A peer-reviewed version of this preprint was published in PeerJ on 12 August 2019.

View the peer-reviewed version (peerj.com/articles/cs-214), which is the preferred citable publication unless you specifically need to cite this preprint.

Willighagen LG. 2019. Citation.js: a format-independent, modular bibliography tool for the browser and command line. PeerJ Computer Science 5:e214 https://doi.org/10.7717/peerj-cs.214

Citation.js: a format-independent, modular bibliography tool for the browser and command line

Lars G Willighagen Corresp.

Corresponding Author: Lars G Willighagen
Email address: lars.willighagen@gmail.com

Background. Given the vast number of standards and formats for bibliographical data, any program working with bibliographies and citations has to be able to interpret such data. This paper describes the development of Citation.js (https://citation.js.org/), a tool to parse and format according to those standards. The program follows modern guidelines for software in general and JavaScript in specific, such as version control, source code analysis, integration testing and semantic versioning.

Results. The result is an extensible tool that has already seen adaption in a variety of sources and use cases: as part of a server-side page generator of a publishing platform, as part of a local extensible document generator, and as part of an in-browser converter of extracted references. Use cases range from transforming a list of Wikidata identifiers into a BibTeX file on the command line, to displaying RIS references on a webpage with added Altmetric badges to generating "How to cite this" sections on a blog.

Conclusions. Citation.js is a library supporting various formats of bibliographic information in a broad selection of use cases and environments. Given the support for plugins, more formats can be added with relative ease.

Citation.js: a Format-independent, Modular Bibliography Tool for the Browser and Command Line

Lars G. Willighagen[1]

[1]**Eindhoven, The Netherlands**

Corresponding author:
Lars G. Willighagen[1]

Email address: lars.willighagen@gmail.com

ABSTRACT

Background. Given the vast number of standards and formats for bibliographical data, any program working with bibliographies and citations has to be able to interpret such data. This paper describes the development of Citation.js (`https://citation.js.org/`), a tool to parse and format according to those standards. The program follows modern guidelines for software in general and JavaScript in specific, such as version control, source code analysis, integration testing and semantic versioning.
Results. The result is an extensible tool that has already seen adaption in a variety of sources and use cases: as part of a server-side page generator of a publishing platform, as part of a local extensible document generator, and as part of an in-browser converter of extracted references. Use cases range from transforming a list of Wikidata identifiers into a BibTeX file on the command line, to displaying RIS references on a webpage with added Altmetric badges to generating "How to cite this" sections on a blog.
Conclusions. Citation.js is a library supporting various formats of bibliographic information in a broad selection of use cases and environments. Given the support for plugins, more formats can be added with relative ease.

INTRODUCTION

With the primary goal of scholarly publishing being the distribution of knowledge, it is important that this knowledge is actually distributed properly, which includes the accessibility and findability of these publications. One way findability has improved in the last few decades, is the transition from text-based citations to the use of Persistent IDentifiers (PIDs), with Digital Object Identifiers (DOIs) being the most common. These PIDs are linked to central stores that provide machine-readable bibliographic information.

However, there are many different stores, most with their own formats: Wikidata (Malyshev et al., 2018) and WikiCite (Taraborelli et al., 2017) have their own scheme; DataCite has DataCite eXtensible Markup Language (XML) and JavaScript Object Notation (JSON) (De Smaele et al., 2017); and CrossRef has CrossRef Unixref XML (noa, d). Similarly, most citation managers have their own formats too: Zotero and EndNote have their own schemes (Vinckevicius, 2017; noa, c), and Office Word has an XML namespace (noa, b). On top of that there are a lot of old and new formats created for a variety of reasons, like BibTeX (Patashnik, 1988), Citation Style Language (CSL) JSON (Zelle, 2012), and Research Information Systems

(RIS) (noa, 2012).

This leads us to two problems. First, citation managers need to maintain parsers for multiple formats. Second, existing managers, like Zotero, either require a server or have entirely proprietary backends. This paper introduces Citation.js, which is a set of different parsers and formatters that works anywhere. By using JavaScript, it can be used as a script, a server, or in the browser. To better suit individual needs, and to minimize unnecessary code, which is especially important in the browser, Citation.js is fully modularised. Formats are bundled in thematic plugins, which can be installed separately. For formatted bibliographies and citations, CSL templates and locales are used with citeproc-js (Zelle, 2012; Bennett et al., 2018). This paper describes how Citation.js is developed, documented, tested, and released.

