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.
- **R2. 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.
- **R3. 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.
- **R4. 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 API

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) Participant Selection: To satisfy the requirement of developers with REST API experience (R3), we have used the following criteria for recruiting the study participants:

- Currently working as a software engineer.
- At least 1 year of industry experience as a software engineer.
- At least 1 year of industry experience with REST APIs.
- No prior experience with WordPress REST API.

Participants were recruited through online announcements posted on Twitter, Facebook, and software developer focused mailing lists.

3) Process: Two pilot studies were performed to evaluate and understand an effective process for performing this study. The first pilot study involved four participants that were invited to join the first author on this paper in-person or using a video conferencing software. The study involved tasks using two APIs, the WordPress REST API V2 and the GMail REST API. Each participant was given one of the two APIs and a set of tasks to complete using the API within an hour. Participants were given an online answer form to record the answers to the tasks. The primary findings from this pilot are as follows: 1) asking participants to use an API to perform the tasks required significant overhead time for them to setup a development environment with the proper API credentials, 2) the intermediate trial attempts of using the API are potentially more valuable than the final answer as it allows us to understand API client developer information needs that may are not answered by the documentation, 3) the number of tasks for the study had to be reduced so the participants could complete the tasks within the one hour limit, and 4) for GMail API, participants used up a large portion of their time on setting up their API credentials that requires understanding of OAuth.

To overcome the shortcomings found from the first pilot study, we decided to develop a web-based REST API explorer as shown on Figure 2 that allows participants to use their browser to make the API calls without setting up any development environment. The web-based API explorer only requires the inputs I1-4, and displays the HTTP response headers and body on the click of a button. Thus the participants could focus on using the right input and verifying the output without having to write any code. The web-base REST API explorer also allowed us to automatically collect all the trial API calls that the participants make for each API task. A second pilot study involving seven new participants was performed to understand the features of the web-based REST API explorer in practice and to improve the user interface based on feedback from the participants. Participants completed the study on their own without having to meet in-person or over video conferencing. Only the WordPress REST API was used to focus on REST APIs without the required learning curve associated with OAuth. We found encouraging results from this pilot study as the collected data showed patterns of mistakes that API client developers make that can be reduced by adding usage examples in the API documentation.

Based on the lessons learned from the pilot studies, we designed the actual study process as follows: the original WordPress REST API documentation was forked and a total of 3 API usage examples were added to show listing of blog posts with query parameters for filtering, a request to create a blog post and a request to update a blog post. Figure 3 shows a screenshot of the original API documentation related to T4 where the API client developers are provided with a reference table describing the different properties that can be used to create a $Post$ object[2]. Figure 4 shows a screenshot of the forked API documentation with a cURL based usage example. cURL is used because it's used by elsewhere in the original API documentation. In the forked API documentation, the example shows one possible API call with realistic values for the data that is described in the reference table and associated API response headers and body.

Participants were divided into two groups, G1 and G2. G1 participants were provided with a link to the official API documentation on the web-based API explorer, and G2 participants were provided with a link to the forked API documentation with usage examples. The web-based API explorer allocated more participants to G1 compared to G2 because we wanted to better understand the impact of the lack of usage examples on API client developer productivity. However, each individual participant was randomly assigned to a group by the web-based API explorer. All participants were given the same set of API tasks and were requested to limit their participation time to a maximum of one hour. No task specific time limit was imposed because we wanted participants to spend sufficient time on each task without forcing them to move the next one. Participants were allowed to access the internet and external resources alongside the provided API documentation to perform the tasks as they'd normally use on a typical work day.

4) Data Collection: The collected data for this study is both quantitative and qualitative. The quantitative part comprises of demographic information, statistics about the API tasks, and a numeric rating of the given API documentation to answer R1. We collected the demographic information to record the business and industry experience information. For each participant, we recorded all trial API calls with timestamps, and compared against the required API call to annotate the tasks answers as successful, partially successful, or unsuccessful. The quantitative data is used to measure REST API client developer productivity as a tuple of success rate, average number of trial API calls for each successful API call, and the average time taken for each successful API task completion by participants in G1 and G2.

