The Uses of Interactive Explorers for Web APIs

John Daughtry
Software Engineer
Google, Inc.
Seattle, WA, USA
daughtry@google.com

Andrew Macvean
User Experience Researcher
Google, Inc.
Seattle, WA, USA
amacvean@google.com

Luke Church
Software Engineer
Google, Inc.
London, UK
lukechurch@google.com

Abstract

Interactive method invocation has become a common interaction pattern in the documentation of web application programming interfaces (APIs). One of the earliest examples of this pattern being applied at scale is the Google APIs Explorer. In this paper, we describe eight ways developers use such tools in software development, grounded in empirical analyses of the Google APIs Explorer. We then explain the utility of these tools by tying the use cases back to prior literature on programming.

CCS Concepts • **Software and its engineering → Integrated and visual development environments**;

Keywords Software Engineering, Human-Computer Interaction, Documentation, Application Programming Interfaces

1 Introduction

Web APIs are APIs invoked via HTTP. While there are a variety of forms these APIs can take (e.g., SOAP, JSON/REST, or gRPC), generally speaking they are a set of functions that are invoked with parameters, do some processing, and return a response.

In a 2013 survey of 200 executives at companies with over $500 million in annual revenue, 77% of respondents rated web APIs as important "to determining the overall market position of companies in [their] sector over the next five years" [4]. Since that time, business practices reflect this sentiment; ProgrammableWeb reports that since 2014, nearly 2000 APIs have been added to their directory per year and in early 2017 the directory crossed the 17,000 mark [12].

APIs are a form of user interface; they connect a human (a developer) with a computer (the capabilities of a backend system). APIs are hard to use for a variety of reasons, including documentation, complex interrelationships between API components, inappropriate abstractions, and incongruence between coordinating APIs [13]. As user interfaces, principles and methods from human-computer interaction can be applied to improve their usability, as Myers and Stylos have demonstrated in a variety of contexts [9]. Even if a developer manages to make an API work, there can be severe issues with the resulting code, such as major security holes [1].

PLATEAU'17 Workshop on Evaluation and Usability of Programming Languages and Tools, October 23, 2017, Vancouver, CA

Hou and Li identified common patterns in online help requests surrounding the learning and use of APIs [3]. Two related paths towards solving these issues are the study of API structure and documentation. The space of API design decisions can be formalized [14]. And we can build knowledge on top of those formalizations; For example, the placement of a method within an API can have a profound impact on development speed [15].

However, not all usability problems are best solved through API design itself, but rather through documentation and tools. Robillard and Deline find that the "biggest hurdle when learning an API is the documentation" [11]. A 2009 survey of software developers at Microsoft found that 78% of respondents claim to learn about an API by reading documentation, while 55% claim they used code examples [10]. In the same survey, the author found that inadequate or missing resources provided the biggest obstacle to learning APIs. "If the documentation for an API is good, it solves 99% of your problems" [11].

There is also a growing body of anecdotal evidence that API usability, heavily influenced by the quality of the documentation and learnability of the API, impacts adoption of an API. Myers and Stylos state that "usability can also affect API adoption; if an API takes too long for a programmer to learn, some organizations choose to use a different API or write simpler functionality from scratch" [9].

The Google APIs Explorer - hereafter referred to as Explorer - is a tool that was developed to make it easier for developers to find and try out Google's APIs before adopting them, without a significant time investment [6]. It can be embedded within API documentation or used standalone. There are numerous similar tools, including the Apigee API Console and the Swagger UI. Over the past few years, these tools have quickly moved from being a novel technique to being a standard pattern applied in industry [8]. Generally speaking, the designs vary slightly, with these elements serving as the common thread:

1. For each API method, a graphical form that supports entering the request parameters and a submit button that executes the request.
2. Slightly more advanced widgets to support setting the appropriate form of authentication (e.g., OAuth), payloads, or headers.

3. Response rendering that depicts the results in a human-readable format (e.g., formatting, syntax highlighting, element folding).

While Explorer was motivated by a desire to provide an easier way to try out an API before adoption, we are aware of no analysis of how these tools are used in practice. Thus, in this paper, we explore the following research questions from various angles:

- **RQ1**: How do developers make use of interactive call invocation in their work?
- **RQ2**: Why do these tools provide utility to developers?

2 Uncovering the uses

To uncover the answers to RQ1, we analyzed qualitative data in StackOverflow reports that mentioned Explorer. We opted to use StackOverflow reports instead of other data sources (e.g., bug reports) because they provide richer contextual information and are more likely to illuminate successful uses of Explorer. In all, 192 threads were analyzed with the following coding scheme:

- How were they using Explorer?
- What problem did it solve?
- What problems did they have using Explorer?

