ffeathers

A technical writing and fiction blog by Sarah Maddox

API types

FEB 16
Posted by **Sarah Maddox**

I'm putting together a list of the various types of API we might encounter. This is primarily a resource for technical writers, who may need to know what type of thing they could be asked to document if they take on the role of API tech writer.

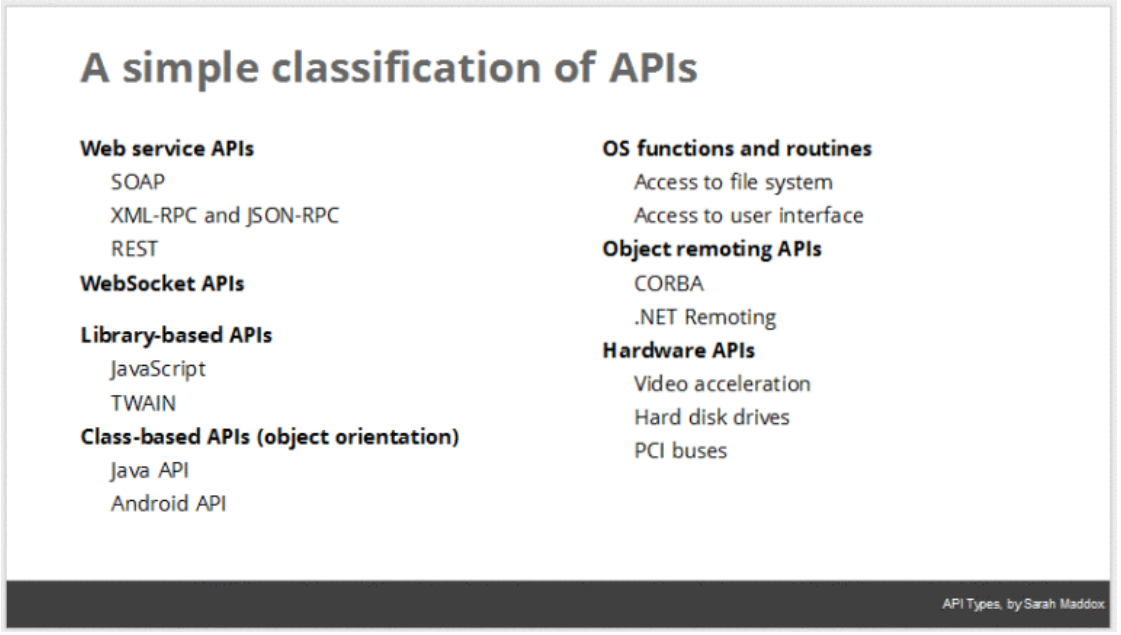
A Google search didn't reveal much material about API types. The best source of information is the Wikipedia page on APIs (http://en.wikipedia.org/wiki/Application_programming_interface).

I tried searching for "API classification" and received plenty of information about engine oil. 😀

So here goes… my attempt at an API classification.

Update: Content now available in a slide deck too

3 May 2014: I've created a slide deck which summarises the information in this post and includes some information from the comments on the post and from discussions with other tech writers. The slide deck is available on SlideShare (http://www.slideshare.net/sarahmaddox/api-types):



(http://www.slideshare.net/sarahmaddox/api-types)

Before we start: What is an API?

API stands for "application programming interface". Put briefly, an API consists of a set of rules describing how one application can interact with another, and the mechanisms that allow such interaction to happen.

What is an interaction between two applications? Typically, an interaction occurs when one application would like to access the data held by another application, or send data to that app. Another interaction might be when one application wants to request a service from another.

A key thing to note: An API is (usually) not a user interface. It provides software-to-software interaction, not user interactions. Sometimes, though, an API may provide a user interface widget, which an app can grab and display.

There are two primary benefits that an API brings:

- Simplification, by providing a layer that hides complexity.
- Standardisation.

Examples:

- Microsoft Word asks the active printer to return its status. Microsoft Word does not care what kind of printer is available. The API worries about that.
- Bloggers on WordPress can embed their Twitter stream into their blog's sidebar. WordPress uses the Twitter API to enable this.

Web service APIs

A web service is a piece of software, or a system, that provides access to its services via an address on the World Wide Web. This address is known as a URI, or URL. The key point is that the web service offers its information in a format that other applications can "understand", or parse.

