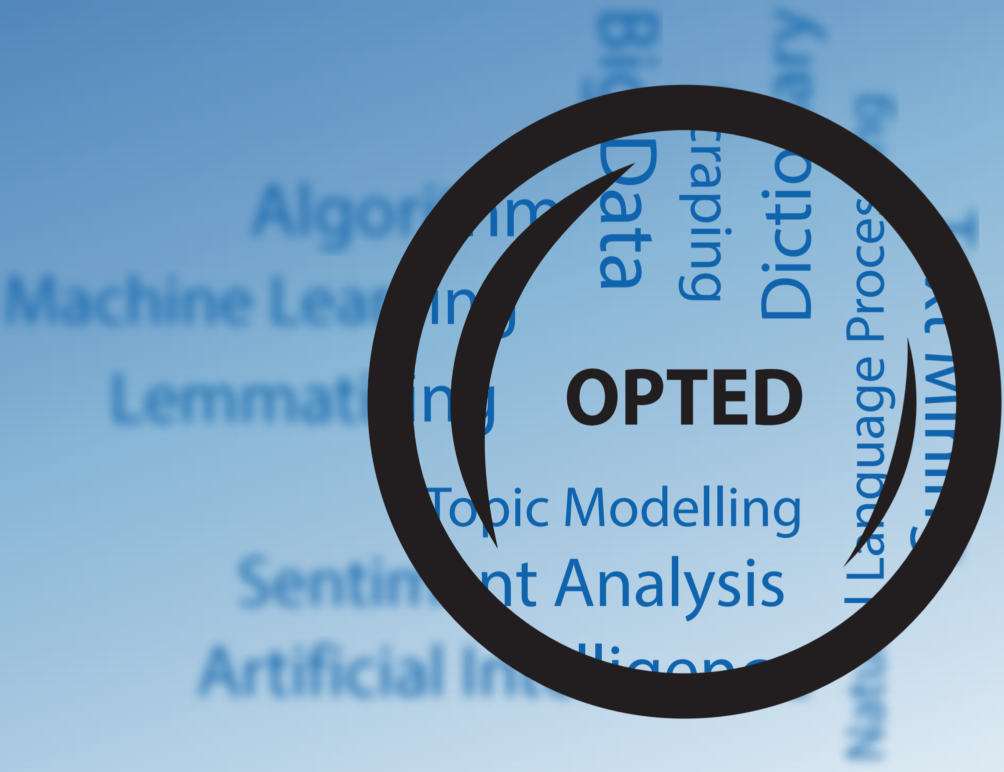
OPTED

Online learning materials

Deliverable 3.4

Paul Balluff, Marvin Stecker, Hajo G. Boomgaarden, and Annie Waldherr

University of Vienna



Disclaimer

This project has received funding from the European Union's Horizon 2020 research & innovation programme under grant agreement No 951832. The document reflects only the authors' views. The European Union is not liable for any use that may be made of the information contained herein.

Dissemination level

Public

Type

Report

OPTED
Observatory for Political Texts in European Democracies:
Designing a European research infrastructure

Online learning materials

Deliverable 3.4

Authors: Paul Balluff, Marvin Stecker, Hajo G. Boomgaarden, and Annie Waldherr

University of Vienna

Due date: March 2023

Contents

Executive Summary

Deliverable 3.4 presents a blueprint for online learning materials in the OPTED research infrastructure. We showcase new training materials that we created and made available at https://tutorials.opted.eu, as well as a collection of third-party materials. This deliverable also explains the workflow for creating new training materials as well as gathering existing materials.

1 Introduction

Work Package 9 conducted a survey for assessing the training and research needs among the user community of the OPTED platform. The findings in D9.3 suggest that the majority of scholars learn computational text-analysis techniques via online learning materials. However, the survey also shows that the availability and quality of materials pose major obstacles for scholars who have not used computational text analysis yet, but who would be interested in doing so. Furthermore, there is also a need for materials for audiences at all levels of experiences (beginner, intermediate, and expert).

With this deliverable, we partially address these needs and also show a blueprint for future learning materials. We showcase the OPTED Tutorial website as well as a curated collection of third-party tutorials. We propose to integrate the OPTED authored learning materials in the OPTED platform and link to third party-materials.

We describe our new online learning materials that includes an outreach and training event held by WP3 in February 2023. The webinar focused on introducing the *Meteor* (see D3.2) platform to more researchers, explaining the design and usage of it, and strengthen the community of *Meteor* users.

In the final sections, we present three tutorials that are also available on the tutorials website. The tutorials aim to both teach specific skills and also act as a showcase for a research workflow involving *Meteor*. In the first tutorial we show first-time users how to leverage *Meteor* (see D3.2). It is an introduction to *Meteor*, established by OPTED in 2022, and it shows how *Meteor* can be used to complement web-scraping projects. The other two tutorials focus on specific python modules for retrieving media text data via the WordPress API and Telegram messenger.

2 Tutorial Website

We created a website dedicated for hosting online learning materials (https://tutorials.opted.eu). It serves as a blueprint for a research infrastructure, and showcases what a workflow for creating and disseminating learning materials can look like.

We use *Quarto*[1] as content management system. *Quarto* is a static site generator, which means that the source material for the webpages is stored in a plain text format which are then compiled into various output formats. The content management system can not only output the plain texts to a website, but also generate PDFs or other formats. The plain text format of the source material also makes it simpler to migrate the content to

[1]https://quarto.org/

other management systems in the future as needed. Of course, there are also other static site generators available, such as Hugo[2] or Jekyll[3]. We chose *Quarto* because it is designed to include executable code blocks. This means that the authors of tutorials can write code in R or Python and *Quarto* executes the code on compilation. This feature is especially useful for generating plots or dealing with dynamically generated data.

The plain text files alongside the styling information is stored in the OPTED Github repository[4]. This allows cooperation among authors, integration between Work Packages, version control, as well as reacting to user feedback. If authors contribute a new tutorial, the repository maintainers can assess the tutorial's quality before it becomes published on the website. GitHub also allows for discussions for new contributions (so-called "pull requests") which facilitates the communication between the repository maintainers and the tutorial authors.

Beyond the technical backbone of the tutorial website, we also propose a set of quality criteria (or minimum standards) that new learning materials should meet:

Experience level & requirements The learning material should explicitly state the experience level of the target group. Users should see at a glance whether the material is adequate for their skill level. Additionally, the technical requirements and previous knowledge should be listed. For example, if a tutorial assumes previous knowledge of a programming language or a specific software package, then it should be stated explicitly. Installation guides should also be provided.

High level introduction The learning material should start with a high level overview that is free of technical jargon. The goals of the material should be mentioned as well as the key learnings outcomes.

Coherence The learning material should be internally coherent. For example, after completing a tutorial, the reader should have a fully working example that serves as a template for other applications.

Core concepts The learning material clearly explains core concepts (theoretical or technical). If a software package is introduced, then the most important functions (or methods) should be explained. If a theoretical construct is used, then the construct should be explained or point to the respective resources.

