How Important are Usage Examples in REST API Documentation?

S M Sohan, Frank Maurer*
* Dept. of Computer Science
University of Calgary
Canada
{smsohan, frank.maurer}@ucalgary.ca

Craig Anslow
School of Eng. and Computer Science
Victoria University of Wellington
New Zealand
craig@ecs.vuw.ac.nz

Martin Robillard
School of Computer Science
McGill University
Canada
martin@cs.mcgill.ca

***Abstract*—**

I. Introduction

II. Methodology

A. Research Goals

This research is aimed at answering the following research questions:

- **RQ1**. What impact does usage examples have on REST API client developer productivity?
- **RQ2**. What problems API client developers face when using a REST API documentation that lacks usage examples?

B. Requirements

To answer the aforementioned research questions, the study has the following requirements:

- **R1. Realistic.** We had to choose an existing REST API that is currently used by API client developers. Selecting a mature REST API for this study reduces the possibility of an underdeveloped solution that may be found if a new or a seldom used API is selected. A familiar domain needed to be selected so that participants are able to relate to the API features without requiring upfront training. In addition to selecting the API, we had to select tasks that are related to the core features provided by the API to represent reality.
- **Open source.** To be able to understand the impact of usage examples, we needed to select an open-source API where we can add new examples to the documentation for performing this study. For proprietary APIs, it won't be possible to retrofit usage examples on existing API documentation without separating the examples from the documentation.
- **R2. Time bound.** The total time spent by each participant is required to be time bound to measure productivity. As a result, the study needs to be setup such that participants are able to focus on performing the tasks minimizing any overhead.
- **R3. Experienced developers.** Developers with prior experience on REST APIs need to be recruited as study participants to perform the study within a limited amount of time and in a realistic setup. Furthermore, to reduce a learning bias, only participants with no prior experince of using the WordPress REST API V2 are accepted for this study.

C. Study Object

We selected the WordPress REST API V2 for this study. WordPress is a blog-like open-source (R2) framework used by over 409 million people to visit 23.6 billion pages each month. The API allows programmatic access to list, create, update, and delete WordPress data such as blog posts, comments, users, images, tags, etc.

The WordPress REST API V2 has been published and maintained since May 2015. Before January 2017, the WordPress REST API V2 was distributed as a plug-in where WordPress users could optionally install the API component. The following statistics are for the plug-in install numbers between May 2015 and October 2016:

- Total installs: aprox. 248K installs of the plug-in.
- Average daily installs: approx. 500.

Starting January 2017, the WordPress REST API V2 is no longer required to be installed as a separate plug-in since it's pre-bundled with every WordPress install. As per the code repository on GitHub, there are a total of 99 and 46 contributors that had at least one commit to the code repository behind the API and and it's documentation, respectively. These properties satisfy R1, our requirements for being realistic.

By selecting an open-source project we are able to access the source-code to inspect the implementation and documentation technique of the WordPress REST API. The API implements a self-documenting feature where API developers expose API endpoints over HTTP OPTIONS verb to explain the API elements. To implement this feature, the API developers describe the API elements in the code. For example:

Listing 1. Example of self-documenting API Code

```
1  public function get_item_schema() {
2    $schema = array(
3      '$schema'      => 'http://json-schema.org/draft
           -04/schema#',
4      'title'        => $this->post_type,
5      'type'         => 'object',
6      /*
7       * Base properties for every Post.
```

```
8         */
9        'properties' => array(
10           'date'                => array(
11               'description' => __( "The date the
                     object was published, in the
                     site's timezone." ),
12               'type'            => 'string',
13               'format'          => 'date-time',
14               'context'         => array( 'view', 'edit
                     ', 'embed' ),
15           ),...
```

On Listing 1, line 4 specifies that this is a schema definition for the API element $Post$. Then, on Line 10, it defines $date$, one of the properties of $Post$, followed by a human readable description and and type information. Then, on line 14, the context of this property is mentioned as $view, edit, embed$, meaning that this property will be returned when the $Post$ object is returned, embedded, or can be used as an input for editing.

This self-documenting feature is leveraged to generate and publish the official API documentation as an HTML website. Figure 1 shows a screenshot of the published documentation for the $date$ attribute of the $Post$ API element[1].