BACKGROUND

Software Development

The software was developed using modern standards: version control with Git, semantic versioning for releases (Preston-Werner, 2013), open source archives on GitHub (`https://github.com/larsgw/citation.js`; `https://github.com/citation-js`) and Zenodo (`https://doi.org/10.5281/zenodo.1005176`), browser bundles with browserify (Halliday et al., 2018), compatible code with Babel (Zhu et al., 2018), integration testing using the Travis-CI service (noa, 2018b), code linting (source code analysis) with ESLint (Zakas et al., 2018) and Standard (Aboukhadijeh et al., 2018), checking `RegExp`'s for ReDOS vulnerabilities with `vuln-regex-detector` (Davis et al., 2018), and detailed documentation with JSDoc (Williams et al., 2018).

The development process took place with Node.js and npm. First off, any changes would be linted and tested with the aformentioned tools. Bugs or new features can also warrant the introduction of new test cases. If the changes work properly, they are then committed into the version control. If the changes warrant a new release, or if enough changes have piled up for a new release, the change log is updated. Updating the version in the package metadata automatically triggers the linters and test runners, preventing accidental mistakes. Afterwards, publishing the package to npm automatically triggers the generation of files necessary for the package. The scripts used for this are described in `https://github.com/larsgw/citation.js/blob/90cd68c/CONTRIBUTING.md#installing`.

Libraries and others

Apart from tools used for development, Citation.js also uses a number of runtime libraries. Their function and the reason for using them is explained below.

`@babel/polyfill` is a runtime library which fills in support for modern APIs on older platforms. It comes with the use of Babel to transform modern syntax for older platforms (Zhu et al., 2018).

`citeproc` is the official CSL formatting library in JavaScript (Bennett et al., 2018; noa, a).

`commander` is a utility library, only used for the Command Line Interface (CLI). It parses the command line arguments and generates documentation (Holowaychuk et al., 2018).

`isomorphic-fetch` is a specific polyfill, a library filling in support, for the Fetch Application Programming Interface (API), a modern way of requesting web resources. It works in both Node.js and browsers (Andrews et al., 2018).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

`sync-request` is a way to request web resources synchronously. While performing such operations synchronously is advised against in JavaScript, it is still useful for non-production scientific scripts, and demos (Lindesay et al., 2018).

`wikidata-sdk` is a utility library for working with the Wikidata API (Lathuilière et al., 2018; Vrandečić and Krötzsch, 2014)

.

RESULTS

Implementation

Citation.js employs a number of ways to achieve a balance between function and ease of use. The program consists of three major parts: the bibliography interface, code handling input parsing, and code handling output formatting. The bibliography interface itself is quite simple; it mainly acts as a wrapper around the parsing and formatting parts. These two parts behave in a modularised way, with a common plugin system.

Input parsing

Input parsing works by registered input formats. These registrations include an optional type recognizer and a synchronous and/or an asynchronous function transforming the input into a format closer to the final format: CSL-JSON. The new input can then be tested again, and will be parsed iteratively until the final format is reached. Plugin authors are encouraged to create input parsers with as small steps as possible, to allow users to input a variety of different formats.

Type recognition is done with a search tree. First of all, types are ordered by the data type of the input. This is one of: `String` (unparsed text, identifiers, etc.), `SimpleObject` (standard JavaScript `Object`), `Array` (a possibly non-uniform list of inputs), `ComplexObject` (other non-literal values) and `Primitive` (numbers, `null`, `undefined`). The data type can be inferred from other format specifications in some cases. Types can also be specified to be a more specific version of something else. For example, a DOI Uniform Resource Locator (URL) is also a normal URL, but should be parsed differently, namely with content negotiation.

Types can then provide a list of predicates, testing if input belongs to that format. To avoid code repetition and make plugin registration code easier to read, certain common tasks can also be accomplished using shortcuts. These shortcuts include testing text against a `RegExp` pattern, checking for certain properties and checking for the value of elements in an array. These properties can also eliminate the need for an explicit data type: for example, if a `RegExp` is provided, input can be expected to be a `String`.

Output formatting

Output formatting is less complicated. Users and developers only have to provide the identifier of the formatter. Further customization can then be done by providing options, which are automatically forwarded to the formatter. This allows the CSL plugin to take in options specifying the template and locale, for example.

