{"logo":"API Evangelist"}

API Industry Guide

API Definitions

July 2015

by Kin Lane, the API Evangelist

This is meant to be a field guide to the fast changing world of API definitions, providing you an overview of companies, tooling, common building blocks, and some of the latest news from across the landscape.

What Is The Current State of API Definitions

Defining the surface area of your APIs in a machine readable way is nothing new--we've had Web Services Description Language (WSDL) since early 2001. While WSDL is still a very viable way to define your web services, there has been many advancements in the ways you can define web APIs in 2015.

I have included WSDL in here for the sake of storytelling and discussion. However, in my research I am focusing on API definition formats that can be applied to web APIs. Web Application Description Language (WADL) was the first format that gave us a way to describe the metadata around an API, but it wasn't until Swagger emerged, that API definitions were even a segment of APIs--resulting in several competing formats to choose from in 2015.

So what does an API definition do? It provides a machine readable format for you to describe the surface area of an API. They often start with the name, description, tags, and usually a base URL that can be used to find the APIs. The formats quickly go further, allowing you to describe each endpoint, including its verbs, parameters, and other detail you need to make an API call. While each format has their own approach, they also provide you with a way to describe the responses returned from each API, which potentially as much detail about the underlying data model as possible.

The most recent wave of changes in the API definition space was triggered by Tony Tam creating Swagger, and just a couple years later we have competing formats with API Blueprint from Apiary, and RAML from Mulesoft. One of the side effects of this type of diversity, is we ended up with multiple ways to craft your definition, in JSON, YAML, or Markdown, adding to the XML that WADL offered. This type of variety is a good thing, providing a very rich environment for API innovation.

My API definition research is born out of my API design research, but I felt I needed a separate space to store organizations, building blocks, tooling, and other resources that were all about the structure and definition of APIs, which in my opinion is fast becoming the heartbeat of API operations. While this research began with the latest wave of API definitions like Swagger, I quickly included hypermedia types, and Schema.org to assist us make sure we craft APIs with a common structure, if possible.

As with the other areas of my work, this API definition research is in flux, and subject to shifting around as I develop my own understanding, and find new elements I feel should be included--I am in the early stages of this research, so check back often.

Some of the Organizations Involved with API Definitions

It can be tougher to track on the companies and organizations behind API definitions, as most of them stand on their own as their own community. The spectrum between Google Discovery and Swagger is pretty wide--Google made their discovery format to fit their needs, and not be a standard, while Swagger has developed a personality of its own.

I guess my definition for an organization to be included in this research, it should stand on its own legs. While SmartBear is the steward of the Swagger format, I think Swagger stands on its own. While API Blueprint is still closely aligned with Apiary, I feel like it has its own following aside from Apiary services.

Here are the companies and organizations I currently am tracking on as part of my API definition research--I will add new ones as I learn about each specification.

- **Apache Avro** - Apache Avro is a data serialization system. Avro relies on schemas. When Avro data is read, the schema used when writing it is always present. This permits each datum to be written with no per-value overheads, making serialization both fast and small. This also facilitates use with dynamic, scripting languages, since data, together with its schema, is fully self-describing.

    - **Website:** https://avro.apache.org/docs/current/

- **API Blueprint** - API Blueprint is a documentation-oriented API description language. A couple of semantic assumptions over the plain Markdown. API Blueprint is perfect for designing your Web API and its comprehensive documentation but also for quick prototyping and collaboration. It is easy to learn and even easier to read u2013 after all it is just a form of plain text.

    - **Website:** https://apiblueprint.org/
    - **Twitter:** https://twitter.com/apiblueprint
    - **Github:** https://github.com/apiaryio/api-blueprint

- **Google Discovery** - You can use the Google API Discovery Service to build client libraries, IDE plugins, and other tools that interact with Google APIs. It provides a lightweight, JSON-based API that exposes machine-readable metadata about Google APIs, but could also be used to describe other APIs.

    - **Website:** https://developers.google.com/discovery/
    - **Github:** https://github.com/google

- **Mashery I/O Docs** - You want to make it fast and easy for developers to start building with your API. Mashery I/O Docs let you achieve that with a clean, powerful interface for executing live API calls right from your API documentation. With Mashery I/O Docs, your developers spend less time toggling, cutting, and pasting and more time coding great apps.

    - **Website:** http://www.mashery.com/product/io-docs
    - **Twitter:** https://twitter.com/mashery
    - **Blog:** http://www.mashery.com/blog
    - **Github:** https://github.com/mashery

- **Open Data Protocol (OData)** - OData (Open Data Protocol) is an OASIS standard that defines the best practice for building and consuming RESTful APIs. OData helps you focus on your business logic while building RESTful APIs without having to worry about the approaches to define request and response headers, status codes, HTTP methods, URL conventions, media types, payload formats and query options.

