Developer Portals Summer 2017

A selection of articles about developer portals and API documentation

PRONOVIX Developer portals

Table Of Contents

Welcome to the API Summer 2017 articles collection

Kristof Van Tomme
CEO and co-founder of Pronovix

With the first half of a busy year behind us, we have taken some time to look back at the last 6 months of research and experiences we accumulated in the developer relations community. This e-magazine is a collection of articles from our developer portals newsletter from January until August 2017, along with some articles we've found interesting but haven't yet sent out. We hope our digest gives you some new insights about developer portals, while you enjoy the last couple of weeks of summer!

Here's the list of topics we featured in this digest:

- API documentation: research findings, free and open source documentation generators, developer portal components, Swagger/ OpenAPI
- APIs in Drupal
- Developer portals: strategy, stakeholders, building trust, developer experience, examples of great developer docs
- Event recaps: API The Docs, Agile The Docs, Write The Docs, DevRelCon Beijing

Subscribe to our newsletter

If you're interested in developer portals and API documentation, make sure you subscribe to our newsletter to receive a copy of our Developer Portal Components white paper and our future research. We also regularly share video recordings of conference talks and workshops. Be the first to hear whenever we have a new blogpost about API documentation, Developer Portals best practices, Developer Evangelism, or about technology that will help you maximise your API's developer experience.

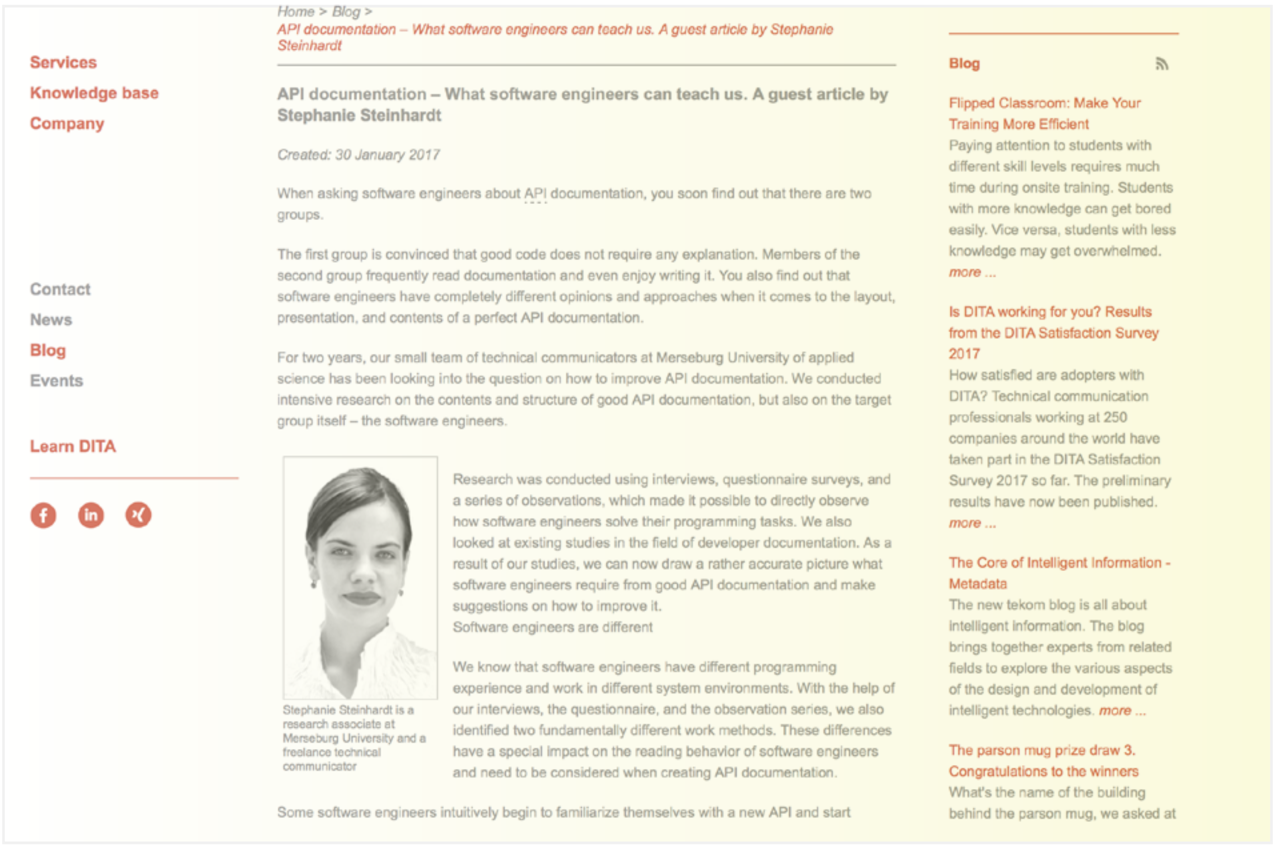
We hope you'll enjoy this emagazine. If so, stay tuned for the next edition that will come to you in winter, early 2018!

Kristof and the Pronovix team

JANUARY 30, 2017

Api Documentation – What Software Engineers Can Teach Us

by Stephanie Steinhardt



Services
Knowledge base
Company

Contact
News
Blog
Events

Learn DITA

Home > Blog >
API documentation – What software engineers can teach us. A guest article by Stephanie Steinhardt

API documentation – What software engineers can teach us. A guest article by Stephanie Steinhardt

Created: 30 January 2017

When asking software engineers about API documentation, you soon find out that there are two groups.

The first group is convinced that good code does not require any explanation. Members of the second group frequently read documentation and even enjoy writing it. You also find out that software engineers have completely different opinions and approaches when it comes to the layout, presentation, and contents of a perfect API documentation.

For two years, our small team of technical communicators at Merseburg University of applied science has been looking into the question on how to improve API documentation. We conducted intensive research on the contents and structure of good API documentation, but also on the target group itself – the software engineers.

Stephanie Steinhardt is a research associate at Merseburg University and a freelance technical communicator

Research was conducted using interviews, questionnaire surveys, and a series of observations, which made it possible to directly observe how software engineers solve their programming tasks. We also looked at existing studies in the field of developer documentation. As a result of our studies, we can now draw a rather accurate picture what software engineers require from good API documentation and make suggestions on how to improve it.
Software engineers are different

We know that software engineers have different programming experience and work in different system environments. With the help of our interviews, the questionnaire, and the observation series, we also identified two fundamentally different work methods. These differences have a special impact on the reading behavior of software engineers and need to be considered when creating API documentation.

Some software engineers intuitively begin to familiarize themselves with a new API and start

Blog

Flipped Classroom: Make Your Training More Efficient
Paying attention to students with different skill levels requires much time during onsite training. Students with more knowledge can get bored easily. Vice versa, students with less knowledge may get overwhelmed. *more ...*

Is DITA working for you? Results from the DITA Satisfaction Survey 2017
How satisfied are adopters with DITA? Technical communication professionals working at 250 companies around the world have taken part in the DITA Satisfaction Survey 2017 so far. The preliminary results have now been published. *more ...*

The Core of Intelligent Information - Metadata
The new tekom blog is all about intelligent information. The blog brings together experts from related fields to explore the various aspects of the design and development of intelligent technologies. *more ...*

The parson mug prize draw 3. Congratulations to the winners
What's the name of the building behind the parson mug, we asked at

https://www.parson-europe.com/en/blog/440-api-documentation.html

When asking software engineers about API documentation, you soon find out that there are two groups.

The first group is convinced that good code does not require any explanation. Members of the second group frequently read documentation and even enjoy writing it. You also find out that software engineers have completely different opinions and approaches when it comes to the layout, presentation, and contents of a perfect API documentation.

For two years, our small team of technical communicators at Merseburg University of applied science has been looking into the question on how to improve API documentation. We conducted intensive research on the contents and structure of good API documentation, but also on the target group itself – the software engineers.

Research was conducted using interviews, questionnaire surveys, and a series of observations, which made it possible to directly observe how software engineers solve their programming tasks. We also looked at existing studies in the field of developer documentation. As a result of our studies, we can now draw a rather accurate picture what software engineers require from good API documentation and make suggestions on how to improve it.
Software engineers are different

We know that software engineers have different programming experience and work in different system environments. With the help of our interviews, the questionnaire, and the observation series, we also identified two fundamentally different work methods. These differences have a special impact on the reading behavior of software engineers and need to be considered when creating API documentation.

Some software engineers intuitively begin to familiarize themselves with a new API and start working from an example. They prefer to immediately dive into the matter. They get to know the API bottom-up, based on the code. Other engineers find it useful to first understand the API as a whole. Before they start working, they read conceptual information and get into the matter top-down. These different patterns became especially obvious during our series of observation (see image 1). Out of 11 test persons, six viewed and partly read the conceptual information of the tested manufacturer documentation. Five ignored the concept part.

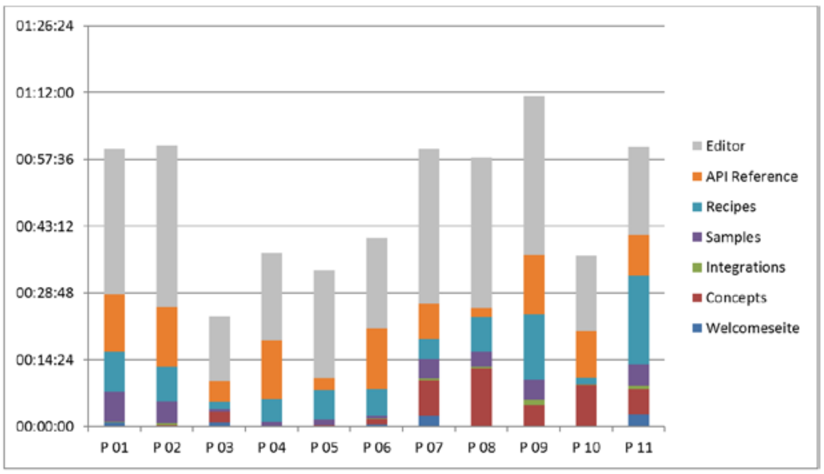


Image 1: Absolute time values: research time (color) vs editing time (gray) of the 11 test persons

Analyzing eye-tracking records, we found that software engineers often did not read pages line by line but rather scanned a page. Visual elements, such as hyperlinks, example code, and the navigation bar, caught their special attention. But they would only actually start reading if the sample code did not solve their problem. This behavior was also influenced by how the software engineers rated the difficulty of a task. If they found a task hard to solve, they were more inclined to read.

Content structure

Looking at the different work methods and the reading behavior of software engineers, it quickly becomes obvious that the design and structure of API documentation is important. In the questionnaire, one of the most mentioned shortcomings in documentation was lack of clarity. That was rated far worse than other shortcomings, such as complicated document structures (see image 2). Today, many manufacturers use different document types to reduce this complexity: getting-started documents, developer guides, references, concepts, and examples. But this classification is not always helpful.

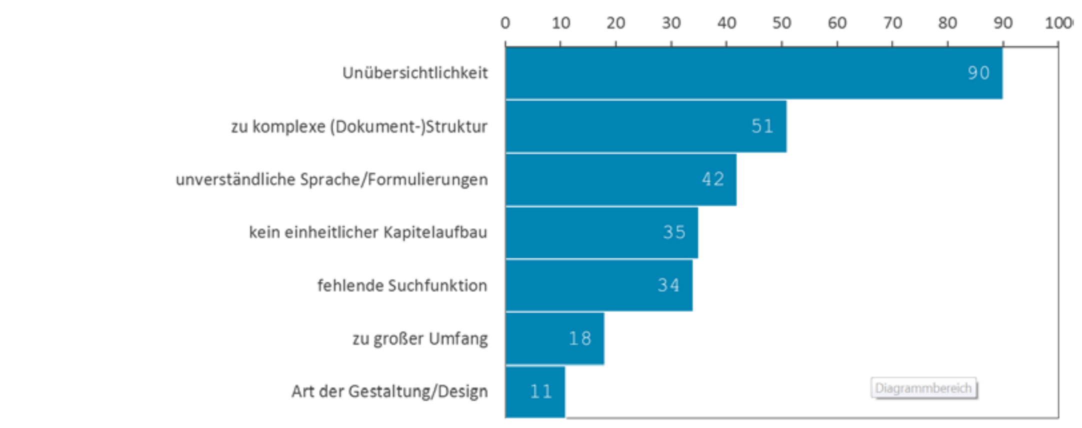


*Image 2: Most mentioned shortcomings in API documentation (113 software engineers could name three problems.) See translation of values at the end of the document.**

In our series of observation, we found that software engineers could not clearly determine which document type would hold the answer to their question. What's the difference between a getting-started document, a tutorial, and a recipe? Technical writers are familiar with the classification of different types of text and information; software engineers are not. For software engineers, it is less important that information is bundled into small portions. They look for answers. They want to quickly find solutions. If they need to decide whether to find the answer in a getting-started document, a reference, or in a tutorial, they are slowed down and not helped sufficiently.

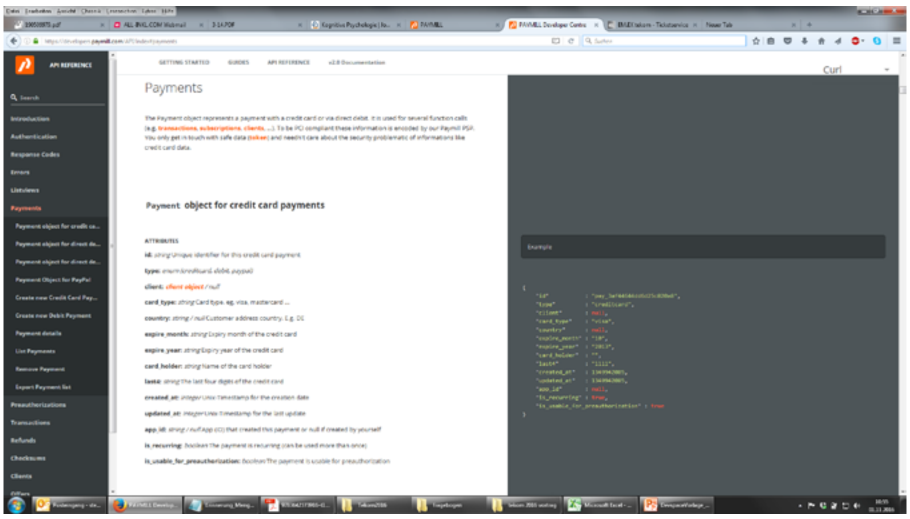
The observation series also showed that software engineers avoided documents with titles they could not clearly interpret. For example, in our study, they ignored the "recipe" document, even though the information they were looking for was in it.

In larger API documentation projects, we still need to cluster information. We may otherwise not find anything. Clustering information by content makes sense, considering that software engineers always want to solve a particular problem with the API. Take a parcel delivery API, for example. To structure it with a getting-started document, concept, or reference would make little sense. The focus should instead be on parcel delivery. This makes it possible to organize the API by service providers, shipments, addresses, and services.

We still need a general overview of and easy access to the API. Whoever decides whether or not to use an API needs to see at first glance what it offers, how it works, and how it integrates. This information should be placed at the same level as the content clusters, for example, as an overview. The title needs to be descriptive so that the reader immediately knows what to expect. This is also important for the integration of the API. Many software engineers told us in interviews and questionnaires that the first steps in an API were the hardest. After this was done, they did not need much documentation. That means that the information on how to integrate the API should also be bundled and made available, similar to a getting-started document.

Navigation is elementary

Inconsistent navigation makes it often even harder to structure API documentation. We frequently found navigation bars in API documentation that disappeared while scrolling, tables of contents at the beginning of chapters, or navigation structures that did not contain all chapter levels. After a few clicks we landed in no-man's land and could only return to our starting point with the browser buttons.
Like in website design we have to consider different criteria for the navigation in API documentation. Breadcrumb menus and foldable navigation bars are useful. Also, the navigation should always be visible so that readers know where exactly they are in the overall structure.



Payments

The Payment object represents a payment with a credit card or via direct debit. It is used for several function calls (e.g. transactions, subscriptions, clients, ...). To be PCI compliant these information is encoded by our Paymill PSP. You only get in touch with safe data (token) and needn't care about the security problematic of informations like credit card data.

Payment object for credit card payments

ATTRIBUTES

id: *string* Unique identifier for this credit card payment

type: *enum(creditcard, debit, paypal)*

client: client object / null

card_type: *string* Card type, eg. visa, mastercard ...

country: *string / null* Customer address country, E.g. DE

expire_month: *string* Expiry month of the credit card

expire_year: *string* Expiry year of the credit card

card_holder: *string* Name of the card holder

last4: *string* The last four digits of the credit card

created_at: *integer* Unix-Timestamp for the creation date

updated_at: *integer* Unix-Timestamp for the last update

app_id: *string / null* App (ID) that created this payment or null if created by yourself

is_recurring: *boolean* The payment is recurring (can be used more than once)

is_usable_for_preauthorization: *boolean* The payment is usable for preauthorization

```
{
    "id"            : "[illegible]",
    "type"          : "creditcard",
    "client"        : null,
    "card_type"     : "visa",
    "country"       : null,
    "expire_month"  : "10",
    "expire_year"   : "2013",
    "card_holder"   : "",
    "last4"         : "1111",
    "created_at"    : [illegible],
    "updated_at"    : [illegible],
    "app_id"        : null,
    "is_recurring"  : true,
    "is_usable_for_preauthorization" : true
}
```

Image 3: Clear overall impression with clear navigation structure, visual division between code examples and descriptive text (source: https://developers.paymill.com/API/index)

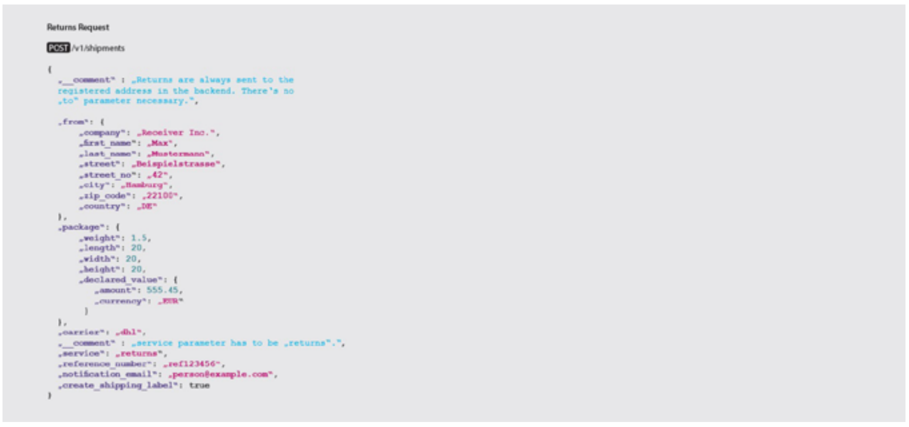
Currently, much of API documentation lacks this kind of transparency. The navigation structure is often too varied. Readers are overwhelmed by the sheer number of possibilities. We need to find a balance between a stern, single-track navigation, which may patronize software engineers, and more diverse approaches that offer, for example, graphical menus for presetting the language, document type or version – but can also quickly cause confusion, despite their inherent flexibility.

Domain knowledge is essential

Results from other research as well as from our interviews also confirmed the findings in our series of observation. Software engineers with background knowledge about the test API could solve tasks much more quickly than those with similar programming experience but without the domain knowledge. The test API in our observation series came from the e-commerce sector. Software engineers who had worked in this field had almost no difficulty using it (see image 1: test persons 3, 4, 5, 6, and 10 came from the e-commerce sector).

That means that building up domain knowledge seems to play an important role for learning a new API. But where do we place this knowledge if we want to avoid a structure by text type and thus skip the "concept" part? Considering the way software engineers work and read, we need to place background knowledge and other important information exactly where they will look.

Based on the results from our interviews, questionnaires, and observation series, only code examples can serve this purpose. Software engineers usually like sample code. In our series of observations, all test persons looked at the samples and used them as a starting point for their own coding. If we disguise important information as code comments and make them visually stand out from other sample code, we can be sure that this content will be seen and also read (see image 4). That does not mean, however, that we should neglect detailed descriptions. They are still read by software developers who are interested in the concepts.



Returns Request

POST /v1/shipments

```
{
  „__comment" : „Returns are always sent to the
  registered address in the backend. There's no
  „to" parameter necessary.",

  „from": {
      „company": „Receiver Inc.",
      „first_name": „Max",
      „last_name": „Mustermann",
      „street": „Beispielstrasse",
      „street_no": „42",
      „city": „Hamburg",
      „zip_code": „22100",
      „country": „DE"
  },
  „package": {
      „weight": 1.5,
      „length": 20,
      „width": 20,
      „height": 20,
      „declared_value": {
         „amount": 555.45,
         „currency": „EUR"
       }
  },
  „carrier": „dhl",
  „__comment" : „service parameter has to be „returns".",
  „service": „returns",
  „reference_number": „ref123456",
  „notification_email": „person@example.com",
  „create_shipping_label": true
}
```

Image 4: Example layout for code comments in sample code (adapted content, based on https://developers.shipcloud.io/recipes/)

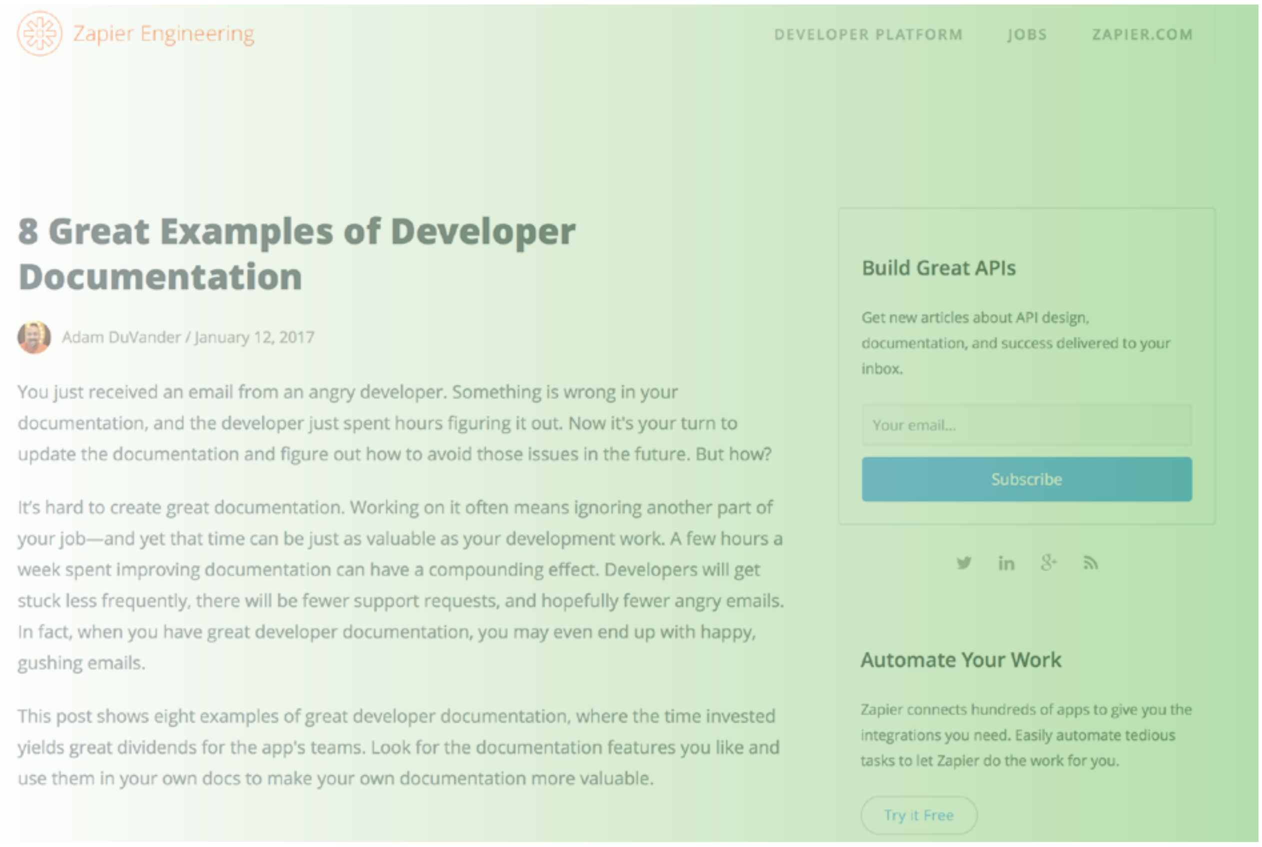
This specific proposal and above mentioned results present only a small part of our previously conducted research. Our research project is still ongoing. We are interested in a larger number of test persons and encourage companies to participate.

** Values in image 2 (top to bottom): complexity, too complex (document) structure, incomprehensible language and phrasing, inconsistent chapter organization, missing search functions, too large, type of layout/design.*
Translation by Uta Lange, parson AG

JANUARY 12, 2017

8 Great Examples of Developer Documentation

by Adam DuVander



8 Great Examples of Developer Documentation

Adam DuVander / January 12, 2017

You just received an email from an angry developer. Something is wrong in your documentation, and the developer just spent hours figuring it out. Now it's your turn to update the documentation and figure out how to avoid those issues in the future. But how?

It's hard to create great documentation. Working on it often means ignoring another part of your job—and yet that time can be just as valuable as your development work. A few hours a week spent improving documentation can have a compounding effect. Developers will get stuck less frequently, there will be fewer support requests, and hopefully fewer angry emails. In fact, when you have great developer documentation, you may even end up with happy, gushing emails.

This post shows eight examples of great developer documentation, where the time invested yields great dividends for the app's teams. Look for the documentation features you like and use them in your own docs to make your own documentation more valuable.

Build Great APIs

Get new articles about API design, documentation, and success delivered to your inbox.

Automate Your Work

Zapier connects hundreds of apps to give you the integrations you need. Easily automate tedious tasks to let Zapier do the work for you.

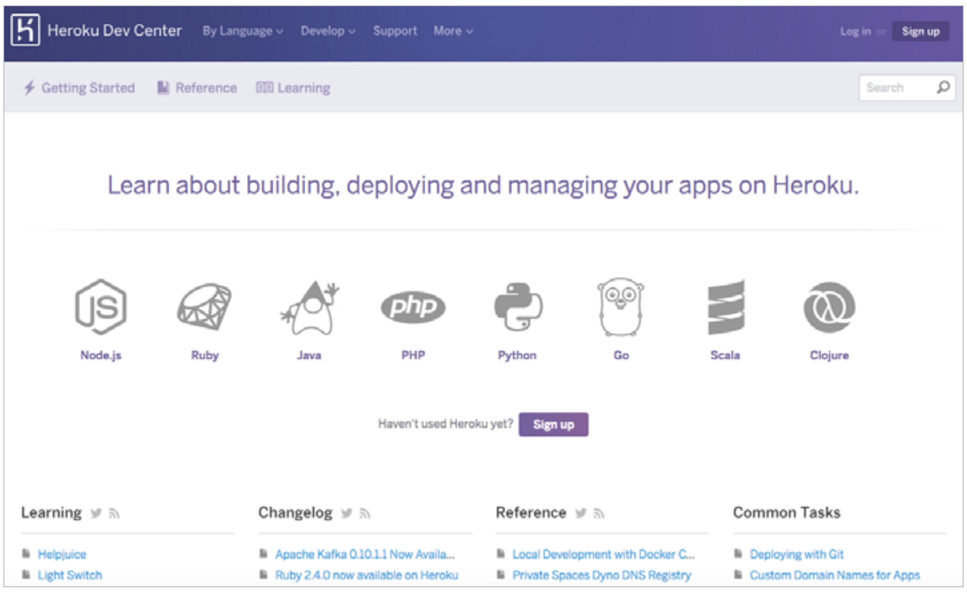
https://zapier.com/engineering/great-documentation-examples/

You just received an email from an angry developer. Something is wrong in your documentation, and the developer just spent hours figuring it out. Now it's your turn to update the documentation and figure out how to avoid those issues in the future. But how?

It's hard to create great documentation. Working on it often means ignoring another part of your job—and yet that time can be just as valuable as your development work. A few hours a week spent improving documentation can have a compounding effect. Developers will get stuck less frequently, there will be fewer support requests, and hopefully fewer angry emails. In fact, when you have great developer documentation, you may even end up with happy, gushing emails.

This post shows eight examples of great developer documentation, where the time invested yields great dividends for the app's teams. Look for the documentation features you like and use them in your own docs to make your own documentation more valuable.

The Developer Home Page



Learn about building, deploying and managing your apps on Heroku.