There was only one coder, but archetypes for each resulting use case were reviewed in a group setting after the analysis was complete.

The purpose of this analysis was not to uncover the distribution of usage across the use cases but rather, to uncover the variety of use cases in which Explorer is utilized. Themes were extracted as the posts were categorized.

There are a variety of stakeholders involved in APIs. In this paper, we adopt the nomenclature described by Stylos and Myers [14]. Specifically, we refer to the developers that build the APIs as API designers and the developers that code against the API as API users.

We found that:

- **API users use Explorer to**:
  1. Find APIs and elements within APIs
  2. Evaluate the utility of APIs
  3. Learn how to invoke APIs
  4. Collaborate with other developers
  5. Use an API without writing code
  6. Reproduce results
- **API designers use Explorer to**:
  7. Learn how to invoke their own API
  8. Test API backend
  9. Test API deployment

We explore each use case in greater detail in the following sections.

2.1 API Users finding APIs

Some interactive API method invocation tools are not integrated with traditional API documentation, while others are. We found evidence that in the case of Explorer, when used as a standalone tool, some developers do use it in place of documentation. Although the primary API documentation contained full descriptions of all API methods and parameters, and contain significant additional information such as code samples and troubleshooting tips, developers still use Explorer in place of said documentation. For example, we found evidence of API users sharing Explorer links when docs links would have been more appropriate. This usage provides evidence that in some cases, developers use Explorer as documentation, which will be discussed in more detail in later sections.

2.2 API Users evaluating the utility of APIs

We found evidence of API users using Explorer to evaluate whether or not an API method will suit their needs. If a particular API call seems to provide a piece of data that you need, Explorer can be used - without signing up for the API, provisioning credentials, or writing any code - to determine whether or not the API does indeed meet your expectations; it allows you to quickly evaluate the utility of the API. This is the use case described in the original launch documentation [6].

2.3 API Users learning how to invoke APIs

When you are trying to invoke an API, it can be difficult to format the request correctly, as discussed previously. We found evidence that API users use Explorer to learn how to format their requests. In some cases, API users were trying to generate the correct payload structure. In other cases, we found evidence of API users leveraging Explorer to learn how to construct curl requests from the command line, using Explorer to create the successful request and browser tools to turn the successful request into a format appropriate for curl.

2.4 API Users collaborating with other developers

When you are having issues writing code to invoke a web API, the simplicity of a form-based invocation allows for collaboration across disparate skills. For example, it becomes trivial for a JavaScript programmer to assist a C developer. An API user can ask a question such as: *why doesn't this call work?* and provide a link to Explorer, which depicts the failing method invocation in language-neutral context.

2.5 API Users invoking Explorer without writing code

When, in the course of software development, you find the need to invoke a web API, this does not necessarily mean that it is worth your time write code to invoke the API.

Perhaps you need to invoke the API once, and only once. Why write code when you can execute the API using a form on a webpage? We found evidence of this in the data, as well as evidence of repeat usage over time (i.e., using Explorer to make an invocation every time it needs to be made).

2.6 API Users reproducing results

Another use case that we discovered was using Explorer as a tool to reproduce issues. Let's say that your code is getting an error, and you don't know whether this is unique to your invocation, or a broad behavioral issue with the API. If you can replicate the behavior in Explorer then you can isolate the issue to either the parameters you are passing into the method or the API behavior itself. Thus, we see evidence of API users making claims such as: *I see this same result when I invoke the API via Explorer* when filing or discussing API issues.

2.7 API Designers learning how to invoke their API

Interestingly, we found evidence of API designers leveraging Explorer to learn how to invoke their own APIs. In retrospect, this makes sense. For example, just because you write the code that tells your system to require authentication doesn't mean that you know anything about how to perform the authentication required to make a call succeed. While this is an extreme example, it illustrates the point that API designers can face many of the same hurdles as API users when trying to invoke their own APIs.

2.8 API Designers testing a backend

We also found evidence that API designers use Explorer to test changes to their API. Rather than (or perhaps in addition to) testing their API via code, API designers can use Explorer to test changes to their API surface (e.g., authentication or required parameters), API behavior (e.g., latency or response codes), or backend code (e.g., does the API behave the same now that I've redeployed) as they iterate on their API.