Plugin system

Apart from being able to add input and output formats and schemes on their own, it is also possible to add them in a thematically linked plugin. For example, a BibTeX plugin might consist of a parser for `.bib` files, a parser for the resulting BibTeX-schemed JSON, and a output formatter to create BibTeX from other sources as well. This plugin could then be combined with, for example, a Bib.TXT plugin, resulting in

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

```
1  let Cite = require('citation-js')
2
3  Cite.plugins.add('bibtex', {
4    input: {
5      '@bibtex/text': {
6        parseType: { ... },
7        parse (text) { ... }
8      },
9      '@bibtex/object': {
10       parseType: { ... },
11       parse (text) { ... }
12     }
13   },
14
15   output: {
16     bibtex (data, options) {
17   ...
18     }
19   },
20
21   config: {
22     labelForm: ['author', 'title', 'issued']
23   }
24 })
25
26 let bibtexConfig = Cite.plugins.config.get('bibtex')
27 bibtexConfig.labelForm = ['author', 'issued', 'year-suffix']
```

Figure 1. Possible structure of a plugin for BibTeX. In this example package, line 1 loads Citation.js and lines 2-24 adds the plugin. This plugin consists of two input formats (4-13), one output format (15-19) and configuration options (21-23). Lines 26-27 show how this configuration would be used. Some code is omitted for the sake of clarity, and is replaced with ellipsis (`...`).

a JavaScript package or module, which could be published in package managers like npm. Code for this plugin would look like Fig. 1.

For configuring plugins there is also a `config` option. As an example a `labelForm` option is added, which could control the way the BibTeX output formatter generates labels. Users of this plugin can then retrieve and modify this configuration. It is also possible to offer internal functions this way, for more fine-grained control.

Bibliography interface

The methods for parsing input and formatting output are also included in a general class, `Cite`. Class instances also have access to opt-in version control – changes are tracked if an explicit flag is passed – and sorting. The latter currently does not have effect on CSL bibliographies unless set with the `nosort` option, as the styles define their own sorting method.

Supported formats

Table 1 shows the formats supported by Citation.js at the moment.

Distribution

Browser use

For in-browser use, there is also a standalone JavaScript file available. This includes dependencies. This bundle is built automatically when publishing, and is available through a number of Content Delivery

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

Table 1. Input and output format support. This table only shows general support. For example, the "Wikidata" format is both used for Wikidata identifiers and Wikidata API results.

Format	Input	Output
BibJSON	x	
BibTeX	x	x
Bib.TXT	x	x
CSL	(JSON)	x
DOI	x	
RIS		x
Wikidata	x	

Networks (CDNs) that automatically distribute npm packages. The `Cite` class can then be imported and used just as the npm package, barring browser limitations.

For simple use cases like inserting static bibliographies, a separate tool, `citation.js-replacer`, was developed. When included on a page, this replaces every HyperText Markup Language (HTML) element matching a certain selector, with a bibliography.

Figure 2 shows an example of another use case. For example, the basic use can be extended to add additional information to references, such as an Altmetric (Adie and Roe, 2013) score icon or Dimensions citation count (Thelwall, 2018).

npm package

Citation.js is published as an npm package on the main npm registry, as `citation-js`. Use of the package is the same anywhere, apart from platform limitations. For example, synchronous requests for web resources, used to get metadata for DOIs, is limited on Chrome as discussed in Willighagen (2017b). Also, the Node.js platform, not being a browser, doesn't have access to the Document Object Model (DOM), and so can't easily use HTML elements as input or output.

Separate components, including formats not included in the standard configuration are available under the `@citation-js` scope.

Use cases for the npm package include using it when generating content (either at runtime or for static websites) like PubPub (Shihipar and Rich, 2018), and setting up APIs (Willighagen, 2017a). It is also useful for converting metadata when text mining. For example, BibJSON is one of the input formats, and can then be converted to BibTeX or formatted. All citations to GitHub projects were created with a simple script running Citation.js.

CLI use

Simple one-time conversions, with no extensive customization, can also be done with Command Line Interface (CLI). The command requires Node.js and npm and can be installed by running `npm install -global citation-js`, which may require root privileges depending your setup. Alternatively, any commands can be prefixed with `npx` instead. The command can read files with `-i, -input`, input text with `-t, -text` or via standard in. Output can be configured with a number of options detailed in the man file, also available by running with the `-h, -help` option. Any output is then written to a file with `-o, -output` and else redirected to standard out. Additional logging and errors can be found on standard error.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

```
1  <html>
2  <head>
3    <!-- Altmetric widget code --> <script
       ↪  src="https://d1bxh8uas1mnw7.cloudfront.net/assets/embed.js"></script>
4    <script src="https://cdn.jsdelivr.net/npm/citation-js"></script>
5  </head>
6  <body>
7    <div id="element"></div>
8    <script>
9      window.onload = async function () {
10       let Cite = require('citation-js')
11       let cite = await Cite.async('10.1371/journal.pone.0185809')
12
13       let bibliography = cite.format('bibliography', {
14         format: 'html',
15         append ({DOI}) {
16           return ` <span class="altmetric-embed" data-badge-type="badge" data-doi="${DOI}"></span>`
17         }
18       })
19
20       let element = document.getElementById('element')
21       element.innerHTML = bibliography
22       _altmetric_embed_init()
23     }
24   </script>
25 </body>
26 </html>
```

Figure 2. Basic use, including appending data to formatted bibliography entries. Here, line 3 loads the Altmetric widget code, line 4 loads the library, line 10 imports `Cite`, and line 11 creates an interface for a bibliography with one entry, with metadata from a DOI. Lines 13-18 render the bibliography, with line 14 setting the output to HTML and lines 15-17 appending an Altmetric widget to the entry. Lines 20-21 show the output on the page. Lines 9 and 22 are to avoid race conditions in DOM access. Line 3 and 22 initialize the Altmetric badge. In the example, (Hallmann et al., 2017) is used.