Learning	Changelog	Reference	Common Tasks
Helpjuice	Apache Kafka 0.10.1.1 Now Availa...	Local Development with Docker C...	Deploying with Git
Light Switch	Ruby 2.4.0 now available on Heroku	Private Spaces Dyno DNS Registry	Custom Domain Names for Apps

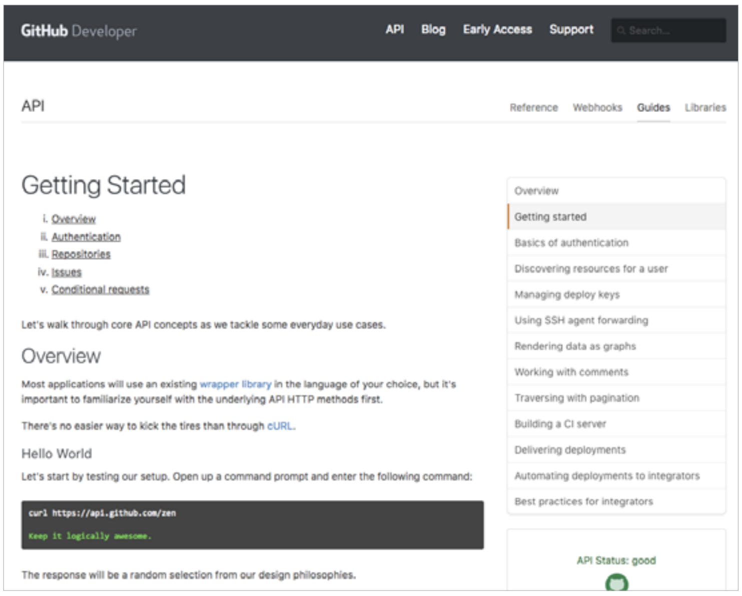
When a developer lands on your documentation home page, they're likely:

1. Curious what you're offering
2. Eager to get started
3. Stuck and in need of help

The documentation home page needs to serve that trio of needs at the same time. The Heroku Dev Center does that with multiple ways to help all three audiences find the information they need. To start, the core non-navigation text on the page shouts the purpose of the site in 30 pixel font: "Learn about building, deploying and managing your apps on Heroku." Below that, it speaks to developers in the eight languages supported by Heroku. Immediately, developers know what Heroku offers and whether there's something for them.

The main and sub navigation also help developers zero in on the reason they're there–whether to solve a problem, get started, or explore more about Heroku. If the major categories don't grab the developer's attention, perhaps that list of common tasks will have what they need. Gather feedback from developers if you aren't sure what to include. Figure out what your readers need most and make sure your developer home page answers it right from the start.

A Getting Started Page



Getting Started

i. Overview
ii. Authentication
iii. Repositories
iv. Issues
v. Conditional requests

Let's walk through core API concepts as we tackle some everyday use cases.

Overview

Most applications will use an existing wrapper library in the language of your choice, but it's important to familiarize yourself with the underlying API HTTP methods first.

There's no easier way to kick the tires than through cURL.

Hello World

Let's start by testing our setup. Open up a command prompt and enter the following command:

```
curl https://api.github.com/zen

Keep it logically awesome.
```

The response will be a random selection from our design philosophies.

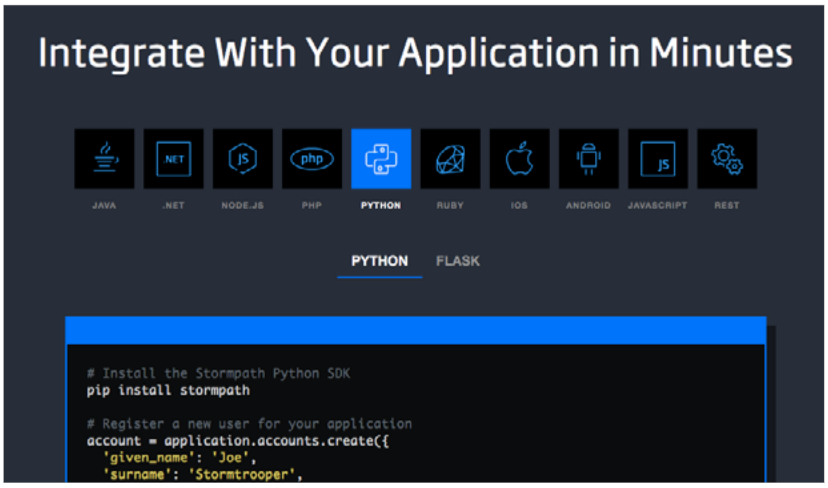
A quickstart or getting started guide plays an important role in introducing new technology to developers. This document or section of your developer website is also part of how you can make your API as popular as pie. And as a likely first impression to developers, it's worth some extra attention.
GitHub is a tool with an advanced audience, but their getting started document doesn't use the reader's knowledge level as an excuse to make the content complex. At over 2,000 words it's not a particularly short guide, but it eases into its overview of what's possible in the API. It starts very simple, working its way up to useful calls including:

1. Un-authenticated test call
2. Request for public user profile
3. Repeat same request with full headers
4. Use basic authentication for the same request
5. Retrieve your own profile with basic authentication

The guide then dives into OAuth authentication, which is admittedly more complex. Developers have already experienced five small victories in successful requests, making them more likely to persevere through the more difficult steps. Many getting started guides would instead begin at this OAuth step, making it more likely for visitors to stop reading. As you consider your own guide, think about how you might start simpler to provide some early wins.

The GitHub guide goes on to cover repositories and issues, both likely key pieces for developers using their API. Then GitHub provides excellent next steps based on use cases, for an obvious next step after developers know the basics.

Language-specific Guides



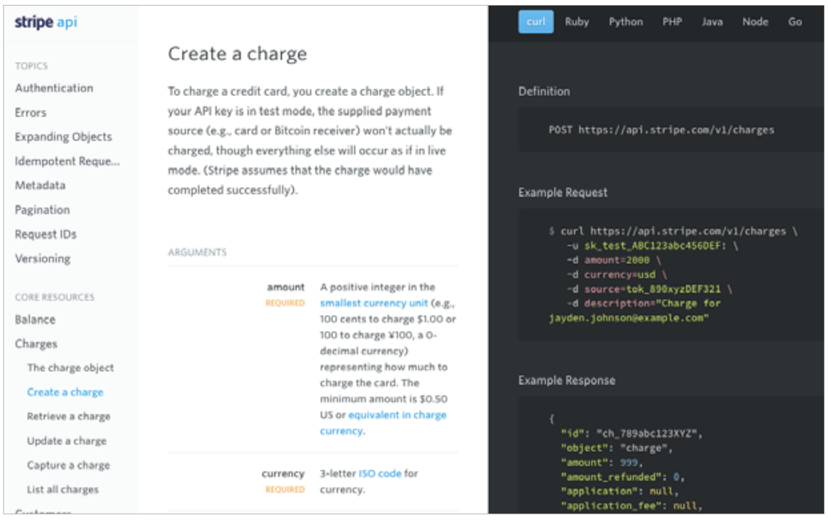
```
# Install the Stormpath Python SDK
pip install stormpath

# Register a new user for your application
account = application.accounts.create({
  'given_name': 'Joe',
  'surname': 'Stormtrooper',
```

The most humbling part of traveling abroad as an English-speaking American is when someone speaks English to me, despite it not being their first language. I get a similar feeling when I come upon documentation that is specific to JavaScript, Python, or another programming language. Great documentation will meet the developers where they are, providing specific instructions tailored to the language or even framework the developer has already chosen.

My entire screen is filled with language options on StormPath's documentation home page. Behind each language is a page with a quickstart, full documentation, an API reference, a project on GitHub, and often more. Each of those resources is specific to the language or framework.

StormPath has 25 distinct language or framework resource pages. That's a lot of effort on their team's part to create and maintain these documents, but it gives them a good chance of speaking the exact dialect of every developer that reaches their site.

Copy-paste Ready Samples



Create a charge

To charge a credit card, you create a charge object. If your API key is in test mode, the supplied payment source (e.g., card or Bitcoin receiver) won't actually be charged, though everything else will occur as if in live mode. (Stripe assumes that the charge would have completed successfully).

ARGUMENTS

amount REQUIRED	A positive integer in the smallest currency unit (e.g., 100 cents to charge $1.00 or 100 to charge ¥100, a 0-decimal currency) representing how much to charge the card. The minimum amount is $0.50 US or equivalent in charge currency.
currency REQUIRED	3-letter ISO code for currency.

Definition

```
POST https://api.stripe.com/v1/charges
```

Example Request

```
$ curl https://api.stripe.com/v1/charges \
   -u sk_test_ABC123abc456DEF: \
   -d amount=2000 \
   -d currency=usd \
   -d source=tok_890xyzDEF321 \
   -d description="Charge for
jayden.johnson@example.com"
```

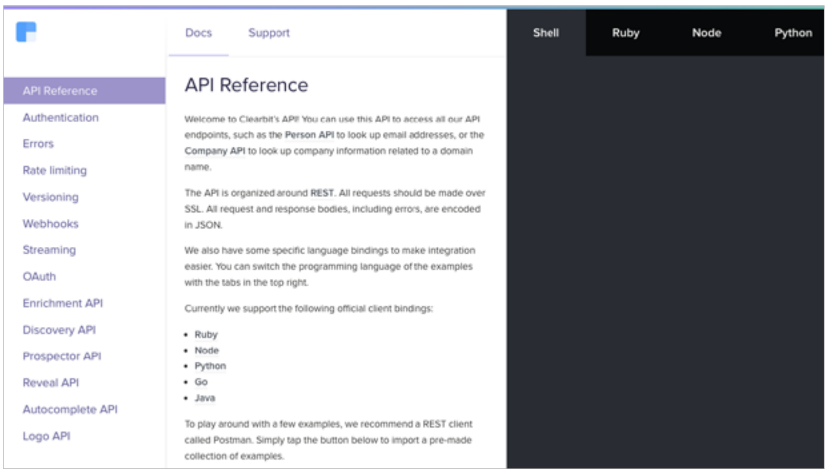
Example Response

```
{
  "id": "ch_789abc123XYZ",
  "object": "charge",
  "amount": 999,
  "amount_refunded": 0,
  "application": null,
  "application_fee": null,
```

Speaking the developer's language is one way to get them started quickly. Another is to provide the code needed in a way the developer can simply copy and paste. You'll find plenty of examples of documentation where the code is almost ready to go: just insert your API key here, or include the appropriate cURL command to make a complete API request. The absolute lowest friction is to supply everything for the developer.

The Stripe API Reference does a fantastic job of copy-paste ready sample calls. Each example request includes the proper cURL parameters, an API key, as well as any identifiers needed for a successful API call. The most impressive part is that you don't need to be logged in, or even have an account, to have a successful API call. That's right: Stripe creates a unique API key for every visitor to its documentation, providing the ultimate low-friction path to sample calls.
Stripe made a huge commitment to its developers, but that's also why the payments company is commonly named amongst the top in providing a great documentation experience. This approach may not be possible for everyone, but it's definitely worth finding ways to reduce friction and make it easier for developers to try your API.

API References



API Reference

Welcome to Clearbit's API! You can use this API to access all our API endpoints, such as the **Person API** to look up email addresses, or the **Company API** to look up company information related to a domain name.

The API is organized around **REST**. All requests should be made over SSL. All request and response bodies, including errors, are encoded in JSON.

We also have some specific language bindings to make integration easier. You can switch the programming language of the examples with the tabs in the top right.

Currently we support the following official client bindings:

- Ruby
- Node
- Python
- Go
- Java

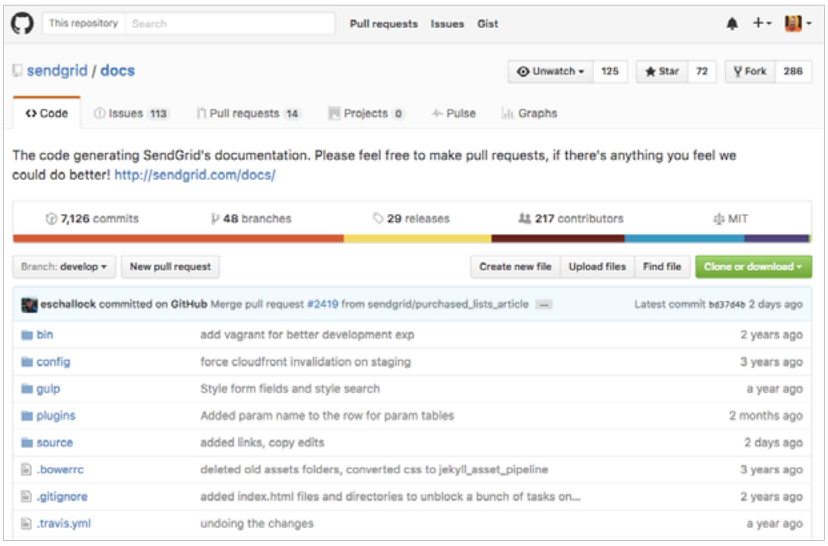
To play around with a few examples, we recommend a REST client called Postman. Simply tap the button below to import a pre-made collection of examples.

Once developers understand the basics of an API, they will likely leave the documentation as they work on their implementation. When they return, they come to answer a specific question. Usually, they'll find the answer in an API reference—something that needs to be easy for them to find.

Clearbit documentation is easy to browse. Since it's on a single page, a great feature of an API reference, it's `Ctrl+F/Cmd+F`-able. That is, you can search using your browser's find functionality. Every section is detailed in the navigation on the left side, which expands as you scroll. The far right column of Clearbit's API reference is dedicated to example requests and responses, organized by language. The snippets can be copied and pasted nearly as-is; you just need to insert your API key.

The best part about Clearbit's API reference, is that it can be yours, too. Clearbit's documentation viewer is based on the open source static documentation tool Slate, which you could use to build your own easily browsable documentation.

Open Source-style Documentation



sendgrid / docs

The code generating SendGrid's documentation. Please feel free to make pull requests, if there's anything you feel we could do better! http://sendgrid.com/docs/

7,126 commits · 48 branches · 29 releases · 217 contributors · MIT

eschallock committed on GitHub Merge pull request #2419 from sendgrid/purchased_lists_article … Latest commit bd37d4b 2 days ago

bin	add vagrant for better development exp	2 years ago
config	force cloudfront invalidation on staging	3 years ago
gulp	Style form fields and style search	a year ago
plugins	Added param name to the row for param tables	2 months ago
source	added links, copy edits	2 days ago
.bowerrc	deleted old assets folders, converted css to jekyll_asset_pipeline	3 years ago
.gitignore	added index.html files and directories to unblock a bunch of tasks on…	2 years ago
.travis.yml	undoing the changes	a year ago

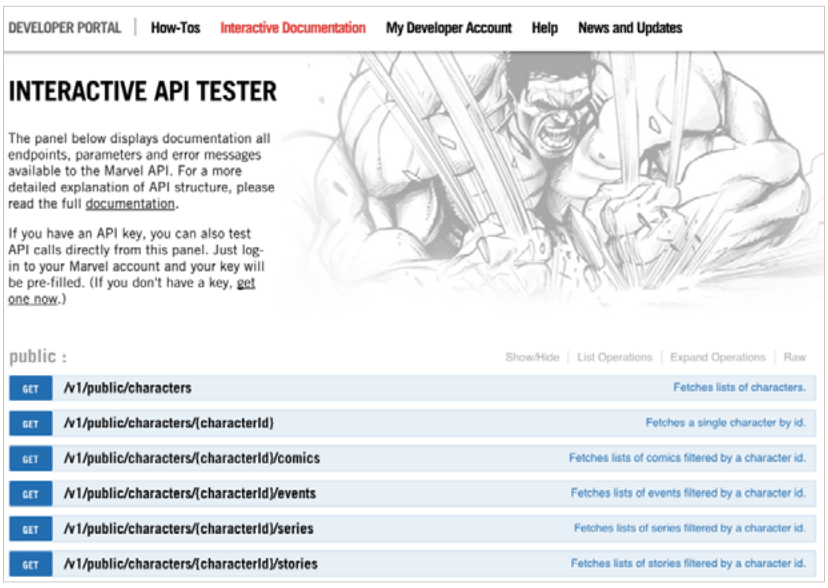
It is important for someone within your company to own your documentation, to ensure its accuracy, and make updates as information changes. That said, you should also solicit feedback from your community–the developers who use your API or tool. One of the best ways to make your commitment to the community clear is to treat your documentation like an open source project.
While I was at SendGrid, my colleague Brandon West open sourced their documentation for a number of reasons:

> *Good documentation allows feedback from readers so they can point out inconsistencies or typos and have them addressed quickly. Even better is providing a feedback loop where those readers can see that their issue has been noted and track progress and see how it fits into the rest of the work to be done. Better still is a place where readers can jump in and submit their own edits if they feel inclined.*

There are now over 200 contributors to the docs repository, most from outside of the company. Plus, hundreds of issues have been tracked and fixed in the three years the repo has been open.
These results sound great, but they require work. For starters, it may take

some engineering effort to extract your documentation from the rest of your codebase. But the real work is the ongoing care of the community: responding to questions, merging pull requests, and ensuring feedback gets to the right internal team.

Interactive Documentation



INTERACTIVE API TESTER

The panel below displays documentation all endpoints, parameters and error messages available to the Marvel API. For a more detailed explanation of API structure, please read the full documentation.

If you have an API key, you can also test API calls directly from this panel. Just log-in to your Marvel account and your key will be pre-filled. (If you don't have a key, get one now.)

public : Show/Hide | List Operations | Expand Operations | Raw

GET	/v1/public/characters	Fetches lists of characters.
GET	/v1/public/characters/{characterId}	Fetches a single character by id.
GET	/v1/public/characters/{characterId}/comics	Fetches lists of comics filtered by a character id.
GET	/v1/public/characters/{characterId}/events	Fetches lists of events filtered by a character id.
GET	/v1/public/characters/{characterId}/series	Fetches lists of series filtered by a character id.
GET	/v1/public/characters/{characterId}/stories	Fetches lists of stories filtered by a character id.

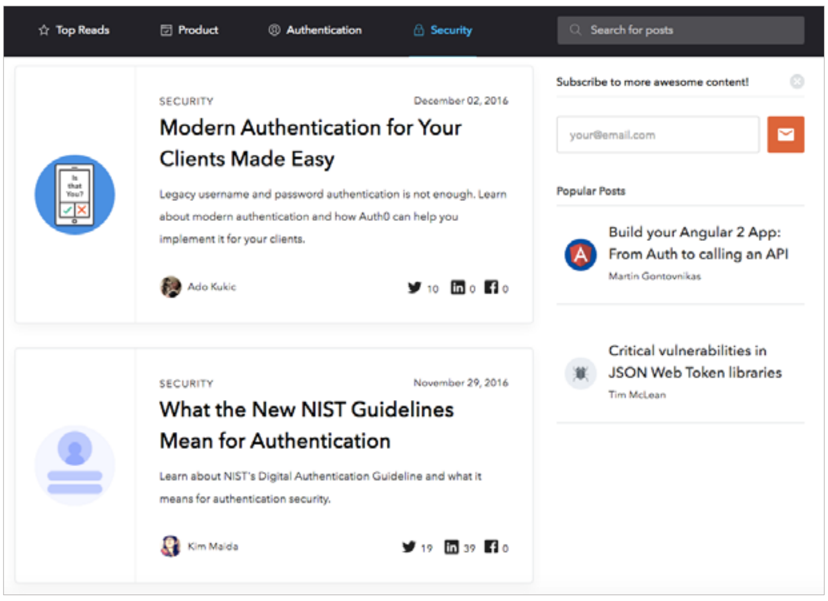
In my teenage years, I played basketball, but I was something of an academic player. I read books on how to practice and improve. One lesson that has stuck with me was the between-the-legs dribble. Once considered a showoff move, the author argued it was now a ball handling requirement. Interactive API explorers are the between-the-legs dribble of developer documentation.

Comic book company Marvel's primary documentation is interactive. Once you have an API key, you can test calls by filling out request fields in a form and clicking the "Try It Out!" button. The response JSON will appear below your form as it returns the same data your application will receive.

The interactive docs are especially useful for the Marvel API, which

requires a hash for live API calls. The Marvel documentation handles the hashing itself, which makes it easier for a developer to see the results before committing the API to code.
Building it doesn't have to be hard, either. There are a number of platforms for interactive documentation, including hosted solutions from Apiary and Readme, to help you make interactive examples for your own documentation

A Developer Blog



SECURITY December 02, 2016
Modern Authentication for Your Clients Made Easy
Legacy username and password authentication is not enough. Learn about modern authentication and how Auth0 can help you implement it for your clients.
Ado Kukic 10 0 0

SECURITY November 29, 2016
What the New NIST Guidelines Mean for Authentication
Learn about NIST's Digital Authentication Guideline and what it means for authentication security.
Kim Maida 19 39 0

Subscribe to more awesome content!
your@email.com

Popular Posts
Build your Angular 2 App: From Auth to calling an API
Martin Gontovnikas
Critical vulnerabilities in JSON Web Token libraries
Tim McLean

The base expectation of documentation is that it accurately describes what's possible with an API or developer tool. Many of the examples in this post have helped developers get started, but there's one more thing you should expect from great documentation: Inspiration. No part of your developer site can provide that as well as a great blog.

While we're pretty big fans of our own developer blog (it's the one you're reading), we also read many others. In fact, we recently shared our 8 Resources for Keeping Up on APIs. One that stands out lately for its useful technical content is Auth0 blog. Since the entire company supports a technical product, this blog also includes the occasional company update, but most of the posts are laser focused on authentication and security topics.

What makes Auth0's take on a developer blog special is that not everything is about their product. The entries understand that developers want to learn something new, so they share knowledge, not features. If a developer has learned a lot about JWT tokens, for example, from reading the blog, they're bound to think of Auth0 when they need to implement the technology.

Creating perfect documentation is difficult, if not impossible. But you can absolutely make your documentation better. Some of these eight examples of great documentation will be a challenge to implement, but there are things you can do to begin today. Improve your getting started guide, organize your documentation by language, or teach one small lesson in a blog post.

DECEMBER 23, 2016

Developer Portal Components - Software Development Kits (SDKs)

by Kathleen De Roo and Kristof Van Tomme

https://pronovix.com/blog/developer-portal-components-part-6-software-development-kits-sdks

The main purpose of Platform Software Development Kits and Helper/ Client Libraries (we'll use "SDKs" to address these collectively in our writing) is to accelerate and simplify development. A well maintained SDK is a trust signal that indicates the level of support and usage of your API for a language, framework, or development platform. So indirectly SDKs work as social proof, that indicates how many communities are already using your API.

In this article, we'll look at how the developer portals in our research sample included SDKs. We'll examine their functions, describe where we found them in the site architecture and deduct best practices.

We'll discuss what kind of SDKs the Portals in our sample used. We'll analyze their choices and evaluate them against the principles that Taylor Barnett from Keen IO shared at APIstrat earlier this year. We'll also talk about the strategic choices that need to be made when deciding what kind of SDKs an API should have.

SDKs are part of the API product

SDKs are software development tools that make it easier to build applications. For web APIs that means SDKs are typically API connector libraries that developers can include into their code. Because SDKs implement APIs in language/platform native functions, they can save developers a lot of time. For this reason **developers will often look for an SDK in their favorite language/framework before they even start exploring your API**.

That is why SDKs need to be done right: it is great if you can offer an SDK for a developer's favorite language, but if you offer one, you need to make sure it works. SDKs should be up-to-date, fully tested and well documented. It is inconvenient if an SDK is missing, but it is way worse if you set an expectation and then break it with a buggy SDK.

In our research, 9 (out of 10) developer portals provided SDKs (Twilio, DigitalOcean, Dwolla, CenturyLink, Keen IO, IBM Cloudant, Apigee, Asana, Mashape).

Why do you need SDKs?

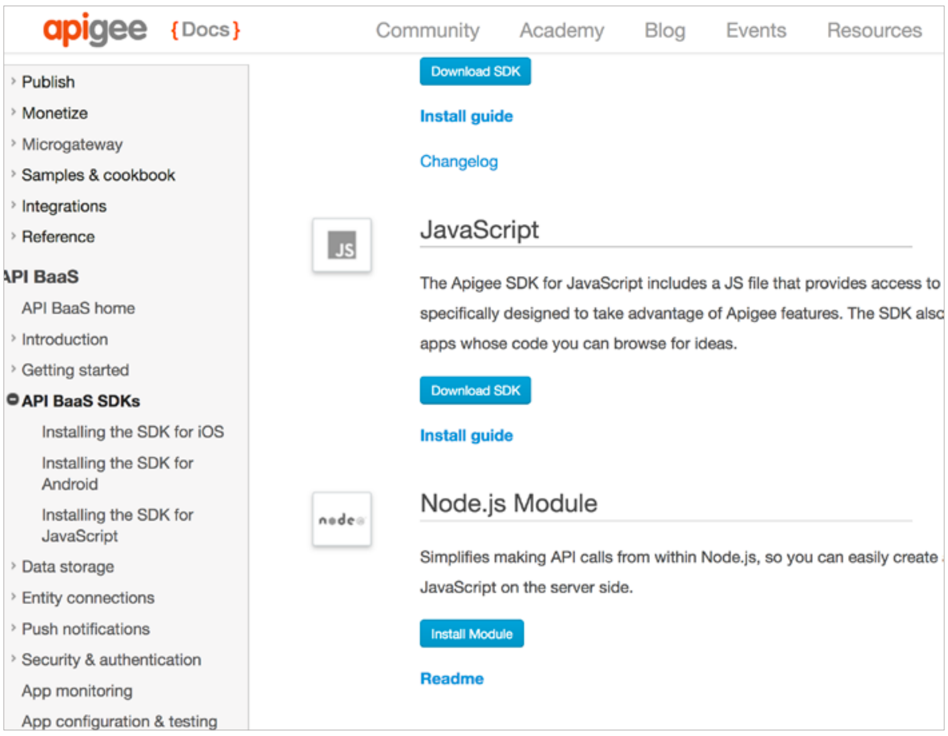
In her 2016 APIstrat talk, Taylor Barnett explained why Keen IO invests in SDKs. The following is derived from her key points:

Better API design through SDKs

A best practice for developer teams is to implement at least one SDK for the APIs they build. This way, during the SDK development, they will experience themselves how easy or hard it is to implement their API. This can help expose bugs or hidden complexity.

Full code coverage

Customers will only implement the API functions they need for their application, an SDK implemented by your API team by contrast can create an API client with full coverage. The resulting SDK will be more useful for your community and will help expose bugs that might otherwise not be found.


apigee {Docs}

Community Academy Blog Events Resources

› Publish
› Monetize
› Microgateway
› Samples & cookbook
› Integrations
› Reference

API BaaS

API BaaS home
› Introduction
› Getting started
API BaaS SDKs
Installing the SDK for iOS
Installing the SDK for Android
Installing the SDK for JavaScript
› Data storage
› Entity connections
› Push notifications
› Security & authentication
App monitoring
App configuration & testing

Download SDK

Install guide

Changelog

JavaScript

The Apigee SDK for JavaScript includes a JS file that provides access to specifically designed to take advantage of Apigee features. The SDK als apps whose code you can browse for ideas.

Download SDK

Install guide

Node.js Module

Simplifies making API calls from within Node.js, so you can easily create JavaScript on the server side.

Install Module

Readme

Example of an SDKs page (Apigee)

SDKs improve the developer experience

Developers just want to get the functionality they seek with as little hassle as possible. It is obviously much easier for them to work with a language or platform that they are already familiar with, that way they don't even need to understand how your API works. Besides this obvious benefit, SDKs also help developers circumvent typical developer experience (DX) problems.

One important example is authentication, which is one of the biggest stumbling blocks when implementing an API. OAuth issues are often cited as a crucial DX problem. While this problem can be alleviated with good documentation, an SDK can help you completely sidestep this problem, allowing developers to use the built-in authentication. An SDK can also implement error handling for your API, which can be a massive boon during debugging.

SDKs demonstrate best practices

For complex APIs, SDKs can help demonstrate best practices to developers. Even if they don't end up using your SDK, developers can see how your APIs are tied together, and how you expect developers to use them.

What types and how many SDKs do you need?

Community SDKs

GitHub enables developers to build and publish an open source community SDK for your APIs.

On first sight that might seem like a great deal: it can be a lot of work to create a good SDK. Not having to pay for the initial development and maintenance saves a lot of costs and if your API becomes very popular this might be a viable strategy. Some companies that have open sourced their application, consciously don't invest in SDKs, and instead expect the community to give back to their platform.

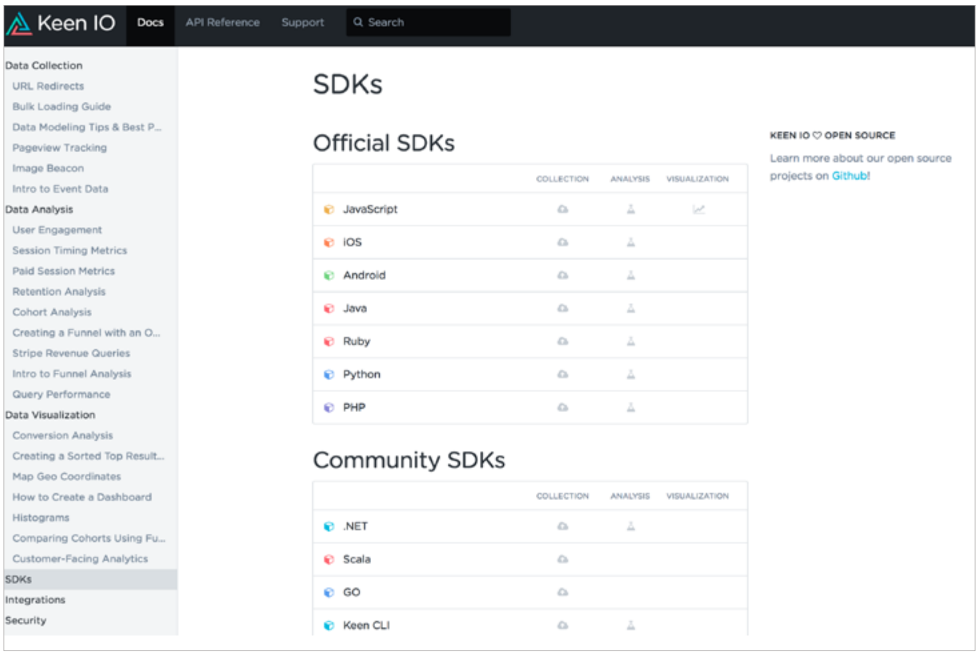
There is, however, a downside. In her talk on SDKs, Taylor Barnett advises that it is better to make what she calls "product" SDKs. She also explains why it is important to clearly differentiate between Product and Community SDKs:

1. To indicate the trustworthiness and the expected longevity of an SDK.
2. To explain differences in the developer experience: community SDKs might not follow all best practices and will probably not be as well documented.