2.9 Testing API deployment

If you are an API designer, and you want to deploy a new instance of a web API, whether to your local machine or to the cloud, there can be a number of steps involved in setting up your new environment. We found evidence of Explorer being used by API designers to test whether or not a deployment was successful.

3 Distribution

In the previous section, we summarized our analysis of how Explorer can be used in the lives of programmers. We found that it served not only API users, but also API designers. And, we uncovered a variety of use cases spanning a variety of development activities, from testing API design changes to collaborating over an API consumption issue. Given these use cases, we next turned to the question of distribution.

Does one particular area of development dominate our usage? To answer this question, we turned to a user survey. Based on our prior analysis, our hypothesis was that Explorer usage would not be dominated by any particular development stage. To test this hypothesis, we implemented into Explorer an in-product intercept survey [7], which surveyed a random subset of users as they worked with the tool. We asked developers to self-report to us the activity in which they were engaged: evaluating the capabilities of an API, testing an API they themselves were building, learning how to use an API, or debugging their use of an API. In total, we gathered feedback from 129 users over the course of a 2 month period.

The results are presented in figure 1, and support our hypothesis that no single use case dominates the usage of Google's APIs Explorer.

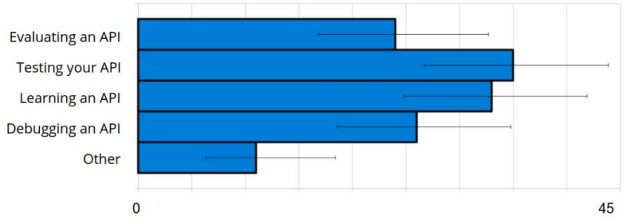

Figure 1. Explorer usage breakdown with 95% confidence intervals.

4 Usage Context

The general interaction pattern employed by the Google APIs Explorer can be housed in many contexts. Indeed, at the time of our analyses, Explorer was available from within our developers console, embedded within our documentation, and as a standalone tool. We analyzed months of usage logs, and again found evidence of Explorer being used as documentation, as well as evidence that Explorer is often used by developers who read API method documentation.

4.1 Standalone Explorer is used as documentation

Explorer is available as a standalone tool as well as being available as a widget within documentation. We knew from our initial analysis that Explorer can be used as a documentation source, so we measured the percentage of sessions where a user navigated the standalone Explorer resource hierarchy but never clicked our Execute button. Our hypothesis was that if Explorer was heavily used as a documentation tool, then we would see that usage reflected in this metric. Indeed, we found that during a one month interval, >50% of usage sessions demonstrated this pattern.

4.2 Explorer is often used within method documentation

If Explorer serves such a variety of use cases, then we hypothesized that its usage within API documentation sessions should be relatively high when a user navigates to the documentation of a specific API method. What we found during our one month analysis window was that nearly 1 in 4 API documentation sessions involved using Explorer. This is significantly higher usage than we expected given that we embedded Explorer far below the fold on these pages (at the time).

5 A tool to overcome barriers

While the analyses presented in earlier sections satisfied our desire to understand how Explorer is used by developers, we did not feel that it adequately addressed our second research question; why are tools like Explorer so useful to developers?

Ko, Myers, and Aung [5] described six barriers to programming. In this section, we discuss our previous analyses through the lens of these barriers.

5.1 Design Barriers

A design barrier is encountered when a developer doesn't know what they want to build. We did not find any evidence of Explorer assisting with design barriers.

5.2 Selection Barriers

A selection barrier is encountered when a developer knows what they want but don't know what to use to accomplish their goal. With respect to web APIs, this includes identifying which method to use or which parameters to pass to a particular method. We saw evidence of Explorer being used in both contexts. For example, Explorer is used to navigate the available methods, execute each quickly (or at least faster than writing code to invoke them) and evaluate the responses, and as a way to collaborate with others to overcome selection barriers.

5.3 Coordination Barriers

A coordination barrier is encountered when a developer knows what things to use but not how to use them together to accomplish their task. With respect to web APIs, this includes the coordination of API invocations with authentication (e.g., OAuth). We saw evidence that Explorer is used to get around authentication coordination issues allowing users to focus on the API specific tasks. For example, you can use Explorer to validate hypotheses about parameters and return types without writing authentication code. We also saw evidence that it helped convey the concepts of authentication and that it can be useful a tool to assist with some of the simplest authentication barriers (e.g., OAuth scope selection).

The collaboration evidence discussed earlier indicates potential improvements that can be made around coordination. For example, cases where a user needs to retrieve a numeric identifier from one method to pass into another method. It is an open research question whether additional lightweight interactions could be automatically generated by the tool to aid in this coordination.