    - **Website:** http://www.odata.org/
    - **Twitter:** https://twitter.com/odataorg
    - **Blog:** http://www.odata.org/blog/
    - **Github:** https://github.com/odata

- **RAML** - RESTful API Modeling Language (RAML) is a simple and succinct way of describing practically-RESTful APIs. It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices. The goal is to help our current API ecosystem by solving immediate problems and then encourage ever better API patterns. RAML is built on broadly-used standards such as YAML and JSON and is a non-proprietary, vendor-neutral open spec.

    - **Website:** http://raml.org/
    - **Blog:** http://open-raml.blogspot.com/

- **RESTdesc** - Semantic descriptions for hypermedia APIs. RESTdesc allows you to capture the functionality of hypermedia APIs, so automated agents can use them. Despite their powerful capabilities, RESTdesc descriptions are easy to master. Description is not a goal in itself: you want your API to be used. See how RESTdesc opens up your API for discovery, based on its functional characteristics.

    - **Website:** http://restdesc.org/

- **Schema.org** - Schema.org is a collaborative, community activity with a mission to create, maintain, and promote schemas for structured data on the Internet, on web pages, in email messages, and beyond. Schema.org vocabulary can be used with many different encodings, including RDFa, Microdata and JSON-LD.

    - **Website:** http://schema.org
    - **Blog:** http://blog.schema.org/

- **Swagger** - Swagger is a machine readable API definition format that has built a number of tools around the specification, including an open source API design editor that allows you to design, import, and export APIs in JSON and YAML, then also generate server, and client side code, as well as interactive documentation.

    - **Website:** http://swagger.io
    - **Twitter:** https://twitter.com/swaggerapi
    - **Blog:** http://swagger.io/blog/
    - **Github:** https://github.com/swagger-api

I would say all of these organization represent the leading API definition formats, some of the other other API definition formats are simply listed below as specifications, etc, and as I evolve this research, or their communities and adoption grows, I may consider adding to this list of organizations.

In my opinion, it would make sense for Mashery and Google to roll their standards out separately from their platforms, and tooling. I'm a firm believer that the API definition formats should be independent from any of its tooling, let alone commercial services. I think Apiary has done well to separate API Blueprint, providing a model for others to follow.

Some of the Common Building Blocks I Have Found

When it comes to the world of API definitions, everything starts with some sort of specification. Then I work to identify other elements, features, services, that are adjacent to the API definition itself. I wanted to identify some of the common building blocks that either generate, manipulate, and work with APIs. I will leave the rest of what you can do with an API definition for other stops along the API lifecycle, housed in other areas of my research like design, deployment, and management.

As I'm going through the communities around each API definition format, here are some of the common building blocks I found, that are making API definition a very important, central truth that can be used throughout the API life-cycle. These building blocks allow you to craft exactly the definition you need to establish an understanding around your APIs, and API operations.

- **Specification** - The root specification for the API definition format, providing a base set of rules for all services and tools that employ the spec.
- **Generator** - Generating an API definition format from an existing system or platform, outputting the portable machine readable format.
- **Parser** - Parses a machine readable API definition and makes ready for use in specific language, structure, or platform.
- **Validator** - Validates an API definition against its formal specification and schema, producing a valid or invalid response, with as much detail as possible.
- **Schema** - Produces a specific schema, from an existing template, in a particular API definition format.
- **Converter** - The conversion of an API definition from one format into another, allowing designers to share API definitions in any format.
- **Database** - Tooling that generates an API definition format from a database connection, allowing for API integration with common database formats.
- **Command-Line** - The usage of machine readable API definitions at the command line.
- **Powershell** - The usage of machine readable API definitions via the powershell interface.
- **Aggregator** - Aggregation of APIs using a machine readable API definition format.
- **Editors** - API definition editor, allowing for the creation, import, and export of API definition formats, providing a simple, IDE like API editing experience.

- **Forms** - Dynamically generating an HTML form from an API definition, allow for the structure, and handling to be driven by an API definition.

These are the common building buildings I'm seeing that directly touch API definitions, and are about making the definition better, more usable, and complete. My goal is to better understand how API definitions are applied, generated, manipulated, aggregated, and ultimately made usable across the API life-cycle.

The best part of this research, is some of these features are only present in some of the API definition formats, and the more we enable interoperability, and encourage common tooling and features across all the API definition formats, the better. This is one of the primary goals of my research, to help standardize how we are viewing each area of the API life-cycle, and encourage as much interoperability, and awareness as we can.

Some of the Open Tooling Available

When I label something tooling, I'm talking about anything you can employ as part of your API operations, that is open--open source, openly licensed, and openly available. I want to understand the common approaches, and what people are using out there in the field, then organize, and share them as part of my research.

I often feel like the tooling area of each API layer is often where the R&D occurs, and people are building the tools they need to get work done. As these tools get adopted, and mature, I will bake them into my building blocks above, and as they grow beyond the people who created them, I will also include in the organizational section of each research. For right now, I just want to go through the wealth of API related tooling I'm finding on Github, and make it known here in my research.

