BibItNow! Site Adjustors – How to contribute?

Langenscheiss

February 11, 2018

Welcome

Hej.
If you have decided to contribute by writing your own site adjustors, thanks a lot! This is really going to help me! So let me return the favor and help you getting started with this little step-by-step guide.

1 Step 1 – Get full source code

The code on this github only exposes the parts of the plugin for which I currently (you may always inspire me to change this policy) accept external contributions. However, for testing/debugging purposes, or for figuring out how the code works if you wish, you obviously need the full source code. You can download the latest developer versions for all currently supported browsers from my website.
Since site adjustors work browser-independently, you may pick whatever browser-variant you prefer as your own developer version. In the following, we will denote the local plugin root directory, i.e., where *manifest.json* is located, as "$PLUGINDIR". The corresponding root directory on github will be called "$GITDIR".

2 Step 2 – Enabled desired site

The next step is to add the desired website to the plugin. Open the Adjustor List located in

`$PLUGINDIR/nameResources/urlSpecificAdjustorList.xml`

This file is currently written in the XML format, but will soon be converted to JSON. Add an adjustor entry to this file. The structure is pretty self-explanatory. Specify the URL scheme, and the filenames for the adjustor scripts. For example, if you wish to add the website "https://www.johndoe.com", you specify "johndoe-com" as URL scheme, and insert "johndoe" or whatever filename you like **without the .js extension** as preformatter and/or prefselector. Details about how the specified scheme is matched with the found URL are described in App. B.

Once you have added an entry to the adjustor list, linking an URL scheme to preformatter/prefselector script files, you need to make sure these files exist, as the plugin will crash otherwise if the URL scheme is positively matched. If

you have specified “johndoe” as preformatter, add a copy of “0_TEMPLATE.js” from the preformatter directory on github

$GITDIR/background/preformatters/

to

$PLUGINDIR/background/preformatters/

and rename it to “johndoe.js”. Follow the same procedure for the “prefselector” specified in the prefselector file, with “0_TEMPLATE.js” now of course taken from

$GITDIR/extractors/prefselectors/

and copied to

$PLUGINDIR/extractors/prefselectors/

.

These 3 steps should be enough to let the plugin know that a prefselector and preformatter should be loaded for the given website. If you haven’t adjusted the template files, and everything is correct, the global web console should show the message “This seems to work!” if you activate the plugin popup while surfing on the given website.

3 Step 3 – Find optimal prefselectors

Now, the potentially more difficult part starts. You need to study the source of abstract/article article pages in order to figure out CSS selectors which select the desired citation info to be read by the extractor. The first thing to look for are meta tags. Decent publishing houses put the most relevant citation info into such meta tags. Hence, **BibItNow!** has a fixed kernel of search queries for most bibliography fields – henceforth shortened as “bibfields” – and if you are lucky, those queries are already enough to complete a citation (details about the bibfields and their corresponding default search queries are presented in App. A). However, often you are not that lucky, and you need to be more inventive in providing custom CSS selectors for search queries. Technically, these search queries are carried out PRIOR to those defined in the fixed kernel, thereby suggesting the name “preferred selector” or “prefselector” as a short form.

Once you think you have found the info in the website HTML code[1] and the corresponding CSS selector for, e.g., the author(s) of an article, refer to App. C and to the example file

$GITDIR/extractors/prefselectors/0_EXAMPLE.js

[1]Note that **BibItNow!** queries code **after** the page has finished loading, possibly including effects of dynamically loaded scripts on the website’s DOM. The static source code might hence not always be representative of the information that is available to the plugin. In other words, be sure to inspect the final DOM in case a CSS selector does not find the desired information.

in order to understand how to precisely link a CSS selector to a certain bibfield, which in this case would be *citation_authors*.

You can add as many custom queries as you want, and each query can be further specified with several arguments next to the CSS selector itself. For example, if you have found the author info in the two non-standard meta tags

```
<meta name="bla_author" content="John Doe"></meta>
<meta name="bla_author" content="Jane Doe"></meta>
```

which are not recognized by the fixed kernel, you will have to add the property

```
citation_authors : [['meta[name="bla_author"]','content']]
```

to the JSON object *prefselectorMsg* in the prefselector script file.

As explained in App. A for the author bibfield, the extractor will select EVERY ELEMENT that is found using the specified prefselector string, and concatenates everything into a semicolon-separated list, as the main parser expects this format. So, if the above author information was instead given in a single tag such as

```
<meta name="bla_author"
        content="John Doe and Jane Doe"></meta>
```

you will first have to replace the ”and“ by a semicolon. This is one main reason why **BibItNow!** has preformatters, see Sec. 5. The file “0_EXAMPLE.js“ shows more examples of adding prefselectors. As a general rule, try to use CSS selectors which read the info in a robust way. Remember that websites change from time to time.

4 Step 4 – Parse a link for dynamic citation export if possible

While not necessarily something for your first shot at writing a site adjustor, remember that a core functionality of **BibItNow!** is to communicate with the citation export/”Download citation“-button offered on abstract pages of most publishers/databases. In the absolute majority of all cases, these buttons are technically form submission buttons or simple file links, i.e., something that can be called with an XHR to a specifically formatted URL. Since the main parser is expecting the downloaded citation to be in the RIS citation format, it is highly recommended – yet not necessary – to parse a URL which links to a resource in this RIS format.