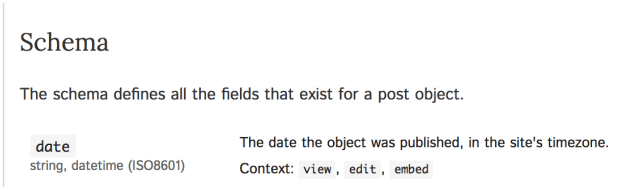

Fig. 1. Screenshot showing auto-generated documentation for Post/date

In addition to the auto-generated documentation of the the schema and API actions, custom content is added by the contributors to the API documentation project to provide prosaic overviews and usage examples.

D. Study Design

1) Tasks: The participants are requested to perform a total of six tasks using the API including one practice task. All of the tasks are related to a single API element, $Post$. The tasks get progressively difficult, and all but the last task can be performed independently of each other. Participants are requested to limit the total time on the study to a maximum of one hour. Participants are encouraged to proceed to the next task when they are either satisfied about or stuck on the right answer.

To use a REST API, the API client developers need to work with the following four inputs over HTTP:

- I1. Request method
- I2. Request URL
- I3. Request headers
- I4. Request body

To verify the response of an API call, the API client developers can use HTTP response headers and/or HTTP response body. To perform the tasks, the participants are required to use one or more of the inputs I1-4. In the following paragraphs, we describe each task with it's description and the intended test scenario against the aforementioned API input and output information. For each of the tasks, the participants are required to use the same WordPress REST API and its documentation.

T1: ListAllPostsTask. We ask the participants to use the WordPress REST API following its documentation to get a list of the blog posts from a live WordPress site. This is the practice task, and the inputs to answer this task are pre-filled for the participants. It allows the participants to understand the tools used for this study as well as get familiarity with the Post API. The answer for this task makes use of I1 and I2.

T2: FilterPostsByAuthorTask. The participants are asked to use the API to filter the list of posts by an author given the author's user name. To answer this correctly, the participants are required to first make an API call to get the numeric ID of the author given the string based user name. Then, the ID needs to be used on the Post API to filter posts by the author. This task allows us to understand the impact of usage examples on API client developers when multiple API calls need to be made to perform a task using the API. Inputs I1-2 are required to complete this task successfully.

T3: ExcludePostsByIdsTask. We ask the participants to use the API to get a list of all posts excluding posts with IDs 1 and 4. Participants need to use the inputs I1-2, and use a desired format on I2 to pass an array of IDs as a parameter. This task allows us to understand how API client developers identify the format for using an array within the URL with respect to the usage examples in the API documentation.

T4: FindTotalPostsTasks. This task requires the participants to use the API to find a total number of posts. Participants need to use the inputs I1-2 and inspect the HTTP response headers to successfully complete this task. This task allows us to understand how API client developer productivity is affected with respect to missing examples about HTTP response headers in the API documentation.

T5: PublishPostTask. We ask the participants to use the API to publish a blog post with a specific title, content, and a published date. To successfully complete this task, the participants are required to use all four input types and inspect both the HTTP response header and the response body. Additionally, the participants are required to use a specific date format that the API accepts as a valid format for date specification. Answers to this task allows us to study API client developer productivity with respect to the usage examples lacking details about the inputs I3-4.

T6: UpdatePostTask. We ask the participants to use the API to update a blog post that they published in T5 with a new excerpt. Similar to T5, this task requires the use of inputs I1-4, but with different values for the inputs. This task allows us to understand API client developer productivity on inter-dependent tasks with respect to usage examples.

To summarize, the tasks allow us to understand how REST API client developers approach API tasks of different com-

[1]http://v2.wp-api.org/reference/posts/

plexity levels involving various input types and available output information with respect to the usage examples in the API documentation.

2) Subject Selection:

3) Process:

4) Data Collection:

5) Data Analysis: Success, Failure, Partial:

III. Results

Observation and Implications For each task, describe the results. Provide examples of failed attempts with counts, and relate to the documentation sources to share an observation. Imply what needs to be done.

IV. Threats to Validity

V. Related Work

VI. Conclusion

Acknowledgment