3. To set proper maintenance and support expectations. Even if you make the distinction, some community SDK issues will inevitably be submitted through your Portal's support channels, not addressing them will damage your brand, but if you don't own the code and if you don't have commit rights this might be difficult.

So while it might be tempting to rely on your community to create and maintain open source SDKs, doing so is a form of technical debt. Community SDK maintainers often disappear, or become upset about not being paid while you benefit from their work. As a result your customers might end up integrating with an older version of your API, unaware of best practices, and get frustrated when an SDK doesn't function properly.

In our research sample, Keen IO, Twilio, Asana, DigitalOcean, CenturyLink, IBM Cloudant, Apigee and Dwolla categorized their SDKs according to maintenance status and/or ownership (product/official/supported vs community/third-party).



Data Collection
- URL Redirects
- Bulk Loading Guide
- Data Modeling Tips & Best P...
- Pageview Tracking
- Image Beacon
- Intro to Event Data

Data Analysis
- User Engagement
- Session Timing Metrics
- Paid Session Metrics
- Retention Analysis
- Cohort Analysis
- Creating a Funnel with an O...
- Stripe Revenue Queries
- Intro to Funnel Analysis
- Query Performance

Data Visualization
- Conversion Analysis
- Creating a Sorted Top Result...
- Map Geo Coordinates
- How to Create a Dashboard
- Histograms
- Comparing Cohorts Using Fu...
- Customer-Facing Analytics

SDKs
Integrations
Security

SDKs

Official SDKs

	COLLECTION	ANALYSIS	VISUALIZATION
JavaScript	✓	✓	✓
iOS	✓	✓	
Android	✓	✓	
Java	✓	✓	
Ruby	✓	✓	
Python	✓	✓	
PHP	✓	✓	

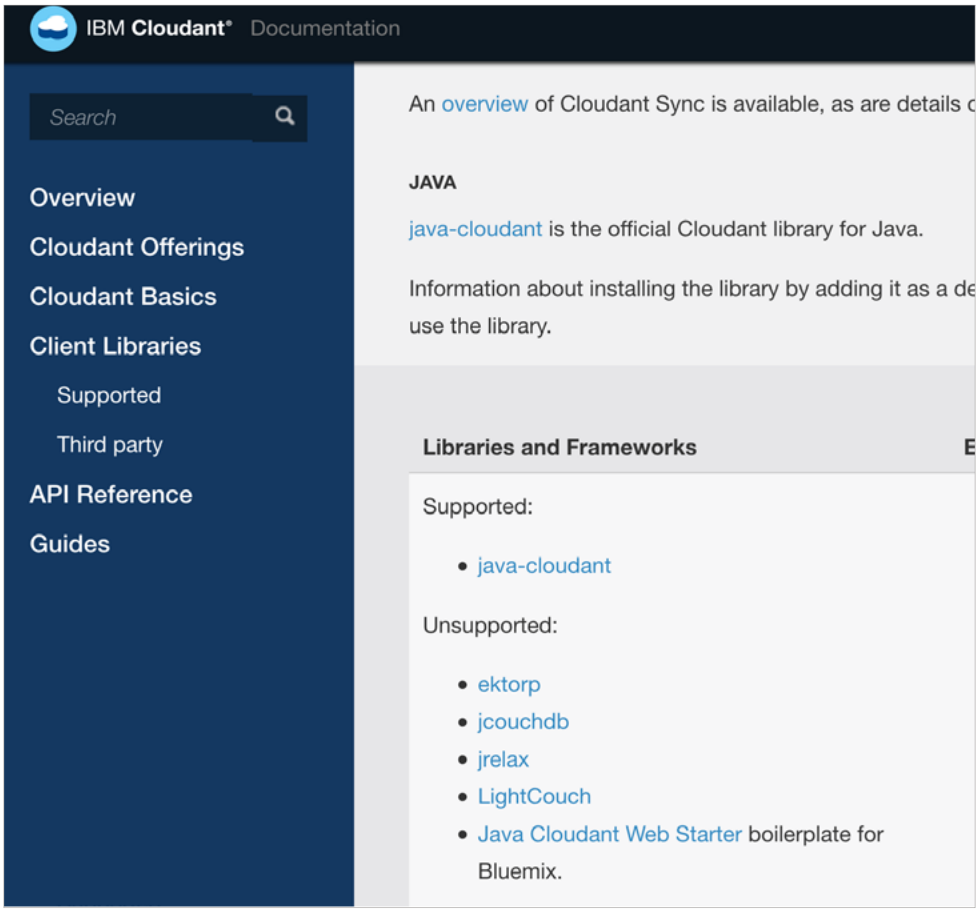
Community SDKs

	COLLECTION	ANALYSIS	VISUALIZATION
.NET	✓	✓	
Scala	✓		
GO	✓		
Keen CLI	✓	✓	

KEEN IO ♡ OPEN SOURCE

Learn more about our open source projects on Github!

Example of SDKs categorized in a product ("official") and a community section. The scope of each SDK is also indicated (Collection, Analysis, Visualization) (Keen IO)



An overview of Cloudant Sync is available, as are details c

JAVA

java-cloudant is the official Cloudant library for Java.

Information about installing the library by adding it as a de
use the library.

Libraries and Frameworks E

Supported:

- java-cloudant

Unsupported:

- ektorp
- jcouchdb
- jrelax
- LightCouch
- Java Cloudant Web Starter boilerplate for Bluemix.

Example of client library types: supported, unsupported, and third party libraries (IBM Cloudant)

Automatic SDK generation

There is a 3rd option besides community and product SDKs: it is also possible to automatically generate SDKs. E.g. APIMATIC is a service that automatically generates SDKs, tests, code samples and documentation. If your API is not too complex, and you don't have the people or resources to make handcrafted product SDKs, this might be worth exploring.

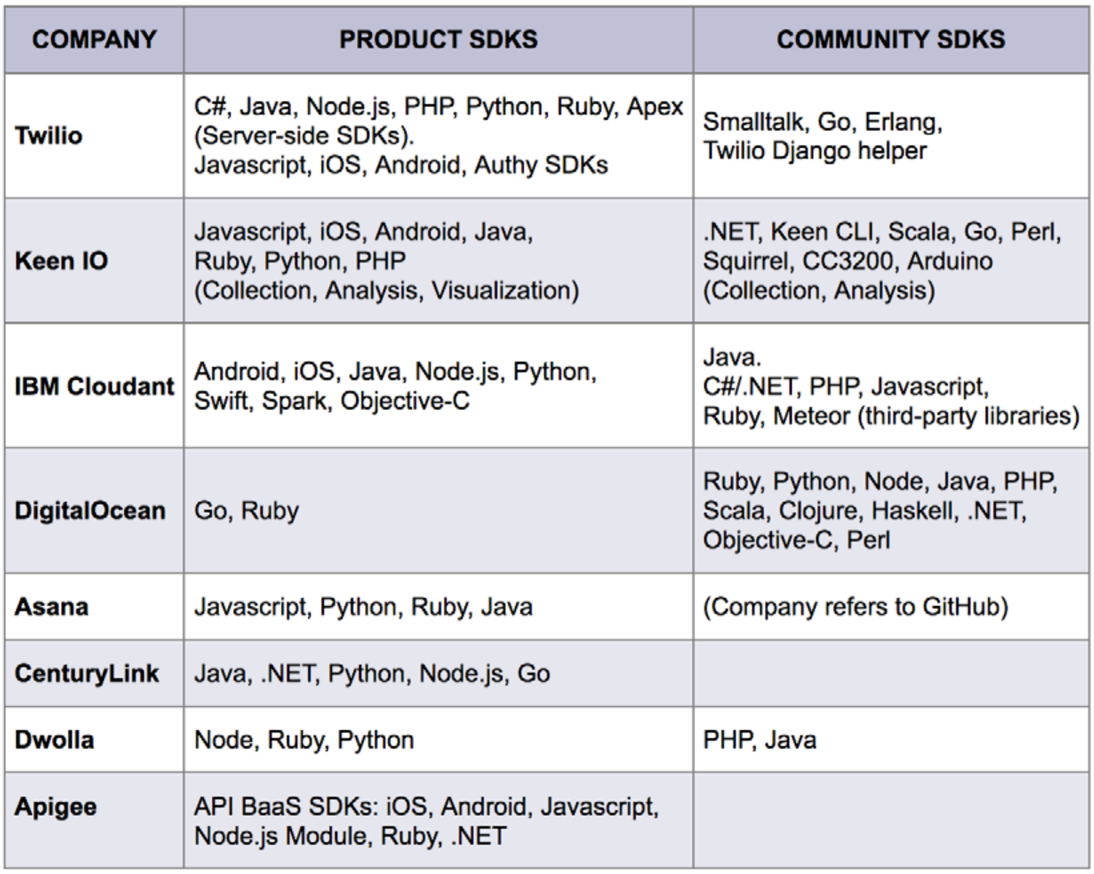
There is, however, a caveat: while automatically generated SDKs save a lot of time and money, and even remove some of the release timing issues caused by sequential development, they lose a lot of the DX benefits that product SDKs give. Without creating at least one SDK, your team won't be able to have immediate feedback on their API design. For the time being, machines also lack the required intelligence of a human developer to extrapolate between the best practices of your API and the programming language or platform the SDK is developed for.

How many SDKs should you create?

It makes sense to split SDKs into functional groups, e.g. to make the distinction between data capture, processing, and visualization: a developer might only need part of this functionality e.g. to integrate with their frontend application. This also means that some parts of your APIs might have SDK coverage in one language and not in another. It is not always straightforward what languages/platforms to build SDKs for, and it might take some investigation to figure out what would be good developer communities to target. **In any case it is a good idea to track the usage of your API, if possible in conjunction with business metrics across different SDKs**. This allows you to analyse what communities are providing you a better income to API calls ratio, so you can maximize your growth and profitability.

In our research sample we found that in the range of published SDKs, the number of product SDKs / community supported SDKs greatly varied from Portal to Portal.

Overview of Libraries and Platform SDKs (as listed on the Portals' SDK pages):

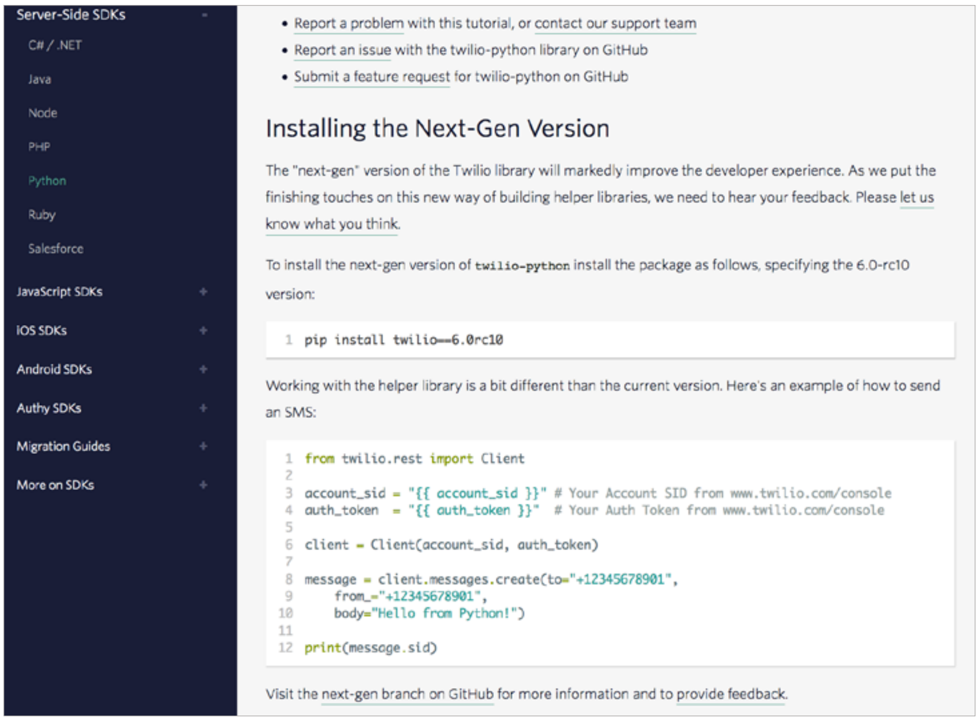


COMPANY	PRODUCT SDKS	COMMUNITY SDKS
Twilio	C#, Java, Node.js, PHP, Python, Ruby, Apex (Server-side SDKs). Javascript, iOS, Android, Authy SDKs	Smalltalk, Go, Erlang, Twilio Django helper
Keen IO	Javascript, iOS, Android, Java, Ruby, Python, PHP (Collection, Analysis, Visualization)	.NET, Keen CLI, Scala, Go, Perl, Squirrel, CC3200, Arduino (Collection, Analysis)
IBM Cloudant	Android, iOS, Java, Node.js, Python, Swift, Spark, Objective-C	Java. C#/.NET, PHP, Javascript, Ruby, Meteor (third-party libraries)
DigitalOcean	Go, Ruby	Ruby, Python, Node, Java, PHP, Scala, Clojure, Haskell, .NET, Objective-C, Perl
Asana	Javascript, Python, Ruby, Java	(Company refers to GitHub)
CenturyLink	Java, .NET, Python, Node.js, Go	
Dwolla	Node, Ruby, Python	PHP, Java
Apigee	API BaaS SDKs: iOS, Android, Javascript, Node.js Module, Ruby, .NET	

Where are SDKs included in a developer portal's information architecture?

The overview page (frontpage) of 6 developer portals provided links to SDKs in their header, footer or body section. Mashape referred to their Unirest libraries in the sidebar menu of its overview page. 2 Portals (Asana and Apigee) included their SDKs in the hierarchical sidebar menu on their documentation pages.

Mashape set up a separate page for its Unirest product, a general purpose library that developers can use to simplify HTTP REST requests. So you could argue that this should not be categorized as a product SDK. The other portals mainly listed their code on GitHub, which is developer-friendly, free for open source projects, and has (at least for the time being) become a de facto standard in the developer community. One company used both GitHub and Google code.



- Report a problem with this tutorial, or contact our support team
- Report an issue with the twilio-python library on GitHub
- Submit a feature request for twilio-python on GitHub

Installing the Next-Gen Version

The "next-gen" version of the Twilio library will markedly improve the developer experience. As we put the finishing touches on this new way of building helper libraries, we need to hear your feedback. Please let us know what you think.

To install the next-gen version of **twilio-python** install the package as follows, specifying the 6.0-rc10 version:

```
pip install twilio==6.0rc10
```

Working with the helper library is a bit different than the current version. Here's an example of how to send an SMS:

```
from twilio.rest import Client

account_sid = "{{ account_sid }}" # Your Account SID from www.twilio.com/console
auth_token  = "{{ auth_token }}"  # Your Auth Token from www.twilio.com/console

client = Client(account_sid, auth_token)

message = client.messages.create(to="+12345678901",
    from_="+12345678901",
    body="Hello from Python!")

print(message.sid)
```

Visit the next-gen branch on GitHub for more information and to provide feedback.

Example of code samples on-site and a reference to GitHub for more information (Twilio)

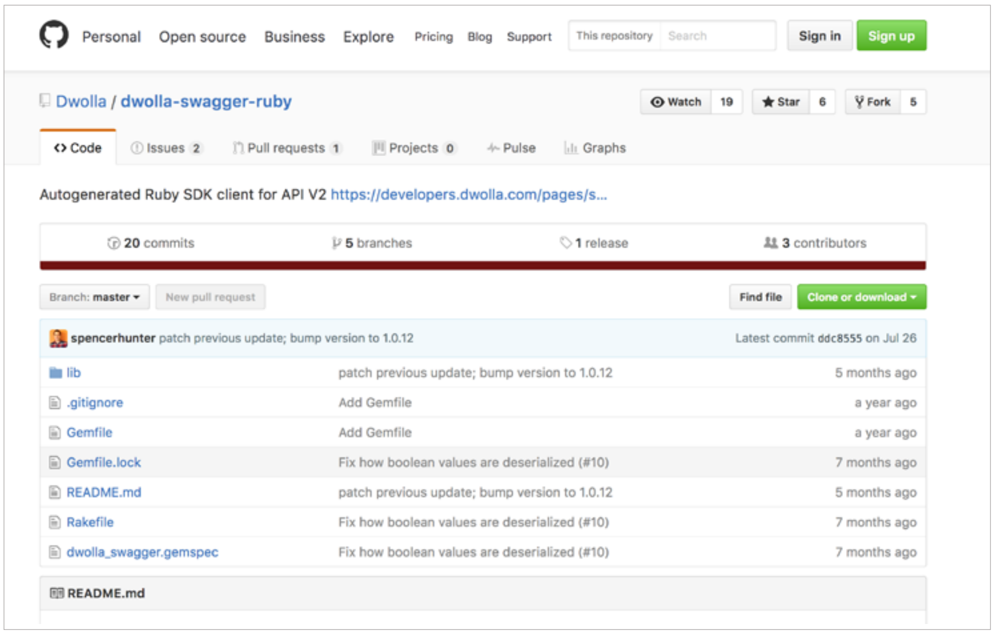
The developer portals in our sample all provided a list of available SDKs.

GitHub is an open platform that doesn't have any barriers that prevent developers from adding community SDKs. It is exactly this permissiveness of GitHub and similar platforms that enables the community SDK phenomenon: allowing API owners and their customers to build further upon the work of other developers that they otherwise might never have access to.

It is however crucial to establish a minimum of community management processes to support and recognize the work of outside developers. API owners need to have a discovery and curation process that helps them identify new community SDKs that need to be evaluated and described so they can be added to the SDK listing on a developer portal.
When community SDKs are not listed on a developer portal, it can be time-consuming for developers to find out what product / community

SDKs are available for an API.

Developer portals need to make it as easy as possible to discover, evaluate, and select an appropriate SDK. Depending on the number of APIs you provide, how related they are, and their complexity you will need to provide tools to help developers navigate your SDK offering.



Dwolla / dwolla-swagger-ruby

Autogenerated Ruby SDK client for API V2 https://developers.dwolla.com/pages/s...

20 commits | 5 branches | 1 release | 3 contributors

spencerhunter patch previous update; bump version to 1.0.12 — Latest commit ddc8555 on Jul 26

lib	patch previous update; bump version to 1.0.12	5 months ago
.gitignore	Add Gemfile	a year ago
Gemfile	Add Gemfile	a year ago
Gemfile.lock	Fix how boolean values are deserialized (#10)	7 months ago
README.md	patch previous update; bump version to 1.0.12	5 months ago
Rakefile	Fix how boolean values are deserialized (#10)	7 months ago
dwolla_swagger.gemspec	Fix how boolean values are deserialized (#10)	7 months ago

README.md

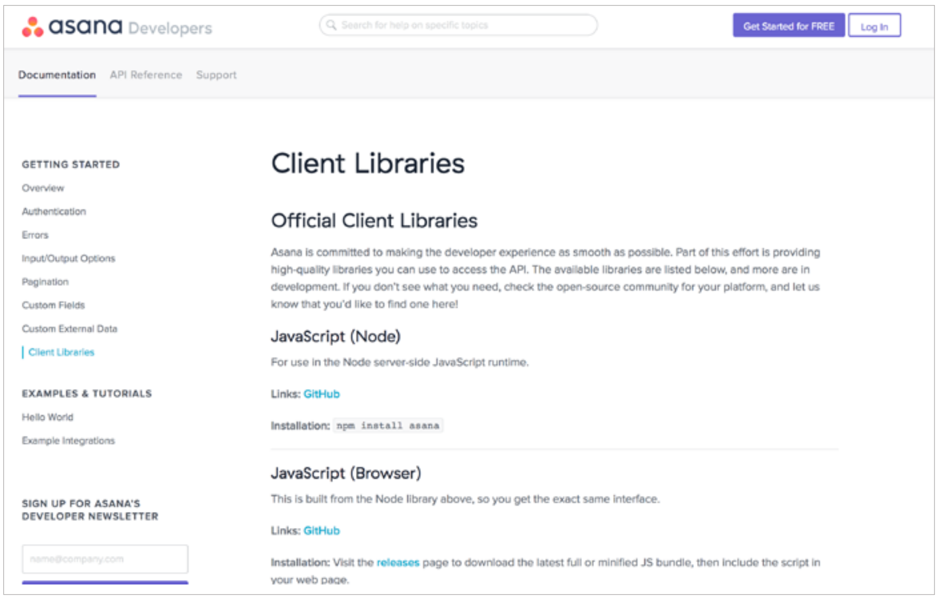
Example of code on GitHub (Dwolla)

Onboarding with SDK documentation

If you want to make it as easy as possible for a developer to get started with your API, you could provide platform specific onboarding documentation. An SDK then becomes a part of your onboarding journey. This allows developers that want to use your API to select their platform, and get instructions on how to develop with your SDK instead of your API. They don't need to switch into an API context and can stay in the context of their platform instead.

If your SDK has any dependencies on third party code, those can become a major DX issue for developers. Taylor Barnett calls it the "Scary world of dependencies" and recommends that SDK developers:

- Carefully evaluate what dependencies to choose when there are multiple options, to make it as easy as possible for your target communities.
- Address dependencies in the onboarding documentation.
- Pay special attention to any changes in dependencies between SDK versions.



Client Libraries

Official Client Libraries

Asana is committed to making the developer experience as smooth as possible. Part of this effort is providing high-quality libraries you can use to access the API. The available libraries are listed below, and more are in development. If you don't see what you need, check the open-source community for your platform, and let us know that you'd like to find one here!

JavaScript (Node)

For use in the Node server-side JavaScript runtime.

Links: GitHub

Installation: `npm install asana`

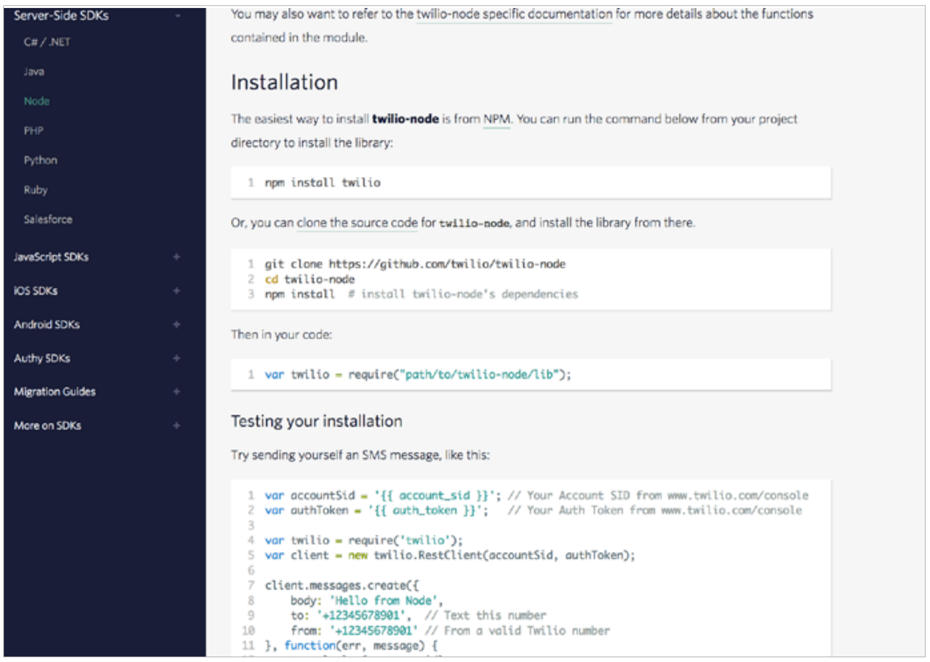
JavaScript (Browser)

This is built from the Node library above, so you get the exact same interface.

Links: GitHub

Installation: Visit the releases page to download the latest full or minified JS bundle, then include the script in your web page.

Example of libraries in the Getting Started section of a developer portal (Asana)



You may also want to refer to the twilio-node specific documentation for more details about the functions contained in the module.

Installation

The easiest way to install **twilio-node** is from NPM. You can run the command below from your project directory to install the library:

```
npm install twilio
```

Or, you can clone the source code for `twilio-node`, and install the library from there.

```
git clone https://github.com/twilio/twilio-node
cd twilio-node
npm install  # install twilio-node's dependencies
```

Then in your code:

```
var twilio = require("path/to/twilio-node/lib");
```

Testing your installation

Try sending yourself an SMS message, like this:

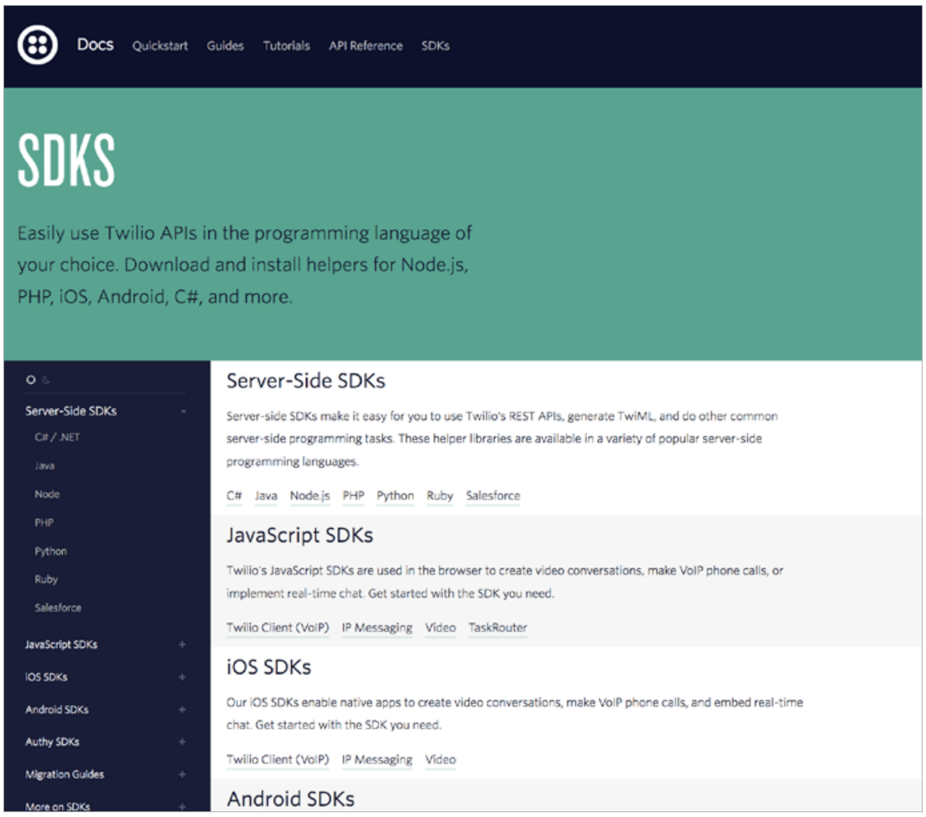
```
var accountSid = '{{ account_sid }}'; // Your Account SID from www.twilio.com/console
var authToken = '{{ auth_token }}';   // Your Auth Token from www.twilio.com/console

var twilio = require('twilio');
var client = new twilio.RestClient(accountSid, authToken);

client.messages.create({
    body: 'Hello from Node',
    to: '+12345678901',  // Text this number
    from: '+12345678901' // From a valid Twilio number
}, function(err, message) {
```

Example of an onboarding guide with SDKs (Twilio)

How are SDKs exposed?

Developer portals implemented:

- Categorization into one, two or three columns that follow a logical grouping for the SDKs so that developers can understand faster where they can find the SDK they need.
- Icons to make it easier to recognise code languages and platforms.
- CTAs (Call to Actions like "View Libraries", "See the source on GitHub") that link to the code repository.
- Headlines ("Find our API in your favorite flavor" - DigitalOcean) to engage users.
- Filtering options (CTAs like "Filter by language" or "Resources by language" or via a hierarchical sidebar menu) for easy content filtering.
- Labelling, like "Product SDKs" and "Community SDKs" (Keen IO) to set proper expectations and to make it clear what the source of an SDK is.
- Visual design elements e.g. change the text and library border color, to make a distinction between product and community SDKs (DigitalOcean).



SDKS

Easily use Twilio APIs in the programming language of your choice. Download and install helpers for Node.js, PHP, iOS, Android, C#, and more.

Server-Side SDKs

Server-side SDKs make it easy for you to use Twilio's REST APIs, generate TwiML, and do other common server-side programming tasks. These helper libraries are available in a variety of popular server-side programming languages.

C# Java Node.js PHP Python Ruby Salesforce

JavaScript SDKs

Twilio's JavaScript SDKs are used in the browser to create video conversations, make VoIP phone calls, or implement real-time chat. Get started with the SDK you need.