Facilitate Self-Learning If there are additional materials available (e.g., official documentations, user guides, or related tutorials), the author should point the user towards them.

Sample data If data is required to complete the learning material, then the author should make sample data available to the user via the OPTED platform.

Screenshots & visualisations The authors are encouraged to include visual materials in their learning materials where possible.

[2]https://gohugo.io/
[3]https://jekyllrb.com/
[4]https://github.com/opted-eu/tutorial_website

Additionally, we propose to link tutorials to the wider OPTED infrastructure by tagging learning materials with a taxonomy. Analogous to the meta-variables of the tool collection (D3.3), learning materials would also be tagged according to operations (e.g., API access, or text cleaning) and conceptual categories (e.g., sentiment analysis, populism detection, ideological positioning of texts) where applicable. The linkage could also go even further, by connecting learning materials with specific data sources (e.g. WP5 parliamentary speeches), or languages (e.g., specific text analysis steps for Hebrew). With such a tagging system, the users of the learning materials could be pointed towards more available resources in the OPTED platform. Additionally, it would enable to discover blank spots where more materials are needed.

3 Third-Party Materials

In D3.3 we presented a collection of tools for text analysis. In some cases the authors of tools point their readers towards tutorials or other learning materials. We included such information in *Meteor*, so that users can not only view meta-information on the tool (see D3.3), but also have a collection of associated learning materials. In the context of a European research infrastructure, we propose that authors of tutorials and other learning materials can add them to the OPTED platform. The authors can either just link to their self-hosted tutorial, or they can also choose to publish their materials on tutorials.opted.eu. The first option gives tutorial authors freedom and flexibility, the latter option takes away the burden of finding a webspace to host and publish their materials. In any case, the OPTED platform should be user-centric in this regard, which means that the users should be able to find high-quality, learning materials to various tools and text-analysis techniques. Materials listed on the OPTED platform should also be tagged (see D3.3), so that users can not only search for training materials based on tools, but also based on operations (e.g., data scraping, deep learning, or part-of-speech tagging) or concepts (e.g., sentiment, frames, or populism).

We show here a curated selection of learning materials for tools (D3.3) that meet the quality criteria outlined above:

AntConc A freeware corpus analysis toolkit for concordancing and text analysis. The developer of the tool created detailed video tutorials[5], where each video is between 5 and 15 minutes and focuses on one specific operation that AntConc can perform.

corpustools The R-package offers various tools for anayzing text corpora. It offers features ranging from corpus management tools such as pre-processing, subsetting, Boolean (Lucene) queries and deduplication, to analysis techniques such as corpus comparison, document comparison, semantic network analysis and topic modeling. It provides a tutorial[6] for all package features alongside with sample data.

eMFDscore a library for the fast and flexible extraction of various moral information metrics from textual input data. The developers provide a beginner friendly tutorial[7]

[5]https://www.youtube.com/playlist?list=PLiRIDpYmiC0R3Vv5NncOuIqaUcyLLW7Ae
[6]https://cran.r-project.org/web/packages/corpustools/vignettes/corpustools.html
[7]https://github.com/medianeuroscience/emfdscore/blob/master/eMFDscore_Tutorial.ipynb

where the core concepts are explained and sample data is provided.

pretext An R package to assess the consequences of text preprocessing decisions. The package authors provide a getting started guide[8], where they introduce the package features with two sample datasets.

quanteda A framework for quantitative text analysis in R. *quanteda* is a good example for a software package with a larger support structure. The authors provide a rich tutorial website[9] as well as a designated space on stackoverflow where users can ask questions[10] specifically about *quanteda*. Authors of quanteda extensions can also add them to the official tutorial website. For example, newsmap[11] that provides a semi-supervised model for geographical document classification.

rsyntax R-package that provides Various functions for querying and reshaping dependency trees, as for instance created with the `spacyr` or `udpipe` packages. This enables the automatic extraction of useful semantic relations from texts, such as quotes (who said what) and clauses (who did what). A theoretical introduction alongside a walkthrough is available[12] where each function of the package is explained.

vosonSML the R package is a suite of easy to use functions for collecting and generating different types of networks from social media data. The package supports the collection of data from twitter, youtube and reddit, as well as hyperlinks from web sites. The tool developers provide a website with detailed documentation and tutorials[13]

Every tool and learning material listed above is also included in Meteor.

4 Training Event

To encourage community involvement for the *Meteor* platform, as well as further promote it's usage in scientific research to academics, WP3 organised a webinar on Zoom to showcase the capabilities of the inventory. The webinar was promoted via Twitter, generating nearly 10,000 impressions and 32 sign ups. On the 28th February 2023, 14 viewers tuned in to engage with the presentation of WP3.

Marvin Stecker, for WP1, introduced the audience to the overall goals of the OPTED project. He highlighted the ambition of the project and the necessity for a text analysis infrastructure to strengthen political communication research. Briefly discussing relevant outputs, he pointed to the various Work Packages and tasks within the consortium: the needs they address, how researchers might make use of them, and how they interlink within OPTED and are used in the project. Amongst them are the different repositories for text sources, but also work on methodological challenges and validation, as well as skills deficits and tailored training.

[8] http://www.mjdenny.com/getting_started_with_preText.html
[9] https://tutorials.quanteda.io/
[10] https://stackoverflow.com/questions/tagged/quanteda
[11] https://meteor.opted.eu/view/Tool/newsmap
[12] https://github.com/vanatteveldt/rsyntax/blob/master/Querying_dependency_trees.pdf
[13] https://vosonlab.github.io/vosonSML/

Paul Balluff then introduced the *Meteor* platform in more detail. He began by laying out the necessity of an overview of media sources and text analysis tools, and how the lack of comparable resources hinders research. He also pointed towards a detailed conceptual work as outlined in D3.1, D3.2 and Balluff et al. (2022).

In a live-demonstration, Balluff explained the user interface of the landing page and demonstrated how to perform queries. He showed various detail views of the different entry types (news source, organization, tool, country, meta variables, etc.). Next, the process of adding new entries was shown on example cases.

Balluff highlighted the community aspect of *Meteor.* It is a freely accessible web resource, open to be used by anyone without requiring a registration. However, a (privacy-focused) user system is implemented to manage user contributions. Signing up is done quickly, and then enables one to add new content to the repository. Experts for a particular's country media system might add or correct information on missing media sources to widen the scope of *Meteor.* Others might develop a software package that works with text data. Extensive guidance exists to help first-time users through this process. The integration of various APIs in the background also helps pull relevant information for the user, e.g. from the CRAN software repository or through a DOI identifier, further lessening the work load for individual contributors. Community involvement strengthens *Meteor*, so Balluff invited attendees of the webinar to identify how they could add information to the repository.

Furthermore, Balluff noted that *Meteor* is still under active development and new features are planned to be added until the end of the project period. Attendees' questions focused on the scope of the repository and the considerations behind the concepts and review system, to which Balluff explained the conception of the plattform and community system (see the preceding Deliverables).

The OPTED team lastly highlighted that work is ongoing on various aspects of the project, encouraging attendees to keep an eye on the project's social media presence or website to receive further updates in the future.