Integrations

The Citation.js npm package can also be used as a library to create integrations with, among other things, word processing systems. For example, ReLaXed (Zulko et al., 2018) integrates Citation.js into the Pug templating language to generate citations when creating Portable Document Format (PDF) documents, and the npm package `citation-js-showdown` was created as a demo on how to introduce syntax for citations in markdown.

Performance

The performance of the Citation.js package has been analyzed on a number of different platforms. Between browsers, compiling the script and importing the library takes about 120 ms, compiling itself taking a little less than half of that. Node.js on the other hand takes about 1 s to initialize, both when the source consisting of multiple files is imported, and when a bundle is imported. This is possibly because Chrome caches compiled JavaScript reducing the compiling times from around 50 ms to about 8 ms, as is explained in Alle (2018).

The results were obtained with the default settings for each of the platforms. In Chrome, this was achieved by using guest mode, while Node and Firefox were not configured to begin with.

As shown in Fig. 4 and Table 2, time taken to import the library mainly consists of importing `@babel/polyfill`. This is because adding the polyfills requires repeated feature detection. After

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

Hallmann, C. A., Sorg, M., Jongejans, E., Siepel, H., Hofland, N., Schwan, H., … de Kroon, H. (2017). More than 75 percent decline over 27 years in total flying insect biomass in protected areas. *PLOS ONE*, *12*(10), e0185809. https://doi.org/10.1371/journal.pone.0185809

Figure 3. Result of the code in Fig. 2. Bibliography consisting of Hallmann et al. (2017) in APA style. Note the Altmetric badge at the end.

that the actual code is imported in two parts. In the first part, where core functionality like the Cite interface is loaded, the main culprit is `addTypeParser`, with 0.13 ms per call on average. In the second part, loading output-related code, importing `citeproc-js` takes the longest with a single call of 2.82 ms. Note that that Firefox uses Just-In-Time (JIT) compilation, compiling pieces of code when they are used a lot.

While code execution is one part, one should also look into the file size. This is especially important in the browser, which has to fetch the library when loading the page.

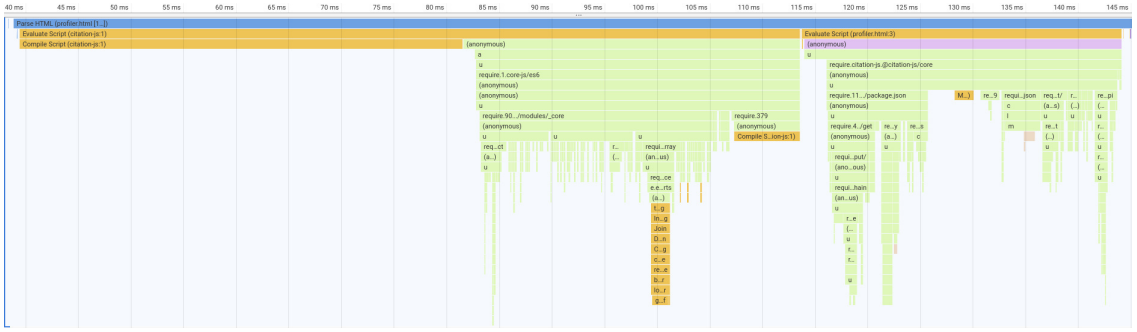
DISCUSSION

Use of formal grammars for parsing

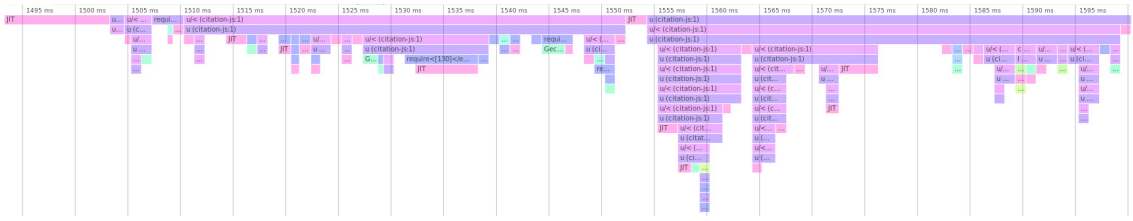
Apart from more common formats like JSON, XML and YAML (YAML Ain't Markup Language), Citation.js has to parse a number of text formats with syntax specific to that format, like BibTeX and RIS. While one can use standard or even built-in parsers for common formats, that is usually not possible for the latter formats. To solve this, one can employ formal grammars, which can be translated into code parsing and validating input. Examples of libraries working with grammars are PEG.js (Majda et al., 2018) and nearley.js (Kartik et al., 2018). Creating grammars has the benefits of not having to write and maintain code validating and parsing input, and having a readable grammar instead of a complex program file.