Twilio Client (VoIP) IP Messaging Video TaskRouter

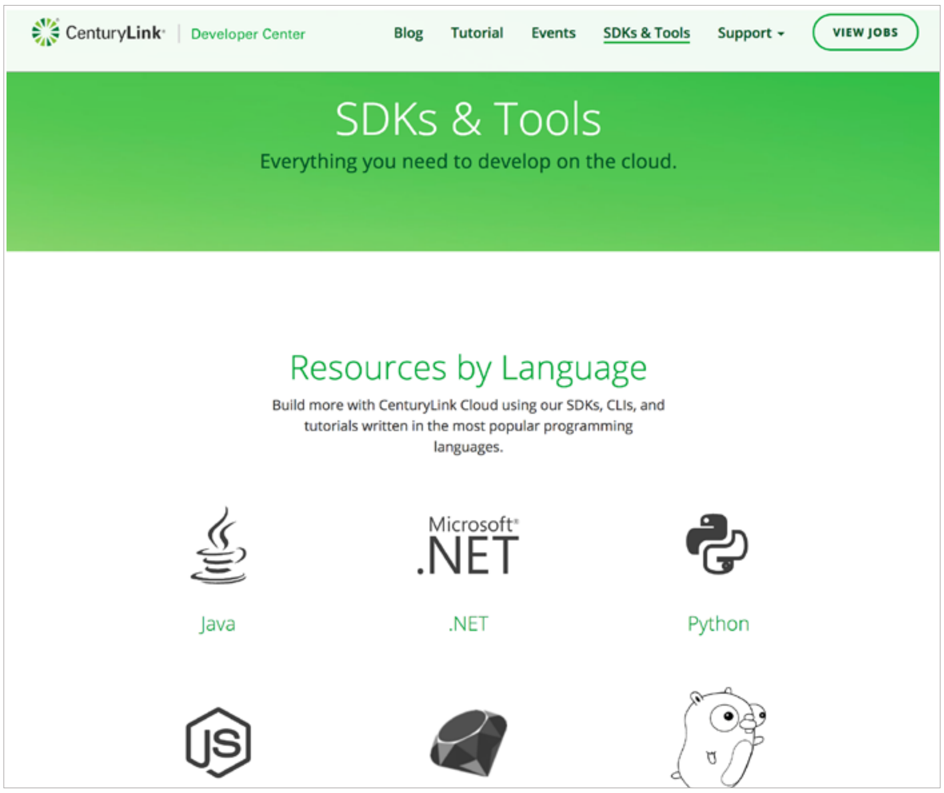
iOS SDKs

Our iOS SDKs enable native apps to create video conversations, make VoIP phone calls, and embed real-time chat. Get started with the SDK you need.

Twilio Client (VoIP) IP Messaging Video

Android SDKs

Example of SDKs with a hierarchical sidebar menu (Twilio)



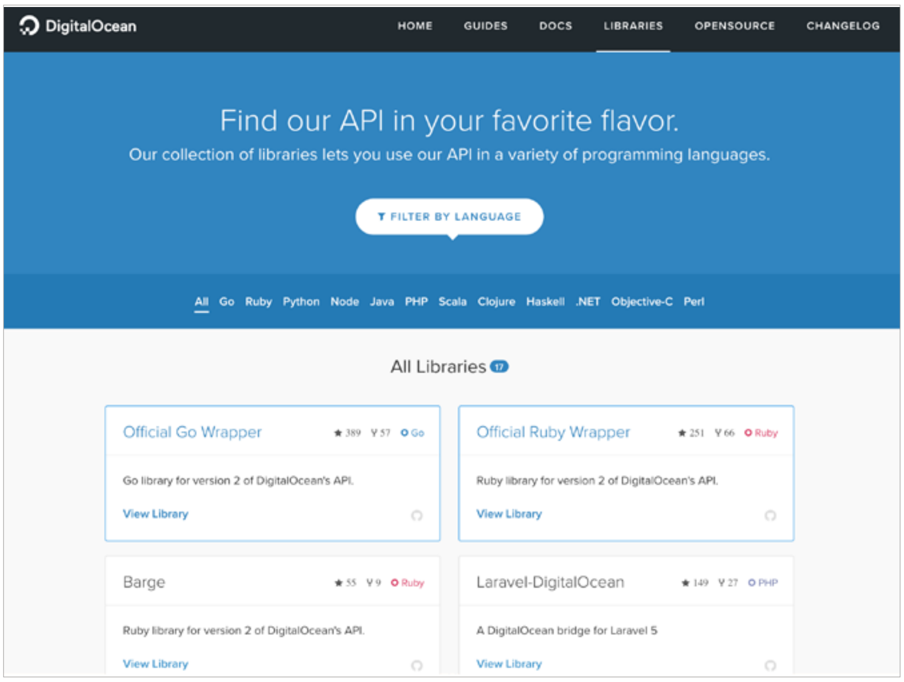
SDKs & Tools

Everything you need to develop on the cloud.

Resources by Language

Build more with CenturyLink Cloud using our SDKs, CLIs, and tutorials written in the most popular programming languages.

Example of an SDK page where the languages are accompanied by their icons (CenturyLink)



Find our API in your favorite flavor.

Our collection of libraries lets you use our API in a variety of programming languages.

All Libraries 17

Official Go Wrapper ★ 389 ⑂ 57 ● Go

Go library for version 2 of DigitalOcean's API.

View Library

Official Ruby Wrapper ★ 251 ⑂ 66 ● Ruby

Ruby library for version 2 of DigitalOcean's API.

View Library

Barge ★ 55 ⑂ 9 ● Ruby

Ruby library for version 2 of DigitalOcean's API.

View Library

Laravel-DigitalOcean ★ 149 ⑂ 27 ● PHP

A DigitalOcean bridge for Laravel 5

View Library

Example of all API Libraries, with language selector, categorized into two columns. The text and table border distinguishes between "official" product and community libraries (DigitalOcean)

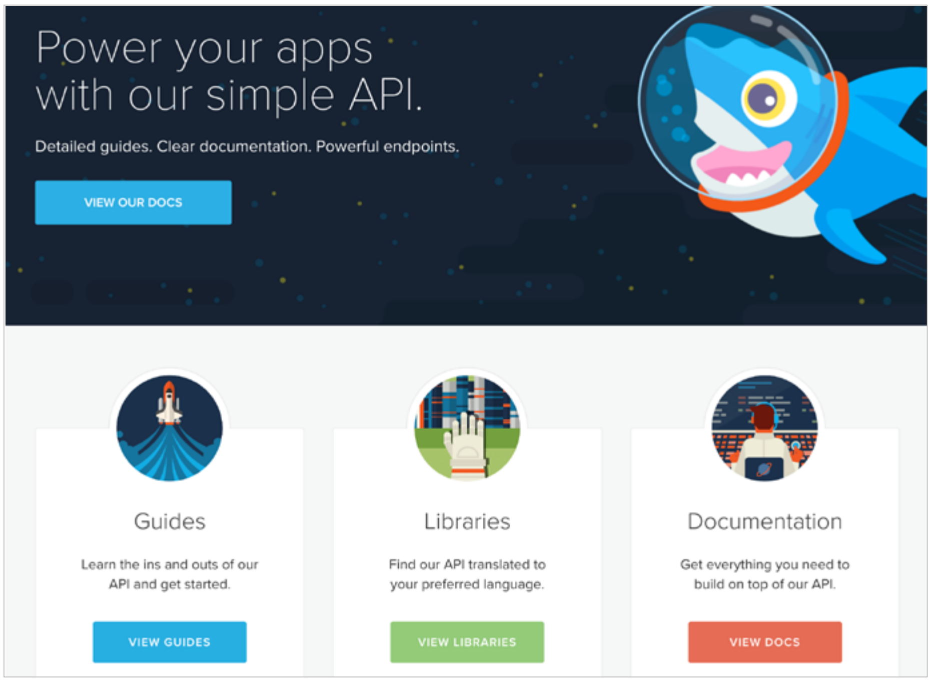
Labels

Developer portals applied the following labels to refer to SDKs:

- SDKs (mentioned on 5 Portals)
- Libraries (4)
- Client Libraries (2)
- Helper Libraries (2)
- Helpers (1)
- Libraries and Frameworks (1)

Some developer portals applied various labels to identify SDKs:

- SDKs / Libraries / Helper Libraries / Helpers (Twilio)
- Client Libraries / Libraries / Libraries and Frameworks (IBM Cloudant)
- SDKs / Helper Libraries (Dwolla)



Power your apps with our simple API.

Detailed guides. Clear documentation. Powerful endpoints.

Learn the ins and outs of our API and get started.

Find our API translated to your preferred language.

Get everything you need to build on top of our API.

Example of SDKs labelled as "Libraries"
(DigitalOcean)

Summary: SDKs on developer portals - best practices

Providing practical examples through code improves DX: it can help developers to learn from existing examples, to onboard easily and save implementation time. The following are key tips on how to include SDKs into a Portal's documentation:

- Be consistent in terminology (choose one word to describe the SDKs)
- Choose a code repository that enables community contributions - in our sample, GitHub was by far the most popular choice
- Include multiple code languages and platforms to target different developer communities
- Add filtering options to make it easier for developers to find the proper SDK

- Make the code overview pages visually attractive: add icons, columns, headlines
- Separate product and community SDKs (Taylor Barnett)
- Measure usage for your SDKs, so you can gain insights about your customer communities (Taylor Barnett)
- Write the SDK documentation first and include sections for troubleshooting and changelog/release notes - this will help you evaluate the developer experience (Taylor Barnett)
- Keep it native: start with the languages that are the most popular for your audience and that the documentation team is familiar with (Taylor Barnett)

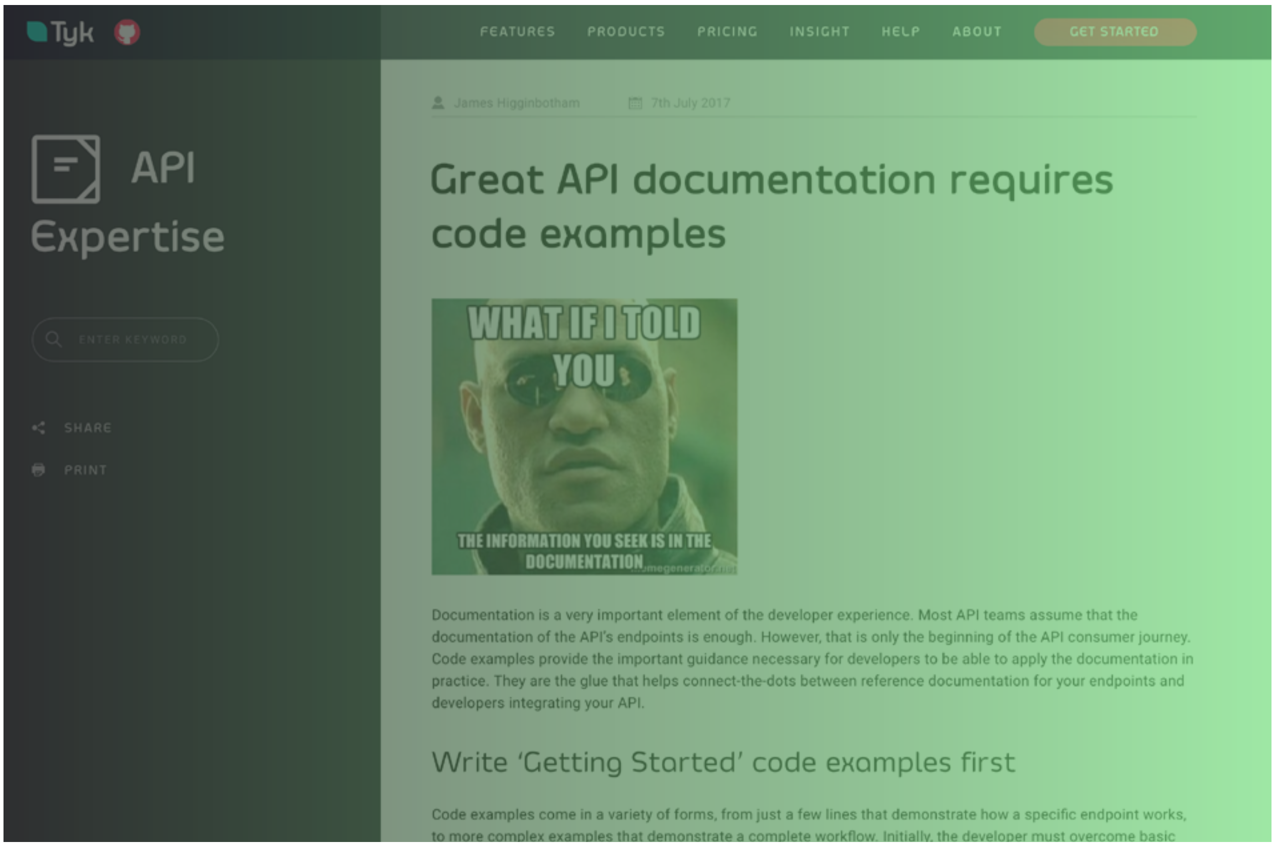
Many thanks to Taylor Barnett from Keen IO for her really insightful talk about SDKs, we leaned heavily on her presentation for this chapter!

If you liked this article, check out our developer portal components series about documentation patterns.

JULY 17, 2017

Great Api Documentation Requires Code Examples

by James Higginbotham



James Higginbotham 7th July 2017

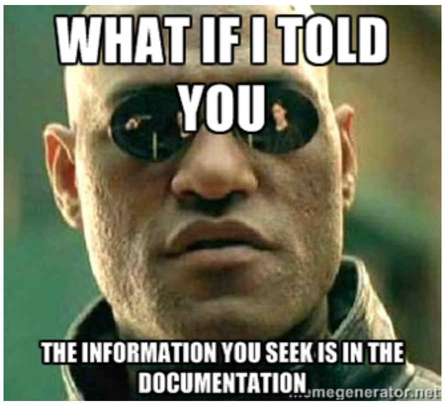
Great API documentation requires code examples

Documentation is a very important element of the developer experience. Most API teams assume that the documentation of the API's endpoints is enough. However, that is only the beginning of the API consumer journey. Code examples provide the important guidance necessary for developers to be able to apply the documentation in practice. They are the glue that helps connect-the-dots between reference documentation for your endpoints and developers integrating your API.

Write 'Getting Started' code examples first

Code examples come in a variety of forms, from just a few lines that demonstrate how a specific endpoint works, to more complex examples that demonstrate a complete workflow. Initially, the developer must overcome basic

https://tyk.io/blog/great-api-documentation-requires-code-examples/



Documentation is a very important element of the developer experience. Most API teams assume that the documentation of the API's endpoints is enough. However, that is only the beginning of the API consumer journey. Code examples provide the important guidance necessary for developers to be able to apply the documentation in practice. They are the glue that helps connect-the-dots between reference documentation for your endpoints and developers integrating your API.

Write 'Getting Started' code examples first

Code examples come in a variety of forms, from just a few lines that demonstrate how a specific endpoint works, to more complex examples that demonstrate a complete workflow. Initially, the developer must overcome basic understanding of your API and how it will help solve their problem. It is important to remember that during this phase, the developer just wants to see something work.
"Time to first Hello World", or TTFHW, is a key metric for determining API complexity. The longer it takes to get a developer to their first "win", the more likely the developer will struggle with your API and perhaps abandon it or build their own solution.
To help developers get started quickly, provide concise examples that remove all need for explicit coding. Look at the following example from Stripe:

```ruby
  require “stripe”
  Stripe.api_key = “sk_test_BQokikJOvBiI2HlWgH4olfQ2”
  Stripe::Token.create(
    :card => {
      :number => “4242424242424242”,
      :exp_month => 6,
      :exp_year => 2016,
      :cvc => “314”
})
```

Notice in this example that there is little code to write – fill-in your API key and the credit card credentials and you are ready to go. Example code that requires that you write lots should be avoided at this stage, as it requires you to learn more about the API before you can try it out. Never require developers to write code to complete an example when first trying out your API – instead, make it easy to get started and see the request work successfully.

Workflow examples

After the developers have had some time to try our your API using some getting started examples, the next step is to begin to demonstrate common use cases and workflows.
Workflow examples focus more on achieving specific outcomes. As a

result, we should use copious inline comments to explain why each step is necessary. Be willing to include hard-coded values for easier reading. Choose variable and method names that make the code easy to read and understand. Below is an example of charging a credit card using Stripe:

```ruby

# Set your secret key: remember to change this to your live secret key in production

# See your keys here: https://dashboard.stripe.com/account/apikeys

Stripe.api_key = “sk_test_BQokikJOvBiI2HlWgH4olfQ2”

# Token is created using Stripe.js or Checkout!

# Get the payment token submitted by the form:

token = params[:stripeToken]

# Create a Customer:

customer = Stripe::Customer.create(

  :email => “paying.user@example.com”,

  :source => token,

)


# Charge the Customer instead of the card:

charge = Stripe::Charge.create(

  :amount => 1000,

  :currency => “usd”,

  :customer => customer.id,

)
```

```
# YOUR CODE: Save the customer ID and other info in a database for later.


# YOUR CODE (LATER): When it's time to charge the customer again, retrieve the customer ID.

charge = Stripe::Charge.create(

  :amount => 1500, # $15.00 this time

  :currency => "usd",

  :customer => customer_id, # Previously stored, then retrieved

)

```
```

It is important to note that while these examples will be more complex than those found from the first milestone, they shouldn't exceed the average screen size. The examples need to be short enough to explain the concepts but not too long that they require considerable time to understand. It is often best to demonstrate scenarios that are easily understood and likely map to your customer needs.

Error case examples

The final step is to help your developers understand how to integrate your API into their production environment. This includes how to catch errors to help developers properly troubleshoot problems, and retry loops when a minor outage occurs. You may also want to demonstrate how to catch and recover from bad data provided by end users. Finally, if you are enforcing rate limiting, then show how to obtain the current rate limits for their account.
```

Where do you include code examples?

Now that you have written some nice code examples to help get developers started, demonstrate common workflows, and how to handle error cases, we need a place to put them. There are a few options you may wish to consider:

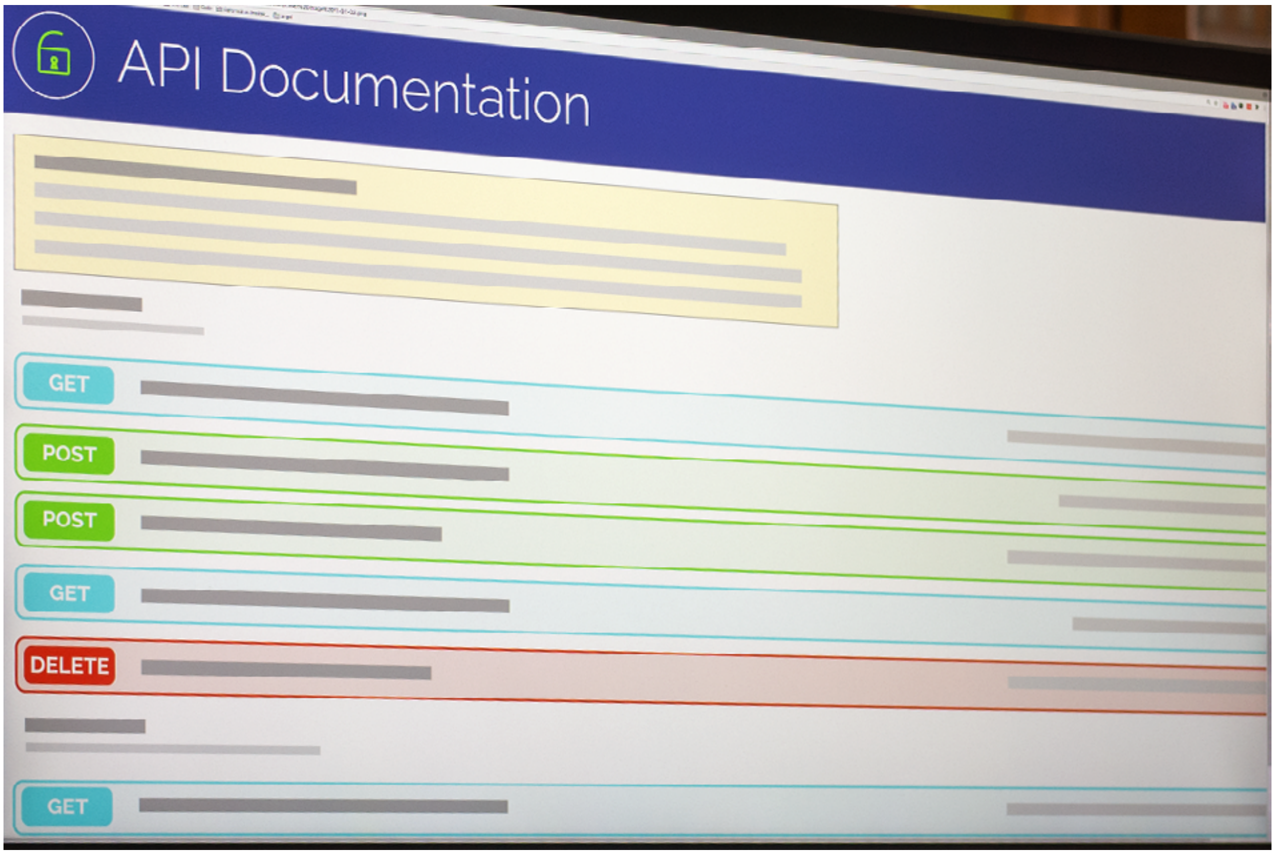
1. Embed the code examples into the description of your OpenAPI definitionUtilize a static site generator such as Jekyll or Hugo to capture your code examples and additional documentation
2. Select a third-party solution such as Readme.io or the documentation feature of your API management layer

No matter the method you choose, sharing code examples that guide developers throughout the integration process will help them be happy and successful – and no one likes a grumpy developer!

MARCH 30, 2017

Free and Open Source API Documentation Tools

by Diána Lakatos



https://pronovix.com/blog/free-and-open-source-api-documentation-tools

We explored free and open source API documentation solutions, and compiled the results of our research in this article.

Introduction

Definitions

An **Application Programming Interface (API)** is a set of clearly defined methods of communication between various software components. Organizations share their APIs so that developers can build applications that use the services of their software.

API documentation describes what services an API offers and how to use those services. Good quality documentation is essential to developer experience, which in turn will impact the adoption and long-term success of an API.

We wrote this article for:

- **API providers**: To provide an overview of free and open source tools for companies that want to share, update or customize their API docs or developer portal.
- **Developer portal builders**: To provide an independent review of existing developer portal solutions that developer teams tasked with building developer portals can use as a reference in discussions with their clients, to make it easier to select the one that best fits their needs.
- **Technical writers**: To create a resource that tech writers can use to select the API documentation infrastructure that fits best with their existing authoring workflows.

Open source API documentation generators

API providers describe their API's functionalities with **specifications and definitions**, like OpenAPI/Swagger, RAML, API Blueprint, I/O Docs or WSDL. **API documentation solutions** convert these definitions into a structured, easy to use API documentation for developers.

API documentation tools are sometimes named after the type of definition they take, e.g. Swagger generates API documentation from Swagger definitions. They also often include the definition in their naming, e.g. RAML 2 HTML.

API documentation generators using the Swagger/OpenAPI specification

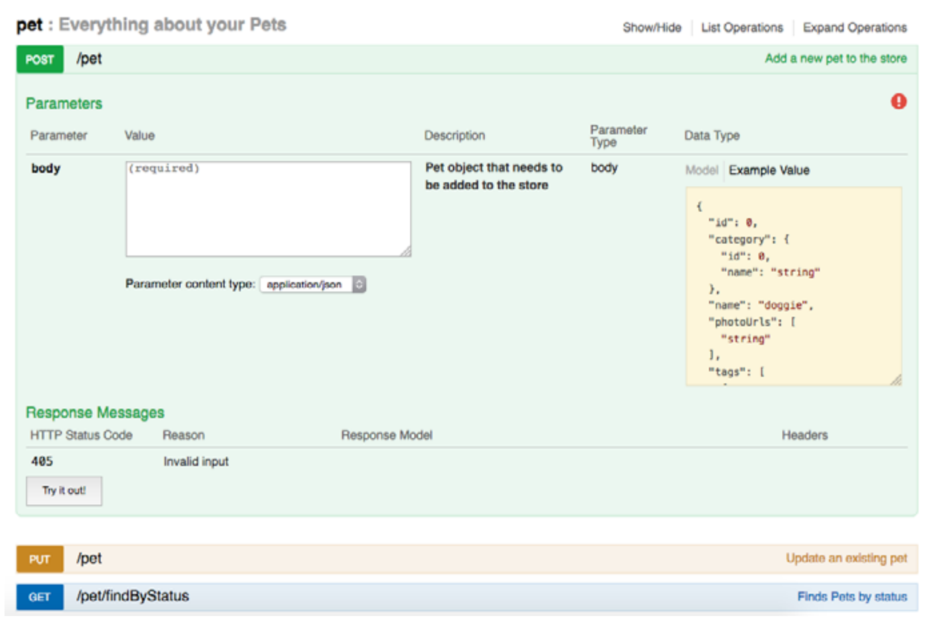
The Swagger specification is a powerful definition format that describes RESTful APIs. It maps all the resources and operations associated with a RESTful interface and makes it easier to develop and consume an API.

Recently the Swagger standard changed its name to Open API, you can find out more about the initiative at the Open API Initiative website. As a leading standard Swagger/OpenAPI has accumulated a large range of API documentation generators that use the specification format.

Swagger

Swagger is a complete framework for describing, producing, consuming, and visualizing RESTful web services.

