<http://www.diva-portal.org>

This is the published version of a paper published in *First Monday*.

Citation for the original published paper (version of record):

Snodgrass, E., Winnie, S. (2019)

API Practices and paradigms: Exploring the protocological parameters of APIs as key facilitators of sociotechnical forms of exchange

First Monday, 24(2)

Access to the published version may require subscription.

N.B. When citing this work, cite the original published paper.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-89509>

API practices and paradigms: Exploring the protocological parameters of APIs as key facilitators of sociotechnical forms of exchange

by Eric Snodgrass and Winnie Soon

Abstract

The use of application programming interfaces (APIs) to develop and design technological forms of structured exchanges is an increasingly prevalent practice at present, allowing data to be shared, automated, circulated and redistributed in wider computational culture. Despite acting as key infrastructural elements and connectors, the prominence and importance of APIs is still under recognised. Via a brief survey of the history of API development and then closer regulatory and technical analysis of the long standing, non-profit *net.art generator* (*nag*) (1997), this study aims to bring to the fore key sociotechnical elements of API design. Particular attention is given to protocological forms of control as these can be enacted through APIs via the regulations, terms of service and specific operational processes and affordances of fetching networked data. *net.art generator* uses the Google API and URL and specific attention is given to Google's setting of terms for exchange. In doing so, we give a few suggestions for how anyone working with APIs might think through certain key questions around the creation and use of APIs, particularly in regards to the parameters of openness, accessibility and terms of inclusivity that APIs set upon practices of knowing, sharing, participation and exchange.

Contents

[Introduction](#)

[Defining terms: A brief history of API development](#)

[net.art generator: Examining protocological parameters of control and exchange in APIs at the regulatory and technical levels](#)

[Moving forward: Key themes and questions in API practice](#)

Introduction

The use of application programming interfaces (APIs) to develop and design technological forms of structured exchanges between software components is an increasingly prevalent practice at present, allowing data to be shared, automated, circulated and redistributed in wider computational culture. Despite acting as key infrastructural elements and connectors within the networked environments of the present, the prominence and importance of APIs is still largely under recognised. This relative lack of discussion on APIs applies not only in terms of their practical usage in things such as application and platform development, but also with regards to the sociotechnical questions of the cultural and political implications of the parameters of accessibility and exchange that APIs have the ability to set, acting in this way as crucial arbitrators for a range of computational infrastructures (Bucher, 2013; Espinha, *et al.*, 2015; Helmond, 2015; Soon, 2016; Snodgrass, 2017; Verborgh, 2017). If, in Helmond's description, APIs can be understood as an "underlying technological glue of the social Web" [1], linking various platforms, applications, and information processes together, it becomes important to pay close attention to what forms of sociotechnical exchanges APIs have tended towards.

Bucher (2013), in one of the few close critical readings of APIs to date, highlights how APIs have become key "protocological objects" of negotiation and exchange. Despite their seeming mundanity and often comparatively simple technical makeup, APIs nevertheless participate in affording a wide range of forms of exchange in their role as connectors of diverse sociotechnical parts and parties. As outlined by Bucher, the task in speaking of APIs is to address the broad range and applicability of APIs in computational culture, encompassing as they do,

a physicality in terms of the corporeal landscape of infrastructure and technology, through to the economic logics at work (*i.e.*, business models, ownership, licencing of the APIs), functions and services (*i.e.*, access to data), practices of users (*i.e.*, forms of labor, play and collaboration), discursive formations (*i.e.*, statements, knowledge, ideas),

rules and norms (*i.e.*, design principles, terms of service, technical standards), as well as social imaginaries and desires. (Bucher, 2013)

At present, there are few existing writings providing an extensive overview of these entwined relations of the social, technical and political questions at play in API design. In addition to writers like Bucher, there have been a few notable discussions within computer science in the past decades around best practices and “styles” of implementation of the networked software architectures that connectors like APIs help to facilitate, with Fielding’s (2000) pioneering work on the commonly invoked REST architectural style being perhaps the most influential of these (used as it is by Facebook and many other of the most dominant networked services of the present). Meanwhile some practitioners in the field of API deployment, such as those working on open source software frameworks for APIs (*e.g.*, the Open API Initiative), have worked to make API providers more aware of the need for further thinking through how APIs as a genre of networked software design might be best practiced within burgeoning and complex API-facilitated ecosystems. Finally, there are writers like Bucher who provide critical understandings to the sociotechnical questions at play in API practices as they pertain to API-supported fields such as app ecosystems and social media platforms, in Bucher’s case by combining insights from fields such as software studies with ethnographic approaches into “how developers produce and make sense of code” in their work with APIs (Bucher, 2013).

This paper represents an attempt to bring together several such perspectives in order to further map out some key issues — past, present and future — at stake in API practices. It begins by establishing a broad definition of APIs via a brief survey of the history of API development. This is followed by an artefact analysis in the form of a detailed regulatory and technical analysis of the long standing, non-profit *net.art generator (nag)* (1997), an online (art) application that requests search engine images via Google’s image search API and recombines them automatically by applying various visual effects, resulting in a new artistic image generated on demand. In doing so, the paper pays particular attention to the sociotechnical underpinnings of API design and their specific protological forms of control as these are enacted through API regulations, terms of service and specific operational processes and affordances of fetching networked data through APIs. The paper highlights the implications of changes to Google’s API policy and how the related *nag* program has been sustained for more than a decade, with a focus on how *nag* has been designed, built, maintained and updated in relation to the changes of Google’s APIs.

Via such an approach, we hope to give some further perspective on the importance of APIs as both practical connective enablers and abstract infrastructures for networked computational practices, highlighting questions of the power and potential of APIs as infrastructural elements, facilitators and arbitrators for the establishing of particular genres and practices of networked and computational exchange. By focusing on both more general but also specific technical practices in API implementation, we aim to give a few suggestions for how anyone working with APIs might think through certain key questions around the creation and use of APIs, particularly in regards to the parameters of openness, accessibility and terms of inclusivity that APIs set upon practices of knowing, sharing, participation and exchange.

Defining terms: A brief history of API development

API is an abbreviation for Application Programming Interface. At a basic level, APIs do the work of exposing and making available the resources of particular programmatic components so that they can be accessed by other programmatic components. APIs typically do so with a view to facilitating forms of exchange and interoperability between these components and the corresponding applications and/or larger systems within which they are situated. Components in this context are the definable and applicable computational resources that will be made available in the creation and use of an API.

In a context of APIs, interface is specifically the set of operational functions and defined parameters that any party using an API must work within in order to enable their component(s) to access another component via the API interface in question. These operational parameters will traditionally include the meaning and limits of information exchanged, calling sequences, data formats and other crucial details of an API’s specification, whose descriptions of intended interactions are typically described by an API provider so as to make clear the available interactions that the API in question aims to permit. As outlined in Shaw and Clements’ article on classifications for networked-based software architectures, an API in this sense can be understood as a mediating “connector” for interaction between components, with connectors in this general sense being any “mechanism that mediates communication, coordination, or cooperation among components” [2].

An API acts as an interface by representing and defining the possible functions of the exposed components in question in the form of tools that express and make available certain functions of the components. In this way, an API provides methods of controlled access to computational components, creating a standardized grammar of functionality in order to facilitate forms of exchange between various components and agents in a manner that typically helps to make them interoperable and independent of their respective implementations. This is done through an API’s establishing of specified procedures, formats and protocols, typically through establishing parameters of access through the assigning of various identifiers, priorities and restrictions in regards to the forms of connection that can occur between prospective components, creating what are sometimes referred to as delegated “endpoints” that can be operated upon within API-facilitated exchanges. In doing so, an API provides a delegated boundary or surface exchange layer of connective functionality between computational components, specifying and facilitating acts of surfacing between components for the purpose of their interaction and exchange. Similarly, in an API’s decision in regards to which components can and cannot be surfaced and shared (whether for security, business and/or practical reasons), an API can be seen as both an entry point into

the black box of a particular computational service, but also as a clearly demarcated barrier towards other possible exchanges with this service.

Additionally, when speaking of interface in the example of APIs, one is speaking specifically of what software engineer David L. Parnas describes as “abstract interfaces” [3]. An API can be understood as an abstract interface in that the key functionality provided by an API is its establishing of parameters for computational exchange, parameters that can in turn be accessed and incorporated for the creation of any number of possible interfaces. In this sense, an API is both a product and a representation of possible products, interface and abstract interface.

At present, most uses and invocations of the term API refer specifically to the category of APIs known as Web-based APIs. A Web API encapsulates and specifies all of the valid messages that two or more computational parties can request and accept while communicating via network protocols. In the case of a Web API, an endpoint is a delegated resource that has been made accessible as a location for defined request-response style exchanges between interested parties in a networked communication channel (usually via a URI, in the form of a URL, to which HTTP requests can be posted).

While functionally different, both network protocols and APIs as a genre participate in the shared task of encapsulating components with clearly specified rules and operators for computational interaction. In attempting a definition of sorts for APIs here, it is worth highlighting how the function of APIs points back to a nested quality of computation in general, and the way in which in software environments this nesting is typically carried out through packagings and repackagings of information. Whether one is speaking of an interface, abstract interface, protocol, API or Web API — each of these various forms of computational agents are essentially performing a general computational function of acting as structuring wrappers of information, describing the executable tasks that can be carried out in any particular computational exchange. As discussed by Langlois, *et al.* (2009) in their helpful tracing of the importance of APIs in the coming into being of “Web 2.0” and its eventual facilitation of the rise of platforms, Web APIs are essentially customized software rearticulations of protocols in the form of computer code that makes use of modularity and recursivity in order to function as a networked protocological interface. With the authors rightly suggesting that, “we need an approach that interrogates the Web as an assemblage of protocols rather than the nesting of one protocol into another” (Langlois, *et al.* 2009).

