will break the data up into numerous batches of equal size (except the last batch).

```
# settings.py

FEEDS = {
    'data/%(name)s/%(name)s_batch_%(batch_id)d.jsonl': {
        'format': 'jsonlines',
        'batch_item_count': 10,
        }
}
```

The resulting batch files, with 10 rows in each.

```
"data/bookspider/bookspider_batch_1.jsonl"
"data/bookspider/bookspider_batch_2.jsonl"
"data/bookspider/bookspider_batch_3.jsonl"
"data/bookspider/bookspider_batch_4.jsonl"
"data/bookspider/bookspider_batch_5.jsonl"
"data/bookspider/bookspider_batch_6.jsonl"
```

If you would like to learn more about saving data, then be sure to check out these guides:

- Saving Data to CSV
- Saving Data to SQLite Database
- Saving Data to MySQL Database
- Saving Data to Postgres Database
- Saving CSV/JSON Files to Amazon AWS S3 Bucket

If you would like to learn more about Scrapy in general, then be sure to check out The Scrapy Playbook.