API Blueprint

- **API Blueprint** (https://github.com/apiaryio/api-blueprint/)- API Blueprint is a documentation-oriented API description language. A couple of semantic assumptions over the plain Markdown. API Blueprint is perfect for designing your Web API and its comprehensive documentation but also for quick prototyping and collaboration. It is easy to learn and even easier to read – after all it is just a form of plain text. API Blueprint, its parser, and most of its tools are completely open sourced so you dont have to worry about vendor lock-in. This also means you can freely integrate API Blueprint into any type of product, commercial or not.

- **Apiary Blueprint Parser** (https://github.com/apiaryio/blueprint-parser)- A JavaScript parser of Apiary API blueprints. Uses Node.js then in browser, include the browser version of the parser in your web page or application using the <script> tag. To parse an API blueprint, just call the parse method and pass the blueprint as a parameter. The method will return an object representing the parsed blueprint or throw an exception if the input is invalid.
- **Paw-APIBlueprintGenerator** (https://github.com/apiaryio/paw-apiblueprintgenerator/)- Paw extension providing support to export API Blueprint as a code generator.
- **snowcrash** (https://github.com/apiaryio/snowcrash/)- Snow Crash is the reference API Blueprint parser built on top of the Sundown Markdown parser.

API Design

- **API Blueprint** (https://github.com/apiaryio/api-blueprint/)- API Blueprint is a documentation-oriented API description language. A couple of semantic assumptions over the plain Markdown. API Blueprint is perfect for designing your Web API and its comprehensive documentation but also for quick prototyping and collaboration. It is easy to learn and even easier to read – after all it is just a form of plain text. API Blueprint, its parser, and most of its tools are completely open sourced so you dont have to worry about vendor lock-in. This also means you can freely integrate API Blueprint into any type of product, commercial or not.
- **RAML Specification** (http://raml.org)- RESTful API Modeling Language (RAML) is a simple and succinct way of describing practically-RESTful APIs. It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices. The goal is to help our current API ecosystem by solving immediate problems and then encourage ever-better API patterns. RAML is built on broadly-used standards such as YAML and JSON and is a non-proprietary, vendor-neutral open spec.
- **Swagger Editor** (http://editor.swagger.wordnik.com/)- Swagger Editor lets you edit API specifications in YAML inside your browser and to preview documentations in real time. Valid Swagger JSON descriptions can then be generated and used with the full Swagger tooling (code generation, documentation, etc).
- **Swagger Specification** (https://github.com/swagger-api/swagger-spec/)- Swagger is a simple yet robust representation of a RESTful API, with a large ecosystem of API tooling that includes code generation, interactive documentation, and much more.

Currently there are thousands of developers supporting Swagger in almost every modern programming language and deployment environment, using the 100% open source software and specification.

API Design Editor

- **Swagger Editor** (http://editor.swagger.wordnik.com/)- Swagger Editor lets you edit API specifications in YAML inside your browser and to preview documentations in real time. Valid Swagger JSON descriptions can then be generated and used with the full Swagger tooling (code generation, documentation, etc).

API Discovery

- **APIs.json** (http://apisjson.org/)- APIs are becoming a crucial part of the Web. Unfortunately however, it remains very difficult to determine the location of these APIs on servers around the Web. The only way to discover APIs and their properties is via human driven search through public search engines or in hand curated API Directory listings. While these methods work, neither can scale to the potentially hundreds of thousands and millions of APIs which will be published over the next few years.

API-Design

- **Apiary Blueprint Parser** (https://github.com/apiaryio/blueprint-parser)- A JavaScript parser of Apiary API blueprints. Uses Node.js then in browser, include the browser version of the parser in your web page or application using the <script> tag. To parse an API blueprint, just call the parse method and pass the blueprint as a parameter. The method will return an object representing the parsed blueprint or throw an exception if the input is invalid.

Client

- **Paw-APIBlueprintGenerator** (https://github.com/apiaryio/paw-apiblueprintgenerator/)- Paw extension providing support to export API Blueprint as a code generator.

Data Specifications

- **JSON API** (http://jsonapi.org/)- If youve ever argued with your team about the way your JSON responses should be formatted, JSON API is your anti-bikeshedding

weapon. By following shared conventions, you can increase productivity, take advantage of generalized tooling, and focus on what matters: your application. Clients built around JSON API are able to take advantage of its features around efficiently caching responses, sometimes eliminating network requests entirely.

- **JSON Schema** (http://json-schema.org/)- Describes your JSON data format in clear, human- and machine-readable documentation that is complete structural validation, useful for automated testing, and validating client-submitted data.
- **JSON-RPC 2.0** (http://www.jsonrpc.org/specification)- JSON-RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over http, or in many various message passing environments. It uses JSON (RFC 4627) as data format.

Forms

- **swagger-form-editor** (https://github.com/swagger-api/swagger-form-editor/)-

GitHub

- **Paw-APIBlueprintGenerator** (https://github.com/apiaryio/paw-apiblueprintgenerator/)- Paw extension providing support to export API Blueprint as a code generator.
- **snowcrash** (https://github.com/apiaryio/snowcrash/)- Snow Crash is the reference API Blueprint parser built on top of the Sundown Markdown parser.

