May 09, 2024 Version 2

Protocol OCPRIP V.2

DOI
dx.doi.org/10.17504/protocols.io.261ge56qwg47/v2

Chiara Parravicini[1], Daniele Spedicati[1], Matteo Guenci[1], Nicole Liggeri[1]

[1]University of Bologna

Nicole Liggeri
University of Bologna

DOI: dx.doi.org/10.17504/protocols.io.261ge56qwg47/v2

Protocol Citation: Chiara Parravicini, Daniele Spedicati, Matteo Guenci, Nicole Liggeri 2024. Protocol OCPRIP. **protocols.io** https://dx.doi.org/10.17504/protocols.io.261ge56qwg47/v2Version created by Nicole Liggeri

Protocol status: Working
We use this protocol and it's working

Created: April 08, 2024

Last Modified: May 09, 2024

Protocol Integer ID: 99468

Abstract

We present a step-by-step methodology for the systematic extraction, alignment, and analysis of peer review data from **Crossref** to enhance the **OpenCitations Index**.
The protocol delineates four key phases: data gathering, alignment, data management and data analysis. In the data gathering phase, peer review data is extracted from Crossref, annotated, and organized into JSON format. The alignment phase adapts the OpenCitations Index workflow to treat peer reviews as first-class data entities, generating CSV files for reviews and provenance.
These datasets are then converted into RDF format, adhering to the OpenCitations Data Model.
Furthermore, the protocol addresses data management by querying OpenCitations Meta database via SPARQL to identify existing peer reviews. This process facilitates the categorization of peer reviews into those already present in the OpenCitations repository and those requiring inclusion. Continuous synchronization with Crossref ensures the currency and accuracy of the dataset.
Lastly, data analysis tasks are outlined, including isolating venues, counting DOIs, and verifying article DOIs.

Gathering Data from Crossref

1 The first step is to isolate all objects described in Crossref as peer-review. In Crossref, these entities are registered with the type "peer-review". A code to make this filtering is created. The dump, due to its dimensions, has been previusly devided in smaller chuncks.

More specifically, for each entity we want to annotate:
- The DOI value of the peer-review
- The type of peer-review
- The year of publication
- The DOI of the article reviewed
- The title of the article
- The ID of the venue (ISSN/ISBN)

The goal is to create a JSON file that will be used as a starting point for future analyses.