APIs as a genre of software development can be seen to take shape almost as soon as software development comes into being as a shared discursive practice within computer engineering. In their exposing and hiding of various computational components, APIs have from the very beginning been used to manage various forms of complexity within computational systems, working to reduce the cognitive load of what information a programmer would need to know about a particular system in order to carry out the tasks they were most interested in. In the 1960s, API-like approaches can be witnessed in the example of code library-based interfaces used for the facilitation of software development. Code libraries were used to separate data definitions from specific program implementation, thus allowing for library code to be accessed and deployed by different programs with no direct connection to one another. This helped in facilitating the reuse of code by independent programs or sub-programs (subroutines), with a programmer only needing to know the code library interface and not the internal details of the library itself (as seen in early examples such as the invoking of sub-programs in the FORTRAN programming language). Code libraries and other examples such as device interfaces helped to facilitate innovative approaches for the decoupling and thus easier sharing and distribution of code in these early days of programming.

In their participation of the breaking up of code into specifically delegated components, such approaches to programming can also be seen to emphasise and encourage a modular approach to application building, with one of the chief benefits of such modularity being the ability to reassemble particular components of a system without the need to reassemble the system as a whole. Such modular approaches can be traced in the development of the Unix family of operating systems (Kernighan and Pike, 1984), as well as the object-oriented programming of Simula languages in the 1960s (Madsen, *et al.*, 1993; Dahl, 2004), in which modular approaches facilitate forms of interoperability and exchange within computing [4]. As described by Parnas in the opening line to his well-known early analysis of modularity, in the period of the late 1960s and early 1970s, “If programmers sang hymns, some of the most popular would be hymns in praise of modular programming” [5]. Parnas and others were also able to see early on that if modularity and the interchangeable calling of different computational tasks in the form of “modules” were to be a key feature of programming, then the “design decisions” that go into the “decomposition” [6] of a system into various modules will be of crucial importance in the overall flexibility and benefits of such systems for those working with them. This modular approach and the potential benefits behind it can also be witnessed in contemporary conversations around best practices for the design and development of APIs within greater networked ecosystems (*e.g.*, Amundsen, 2017; Verborgh, 2018).

The 1970s and 1980s saw the rise of the personal computer and also an increase in networked communications of various kinds between computers, together with an increased commercialization of software. In this period, many practices involving the use of APIs shifted to a focus on techniques aimed at both facilitating and easing the distribution and making interoperable of the new masses of shareable data and resources that these new networked environments made possible. Unix itself had developed out of early practices in distributed computing and mainframe time sharing systems, and one key development in the history of network-based APIs is the introduction of Remote Procedure Call (RPC) systems during this time, used to help facilitate forms of client-server interactions, most commonly in the form of request-response message-passing interactions between a local client and remote server. With such forms of inter-process communication, in which a computer program could call a certain local procedure to execute in an external and remote address space, techniques arose in which APIs could now function as intermediaries for the handling and exchanging of data and procedures between local and remote machines. Something that in turn can be seen to have further encouraged certain clearly demarcated subject positions in such client-server relations (*e.g.*, the establishing of a transactional relation of “end-user” and provider sides within such networked environments). In the growing distributed computing environments that often spanned multiple platforms, operating systems and network protocols,

APIs acted as one tool for ensuring a reliable level of interoperability within such complex networked computational environments. As noted by Bucher, APIs in this sense often “signify contracts of sorts, promises of stability and deliverance” (Bucher, 2013).

Alongside growing practices of the time like object oriented programming (OOP) and the ability to use APIs in the establishing of classes of defined interfaces for interacting with data “objects” [2], came the gradual emergence and spread of the World Wide Web (WWW) as a paradigm for networked client-server interactions, with its use of hypertext transfer protocol (HTTP) as an application protocol for distributed computing. In these new environments, the earlier RPC model could be combined with OOP-like approaches to allow for remote access to object instances, resulting in what can be understood as a resource-oriented architecture of resources and resource identifiers within this WWW environment. An API in such instances might be implemented so as to abstract away the underlying transport protocol and data handling layers from the programmer. As would be gradually and more concretely understood over time, the Web, and more specifically, HTTP, becomes in this instance a medium for programming networked interactions via, with the URI — specifically in its common form of the URL — becoming the key representational endpoint for designating and interacting with resources of different kinds (as will be discussed in further detail in the next section).

The quick adoption of the WWW and HTTP generated many new practices around API development. These included the use of the structuring capabilities provided by languages and formats like XML (Extensible Markup Language) and later JSON (JavaScript Object Notation) to create powerful forms of user and machine readable data generation and processing, all of which could be further exchanged between applications of different kinds. The introduction of the programming languages, such as Perl in 1987 and JavaScript in 1995, helped facilitate both client and server side capacities that could be used for the development of Web-based interactions in the form of programs that could be run on a Web server and represented on a Web browser, and would prove to be an essential feature for many Web-based applications and API integrations in the years to come.

Beyond individual examples of Web-based application development, this busy period also saw important developments and discussion on the question of shared Web architectural styles for resource-oriented application development, whether in general discussions around the “semantic Web” and “Web 2.0”, or in concrete approaches such as CRUD (create, read, update, and delete), SOA (service-oriented architecture) and REST (representative state transfer) [8]. This standardising and working with specifically designed architectural styles helped to facilitate greater productivity in handling and making sense of increasingly larger forms of information, aiding the expansion of practices in areas like Web services and platform development. When successfully implemented, resource-oriented APIs built upon clearly defined styles and structures often ended up replacing some of the more baroque forms of middleware that had developed before them, usually by offering increased levels of functional granularity combined with better scalability and practicality in implementation.

From the early 2000s onwards, the spread of Web APIs gathered pace, with a string of early hits in fields such as e-commerce (Salesforce, eBay, Amazon). Their important adoption in early forms of social applications (e.g., LiveJournal, Flickr, Foursquare) were eventually overtaken by the now dominant example of social networking platforms (e.g., Facebook, Twitter, WeChat) and these platform’s own implementations of APIs for giving access to and further circulating their particularly oriented forms of exchange. In each of these cases, the services in question typically provide developer program style initiatives for sharing details and documentation on their various API specifications [9] in a manner that aims to make accessible and promote the uptake of their services on the part of third-party developers. Such APIs opened up a range of data flows between these Web developer services and an API creator’s own service in the form of establishing structured forms of permission to get data from (read) and send data back (write) between the services in question. A key insight on the part of some of the now dominant platform providers as to the value of Web APIs during this period was that APIs could facilitate rapid modes of expansion in regards to the reach of a particular service. As witnessed in popular examples such as Google’s use of APIs as tools for supporting various forms of third-party applications (e.g., Google Maps API), YouTube, Flickr embeds, Facebook’s suite of Social Plug-in and Graph APIs (Snodgrass, 2017), creative bots and other projects using Twitter (Bucher and Pipkin, 2015; Soon, 2018, 2011), along with many more examples, APIs in such cases played a key role in facilitating the projection of a service beyond its own pages and into the WWW and Web-facilitated services as a whole via their uptake by other Web and application developers.

Underpinning such an extensibility to these Web services and platforms, and the transition from so-called Web 1.0 to Web 2.0 (O’Reilly, 2005), was a gradual pivot in understanding the resource-oriented nature of the WWW into something that could be made to be specifically service- or platform-oriented. In this shift, HTML gradually came to be seen as “one among a multiplicity of co-existing and competing languages and protocols”, with the increasingly automated generation of HTML tags, HTTP cookies and carefully defined URLs used as a function for “the re-articulation of hyperlink protocols within other software and protological processes” (Langlois, *et al.*, 2009). With the continued rise of JavaScript and expansion of tools like JSON, AJAX and PHP, it became increasingly possible to submit growing amounts of user-generated content to automated forms of processing of different kinds. Such developments helped to further facilitate key platform-centric accumulators, such as data hoarding, user profiling and algorithmic analysis and filtering of information and content. APIs acting as one specific tool in this rearticulated assemblage of protocols, information and programs. A rearticulation that Langlois, *et al.* describe as the discursive stabilization of this new Web 2.0 world as a “meta-platform” from which a host of new platform-based approaches quite readily evolved out of (Langlois, *et al.*, 2009). As abstract interfaces that facilitate the spread of these aggregate interfaces that we now give the name of platform to, APIs in these instances can be seen as making available and extensible the parts of a particular system so as to turn this system into a more expandable, dynamic, but also oftentimes hegemonic or asymmetrically aligned “ecosystem”.

A constant driver in the creation and adoption of APIs has been their ability to facilitate the sharing or delegating of certain computational tasks with other parties that are better able or more well-placed to

carry out such tasks. This can be seen today in the common example of Web APIs that permit an application to access and leverage another its own already well-established resources, such as making use of secure identification procedures from outside applications in order to simplify or streamline a login procedure (e.g., the Facebook Login or Google Sign-In APIs). Such an implementation of APIs can have the effect of consolidating an API's provider services over time (as seen in the present precedence given to privately controlled login services over earlier non-profit and open standard efforts such as OpenID).