Hypermedia

- **Collection+JSON** (http://amundsen.com/media-types/collection/)- Collection+JSON is a JSON-based read/write hypermedia-type designed to support management and querying of simple collections.
- **HAL** (http://stateless.co/hal_specification.html)- HAL is a simple format that gives a consistent and easy way to hyperlink between resources in your API. Adopting HAL will make your API explorable, and its documentation easily discoverable from within the API itself. In short, it will make your API easier to work with and therefore more attractive to client developers. APIs that adopt HAL can be easily served and consumed using open source libraries available for most major programming

languages. Its also simple enough that you can just deal with it as you would any other JSON.

- **JSON-LD** (http://json-ld.org/)- JSON-LD is a lightweight Linked Data format. It is easy for humans to read and write. It is based on the already successful JSON format and provides a way to help JSON data interoperate at Web-scale. JSON-LD is an ideal data format for programming environments, REST Web services, and unstructured databases such as CouchDB and MongoDB.
- **Mason** (https://github.com/jornwildt/mason)- Mason is a JSON format for introducing hypermedia elements to classic JSON data representations. With Mason you get hypermedia elements for linking and modifying data, features for communicating to client developers and standardized error handling. Mason is built on JSON, reads JSON, writes JSON and generally fits well into a JSON based eco-system.
- **RESTdesc** (http://restdesc.org/)- Semantic descriptions for hypermedia APIs. RESTdesc allows you to capture the functionality of hypermedia APIs, so automated agents can use them. Despite their powerful capabilities, RESTdesc descriptions are easy to master. Description is not a goal in itself: you want your API to be used. See how RESTdesc opens up your API for discovery, based on its functional characteristics.
- **Siren** ()- Siren is a hypermedia specification for representing entities. As HTML is used for visually representing documents on a Web site, Siren is a specification for presenting entities via a Web API. Siren offers structures to communicate information about entities, actions for executing state transitions, and links for client navigation.
- **UBER** (https://rawgit.com/mamund/media-types/master/uber-hypermedia.html)- The Uber message format is a minimal read/write hypermedia type designed to support simple state transfers and ad-hoc hypermedia-based transitions. This document describes both the XML and JSON variants of the format and provides guidelines for supporting Uber messages over the HTTP protocol.

Media Type

- **Collection+JSON** (http://amundsen.com/media-types/collection/)- Collection+JSON is a JSON-based read/write hypermedia-type designed to support management and querying of simple collections.
- **HAL** (http://stateless.co/hal_specification.html)- HAL is a simple format that gives a consistent and easy way to hyperlink between resources in your API. Adopting HAL

will make your API explorable, and its documentation easily discoverable from within the API itself. In short, it will make your API easier to work with and therefore more attractive to client developers. APIs that adopt HAL can be easily served and consumed using open source libraries available for most major programming languages. Its also simple enough that you can just deal with it as you would any other JSON.

- **JSON API** (http://jsonapi.org/)- If youve ever argued with your team about the way your JSON responses should be formatted, JSON API is your anti-bikeshedding weapon. By following shared conventions, you can increase productivity, take advantage of generalized tooling, and focus on what matters: your application. Clients built around JSON API are able to take advantage of its features around efficiently caching responses, sometimes eliminating network requests entirely.
- **JSON-LD** (http://json-ld.org/)- JSON-LD is a lightweight Linked Data format. It is easy for humans to read and write. It is based on the already successful JSON format and provides a way to help JSON data interoperate at Web-scale. JSON-LD is an ideal data format for programming environments, REST Web services, and unstructured databases such as CouchDB and MongoDB.
- **Mason** (https://github.com/jornwildt/mason)- Mason is a JSON format for introducing hypermedia elements to classic JSON data representations. With Mason you get hypermedia elements for linking and modifying data, features for communicating to client developers and standardized error handling. Mason is built on JSON, reads JSON, writes JSON and generally fits well into a JSON based eco-system.
- **RESTdesc** (http://restdesc.org/)- Semantic descriptions for hypermedia APIs. RESTdesc allows you to capture the functionality of hypermedia APIs, so automated agents can use them. Despite their powerful capabilities, RESTdesc descriptions are easy to master. Description is not a goal in itself: you want your API to be used. See how RESTdesc opens up your API for discovery, based on its functional characteristics.
- **Siren** ()- Siren is a hypermedia specification for representing entities. As HTML is used for visually representing documents on a Web site, Siren is a specification for presenting entities via a Web API. Siren offers structures to communicate information about entities, actions for executing state transitions, and links for client navigation.

Parser

- **Apiary Blueprint Parser** (https://github.com/apiaryio/blueprint-parser)- A JavaScript parser of Apiary API blueprints. Uses Node.js then in browser, include the browser version of the parser in your web page or application using the <script> tag. To parse an API blueprint, just call the parse method and pass the blueprint as a parameter. The method will return an object representing the parsed blueprint or throw an exception if the input is invalid.
- **snowcrash** (https://github.com/apiaryio/snowcrash/)- Snow Crash is the reference API Blueprint parser built on top of the Sundown Markdown parser.