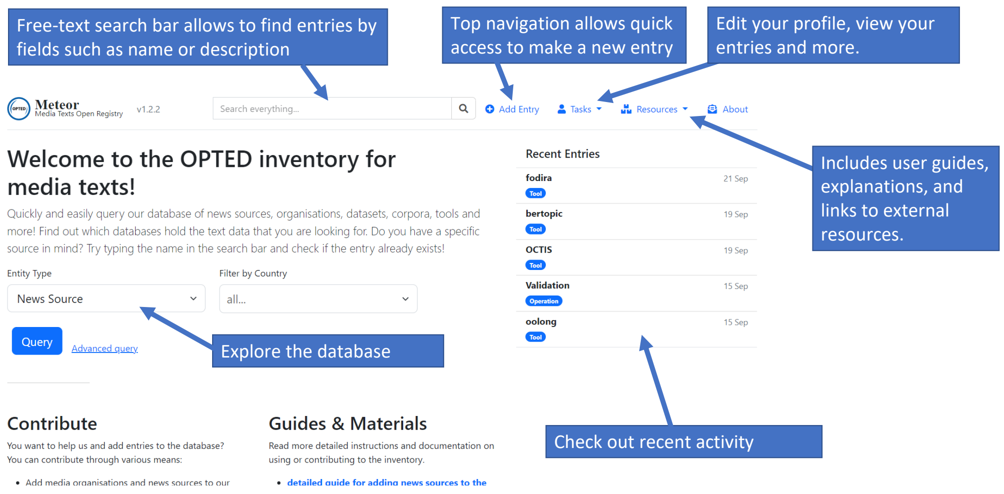
5 Tutorials

For sake of completeness and archival purposes, we attach the tutorials that we created for tutorials.opted.eu. Please note that the collection of tutorials has not been completed yet, since we are aiming to integrate the output of other Work Packages on the tutorial website as well.

5.1 A Quick Tour through Meteor

OPTED *Meteor* (Media Text Open Registry) is a comprehensive and curated platform where researchers can search for text analysis tools, news sources, media organizations, data archives, and corpora. The platform allows you not only to browse, query, and download entries, but also make contributions to the platform. With your help, we are keeping *Meteor* dynamic and adaptable. You can to quickly find ways to access news media data as well as suitable text analysis tools and other digital resources. Figure 1 shows the landing page of the platform and provides an overview of the most important features.

Figure 1: Landing page of the platform


Free-text search bar allows to find entries by fields such as name or description

Top navigation allows quick access to make a new entry

Edit your profile, view your entries and more.

Includes user guides, explanations, and links to external resources.

Explore the database

Check out recent activity

Meteor
Media Texts Open Registry
v1.2.2

Search everything...

Add Entry · Tasks · Resources · About

Welcome to the OPTED inventory for media texts!

Quickly and easily query our database of news sources, organisations, datasets, corpora, tools and more! Find out which databases hold the text data that you are looking for. Do you have a specific source in mind? Try typing the name in the search bar and check if the entry already exists!

Entity Type: News Source
Filter by Country: all...

Query · Advanced query

Recent Entries

fodira Tool	21 Sep
bertopic Tool	19 Sep
OCTIS Tool	19 Sep
Validation Operation	15 Sep
oolong Tool	15 Sep

Contribute

You want to help us and add entries to the database? You can contribute through various means:

- Add media organisations and news sources to our

Guides & Materials

Read more detailed instructions and documentation on using or contributing to the inventory.

- detailed guide for adding news sources to the

5.1.1 What's inside *Meteor*?

Meteor has different types of entries. The primary types are:

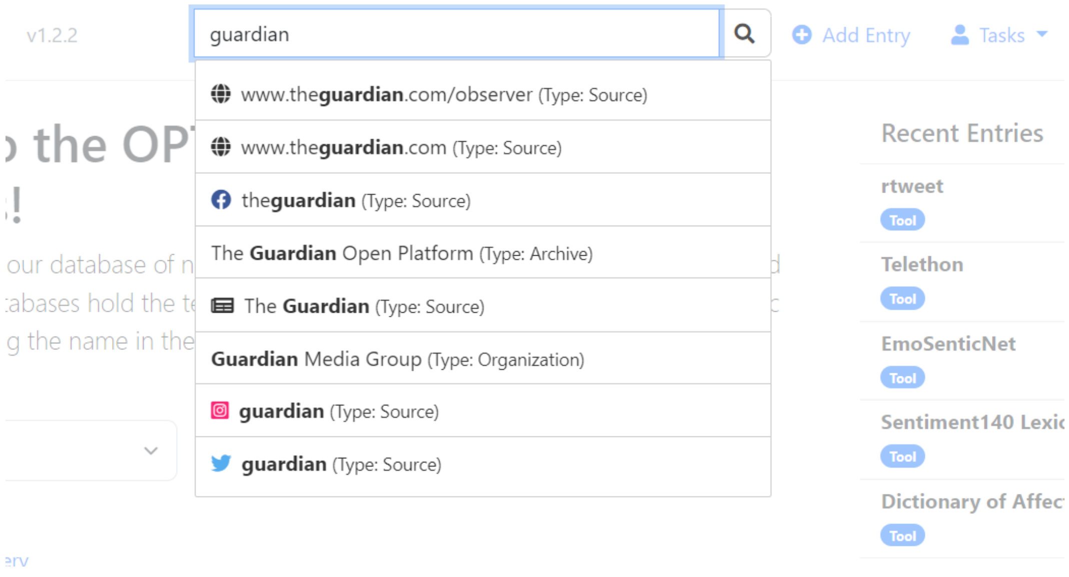
- Journalistic News Sources
- Media Organizations
- Datasets and Corpora
- Text Archives
- Tools for Text Analysis

There are also secondary types that help us to link entries with each other for example countries, or channels such as Twitter or Instagram.

5.1.2 Exploring *Meteor*

You can find specific resources with a free text search field that provides instant results (Figure 2).

Figure 2: Free text search with instant results.



guardian

www.theguardian.com/observer (Type: Source)

www.theguardian.com (Type: Source)

theguardian (Type: Source)

The Guardian Open Platform (Type: Archive)

The Guardian (Type: Source)

Guardian Media Group (Type: Organization)