A further important development in practices around APIs in the last decade has been the spread of what has come to be called "cloud computing" and its making available of — via APIs and other methods — a scalable backend of networked data storage space and processing power, the providing access to which has added still further capacities for application building. These include the notable example of Amazon's Web Services and their various interfaces for storing and retrieving large amounts of data from anywhere on the Web in a manner that has given developers access to fast, scalable, reliable and comparatively inexpensive forms of data storage infrastructures and cloud computing capacities. In these examples, the delegating and sharing of data and tasks by an API is not limited to information sharing, but can also be used to access and leverage remote resources and processing power for the running of computationally intensive tasks. This can be seen in the practices of machine learning and their accompanying and growing machine learning infrastructures, in which APIs give access to both large databases for training machine learning processes, and also remote forms of computing power for the often intensive processing of data that is required in machine learning (e.g., Amazon Web Service's various machine learning services, IBM's Watson features, Google Cloud Vision's image recognition API or Microsoft's Azure Text Analytics API).

While these important services for programming in fields like machine learning currently reside in the hands of a few dominant platform providers, Mackenzie points to the way in which the kind of programming with predictive parameters and the constitutive "margin of indeterminacy" [10] that occurs in such modes of programming potentially opens these platforms up to "slippery combinations of openness and opacity" [11], "an opacity that does not derive from the proprietary interests or business models, but precedes it" [12]. This is something that can similarly be said of APIs themselves, which have always been inherently experimental interfaces for creating and testing out forms of interoperability and exchange, with the often mutating parameters in what are at times symbiotic, but also often parasitic or even adversarial forms exchanges pointing to the always ongoing, slippery and political questions that arise in differing constructions of transparency, openness and accessibility.

As important facilitators for information sharing and capacity building, APIs continue to be created, designed and developed. In the Wikipedia (2018) definition of API and in many other common discussions around APIs, a "good API" is typically described as a set of well-defined routines and agreements that can act as useful "building blocks" for programmers by providing useful and valuable forms of programmatic accessibility, applicability, interoperability, transferability and also durability, while at the same time not compromising the individual programmatic components and parties involved in any particular exchange. But what or who defines value and "usefulness", and what might constitute "compromise" in such instances? Having touched here on some of these histories of API development, and in advance of concluding the paper with some suggestions as to how address these important questions of the sociotechnical politics of exchange that APIs play an important role within, it can now be instructive to examine at greater length a specific example, so as to see in more concrete detail some of the ways in which APIs participate in this establishing of different parameters and practices of sociotechnical exchange.

net.art generator: Examining protocological parameters of control and exchange in APIs at the regulatory and technical levels

This section moves from the previous section's overview of the historical development and common practices around APIs to a more specific artefact and API, giving a regulatory and technical analysis intended to highlight certain important sociotechnical aspects of APIs in more concrete detail. We have chosen a long-standing Web art piece called *net.art generator (nag)*, first conceptualized by artist Cornelia Sollfrank in 1997, to closely examine the changes and implications of APIs over time. In this work, a program runs on a browser to fetch and recombine data from the Internet so as to create a new Web site or a new image, challenging notions of authorship, copyright and power structures through the means of automated machinic production. There are five different versions of *nag* that have been realised by seven programmers working at different stages of the project [13]. For the purposes of this article, we examine specifically the image generator of versions 5 and 5b of *nag* (see [Figure 1](#)), which is an endless generative machine that utilizes a Google URL and Google API to fetch data from the search platform, generating a collage image with special visual effects that are based on user input keywords (see [Figures 2 and 3](#)). Versions 5 and 5b were programmed and updated by Panos Galanis in 2003 and Winnie Soon in 2017 respectively, each version implementing significant changes under the hood of *nag* as a result major changes to the API, but in each case with no visibly apparent difference in the user interface between the two versions. This analysis of *nag* is used as an example of an API in action, and of some of the ways in which various forms of protocological and asymmetric exchange can be created within the technical parameters of a Web API.

Figure 1: The interface of *net.art generator* (ver 5b).

Note: Larger version available [here](#).

Figure 2: The user input interface of *net.art generator* (ver 5b).

Note: Larger version available [here](#).

Figure 3: A sample of an image generation by *net.art generator* (ver 5b).

Note: Larger version available [here](#).

In 2015, version 5 of the program experienced a malfunction (see [Figure 4](#)), putting *nag* out of order. This was due to Google introducing a change in their services that barred access to the search URL at the *nag* program and application level (see [Figure 5](#) [14]). The URL can be interpreted not only as a URL, but also as a form of code that can be executed: sending a request from a Web browser to Google to specify data (in the given example, the image keyword search as 'hello'), with Google returning the corresponding images based on the request. But with the malfunction experience, this short line of code can be also understood as more than law (Lessig, 1996; Chun, 2011), because there is a change in access permissions which takes effect from running this piece of code — with both occurring at the same time. This URL functions normally if it is manually accessed through a Web browser, while no images could be fetched or extracted computationally. In other words, this change of permission with its immediate effect suggests the URL is conflated with regulation and code execution. In this code and law relationship, studying code and technical processes unfolds and entails some forms of control.

Figure 4: *net.art generator's* page with an error message.

Note: Larger version available [here](#).

<https://www.google.com/search?safe=active&tbm=isch&source=hp&q=hello>

Figure 5: Search URL in *nag* (ver 5).

Note: Larger version available [here](#).

A change of an API's implementation alters operative and computational processes. Such seemingly small changes on the part of an API provider can have large consequences for those dependent on them. See, for instance, Bucher's (2013) study of the effect of Twitter's changes to their APIs on third-party developers, or Rieder's (2016) posts on the difficulties involved in creating and maintaining effective digital methods techniques and other modes of computational research that rely on API access. In the case of *nag* version 5, it took more than six months to complete the update of code, including to figure out the exact problem and to consider other alternatives and uses of the official Google API.

Although version 5b is up and running again, it contains a deliberate decision to let users experience the limitation of *nag* as it currently stands in relation to the terms of the Google API, where at present only 100 images can be generated per day, thus establishing a limit to this particular generation of discourse through art (see the warnings and errors in [Figure 6](#)). This limitation is directly related to the API's request limit that has been imposed by Google for using their free version of image search APIs. Although employing APIs, in particular Web APIs like Google's, could be considered as an art-making enabler (Soon, 2011) or a connective and generative glue of the social Web (Helmond, 2015) that allows data to be requested, retrieved, remixed, circulated and distributed for cultural, artistic, or even commercial use, it becomes clear that Web APIs, building upon the previous relation between code and law, are also what Bucher (2013) specifically describes as "protocological software objects" that exhibit and execute various forms of regulatory control. Such protocological objects, in the very act of creating functional facilitations of exchange and sharing of data in the form of an API, do so in a controlled manner whose designating of certain parameters of access have the immediate effect of governing the relations within such exchanges. In this context, *nag* partly aims to highlight questions around what forms of exchange Web APIs such as this Google API have entailed, and also what kinds of power relations are exposed through employing APIs to develop a piece of software like *nag*?

Figure 6: The warning and error message after reaching 100 API requests [[15](#)].

Note: Larger version available [here](#).

Bucher examines such questions in relation to the Twitter APIs by using an ethnographic approach aimed at understanding how APIs are used by a community of developers, or specifically how APIs function as “a management style” (Bucher, 2013) where forms of protocological control are used for regulating, governing and executing data processing. Meanwhile, this artefact analysis aims to offer its own insights on these questions by studying the specific technical challenges, changes and updates that have arisen over time during *nag*’s now long-standing and codependent relation with the Google image search API since 2003. In particular, the project is relevant for the way in which it specifically reveals not simply negotiations, but a constant struggle and configuration with this service of Google’s, in terms of policies, charging models, responsibility and, perhaps most importantly, the technical parameters and computational principles of the query processes involved. Some of the most used and important Web APIs can be understood as involving asymmetric forms of protocological exchange. We don’t view APIs simply as technical building blocks or connectors of components for *nag* to generate images, but more generally as protocological and discursive sociotechnical devices for generating discourse to reflect upon the forms of (big) data exchange in contemporary conditions, both technically and culturally (Sollfrank, *et al.*, 2017).

Regulatory parameters of control and exchange

The triggering point on the new version 5b of *nag* was caused by the sudden and instant malfunction of the aforementioned URL from Google in 2015, resulting in failure of any image fetch. The reasons behind this change of decision by Google are unclear, but could likely be related to a perceived violation of Google’s terms and conditions. Google’s Terms of Service prohibit the accessing of their services through what it describes as “automated means (including use of scripts or Web crawlers)”, except “through the interface that is provided by Google” (Google, 2007). The key term here is “automated means”, which provides an answer on why the URL is still workable through a human manual input on a browser [16]. *nag* is in such an instance regarded as an automated generative machine, and once it receives the keywords input from a user, it will go through the *nag* program and server to post a URL request to Google automatically. There is a significant difference, at least from the perspective of Google, between *nag* (a machine) and a user who pastes the exact URL onto a Web browser. Additionally, Google has made a decision that if one requires access to their services through automated means, then this must be done by going through the official API implementation rather than just the URL, which, as stated above, any person could use. This is most likely partly because a machine has the potential to exploit services by a customized program to extract more data and at a much faster rate without being identified. Additionally, it is difficult to monitor, track, control and regulate the corresponding usage, as the same URL can be used by different machines and programs. In relation to such issues, one question that arises is to what extent an API is better than a URL query? How best to retrieve and extract data through APIs? What are the other benefits of implementing an API? What can be exchanged with and through APIs?

Many platform providers implement an API key [17] as part of their Web API(s), which is normally expressed in a list of symbols for the purpose of identification, authentication and authorization between clients and servers. This is treated as a form of ‘token’ on the part of the API design architecture, which has to be submitted alongside the query in order to identify the origin of the requester. An API key provides access to the requested service and is also treated as a unique identifier for tracing corresponding usage, in terms of when and how an API has been used, as well in regards to frequency and volume of API-based requests [18]. The user of an API key is required to provide “accurate” and “up to date” data (Google, 2014) for the API key registration, such as contact details and full name. Control and measures can be exercised down to the individual level through this identifiable key, such as controlling the request volumes and data rate.