To determine the URL per citation, **BibItNow!** provides 2 stages:

1. In the *prefselectorMsg* object, the *citation_download* property allows to query a download link from the abstract page.

2. If such a download link is found, this link together with all extracted static data is passed to the second stage – the *formatCitationLink* function defined in the prefselector script file. Possibly using all static citation data including a citation URL, this function allows to specifiy the XHR method (GET or POST) and requires you to return a formatted and finalized request URL. **Note** that if an invalid URL is returned, or if **no preformatting script has been found for the website**, the plugin will skip the dynamic citation download request altogether. If, however, a request is sent, and if it finishes with status code 200 (OK, successful), the response data will be saved as text into the *citation_download* property of the *metaData* object accessible in preformatting.

There are a number of important rules to obey in using the static citation data, and in parsing the request URL.

1. **Rule 1: Never parse any data to anything but text.** It is sometimes tempting to use *eval* on data, or to assign it to the *innerHTML* property of a DOM node. However, for security reasons, this will not be tolerated in any code of **BibItNow!**, including site adjustors. In particular, Mozilla warns against this practice when submitting web extensions. It is allowed to "try-catch" a *JSON.parse* on text data, as the content of the returned JSON object is interpreted as data only. However, the author generally advises against parsing with anything but string methods if it does not require too much extra code. Reading, e.g., a single property from a JSON string can often be done with a simple regular expression; it does not require you to parse the whole string into a JSON object. Note also that performance is currently not a bottle neck for **BibItNow!**, so you can always afford to manually parse raw strings with regexp magic.

2. **Rule 2: Avoid cross-site and mixed-content requests.** Modern browsers prohibit cross-site and mixed-content XHR. In other words, for the XHR to be successful, you need to stay on the same domain as in the active tab, from which you have extracted the static citation data, and you may not switch between "http" and "https". There is only one exception allowed by **BibItNow!** : cross-site requests to "citation-needed.springer". This exception currently exists because the publisher "Nature" is in the process of merging with the publisher "Springer".

Refer to the example file "0_EXAMPLE.js" for a demonstration of how to successfully parse a citation download link.

5 Step 5 – The most important step: preformatting

As already mentioned in Sec. 3, the raw data extracted from the website source is, in many cases, not immediately ready to be understood by the main parser, and sometimes not even available at all (which means you need to hardcode it). As the name suggests, the purpose of the preformatting stage is to preformat the

data and to correct all these flaws before the main parser takes over. For more details and "hands-on" instructions, refer to the file "0_EXAMPLE.js" in

$GITDIR/background/preformatters/

In the above mentioned example of two author names provided in one meta tag, but not in a semicolon-separated list, the *preformatData* function in the preformatting script file would have to contain a line similar to

```
metaData["citation_authors"] = metaData["citation_authors"]
                              .replace(/[\s]+and[\s]+/gi," ; ");
```

in order to correct for this mistake. The example file illustrates more complex modifications, and App. A explains which bibfield expects precisely which data structure in the main parser. **Note carefully** that the same **rules and restrictions to parsing data as stated in Sec. 4** also apply to the entire preformatting stage.

The preformatting stage is divided into 2 functions that are called in the following sequence:

1. **The function *preformatRawData*** is only called if the dynamic citation download request yielded a positive response with valid, non-empty response data. The *citation_download* property of the *metaData* JSON-object passed to this function then contains the raw response text. In the subsequent parser stage, this text is assumed to represent citation data in the RIS-format! Hence, if this is not already the case at this stage, you need to reformat the data by modifying it in accordance with the restrictions stated in Sec. 4 (see the site adjustor for PubMed as an example of how to deal with this situation!).

2. The function *preformatData* is called in any case. If any data from the dynamic citation download could – after calling *preformatRawData* – be successfully parsed, it will be accessible as a JSON object linked to in the *citation_download* property of the *metaData* object. The bibfields in this object are associated with exactly the same properties as in the *metaData* object itself. For example,

   ```
   metaData["citation_title"]
   ```

   contains the title of the citation as obtained from the static data, while

   ```
   metaData["citation_download"]["citation_title"]
   ```

   contains the title obtained from the dynamic download request in case the latter was successful. **Importantly**, after the *preformatData* function has returned (it returns void), any non-empty string in the JSON object linked to in the *citation_download* property will replace the corresponding static data. In other words, if you, for example, want the plugin to prefer the statically obtained citation title, you will have to add a code similar to

```
if (metaData["citation_title"] != "") {
        let download = metaData["citation_download"];
        if (download != null
                && typeof(download) == 'object') {
                    download["citation_title"] = "";
        }
}
```

to the *preformatData* function in order to erase the dynamically obtained data. Note that since *metaData["citation_download"]* is only an (empty) string if no data was obtained through a dynamic download, you always need to properly check for that to avoid crashes!

6 Step 6 – Debugging and Submission

Once you have written everything, continue testing your site adjustor with various sources on the website of interest. Typically, one overlooks edge cases that need adjustments either in the definition of the preferred selectors, or in the preformatting stage. The more robust your adjustor is, the better.

Finally, once everything is ready, use the github repository to propose a new addition. You do not need to upload a new version of the URL adjustor list. Simply state the URL scheme in the header of your adjustor script files, and the adjustor will be added for the next adjustor upgrade release (which may appear in a different frequency compared to feature updates), given that all criteria for a correct submission are fulfilled.

THANKS!

A Bibfields

Soon!

B URL Matching

Soon!

C Format of prefselectors

Even sooner!