Note

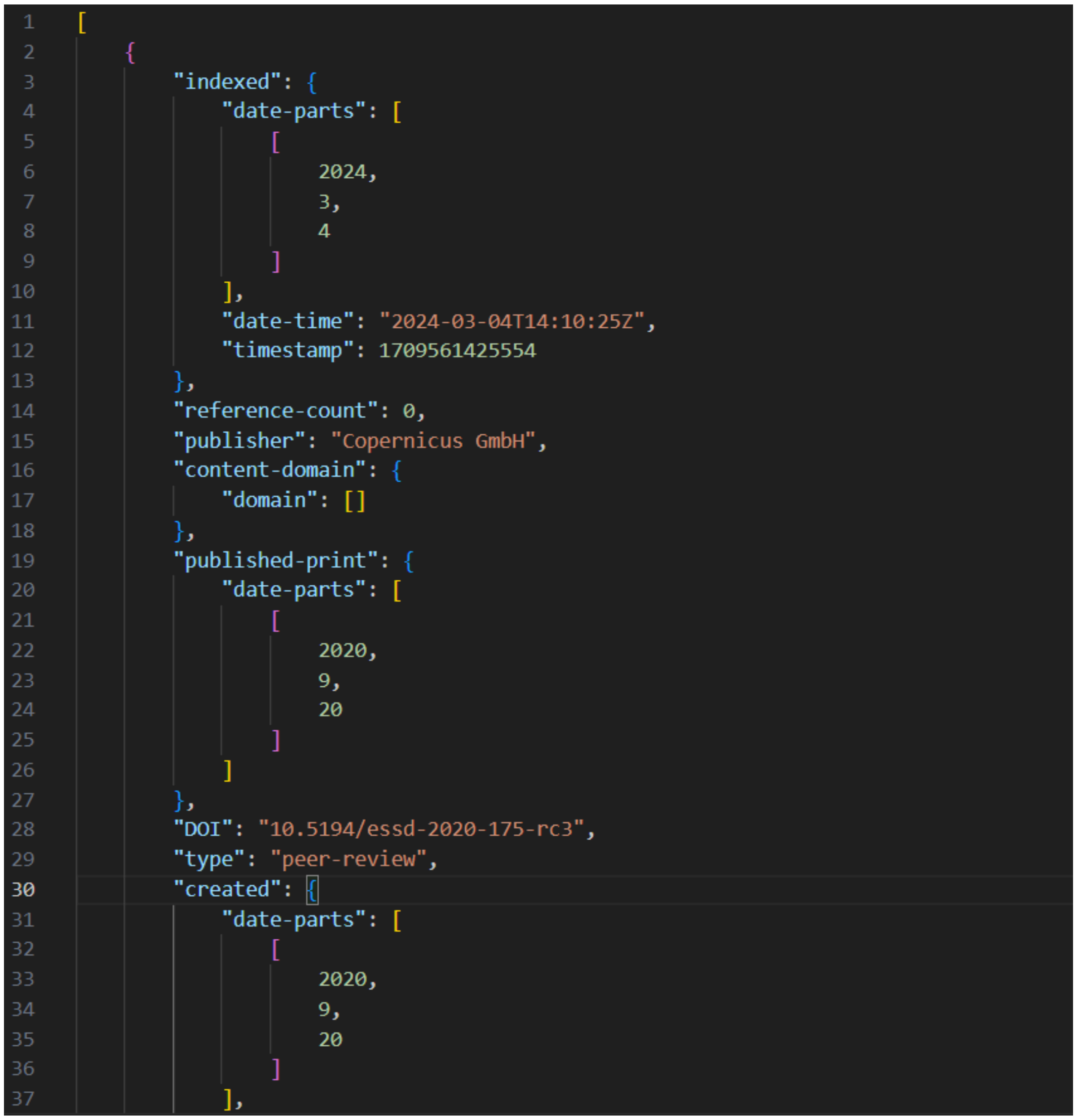
Input: Code to extract all items with type "peer-review".

1.1 To achieve this goal, we created the followin code

```
import json

files = ["peer_review_items.json", "peer_review_items2.json",
"peer_review_items3.json", "peer_review_items4.json",
"peer_review_items5.json"]
output_filenames = ["doi_pairs.json", "doi_pairs2.json",
"doi_pairs3.json", "doi_pairs4.json", "doi_pairs5.json"]
def parse_date(date_parts):
    if date_parts:
        try:
            return "-".join(map(str, date_parts))
        except ValueError:
            pass
    return None



for file, output_filename in zip(files, output_filenames):
    with open(file, "r", encoding="utf-8") as json_file:
        json_data = json.load(json_file)
        url_pairs = []
        for element in json_data:
            url1 = element["URL"]
            url2 = element.get("resource", {}).get("primary",
{}).get("URL")
            date_peer_review = element.get("created",
{}).get("date-parts", [[]])[0]
            date_article_published = element.get("published",
{}).get("date-parts", [[]])[0]
            author_info = element.get("editor", [{}])[0]
            given_name = author_info.get("given", "")
            family_name = author_info.get("family", "")
            if url1 and url2:
                url_pairs.append({
                    "URL_peer": url1,
                    "URL_article": url2,
                    "date_peer_review":
parse_date(date_peer_review),
                    "date_article_published":
parse_date(date_article_published),
                    "author_given_name": given_name,
                    "author_family_name": family_name
                })

    with open(output_filename, 'w') as output_file:
        json.dump(url_pairs, output_file, indent=4)

    print("URL pairs from", file, "saved to", output_filename)
```