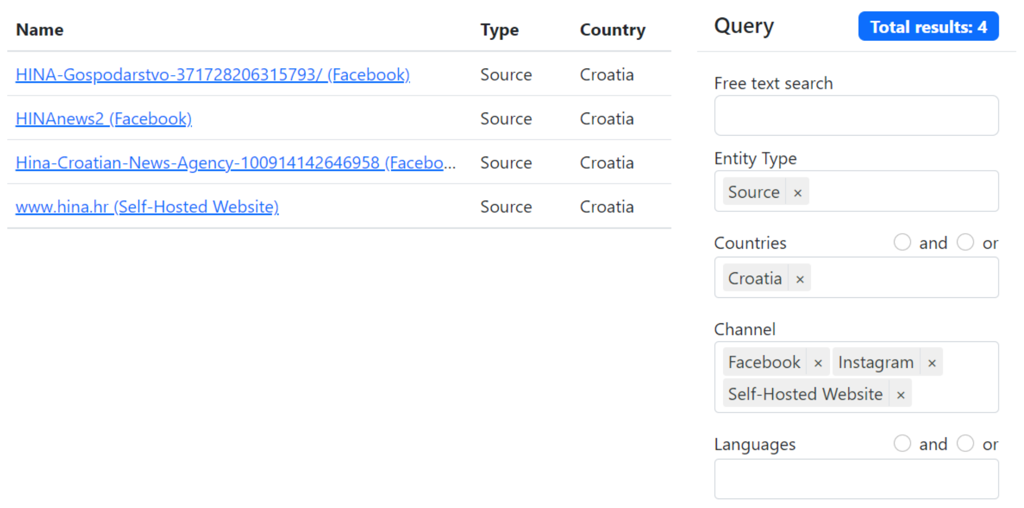
guardian (Type: Source)

guardian (Type: Source)

You can either click on one of the instant results, or click on the magnifying glass to view even more results. The free text search also looks into the description text of entries, alternative names, or the author field. So you can also find datasets or tools according to the names of authors.

Another method of exploring the platform is by using the query screen which offers a variety of filters (Figure 3).

Figure 3: Example of query with filters.



Name	Type	Country
HINA-Gospodarstvo-371728206315793/ (Facebook)	Source	Croatia
HINAnews2 (Facebook)	Source	Croatia
Hina-Croatian-News-Agency-100914142646958 (Facebo...	Source	Croatia
www.hina.hr (Self-Hosted Website)	Source	Croatia

Query — Total results: 4

Free text search:

Entity Type: Source ×

Countries (and / or): Croatia ×

Channel: Facebook × Instagram × Self-Hosted Website ×

Languages (and / or):

You can find news sources depending on search criteria such as language, country, channel, or even the size of the followers. Tools can also be queried according to various meta information, such as programming languages, concepts, operations and so forth.

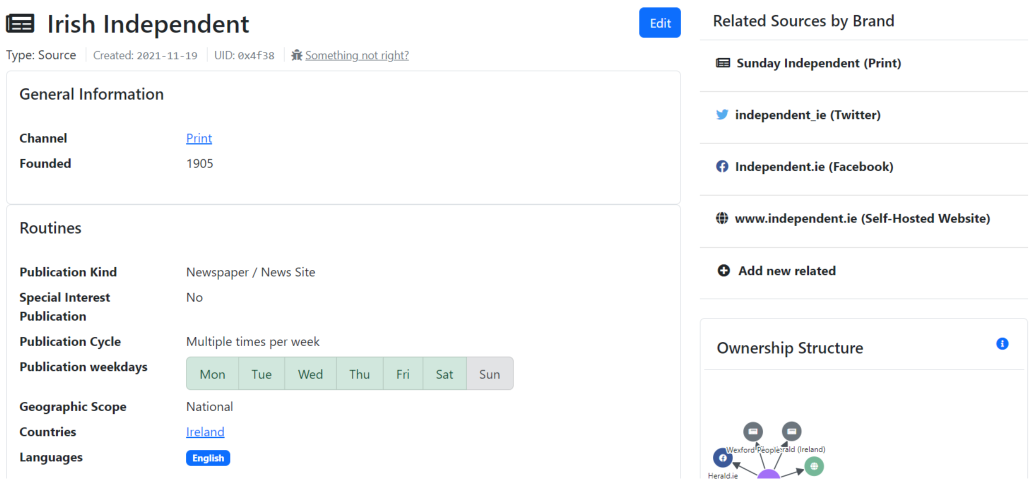
You then download the results in JSON format. One use case for this feature is that a researcher can query the platform for Twitter accounts in a specific country, download the query results and load them into a software for tweet collection.

Note: *The download feature is not ready yet, but is coming very soon.*

5.1.3 Detail View of Entries

Once you click on an entry, you get to the detail view (see Figure 4).

Figure 4: Example of detail view for a news source.



Irish Independent

Edit

Type: Source | Created: 2021-11-19 | UID: 0x4f38 | Something not right?

General Information

Channel	Print
Founded	1905

Routines

Publication Kind	Newspaper / News Site
Special Interest Publication	No
Publication Cycle	Multiple times per week
Publication weekdays	Mon Tue Wed Thu Fri Sat Sun
Geographic Scope	National
Countries	Ireland
Languages	English

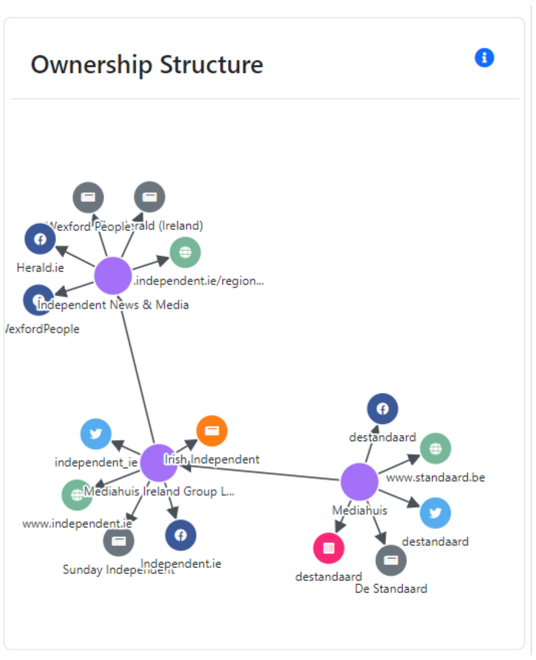
Related Sources by Brand

- Sunday Independent (Print)
- independent_ie (Twitter)
- Independent.ie (Facebook)
- www.independent.ie (Self-Hosted Website)
- Add new related

Ownership Structure

All related news sources are listed at the sidebar, which enables quick browsing through *Meteor*. Since the ownership structure of news sources is stored in the database as well, we also provide a network plot that helps you to explore these structures (Figure 5).