However, there are also drawbacks. Generating these grammars requires an extra build step, which in the case of Citation.js cannot be integrated with the preceding step due to a lack of appropriate tooling. On top of that, early tests have shown generated code to have has poor performance or large size, and in the case of nearley.js requires a runtime library. Therefore, it may be preferable to write custom parsers in some cases. For example for RIS, which has no balanced brackets or quotes, and does not need much more than simply iterating over the individual lines.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019



(a) Chrome 71



(b) Firefox 64

Figure 4. Initialization performance results on different platforms. Actual timings may vary depending on the device, operating system and cache. Note that Chrome starts with 40 ms of compiling time that is cached on subsequent runs. Firefox compiles JIT, while the code is running. Both graphs show three parts, one loading polyfills from `@babel/polyfill` taking up half the loading time, followed by two parts mainly loading core functionality and plugins respectively. Profiling data is available in supplemental information.

Converting between formats and standardized crosswalks with linked data

Converting input data like parsed BibTeX, BibJSON or Wikidata API results into another format and back can get very repetitive in terms of code. Yet, there are still cases where special handling is needed. Since different formats call for different needs, each plugin has developed its own system to deal with this. Unifying this into a single, performant, reusable and developer-friendly system would be preferable.

To convert one data format (or scheme) to another, a number of different things must be done. First of all, for most properties a simple mapping suffices: `title` in BibTeX refers to the same concept as it does in BibJSON and CSL-JSON. On top of that, the property coincides with `TI` in RIS. This mapping could come in the form of a JSON Linked Data (JSON-LD) context, as done by CodeMeta (Jones et al., 2017).

Second, the data format needs to be converted. While `title` in BibTeX can have formatting in the form of TEX, `title` in CSL-JSON uses a subset of HTML for formatting, and `TI` in RIS does not have formatting at all. Properties can also have different data types. In CSL-JSON, `author` is a list of objects, while `authors` in BibTeX, which describes the same concept, is serialized text delimited by `" and "`. Both of these examples could be handled by defining two converters for each pair of formats, one for converting from A to B and one for converting back. It would also be possible to convert every format to a central format C, and convert C to every format. This is what Citation.js currently does with CSL-JSON as the

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

Table 2. Browser bundle breakdown. Running time is the time it takes to import that part of the script with browserify `require` in the Chrome data set. Note that a big part of the `plugin-csl` "own" code is serialized styles and locales from the CSL repository. Minified the code is 702 kB, which is reduced to 177 kB with gzip and 164 kB with Brotli, both with default compression levels.

Part		Size			Running time
		Own	Dependencies	Total	
Citation.js	Backport	5.9 kB	-	5.9 kB	5.9 ms
	core	99.7 kB	33.9 kB	133.5 kB	8.3 ms
	plugin-bibjson	7.6 kB	-	7.6 kB	3.5 ms
	plugin-bibtex	42.9 kB	-	42.9 kB	2.6 ms
	plugin-csl	87.0 kB	461.8 kB	548.9 kB	3.2 ms
	plugin-doi	6.6 kB	-	6.6 kB	0.6 ms
	plugin-ris	11.1 kB	-	11.1 kB	0.5 ms
	plugin-wikidata	22.1 kB	40.4 kB	62.4 kB	2.2 ms
	name	16.6 kB	-	16.6 kB	1.1 ms
	date	7.3 kB	-	7.3 kB	0.2 ms
Additional	@babel/polyfill	-	197.3 kB	197.3 kB	32.0 ms
	browserify	-	6.8 kB	6.8 kB	0.2 ms
Total		306.8 kB	740.1 kB	1046.8 kB	60.3 ms

central format. It is however important that format C can hold as much information as could be represented in any other format, as to prevent information loss when converting between two formats. This is currently the case with the software entry type, which does not exist in CSL 1.0.1, and is represented by the `book` type by convention.