Examples: The Flickr API (http://www.flickr.com/services/api/), the Google Static Maps API (https://developers.google.com/maps/documentation/staticmaps/), and the other Google Maps web services (https://developers.google.com/maps/documentation/webservices/).

A web service uses HTTP to exchange information. (Or HTTPS, which is an encrypted version of HTTP.)

When an application, the "client", wants to communicate with the web service, the application sends an HTTP request. The web service then sends an HTTP response.

In the request, much of the required information is passed in the URL itself, as paths in the URL and/or as URL parameters.

For example:

`http://maps.googleapis.com/maps/api/staticmap?center=Sydney,NSW&zoom=14&size=400x400&sensor=false` (`http://maps.googleapis.com/maps/api/staticmap?center=Sydney,NSW&zoom=14&size=400x400&sensor=false`)

In addition to the URL, HTTP requests and responses (http://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html) will include information in the header and the body of the message. Request and response "headers" include various types of metadata, such as the browser being used, the content type, language (human, not software), and more.

The body includes additional data in the request or response. Common data formats are XML and JSON. The process of converting data from internal format (for example, a database or a class) to the transferrable format is called "data serialization".

Most often-used types of web service:

- SOAP
- XML-RPC
- JSON-RPC
- REST

SOAP (Simple Object Access Protocol)

SOAP is a protocol that defines the communication method, and the structure of the messages. The data transfer format is XML.

A SOAP service publishes a definition of its interface in a machine-readable document, using WSDL – Web Services Definition Language.

XML-RPC

XML-RPC is an older protocol than SOAP. It uses a specific XML format for data transfer, whereas SOAP allows a proprietary XML format. An XML-RPC call tends to be much simpler, and to use less bandwidth, than a SOAP call. (SOAP is known to be "verbose".) SOAP and XML-RPC have different levels of support in various libraries. There's good information in this Stack Overflow thread (http://stackoverflow.com/questions/80112/whats-the-difference-between-xml-rpc-and-soap).

JSON-RPC

JSON-RPC is similar to XML-RPC, but uses JSON instead of XML for data transfer.

REST (Representational state transfer)

REST is not a protocol, but rather a set of architectural principles. The thing that differentiates a REST service from other web services is its architecture. Some of the characteristics required of a REST service include simplicity of interfaces, identification of resources within the request, and the ability to manipulate the resources via the interface. There are a number of other, more fundamental architectural requirements too.

Looked at from the point of view of a client application, REST services tend to offer an easy-to-parse URL structure, consisting primarily of nouns that reflect the logical, hierarchical categories of the data on offer.

For example, let's say you need to get a list of trees from an API at example-tree-service.com. You might submit a request like this:
`http://example-tree-service.com/trees` (http://example-tree-service.com/trees)

Perhaps you already know the scientific name of a tree family, *Leptospermum*, and you need to know the common name. You request might look like this:
`http://example-tree-service.com/trees/leptospermum` (http://example-tree-service.com/trees/leptospermum)

The tree service might then send a response containing a bunch of information about the *Leptospermum* family, including a field "common-name" containing the value "teatrees".

An example of a REST API: The JIRA REST APIs (https://developer.atlassian.com/display/JIRADEV/JIRA+REST+APIs) from Atlassian.

The most commonly-used data format is JSON or XML. Often the service will offer a choice, and the client can request one or the other by including "json" or "xml" in the URL path or in a URL parameter.

A REST service may publish a WADL document describing the resources it has available, and the methods it will accept to access those resources. WADL stands for Web Application Description Language. It's an XML format that provides a machine-processable description of an HTTP-based Web applications. If there's no WADL document available, developers rely on documentation to tell them what resources and methods are available. Most web services still rely on documentation rather than a machine-readable description of their interface.

In a well-defined REST service, there is no tight coupling between the REST interface and the underlying architecture of the service. This is often cited as the main advantage of REST over RPC (Remote Procedure Call) architectures. Clients calling the service are not dependent on the underlying method names or data structures of the service. Instead, the REST interfaces merely represent the logical resources and functionality available. The structure of the data in the message is independent of the service's data structure. The message contains a representation of the data. Changes to the underlying service must not break the clients.

Library-based APIs

To use this type of API, an application will reference or import a library of code or of binary functions, and use the functions/routines from that library to perform actions and exchange information.

JavaScript APIs are a good example. Take a look at the Google Maps JavaScript API (https://developers.google.com/maps/documentation/javascript/tutorial#Loading_the_Maps_API). To display an interactive Google Map on a web page, you add a <script> tag to include the JavaScript library provided by Google. Then you write your own JavaScript code, calling the Google Maps functions as needed.