Version 5b of *nag* employs the official Google API, called “Custom Search API”, through the implementation of the key/token concept. In order to access the image search service it essentially requires two keys: the API key and the search engine ID. These two parameters are the prerequisites for fetching any images. Each project will be given a different key but with a similar combination of alphanumeric identifiers in the form of a long string. In the act of procuring an API key, an agent is also agreeing to the API’s terms, in this case, “that Google may monitor use of the APIs to ensure quality, improve Google products and services, and verify your compliance with the terms” (Google, 2014). Additionally, Google may also “change, suspend or discontinue all or any aspect of the Service, including their availability, at any time, and may terminate Your use of the Service at any time” (Google, 2018). In other words, Google reserves the right to make any changes at any time. This puts *nag* and anyone else dependent on the API at risk. The “risk territory” involved in such terms and uses of an API arises due to the power of API service providers to not only increase restrictions and impose contractual limitations at any time (Bucher, 2013), but also to potentially terminate or discontinue at any time and without any sufficient period of notice for developers or business corporations to react. As a result, organizations have to constantly “stay alert” (Bucher, 2013).

nag is not a business. It is a project run by artists in a non-profit manner. Unfortunately, *nag* is just one of many other artworks [19] that are dependent on Google’s services. Google has continuously modified its API services, from a policy perspective, to service details and onwards into technical implementations over the past decades. In relation to *nag*, such modifications significantly change the programming practices involved, requiring a large amount of resources towards the maintenance of the availability of its artworks. Such consequences of program updates, system upgrades and documentation revisions are felt by all — programmers, users, networked computational agents, etc. — working with a particular API that has been modified, affecting a range of different practices in potentially multiple and varying industries and cultural sectors. In February 2018, Twitter officially announced a new policy regarding “the use of any form of automation (including scheduling) to post identical or substantially similar content” (Roth, 2018),

specifically targeting those with multiple accounts and especially automated bots. As a result, many bot accounts were blocked, including creative ones for non commercial purposes. It was not until August in the same year that Twitter finally tweeted a statement about their commitment on supporting creative bots and announced their new application process on using their APIs [20]. In other words, a change in format, specification or policy is not a mere technical shift, but additionally, as described by Sterne, one which “may mark a significant cultural shift” [21]. These at times seemingly minor modifications to a key relational computational entity like an API can have a significant impact on the ecology of software practices dependent on such APIs, and point to just how important APIs have become as facilitators of networked computational interactions.

It should be noted that there was no apparent usage limitations with the former Google URL request, at least from the experience of the smooth running of *nag* from 2003 to 2015. Currently, the charging model for the Custom search API is mainly categorised as twofold. The first is the free version, with its hard limit of 100 search queries per day. This is the version used for *nag* version 5b. In contrast to the free version, Google supports additional requests at a cost of US\$5 per 1,000 queries and with a maximum of 10,000 queries per day. This price model is only applied on search engine APIs, while Google offers 2,500 free requests per day for its Maps API service [22] and imposes a different ceiling, with a limit of 100 thousand requests per day as of June 2018. However, Google released a new charging and registration process in July 2018 in relation to their Maps APIs, in which Google no longer sets the usage ceiling because of the availability of bulk purchase discount. Additionally, Google has requested that all accounts add a credit card billing account with a given US\$200 credit for each month, which is roughly the equivalent to 100 thousand free requests per day, but charging will automatically be applied thereafter (see Figure 7 for a summary). One might wonder, how does Google determine the price of these two services, especially the more than 200 fold difference in the amount of free requests allowed? Nevertheless, from these different pricing models, one can conclude that a stricter and higher pricing scheme has been implemented in search services. From a business and marketing perspectives, it could imply that Google’s various search services are more in need. Or one might also surmise that Google is able to capitalise upon its current monopoly in search, as it occupies more than 70 percent of the global market share [23] of leading search engines.

Items	Free requests / day	Ceiling (additional charges applied)
Image Search API	100	10,000
Map static API (before July 2018)	2500	100 thousand
Map static API (after July 2018)	100 thousand *based on the monthly given \$200 credit for each billing account	Unlimited

Figure 7: An overview of different limitations on Google’s API services.

Note: Larger version available [here](#).

However, one of the core differences of the search service and the map service is that the search data that is managed by Google is essentially harvested from the entire public domain, where people, organizations and corporations have uploaded data, which facilitates search operations in the first place. In other words, this specific search engine service relies very much on user-generated content to exist. Importantly, search engines are directly related to everyday knowledge seeking and production (Introna and Nissenbaum, 2000; Lovink, 2011), and accessibility is something that cannot be underestimated. It is not only a provided service *per se*, but may also have to take corporate social responsibility into consideration.

Although Google supports non-profit organisations such as education for what it terms as “high quality” search, this refers only to advertisement removal with the utilisation of Google Custom Search Engine specifically. As such, all units — regardless of whether they are educational, art projects or business services and support for non-profit organisations — are basically charged at the same model when it comes to the use of its APIs. If we consider Google as a giant and listed corporation that harvests worldwide public data for profit, what kinds of principles of social responsibility might Google consider when devising the particular protocological parameters of its APIs? Indeed, Google has worked to project an image as an organisation that supports arts and culture. In 2011, an online platform formerly called Google Art Project [24] was launched as a virtual platform to provide access to exhibitions and digitalised artworks. More recently, Google has supported artistic and technological innovation by collecting different experiments, offering a platform called Experiments with Google [25] to showcase different public contributions as a way to encourage new forms of computationally-facilitated experimentation. On a more technical level, Google also developed initiatives such as its Machine Learning educational site [26] to increase accessibility to various state-of-the-art technologies. In all of these examples, one can see that Google has a specific agenda in participating with a certain brand of corporate social responsibility, but the various limits of Custom Search Engine APIs have also set out certain very specifically determined technical limits and parameters that should also be understood as enactments of values on Google’s part.

Sociomaterial negotiations and their specific forms of networked computational exchange are never produced as even, neutral and symmetrical forms of exchange. As stated clearly in the terms of Google, "If you do not agree to the modified terms of a Service, you should discontinue your use of that Service" (Google, 2017). Such terms and contracts impose significant forms of restriction in their setting of terms and enforcement of these terms via the use of an API keys' registration and monitoring. These restrictions are further implemented via things such as price schemes that impose further restrictions and limits to the amount of free requests and corresponding usage ceilings, particularly when compared with other Google services. Additionally, all non-profit and educational projects would fall under the category of the highly limited free usage terms when unable to pay for greater access to Google's services.

One sees how APIs establish particular forms of exchange, with *nag* using the API service provided by Google, which in return involves limited usage and monitoring of all elements being exchanged via the API. An exchange that may be comparable to the free services provided by other platforms, in particular to social media platforms. Andersen and Pold have described such platforms as a "metainterface industry" [27], where ongoing control of production, distribution and consumption is exercised. Extending from a platform as a whole to the scope of APIs, these platforms inscribe third-party developers into an infrastructure as both producers and consumers (producing other "services" based on the APIs and consuming of the APIs), all housed within "automated control mechanisms" whose parameters and affordances are part of the business model as a whole.

As seen in examples such as the interviews conducted by Bucher with the early developers' community around Twitter, these exchanges are far from being symmetric. One can make a strong argument that third-party developers often bring a platform's service into a more imaginative and engaging light, and in doing so often enhance the ecology of a platform's success (Bucher, 2013). But such added value is rarely rewarded by API providers or treated as an incentive to create more open APIs. Indeed, any such openness often tends to eventually be shut down once a platform has achieved relative dominance, as Twitter has done with many of its APIs. Helmond describes implementations and pivots such as these as a "double logic of platformization", a two-way process whereby a decentralising platform feature, such as an API, is often crafted with the specific aim of using its features to make data or other forms of value external to the platform capturable according to platform-native objects and techniques of the platform itself, thus resulting in an expansion of the platform's dominance, and what, in the larger picture, results in a recentralization of the platform's computational and economic models for exchange. A paradigm for protocological control over both the gathering of data collection and the eventual managing of the terms of access to such harvested data that can be seen in examples such as that of Google, Facebook and other dominant networked platforms of today.

In this section, we have examined Google's search APIs through the artwork *nag*, specifically focusing on the registration process, terms and conditions, and also the charging schemes and limitations in order to identify what we consider to be asymmetric forms of protocological exchange. In the next section, we will continue to look at this, but with a focus on technical parameters, so as to continue to establish a notion of protocological parameters in greater detail.