Third, there might not be a one-to-one mapping between properties. For instance, `pages` in CSL-JSON maps to both `start` and `end` in Citation File Format (CFF) (Druskat et al., 2018). Similarly, `pages`, `issue`, `volume` and `ISSN` are all top-level properties in CSL-JSON, while the corresponding properties in Wikidata are proposed to be nested in the journal property. If the nested values are deserialized, this could be expressed in JSON-LD contexts. However, in the first example it cannot, since JSON-LD cannot distinguish between parts of strings.

Lastly, some mappings are context-dependent. For example, consider the CSL-JSON properties `author` and `reviewed-author` in relation to the RIS properties author (`AU`) and reviewer (usually `C4`). In normal entries, `AU` maps to `author`. However, if the entry being converted is a review, `AU` maps to `reviewed-author` while `author` maps to `C4`.

In conclusion, a number of different things need to be taken into account when writing a system for crosswalks. While a JSON-LD context would scale very well without a central format, most cases restrict the usefulness. Alternatively, a custom system could be developed that defines as much mapping as possible to and from a central format, with special cases for the third and last restrictions. Additional mappings to common formats, to cover properties missing from the central format could be added, with caution not to create critical dependencies on other formats. Or, given the complexity of the systems, it may be best to create for each format, allowing optimization for the special cases of that format.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

Use of GraphQL for API queries

With REpresentational State Transfer (REST) APIs comes the problem of over-fetching and under-fetching. This means that when fetching a resource it may contain too much unneeded information, require additional calls to the API, or both. This causes unnecessary load on both the client and the server, as both have to process more calls with more network bandwidth.

Using GraphQL APIs circumvents this problem by allowing the client to specify exactly what data it needs (noa, 2018a). However, not many endpoints have a GraphQL API available, so this solution cannot be used everywhere.

Support for additional formats

Apart from the formats currently supported in Citation.js (see Table 1), there are plans to include more formats such as EndNote, Machine-Readable Cataloging (MARC) (Avram, 2003), the Zotero scheme and Office XML. These will be published in thematic plugins. For example, formats used to describe software projects are joined in the plugin `@citation-js/plugin-software-formats`. These formats will also include linked data scraped from web pages.

Scraping from source versus fetching from central stores

When getting data from, for example, the Wikidata API or scraped from a web page, that data may be incomplete. However, if part of the data you get is the DOI linked to the entity queried, you could amend that data with data fetched from a central store like Crossref or DataCite. Due to difficulties with prioritizing data sources and non-trivial merging conflicts this has not been implemented yet. Additionally, if the user specifically requests data from a specific API, it can be assumed they want that specific data to be used.

CONCLUSIONS

With the here introduced Citation.js we now have a library that supports bibliographic information in various formats, from multiple sources. By using JavaScript we ensure it can be used in a wide variety of use cases in the web browser, on the command line, and in a server environment. The tool is developed using modern approaches and released via the npm network and archived on GitHub and Zenodo. In addition to machine-readable formats such as BibTeX and RIS, the support for CSL templates ensures that citations and bibliographies can be formatted in many textual representations. Additional content can be easily added to those representations, such as Altmetric icons. The support for plugins allows additional formats to be integrated with relative ease, and without the need of a central repository managing those plugins.

ACKNOWLEDGMENTS

Thanks to JS.org for providing the (sub)domain name for the homepage of Citation.js. Thanks to the many people submitting bug reports, pull requests, and other kinds of feedback during the development of Citation.js.

REFERENCES

CSL Processors - Developers. `https://citationstyles.org/developers/#/csl-processors`.

DocumentFormat.OpenXml.Bibliography Namespace. `https://docs.microsoft.com/en-us/dotnet/api/documentformat.openxml.bibliography`.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

EndNote X6 Help User Guide (For Macintosh). https://web.archive.org/web/20180127234028/http://endnote.com/sites/en/files/support/endnotex6machelp.pdf.

UNIXREF query output format. http://support.crossref.org/hc/en-us/articles/214936283-UNIXREF-query-output-format.

(2012). RIS Format Specifications. https://web.archive.org/web/20120526103719/http://refman.com/support/risformat_intro.asp.

(2018a). GraphQL. https://facebook.github.io/graphql/June2018/.

(2018b). Travis CI - Test and Deploy Your Code with Confidence. https://travis-ci.org/.

Aboukhadijeh, F., Flettre, D., developerjin, Yang, A., Unnebäck, L., Lavieri, D., Sta. Cruz, R., Cousens, D., Greenkeeper, Comnes, B., Littauer, R., Nassri, A., Liu, W., Colvin, J., Barros, R., Robin, Emanuele, Spac, J., Watson, T., Mathur, T., Varela, P., Reggi, T., Wuyts, Y., Chen, S., M., A., Horst, B., Charlie, and Prigara, E. (2018). standard/standard. https://github.com/standard/standard.