RAML

- **RAML Specification** (http://raml.org)- RESTful API Modeling Language (RAML) is a simple and succinct way of describing practically-RESTful APIs. It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices. The goal is to help our current API ecosystem by solving immediate problems and then encourage ever-better API patterns. RAML is built on broadly-used standards such as YAML and JSON and is a non-proprietary, vendor-neutral open spec.

Semantics

- **ALPS - Application-Level Profile Semantics** (http://amundsen.com/hypermedia/profiles/)- The purpose of Application-Level Profile Semantics (ALPS) is to document the application-level semantics of a particular implementation. This is accomplished by describing elements of response representations for a target media type. For example identifying markup elements returned (i.e. semantic HTML ala Microformats) and state transitions (i.e. HTML.A and HTML.FORM elements) that advance the state of the current application.
- **JSON-LD** (http://json-ld.org/)- JSON-LD is a lightweight Linked Data format. It is easy for humans to read and write. It is based on the already successful JSON format and provides a way to help JSON data interoperate at Web-scale. JSON-LD is an ideal data format for programming environments, REST Web services, and unstructured databases such as CouchDB and MongoDB.

Specification

- **API Blueprint** (https://github.com/apiaryio/api-blueprint/)- API Blueprint is a documentation-oriented API description language. A couple of semantic assumptions over the plain Markdown. API Blueprint is perfect for designing your Web API and its comprehensive documentation but also for quick prototyping and collaboration. It is easy to learn and even easier to read – after all it is just a form of plain text. API Blueprint, its parser, and most of its tools are completely open sourced so you dont have to worry about vendor lock-in. This also means you can freely integrate API Blueprint into any type of product, commercial or not.
- **Barrister RPC** (http://barrister.bitmechanic.com/)- Barrister is a RPC system that uses an external interface definition (IDL) file to describe the interfaces and data structures that a component implements. It is similar to tools like Protocol Buffers, Thrift, Avro, and SOAP.
- **Home Documents for HTTP APIs** (http://tools.ietf.org/html/draft-nottingham-json-home-02)- JSON Home Document is an HTTP API definition formated that follows the RFC4627 specification, and has the media type application/json-home.
- **Interpol** (https://github.com/seomoz/interpol)- Interpol is a toolkit for policing your HTTP JSON interface. To use it, define the endpoints of your HTTP API in simple YAML files. Interpol provides multiple tools to work with endpoint definitions.
- **JSON API** (http://jsonapi.org/)- If youve ever argued with your team about the way your JSON responses should be formatted, JSON API is your anti-bikeshedding weapon. By following shared conventions, you can increase productivity, take advantage of generalized tooling, and focus on what matters: your application. Clients built around JSON API are able to take advantage of its features around efficiently caching responses, sometimes eliminating network requests entirely.
- **Markdown Syntax for Object Notation (MSON)** (https://github.com/apiaryio/mson)- MSON is a plain-text, human and machine readable, description format for describing data structures in common markup formats such as JSON, XML or YAML.
- **Open Data Protocol (OData)** (http://www.odata.org/)- OData (Open Data Protocol) is an OASIS standard that defines the best practice for building and consuming RESTful APIs. OData helps you focus on your business logic while building RESTful APIs without having to worry about the approaches to define request and response headers, status codes, HTTP methods, URL conventions, media types, payload formats and query options etc.

- **Postman Collections** (https://www.getpostman.com/docs/collections)- A collection lets you group individual requests together. These requests can be further organized into folders to accurately mirror your API. Requests can also store sample responses when saved in a collection. You can add metadata like name and description too so that all the information that a developer needs to use your API is available easily.
- **RAML Specification** (http://raml.org)- RESTful API Modeling Language (RAML) is a simple and succinct way of describing practically-RESTful APIs. It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices. The goal is to help our current API ecosystem by solving immediate problems and then encourage ever-better API patterns. RAML is built on broadly-used standards such as YAML and JSON and is a non-proprietary, vendor-neutral open spec.
- **RESTful API Description Language (RADL)** (https://github.com/restful-api-description-language/radl)- RESTful API Description Language (RADL) is an XML vocabulary for describing Hypermedia-driven RESTful APIs. Unlike most HTTP API description languages, RADL focuses on defining a truly hypermedia-driven REST API from the clients point of view. Unlike description languages based on JSON or Markdown, RADL makes it easy to integrate documentation written in HTML or XML. The APIs that RADL describes may use any media type, in XML, JSON, HTML, or any other format.
- **RESTful Service Description Language (RSDL)** (http://www.balisage.net/proceedings/vol10/html/robie01/balisagevol10-robie01.html) - The RESTful Service Description Language (RSDL) is a machine- and human-readable XML description of HTTP-based web applications (typically REST web services).
- **Swagger Specification** (https://github.com/swagger-api/swagger-spec/)- Swagger is a simple yet robust representation of a RESTful API, with a large ecosystem of API tooling that includes code generation, interactive documentation, and much more. Currently there are housands of developers supporting Swagger in almost every modern programming language and deployment environment, using the 100% open source software and specification.
- **Web Application Description Language (WADL)** (https://wadl.java.net/)- The Web Application Description Language (WADL) is a machine-readable XML description of HTTP-based web applications (typically REST web services). WADL models the resources provided by a service and the relationships between them. WADL is intended to simplify the reuse of web services that are based on the existing HTTP

architecture of the Web. It is platform and language independent and aims to promote reuse of applications beyond the basic use in a web browser.