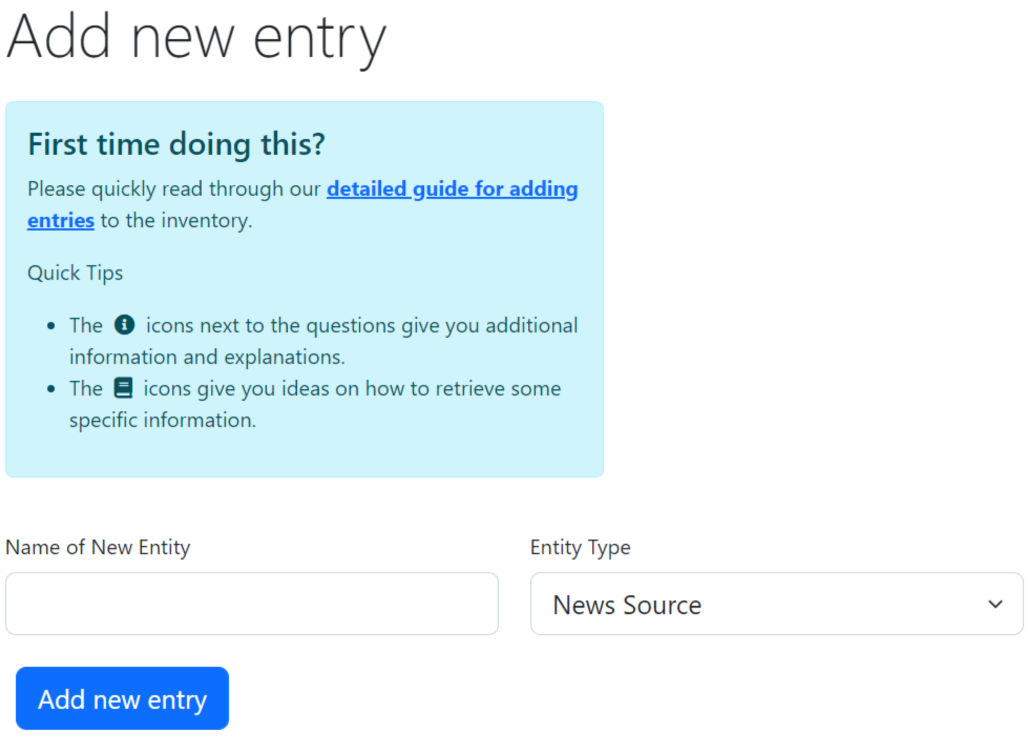
Figure 5: Example of ownership network.



5.1.4 Contributing Entries

Meteor is fueled by its community. So we encourage users to contribute entries to the database. We aim to make it as convenient as possible for to make contributions. A questionnaire guides you to enter the required meta-information (Figure 6). We provide different questionnaires depending on the type of entry you are planning to make.

Figure 6: Screen for making a new entry.



Add new entry

First time doing this?

Please quickly read through our **detailed guide for adding entries** to the inventory.

Quick Tips

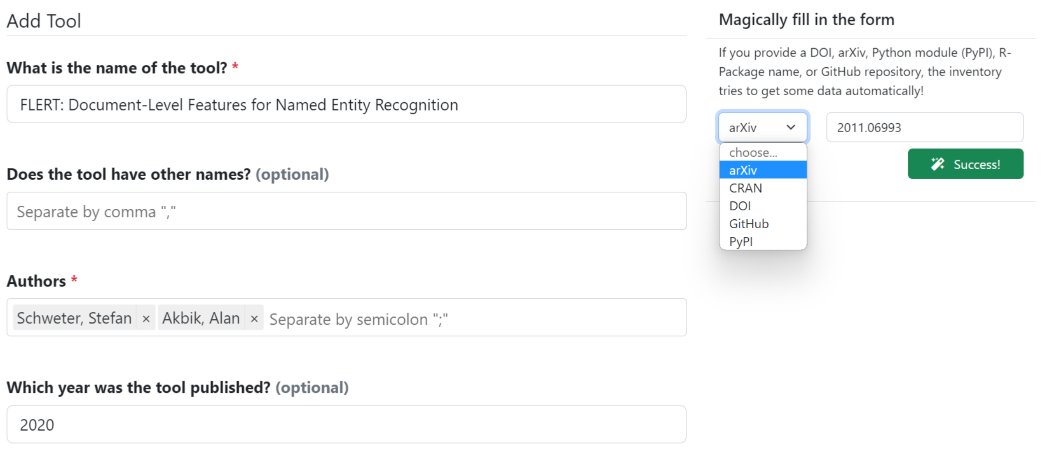
- The ℹ icons next to the questions give you additional information and explanations.
- The 📘 icons give you ideas on how to retrieve some specific information.

Name of New Entity	Entity Type
	News Source

Meteor leverages available APIs to enrich the information automatically. For example, we query Wikidata and Openstreetmap to retrieve general information for news sources and organizations, such as geographic names or addresses. For channel-specific information we call various APIs, such as siterankdata.com to retrieve information about daily website visitor count, or the Twitter API to get the follower count of an account.

The metadata for tools, corpora, or datasets can be semi-automatically imported via several APIs (Figure 7). We currently support CRAN, PyPI, arXiv, DOI, and GitHub.

Figure 7: Automatically retrieve meta-information from an API.



Add Tool

What is the name of the tool? *

FLERT: Document-Level Features for Named Entity Recognition

Does the tool have other names? (optional)

Separate by comma ","

Authors *

Schweter, Stefan × Akbik, Alan × Separate by semicolon ";"

Which year was the tool published? (optional)

2020

Magically fill in the form

If you provide a DOI, arXiv, Python module (PyPI), R-Package name, or GitHub repository, the inventory tries to get some data automatically!

arXiv 2011.06993

5.1.5 Using Meteor in the Classroom

Beyond a resource for scholarly research, Meteor can also be used in courses with (under-) graduate students. We successfully conducted a master level seminar where students learned about comparative research, as well as the theoretical foundations of media systems. The course consisted of two parts. The first part discussed selected readings about comparative research and (hybrid) media systems. The second part put these theoretical considerations into practice by systematically comparing countries based on the landscape of available news sources. For that purpose, groups of students chose two media systems in Europe and defined the subsection of news media that they were interested in (e.g., traditional print press). Meteor allowed the students to compare the two systems based on their sample of news sources.

You can find all the resources for the course we designed in the guides section of Meteor[14].

[14]https://meteor.opted.eu/guides/teaching-materials

5.2 Elegant Web-Scraping: WordPress API

- Difficulty: Medium
- Requirements:
  - Basic knowledge of the python syntax
  - Knowledge on how to use `pip`
  - A favourite code editor (or so called “IDE”)[15]

5.2.1 Introduction

In this tutorial, we show how journalistic media texts can be retrieved by leveraging the standard API provided by WordPress. WordPress was originally designed as a blogging software, but it evolved to a complex content management software. Because is open-source and can be used for free, it has become popular among professional news outlets, citizen journalists, alternative media, and even online retailers (just to name a few). According to the developers of WordPress, around 43% of all websites run on WordPress!

Fortunately, WordPress provides a API which can be queried to retrieve posts (or news articles) in a standardized format. This means that most websites that use WordPress as their content management system, all provide the exact same API. This is especially useful for researchers, because high quality and rich data can be retrieved form a variety of news sources without much customization.

We showcase how to retrieve posts using a python module[16] developed by *Mickaël “Kilawyn” Walter* on the example website Guido Fawkes[17]. The website is a good example, because it is an alternative media outlet commenting on politics in the UK, but it is not available in “traditional” data archives. Find out more about the website, including meta-data, on Meteor[18].

5.2.2 Overview

1. Installing the python module
2. Testing the API
3. Exploring the content
4. Querying a single post
5. Downloading and storing posts
6. Summary

A set of finished sample scripts can be downloaded at the bottom of this page.