Another example is the JavaScript Datastore API from Dropbox (https://www.dropbox.com/developers/datastore/sdks/js). And the Twilio APIs (http://www.twilio.com/docs/libraries) offer libraries for a range of languages and frameworks, including PHP, Python, JavaScript, and many more.

TWAIN (http://en.wikipedia.org/wiki/TWAIN) is an API and communications protocol for scanners and cameras. For example, when you buy an HP scanner you will also get a TWAIN software library, written to comply with the TWAIN standard which supports multiple device types. Applications will use TWAIN to talk to your scanner.

The Oracle Call Interface (OCI) consists of a set of C-language software APIs which provide an interface to the Oracle database.

Class-based APIs (object oriented) – a special type of library-based API

These APIs provide data and functionality organised around classes, as defined in object-oriented languages. Each class offers a discrete set of information and associated behaviours, often corresponding to a human understanding of a concept.

The Java programming community offers a number of good examples of object oriented, or classed-based, APIs. For example:

- The Java API (http://docs.oracle.com/javase/7/docs/api/) itself. This is a set of classes that come along with the Java development environment (JDK) and which are indispensable if you're going to program in Java. The Java language includes the basic syntax and primitive types. The classes in the Java API provide everything else – things like strings, arrays, the renowned Object, and much much more.
- The Android API (http://developer.android.com/reference/packages.html).
- The Google Maps Android API (https://developers.google.com/maps/documentation/android/).

As an example for C#, there's the MSDN Class Library for the .NET Framework (http://msdn.microsoft.com/en-us/library/d11h6832%28v=vs.71%29.aspx). The Twilio APIs (http://www.twilio.com/docs/libraries) mentioned above also include both Java and C#.

Functions or routines in an OS

Operating systems, like Windows and UNIX, provide many functions and routines that we use every day without thinking about it. These OSes offer an API too, so that software programs can interact with the OS.

Examples of functionality provided by the API: Access to the file system, printing documents, displaying the content of a file on the console, error notifications, access to the user interface provided by the OS.

Object remoting APIs

These APIs use a remoting protocol, such as CORBA – Common Object Request Broker Architecture. Such an API works by implementing local proxy objects to represent the remote objects, and interacting with the local object. The same interaction is then duplicated on the remote object, via the protocol.

As far as I can tell, most of these APIs are now considered legacy. Another example is .NET Remoting.

Hardware APIs

Hardware APIs are for manipulating addressable pieces of hardware on a device – things like video acceleration, hard disk drives, PCI buses.

Other developer products

There's more to life than APIs, of course. 🙂 A technical writer may be called upon to document other developer-focused products:

- SDKs – software development kits, which typically contain a set of tools that developers use to interact with, and develop on top of, your product.
- IDE plugins – custom additions to standard development environments, which give developers the extra tools they need to interact with your product from within a development environment like Eclipse, IntelliJ IDEA, or Visual Studio.
- Code libraries that developers can import into their projects.
- Other frameworks that support software development in a specific environment, such as custom XML specifications, templates, UI guidelines.

There's more than one way to can has a cat

Your turn. What have I missed, and are there more useful ways of classifying APIs?

(https://ffeathers.files.wordpress.com/2015/02/screen-shot-2015-02-21-at-6-10-38-am.png)

About Sarah Maddox

Technical writer, author and blogger in Sydney

View all posts by Sarah Maddox »

Posted on 16 February 2014, in APIs, technical writing and tagged API classification, API documentation, JavaScript APIs, REST APIs, technical documentation, technical writing. Bookmark the permalink. 36 Comments.

- ## Leave a comment
- ## Trackbacks 10
- ## Comments 26

David Farbey (@dfarb) | 17 February 2014 at 6:29 am
Hi Sarah,

I think this is a great summary of the types of API that exist. A lot of people don't realise how many different formats APIs can come in, so your links to different examples are very valuable.

Your article explains why a technical writer has to approach creating documentation for each type of API in a different way. For example, if you need to create documentation for an Object Oriented API, that's going to look like the Java Platform API you linked to. However, documentation for a "RESTful" API is going to look completely different, like the JIRA REST API you linked to.