- **Web Services Description Language (WSDL)** (http://www.w3.org/tr/wsdl)- WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

Swagger

- **Swagger Specification** (https://github.com/swagger-api/swagger-spec/)- Swagger is a simple yet robust representation of a RESTful API, with a large ecosystem of API tooling that includes code generation, interactive documentation, and much more. Currently there are housands of developers supporting Swagger in almost every modern programming language and deployment environment, using the 100% open source software and specification.
- **swagger-core** (https://github.com/swagger-api/swagger-core/)- Examples and server integrations for generating the Swagger API Specification, which enables easy access to your REST API
- **Swagger2RAML** (https://github.com/8x8cloud/swagger2raml)- A utility to generate RAML documentation from Swagger JSON.

Swagger-Github-Search

- **swagger-core** (https://github.com/swagger-api/swagger-core/)- Examples and server integrations for generating the Swagger API Specification, which enables easy access to your REST API
- **swagger-form-editor** (https://github.com/swagger-api/swagger-form-editor/)-

I have about another 100+ tools to evaluate and potentially include in here. If you know of anything that should be here, let me know. You can also just submit a pull request on the JSON

for the project. I have a pretty high bar for what I include in here, and before I add to the list I try and include as much information as possible, and verify the project is still alive and active.

One of the things I'm trying to accomplish with this research is better understanding of where the opportunity is as well, not just the tooling that exists. I want to better understand what is needed based upon what people are already building. There is a lot of innovation going on around API definition, and the more we share the story of what we are using, the healthier the space will be.

Closing

I took a look at WADL back in 2011, but really didn't excite me. I didn't really catch the API definition bug until Swagger came along. It was Apiary that I think lit fire in this movement, when it came to the potential of API definitions. We still have a lot of work ahead of us when it comes to API definitions, but the amount of incentives for crafting API definitions in 2015, has dramatically increasing--this is me trying to understand and quantify this movement.

This particular research area is meant to understand working with API definitions in particular, but you will actually see API definitions as a potential layer of every one of research areas. The line between API definitions, and API design research is a thin one, as well as with API deployment, and management. I will work to shed anything that doesn't need to be in here, keeping this portion of research just about ways to define APIs, and their overall operations.

If you are new to using API definitions in your API operations, I recommend playing with each of the specifications, via some of the API design editors that have emerged. Each API definition has its benefits, and while I really enjoy using Swagger, I am intrigued with where API Blueprint is going with their defining of hypermedia resources. Each community has their own way of looking at defining APIs, so make sure and do your homework.

As this world comes into focus for me, I'm starting to see the overlap between each areas. As I work to continue understanding hypermedia, starting with Siren, I will do a better job of connecting the dots. Where this research overlaps with my hypermedia research, I will provide links from here to that area. You can also follow my deeper dives into Swagger and API Blueprint, each definition having their own research areas as well.

This is a fast moving area, and suspect it will come into better focus, in the coming months. I'm spending a lot of time crafting API definitions, and talking to people who are building interesting tooling around them. I'm also carving out more time to expand my hypermedia awareness--so stay tuned. Until then, here is some curated news to keep you busy.

Some Of The API Definition News

This is a handful of the latest news I curated, read, and incorporated into my storytelling, and overall research for API definitions. You can find the full list at my API definition research repository, as well as the complete lists of organizations, building blocks, and tooling.

- **MSON Tutorial on Github** (08-14-2015 on github.com) - https://github.com/apiaryio/mson/blob/master/tutorial.md
- **Modern API Service Providers Need To Speak Common API Definition Formats** (08-13-2015 on feedproxy.google.com) -http://feedproxy.google.com/~r/apievangelist/~3/nq7xiz8l9k4/modern-api-service-providers-need-to-speak-common-api-definition-formats
- **Customizing your auto-generated Swagger Definitions in 1.5.x | Swagger** (08-11-2015 on snip.ly) - http://snip.ly/3voo
- **ALPS Mapping Guidelines for UBER** (07-17-2015 on rawgit.com) - https://rawgit.com/alps-io/misc-docs/master/alps-to-uber.html
- **How To Design Great APIs With API-First Design and RAML** (07-10-2015 on www.programmableweb.com) -http://www.programmableweb.com/news/how-to-design-great-apis-api-first-design-and-raml/how-to/2015/07/10
- **Monitor RAML APIs with API Science** (07-08-2015 on blogs.mulesoft.com) - http://blogs.mulesoft.com/monitor-raml-apis-api-science/
- **Introducing Postman Collection Format Schema** (07-02-2015 on blog.getpostman.com) - http://blog.getpostman.com/2015/07/02/introducing-postman-collection-format-schema/
- **APIs.json Driven API Dictionaries For Use In Atom IDE Autocomplete Packages** (06-26-2015 on apievangelist.com) -http://apievangelist.com/2015/06/27/apisjson-driven-api-dictionaries-for-use-in-atom-ide-autocomplete-packages/
- **The Responsive Swagger Driven Version of Slate API Documentation I Was Looking For** (06-26-2015 on apievangelist.com) -http://apievangelist.com/2015/06/27/the-responsive-swagger-driven-version-of-slate-api-documentation-i-was-looking-for/
- **We Should Be Generating Slate From Swagger So We Maintain A Machine Readable Core** (06-25-2015 on apievangelist.com)