5.2.3 Installation

First, clone (or download) our repository for `wp-json-scraper`. The module was originally developed by *Mickaël “Kilawyn” Walter* and it provides a convenient wrapper for the

[15]If you are not sure about which IDE to use, we recommend VSCode (https://code.visualstudio.com/), PyCharm (https://www.jetbrains.com/pycharm/), or Jupyter Notebook (https://jupyter.org/)
[16]https://github.com/opted-eu/wp-json-scraper
[17]https://order-order.com/
[18]https://meteor.opted.eu/view/Source/https_order_order_com_website

WordPress API in python. We created a fork in our OPTED repository, where we made some minor improvements to the already excellent software.

You can clone the repository with this command in your Terminal or shell:

```
git clone https://github.com/opted-eu/wp-json-scraper.git
cd wp-json-scraper
```

Alternatively, you can go to the GitHub repository, download it as zip file, and extract it in the destination of your choice.

Next, open the root folder of the repository in your favourite code editor. **For the remainder of the tutorial, we always assume that you are in root directory of `wp-json-scraper`** (where you can find the files `README.md` and `requirements.txt`).

Now it is time to install the required modules for `wp-json-scraper`, you can do that by opening a terminal and entering this command:

```
pip install -r requirements.txt
```

Additionally, we need the `bs4` module for this tutorial:

```
pip install bs4
```

Finally, we make sure that the installation worked by creating a new python script where we try to load the module:

```python
from lib.wpapi import WPApi
```

This should run without errors.

5.2.4 Testing the API

First, we need to ensure that the website that we want to scrape actually uses WordPress and also has the API exposed. For this, we create a new script that we name `check_api.py`. We load the required modules as follows:

```python
from lib.wpapi import WPApi
from pprint import pprint # helper to pretty print output
```

Next, we declare the website that we want to check:

```python
target = "https://order-order.com/"
```

Checking the availability of the API is rather simple, we just have to create an instance of the `WPApi` class where we pass in our `target` as the first (and only) argument:

```python
wordpress = WPApi(target)
```

Next, we get the basic information of the website and have it printed:

```
info = wordpress.get_basic_info()
pprint(info)
```

Depending on the website, you will get more or less output here. If the website does not have WordPress (or the API is disabled), then the `WPApi` class will throw an error (`lib.exceptions.NoWordpressApi`).

5.2.5 Exploring the content

Now that we have established that the API works as excected, we can move ahead and explore the content. First, we want to see how many posts are availabale in total:

```
total_posts = wordpress.total_posts()
print(total_posts)
```

There are over 40,000 posts ready for download from the page. But before we move on, let's explore some other aspects.

For example, most blogs have categories associated with their posts:

```
categories = wordpress.get_categories()
print(len(categories))
```

The `get_categories()` method returns a list of dictionaries, where each represents a single category. In this case, the `categories` object should have a length of 13. Let's print out the names of these categories:

```
for category in categories:
    print(category['name'], category['id'])
```

We could also explore all available tags with `get_tags()` or all blog authors with `get_users()`:

```
users = wordpress.get_users()
print(len(users))
for user in users:
    print(user['name'], user['link'])
```

Another feature is to search posts based on keywords. Wordpress has a fulltext index of all posts, so you can query posts based on keywords that you find interesting:

```
europe = wordpress.total_posts(search_terms='europe')
print(europe)
```

There are over 2300 posts that contain the keyword `'europe'`. You can also try other keywords and check your results.

5.2.6 Querying a single post

Before we scrape the entire contents of the website, let's check the data structure of a single post first. We can retrieve posts with the `get_posts()` method, which takes four keyword arguments:

- `comments` (`bool`, default: `False`): indicate whether you want to retrieve comments as well.
- `start` (`int`, default: `None`): select starting number of posts to retrieve. Default setting is that it starts at the first post (usually sorted by date).
- `num` (`int`, default: `None`): set the limit of total posts to retrieve. Default setting is without any limit.
- `force` (`bool`, default: `False`):: indicate whether you want to force downloading. The `WPApi` client caches all posts in the background. If you do not want to use the cached posts, then you select `True`.

To retrieve the newest post, we call the method with the following arguments:

```
posts = wordpress.get_posts(num=1)
print(len(posts))
```

The `posts` object is a list of dictionaries, where each dictionary represents a single post. Because we set the limit (`num`) to `1`, the list has the length of 1. So there is only one post that we can unpack and inspect as follows:

```
post = posts[0]
print(post.keys())
```

The `keys()` method shows us all fields that a single object contains. We get meta-information in a beautiful and standardized format. For example, regardless of the blog layout or language, the `date` field is always in a machine readable format. The exact fields that are interesting for your research might vary, but typically the most interesting fields are:

- `id` (`int`): numeric ID of the post.
- `date` (`str`): Date and time post was published. Format `YYYY-MM-DD HH:MM:SS`.
- `modified` (`str`): Date and time post was modified. Format `YYYY-MM-DD HH:MM:SS`.
- `link` (`str`): Official link to post. This is useful for checking the content later.
- `title` (`dict`): Title or headline of the post. The data is a dictionary that contains the key `rendered`, which shows the title as it is served to the user.
- `content` (`dict`): Content (or body text) of the post. The data is a dictionary that contains the key `rendered`, which shows the content as it is served to the user.
- `excerpt` (`dict`): Excerpt (or a summary) of the post. Same as above, the key of interest is `rendered`.
- `author` (`int`): numeric ID of the author. To resolve the author names, we can use the `get_users()` method.
- `categories` (`list`): a list of numeric category IDs. We can resolve the category names by using the `get_categories()` method.

- `tags` (`list`): similarly to `categories`, this is a list of numeric tag IDs that we can resolve with the `get_tags()` method.

5.2.7 Reformatting a post

Of course, we could just take the `post` object and store it as a JSON file. However, then we would also store less interesting information and would store the nested structure. Therefore, we reformat some fields and also unnest the content.

First, we create a new empty dictionary that will hold the reformatted post and we can already access some fields that we do not need to reformat:

```
cleaned_post = {'id': post['id'],
                'link': post['link'],
                'date_published': post['date'],
                'date_modified': post['modified']}
```

To unnest the `title` field, we can do the following:

```
cleaned_post['title'] = post['title']['rendered']
```

The `content` field is a bit tricky, because it often also contains HTML fragments that are used for formatting. There are several ways to approach this. In this tutorial, we are going to use the `bs4` module which we downloaded in the installation section. We import the `BeautifulSoup` class, which can parse HTML and remove all kinds of unwanted tags.

```
from bs4 import BeautifulSoup
```

The `BeautifulSoup` class handles all the troublesome aspects of parsing HTML and helps us to simply return cleaned text by accessing the `text` attribute:

```
content = BeautifulSoup(post['content']['rendered'])
cleaned_post['content'] = content.text
```

Note: *we could also extract links to other pages in this step, if we were interested in that.*
Same applies to the `excerpt` field

```
excerpt = BeautifulSoup(post['excerpt']['rendered'])
cleaned_post['excerpt'] = excerpt.text
```

The next part that is tricky: it is to resolve the author, category, and tag IDs to their names. It works the same way for all three IDs, so we show only here how to do it for the author IDs. First we have to get all authors with the `get_users()` method:

```
users = wordpress.get_users()
```

As mentioned above, this returns a list of dictionaries where each dict represents meta information on a single author. We want to know which author has which ID, so we can simply reformat the `users` list to a dictionary. The dictionary will have the author ID as key and the author name as value:

```python
authors = {}
for user in users:
    authors[user['id']] = user['name']

print(authors)
```

We now got a dictionary where we can lookup authors by ID:

```python
our_author = authors[post['author']]
print(our_author)
```

Finally, we can add that to our `cleaned_post` dictionary:

```python
cleaned_post['author'] = authors[post['author']]
```

Finally, we have one post cleaned and reformatted. Let's admire it:

```python
pprint(cleaned_post)
```

5.2.8 Downloading and storing posts

In this section we cover how to download **all** posts that were published on the website. Please proceed with care, because some websites have a lot of content. For the purpose of the tutorial, we limit our scraping to 100 articles.