The data from qualitative analysis comprises of a text document that is exported from the database behind the web-based REST API explorer. For each API call made by the participants, inputs I1-4 along with the HTTP response headers

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

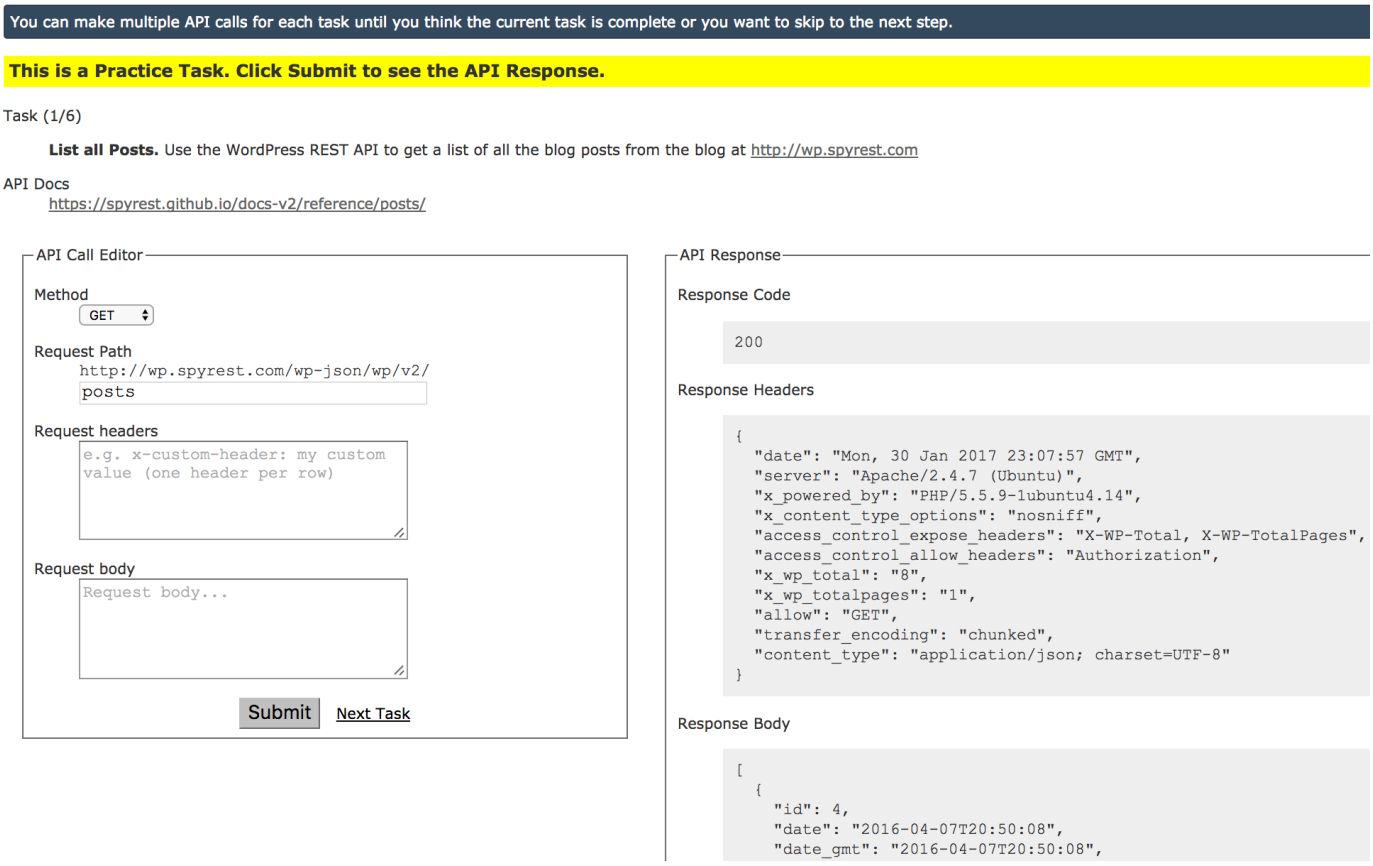


Fig. 2. Screenshot of the web-based REST API explorer

and body are automatically saved on a database by the web-based API explorer. Additionally, the