Use the Swagger ecosystem to create your API documentation: document APIs with JSON using the Swagger spec, and use the Web UI to dynamically convert it into API documentation in a web page. Your API documentation will be displayed through the Swagger UI, which provides a well-structured and good-looking interface.



pet : Everything about your Pets

POST /pet — Add a new pet to the store

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	(required)	Pet object that needs to be added to the store	body	Example Value

Parameter content type: application/json

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
```

Response Messages

HTTP Status Code	Reason	Response Model	Headers
405	Invalid input		

PUT /pet — Update an existing pet

GET /pet/findByStatus — Finds Pets by status

Example of an API documentation displayed with the Swagger UI

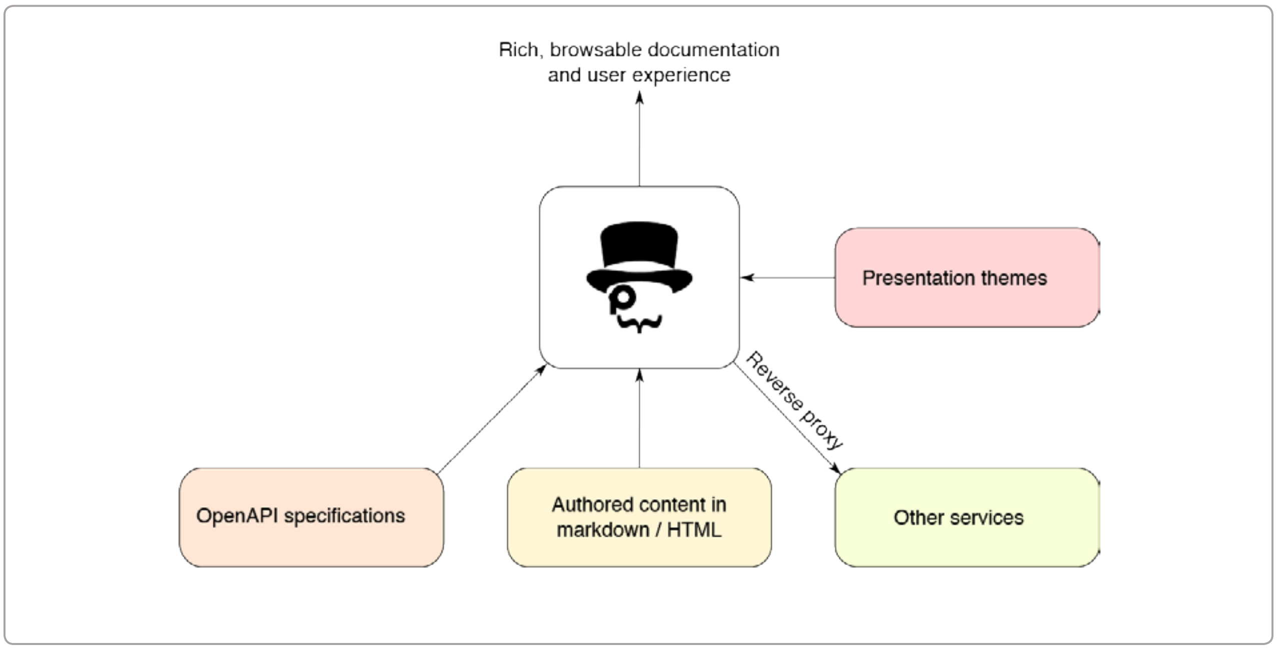
Swagger is free to use, licensed under the Apache 2.0 License. You can find all Swagger-related public tools under the swagger-api GitHub account.

Many open source projects and commercial vendors provide Swagger integrations, so make sure to check out the list of available solutions before building new tooling - there is a big chance you will find an existing solution that fits the needs of your project.

As today's leading API ecosystem, it's also the best documented and supported. Should you decide to document your APIs with Swagger, you can find plenty of resources, tutorials, examples and help online.

DapperDox

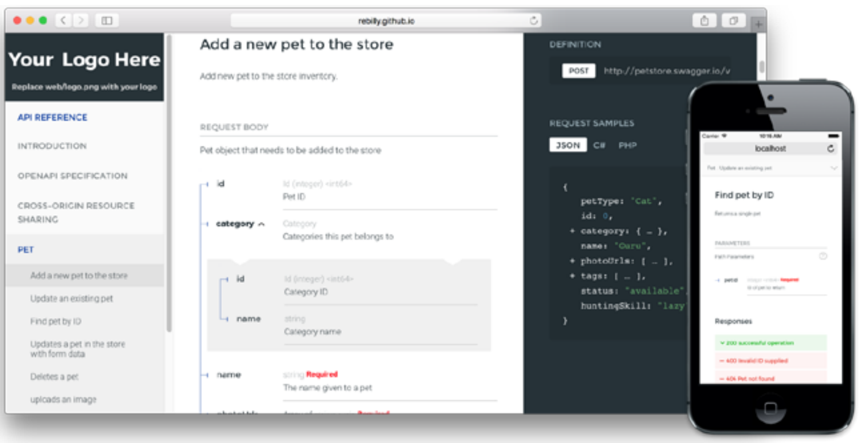
With DapperDox you can author readable guides and have them form part of a cohesive set of documentation along with the API specifications: You can inject relevant documentation into the rendered specification page.


Rich, browsable documentation and user experience
Presentation themes
Reverse proxy
OpenAPI specifications
Authored content in markdown / HTML
Other services

To create your API documentation with DapperDox, point DapperDox at your **OpenAPI/Swagger** specifications, add some documentation in **Markdown** and let DapperDox do the rest.

ReDoc

ReDoc uses the OpenAPI specification and generates a responsive site with a three-panel design. It pulls markdown headings from the OpenAPI description field into the side menu, and supports deep linking.
ReDoc aims to make deployment extremely easy, provides a wide support for OpenAPI objects, and offers interactive documentation for nested objects. You can include code samples via a third-party extension.



Add a new pet to the store

Add new pet to the store inventory.

REQUEST BODY

Pet object that needs to be added to the store

id	Id (integer) <int64> Pet ID
category	Category Categories this pet belongs to
id	Id (integer) <int64> Category ID
name	string Category name
name	string Required The name given to a pet

http://petstore.swagger.io/v

```
{
  petType: "Cat",
  id: 0,
+ category: { … },
  name: "Guru",
+ photoUrls: [ … ],
+ tags: [ … ],
  status: "available",
  huntingSkill: "lazy"
}
```

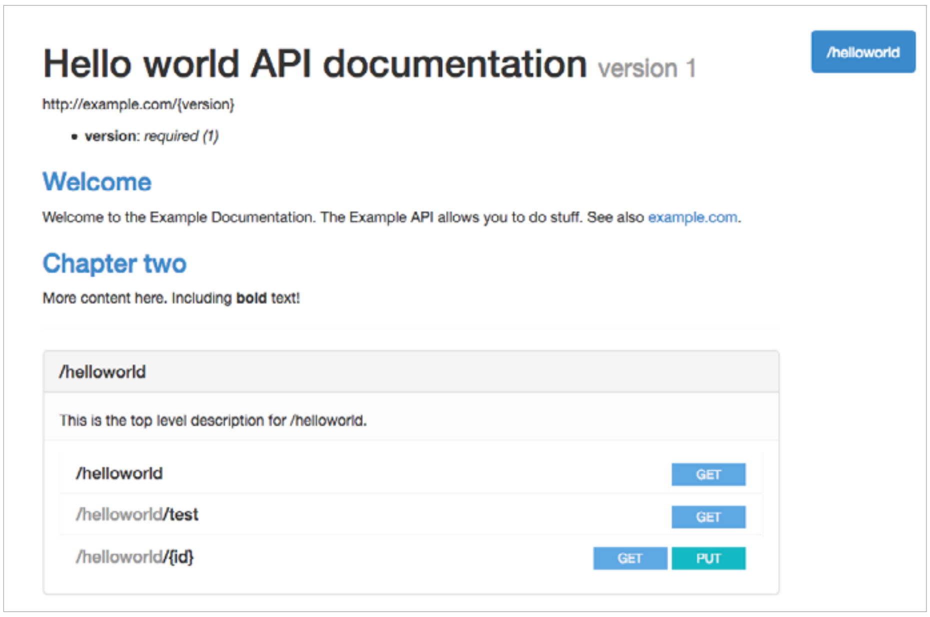
API documentation generators using the RAML specification

RAML (RESTful API Modeling Language) helps you manage the whole API lifecycle from design to sharing.
RAML is built on broadly-used standards such as YAML and JSON, and is language neutral with tools for: Java, Javascript, .Net, PHP, Python, Ruby, etc.

To create your API documentation with RAML, you can choose open source tools like the API Console or RAML 2 HTML . Documentation can be generated quickly and on the fly. With parsers available for many languages you can create your own custom docs and interactive scripts like e.Pages and Spotify.

RAML 2 HTML

RAML 2 HTML is a simple RAML to HTML documentation generator with theme support, written for Node.js.


Hello world API documentation version 1

http://example.com/{version}

- **version**: *required (1)*

Welcome

Welcome to the Example Documentation. The Example API allows you to do stuff. See also example.com.

Chapter two

More content here. Including **bold** text!

/helloworld

This is the top level description for /helloworld.

/helloworld GET

/helloworld/test GET

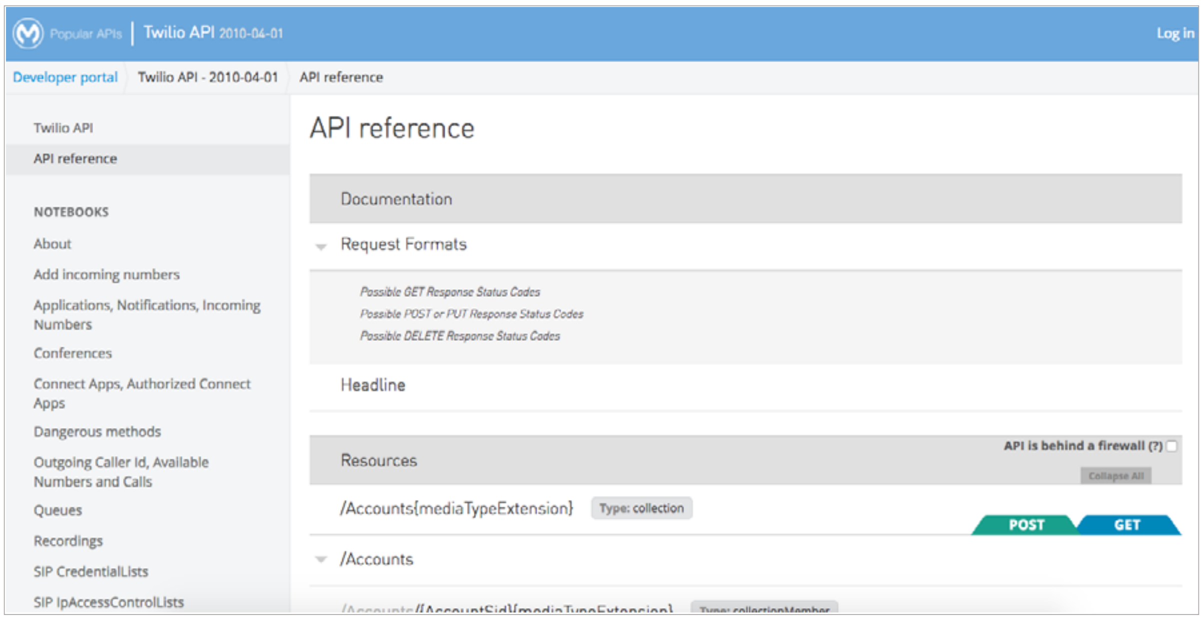
/helloworld/{id} GET PUT

Example of an API documentation displayed with RAML 2 HTML's default theme

RAML 2 HTML ships with a default theme, but you can install more from NPM. For example, to render RAML to Markdown, you can install the raml2html-markdown-theme.

RAML Api Console

Using the RAML API Console you can create HTML documentation from a RAML specification. It allows browsing of API documentation and in-browser testing of API methods.
There are two ways you can include the console: directly, or within an iframe.



Example of an API documentation displayed with the RAML API Console

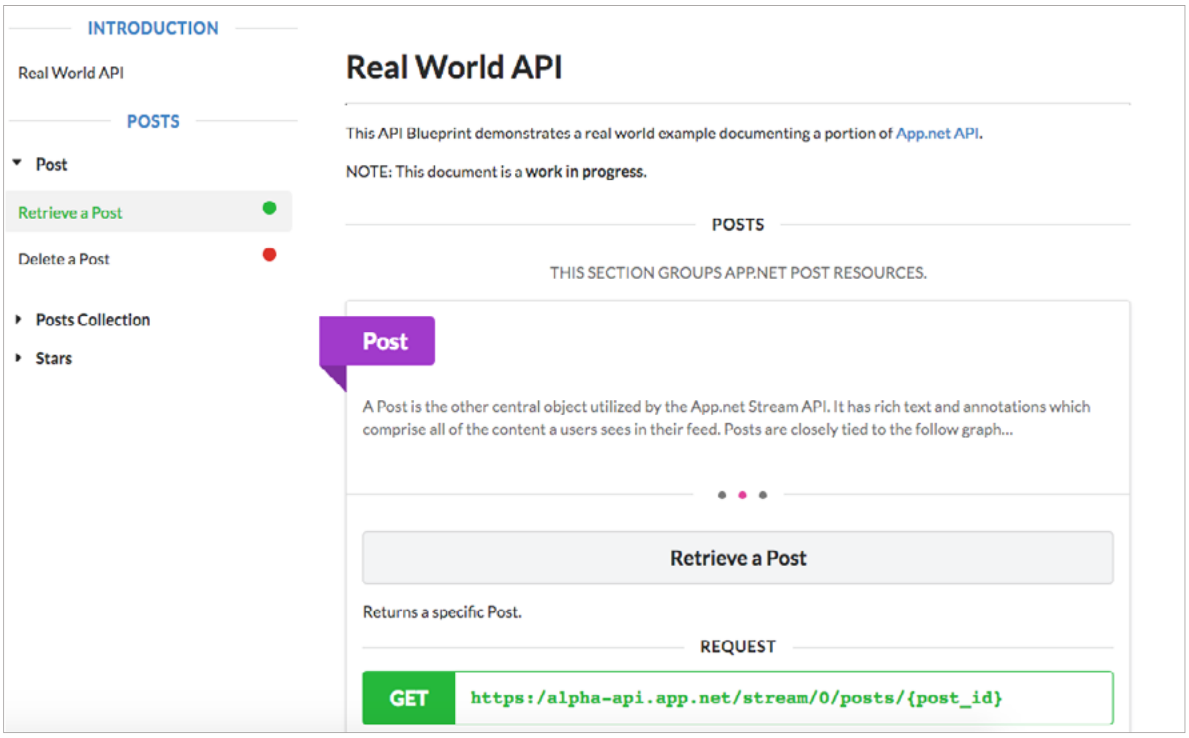
API documentation generators using the API Blueprint specification

API Blueprint is a Markdown-based document format for writing API descriptions and documentation. With API Blueprint you can quickly design and prototype APIs to be created, or document and test already deployed APIs.

Thanks to its broad adoption there is a wide range of tools built for API Blueprint. From various standalone tools such as mock server, documentation and testing tools to full-featured API life-cycle solutions.

Snowboard

Snowboard is an API Blueprint parser and renderer. It offers a colourful default theme illustrating API request types and responses, and can also be used with custom templates.



Real World API

This API Blueprint demonstrates a real world example documenting a portion of App.net API.

NOTE: This document is a **work in progress**.

POSTS

THIS SECTION GROUPS APP.NET POST RESOURCES.

A Post is the other central object utilized by the App.net Stream API. It has rich text and annotations which comprise all of the content a users sees in their feed. Posts are closely tied to the follow graph...

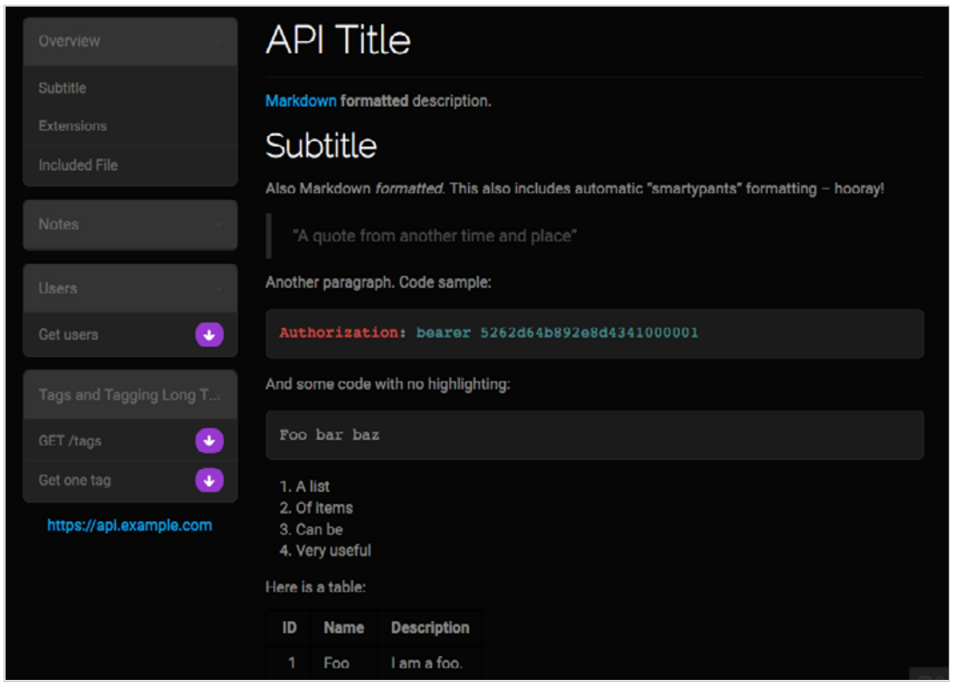
Returns a specific Post.

```
https:/alpha-api.app.net/stream/0/posts/{post_id}
```

Example of an API documentation displayed with Snowboard

Aglio

Aglio renders HTML from API Blueprint files, with support for custom colors, templates and themes.



API Title

Markdown **formatted** description.

Subtitle

Also Markdown *formatted*. This also includes automatic "smartypants" formatting – hooray!

> "A quote from another time and place"

Another paragraph. Code sample:

```
Authorization: bearer 5262d64b892e8d4341000001
```

And some code with no highlighting:

```
Foo bar baz
```

1. A list
2. Of items
3. Can be
4. Very useful

Here is a table:

ID	Name	Description
1	Foo	I am a foo.

Example of an API documentation displayed with Aglio (Cyborg two-column theme)

Other free and open source API documentation generators

Besides the ones detailed above, there are plenty of different open source API documentation generators for different languages and API specifications. Here's a brief summary of the ones we've explored:

- I/O Docs: I/O docs is an API definition format for the TIBCO Mashery network that comes with a live interactive documentation system for RESTful web APIs. By defining APIs at the resource, method and parameter levels in a JSON schema, I/O Docs will generate a JavaScript client interface.
- Slate: Slate helps you create responsive API documentation with a clean, intuitive design. Although it's built in Ruby, when you write docs with Slate, you're just writing Markdown, which makes it simple to edit and understand. By default, your Slate-generated documentation is hosted in a public Github repository, which makes it simple for other developers to make pull requests to your

docs if they find typos or other problems. Of course, if you don't want to use GitHub, you can also host your docs elsewhere.

- Whiteboard: A NodeJS based project started from Slate.
- apiDoc: Inline documentation for RESTful web APIs, that creates a documentation from API annotations in your source code.
- CUUBEZ API Visualizer: Java based API solution to visualize the documentation of RESTful web APIs. This API visualizing framework supports all JAXRS based java REST frameworks and non-JAXRS java based REST frameworks that are currently available in the industry.
- Apidox: XML powered live interactive API documentation and browsing for RESTful APIs.
- Carte: A simple Jekyll based documentation website for APIs. Designed as a boilerplate to build your own documentation, heavily inspired by Swagger and I/O docs.
- Docbox: A responsive website generated from Markdown documentation content. It's dynamically updated with React.

And a free one:

- API Docs: Although not open source, API Docs provides a hosted public API documentation service for OAS (Swagger) and RAML specifications for free. Features like custom domains, themes, and analytics, are available for a nominal cost through the StopLight integration.

General purpose open source documentation tools

Although very handy, API documentation generators are not the only way to render and display your API docs. Many general purpose documentation tools can also get the job done. You could consider using them if you already have one in place, or if you have more documentation tasks than documenting your API alone.
A couple of documentation tools you can check out:

- Dexy: Dexy is a multi-purpose project automation tool with lots of features designed to work with documents. It does the repetitive parts for you, and thus makes it easier to create technical documents. Many developers use it to document APIs, because combined with other open source tools, Dexy is able to run your example code, save the results, fetch data from an API, and post your docs to a blog or a wiki.
- Docco: Docco is a quick-and-dirty documentation generator. It produces an HTML document that displays your comments intermingled with your code.
- Doxygen: Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, IDL, Fortran, VHDL, Tcl, and to some extent D. To document your API, generate an online HTML documentation browser or an offline reference manual, and configure Doxygen to extract the code structure from your source files.

We mentioned these tools to give you an idea of how you can use general documentation tools for API documentation, but there are many more to choose from, if you'd like to follow this approach.

Developer portals

Good API documentation is necessary, but not sufficient for a great developer experience, so it's better to think about the whole experience in terms of a developer portal that will fulfill all developer needs. Besides the API documentation, a developer portal can include guides and tutorials, reference pages, FAQs, forums, other support resources, software development kits, etc. For an overview of all the different types of documentation a good developer portal needs, check our blog post series on developer portal components or receive it as a white paper in

your mailbox by subscribing to our Developer Portal mailing list.
At Pronovix, we work with Drupal, an open source content management system to build a full-featured developer portal, a toolbox for developer relations with integrated API documentation.
Drupal has a couple of modules that you can use to document your APIs, one of which is the API module originally developed to produce the Drupal developer documentation available at api.drupal.org.
It implements a subset of the Doxygen documentation generator specification, with some Drupal-specific additions. If you'd like to publish your API documentation and you plan to extend it into a developer portal, you could give Drupal a try, as it's free, open source, and has extensive documentation both for the core CMS and the API module.
We have done extensive work with Apigee's developer portal that is built in Drupal 7, and we are building a new developer portal in Drupal 8, Drupal's latest release. As API documentation is a key requirement, it will include a custom API documentation generator that can import Swagger/OpenAPI files and that splits the documentation for individual endpoints into separate entities so that you can control access granularly and easily extend your documentation (especially important for partner portals and for organisations that have strong security requirements). Our ultimate goal is to share our developer portal package as an open source Drupal distribution.

Conclusion

As you can see, with some research and hopefully with the help of this article, you have a good chance to find an open source API documentation tool that fits the needs of your project.
Although this article features quite a few solutions, there are many others available or in development, and new ones are popping up continuously.
Please let us know in the comments if you've tried a solution that you'd recommend to others!

Comparison table

	Quick summary	Source (specification)	Live demo
Swagger	Whole ecosystem, lots of integrations Good-looking UI for docs Widely used, many resources available	Swagger/ OpenAPI	Swagger demo
DapperDox	Inject relevant documentation right into the rendered specification page	OpenAPI, Markdown	DapperDox demo
ReDoc	Easy deployment Wide support for OpenAPI objects Interactive, responsive documentation	OpenAPI	ReDoc demo
RAML 2 HTML	Simple RAML to HTML documentation generator theme support	RAML, NodeJSwith	RAML 2 HTML demo
RAML API Console	Browsing of API documentation and in-browser testing of API methods	RAML, NodeJS	RAML API Console demo
Snowboard	API Blueprint renderer	API Blueprint	Snowboard demo
Aglio	API Blueprint renderer with many custom themes	API Blueprint	Aglio demo

I/O Docs	Live interactive API documentation system for I/O Docs specification format	I/O Docs (JSON)	I/O Docs demo
Slate	Clean, intuitive design Write in Markdown Collaboration through GitHub	Markdown (Ruby)	Slate demo
Whiteboard	NodeJS based Slate alternative	NodeJS	Whiteboard demo
apiDoc	Inline documentation for RESTful web APIs	NodeJS	apiDoc demo
CuuBEZ API Visualizer	Visualize the documentation of RESTful web APIs	Java	CuuBEZ API Visualizer demo
Apidox	XML powered live interactive API documentation and browsing for RESTful APIs	XML, PHP	Apidox demo
Carte	A simple Jekyll based documentation website for APIs	Jekyll, YAML	Carte demo
Docbox	A responsive website generated from Markdown documentation content	Markdown	Docbox demo
API Docs	Free, hosted API documentation	OpenAPI, Swagger, RAML	API Docs demo

JUNE 3, 2017

Documenting APIs with Swagger

by Adam Locke

https://www.docslikecode.com/articles/api-docs-with-code/

A Pirate's Life for Me: Documenting APIs with Swagger

Our team starting developing a new API (in C#), which I took as an opportunity to implement Swagger (now the OpenAPI Specification), an open source project used to describe and document RESTful APIs. I wanted to show our developers and support engineers that injecting documentation into the code can reduce response time, mitigate errors, and decrease the point of entry for new hires. To illustrate those gains, I needed to develop a proof of concept.

Why Swagger?

Swagger is open source and includes a UI to display your API documentation, which can be built from source code or manually in JSON. Swashbuckle, a combination of ApiExplorer and Swagger UI, enables Swagger for .NET environments, which was just what we needed.

> Note: This article applies to .NET environments. Swashbuckle uses a different package for .NET Core environments.

Prepare to Swashbuckle

Swashbuckle requires a bit of coding to implement, but using Paket helps to manage .NET dependencies. With Paket, I can add the necessary Swashbuckle NuGet packages to my API project and ensure that they are current. If I need to add more packages, I can install and manage those packages through Paket.After installing Paket, I run the following command to add Swashbuckle as a dependency to my C# project.

```
paket add nuget Swashbuckle.Core project <projectName>
```

Now that Swashbuckle is available to my project, I can add Swashbuckle to the `Startup.cs` file, which is the application startup file for the API. I add each of the following Swashbuckle libraries so that the solution can access the necessary methods.

```
using Swashbuckle.Application;
using Swashbuckle.Swagger.Annotations;
using Swashbuckle.Swagger;
using Swashbuckle.Swagger.XmlComments;
```

Then, I add the following code (see example that follows), much of which is supplied in a Swashbuckle example file. In the `SwaggerGeneratorOptions` class, I specify the options that I want Swashbuckle to enable.

- `schemaFilters` post-modify complex schemas in the generated output. You can modify schemas for a specific member type or across all member types. The `IModelFilter` is now the `ISchemaFilter` ISchemaFilter. We created an `IModelFilter` to fix some of the generated output.
- `operationFilters` specifies options to modify the generated output. Each entry enables a different modification for operation descriptions.

```
namespace LinkInterface
{
  public class Startup
  {
//Enables Swashbuckle and related Swagger options.
    private void generateSwagger(HttpConfiguration config)
    {
      config.EnsureInitialized();
      var swaggerProvider = new SwaggerGenerator(
      config.Services.GetApiExplorer(),
      config.Formatters.JsonFormatter.SerializerSettings,
      new Dictionary<string, Info> {
        version = "v1", title = "My API", description = "Provides an interface
        between our API and third-party services"
      }
      new SwaggerGeneratorOptions(
    //Apply your Swagger options here.
        schemaIdSelector: (type) => type.FriendlyId(true),
    //Implements the SwaggerTitleFilter class, which generates title members
    //in the definitions model of the swagger.json file.
        modelFilters: new List<IModelFilter>(){new SwaggerTitleFilter()},
        conflictingActionsResolver: (apiDescriptions) => apiDescriptions.GetEnumerator().Current,
        schemaFilters: new List<ISchemaFilter>(){new ApplySwaggerSchemaFilterAttributes()},
        operationFilters: new List<IOperationFilter>()
          {
        //Enables XML comments and writes them to the MyAPI.XML file. These comments
        //are included in the generated swagger.json file.
            new ApplyXmlActionComments("MyAPI.XML"),
        //Enables the SwaggerResponse output class, to specify multiple
        //response codes for the API.
            new ApplySwaggerResponseAttributes()
          }
        );

        var swaggerString = JsonConvert.SerializeObject(
          swaggerDoc,
          Formatting.Indented,
          new JsonSerializerSettings
          {
            NullValueHandling = NullValueHandling.Ignore,
            Converters = new[] {new VendorExtensionsConverter()}
          }
        );

    //Writes the swagger.json file to the \output directory so that it can
    //be consumed by statis site generators.
        System.IO.StreamWriter file = new System.IO.StreamWriter("swagger.json");
        file.WriteLine(swaggerString);
        file.Close();
      );
    }
  }
}
```

After enabling these options, I could include code that enables the Swagger UI, but that interface looks a bit outdated. Also, I want to incorporate additional documentation written in Markdown, which the Swagger UI does not support. After reading online forums and posting questions to the Write The Docs channel on Slack, I discovered DapperDox.

Using DapperDox

DapperDox is an open source documentation framework for OpenAPI specifications. Instead of having Swashbuckle publish our API specification in the Swagger UI, I added the following code to the Startup.cs file. This code writes the Swagger specification to a `swagger.json` file.