We will proceed as in the previous section, but this time we do not only apply it to one article but to many articles in a for loop.

So a lot of code from above will be repeated. At some spots, we also make our code more efficient.

5.2.9 Making preparations

Let's ensure that we really have all authors, categories, and tags ready so we can resolve their IDs. We use a shorthand notation here (see: dictionary comprehension if you want to learn more), which is a bit harder to read, but does exactly the same as we have done above:

```python
users = wordpress.get_users()
authors = {user['id']: user['name'] for user in users}

categories = wordpress.get_categories()
categories = {c['id']: c['name'] for c in categories}

tags = wordpress.get_tags()
tags = {t['id']: t['name'] for t in tags}
```

Next, we need to set a directory where we will store our articles. There are many ways to to that. In this tutorial, we will save every article as a single JSON file. We use the `pathlib` here, which is very convenient for handling paths:

```python
from pathlib import Path
p = Path.cwd() # get current working directory
output_dir = p / 'output' / 'order-order.com'

if not output_dir.exists():
    output_dir.mkdir(parents=True)
```

This code simply creates a new directory structure while making sure that nothing is overwritten. If you execute this code, a new folder will appear in your current working directory.

Final preparation is to ensure that we loaded the `json` module:

```python
import json
```

5.2.10 Downloading and parsing several posts

To download many posts, we added the `yield_posts()` method to the `WPApi` class, which can handle downloading larger amounts of data. This method is a generator and returns one post at a time as soon as it is downloaded. This allows us to process the post as soon as it is downloaded and then store it to our output directory as a single JSON file.

Note: *As mentioned above, we will limit our request here to 100 posts by using the num keywords argument.*

```python
for post in wordpress.yield_posts(num=100):
    post_id = post['id']
    print(post_id)

    cleaned_post = {'id': post_id,
                    'link': post['link'],
                    'date_published': post['date'],
                    'date_modified': post['modified']}

    title = BeautifulSoup(post['title']['rendered'])
    cleaned_post['title'] = post['title']['rendered']

    content = BeautifulSoup(post['content']['rendered'])
    cleaned_post['content'] = content.text

    excerpt = BeautifulSoup(post['excerpt']['rendered'])
    cleaned_post['excerpt'] = excerpt.text

    cleaned_post['author'] = authors[post['author']]
    cleaned_post['categories'] = [categories[c] for c in post['categories']]
    cleaned_post['tags'] = [tags[t] for t in post['tags']]

    with open(output_dir / f'{post_id}.json', 'w',
              encoding = "utf8") as f:
        json.dump(cleaned_post, f,
                  ensure_ascii=False)
```

Some additional information on above code block. We print out the current `post_id`, because then we know whether the download is still running. The middle part is just the condensed version of the code that we explained above. We also use the shorthand notation for resolving the categories and tags. And finally we use `json.dump()` to store the post in the output directory where the file name is the post ID.

When you execute this code block you can observe how the output directory is slowly filled with single JSON files.

5.2.11 Summary

We have shown how to leverage the WordPress API to download media text data in a structure and clean format. The example shown here was an alternative media outlet from the UK. But the great advantage of this method is that above code works on a large number of websites and does not require much adjustment.

5.3 Scraping Telegram: Alternative News Sources

- Difficulty: Medium
- Requirements:
  - Basic knowledge of the python syntax
  - Knowledge on how to use `pip`
  - A favourite code editor (or so called "IDE")
  - A Telegram account (alongside a phone with telegram installed)

5.3.1 Introduction

In this tutorial, we show how journalistic media texts can be retrieved from Telegram. Many news outlets, especially alternative media platforms use Telegram to engage with their audiences. The low degree of moderation and supervision makes Telegram not only an interesting platform for media outlets spreading (mis-)information, but also for fringe-groups of the political spectrum. They tend to use public channels on Telegram to establish their narratives or to recruit and to mobilise supporters.

This tutorial shows how to use the official API while respecting Telegram's terms of services. You can even retrieve many data points beyond only the message content. Because you need to specify only the channel name, you can easily get started and study information flows between channels or different social media platforms.

5.3.2 Preparations & Installation

To keep things neat and tidy, create a new directory for this tutorial and name it `telegram_scraping`. For the remainder of the tutorial, we assume that all commands are run from this directory.

Telegram API Account

Before we jump into this tutorial, make sure that you have a Telegram account and retrieved your API credentials. Please note that a valid phone number is required to open an account.

The exact procedures for obtaining API credentials might change from time to time. Therefore, please follow the steps outlined in the official documentation for Telegram developers[19] and then come back to this tutorial with a valid **API ID** and **API Hash**.

Note: *You can also find more information on how to retrieve API credentials in the Telethon documentation*[20].

Required Packages

To install the Telethon library open a terminal and install it via `pip`

```
pip install telethon
```

5.3.3 Testing the API

Create a new file called `secrets.json`, this is where you store your Telegram API credentials. Saving them in a separate file keeps your credentials detached from the scraping scripts. Open `secrets.json` and edit it accordingly:

```
{
    "api_id": "<your api id>",
    "api_hash": "<your api hash>"
}
```

Note: *Please do not share your API credentials with anyone.*

Next, we create a new python script and name it `check_api.py` in which we import the following:

```
import json # to load our API credentials
from telethon.sync import TelegramClient # the Telegram client
```

If everything is installed correctly, these imports should run without errors.

Now, we load our API credentials:

```
with open('secrets.json') as f:
    credentials = json.load(f)
```

Next, we instantiate a new `TelegramClient` where we pass in our API credentials:

```
client = TelegramClient('user',
                        credentials['api_id'],
                        credentials['api_hash']).start()
```

The `TelegramClient` class needs three arguments: the session name (`'user'`), the API ID and the API hash. The session name tells Telethon where to store the session credentials. The first time you run above code, you will be prompted to enter your phone number to verify it is really you:

[19]https://core.telegram.org/api/obtaining_api_id
[20]https://docs.telethon.dev/en/stable/basic/signing-in.html

```
Please enter your phone (or bot token): 3551337
```

Once you entered your phone number, Telegram will send you a verification code to your phone. You get prompted again to enter the verification code in python:

```
Please enter the code you received: 12345
Signed in successfully as User
```

We have to complete this only once. If you observe your current directory carefully, you will notice that a new file appeared: `user.session`. Telethon stores the session information in this file. The file name is specified in the first argument when instantiating the `TelegramClient` (remember, we passed in `'user'`). If you delete `user.session`, you will have to repeat the login verification.