Technical parameters of control and exchange

This section takes a closer look at actual computational query processes through an examination of the Google Custom Search Engine API that is implemented in *nag*, from the protocol of 'getting' and 'requesting' image data to the specific parameters of setting the data range, as well as to the received data format. These processes are crucial in understanding how Web APIs work to fetch data at the informational and computational level, but are also an important element for showing how asymmetric vectors of exchange can be opened up and exercised at both a technical and also regulatory level.

```

1 #---search criteria---
2 my $api = "https://www.googleapis.com/customsearch/v1";
3 my $key = 'YOURAPP_ID';
4 my $cx = 'YOUR_SEARCHENGINE_ID';
5 my $searchType = 'Image';
6 my $query = shift;
7 my $fileType = shift;
8 my $imgSize = 'xlarge';
9 my $safe = 'high';
10 my $url = $api . "key=" . $key . "&cx=" . $cx . "&q=" . $query . "&searchType=" . $searchType . "&fileType=" . $fileType . "&imgSize=" . $imgSize . "&safer=" . $safe;
11 #---end of search criteria---
12 my $ua = LWP::UserAgent->new;
13 my $request = HTTP::Request->new("GET" => $url);
14 my $response = $ua->request($request);
15 my @links = ();
16 my $message = $response->decoded_content;
17 my $returned_query = decode_json($message);
18
19 if ($response->is_success) {
20   my $extract1 = $returned_query->{items};
21   for my $pick (@$extract1) {
22     push(@links, "$pick->{link}");
23   }
24 } else { #unsuccessful query with error code checking
25   my $extract2 = $returned_query->{error}->{errors}->[0]->{reason};
26   my $extract3 = $returned_query->{error}->{code};
27   push(@links, $extract2) if ($extract2 eq "DailyLimitExceeded" && ($extract3 == 403)); #DailyLimitExceeded and error code 403 - msg by Google
28 }
29 return @links; #@links will be either blank (other kinds of unknown error), or with links (valid query), or error msg by Google
30 }
31 #---end of new Google API parsing---
32

```

Figure 8: A snippet of *nag* (ver. 5b) source code relating to search criteria and parsing.

Note: Larger version available [here](#).

[Figure 8](#) shows a snippet of *nag* code related to the core part of structuring a query. During the rise of networked computing, APIs could be used for the task of making available secure access to databases of various kinds. Developments in this area included the design and use of SQL (structured query language) as a domain-specific language for handling structured data (such as data held in a relational database). Innovations like SQL enable a flexibility in manipulating records of massive and structured data via specific commands. An API is considered as a query form for extracting specifically required data within many databases and tabular records, providing greater flexibility in access to data and allowing developers greater freedom in specifying a data request based more on the particular needs rather than those identified by the database itself.

A query is technically understood as a two-way communication process, both a request and a response. This is what an API does in relation to querying data, including selecting, extracting and presenting data through the logic of request and response. In relation to Web APIs, a query is executed through the hypertext transfer protocol (HTTP), the foundation of the World Wide Web for facilitating forms of exchange. Such facilitations on the part of APIs might involve the sharing and making available of information for various productive purposes, what has become one of the most common and impactful uses of APIs within the so-called big data economies of the present. Information sharing of this kind helps in the facilitating of the generative “remixability” of data that one sees today, as well as providing a key support layer for the kind of data “flows” that can occur within and across the different platform environments of the present (O’Reilly, 2005; Helmond, 2015).

A secure version of HTTP refers to HTTPS (hypertext transfer protocol secure), where the communication process is encrypted. This HTTPS protocol allows a secure exchange of data in the form of request and response communication. One of the examples of request-response would be *nag* (a client) requesting certain image data ([Figure 8](#), line 13) from Google (the server) in the form of a query URL with a combination of parameters ([Figure 8](#), line 10, with further explanation below), and the response could be data of the images or an error code ‘403’ whenever the daily free limit of 100 requests is exceeded. Such exchange of data and information between two parties is communicated through a URL over the HTTP(S) request-response protocol on a network layer.

HTTPS as a protocol outlines specific technical standards [28], defining how data is to be transmitted and exchanged. Google APIs run on HTTPS, and specify the communication at the level of code using the ‘GET’ ([Figure 8](#), line 13) method of data transfer. The GET method adds the query data (the requested parameters) to a URL string ([Figure 8](#), line 10), and is mainly used for viewing but not for updating. In other words, a GET request is used for the reading of data, while other methods such as ‘POST’ or ‘DELETE’ can be invoked for writing and updating data (Fielding and Reschke, 2014). The GET method is primarily used for requesting data from a specified source and it performs a *read-only operation*, which is also one of the parameters set by Google’s custom search API. By examining this method at the protocol level, one can discern what kind of data exchange is allowed within a Web API at the technical level. At this level, the exchange is based primarily on a “logic of request && response” [29]. But there is more to say about the capacity to specify the ranges and parameters of data to be processed and read, and how to make sense of not only the capacities, but also the limitations and implications to specify and identify data relations through this logic of request and response.

Within the example of *nag*, the main query parameter is the keyword(s) that a user is searching for, indicated in line 6 of [Figure 8](#). In this example, the keywords involved are the concatenated words ‘What’ ‘is’ ‘APIs’. This is the query parameter that is customized by a user freely on the *nag* Web interface (see [Figure 3](#)). Within an unthinkable, or unimaginable, amount of images in Google’s data banks, these concatenated words become a command that specifies and selects what data is needed at the most basic level. Given the fact that databases are volatile and are constantly being updated, the keyword query creates a new spatial and temporal relation for the selected images at a specific moment, presented in the form of a structured response. But of course, it is too simplistic to think of the keywords as the only parameter that defines what data should be extracted. There are other processes set by Google to narrow down from giant databases to just a few data records, such as other metadata embedded in an image, as well as computing the relevance, ranking, popularity, user engagement, and many other factors, priorities and bias, all of which inform and going into the composition of VisualRank algorithm, itself based on Google’s famous PageRank algorithm (Page, et al., 1998; Introna and Nissenbaum, 2000; Jing and Baluja, 2008). According to Jing and Baluja, one of the goals of image search is to ensure a selection that is “relevant to the query” and “diverse enough to cover variations of visual or semantic concepts” [30]. In other words, multiple forms of computational analysis, as well as other factors, are inevitably at play when processing a query request.

In Google’s Web API search, the free version can return up to 10 records per request that are based on the entire databases. Within mathematics, set theory is used to study the collection of data as a set, and how data comes into relation through operations like intersection, union and difference. In a similar fashion, the 10 records of the data result all have some common properties that are defined as a set. Mathematical operators are crucial in playing the role of data selection at the code level. Taking the example of a search for ‘What is APIs’, the implementation of this is to join the three words by using the ‘+’ sign operator, and the operation is called the intersection of sets to make sense of the keywords as a sentence, where the result is considered as the intersection of the three sets. As the result indicates in [Figure 3](#), one of the images returned with the exact phrase ‘WHAT IS APIS’ on the navigation bar. In this way, the VisualRank algorithm needs to make sense of the three words together, but not in separation.

Additionally, extracting the top 10 records from a search engine requires several more parameters other than keywords. Jing and Baluja reveal that the indexing of image ranking is also about the ‘scores’ among images which determine their visibility [31], as well as other commercial and advertising considerations. But what we want to partly emphasise here is that such mathematical operators and operations play a vital role in the inclusion and exclusion of data, setting the range of needed data and identifying what data should be grouped together and ranked differently. The production of sets involves not only technical

parameters, but also cultural, economic and political ones, the imbrications of which have specific forms of consequences. As Mackenzie writes, "Sets make powerful calls to order, and act as world-making forms of collection and grouping" [32]. Such complex logics of how the collection and grouping of data are established and presented are the fundamental part that constitutes how search algorithms work, and, consequently, how images are being selected and designated as being in the top ten of outputted results.

From a developer side, there are other parameters, beyond keywords, defined in the requested API to specify the data that is needed. While Figure 8 shows the possible configured parameters of Google image search, lines 2–10 from Figure 8 shows the parameters' breakdown in *nag*. Other than the API key/id, search engine id (Figure 8, lines 3–4) and keywords search, there are other parameters [33], including search type (Figure 8, line 5), file type (line 7), file size (line 8) and safety level (line 9). Since *nag* versions 5 and 5b are mainly operated through the manipulation of networked images, the value of 'search type' is set to 'image'. The *nag* Web site only supports images of the widely used Internet formats of JPEG and PNG, where this parameter is a variable and is subjected to the user's selection (see the 'extension' dropdown menu in Figure 2). In the API query, one can also specify the image size, where the parameter is called 'imgSize' with values of huge, icon, large, medium, small xlarge and xlarge, while the parameter 'safe' stands for the search safety level with values of high, medium and off that correspond to a highest, moderate or disabled level of filtering rule, as established by Google. Safe search detection [34] is a function implemented by Google to automatically filter pornography or potentially offensive content, and this is also a feature which an ordinary user can turn on via the Google Web search interface. In terms of how Google decides the safe search, machine learning models are used for such image classification. One of the examples given by Google is what they call "The adult content classifier", which trains sample data in order to classify/separate pornographic and non-pornographic images. This classifier is often smart enough to decide and allow nudity in the context of medical, scientific, educational or artistic context (Robinson, 2016). In this way, such machine learning models are able to analyze the context of an image and make decisions automatically, and even on the fly in real-time as data is being requested. As such, submitting an API request does not only mean to make a query, but also to put in motion other forms of (automated) decisions which themselves have to be processed before the act of extracting selected data from databases.

```
https://www.googleapis.com/customsearch/v1?q={searchTerms}&num={count?}
&start={startIndex?}&lr={language?}&safe={safe?}&cx={cx?}&sort={sort?}&filter={filter?}
&gl={gl?}&cr={cr?}&googlehost={googleHost?}&c2coff={disableCnTwTranslation?}
&hq={hq?}&hl={hl?}&siteSearch={siteSearch?}&siteSearchFilter={siteSearchFilter?}
&exactTerms={exactTerms?}&excludeTerms={excludeTerms?}&linkSite={linkSite?}
&orTerms={orTerms?}&relatedSite={relatedSite?}&dateRestrict={dateRestrict?}
&lowRange={lowRange?}&highRange={highRange?}&searchType={searchType}
&fileType={fileType?}&rights={rights?}&imgSize={imgSize?}&imgType={imgType?}
&imgColorType={imgColorType?}&imgDominantColor={imgDominantColor?}&alt=json
```

Figure 9: The official Google search API with various possible fields.

Note: Larger version available [here](#).