Thanks
David

> *Sarah Maddox* | 17 February 2014 at 6:58 am
> Hallo David 🙂
> You make a great point about the types of documentation required for the different types of APIs. That would be a good follow-up blog post!
> Cheers
> Sarah

Sarah Maddox | 18 February 2014 at 7:48 am
An interesting comment from Whui Mei Yeo on Google+:
Sony has an API for some of their cameras to connect to a device over WiFi Direct and HTTP:
https://developer.sony.com/develop/cameras/

Lois Patterson | 19 February 2014 at 12:11 pm

Thanks for the magnificent post, Sarah. It's particularly helpful for what I am doing right now (API documentation for two different kinds of APIs).

> ***Sarah Maddox*** | 19 February 2014 at 12:17 pm
> Thanks Lois, I'm so glad it's useful!
> Cheers
> Sarah

Thomas Tregner | 8 March 2014 at 1:02 am
Hello Sarah,

Nice list! In and around the term RESTful API, there is Hypermedia API. Depending on who you ask, this a variation, alternative, or a constraint (Hypermedia as the Engine of Application State) of the RESTful style. But it may warrant inclusion in a taxonomy of API styles.

Thanks,

Tom

> ***Sarah Maddox*** | 9 March 2014 at 12:16 pm
> Hallo Tom
>
> Thanks so much! I hadn't heard the term Hypermedia API before. Now I see it's definitely "a thing". 😉 As you hinted, there's a fair bit of discussion about it, and about whether it's a thing in its own right, or simply one of the constraints of REST.
>
> For anyone interested, here are some of the resources I've come across after reading Tom's comment:
>
> HATEOAS, on Wikipedia: http://en.wikipedia.org/wiki/HATEOAS
>
> Getting hyper about hypermedia APIs, on Signal vs Noise: http://signalvnoise.com/posts/3373-getting-hyper-about-hypermedia-apis?56#comments
>
> Hypermedia API, a presentation on SlideShare:
> http://www.slideshare.net/SvitlaSystems/hypermedia-api
>
> A video tutorial: http://www.layer7tech.com/tutorials/hypermedia-apis
>
> Cheers
> Sarah

vola | 11 March 2014 at 4:38 pm
Hello Sarah,
I am just beginning with API documentation and your post is very helpful!
Many thanks!

Sarah Maddox | 9 April 2014 at 5:16 pm
I gave a short presentation based on this post at work today, and a colleague pointed out another group of APIs that I missed: WebSocket APIs. It's great to learn something new! Here's some information about WebSockets:

Introducing WebSockets: Bringing Sockets to the Web from *HTML5 Rocks*

An Introduction to WebSockets from the *Treehouse Blog*

WebSockets from *Mozilla Developer Network*

Thomas Tregner | 9 April 2014 at 11:24 pm

This is where building a taxonomy gets tricky. WebSockets APIs is a great addition to the list. The issue is how we describe abstraction layers for concepts. WebSockets is also the name for an underlying protocol that could apply to any number of APIs that use the protocol. One particular API is WebSockets API which enables access to WebSockets protocol from web pages. When we discuss APIs in the context of WebSockets we may be referring an API for WebSockets protocol, an API built on top of WebSockets API, or an API built on top of the WebSockets API.

For example company #1 may develop a two way service that operates over WebSockets protocol but doesn't use WebSockets API at all. Company #2 may develop a web site that uses WebSockets API for two way interaction between a web page and a server. Company #3 may build their own API that rests on top of WebSockets API and can be consumed by other companies on their web pages. It is tempting to call each of these a WebSockets API. But there is already the WebSockets API. A shared understanding of a term for each would go a long way toward clearer documentation.

This is not an issue limited API terminology. But it seems to happen a lot in the API space. I think that is because information systems are so often described as abstraction layers. Mentioning the layer of concern and giving some context such as a stack diagram in documentation can go a long way toward clearing up these ambiguities.

Sarah Maddox | 10 April 2014 at 7:45 am

Hallo Thomas

Thanks for all the insight into WebSockets! And I think you're right about layers of abstraction. Talking to people as a result of this post, and reading your excellent comment, I've realised that my classification is based both on the implementation of the API itself, and on the way we use it. This makes things a bit murky. And as you've pointed out, there are APIs built on APIs.

Another point of comparison is whether the processing happens client side (such as in a JavaScript API) or server side (such as a web service).