5.4 Use Barriers

Use barriers are encountered when you know what to use, but you don't know how to use it. This is where Explorer's utility is most clear to the casual observer. Explorer provides a quick and easy way to learn about the API parameters and responses without having to simultaneously deal with other issues, including but not restricted to: authentication, client libraries, platforms, credential provisioning, language idioms, and syntactic issues. There are a number of avenues where support for these cases could be improved, to include: conditional field requirements, field format guidance, and error response interpretation.

5.5 Understanding Barriers

An understanding barrier is encountered when a developer knows what they want but it didn't do what they expected. With respect to web APIs, we see evidence of Explorer being used to help overcome these barriers through collaboration. As discussed previously, developers discuss issues over Explorer invocations instead of over code, which removes language and execution overhead.

5.6 Information Barriers

Information barriers are encountered when you have a hypothesis about why something occurred, but can't find information that can verify your understanding. We found evidence of developers using Explorer to quickly test hypotheses that could be hard to test otherwise. For example, a developer whose code is failing can try the same parameters in Explorer to test their hypothesis that the parameters are correct. Without Explorer, their test would be significantly more complicated to test, and in some cases insurmountable (e.g., a developer than only known Python can't be expected to test this by recreating the call in Java). This is particularly relevant to issues that can surround client libraries, as Explorer exposes the raw API invocations while client libraries often wrap the raw invocation into constructs idiomatic to a particular language. Another example we have seen is testing a hypothesis that the API itself has broken as opposed to the developer's own code.

6 Cognitive Dimensions Analysis

The analysis in sections 2-5 present an understand of how tools like the Explorer are used and why they are of benefit to developers. We can use this information to continue to guide their development using the lens of the Cognitive Dimensions of Notations (CDNs) [2]. In CDNs terms the

activities that the users are performing with the tool are Exploratory Design, Incrementation, and Search. We'll consider the Explorer interface's support for these activities along a few critical dimensions.

6.1 Progressive Evaluation, Viscosity, and Hidden Dependencies

For developers exploring whether an API fits their needs or learning how to invoke an API, it is important that they can get feedback about whether they are going in the right direction and to make changes with as little effort as possible. The Explorer supports this workflow via good progressive evaluation (each call can be run one at a time), low viscosity (everything is atomic, so there is little knock-on or repetition viscosity). The nature of the Explorer as a tool for issuing single concrete API calls results in few hidden dependencies other than on the authentication state of the API which is explicitly displayed. It is of course possible for an API to add hidden dependencies, beyond those in the tool.

6.2 Juxtaposition and Visibility

As well as discouraging hidden dependencies, Web APIs tend to encourage visibility of possible options (via the API listing) and of the information being transferred (via the explicit display of the request and response). Juxtaposition is an important property when debugging, where it is useful to be able to display multiple API calls and results side by side for comparison. This is achieved for free by using a stateless browser based UI.

6.3 Abstraction, Secondary Notation and Sharing

The primary design tradeoff in the simplicity of the Explorer interface is abstraction. The interface has a very low abstraction barrier. The developer needs to understand little more than the API that is being explored. However, it is also an 'abstraction hating' system in that there is little ability to define and reuse new abstractions within the system other than by sharing an API call and its parameters as a URL.

It is an open research question whether there additional lightweight abstraction mechanisms could be supported to enable more complex use cases without interfering with the current usability properties of the system.

As well as abstraction, it's very useful for developers collaborating around an API to be able to add commentary as to what a particular API call is doing, or to comment on the result before sharing. There is currently no explicit support for such secondary notation in the URL sharing mechanism. This means that comments can't be shared associated with the call and instead need to be written in, for example, a separate email. We observed that in the absence of this support separate tools like StackOverflow were used to discuss the APIs, providing Secondary Notation by wrapping the API in free text. Whilst this is clearly effective, it decouples the API from the discussion. We suggest that exploring explicit secondary notation support in the design of interfaces for interactive method invocation would be a fruitful avenue for further research.

7 Conclusions

In this paper, we explored two research questions. First, we addressed the question of how these tools are used by developers in the wild by analyzing the usage of the Google APIs Explorer, and described how Explorer is used in a variety of ways by both API designers and API users. Second, we addressed the question of why these tools provide value by tying them to the barriers of programming uncovered by Ko et al and CDNs. Further, our CDNs analysis highlighted possible areas of improvement via better abstraction support and secondary notation. We see many opportunities for continued applied research within this domain.