The final part of the query processing in this example is the parsing of the result data from the API. The search result data is encapsulated in the Javascript object notation (JSON) format, which has become one of the most popular formats used for data-interchange between applications. JSON is designed for interoperability, acting as a standardised format used to exchange data between applications written in different programming languages [35]. Such standardised formats are widely used in programming, allowing millions of developers in the world working on different programming languages and platforms to retrieve data, process queries and interpret information. At the most basic level, JSON is formatted in the style of "name:value" pairs (see Figure 10) and each set of data is contained within the symbols of curly bracket, following a particular "JSON Grammar" [36]. Using defined semantic symbols, such as name, colon and curly brackets, one can write a program to automate the parsing of the specific data value (e.g., the image URL titled 'link' in Figure 8) precisely provided so that the same structure is responded upon every request. In other words, using a standardized format that can be easily parsed enables wider distribution, circulation and application of data. Specifically, in the example of a Google API exchange, this may mean that more data can be reached on the part of an end user's side, even outside of the Google interface itself. As a result, services can be provided by third-parties like *nag*, resulting in more applications being dependent on Google services. Inversely speaking, data can also be captured through a queries' request as part of the contribution to the VisualRank or PageRank algorithm. In all of this, Google's technical, economic and cultural influence can be seen to have far-reaching effects in a number of contexts, many of which can be seen to be concretized in the seemingly simple artefact of this API and the practices within and around it.

```
{
  "kind": "customsearch#result",
  "title": "What Is API Virtualization | SmartBear",
  "htmlTitle": "\u003cb\u003eWhat Is API\u003c/b\u003e Virtualization | SmartBear",
  "link": "https://smartbear.com/SmartBear/media/images/Ebook%20IMG/API-Mocking.jpg",
  "displayLink": "smartbear.com",
  "snippet": "What Is API Virtualization | SmartBear",
  "htmlSnippet": "\u003cb\u003eWhat Is API\u003c/b\u003e Virtualization | SmartBear",
  "mime": "image/jpeg",
  "image": {
    "contextLink": "https://smartbear.com/learn/api-testing/what-is-api-virtualization/",
    "height": 1165,
    "width": 892,
    "byteSize": 68912,
    "thumbnailLink": "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRC6H5_P9Yk_TII763ZI5Kegw-Z1-oDdU6I3gOLi51biBSQmap9LMOyH1_A",
    "thumbnailHeight": 150,
    "thumbnailWidth": 115
  }
}
```

Figure 10: A sample result of an unformatted JSON from Google search.

Note: Larger version available [here](#).

This section has walked through a specific and selected query process (however the processes involved here are very common within Web API practices), revealing protocological forms of control via technical parameters and operational processes, including network protocols of HTTP(S), the query's request and response logic, some of the fundamental elements in SET operations, as well as standardization of data file exchange formats. What we want to highlight here is that these discussions are not simply of a technical nature. Rather, these technical questions and their often shifting specifications and parameters of possibility point to how processes of control and exchange are enacted through the execution of technical parameters with both immediate and longer term effects. We argue that technical parameters should be understood as protocological parameters that exhibit particular forms of regulatory control and data exchange, parameters of control that are worked out across the technical, the economic, and further into what become the wider parameters for social and cultural possibilities of exchange.

Moving forward: Key themes and questions in API practice

This paper has traced some of the history of API design, development and practice in order to understand how an API can be understood as a connector of components, interface, abstract interface and networked protocological interface for facilitating forms of exchange and interoperability. We have further discussed how APIs are used as building blocks for key computational processes, such as in the distributing and sharing of data and computing power across components, systems, platforms and networked environments. The aim throughout has been to highlight some key issues and paradigms within API practices, both in order to better understand APIs as important infrastructural elements within computing practices, and also to identify some key areas for consideration in regards to API development going forward.

Among various contemporary API practices, the establishing of protocological parameters such as those studied here point to the particular forms of relations and power that can be built off of and crafted via protocols. In recent times, dominant modes of protocological relations in networked exchanges have often resulted in a centralization of power on the part of certain API providers. This can be witnessed, for instance, in the example of Facebook's Social Plug-in APIs and the way in which their decentralization of data mining (via something like the embedding of a Like or Share button on an external Web site) is carried out with a more overarching centralizing aim of using these APIs to make data external to the platform capturable according to platform-native objects and techniques of the platform itself. An implementation of APIs that results in a clear expansion of what, in the larger picture, is a general centralization and solidification of a platform's power and reach over sociotechnical modes of cultural exchange (Gerlitz and Helmond, 2013; Helmond, 2015; Snodgrass, 2017). Such methods and implementations have the effect of leading to significant potential situations of what Tufekci (2014) describes as "information asymmetry" between these dominant API providers and their users, as well as other would be services operating in their domain. In what is a growing API-facilitated ecology of networked interactions (Espinha, *et al.*, 2015; Verborgh, 2018), it is important to take note, as Gerlitz, *et al.* (2016) do, of the way in which Web sites and now apps are typically no longer self-sufficient, standalone, "single medium" style software interfaces, but rather "are embedded in, built on top of or in relation to" certain dominant platform providers and their centralized databases and other related services. The API, and the Web API in particular at present, becomes one assembling method for rearticulating and programming on top of the affordances of HTTP and the WWW. We argue here that APIs as abstract interfaces and connective enablers are one such key technical facilitator and entrance point towards studying these emerging knotted aggregations of interfaces, applications and platforms that we see developing via the articulating and rearticulating of a range of protocols of different kinds.

In an age in which much power has been centralized and housed within a handful of dominant platforms, APIs at the same time provide one of the few connectors and gateways to the opaque insides of these platforms. In their always co-constituted act of closing off access to certain components of a black box while also exposing and making addressable specific internal components for exchange, APIs “construct modes of transparency” in which one can trace what Fuller (2015) describes as a “fundamental architectural propensity to negotiating the relationship between transparency, openness and integration of systems, always though interpellated by the question of what data is closed, what is open, what is made accessible, what is kept internal to a system.” In this way, APIs are also important for the way in which they can give some sense of how data is being circulated, made accessible and inaccessible. By studying and paying attention to the makeup of their structure and parameters one can also begin to detect the priorities, and thus politics, of the platforms in question.

In tracing and articulating such relationships of power and exchange in what is a rather closed nature of many of today’s dominant and proprietary systems, we adopt a regulatory and technical perspective for analysing the specific artefact of *nag*. An approach that scrutinizes concrete and further details, not only of how certain APIs have operated in the past decade in business and technical terms, but also, more importantly, laying out the sociotechnical issues and power relations regarding the forms of (big) data exchange in contemporary conditions. APIs as a genre of software development, or more specifically networked computational exchange, can seem relatively minor when compared with the bright stars of areas like big data, social networks, machine learning and the Internet of Things. And yet, such protocological objects, as we have aimed to demonstrate, act in a controlled manner, and in their designating of certain parameters of access have the immediate effect of governing the relations within such exchanges. In particular, the many daily encountered networked objects and interfaces that are built upon API services, objects whose actual processes of creation and negotiated exchange are often carried out very much in the background.

Sociomaterial negotiations and their specific forms of networked computational exchange are never produced as even, neutral and symmetrical forms of exchange. The technical parameters at play in APIs should be understood as protocological parameters that exhibit particular forms of regulatory control; parameters of control that are worked out across technical, economic and also wider parameters and possibilities for social and cultural exchange. Building upon established scholarship in the field of software studies (Chun, 2011; Frabetti, 2014; Fuller, 2017, 2003; Mackenzie, 2006) as well other related fields (e.g., computer science, artistic practice and research, digital methods), we believe the kind of technical and cultural approach carried out here can provide further insights as well as a useful method for bringing to the fore issues that are often not readily apparent and visibly registered, as witnessed in the often overlooked but key infrastructural and sociotechnical practice of API development and deployment.

Moving forward, it is important to take seriously the way in which, as Bucher (2013) highlights, “APIs are anticipatory in their very operational logic.” This involves a consideration and study of both direct implementations of API interfaces of various kinds, just as it also requires a step back and further looking into so-called higher-level considerations of the underlying operational logics at play in an API’s position as interface and abstract interface, application and infrastructure, system and ecosystem. Current practices aimed at creating decentralized API ecosystems represent one such attempt to anticipate the problematics and points for working on in order to realize such a decentralized API world (Verborgh, 2018, 2017). Regardless of the details of any overall paradigm at play in API deployment, we argue that an API provider, and also developers using a particular API, should always work to consider the forms of programmatic accessibility, applicability, interoperability, transferability and durability that are provided by API-facilitated exchanges. Preferably in a manner that is open to input from those using an API. Hence, in the instance of *nag*, we make the suggestion that API providers consider giving more generous terms of access to non-profit, educational research and art-focused projects. Or in similar examples, such as Rieder’s (2016) discussion of the closing down of APIs that have proven important tools for research methods and/or as forms of scrutinizing large scale platforms, it can be important to consider how “we need a legal framework to guarantee at least some access to API data, at least for some people.” Doing so in order to keep APIs active, evolvable and accessible as forms of exchange and also interrogation across a range of possible practices.