-http://apievangelist.com/2015/06/25/we-should-be-generating-slate-from-swagger-so-we-maintain-a-machine-readable-core

- **We Should Be Generating Slate From Swagger So We Maintain A Machine Readable Core** (06-24-2015 on apievangelist.com) -http://apievangelist.com/2015/06/25/we-should-be-generating-slate-from-swagger-so-we-maintain-a-machine-readable-core/
- **APIMATIC Adds New API Validation Endpoint To Their API Client Code Generation API Stack ·** (06-22-2015 on apievangelist.com) -http://apievangelist.com/2015/06/22/apimatic-adds-new-api-validation-endpoint-to-their-api-client-code-generation-api-stack/
- **APIMATIC Adds New API Validation Endpoint To Their API Client Code Generation API Stack** (06-21-2015 on apievangelist.com) -http://apievangelist.com/2015/06/22/apimatic-adds-new-api-validation-endpoint-to-their-api-client-code-generation-api-stack/
- **Parsing Charles Proxy Exports To Generate Swagger Definitions, While Also Linking Them To Each Path** (06-20-2015 on feedproxy.google.com) -http://feedproxy.google.com/~r/apievangelist/~3/8tkdnlfmbio/parsing-charles-proxy-exports-to-generate-swagger-definitions-while-also-linking-them-to-each-path
- **The Swagger Definitions Collection Is The Cherry On Top Of Each API That I Profile** (06-20-2015 on feedproxy.google.com) -http://feedproxy.google.com/~r/apievangelist/~3/ykcmadynh6c/the-swagger-definitions-collection-is-the-cherry-on-top-of-each-api-that-i-profile
- **The APIs.json Discovery Format: Potential Engine in the API Economy** (06-19-2015 on www.infoq.com) - http://www.infoq.com/articles/apis-json-discovery-format#.vyrxchb1gr4.twitter
- **You'll never guess which API documentation tools we use** (06-19-2015 on blog.lateral.io) - https://blog.lateral.io/2015/06/youll-never-guess-which-api-documentation-tools-we-use/
- **Schemas – Successfully Navigating API Complexity** (06-17-2015 on quickbase.intuit.com) - http://quickbase.intuit.com/blog/2015/06/17/schemas-successfully-navigating-api-complexity/
- **What are API Description Languages?** (06-17-2015 on api-university.com) - http://api-university.com/blog/what-are-api-description-languages/

- **My Minimum Viable Definition For A Complete Swagger API Definition ·** (06-15-2015 on apievangelist.com) - http://apievangelist.com/2015/06/15/my-minimum-viable-definition-for-a-complete-swagger-api-definition
- **How to describe APIs?** (06-15-2015 on api-university.com) - http://api-university.com/blog/how-to-describe-apis/
- **My Minimum Viable Definition For A Complete Swagger API Definition** (06-14-2015 on apievangelist.com) - http://apievangelist.com/2015/06/15/my-minimum-viable-definition-for-a-complete-swagger-api-definition/
- **Swagger Represents The API Value Possible, Postman Is Unit Readied As Transaction, And HAR Could Be Evidence Of Value Actually Having Occurred ·** (06-13-2015 on apievangelist.com) - http://apievangelist.com/2015/06/13/swagger-represents-the-api-value-possible-postman-is-unit-readied-as-transaction-and-har-could-be-evidence-of-value-actually-having-occurred/
- **You Gotta Keep Em Separated: Breaking Down APIs Into Smaller Swagger Files ·** (06-13-2015 on apievangelist.com) -http://apievangelist.com/2015/06/13/you-gotta-keep-em-separated-breaking-down-apis-into-smaller-swagger-files/
- **You Gotta Keep Em Separated: Breaking Down APIs Into Smaller Swagger Files** (06-12-2015 on feedproxy.google.com) -http://feedproxy.google.com/~r/apievangelist/~3/-hic39hc0f0/you-gotta-keep-em-separated-breaking-down-apis-into-smaller-swagger-files
- **APIs with Swagger : An Interview with Reverb's Tony Tam** (06-10-2015 on www.infoq.com) - http://www.infoq.com/articles/swagger-interview-tony-tam
- **New JSON API Specification Aims to Speed API Development** (06-10-2015 on www.programmableweb.com) -http://www.programmableweb.com/news/new-json-api-specification-aims-to-speed-api-development/2015/06/10
- **My New API For Asking Questions Of APIs - The Swagger Editio** (06-09-2015 on apievangelist.com) - http://apievangelist.com/2015/06/09/my-new-api-for-asking-questions-of-apis--the-swagger-edition/
- **A Walk Through A Swagger API Definition To Identify The Moving Parts ·** (06-06-2015 on apievangelist.com) -

http://apievangelist.com/2015/06/06/a-walk-through-a-swagger-api-definition-to-identify-the-moving-parts/