Adie, E. and Roe, W. (2013). Altmetric: enriching scholarly content with article-level discussion and metrics. *Learned Publishing*, 26(1):11–17.

Alle, M. (2018). Improved code caching. https://v8.dev/blog/improved-code-caching.

Andrews, M., Tschinder, D., Knuth, E., Josiah, Cirkel, K., Melnikow, P., Evans, R., Giles, S., and Simeon (2018). matthew-andrews/isomorphic-fetch. https://github.com/matthew-andrews/isomorphic-fetch.

Avram, H. D. (2003). Machine-readable cataloging (MARC) program. In *Encyclopedia of Library and Information Science*, volume 3, pages 1712–1730. CRC Press, second edition.

Bennett, F., Adam, Sifford, D., Maloney, C., Ruf, D., Mower, M., C-P, F., SM, M., and Zelle, R. M. (2018). Juris-m/citeproc-js. https://github.com/Juris-M/citeproc-js.

Davis, J. C., Coghlan, C. A., Servant, F., and Lee, D. (2018). The impact of regular expression denial of service (ReDoS) in practice: an empirical study at the ecosystem scale. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 246–256, Lake Buena Vista, FL, USA. ACM Press.

De Smaele, M., Starr, J., Ashton, J., Birt, N., Dietiker, S., Elliott, J., Fenner, M. Hatfield Hart, A., Hugo, W., Jakobsson, S., Bernal Martínez, I., Rücknagel, J. Yahia, M., Ziedorn, F., and Zolly, L. (2017). DataCite Metadata Schema Documentation for the Publication and Citation of Research Data v4.1. https://schema.datacite.org/meta/kernel-4.1/index.html.

Druskat, S., Chue Hong, N., Haines, R., and Baker, J. (2018). Citation File Format (CFF) - Specifications. https://zenodo.org/record/1003149.

Halliday, J., Suarez, A., Kooi, R., Aboukhadijeh, F., Denicola, D., McCarthy, J., Lindesay, F., Popp, A., Stock, T., Lorenz, T., Duff, J., Shtylman, R., Kennedy, H., Harband, J., Patrick, Mao, S., Anatoliy, Metcalf, C., McConnell, C., Kastner, D., Govett, D., Wright, J., Ewald, J., Parodi, A., Comnes, B., Macabiau, C., Gozalishvili, I., and Hanson, J. (2018). browserify/browserify. https://github.com/browserify/browserify.

Hallmann, C. A., Sorg, M., Jongejans, E., Siepel, H., Hofland, N., Schwan, H., Stenmans, W., Müller, A., Sumser, H., Hörren, T., Goulson, D., and de Kroon, H. (2017). More than 75 percent decline over 27 years in total flying insect biomass in protected areas. *PLOS ONE*, 12(10):e0185809.

Holowaychuk, T., Li, Z., Abe, T., Koutnik, R., Geraghty, T., Lukasavage, T., Allen, D., Agius, A., Petersen, J., Neeman, I., De Bollivier, S., Nichols, A., Lyons, P., Sorohan, B., Geisendörfer, F., Yaroshevich, A., Franzoia, A., Ruf, D., George, J., Robertson, J., Hamlet, J., Brooks, M., Thomas, R., Vanesyan, R., Ilia, Buathier, Q., and Nitta, A. (2018). tj/commander.js. https://github.com/tj/commander.js.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

Jones, M. B., Boettiger, C., Mayes, A. C., Slaughter, P., Niemeyer, K., Gil, Y., Fenner, M., Nowak, K., Hahnel, M., Coy, L., Allen, A., Crosas, M., Sands, A., Hong, N. C., Cruse, P., Katz, D., and Goble, C. (2017). CodeMeta: an exchange schema for software metadata. https://raw.githubusercontent.com/codemeta/codemeta/2.0/codemeta.jsonld.

Kartik, Radvan, T., Stewart, A., Corbin, J. T., Windels, R., Marinov, B., Emanuel, K., Viet, L. Q., Itzhaky, S., alex, Litvin, N., Olmsted, S., Hunter, C., Edelman, J., Kanefsky, B., Jake, Aukia, J., Kemp, K. J., Ljunglöf, P., Rose, R., Hildebrandt, S., Trefz, A., Rosenzweig, A., Gunderson, B., Meadors, C., Bertolini, F., Victorio, F., and Quigley, J. (2018). kach/nearley. https://github.com/kach/nearley.

Lathuilière, M., Voß, J., Simantov, M., Willighagen, L., Roberts, L., and offirmo (2018). maxlath/wikidata-sdk. https://github.com/maxlath/wikidata-sdk.