participants provide a free-form feedback about the experience of using the given REST API documentation to perform the tasks. The qualitative data was analyzed to answer R2.

5) Data Analysis: Success, Failure, Partial:

III. Results

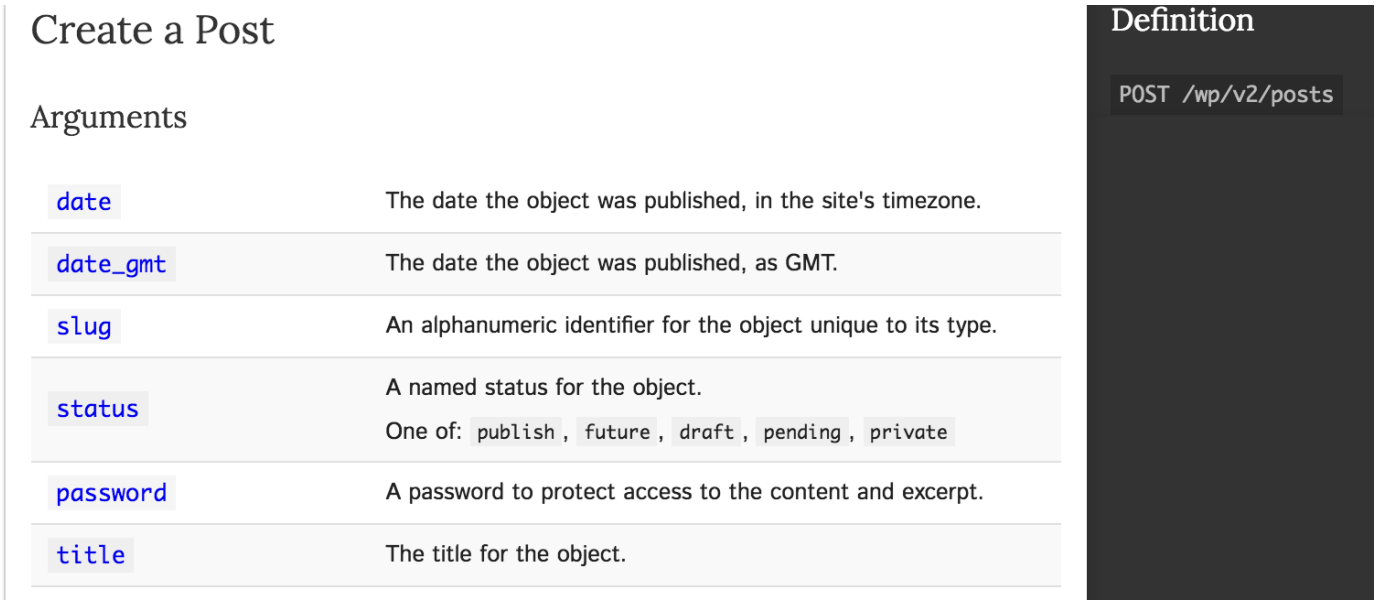
Observation and Implications For each task, describe the results. Provide examples of failed attempts with counts, and relate to the documentation sources to share an observation. Imply what needs to be done.