This is one of the best aspects of trying a new classification: The resulting discussions. 🙂

Cheers
Sarah

Sarah Maddox | 3 May 2014 at 6:53 pm

Hallo all
Thanks so much to everyone for all the information. I've created a slide deck which summarises the information in this post and includes some information from the comments on the post and from discussions with other tech writers. The slide deck is available on SlideShare:
http://www.slideshare.net/sarahmaddox/api-types

Cheers
Sarah

Annie | 9 July 2014 at 11:06 am
Thanks so much for this incredibly helpful article, and your whole blog!

I stumbled on this very ingenious explanation of REST and web services and the miracle of the WWW. Thought you and your readers might enjoy it if you haven't seen it:
http://www.looah.com/source/view/2284

> *Sarah Maddox* | 9 July 2014 at 4:35 pm
> Hallo Annie
>
> Thanks for that link! Now that you've pointed it out, I remember reading it a while ago, but it's disappeared from more general viewing due to the "gender-oriented nature" of the article (author's own words). I remember feeling uncomfortable on that front when I first saw the article, and again on re-reading it now.
>
> Even so, he makes some very good points in the article, and it's a good read. I love the "maybe I will (say something)" at the end. I wish he'd republished it in slightly different form.
>
> So, thanks too for an interesting discussion point!
>
> Cheers
> Sarah

Linda Vaitkus | 9 February 2015 at 4:48 am
Count me in! I'm looking forward to this. I taught myself "how to" many years ago (over 20), and feel every project is unique. This will help my confidence …

Jerry Sherman | 21 September 2015 at 9:51 pm
Hi Sarah,
Thank you for all of this information and I am looking forward to learning much more in your workshop today!
Thanks,
Jerry

Jyotika Rajput Mehra | 19 February 2016 at 4:55 am
Hi Sarah,

Thanks a lot for sharing this easy to understand information. This is very helpful. 🙂

RAMESH YADAV | 3 June 2016 at 12:59 am
hi really very thankful to you mam ……

Astra | 17 June 2016 at 8:09 pm
I was confused about API before reading this! Especially when thinking that all APIs are REST API which makes me confused because WordPress has internal API as well and correct me if I'm wrong but WP internal API should be a class based API.

Naveen | 16 August 2016 at 5:58 am

Hi Sarah,

What is the difference between Library-based APIs and REST APIs? In your description, both APIs are using HTTP for request and response.
Could you please elaborate on this?

Thanks
Naveen

> ***Sarah Maddox*** | 17 August 2016 at 3:53 pm
> Hallo Naveen
>
> When using a library-based API, a developer needs to import the API library into her project in order to use the objects/methods etc that the API provides. This isn't strictly necessary when using a REST API, as you simply send an HTTP request over the wire. Sometimes that HTTP request can contain simply the URL plus standard headers. Sometimes you need to send more info in the HTTP body.
>
> Now, it so happens that many REST API providers also supply a client library to help developers use the API. That muddies the waters a bit.
>
> Another way of looking at this, is to talk about client-side APIs and server-side APIs. A JavaScript API is client-side. So is an iOS or Android API. But a REST API and other web service APIs are server-side.
>
> Hope that helps!
> Cheers
> Sarah

Naveen | 20 August 2016 at 9:42 pm
Thank you, Sarah!

Sushobhan | 24 March 2017 at 4:39 pm
What a fantastic piece Sarah. I am not a techie nor a technical writer but as a marketer this is of immense value to me. The know-how is very important. Thank you for the learning.

> ***Sarah Maddox*** | 24 March 2017 at 5:01 pm
> Hallo Sushoban,
> I'm so glad it's useful. Thanks for a lovely comment!
> Cheers
> Sarah

Arun | 17 April 2017 at 2:02 pm
Hi Sarah,

Awesome information.. Masterpiece article about APIs. Keep up the goodwork :).

java366 | 31 January 2019 at 7:57 am
Reblogged this on Site Title.

1. Pingback: **I don't get conferences | Customers and Content**

2. Pingback: **The most popular type of APIs that technical writers document | I'd Rather Be Writing**
3. Pingback: **Breaking Down a Web Service | LogiGear Magazine**
4. Pingback: **What is an API client library? | ffeathers**
5. Pingback: **How a developer accesses an API | ffeathers**
6. Pingback: **API documentation resources | INFORMAZE**
7. Pingback: **RESTful API Testing – VikramVI Knowledge Sharing**
8. Pingback: **Featured Content – Site Title**
9. Pingback: **What is an API? A beginner's introduction | CALLR Blog**
10. Pingback: **What is an API? – Jomari Alang**