Lindesay, F., Joppi, D. H., Kannan, S., Irving-Beer, A., Double, C., Dascalescu, D., Zaharee, D., Hoffmann, D., Krems, J. O., Hong, J., Bílek, K., Willighagen, L., and Zoltu, M. (2018). Forbeslindesay/sync-request. https://github.com/ForbesLindesay/sync-request.

Majda, D., Ryuu, F.-z., Breault, A., Ruciński, R., felix, https://github.com/josephfrazier, Sampson, A., Tavakoli, A., Vrána, J., Blank, J., Pirsch, F., Ramjiawan, A., Mimms, A., Almad, Neculau, A., Kutil, B., Hearon, C., Davies, J., Doersing, N., Brandt, P., Lukasavage, T., and chunpu (2018). pegjs/pegjs. https://github.com/pegjs/pegjs.

Malyshev, S., Krötzsch, M., González, L., Gonsior, J., and Bielefeldt, A. (2018). Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In Vrandečić, D., Bontcheva, K., Suárez-Figueroa, M. C., Presutti, V., Celino, I., Sabou, M., Kaffee, L.-A., and Simperl, E., editors, *The Semantic Web – ISWC 2018*, volume 11137, pages 376–394. Springer International Publishing, Cham.

Patashnik, O. (1988). BibTeXing. http://mirrors.ctan.org/biblio/bibtex/base/btxdoc.pdf.

Preston-Werner, T. (2013). Semantic Versioning 2.0.0. https://semver.org/.

Shihipar, T. and Rich, T. (2018). PubPub: Open publishing. https://pubpub.org.

Taraborelli, D., Pintscher, L., Mietchen, D., and Rodlund, S. (2017). WikiCite 2017 report. https://figshare.com/articles/WikiCite_2017_report/5648233/3.

Thelwall, M. (2018). Dimensions: A competitor to Scopus and the Web of Science? *Journal of Informetrics*, 12(2):430–435.

Vinckevicius, A. (2017). Zotero to CSL extension and mappings. https://aurimasv.github.io/z2csl/typeMap.xml.

Vrandečić, D. and Krötzsch, M. (2014). Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

Williams, J., Mathews, M., Frank, J., Wrzeszcz, R., Blank, B., Colter, M., Schaub, T., Hayes, L., Haagsman, E., Pozin, K., Kim, M., Phasmal, Harrtell, B., Kerns, B., Chambers, D., Schonning, N., Droogmans, P., Taylor, R., Tutt, B., Parks, C., Locke, K., Dubeau, L.-D., Voyer, V., Nicksay, A., Kienzle, S., Nison, M., and Wytrębowicz, T. (2018). jsdoc3/jsdoc. https://github.com/jsdoc3/jsdoc.

Willighagen, L. (2017a). Citation.js: Endpoint on RunKit. https://larsgw.blogspot.com/2017/08/citationjs-endpoint-on-runkit.html.

Willighagen, L. (2017b). Make a synchronous redirectable CORS request in Chrome. https://stackoverflow.com/questions/45238537/make-a-synchronous-redirectable-cors-request-in-chrome.

Zakas, N. C., Volodin, I., Katz, T., Singh, G., Nagashima, T., alberto, Mills, B., Partington, K., Ficarra, M., Cataldo, K., Kaya, B. Y., Myers, I. C., Schreck, M., VanSchooten, I., DuVall, M., Pedrotti, M., Balocco, V., Aliaksei, Rajavuori, J., Anson, D., Cochard, G., Allardice, J., Fang, P., Hom, V., Pool, J., Harband, J.,

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019

@scriptdaemon, and Vidal, J. R. (2018). eslint/eslint. https://github.com/eslint/eslint.

Zelle, R. M. (2012). CSL 1.0.1 Specification. http://docs.citationstyles.org/en/1.0.1/specification.html.

Zhu, H., Smyth, L., Ng, B., Haverbeke, M., Masad, A., Tschinder, D., Stepanyan, I., Jamie, Ridgewell, J., Ribaudo, N., Sauleau, S., Yavorsky, A., Hanson, A., McCarthy, J., Burzyński, M., Donovan, B., Goldman, S., DiGioia, J., Zilberman, M., Franco, J., Bynens, M., Kappert, L., Kushwaha, P., Bedford, G., Newman, B., Gonçalves, A. A. S., Cataldo, K., and Abramov, D. (2018). babel/babel. https://github.com/babel/babel.

Zulko, Drew, Townsend, T., Ruf, D., Li, J., Mitra, N., Forrest, G., Koska, K., David, and ldrovira (2018). Relaxedjs/relaxed. https://github.com/RelaxedJS/ReLaXed.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27466v1 | CC BY 4.0 Open Access | rec: 5 Jan 2019, publ: 5 Jan 2019