We do not claim that our list of uses is exhaustive. This paper is the result of industrial research aimed at informing tool design; while we were conscious of methodological rigor, we stopped our explorations when we felt that we collected the most important and most actionable results. Were we to continue this work with the aim of a complete descriptive theory of use, we would focus on other sources of data, such as conducting similar evaluations on other tools in this space (e.g., SwaggerUI). However, we do feel that it provides enough context and variety to inform the design and maintenance of such tools.

Explorer, and the general interaction pattern it employs, could be broadly characterized as a graphical notation system. Ultimately, our focus was on understanding our tool so that as we built new versions of Explorer, we could retain what was working well for developers and extend the functionality in useful ways. Explorer continues to evolve; we released a new version earlier this year and continue to iterate on it's design.

Acknowledgments

Special thanks to the many people who have contributed to the Google APIs Explorer, including Jason Hall, Jake Moshenko, Tony Aiuto, John Boswell, Joe Ashear, Ryan Kuykendall, Mugur Marculescu, Mo Chang, and Craig Citro. We'd also like to thank those who reviewed drafts of this paper, including John Boswell and Emerson Murphy-Hill.

References

[1] Sascha Fahl, Marian Harbach, Henning Perl, Markus Koetter, and Matthew Smith. 2013. Rethinking SSL Development in an Appified World. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*. ACM, New York, NY, USA, 49–60. https://doi.org/10.1145/2508859.2516655

[2] T. R. G. Green. 1989. Cognitive Dimensions of Notations. In *Proceedings of the Fifth Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*. Cambridge University Press, New York, NY, USA, 443–460. http://dl.acm.org/citation.cfm?id=92968.93015

[3] Daqing Hou and Lin Li. 2011. Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension (ICPC '11)*. IEEE Computer Society, Washington, DC, USA, 91–100. https://doi.org/10.1109/ICPC.2011.21

[4] Bryan Kirschner. 2015. The Perceived Relevance of APIs. (2015). Retrieved Jul 2, 2017 from http://apigee.com/about/api-best-practices/perceived-relevance-apis

[5] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC '04)*. IEEE Computer Society, Washington, DC, USA, 199–206. https://doi.org/10.1109/VLHCC.2004.47

[6] Anton Lopyrev and Jason Hall. 2011. Introducing the Google APIs Explorer. (March 2011). Retrieved Jul 2, 2017 from https://developers.googleblog.com/2011/03/introducing-google-apis-explorer.html

[7] Hendrik Müller and Aaron Sedley. 2014. HaTS: Large-scale In-product Measurement of User Attitudes & Experiences with Happiness Tracking Surveys. In *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: The Future of Design (OzCHI '14)*. ACM, New York, NY, USA, 308–315. https://doi.org/10.1145/2686612.2686656

[8] John Musser. 2014. 10 Reasons Developers Hate Your API (and what to do about it). (May 2014). GlueCon.

[9] Brad A. Myers and Jeffrey Stylos. 2016. Improving API Usability. *Commun. ACM* 59, 6 (May 2016), 62–69. https://doi.org/10.1145/2896587

[10] Martin P. Robillard. 2009. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Softw.* 26, 6 (Nov. 2009), 27–34. https://doi.org/10.1109/MS.2009.193

[11] Martin P. Robillard and Robert Deline. 2011. A Field Study of API Learning Obstacles. *Empirical Softw. Engg.* 16, 6 (Dec. 2011), 703–732. https://doi.org/10.1007/s10664-010-9150-8

[12] Wendell Santos. 2013. ProgrammableWeb API Directory Eclipses 17,000 as API Economy continues to surge. (March 2013). Retrieved Jul 2, 2017 from https://www.programmableweb.com/news/programmableweb-api-directory-eclipses-17000-api-economy-continues-surge/research/2017/03/13

[13] Christopher Scaffidi. 2006. Why Are APIs Difficult to Learn and Use? *Crossroads* 12, 4 (Aug. 2006), 4–4. https://doi.org/10.1145/1144359.1144363

[14] Jeffrey Stylos and Brad Myers. 2007. Mapping the Space of API Design Decisions. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '07)*. IEEE Computer Society, Washington, DC, USA, 50–60. https://doi.org/10.1109/VLHCC.2007.36

[15] Jeffrey Stylos and Brad A. Myers. 2008. The Implications of Method Placement on API Learnability. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '08/FSE-16)*. ACM, New York, NY, USA, 105–112. https://doi.org/10.1145/1453101.1453117