Expected result

A json file containing all the peer reviews registered in Crossref.



```
[
    {
        "indexed": {
            "date-parts": [
                [
                    2024,
                    3,
                    4
                ]
            ],
            "date-time": "2024-03-04T14:10:25Z",
            "timestamp": 1709561425554
        },
        "reference-count": 0,
        "publisher": "Copernicus GmbH",
        "content-domain": {
            "domain": []
        },
        "published-print": {
            "date-parts": [
                [
                    2020,
                    9,
                    20
                ]
            ]
        },
        "DOI": "10.5194/essd-2020-175-rc3",
        "type": "peer-review",
        "created": {
            "date-parts": [
                [
                    2020,
                    9,
                    20
                ]
            ],
```

example_query.json 1KB

Aligning data to Open Citations Data Model

2 The Alignment phase of our workflow is based on the Ingestion workflow developed for **COCI**, the OpenCitations Index of Crossref, as proposed in **this article**. (https://doi.org/10.1007/s11192-019-03217-6). Following this approach we create our Index of Peer-Reviews considering them as first-class data entities (see the section *Indexing citations as first-class data entities* of the above cited article).

2.1 Phase 1: global data generation: Once extracted data as previously described, they are organised in three dataset:

- Dates, the publication dates of all the reviews and the reviewed entities.
- ISSN: and the publication type in which the reviews are stored.
- ORCID (if any) associated with the authors.

2.2 Phase 2: CSV generation As in our workflow model, we generate two CSV files, one for the Reviews dataset and one for the Provenance dataset.

1. The Reviews CSV will associate the following informations to each row which represents a review statement:

- OCI which is an identifier to refer to the Citations.
- The DOI of the review.
- The Creation date of the review(which correspond to the date of creation of the citing entity and will be gained by the Dates dataset)
- We retrieve the publication date of the reviewed entity and calculate the timespan (the timespan between the publication of the review and the reviewed entity).
- We specify the Journal self-citation in case the Review and the reviewed entity are published in the same Journal Using the ISSN dataset.

2. The Provenance CSV will contain information about the provenance information for each Review, the data for each row will be:

- OCI as identifier of the review.
- The agent responsible for the generation of the citation.
- The Crossref API call that refers to the data of the review.

The major change from our workflow and the general one in COCI is the removal of the Author Self Citation check in this phase. Infact, dealing with Reviews, it is impossible that the author is reviewing its own work.

2.3 Phase 3: converting into RDF: the data from the CSV are converted in N-Triples format following the **OpenCitations Data Model** (OCDM). The DOIs of the reviews and the reviewed entities become DOI URLs, while the IRI of the Citations are the Entity subjects of our triples. The reification of the Citation to make it a first-class data entity will be made according to the following snippet as in the following example (slight modification of the source http://dx.doi.org/10.6084/m9.figshare.1512816)

```
@prefix : <http://www.sparontologies.net/example/> .
@prefix cito: <http://purl.org/spar/cito> .

# Direct form for a citation
:paper-a cito:reviews :paper-b .

# Reified form for a citation sharing the same
# citation function of the above one
:citation a cito:Citation ;
    cito:hasCitingEntity :paper-a ;
    cito:hasCitationCharacterization cito:reviews ;
    cito:hasCitedEntity :paper-b .
```

2.4 We don’t consider Phase 4: updating the triplestore since it is not required in this specific project.

Data management

3 This step of our workflow involves querying the Digital Object Identifiers (DOIs) of the peer reviews obtained from the Crossref dump within the OpenCitations database, using SPARQL queries.