```
System.IO.StreamWriter file = new System.IO.StreamWriter("swagger.json");
   file.WriteLine(swaggerString);
   file.Close();
```

DapperDox reads this file and displays it in its own UI. I installed DapperDox and pointed it at my `swagger.json` file, and saw nothing but error messages in my command prompt.
Reading through the DapperDox documentation, I discovered that *"When specifying a resource schema object…DapperDox requires that the optional schema object title member is present."* This requirement was problematic, because Swashbuckle does not include a method for adding members to a schema in the generated `swagger.json` file. Additionally, it took some tinkering in the code for me to realize that the missing title member on the `definitions` model is what caused DapperDox to break.

Fixing the output

The Swashbuckle documentation offered little help in this regard, so I turned to one of our developers. After reviewing the code together, my

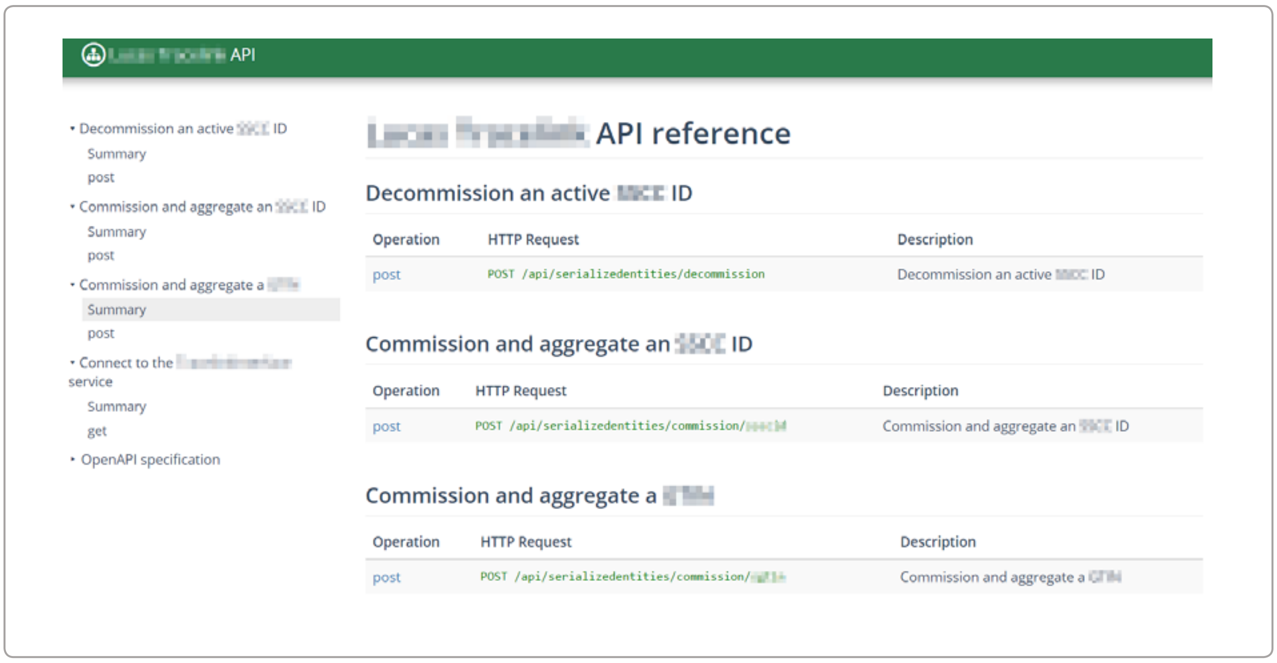
developer counterpart created a `SwaggerTitleFilter` method that adds a `title` member to the `definitions` model in the resulting `swagger.json` file. The title member displays in the generated documentation as a link to the referenced object, creating a hyperlink between the two objects.
The following code implements an `IModelFilter` that causes Swashbuckle to generate a title member for any schema. The `SwaggerTitleFilter` was referenced in the previous code sample I compiled the code and Swashbuckle generated an updated

```
namespace MyAPI.Swagger
{
    public class SwaggerTitleFilter : IModelFilter
    {
        public void Apply(Schema schema,  ModelFilterContext mfc)
        {
            schema.vendorExtensions.Add("title", mfc.SystemType.Name);
        }
    }
}
```

`swagger.json` file. With the `title` member added to the `swagger.json` output, I pointed DapperDox at the directory containing my `swagger.json` file.
I opened a browser and entered `http://localhost:3123` , which is