IV. Threats to Validity

V. Related Work

VI. Conclusion

Acknowledgment



Create a Post

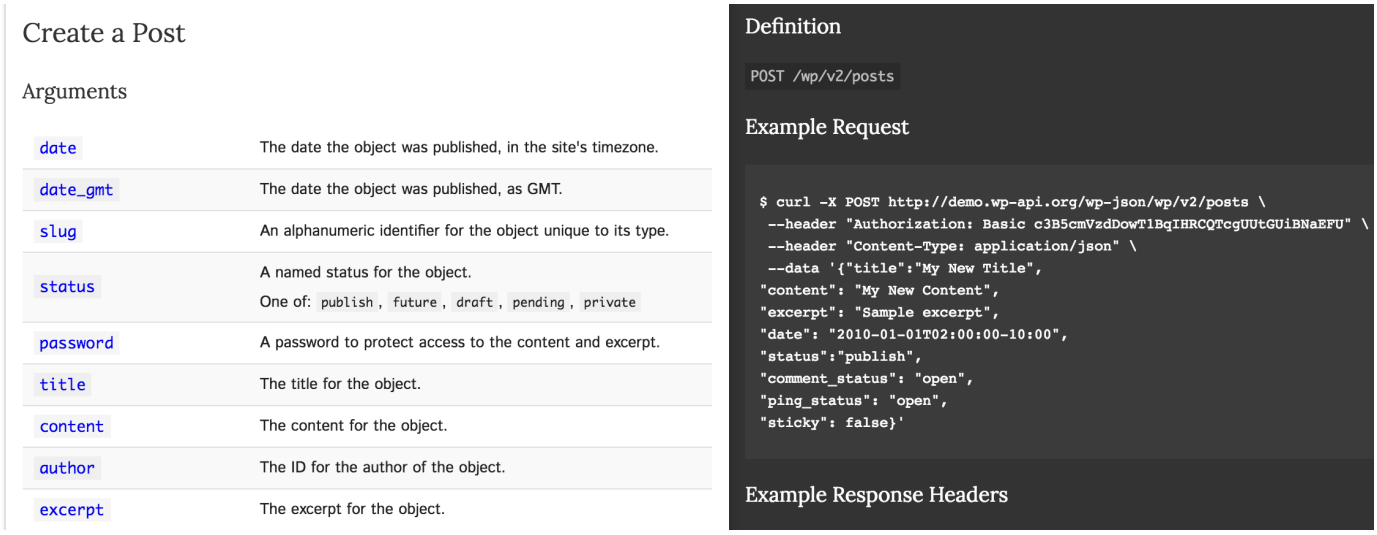
Arguments

`date`	The date the object was published, in the site's timezone.
`date_gmt`	The date the object was published, as GMT.
`slug`	An alphanumeric identifier for the object unique to its type.
`status`	A named status for the object. One of: `publish` , `future` , `draft` , `pending` , `private`
`password`	A password to protect access to the content and excerpt.
`title`	The title for the object.

Definition

```
POST /wp/v2/posts
```

Fig. 3. Screenshot of the original WordPress REST API documentation



Create a Post

Arguments

`date`	The date the object was published, in the site's timezone.
`date_gmt`	The date the object was published, as GMT.
`slug`	An alphanumeric identifier for the object unique to its type.
`status`	A named status for the object. One of: `publish` , `future` , `draft` , `pending` , `private`
`password`	A password to protect access to the content and excerpt.
`title`	The title for the object.
`content`	The content for the object.
`author`	The ID for the author of the object.
`excerpt`	The excerpt for the object.

Definition

```
POST /wp/v2/posts
```

Example Request

```
$ curl -X POST http://demo.wp-api.org/wp-json/wp/v2/posts \
 --header "Authorization: Basic c3B5cmVzdDowT1BqIHRCQTcgUUtGUiBNaEFU" \
 --header "Content-Type: application/json" \
 --data '{"title":"My New Title",
"content": "My New Content",
"excerpt": "Sample excerpt",
"date": "2010-01-01T02:00:00-10:00",
"status":"publish",
"comment_status": "open",
"ping_status": "open",
"sticky": false}'
```

Example Response Headers

Fig. 4. Screenshot of the forked WordPress REST API documentation with a usage example