3.1 This query process aims to identify whether the peer reviews are already present in the OpenCitations Meta database. After executing the SPARQL queries, the retrieved DOIs will be cross-referenced with the OpenCitations dataset. Subsequently, the results will be categorized into two distinct lists (or dictionary, depending on what will prove to be the more convenient option): List A will comprise those peer reviews already present in the OpenCitations database, while List B will contain those that are not (either because the review is not present, or because it is present but it has not been labeled as a review). This separation facilitates a comprehensive understanding of the coverage of peer reviews within the OpenCitations repository, allowing us to discover any potential gaps in data representation or registration. For instance, assuming we have retrieved a set of peer review DOIs from the Crossref dump, we would execute SPARQL queries to identify whether these DOIs exist within the OpenCitations dataset.
An initial hypothesis for executing such a query could be:

```
PREFIX cito: <http://purl.org/spar/cito/>
PREFIX pro: < http://purl.org/spar/pro >
SELECT ?peerReview
WHERE {
  ?peerReview a cito:Citation ;
      cito:hasCitationCharacterization cito:reviews ;
      cito:hasCitingEntity ?PeerReview.
  ?PeerReview pro:isDocumentContextFor ?RoleInTime.
  ?RoleInTime pro:withRole pro:PeerReviewer.
}

  FILTER (?peerReview IN (doi:10.1234/peerreview1,
doi:10.5678/peerreview2, ...))
}
```

In this SPARQL query, we specify the prefix for the Citation Typing Ontology (CiTO) namespace, which encompasses terms related to citation types. Then, we select the ?peerReview variable, representing the DOI of the peer reviews. We filter the results to include only those DOIs that match our list of retrieved peer review DOIs from the Crossref dump. This query enables us to identify peer reviews already present in the OpenCitations dataset, leading to the categorization into List A and List B for subsequent analysis.

3.2

Additionally, we plan to make sure that the software will incorporate functionality to handle updates from Crossref, ensuring the ongoing synchronization of data between Crossref and OpenCitations. Upon each execution, the software will compare the existing dataset with the latest information available from Crossref, identifying any additions or deletions of peer reviews.
By continuously monitoring updates from Crossref, the software will maintain the currency and accuracy of the dataset used for subsequent analyses, ensuring the reliability of research insights derived from the OpenCitations repository.

Data analysis

4 After completing all the previous tasks for resolving the issues presented so far, it is necessary to proceed and resolve the remaining operations to be performed:

- Isolating the Venues from the List (A): One could imagine a resolution based on iterating through list A (using for or while loops) and consequently using conditional blocks (if statements) to check if an element is a "venue". In case of a positive result, it could be added to a new list (or any other data structure that is most convenient to use) using the appropriate method. This process could allow isolating all the "venues" and inserting them into a separate container.

- Counting the Number of DOIs in the List (B): To count the number of DOIs in list B, one could use Python's len() function, which would return the number of DOI elements in the list. This process involves two branches: one assumes that the DOIs are strings within the list, in which case len() would directly return the number of DOIs. If, instead, the DOIs are further nested (for example, as elements of sublists), an additional for loop may be necessary to access these DOIs.

- Taking the DOIs of the Articles from List (B) and Checking: For this step, a method similar to the first step described in this section of the protocol could be used, i.e., an iteration (using for or while loops) applied to the DOIs of the articles. For each DOI present, a check should be applied using, for example, Python's "in" operator to verify if the DOI is present in list B. If this check yields a positive result, the outcome could be saved using the appropriate action (for example, adding the result to a list or any other data structure deemed suitable at the time of actual software development).

As of today, we are not able to predict what results this software will offer, as we are still in the embryonic stages of software development, making it impossible to anticipate the possible errors and changes to be made to the code, let alone predict the results with acceptable accuracy. Without going too far, we can imagine generally obtaining matches between Crossref and OpenCitations dumps (with the former providing more data than the latter). In addition to this, it is impossible for us to accurately imagine the type of libraries, specific implementation logic, algorithms, methods, and functions we will use in this final step, since, as I mentioned before, we are still in the early phase of the project, and it would be counterproductive to take for granted the uses of certain things. As the software comes to life, the protocol will be updated with more precision and consistency.