APIs are infrastructural just as they also impose structures and parameters of various kinds, both on the data being exchanged, as well as those who use or come into contact with these exchanges. During the last two formative decades of API development, Amazon has made for one of the most interesting and instructive examples of API development, both given its long history of using APIs and for the way in which its current cloud services point towards the powerful infrastructural nature and potential of APIs — what has been labeled as the “Infrastructure as a Service” approach of on-demand cloud services. As discussed in a mistakenly publicised insider rant comparing Google’s and Amazon’s differing approaches to platform development, then Google employee and ex-Amazon programmer Steve Yegge (2011) discusses how Amazon, when he was working with them in the mid-2000s, was able to impose a corporate culture of always working with extensible, service interface-oriented approaches throughout the company. A corporate culture and approach that, over time, made it well-positioned as a company full of platform-ready approaches and products. In addition to discussing key issues of service interface and API approaches (such as the inevitable tensions that arise between the needs of accessibility vs. those of security), Yegge highlights how “The Golden Rule of Platforms” is to “Start with a Platform, and Then Use it for Everything”. Or, as he also puts it, “Eat Your Own Dogfood” (Yegge, 2011). In a platformized world, it is not just the product itself that is key, but the careful thinking through and implementing of its accessibility and extensibility for both existing but also future possible products and platforms. This is done specifically via the use of APIs and other platform-oriented approaches that make a product — and by extension, a company creating such products — platform ready.

In the example of Amazon — but also as seen in the example of Unix examined earlier — of a computational structure such as an API acting as both a product and “philosophy”, one sees how the technical quality of APIs as instantiating and governing particular forms of access and exchange can have greater implications as to the parameters and paradigms that present themselves as most likely and available to the parties working with and within them. The perceiving and setting of potential terms,

definitions and parameters of interoperable exchange is programmatic and composed of specifically crafted technical affordances, just as it is also social and material when successfully able to gain purchase amongst a gathering range of agents and actors. With the sedimenting of norms or paradigms around API-informed practices and approaches, it becomes possible to trace out certain kinds of actual and potential philosophies and cultures of code and computation, each with their own approaches, “best practices” and implications for those working within and around them. With the question perhaps arising, what kind of dog food is it exactly that we are eating?

About the authors

Eric Snodgrass is a senior lecturer at the Department of Design+Change, Linnaeus University, Sweden. His research looks into the intersections of computation, culture, politics and technology, with a current focus on computational infrastructures and politically-oriented forms of intervention. His recent work includes the co-edited volume *Executing practices (Data browser 06)* and a Ph.D. dissertation, “Executions: Power and expression in networked and computational media.”

Web: <https://lnu.se/en/staff/eric.snodgrass/>

Twitter: @snoddy

E-mail: eric [dot] snodgrass [at] lnu [dot] se

Winnie Soon (HK/DK) is an artist-researcher, examining the cultural implications of technologies in which computational processes and infrastructure underwrite our experiences, which are ever more programmed. Her works explore themes/concepts around computational culture, specifically concerning Internet censorship, data circulation, real-time processing/liveness and the culture of code practice. Winnie’s projects have been exhibited and presented internationally at museums, festivals, universities and conferences across Europe, Asia and America. Her current research focuses on aesthetic coding in the realm of software studies, working on two books titled *Aesthetic programming: A handbook of software studies* (with Geoff Cox) and *Fix my code* (with Cornelia Sollfrank). She is assistant professor at Aarhus University (Denmark).

Web: <http://www.siusoon.net>

Twitter: @siusoon

E-mail: wsoon [at] cc [dot] au [dot] dk

Notes

1. Helmond, 2015, pp. 6–7.

2. Shaw and Clements, 1997, p. 3.

3. Parnas, 1977, p. 6.

4. See McPherson (2012) for a critique of the “non-benevolent” potentials that the “divide and conquer” method of modularity potentially opens onto.

5. Parnas, 1972, p. 1.

6. Parnas, 1972, pp. 2–3.

7. It is worth noting in this brief history of APIs that the names given to different forms of encapsulation have varied from “modules” to “components” to “objects”, “resources”, “messages” and onwards, but that in each case what is important in relation to an API is the designation and making accessible of bounded computational units upon which defined programmatic work of some kind can be applied.

8. See Amundsen (2011) for a brief overview video on these three common architectural styles: <https://www.youtube.com/watch?v=GEZagRDLhTA>.

9. See, for instance, <http://apistylebook.com> for a collection of such documentation.

10. Mackenzie, 2018, p. 10.

11. Mackenzie, 2018, p. 2.

12. Mackenzie, 2018, p. 6.

13. Ver 01: Ryan Johnston. Ver 02: Luka Frelih. Ver 03: Barbara Thoens and Ralf Prehn. Ver 04: Richard Leopold. Ver 05: Panos Galanis. Ver 05b: Winnie Soon.

14. One can paste this line of query on a Web browser and it will return a grid of images as the search result.

15. To address specifically this 100 requests limitation, *nag_extensions* (2018) is developed and exhibited to showcase all the limited images that are generated within a particular period of time in a gallery setting. More detail at <https://www.netzkunst.berlin/>.

16. Indeed, using a URL to fetch Web data in an automated generative machine has been a fairly common method since the early days of the Web. Other Google services like Google Maps and other data platform providers like OpenWeatherMap (see <https://openweathermap.org/>) also allow similar acts of posting a URL query to extract data.

17. There are also other authentication methods, such as OAuth access tokens and JSON Web tokens, but this article will focus primarily on basic API keys as this is what *nag* has used.
18. Statistics and logs are provided as part of the Google Custom Search Console, and can be broken down by day, week, month and year. Additionally, the console also provides the count of popular query terms.
19. See <http://www.250miles.net/techno-mourning/> for a listing of several important art projects that were affected by a change to Google Earth API.
20. See the documentation of the process at <https://slate.com/technology/2018/08/twitters-new-developer-guidelines-might-end-fun-bot-accounts.html>.
21. Sterne, 2012, p. 12.
22. We have specifically looked into Maps static API as an example (as of 26 September 2018). See <https://developers.google.com/maps/documentation/maps-static/usage-and-billing>.
23. As of 21 June 2018. See <https://www.statista.com/statistics/216573/worldwide-market-share-of-search-engines/> for latest numbers on worldwide desktop market share of leading search engines.
24. See Google Art & Culture at <https://artsandculture.google.com/>.
25. See Experiments with Google at <https://experiments.withgoogle.com/>.
26. See Learn with Google AI at <https://ai.google/education/>.
27. Andersen and Pold, 2018, p. 51.
28. Galloway, 2004, p. 6.
29. Pritchard, *et al.*, 2018, at http://www.data-browser.net/pdf/DB06_Executing_Practices.pdf.
30. Jing and Baluja, 2008, p. 310.
31. Jing and Baluja, 2008, p. 309.
32. Mackenzie, 2012, p. 221.
33. See the full CSE: list at <https://developers.google.com/custom-search/json-api/v1/reference/cse/list>.
34. See the announcement of the SafeSearch feature for Web searching at <https://googleblog.blogspot.com/2009/11/locking-safesearch.html>.
35. Crockford, 2006, p. 7.
36. Crockford, 2006, p. 2.

References

- Mike Amundsen, 2017. "Twelve patterns for hypermedia microservices" (4 November), at <https://www.infoq.com/presentations/12-patterns-api-microservices>, accessed 17 May 2018.
- Mike Amundsen, 2011. "Three common Web architecture styles," at <https://www.youtube.com/watch?v=GEZagRDhTA>, accessed 15 May 2018.
- Christian Ulrik Andersen and Søren Bro Pold, 2018. *The metainterface: The art of platforms, cities, and clouds*. Cambridge, Mass.: MIT Press.
- Taina Bucher, 2013. "Objects of intense feeling: The case of the Twitter API," *Computational Culture*, number 3 (16 November), and at <http://computationalculture.net/objects-of-intense-feeling-the-case-of-the-twitter-api/>, accessed 1 May 2018.
- Taina Bucher and Katie Rose Pipkin, 2015. "About a bot: Interview with Katie Rose Pipkin," *Furtherfield* (7 December), at <https://www.furtherfield.org/about-a-bot-interview-with-katie-rose-pipkin/>, accessed at 1 June 2018.
- Wendy Hui Kyong Chun, 2011. *Programmed visions: Software and memory*. Cambridge, Mass.: MIT Press.
- Douglas Crockford, 2006. "JSON" (10 June), at <https://tools.ietf.org/html/draft-crockford-jsonorg-json-00>, accessed 10 August 2018.
- Ole-Johan Dahl, 2004. "The birth of object orientation: The simula languages," In: Olaf Owe, Stein Krogdahl and Tom Lyche (editors). *From object-orientation to formal methods: Essays in memory of Ole-Johan Dahl. Lecture Notes in Computer Science*, volume 2635. Berlin: Springer, pp. 15–25. doi: https://doi.org/10.1007/978-3-642-30779-9_30, accessed 10 January 2019.
- Tiago Espinha, Andy Zaidman and Hans-Gerhard Gross, 2015. "Web API growing pains: Loosely coupled yet strongly tied," *Journal of Systems and Software*, volume 100, pp. 27–43. doi: <https://doi.org/10.1016/j.jss.2014.10.014>, accessed 10 January 2019.