```
.\dapperdox -spec-dir=C:\Bitbucket\APIproject\source
```

where DapperDox runs by default, and it worked! DapperDox displayed my `swagger.json` file and created interactive documentation that clearly displays the requests, responses, and query parameters for the API. I demoed the output for a few developers and support engineers, and they were over the moon.


[redacted] API

- Decommission an active [redacted] ID
  - Summary
  - post
- Commission and aggregate an [redacted] ID
  - Summary
  - post
- Commission and aggregate a [redacted]
  - Summary
  - post
- Connect to the [redacted] service
  - Summary
  - get
- OpenAPI specification

[redacted] API reference

Decommission an active [redacted] ID

Operation	HTTP Request	Description
post	`POST /api/serializedentities/decommission`	Decommission an active [redacted] ID

Commission and aggregate an [redacted] ID

Operation	HTTP Request	Description
post	`POST /api/serializedentities/commission/`[redacted]	Commission and aggregate an [redacted] ID

Commission and aggregate a [redacted]

Operation	HTTP Request	Description
post	`POST /api/serializedentities/commission/`[redacted]	Commission and aggregate a [redacted]

Next steps

With this framework in place, we can extend Swashbuckle to future APIs and use DapperDox to host the `swagger.json` file for each. The resulting output lives with the code, and provides documentation that developers and support engineers can access locally by running a single command.

To add documentation beyond just the generated JSON output, DapperDox works incredibly well. I can author short tutorials that describe how to integrate our API with third-party services, which developers can easily review and modify through pull requests. As the API grows, we can add a README file that describes enhancements, modifications, and new integration points. Non-API documentation will live in an `\assets` directory, which DapperDox includes at build time.

Each time that the code builds, the `swagger.json` file updates with the most current information. Developers and support engineers just run the `.\dapperdox` command and specify the directory where the `swagger.json` file lives. As the code changes, so does the documentation, so technical debt approaches zero.

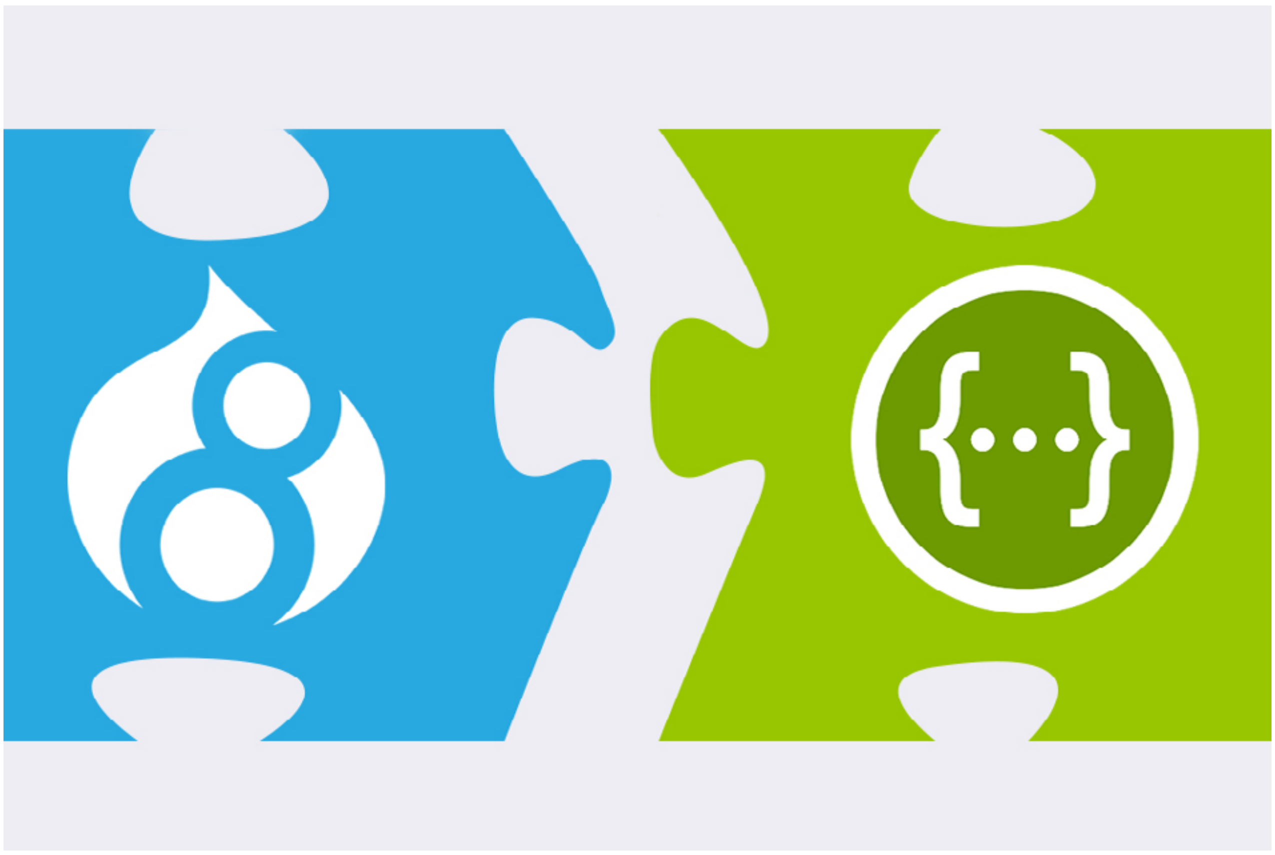
Lessons learned

Static site generators are all the rage, and for good reason. Providing a lightweight framework that can be deployed quickly is a huge asset when documenting APIs, especially external-facing documentation. Numerous options are available, but DapperDox felt like the right fit for our needs. The pain of determining why DapperDox was broken and the additional coding required to fix the problem was worth the effort, and we are poised to integrate this process into the next set of APIs that our team develops.

APRIL 20, 2017

Documenting web APIs with the Swagger / OpenAPI Specification in Drupal

by Kitti Radovics



https://pronovix.com/blog/documenting-web-apis-swagger-openapi-specification-drupal

As part of our work to make Drupal 8 the leading CMS for developer portals, we are implementing a mechanism to import the OpenAPI (formerly known as Swagger) specification format. This is a crucial feature not only for dedicated developer portals, but also for all Drupal sites that are exposing an API. Now that it has become much easier to create a RESTful API service in Drupal 8, the next step is to make it more straightforward to create its API reference documentation. That is why we think our work will be useful for site builders, and not just for technical writers and API product owners.
Swagger is a REST API documentation tool, it provides a specification language for describing the APIs and also a set of support tools. One of those tools is Swagger UI, which generates an appealing and readable layout for API endpoints and methods. The Drupal community is considering using the Swagger specification to document Drupal 8 core web services, and Swagger tool adaptations can be found in several contributed modules. In this article we will introduce some of these modules and explain how we want to go beyond the shallow integration that most of them have done, to take full advantage of Drupal's architecture. But before diving into the technical details, we want to list the features that we seek in the ultimate API reference CMS.

6 features that make the difference between a good and a great API reference system

The following are 6 technical developer portal features that customers have requested from us in the past 2 years working with Apigee's developer portal. They provide functionality that go beyond what most API management platforms provide.

This feature list is based on our investigations of existing developer sites, our practical experience from creating developer portals and architecture workshops we've held.

1. Storing multiple API versions: versioning

As API services can have multiple supported versions (e.g. v1, v2) in parallel, a Developer Portal should provide a clear user experience that gives visitors the option to choose which version they would like to read about (but still default to the latest supported stable one).

2. Track changes in the documentation of each API version: revisioning

Most developer portal platforms rebuild the documentation as part of an automated build process, Drupal's revisioning system allows editors and site maintainers to make and track changes in specific versions of the API documentation. While this is less important for the developers that use the documentation, it is an editorial feature that can be useful for technical writers and site owners.

3. Possibility to attach conceptual content to the API reference

API references are very technical and factual. Sometimes developers need more verbose documentation that provides a longer explanation of the context an API operates in. That is why, several of our customers have asked us to add conceptual documentation to their imported content - about domain language, underlying architecture, data models, or code samples that surround an API call.

4. Access control for individual API methods

In order to restrict the visibility of certain API methods (e.g. for partner APIs), a Developer Portal must allow site maintainers to set granular access permissions/restrictions for specific versions, endpoints or other parts of the documentation.

5. Trying out API calls on the Developer Portal's UI

Integrating a system with an API service can be accelerated by a *Try it out* feature, that helps developers to decide which API endpoint with what parameters to use in order to get the expected result.

6. Importing reference documentation from a version control system

Recently technical writers have also started using the online collaboration and versioning tools that developers work with. Documentation is now often committed into code repositories, especially when developers contribute to the writing process. One key problem with this approach is that, apart from the API reference documentation where most teams use the Swagger specification, there is no obvious standard to store the content and layout of documentation. We've been working with markdown topics, and manifest files to allow technical writers to store conceptual documentation and their navigation structure (what we used to organize in a book hierarchy in Drupal) separate from the API specification. This way all the documentation can be stored in the version control system (e.g. a GitHub or GitLab repository).

Existing Swagger modules in Drupal

The Swagger API documentation toolset covers the entire publishing process: building (Swagger Codegen), documenting, and visualizing (Swagger Editor, Swagger UI). Existing Drupal modules typically focus on the building and visualization steps.

As usual Drupal.org has some modules that seem to be abandoned, but there are two Swagger docs related modules that have been maintained in the past year. One is the *Swagger UI Field Formatter* module, it renders fields with valid Swagger source file using the Swagger UI tool. The other is *Swagger php* module (sandbox only), it can generate JSON formatted Swagger code based on annotations and it can render that code using the Swagger UI.

Both of these modules use the Swagger UI project to generate a human readable output from the specification. Swagger UI only needs a valid source file to generate the output and the 'Try it out' section (for sending requests to the endpoints); it is useful if you only need to publish the content, but it has its limitations.

The problem we see with this solution and most other API documentation tools is that API providers usually need access control, search, and conceptual documentation for their API descriptions. These functions demand a different approach.

Don't just show it, integrate it!

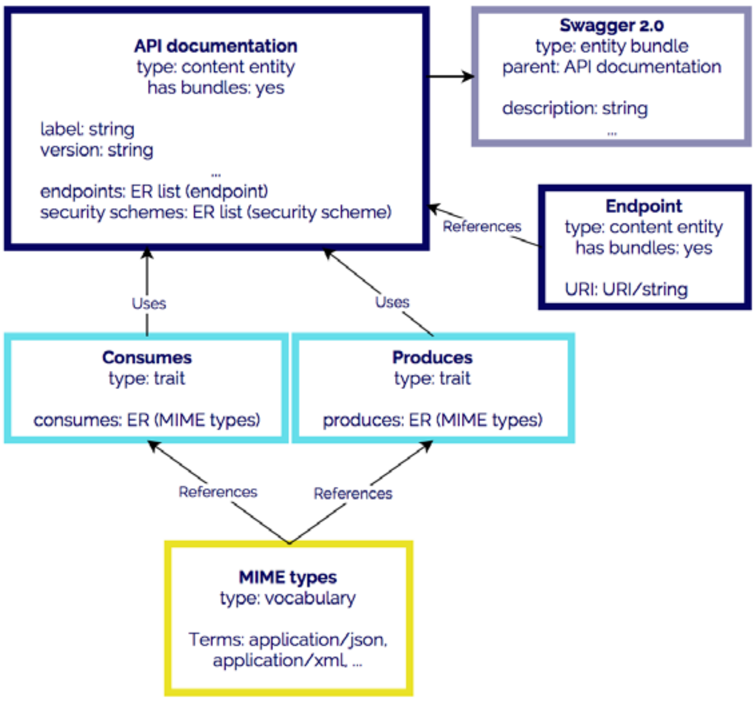
After careful evaluation, we came to the conclusion that the currently existing Swagger tools can't support the 6 advanced API documentation features our customers request from us. To make the API documentations fieldable, revisionable and to be able to apply custom access control on all components of them in Drupal, a more robust API integration is needed. No open-source module is available for Drupal 8 that does this, so we decided to make it a key contribution we would work on with our team.
Since there are other specification languages (such as RAML or I/O Docs) that are widely used and that store similar information as the OpenAPI format, we take great care to make sure that our architecture would be extendable and reusable.

Mapping Swagger objects into Drupal data types

To get a flexible system that can be extended and altered with proper Drupal 8 solutions, we designed custom Drupal 8 entities and field types for every piece of a Swagger source file. The first step was to observe the individual Swagger specification elements and to decide the most suitable Drupal 8 data types for storing them.

We just finished the planning phase of the entity architecture, the overall structure won't change much, but there might be some small changes during the implementation period.

The below image describes a small part of the planned entity architecture. We defined a vendor independent API documentation as a content entity (basically the root entity) which might have bundles, providing the ability to extend the base system with vendor specific formats other than Swagger (e.g. I/O Docs or Raml). Based on this concept, each specification language format makes a new bundle with vendor specific fields. By default the Swagger 2.0 specification format bundle is provided. Each piece of content in a bundle represents a different API version, so multiple versions (e.g. v1, v2) can be made available in parallel on the Developer Portal (feature1).



All of the documentation components are tied to the properties or references of an API documentation entity. For example API endpoints form another content entity type, which can get referenced from the root (API documentation) entity. Moreover, as we are planning to use fieldable entities, any additional information can be attached to them easily (feature 3).

Thanks to the OOP nature of Drupal 8, reusable properties and methods can be attached to entity classes through traits. For example, base field definitions of the `consumes` and the `produces` specification properties can be defined in traits and used in multiple entities, as they can be attached to the API documentation entity or to an API method/operation (overriding the default global settings of these properties). The `consumes` and the `produces` properties in the Swagger source are technically MIME types (such as `application/json`), so they can be collected into a vocabulary as taxonomy terms.

Thinking in traits will also enable us to extend the default API specification with custom properties (e.g. extend Swagger specification objects with 'x-' properties). Code snippets could for example be included for different programming languages (such as Java, PHP, Python), these might help the readers to understand the API reference.

With the above architecture we can map specification languages into a Drupal entity system where basic revisioning is supported by default (feature 2). Although custom access control can also be added to any type of entity and its fields (feature 4), it's not as powerful as Drupal's node access control system. There is already a Drupal core issue that tries to expand the node access grants system to a generic entity grants system, and we are trying to contribute to it while working on our Drupal Developer Portal.

Importing the data into the Drupal system

For the import process we leverage Drupal 8's Migrate API to import any type of API specification formats to our custom entities and to store them in a unified way. Source files can be either uploaded in the UI or imported from a Github repository (feature 6) through a documentation importer that we are building to support editorial workflows that rely on code repositories and that automatically publish to Drupal as part of a continuous integration process.

If you are interested in our GitHub importer and Migrate processing solution, join our Developer Portal mailing list to receive notifications about blog posts on the subject.

Why Drupal?

We chose Drupal 8 as the framework for our Developer Portal, because it already had a large number of features that our customers need. With a 10 year long history in Drupal, we are obviously somewhat biased, but even if we disregard our prior expertise, we believe that Drupal is one of the best CMSs for building documentation and developer portals.

That is why we decided to extend the existing solutions, with a sufficiently complex system that would enable us to address all the needs our customers have. Some of our code is still in stealth mode, the developer portal market is a relatively small niche, and we need to make sure we can find a sustainable way to give back to the Drupal community. That is why in parallel to our development, we are working on a new business model for our distribution to make sure we will be able to continue sharing our work with the wider community. We are committed to the open source community and credo, but we want to prevent some of the failures we have seen with previous Drupal distributions, more about that in a later article...

MARCH 23, 2017

Web APIs in Drupal: Success Takes More than an Endpoint

by Dezső Biczó

https://pronovix.com/blog/web-apis-drupal-success-takes-more-endpoint

Introduction

Web APIs are not just useful when making headless sites in Drupal: large Drupal sites often hold valuable information that could also be useful outside the site's context. Media companies might want to expose historical media content, community sites could show data about their community activities, e-commerce sites tend to open an API for their affiliates and partners.

While it is possible to use Drupal 7 and Drupal 8 as an API backend, a lot of functionalities that describe a mature API service do not come out of the box. In this article we will explain what key concepts you have to keep in mind when designing an API service, why they are important and how APIgee Edge can make it easier to build a full-featured API webservice in Drupal successfully.

Designing APIs: the API first strategy

In a large part of the software development industry, API first thinking is replacing a user interface design approach. **API first design is about planning and documenting your API before it would be implemented in code.** If you set up your backend service this way, you can use it with different clients regardless of the way they were implemented. API first strategy allows you to diversify user interfaces: UI developers can work without knowing how your backend service works.

Building good backend services is not easy, there are plenty of pitfalls on the road and most of them only reveal themselves during development. Your responsibilities as a service provider grow with the number of clients: maintaining the security of your services (especially if you are providing paid services),

handling compatibility problems between client apps and different app versions,

ensuring that your services are able to handle unexpected loads.

You can't handle all of these tasks without monitoring the services. Especially for monetization, monitoring is crucial.

Features to keep in mind for building good API backend services

Security

Security is one of the most important trust signals of a mature API. A multi-layered protection system should be able to hide your non-public services from the public, handle the authorization processes, and protect the original resources from attackers.

Compatibility

Compatibility issues are the nightmares of service providers: versioning your APIs is your first step to harmony.

Scalability

Successful services have to handle an enormous number of requests every second and your services have to scale with the number of your new clients. Sometimes moving your backend to better hardware does not help, because the root of the problem is in the initial architectural decisions or implementations.

Monitoring

You need exact analytics about the usage of your API: it is indispensable for monetization purposes, plus you could use analytics data to improve your service and to understand your users' behavior.

Documentation

Good documentation is an essential part of the API service, as this is the first line of support for developers trying to understand and learn how to use the API. Developer portals often have different kinds and levels of supporting material from getting started pages to various guides, case studies, playbooks, and tutorials.

Monetization

You will need an authorization and monitoring system to efficiently track and bill customers for using your services. Exposing different resources of your APIs individually or grouped, and setting up usage limits based on these "API products" can be a time consuming task.

Companies specialized in API management solutions

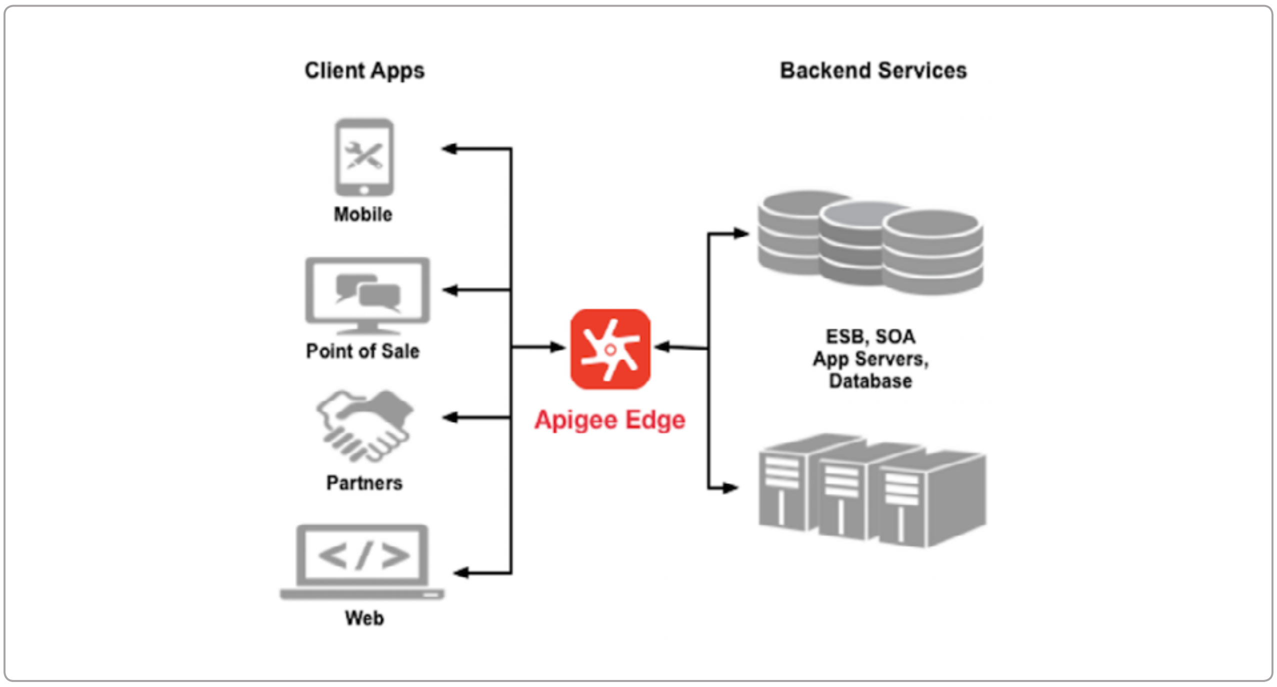
You can choose from many API management technologies to build an API service, but each technology stack has its own limitations. Some companies have specialized to help you solve (a part of) the problems that might occur. Our non-exhaustive list of such companies as an example (company descriptions are from Crunchbase):

- 3scale's API management platform empowers API providers to easily package, distribute, manage and monetize APIs.
- Apiphany provides API management and delivery solutions that enable organizations to leverage the mobile, social and app economy.
- Layer 7 Technologies provides security and management products for API-driven integrations spanning the extended hybrid enterprise.

- MuleSoft provides integration software for connecting applications, data and devices. MuleSoft's software platform enables organizations to build application networks using APIs.
- Mashery is a TIBCO company providing API management services that enable companies to leverage web services as a distribution channel.
- StrikeIron offers a cloud-based data quality suite offering web-based infrastructure to deliver business data to internet-connected systems.
- Apigee is the leading provider of API technology and services for enterprises and developers.

The rest of this article will focus on Apigee (recently acquired by Google). Disclaimer: Pronovix is an Apigee partner, so we are somewhat biased. However, even if we wouldn't be partners, we believe they are probably the best API management service provider for Drupal projects. They are not only a market leader in the space, they have also invested in a Drupal integration: Apigee Edge.

Apigee Edge: a Drupal integration for API services



Built in JAVA, Apigee Edge is able to replace or enhance complicated parts of your services. **API proxies will protect your services from direct customer access (as they guard the backend code), and add the above mentioned 6 key features to your APIs.** Apigee Edge manages these features in a specific way.

Policies

Apigee Edge enables you to control the behavior of your APIs (without writing a single line of code) via policies. A policy is similar to a module that implements a specific, limited function that is part of the proxy request/response flow. You can add various types of management features to an API by using policies.

Traffic management policies

With cache policies you can set up traffic quotas and concurrent rate limits on your API proxies.

Mediation policies

Mediation policies let you do custom validation and send back custom messages and alerts to your clients, independently from your backend services. Moreover, you do not need to implement separate xml serialization in your services to accept requests or send responses in XML, because the JSON to XML and XML to JSON policies are capable to do automatic conversions between these formats.

Security policies

Security policies give access control management to your APIs with different types of authorization methods and protection features.

Extension policies

If you haven't found an existing policy for a special task, you can implement your own policy in Java, Javascript or Python with the help of the Extension policies, which also contain policies for external logging and statistics collecting.

Developer portals

A great developer experience is crucial for API adoption. Apigee Edge has a developer portal solution that is built in Drupal 7 with API documentation, forums and blog posts that come out of the box. API developer portals done well are your power tools to help the adoption of your API and build strong communities. Apigee's dev portals could be hosted either on cloud or on-premises with Apigee Edge for Private Cloud.

We hope this introduction gave you some insight into building high-performance API web services.

Disclaimer: When we specialised Pronovix in API documentation and developer portals we started a partnership with Apigee, and we do extensive work customising the Apigee developer portal.

JANUARY 25, 2017

7 Trust Signals That Help an API Succeed

by Kristof Van Tomme

https://pronovix.com/blog/7-trust-signals-help-api-succeed-developer-portal-strategy-part-1

PRONOVIX
Developer portals

Developer portals are important for your API's adoption and support. They are also a trust signal: a well designed and actively maintained developer portal shows that an organization is investing in its APIs. It helps convince developers that they can rely on them.

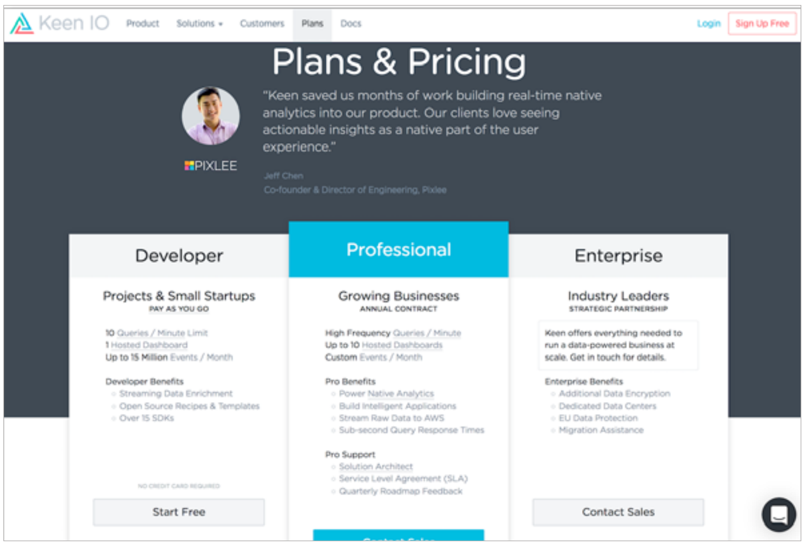
This matters: many developers have in their career dealt with the fallout of a deprecated or suddenly discontinued API, especially more experienced developers will be cautious when introducing dependencies. API trust signals are therefore crucial when you run an API program that primarily targets developers outside of your business, but they can also play a role for internal APIs in large organizations where business unit politics can result in information asymmetries.

In this article I'll zoom in on 7 trust signals that I think are important, all of them - except maybe for nr. 3, API quality - can be asserted through an API's developer portal.

1. Business model

In 2014, Linkedin changed their terms of service. Overnight the majority of CRM projects found themselves shut out of Linkedin's API program. Linkedin had decided to limit API access so that only Salesforce and Microsoft Dynamics would be able to use it to augment their CRM products. This was a disaster for several smaller CRM solutions that had made their integration with Linkedin a key strategic differentiator.

Because of stories like this, developers have become more careful about the APIs they invest their time in. It is extremely important to be upfront about the business model of your API. If your API is free, you need to explain why and how you will keep on supporting it. If you have a paying API you need to make it clear that your plans are sustainable, and that you won't suddenly change the conditions for your customers and partners.



Plans & Pricing

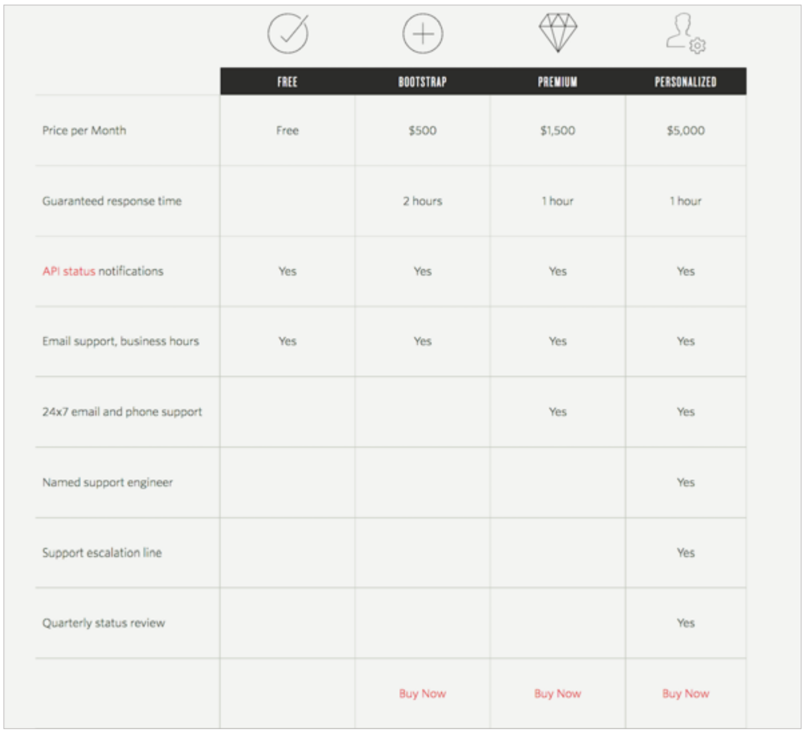
"Keen saved us months of work building real-time native analytics into our product. Our clients love seeing actionable insights as a native part of the user experience."

PIXLEE

Jeff Chen
Co-founder & Director of Engineering, Pixlee

Developer	Professional	Enterprise
Projects & Small Startups PAY AS YOU GO	Growing Businesses ANNUAL CONTRACT	Industry Leaders STRATEGIC PARTNERSHIP
10 Queries / Minute Limit 1 Hosted Dashboard Up to 15 Million Events / Month	High Frequency Queries / Minute Up to 10 Hosted Dashboards Custom Events / Month	Keen offers everything needed to run a data-powered business at scale. Get in touch for details.
Developer Benefits ◦ Streaming Data Enrichment ◦ Open Source Recipes & Templates ◦ Over 15 SDKs	Pro Benefits ◦ Power Native Analytics ◦ Build Intelligent Applications ◦ Stream Raw Data to AWS ◦ Sub-second Query Response Times	Enterprise Benefits ◦ Additional Data Encryption ◦ Dedicated Data Centers ◦ EU Data Protection ◦ Migration Assistance
	Pro Support ◦ Solution Architect ◦ Service Level Agreement (SLA) ◦ Quarterly Roadmap Feedback	
NO CREDIT CARD REQUIRED Start Free		Contact Sales

Keen IO's Business Models (screenshot January 2017)

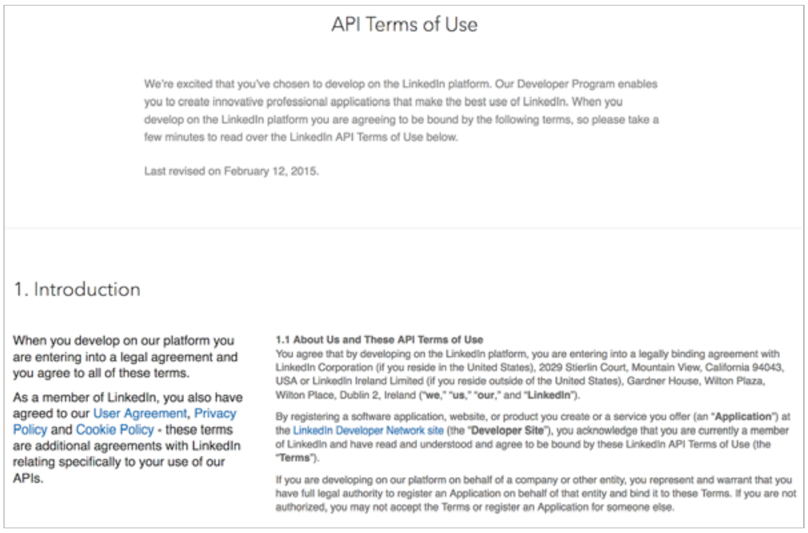


	FREE	BOOTSTRAP	PREMIUM	PERSONALIZED
Price per Month	Free	$500	$1,500	$5,000
Guaranteed response time		2 hours	1 hour	1 hour
API status notifications	Yes	Yes	Yes	Yes
Email support, business hours	Yes	Yes	Yes	Yes
24x7 email and phone support			Yes	Yes
Named support engineer				Yes
Support escalation line				Yes
Quarterly status review				Yes
		Buy Now	Buy Now	Buy Now

Twilio's personalized support models

2. Partner policy

Related to point 1, it is important to have a clear partner policy. Successful APIs allow organizations to turn their services into a platform on top of which other businesses can innovate. In their book "Platform Revolution" the authors describe why platforms need to absorb popular features originally developed by their partner ecosystem back into their core product. If they don't do so, they risk being replaced by a more popular partner who could capture the market with a better default core product. In the book, the authors, also describe that this needs to be done carefully, to make it clear that partners will be able to profit for at least some time, before the platform absorbs those features. If it is your ambition to build a large sustainable ecosystem around your business to make your product more robust and innovative, it is important to consider a platform strategy as part of your your partner policy.



API Terms of Use

We're excited that you've chosen to develop on the LinkedIn platform. Our Developer Program enables you to create innovative professional applications that make the best use of LinkedIn. When you develop on the LinkedIn platform you are agreeing to be bound by the following terms, so please take a few minutes to read over the LinkedIn API Terms of Use below.

Last revised on February 12, 2015.

1. Introduction

When you develop on our platform you are entering into a legal agreement and you agree to all of these terms.

As a member of LinkedIn, you also have agreed to our User Agreement, Privacy Policy and Cookie Policy - these terms are additional agreements with LinkedIn relating specifically to your use of our APIs.

1.1 About Us and These API Terms of Use
You agree that by developing on the LinkedIn platform, you are entering into a legally binding agreement with LinkedIn Corporation (if you reside in the United States), 2029 Stierlin Court, Mountain View, California 94043, USA or LinkedIn Ireland Limited (if you reside outside of the United States), Gardner House, Wilton Plaza, Wilton Place, Dublin 2, Ireland ("**we**," "**us**," "**our**," and "**LinkedIn**").

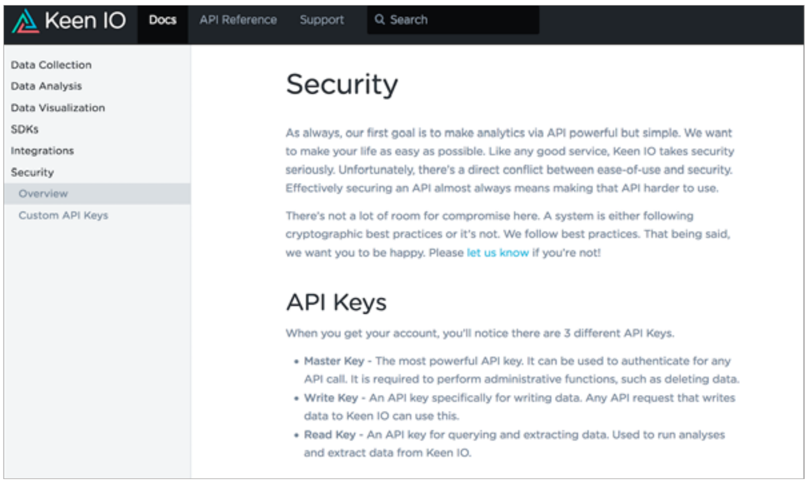
By registering a software application, website, or product you create or a service you offer (an "**Application**") at the LinkedIn Developer Network site (the "**Developer Site**"), you acknowledge that you are currently a member of LinkedIn and have read and understood and agree to be bound by these LinkedIn API Terms of Use (the "**Terms**").

If you are developing on our platform on behalf of a company or other entity, you represent and warrant that you have full legal authority to register an Application on behalf of that entity and bind it to these Terms. If you are not authorized, you may not accept the Terms or register an Application for someone else.

A clear terms of use for an API prevents misunderstandings and abuse. But caution is needed, if the terms are too strict or avoid a clear commitment this might create suspicion and undermine trust.

3. API quality

The quality of your API will of course be one of the most important trust signals for developers once they start working on their integration. A badly designed API will not only reduce the developer experience, it will also raise doubts about your API team, their resources, and commitment. Joshua Tauberer wrote a blog post that lists out a number of qualities that can improve your API on his blog. Another great resource is 5 Features of a Good API Architecture, a talk Rob Allen gave at OSCON.



Security

As always, our first goal is to make analytics via API powerful but simple. We want to make your life as easy as possible. Like any good service, Keen IO takes security seriously. Unfortunately, there's a direct conflict between ease-of-use and security. Effectively securing an API almost always means making that API harder to use.

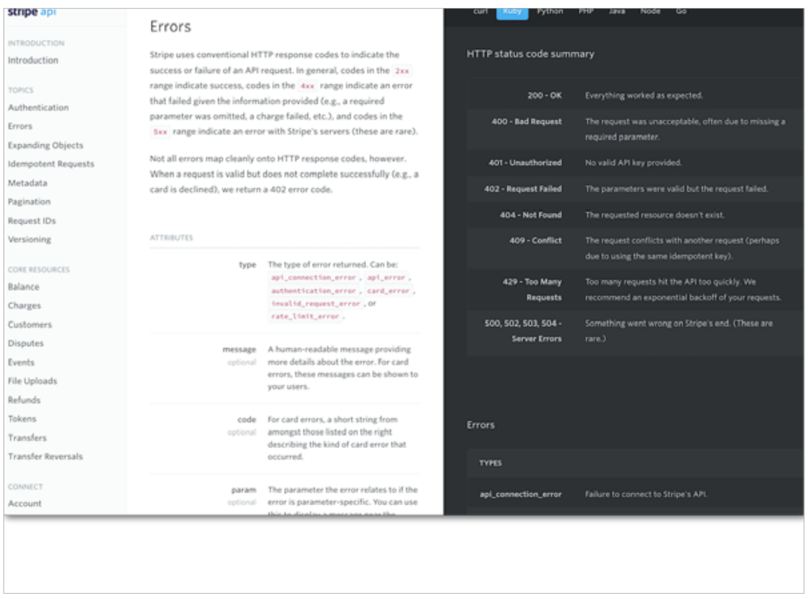
There's not a lot of room for compromise here. A system is either following cryptographic best practices or it's not. We follow best practices. That being said, we want you to be happy. Please let us know if you're not!

API Keys

When you get your account, you'll notice there are 3 different API Keys.

- **Master Key** - The most powerful API key. It can be used to authenticate for any API call. It is required to perform administrative functions, such as deleting data.
- **Write Key** - An API key specifically for writing data. Any API request that writes data to Keen IO can use this.
- **Read Key** - An API key for querying and extracting data. Used to run analyses and extract data from Keen IO.

"A good API is secure" (Rob Allen) - Example of a security section (Keen IO docs)



Errors

Stripe uses conventional HTTP response codes to indicate the success or failure of an API request. In general, codes in the `2xx` range indicate success, codes in the `4xx` range indicate an error that failed given the information provided (e.g., a required parameter was omitted, a charge failed, etc.), and codes in the `5xx` range indicate an error with Stripe's servers (these are rare).

Not all errors map cleanly onto HTTP response codes, however. When a request is valid but does not complete successfully (e.g., a card is declined), we return a 402 error code.

ATTRIBUTES

type	The type of error returned. Can be: `api_connection_error`, `api_error`, `authentication_error`, `card_error`, `invalid_request_error`, or `rate_limit_error`.
message optional	A human-readable message providing more details about the error. For card errors, these messages can be shown to your users.
code optional	For card errors, a short string from amongst those listed on the right describing the kind of card error that occurred.
param optional	The parameter the error relates to if the error is parameter-specific. You can use

HTTP status code summary

200 - OK	Everything worked as expected.
400 - Bad Request	The request was unacceptable, often due to missing a required parameter.
401 - Unauthorized	No valid API key provided.
402 - Request Failed	The parameters were valid but the request failed.
404 - Not Found	The requested resource doesn't exist.
409 - Conflict	The request conflicts with another request (perhaps due to using the same idempotent key).
429 - Too Many Requests	Too many requests hit the API too quickly. We recommend an exponential backoff of your requests.
500, 502, 503, 504 - Server Errors	Something went wrong on Stripe's end. (These are rare.)

Errors

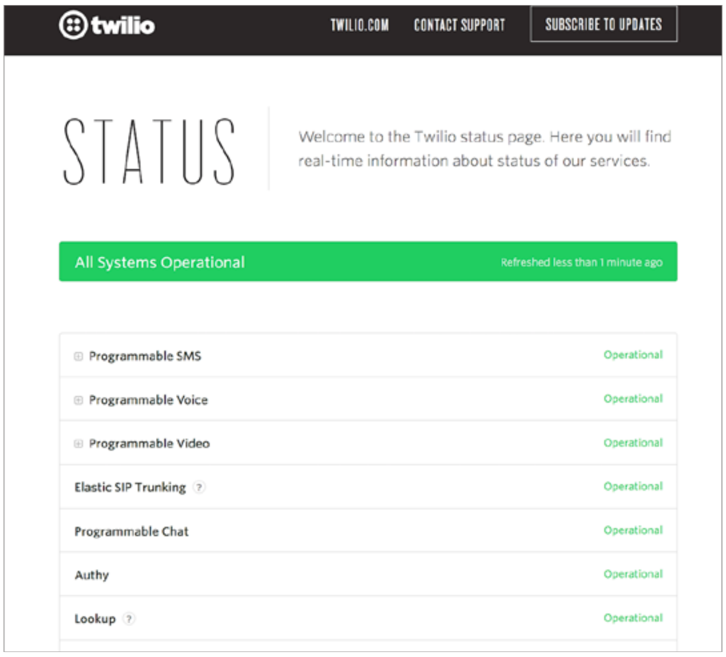
TYPES	
api_connection_error	Failure to connect to Stripe's API.

Example of listing and describing error messages to speed up the developer's job (Stripe)

4. API uptime status

Another way to build trust is to provide a page on your portal where developers can check the current API status of the system they're working with.

Bonus points if you include a sign up possibility to receive updates and a diary of past incidents. Twilio, Dropbox and Vimeo use Atlassian's StatusPage product to do so.



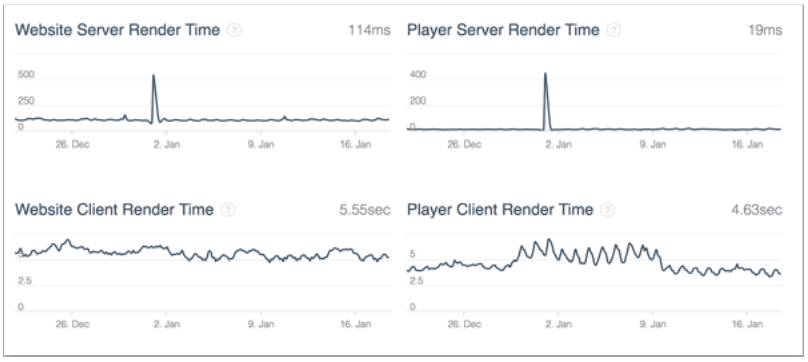
STATUS

Welcome to the Twilio status page. Here you will find real-time information about status of our services.

All Systems Operational — Refreshed less than 1 minute ago

Programmable SMS	Operational
Programmable Voice	Operational
Programmable Video	Operational
Elastic SIP Trunking	Operational
Programmable Chat	Operational
Authy	Operational
Lookup	Operational

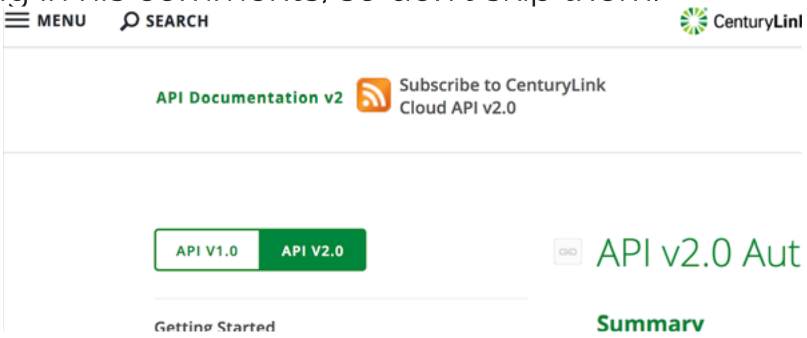
Including a general API status overview helps assert the quality of an API product (Twilio)



API status system metrics (Vimeo)

5. Versioning policy

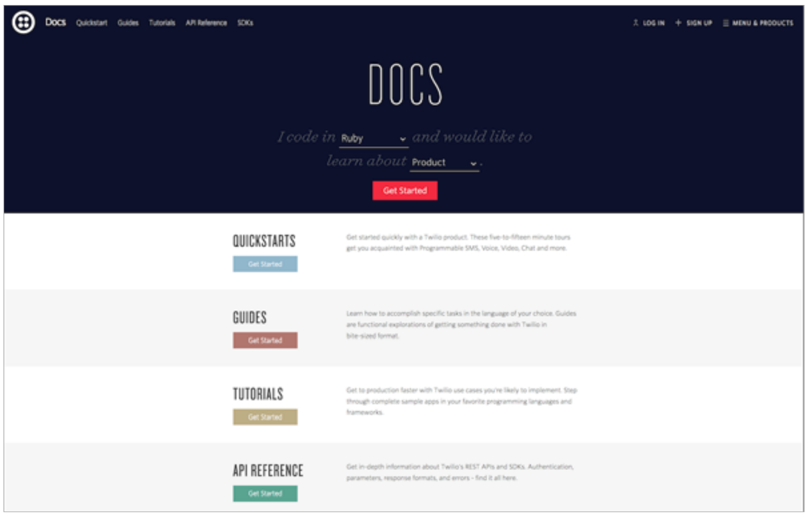
The long term stability of your API will depend on a proper versioning strategy. Having a versioning policy in place from the start will show that you are planning for the future and that there will be further investments in your API. Most web APIs nowadays use URL versioning, but there are arguments against this approach. To learn more about versioning options, read Troy Hunt's blogpost on the subject, he also has a discussion about URL versioning in his comments, so don't skip them.



Example of a docs page with buttons to switch API versions (CenturyLink)

6. Documentation

Even when developers see the importance of documentation and/or like writing docs, they often don't get enough time to do so. The results are unmistakable, according to Stackoverflow's 2016 survey, "poor documentation" was the 2nd most important challenge that 34,7% of developers faced at work just 0.2% after "unrealistic expectations". Documentation also functions as a quality signal that shows the level of investment you have made in the developer experience of your API. Do you have reference documentation for your API? Is it interactive? Do you use one of the REST API documentation standards? Did you create a thesaurus of the domain language, to explain terms that developers, new to your industry, might not be familiar with? Do you have tutorials and getting started documentation? A lot of API teams mistakenly believe that reference documentation is the only type of documentation an API needs. To learn more about the different documentation components check out Kathleen's blogpost series about documentation patterns on developer portals or subscribe to our newsletter to get our white paper about developer portal components:



DOCS

I code in Ruby and would like to learn about Product.

Get Started

QUICKSTARTS

Get Started

Get started quickly with a Twilio product. These five-to-fifteen minute tours get you acquainted with Programmable SMS, Voice, Video, Chat and more.