A great way of checking whether the client is configured correctly is by sending a message to yourself:

```python
client.send_message('me', 'Hello to myself!')
```

If you check Telegram on your phone, you should have received a message.

5.3.4 Connecting to a chat

In this tutorial we will explore the Telegram account by ovalmedia (see the entry on Meteor for more information[21]). It is an alternative media outlet that mainly offers content on alternative video streaming platforms. They promote their recent videos on telegram alongside a summary of each video. Of course, you could also use any other Telegram account for this tutorial; feel free to experiment.

We create a new script `connect.py` where import the same modules as before and instantiate the `TelegramClient`:

```python
import json # to load our API credentials
from telethon.sync import TelegramClient # the Telegram client
from pprint import pprint # helper to pretty print

with open('secrets.json') as f:
    credentials = json.load(f)

client = TelegramClient('user',
                        credentials['api_id'],
                        credentials['api_hash']).start()
```

Next, we save the username of ovalmedia in an object and use the `get_entity()` method to retrieve the account:

```python
account_name = 'ovalmedia_english'
chat = client.get_entity(account_name)
```

[21]https://meteor.opted.eu/view/Source/ovalmedia_english

We can now check the official name of the account by accessing the `title` attribute of the `chat` object:

```
chat.title
# 'OVALmedia | English'
```

With this step, we can also verify whether we really got the account that we were looking for.

5.3.5 Retrieving messages

Before we scrape all messages in a channel, let's check a single message first. We can retrieve messages with the `get_messages()` method, which takes the `chat` object as its argument:

```
messages = client.get_messages(chat)
```

The `messages` object behaves like a python list, but additionally as a `total` attribute that shows the total number of messages in the chat:

```
print(messages.total)
```

The first element in the list is also the most recent message, and we can access it like this:

```
message = messages[0]
```

`message` the message object has a series of attributes, the most interesting for us are:

- `id` (`int`): message id within this chat
- `date` (`datetime`): timestamp when message was sent
- `message` (`str`): the actual message content
- `forwards` (`int`): number of times the message was forwarded
- `views` (`int`): number of chat members who have seen the message

Let's have a look at the most recent message in the channel:

```
print('Newest message:')
print(message.id, message.date)
print('Message content:')
print(message.message)
print('Total views:', message.views, 'Total forwards:', message.forwards)
```

We can also use the `to_dict()` method to get all message contents and attributes as a python dictionary:

```
pprint(message.to_dict())
```

Note: *There is even more interesting data contained in the* `message` *object. For example, the* `entities` *attribute is a list of message elements and can contain URLs stored as* `MessageEntityTextUrl` *objects. This is potentially interesting, if you want to study information flows. Another interesting attribute is* `media` *where you can retrieve attached images.*

Reformatting a message

Of course, we could just take the `message.to_dict()` object and store it as a JSON file. However, then we would also store less interesting information and would store the nested structure. Therefore, we reformat some fields and also unnest the content.

```python
cleaned_message = {'channel_name': chat.username, # keep track on where we got
 ↪  the message from
                   'id': message.id,
                   'date': message.date,
                   'content': message.message,
                   'forwards': message.forwards,
                   'views': message.views}
```

5.3.6 Downloading and storing all messages

If we want to retrieve **all** messages in from a telegram channel, we could use the `get_messages()` method and iterate over the resulting list. This works well for small chats, but less well for chats with thousands of messages. Instead, we use the `iter_messages()` method, which retrieves messages in batches and also has useful features such as limiting the number of messages to retrieve, or to search for keywords.

We create a new script that we name `scrape_channel.py` and we make the same imports as before, but also import some utility modules

```python
import json # to load our API credentials
from telethon.sync import TelegramClient # the Telegram client
from pprint import pprint # helper to pretty print
from pathlib import Path # makes handling file paths a breeze

with open('secrets.json') as f:
    credentials = json.load(f)

client = TelegramClient('user',
                        credentials['api_id'],
                        credentials['api_hash']).start()

account_name = 'ovalmedia_english'
```

We prepare an output folder with the `Path` class:

```python
p = Path.cwd()

output_dir = p / 'output' / account_name
if not output_dir.exists():
    output_dir.mkdir(parents=True)
```

Next, we get the chat as we did before:

```python
chat = client.get_entity(account_name)
```

Now, we use the `iter_messages()` method to incrementally retrieve messages (from newest to oldest). We use the method as for-loop generator, where with every iteration, we get a message, reformat it and then store it as a JSON file:

```python
for message in client.iter_messages(chat):
    print('Retrieving message:', message.id)

    cleaned_message =  {'channel_name': chat.username,
                        'id': message.id,
                        'date': message.date,
                        'content': message.message,
                        'forwards': message.forwards,
                        'views': message.views}

    file_name = output_dir / f'{message.id}.json'

    with open(file_name, 'w') as f:
        json.dump(cleaned_message, f, indent=True, default=str)
```

Rate Limits: *Telegram has rate limits in place. So if the chat that you want to scrape has more than 3000 messages, you should adjust your code. Telethon provides the* `wait_time` *keyword argument for this purpose where you can set a wait time in seconds between requests: e.g.* `client.iter_messages(chat, wait_time=10)`

When you execute the code block above, you can observe how the output directory is filled with single JSON files, where each file represents a single message. Of course you can also use pandas instead and construct a data frame:

```python
import pandas as pd

tmp = []

for message in client.iter_messages(chat):
    print('Retrieving message:', message.id)
    cleaned_message =  {'channel_name': chat.username,
                        'id': message.id,
                        'date': message.date,
                        'content': message.message,
                        'forwards': message.forwards,
                        'views': message.views}

    tmp.append(cleaned_message)

df = pd.DataFrame(tmp)
```

Filtering Messages

We could also filter the results by using the `search` keyword argument, e.g.:

```python
for message in client.iter_messages(chat, search='europe'):
    ...
```

Or set an offset date to retrieve only messages before a certain date, e.g.:

```
from datetime import datetime
my_datetime = datetime.strptime('2022-12-31', '%Y-%m-%d')

for message in client.iter_messages(chat, offset_date=my_datetime):
    ...
```

5.3.7 Summary

We have shown how to use the Telethon library to download media text data in a structure and clean format. The example shown here was an alternative media outlet from Germany. The advantage of this method is that the above code works on a large number of accounts, since Telegram is also popular among political parties to engage with their supporters.

References

Balluff, P., Lind, F., Boomgaarden, H. G., & Waldherr, A. (2022). Mapping the European media landscape – Meteor, a curated and community-coded inventory of news sources. *European Journal of Communication*. https://doi.org/10.1177/02673231221112006