- Roy Fielding, 2000. "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, at <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>, accessed 1 May 2018.
- Roy Fielding and Julian Reschke (editors), 2014. "Hypertext transfer protocol (HTTP/1.1): Message syntax and routing," *Internet Engineering Task Force, Request for Comments*, number 7230, at <https://www.rfc-editor.org/info/rfc7230>, accessed 15 May 2018.
doi: <https://doi.org/10.17487/RFC7230>, accessed 10 January 2019.
- Federica Frabetti, 2014. *Software theory: A cultural and philosophical study*. Plymouth, U.K.: Rowan & Littlefield International.
- Matthew Fuller, 2017. "Software studies methods," In: Matthew Fuller. *How to be a geek: Essays on the culture of software*. Cambridge: Polity Press, pp. 55–68.
- Matthew Fuller, 2015. "Black sites & transparency layers," video of presentation at the #Neoliberalism conference in Goldsmiths University (9 July), at <https://www.youtube.com/watch?v=U2R91sn04NQ>, accessed 15 May 2018.
- Matthew Fuller, 2003. *Behind the blip: Essays on the culture of software*. New York: Autonomedia.
- Alexander Galloway, 2004. *Protocol: How control exists after decentralization*. Cambridge, Mass.: MIT Press.
- Carolin Gerlitz and Anne Helmond, 2013. "The like economy: Social buttons and the data-intensive Web," *New Media & Society*, volume 15, number 8, pp. 1,348–1,365.
doi: <https://doi.org/10.1177/1461444812472322>, accessed 10 January 2019.
- Carolin Gerlitz, Fernando N. van der Vlist, Anne Helmond and Esther Weltevredé, 2016. "App support ecologies: An empirical investigation of app–platform relations," *Infrastructures of Publics — Publics of Infrastructures, First Annual Conference 2016 of the DFG Collaborative Research Centre 1187 'Media of Cooperation'* (Artur-Woll-Haus, University of Siegen, Germany), at <http://appstudies.org/2017/04/05/app-support-ecologies-an-empirical-investigation-of-app-platform-relations/>, accessed 10 January 2019.
- Google, 2018. "Getting started with custom search: CSE terms of service," at <https://support.google.com/customsearch/answer/1714300>, accessed 10 August 2018.
- Google, 2014. "Google APIs terms of service" (5 December), at <https://developers.google.com/terms/>, accessed 10 August 2018.
- Google, 2007. "Google terms of service" (16 April), at <https://www.google.com/intl/en/policies/terms/archive/20070416/>, accessed 10 August 2018.
- Anne Helmond, 2015. "The Web as platform: Data flows in social media," Ph.D. thesis, Amsterdam School for Cultural Analysis (ASCA), at <http://hdl.handle.net/11245/1.485895>, accessed 10 January 2019.
- Lucas D. Introna and Helen Nissenbaum, 2000. "Shaping the Web: Why the politics of search engines matters," *Information Society*, volume 16, number 3, pp. 169–185.
doi: <https://doi.org/10.1080/01972240050133634>, accessed 10 January 2019.
- Yushi Jing and Shumeet Baluja, 2008. "PageRank for product image search," *WWW '08: Proceedings of the 17th International Conference on World Wide Web*, pp. 307–316.
doi: <https://doi.org/10.1145/1367497.1367540>, accessed 10 January 2019.
- Brian Kernighan and Rob Pike, 1984. *The UNIX programming environment*. Englewood Cliffs, N.J.: Prentice-Hall.
- Ganaele Langlois, Fenwick McKelvey, Greg Elmer and Kenneth Werbin, 2009. "FCJ-095 Mapping commercial Web 2.0 worlds: Towards a new critical ontogenesis," *Fibreculture Journal*, number 14, at <http://fourteen.fibreculturejournal.org/fcj-095-mapping-commercial-web-2-0-worlds-towards-a-new-critical-ontogenesis/>, accessed 16 May 2018.
- Geert Lovink, 2011. "Society of the query: The Googlization of our lives," In: Geert Lovink. *Networks without a cause: A critique of social media*. Cambridge: Polity Press, pp.146–157.
- Adrian Mackenzie, 2018. "From API to AI: Platforms and their opacities," *Information, Communication & Society* (11 June).
doi: <https://doi.org/10.1080/1369118X.2018.1476569>, accessed 10 January 2019.
- Adrian Mackenzie, 2012. "Set," In: Celia Lury and Nina Wakeford (editors). *Inventive methods: The happening of the social*. Abington: Routledge, pp. 219–231.
- Adrian Mackenzie, 2006. *Cutting code: Software and sociality*. New York: Peter Lang.
- Ole Lehrman Madsen, Birger Møller-Pedersen and Kristen Nygaard, 1993. *Object-oriented programming in the BETA programming language*. Reading, Mass.: Addison-Wesley.
- Tara McPherson, 2012. "U.S. operating systems at mid-century: The intertwining of race and UNIX," In: Lisa Nakamura and Peter A. Chow-White (editors). *Race after the Internet*. New York: Routledge, pp. 21–37.

Tim O'Reilly, 2005. "What is Web 2.0: Design patterns and business models for the next generation of software" (30 September), at <https://www.oreilly.com/pub/a/web2/archive/what-is-web-2.0.html>, accessed 20 May 2018.

Larry Page, Sergey Brin, Rajeev Motwani and Terry Winograd, 1998. "The PageRank citation ranking: Bringing order to the Web," *Proceedings of the Seventh International World Wide Web Conference*, pp. 161–172; version at <http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf>, accessed 10 January 2019.

David L. Parnas, 1977. "Use of abstract interfaces in the development of software for embedded computer systems," *U.S. Naval Research Laboratory (NRL), report 8047* (June 3), at <https://apps.dtic.mil/docs/citations/ADA043369>, accessed 10 January 2019.

David L. Parnas, 1972. "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, volume 15, number 12, pp. 1,053–1,058. doi: <https://doi.org/10.1145/361598.361623>, accessed 10 January 2019.

Helen Pritchard, Eric Snodgrass and Magda Tyżlik-Carver (editors), 2018. *Executing uncertainties*. Open Humanities Press, at http://www.data-browser.net/pdf/DB06_Executing_Practices.pdf, accessed at 10 January 2019.

Bernhard Rieder, 2016. "Closing APIs and the public scrutiny of very large online platforms," *Politics of Systems* (27 May), at <http://thepoliticsofsystems.net/2016/05/closing-apis-and-the-public-scrutiny-of-very-large-online-platforms/>, accessed 23 August 2018.

Sara Robinson, 2016. "Filtering inappropriate content with the Cloud Vision API," *Google Cloud Platform* (17 August), at <https://cloud.google.com/blog/products/gcp/filtering-inappropriate-content-with-the-cloud-vision-api>, accessed 12 August 2018.

Yoel Roth, 2018. "Automation and the use of multiple accounts," *Twitter Developer Blog* (21 February), at https://blog.twitter.com/developer/en_us/topics/tips/2018/automation-and-the-use-of-multiple-accounts.html, accessed 23 August 2018.

Mary Shaw and Paul Clements, 1997. "A field guide to boxology: Preliminary classification of architectural styles for software systems," *Proceedings Twenty-First Annual International Computer Software and Applications Conference (COMPSAC '97)*, pp. 6–13. doi: <https://doi.org/10.1109/COMPSAC.1997.624691>, accessed 10 January 2019.

Eric Snodgrass, 2017. "Engineering relations: APIs and the structuring of networked flows and connections," In: Eric Snodgrass. "Executions: Power and expression in networked and computational media," Ph.D. dissertation, Malmö University, at <http://muep.mau.se/handle/2043/22834>, accessed 10 January 2019. doi: <https://doi.org/10.24834/2043/22834>, accessed 10 January 2019.

Cornelia Sollfrank, Winnie Soon, Morgane Stricot and Matthieu Vlaminck, 2017. "Podium discussion: net.art generator/generating discourse" (13 December), at <https://zkm.de/en/event/2017/12/podium-discussion-netart-generator-generating-discourse>, accessed 10 January 2019.

Winnie Soon, 2018. "Executing queries as a form of artistic practice," In: Madeline Eschenburg, Ellen Larson, Wenlong Wang and Zhuxin Wang (editors). *Overwhelming imagination: Achieving and undermining contradictions*. Beijing, China: Si Shang Art Museum.

Winnie Soon, 2016. "Executing liveness: An examination of the live dimension of code inter-actions in software (art) practice," Ph.D. dissertation, School of Communication and Culture, Aarhus University, and at http://siusoon.net/home/me/doc/soon_PhD_FINAL.pdf, accessed 10 January 2019.

Winnie Soon, 2011. "The public interface as an art-making enabler," *Parsons Journal for Information Mapping*, volume 3, number 4, pp. 1–7, and at http://piim.newschool.edu/journal/issues/2011/04/pdfs/ParsonsJournalForInformationMapping_Soon_Winnie.pdf, accessed 10 January 2019.

Jonathan Sterne, 2012. *MP3: The meaning of a format*. Durham, N.C.: Duke University Press.

Zeynep Tufekci, 2014. "Engineering the public: Big data, surveillance and computational politics," *First Monday*, volume 19, number 7, at <http://firstmonday.org/article/view/4901/4097>, accessed 15 May 2018. doi: <http://dx.doi.org/10.5210/fm.v19i7.4901>, accessed 10 January 2019.

Ruben Verborgh, 2018. "A Web API ecosystem through feature-based reuse" (1 May), at <https://ruben.verborgh.org/articles/web-api-ecosystem/>, accessed 22 August 2018.

Ruben Verborgh, 2017. "Paradigm shifts for the decentralized Web" (20 December), at <https://ruben.verborgh.org/blog/2017/12/20/paradigm-shifts-for-the-decentralized-web/>, accessed 22 August 2018.

Wikipedia, 2018. "Application programming interface," at https://en.wikipedia.org/wiki/Application_programming_interface, accessed 20 June 2018.

Steve Yegge, 2011. "Stevey's Google platforms rant" (12 October), at <https://plus.google.com/+RipRowan/posts/eVeouesvaVX>, accessed 22 June 2018.

Received 16 November 2018; accepted 10 January 2019.

This paper is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

API practices and paradigms: Exploring the protocological parameters of APIs as key facilitators of sociotechnical forms of exchange
by Eric Snodgrass and Winnie Soon.

First Monday, Volume 24, Number 2 - 4 February 2019

<https://firstmonday.org/ojs/index.php/fm/rt/prINTERfriendly/9553/7721>

doi: <http://dx.doi.org/10.5210/fm.v24i2.9553>