- **A Walk Through A Swagger API Definition To Identify The Moving Parts ·** (06-06-2015 on apievangelist.com) - http://apievangelist.com/2015/06/06/a-walk-through-a-swagger-api-definition-to-identify-the-moving-parts/
- **How Do You Know When A Swagger API Definition is Complete? ·** (06-06-2015 on apievangelist.com) - http://apievangelist.com/2015/06/06/how-do-you-know-when-a-swagger-api-definition-is-complete
- **Join @apimatic, @blockspring, and @apievangelist In Completing API Definitions For 1000 Companies In The API Stack ·** (06-06-2015 on apievangelist.com) - http://apievangelist.com/2015/06/05/join-apimatic-blockspring-and-apievangelist-in-completing-api-definitions-for-1000-companies-in-the-api-stack/
- **Join @apimatic, @blockspring, and @apievangelist In Completing API Definitions For 1000 Companies In The API Stack ·** (06-06-2015 on apievangelist.com) - http://apievangelist.com/2015/06/05/join-apimatic-blockspring-and-apievangelist-in-completing-api-definitions-for-1000-companies-in-the-api-stack
- **Comparison of Automatic API Code Generation Tools For Swagger ·** (06-06-2015 on apievangelist.com) -http://apievangelist.com/2015/06/06/comparison-of-automatic-api-code-generation-tools-for-swagger/
- **Comparison of Automatic API Code Generation Tools For Swagger ·** (06-06-2015 on apievangelist.com) -http://apievangelist.com/2015/06/06/comparison-of-automatic-api-code-generation-tools-for-swagger
- **API Blueprint on GitHub—Apiary Blog** (06-06-2015 on blog.apiary.io) - http://blog.apiary.io/2015/06/03/api-blueprint-github
- **Cerebris :: JSON API 1.0** (06-05-2015 on www.cerebris.com) - http://www.cerebris.com/blog/2015/06/04/jsonapi-1-0/
- **Hyperdrive—Apiary Blog** (06-05-2015 on blog.apiary.io) - http://blog.apiary.io/2015/06/04/hyperdrive/
- **A Walk Through A Swagger API Definition To Identify The Moving Parts** (06-05-2015 on apievangelist.com) -

http://apievangelist.com/2015/06/06/a-walk-through-a-swagger-api-definition-to-identify-the-moving-parts/

- **How Do You Know When A Swagger API Definition is Complete?** (06-05-2015 on apievangelist.com) - http://apievangelist.com/2015/06/06/how-do-you-know-when-a-swagger-api-definition-is-complete/
- **The Power of RAML** (06-04-2015 on www.infoq.com) - http://www.infoq.com/articles/power-of-raml
- **Join @apimatic, @blockspring, and @apievangelist In Completing API Definitions For 1000 Companies In The API Stack** (06-04-2015 on apievangelist.com) - http://apievangelist.com/2015/06/05/join-apimatic-blockspring-and-apievangelist-in-completing-api-definitions-for-1000-companies-in-the-api-stack/
- **Parse Adds New Schema API and API Console** (06-04-2015 on www.infoq.com) - http://www.infoq.com/news/2015/06/parse-schema-api-console
- **API Blueprint on GitHub** (06-03-2015 on blog.apiary.io) - http://blog.apiary.io/2015/06/03/api-blueprint-github/
- **We got the moves like swagger!** (06-02-2015 on www.apiman.io) - http://www.apiman.io/blog/api-manager/swagger/service/ui/2015/06/02/swagger.html
- **Article: Article Series: Description, Discovery, and Profiles : The Next Level in Web APIs** (05-29-2015 on www.infoq.com) -http://www.infoq.com/articles/description-discovery-profiles-series-intro
- **API Blueprint on GitHub** (05-28-2015 on blog.apiary.io) - http://blog.apiary.io/2015/05/28/api-blueprint-github
- **Rolling The Dice - Swagger, Postman, and ALPS ·** (05-27-2015 on apievangelist.com) - http://apievangelist.com/2015/05/27/rolling-the-dice--swagger-postman-and-alps/
- **Rolling The Dice - Swagger, Postman, and ALPS ·** (05-27-2015 on apievangelist.com) - http://apievangelist.com/2015/05/27/rolling-the-dice--swagger-postman-and-alps/
- **Founder of API Blueprint Discusses Progress** (05-27-2015 on www.infoq.com) - http://www.infoq.com/news/2015/05/api-blueprint-progress

Thanks for Tuning Into My Research!

Rembember -- You Can Find All Of This On Github (https://github.com/kinlane/api-definitions/), If You Want To Get Involved!

{"logo":"API Evangelist"}

@kinlane

@apievangelist

Make Sure And Share Your Public API Designs At The API Stack--Otherwise All Of This Won't Work!