GUIDES

Get Started

Learn how to accomplish specific tasks in the language of your choice. Guides are functional explorations of getting something done with Twilio in bite-sized format.

TUTORIALS

Get Started

Get to production faster with Twilio use cases you're likely to implement. Step through complete sample apps in your favorite programming languages and frameworks.

API REFERENCE

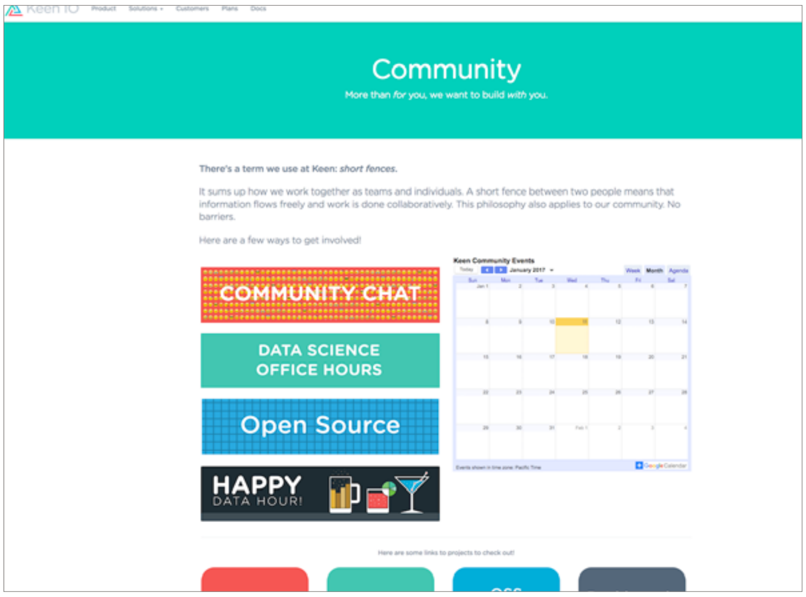
Get Started

Get in-depth information about Twilio's REST APIs and SDKs. Authentication, parameters, response formats, and errors - find it all here.

Twilio's documentation overview page, including Quickstarts, Guides, Tutorials, API reference and SDKs sections

7. Developer portal production quality

Last but not least, don't underestimate the importance of your developer portal's production quality. The design and completeness of your portal will give an impression about the trustworthiness of your API. Developers might not consciously think about this, but a good developer portal can be a signal that you are not cutting corners, and that you are committed to your API program. Regular updates to your portal, through blog posts, event postings, and documentation updates show that you are (still) investing in your APIs.



Community

More than *for* you, we want to build *with* you.

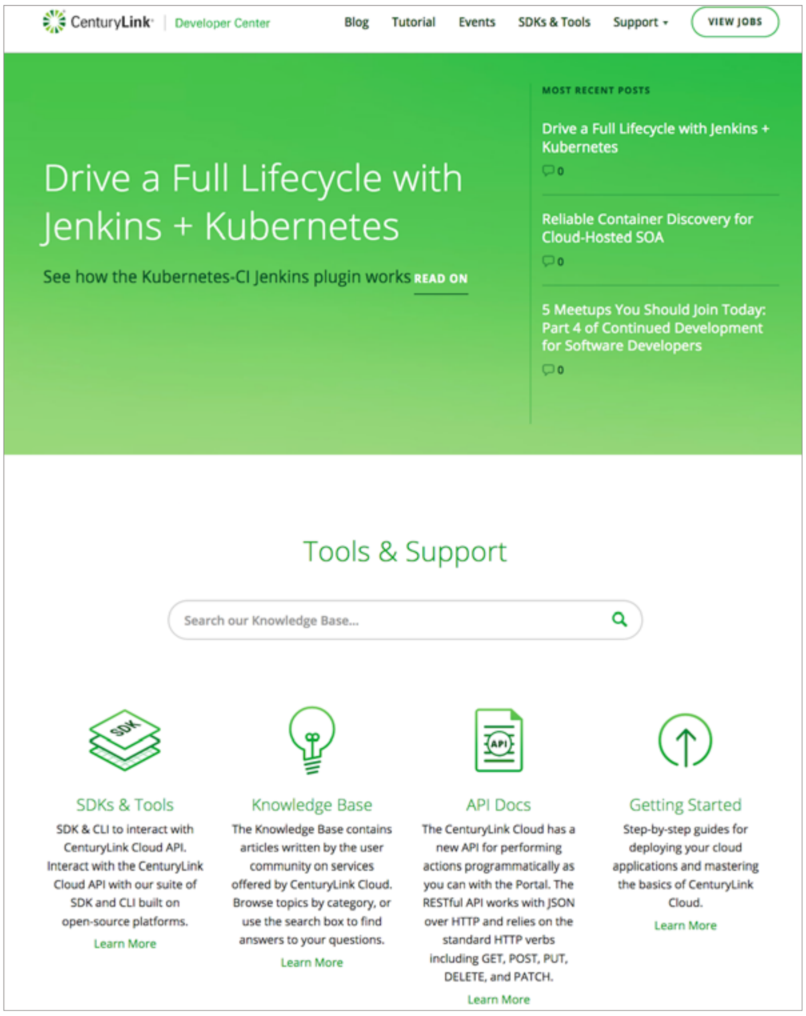
There's a term we use at Keen: *short fences.*

It sums up how we work together as teams and individuals. A short fence between two people means that information flows freely and work is done collaboratively. This philosophy also applies to our community. No barriers.

Here are a few ways to get involved!

Here are some links to projects to check out!

Keen IO's community portal provides developers with several possibilities to communicate and meet



Drive a Full Lifecycle with Jenkins + Kubernetes

See how the Kubernetes-CI Jenkins plugin works READ ON

MOST RECENT POSTS

Drive a Full Lifecycle with Jenkins + Kubernetes
0

Reliable Container Discovery for Cloud-Hosted SOA
0

5 Meetups You Should Join Today: Part 4 of Continued Development for Software Developers
0

Tools & Support

Search our Knowledge Base...

SDKs & Tools

SDK & CLI to interact with CenturyLink Cloud API. Interact with the CenturyLink Cloud API with our suite of SDK and CLI built on open-source platforms.

Learn More

Knowledge Base

The Knowledge Base contains articles written by the user community on services offered by CenturyLink Cloud. Browse topics by category, or use the search box to find answers to your questions.

Learn More

API Docs

The CenturyLink Cloud has a new API for performing actions programmatically as you can with the Portal. The RESTful API works with JSON over HTTP and relies on the standard HTTP verbs including GET, POST, PUT, DELETE, and PATCH.

Learn More

Getting Started

Step-by-step guides for deploying your cloud applications and mastering the basics of CenturyLink Cloud.

Learn More

CenturyLink's overview page for developers includes links to documentation sections and to their blog

APRIL 6, 2017

The 8 Stakeholders of Developer Portals

by Kristof Van Tomme

https://pronovix.com/blog/8-stakeholders-developer-portals-developer-portal-strategy-part-2

PRONOVIX
Developer portals

Different stakeholders interact with a developer portal throughout an API's lifecycle. In this article I'll list 8 stakeholders and explain what they need to do their jobs.

1. API developers

Developer teams that build APIs often also end up designing its developer portals. This is often done almost as an afterthought without properly considering all the stakeholders that will need to interact with a developer portal.

Developers typically care most about the ability to deploy documentation automatically as part of the development process, and might forget that other, less technically people, will also need to interact with the site.

From a continuous integration perspective it very attractive to build a static site as part of the deployment infrastructure. But from the moment that portal stakeholders will demand slightly more complex requirements (like access restriction for your docs or conditional text for code samples), your API developers will spend a lot of time, over and over again, building custom solutions for a problem space they don't fully understand. Company developers will learn how to balance the needs of all the stakeholders with a developer portal that is usable and that serves your business needs, but only after going through a time-consuming learning process that can introduce a lot of technical debt.

To speed up the procedure, we believe it is better to **work with a dedicated team member, or** - shameless plug - **a developer portal specialist like Pronovix**, especially when your API team doesn't have sufficient web and documentation tooling experience.

2. API consumers (developers of the API client and end-consumers)

APIs have two types of customers:

- the developers that build on top of an API,
- the actual API end-consumers that use the service that embeds the API.

Both types of customers need to **get value** out of your API product. But to get a chance to offer that value as a technology company, **developer experience is of primary importance**: **developers** often decide what API they will use, so they act as a decision gate: if you fail to offer them a good experience, they will often not use your API, regardless of its downstream value for end-consumers. **End-consumers** are typically one step removed from the decision process and do not directly influence API selection. They might even end up paying a premium because their interests are not perfectly aligned with the interests of developers.

This agency problem explains why it is dangerous for technology companies to ignore the developer experience of their developer portals.

3. Product owners

APIs should be treated as products, not as projects. Projects are too ephemeral to provide the continuity API consumers need. Product owners are the stakeholders that **manage an API product**.
Ideally, product owners can use developer portals to:

- communicate and get feedback about the product road map,
- gather new ideas for product features,
- ideate new features,
- get feedback about the popularity and performance of current features (e.g. via voting or usage statistics),
- build a relationship with the user community and test new ideas,

e.g. via forums that can provide information about users.

4. Marketing

Apigee, our partner, published a white paper about developers hating marketing. Yes, developers prefer to see code instead of marketing copy, but it is also important to keep in mind that **your developer portal contributes to your marketing objectives**.

Marketeers need a developer portal to:

- perform well on search engines, so that new leads will find your API,
- have a section where decision makers can learn about the benefits and features of your API,
- be an effective conversion tool, that helps to qualify leads,
- measure the effectiveness of the API marketing program.

We believe that it is manipulative marketing that developers really hate. Marketing that tries to directly influence emotions with little substance or evidence about the actual benefits of a product. Like any other human, developers sometimes need help to make decisions. If you have a great product, and if your marketing content is factual, developers might even welcome that marketing and help spread it. But since the content needs to be evidence based, this type of marketing starts to look a lot like well executed documentation, with adaptive content that opens with claims that are backed by layers of increasing detail.

5. Sales

Recent innovations in lead generation tools and marketing automation have blurred the lines between marketing and sales. This trend goes hand in hand with an increasing number of automated processes that are triggered when a potential customer interacts with marketing assets. The developer portal, as the digital gateway to your API, plays an important role in facilitating these new hybrid processes.

A company's business model influences the types of sales functions you need in your developer portal. Developers act as both implementers and decision makers and will play an intermediary role between your business and your API end consumers: a developer portal, therefore, often will have business development, partner recruitment and sales functions.

6. Developer evangelists

As Christian Heilmann explains in his book on Developer evangelism:

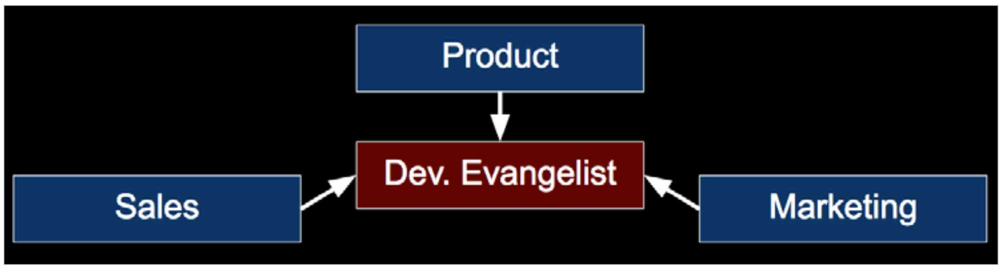
> *"Developer Evangelism is a new kind of role in IT companies. A developer evangelist is a spokesperson, mediator and translator between a company and both its technical staff and outside developers".*

Developer evangelists (or developer advocates) help a company to bridge the gap between its developer customers (developers of the API client) and its sales, marketing, and product development departments. This new role is indispensable indeed: developers require more authenticity and responsiveness than the sales and marketing departments traditionally exhibit. Developer evangelists perform a range of jobs to help you grow your API. In a discussion at CLSx London, we identified a number of jobs a developer evangelist performs:

- community manager
- writer
- engineer
- teacher
- speaker
- event manager

The actual roles that an individual developer evangelist will take on depend on the size of your API team (think unicorn magician on a collision course with burn-out). It is probably healthier to split up tasks between several specialists.

One of my key learnings from DevRelCon 2015 was that that developer evangelists typically will report to a product, sales, or marketing department. And that the type of department they report to influences what jobs and what specific tasks they prioritise.



Developer evangelists balance three interests.

7. Support

Maybe the most important role of a developer portal is **self-service support**. Without proper documentation, companies will spend an enormous amount of time and money on workshops and trainings.
A portal could opt for various support resources:

- **Staffed support options**: e.g. a FAQ page, knowledge base page, or support pages (like a contact form or a live chat window),
- **Peer-to-peer support**: e.g. community sections, forums, and third-party community pages.

The formula for combining several support options depends on your primary personas, your business model and the maturity of your API community.

Investing in a support team is expensive, but the results of their interactions with various user groups are indispensable. The support team:

- provides **feedback** on the current documentation,
- adds **new value** to the current documentation in specific problem areas.

In **"FAQs, Forums and Other Support Resources"** we analyzed the characteristics of support options and looked at how they involve users to develop information about the problem areas in an API's use.

8. Documentarian

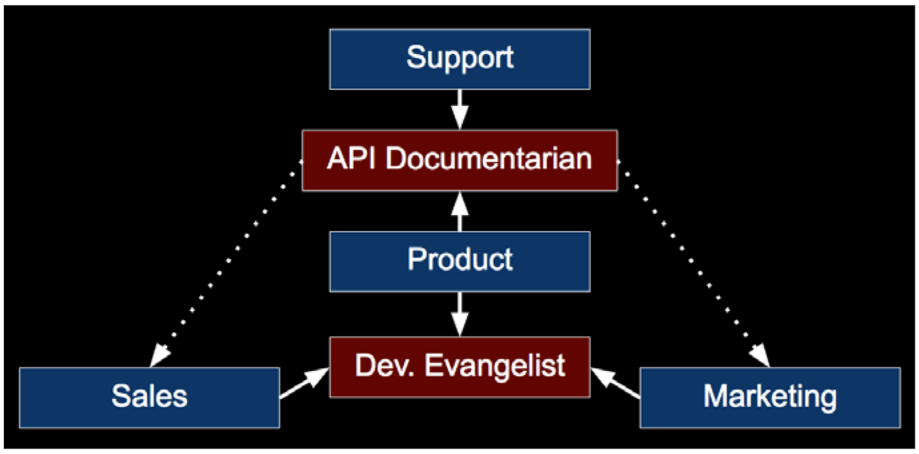
Documentarians, or technical writers, write product specific content, like tutorials, guides, API reference, and onboarding information for an API.

While they might have a technical background, they are often not practicing developers. In other words: they need a portal that allows them to manipulate content without necessarily having to work with the command line or to consult with a developer (we already mentioned the flipside of static developer portals, where changing requirements always comes down to developer work).

Documentarians need a portal setup that:

- allows them to stage content and efficiently test the documentation,
- can publish the documentation format their team has chosen (e.g. Markdown, DITA, Restructured text, Docbook, or Asciidoc),
- integrates with the authoring tools that they use,
- provides them with an accessible publishing tool chain that is easy to operate,

- provides an efficient writing environment that doesn't require elaborate initiation processes to start writing, so that it becomes easy for people to contribute documentation, even if they are not full time involved in a specific project (e.g. I have heard stories about writing environments that require an elaborate synchronisation process before each writing session),
- provides tools that make it easier to ensure quality.



Ideally, the support team co-operates with the documentarians and product developers to optimise the documentation on a developer portal.

MAY 10, 2017

Upstream Developer Experience: a Role for Developer Portals in Enterprise API Design

by Kristof Van Tomme

https://pronovix.com/blog/upstream-developer-experience-role-developer-portals-enterprise-api-design-developer-portal

Most of the time when we talk about developer experience, we mean downstream DX, the experience of developers that implement APIs. But what about the developers that create APIs?

In a previous article we wrote about the 8 stakeholders of developer portals, we argued that while the developers that use APIs are important, we shouldn't forget about the experience of other stakeholders of a developer portal. In this article I'll explore the experience of one of these audiences - the API developers - and explain what upstream DX is, when it matters, and how you can use a developer portal to improve it.

Gamifying digital transformation initiatives

Most organizations never consciously address the experience of the developers that create APIs, upstream DX. This is only logical, upstream DX doesn't always need attention: a team that is dedicated to APIs will overcome friction on its own terms. In large enterprise organizations, however, where API development might be a secondary priority of a team, upstream DX can make the difference between a failed and a successful digital transformation initiative.

A few months ago while I was talking with an enterprise architect about the developer portal he was planning, I realized how different his requirements were from the more traditional outward facing API initiatives that we build most of our portals for. In his company the API initiative was a change project. He needed to convince people about the importance of APIs, explain how it is best to create them, and set out an evaluation process that could help his company to measure the progress in different business units.

We started talking about a workflow tool that would make it easy for employees from all over the organization to submit potential API resources before the actual APIs are built. Developers could then be guided through a set of consecutive steps that help them to develop quality APIs. I explained how a dashboard with an overview of the submitted resources, with a rating that indicates the progress status along a series of workflow steps, would then allow managers to monitor and compare progress between business units.

I believe that such a workflow tool, combined with a well defined process, can reduce the friction of a change initiative. Instead of trying to do everything at once, the most valuable resources can be turned into APIs through a step-by-step process with clear and tangible tasks. Instead of a dedicated API team, the API initiative can become a distributed process that uses feedback from your developer community. A conversation between API creators and consumers prioritises the development of APIs for the most important resources. Gamification (e.g. a dashboard with a 5-star rating), transparency and simplification can be used to increase the likelihood of success of a distributed effort to build quality APIs.

If your objective is to transform your business, and to make APIs an integral part of how your organization works (the way Amazon transformed its business), then you need to distribute API ownership throughout your organization. A central API team is great to prove the business value of APIs, but if you build APIs as a one off project instead of a continuously developed product, your APIs will soon fall in disrepair.

That is why I think central API teams should transition from an "API pilot team modus" to an expertise center that supports developers throughout the organization. Sharing scarce skill sets like API documentation expertise and developer experience best practices. I think this is the best way to transition an organization into the API-first mindset, necessary to capture the value of internal agility.

Upstream DX: the hidden developer journey

But what are the steps that need to be made to design an API, and how can we remove friction from the process?

1. Engage

Digital transformation initiatives in enterprise organisations often meet inertia or outright resistance. Employees might not have the time or appetite to participate in yet another change program of which they don't understand the value. That is why a digital transformation program will first need to convince developers about the value of web APIs to be successful. The first job of an internal developer portal is to become an education and engagement tool.

- How can you convince people to put their efforts into an API program?
- How can you engage developers?
- Can you implement a reward model, or gamification system that will help them to prioritise the initiative?

2. Catalog

In large organizations it can be hard to discover reusable assets from different business units. This creates waste: different teams in the same organization might implement parallel solutions for similar problems. This is one of the most important reasons why many businesses start an API program, they want to create standardized interfaces between their departments and thus prevent the creation of one-off integrations. These standard interfaces can then later evolve into assets that are valuable in their own right, to facilitate innovation or to build new products.

- How do business units interact, can they be incentivized to interact through APIs?
- Do you already have a catalogue of digital assets and services that could be turned into APIs?

3. Design

API specification languages like Open API/Swagger, RAML, and API Blueprint make it possible to create a "contract" before an API is implemented. This is a best practice, as it makes sure that an API will meet the requirements of both the API providers and consumers. A developer portal could make it easier for API producers to communicate with the consumers of their APIs. Once announced, potential customers could express their interest in an API resource. The resulting dialogue between the API producers and consumers can then help prioritize what APIs get implemented first.

- Does your organization have an API design guide?
- Can you formalize a design process that involves both API creators and consumers?
- Do you already have communication tools that can be used to facilitate the design discussions?

4. Implement

While it is often straightforward to create a web API, it can be a lot of work to implement all the features needed for a mature API. Metrics, scalability, and especially security can add a lot of complexity to an API program. API management gateways solve these problems, and keep on innovating on the API management layer (e.g. Apigee's machine learning solution Apigee Sense that helps API teams recognize security threats).
Remove as much friction as possible from the implementation process. Ideally an organization will provide an API management layer, so that API developers don't need to address these individually. A central team can

provide resources, tools and experts that can help accelerate the API development process.

- Provide tools to develop APIs.
- Make it easy to submit and update API documentation (e.g. through an integration with your code repository).
- Add documentation that explains best practices.

5. QA and Publish

Developer portals can play a role in the API quality assurance and publication process as a workflow and governance tool that you can use to define standards and implement control mechanisms.
In large organizations gamification features on a developer portal could provide an objective and transparent metric for API quality.

- Award quality indicators for each step completed towards a desired outcome (e.g. you get your 5th star when the onboarding documentation for X commonly used platforms is available on the developer portal).
- Create a "dashboard" for the organization that gives management insight into the health of the API program in their division and across the organization.

6. Feedback and support

As a final step it is important to give developers feedback about the importance of their work. Metrics should be available in the developer portal, with dashboards about the usage of individual APIs. Notifications about milestones in the usage of API resources can help to keep developers engaged in the API initiative. Bringing them back to the developer portal to invest additional time in improvements of their most used APIs. A centralized support team can act as a first line of defense that solves the most common problems, so that API customers don't need to disturb the developers of an API.

- How will you measure API usage?
- What is your support plan?
- How will you keep developers engaged?

Conclusion

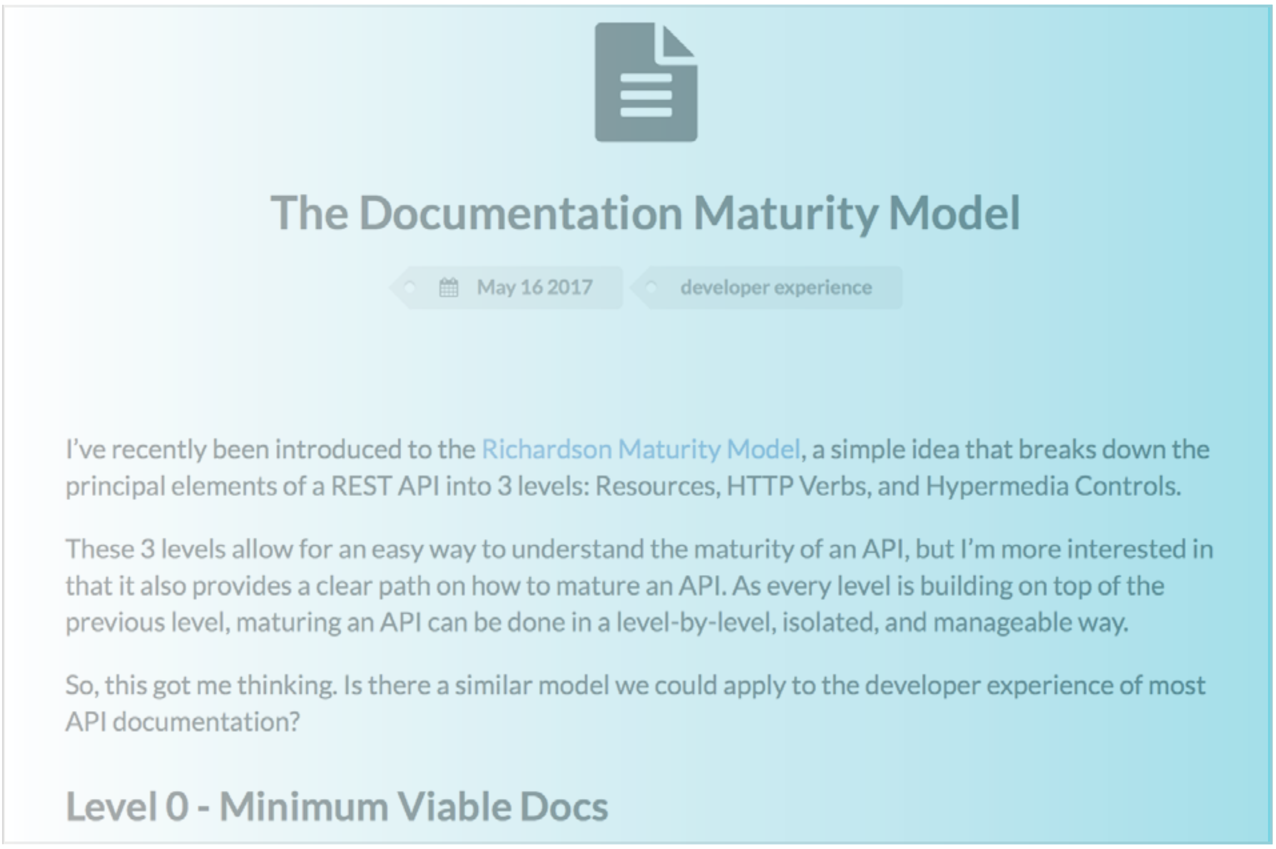
A small API team doesn't need to worry too much about their upstream DX. That is why API teams tend to forget about the upstream developer journey: they are already familiar with the API development workflow and have less need for tools or interfaces that facilitate the process. But in larger companies (e.g. if your business has business units or other types of information silos), improvements in the upstream developer journey can help drive engagement and adoption.

If you liked these articles, check out our series about API strategies.

MAY 16, 2017

The Documentation Maturity Model

by Cristiano Betta



The Documentation Maturity Model

May 16 2017 developer experience

I've recently been introduced to the Richardson Maturity Model, a simple idea that breaks down the principal elements of a REST API into 3 levels: Resources, HTTP Verbs, and Hypermedia Controls.

These 3 levels allow for an easy way to understand the maturity of an API, but I'm more interested in that it also provides a clear path on how to mature an API. As every level is building on top of the previous level, maturing an API can be done in a level-by-level, isolated, and manageable way.

So, this got me thinking. Is there a similar model we could apply to the developer experience of most API documentation?

Level 0 - Minimum Viable Docs

https://betta.io/blog/2017/05/16/api-documentation-maturity-model/

I've recently been introduced to the Richardson Maturity Model, a simple idea that breaks down the principal elements of a REST API into 3 levels: Resources, HTTP Verbs, and Hypermedia Controls.
These 3 levels allow for an easy way to understand the maturity of an API, but I'm more interested in that it also provides a clear path on how to mature an API. As every level is building on top of the previous level, maturing an API can be done in a level-by-level, isolated, and manageable way.
So, this got me thinking. Is there a similar model we could apply to the developer experience of most API documentation?

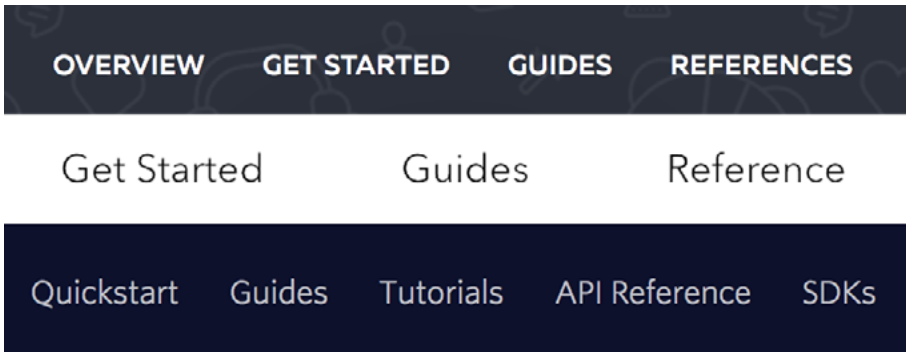
Level 0 - Minimum Viable Docs

I know I said we have 3 levels, but the first level is to have some docs in the first place. Often this is written by some of the first developers on the project, often written for themselves or their first client.
The documentation at this level can be documentation of any type: reference, tutorial, or other. They will be mixed, for example providing you with a guide on how to get started, followed by the reference documentation on the same page.
Often, this level of documentation exists out of mostly reference documentation, focussing on providing a single source of truth and only considering the educational aspect as a secondary objective.

Level 1 - Documentation Types

Reduce complexity by using divide and conquer

At the next level the documentation is fleshed out into separate documentation types. Often this will fall into some variation of Reference, Get Started Guides, Tutorials, Guides, and Exploration documentation.



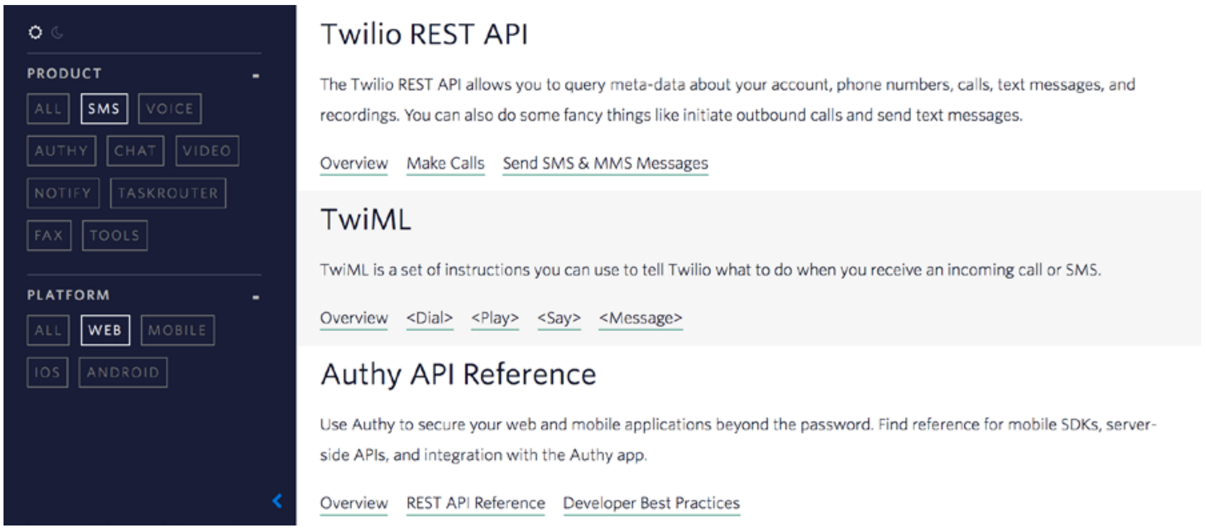
How Virgil Security, Braintree, and Twilio divide their documentation into different types.

By providing a better information architecture in the documentation it becomes easier for developers to find the right documentation for them. As I pointed out in a previous post, the developer journey consists of a few different steps and allowing developers to find the right step for them is essential to a great experience.
In essence, level 1 documentation reduces complexity by dividing the documentation into various documentation types. With this in place each documentation type can own the educational goals of that part of the developer experience.

Level 2 - API Building Blocks

Remove unnecessary variation by increasing reusability

At the next level, the documentation is standardised to provide well defined building blocks for the product. Often this involve splitting previously large parts of the documentation into smaller, more focussed parts which will will have their own standardised, human-readable URLs. Deciding what granularity is appropriate here is probably the hardest part and is very dependent on the product and its core use cases.



Twilio REST API

The Twilio REST API allows you to query meta-data about your account, phone numbers, calls, text messages, and recordings. You can also do some fancy things like initiate outbound calls and send text messages.

Overview Make Calls Send SMS & MMS Messages

TwiML

TwiML is a set of instructions you can use to tell Twilio what to do when you receive an incoming call or SMS.

Overview <Dial> <Play> <Say> <Message>

Authy API Reference

Use Authy to secure your web and mobile applications beyond the password. Find reference for mobile SDKs, server-side APIs, and integration with the Authy app.

Overview REST API Reference Developer Best Practices

How Twilio has split their core building blocks into unique pages.

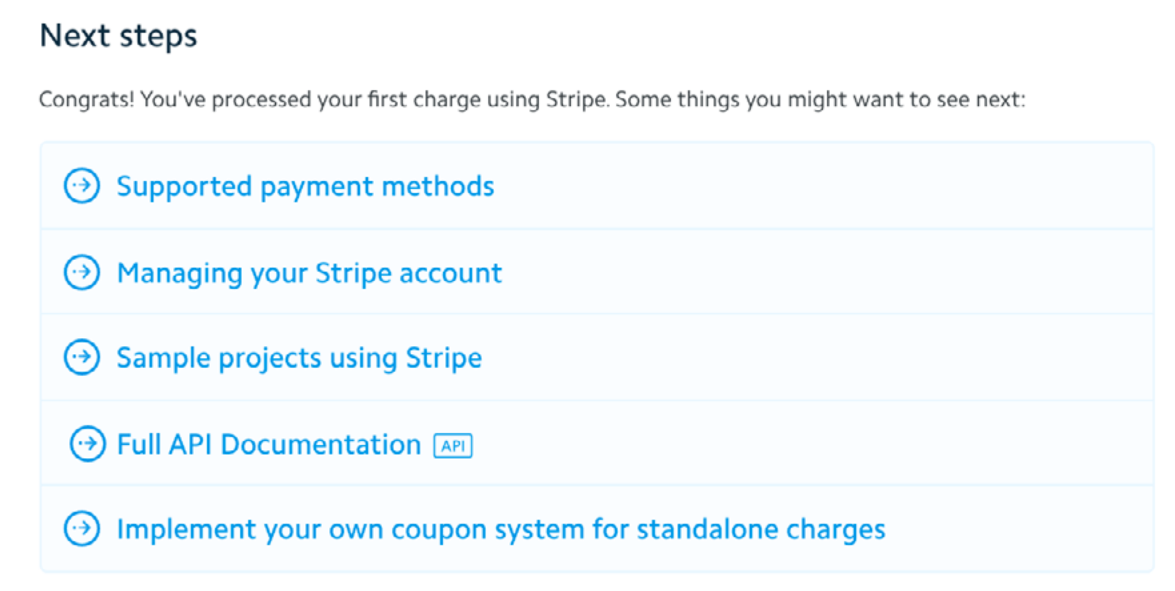
Not all work done at this level is visible to outsiders. Many companies put a lot of effort into standardising code samples and URL schemes so they can be re-used across the documentation and even be integrated into the onboarding experience. The benefit of this is that common mistakes can be easily prevented.

In essence, level 2 documentation removes unnecessary variation by increasing reusability both for external developers and internal copy writers. A strategy is put in place to ensure every part of the product is documented to the same standard. By creating a clear structure and overview of the building blocks that make up the product it can be ensured that each one of these blocks is documented to the same standard.

Level 3 - Cross-References

Introduce discoverabilty across building blocks and documentation types

In the final level we can use the increased structure and reusability of the documentation to turn a strict hierarchy into a fully functional web of documentation. Before this level, the documentation already supported linear navigation through well defined paths.


Next steps

Congrats! You've processed your first charge using Stripe. Some things you might want to see next:

- Supported payment methods
- Managing your Stripe account
- Sample projects using Stripe
- Full API Documentation API
- Implement your own coupon system for standalone charges

How Stripe links out to related content at the end of pretty much every page.

Now, by linking across documentation type and building blocks exploration has been increased massively. At the one hand, providing links that move into more/less indepth topics allow developers to find the level of documentation that best matches their experience level.
An example of this would be to add a link on a guide to the reference documentation of the same topic.
On the other hand, providing links that move laterally into related topics allow developers to always be learning new concepts and ideas, as well as providing a single source of truth for some often used building blocks.
An example here would be the installation and initialisation of an SDK; often this is a prerequisite for most guides and tutorials. Another example is taken straight from the Stripe documentation where the "Payment Quick Start" directly links to both another Quick Start on "Subscriptions", and a guide on "Getting Paid".
In essence, level 3 documentation introduce discoverabilty across building blocks and documentation types by adding links to every page that allow a developer to move from linear paths into related topics.
These related topics can be vertical moves, pointing at content covering the same topic but at a different complexity level. Alternatively, they can be horizontal moves that point at content covering a different topic but at the same complexity level.

Final thoughts

Even though this is just a first draft, I think works out quite well. As Martin Fowler pointed out the 3 levels are not just about the direct aspects of APIs (Resources, etc) but also about the common design techniques that are added at each level. I've taken those techniques and applied them to documentation:

1. Reduce complexity by using divide and conquer
2. Remove unnecessary variation by increasing reusability
3. Introduce discoverabilty

I want to reiterate that this maturity model is not something that should be used to judge documentation, rather more interestingly I think it can be used to provide a clear path for improvements. As each level builds on top of the strengths of the previous ones it can be used as a guide on how to incrementaly improve any documentation.

EVENTS

JUNE 20, 2017

API The Docs, London

A one-day Write The Docs conference about all aspects of API documentation. Explore latest best practices, new trends, and strategies relevant to API documentation.

https://pronovix.com/api-docs-london-2017

- **Daniele Procida:** What Nobody Tells You About Documentation
- **Daniel D. Beck:** 410 Gone. Documenting API Deprecations and Shutdowns
- **Jessica Parsons:** The Best of Both Worlds: A Git-Based CMS for Static Sites
- **Rosalie Marshall:** Creating Consistent API Documentation in Government
- **Ben Hall:** The Art of Documentation and Readme.md
- **Jaroslav Machaň:** Opening a Door to a Sleeping Castle
- **Andrew Johnston:** Documenting GraphQL at Shopify
- **Jennifer Riggins:** How Can API Documentation Become Inherently Agile?

What Nobody Tells You About Documentation

Daniele Procida

Community manager at Divio AG

- "If you give insufficient documentation, the world may decide that your software is just too much trouble."
- But there isn't the one documentation: there are four.
- **4 types of documentation represent 4 specific purposes**. Each one has one specific job and requires a distinct mode of writing, but there are gravitational pulls working among them.
    - **Tutorials are learning-oriented** and provide lessons that take the reader by the hand through a series of steps to complete a project. Tutorials are useful when studying and provide practical steps.
    - **How-to guides are problem- and goal-oriented.** They take the reader through the steps that are required to solve

    a problem. How-to guides are useful when coding, they provide practical steps.
  - **Reference documentation is information-oriented**, it includes technical descriptions of the machinery and how to operate them. References are useful when coding, they provide theoretical knowledge.
  - **Discussions**, also known as background topics, key topics or topic guides **are understanding-oriented**: they give explanations that clarify and illustrate a particular topic. They are useful when studying and provide theoretical knowledge.
- **The right way is the easier way, for both authors and readers**. For authors, the 4 documentation types will define what to write, how to write it and where to put it. This distinction makes maintaining the documentation easier, and also enables readers to use the software more efficiently.

Daniele's presentation

Further reading: What nobody tells you about documentation (blog post)

---

410 Gone: Documenting API Deprecations and Shutdowns

Daniel D. Beck
Technical writer, https://ddbeck.com/

- Communication about how to cancel a project is under-represented.
- As the tech writer, all your feelings about the deprecation are reasonable. But you cannot inflict them on your team or your users.

- You have impact as a writer: it is important to get into the right mindset, plan the communication ahead in order to say the right things right.
- Do not commit to near-future events, buy as much time as possible.

Plan:

- involve the operations and the support team,
- stop the existing outreach,
- list communication channels,
- fit everything into an overall escalation strategy,
- make a time schedule (use not-earlier-than-dates rather than deadlines).

Let users down easy:

- be aware that you and your users might experience the shutdown differently,
- explain what happens when and what is the user to do,
- offer alternatives, give users an exit,
- do not necessarily explain "why" the shutdown happened nor show any hesitation. This is not the time for subtlety or being cool (even if that is in your general styleguide),
- show gratitude.

Find Daniel's slides and presentation

The Best of Both Worlds: A Git-Based CMS for Static Sites

Jessica Parsons
Developer Support Engineer at Netlify

API documentation is necessarily a collaboration between technical writer and developer, but merging their disparate workflows can be a challenge.

Content Management Systems (CMS) are feature-rich, widely used tools to edit and update content without touching code. Their downside is that the content and the development workflow are separated.
With Static Site Generators (SSG) documentarians are able to write by following the dev workflow, e.g. storing documentation in a Git repo, using branches for modification. Hosting is (mostly) free and the content is separated from the tools. There are many SSGs, some made **specifically for documentation**: Sphinx, MkDocs, GitBook, Slate (API docs generator). The downsides are that content must be written in code and they are not as full-featured as content management systems.

Netlify CMS helps bridge this gap between CMSs and SSGs, by providing a simple UI wrapper for Git functions, with a real-time markdown preview. Netlify works with the GitHub API, uses Markdown for content and provides an editorial workflow.

Jessica's slides and presentation

Creating consistent API documentation in government

Rosalie Marshall

Technical Author at GDS

Main goal of GDS is to build government as a platform. Developers need to be able to integrate without wasting time on deciphering poorly written documentation. Encourage code sharing and use of open source

software by providing strong documentation.
Research on API docs needs for GDS:

1. Testing with 30 technical architects and developers:
2. opinion on existing docs sites in and out of GDS
3. card sorting and diary studies on all docs they read in a week
4. Asking companies like Stripe and GoCardLess on what they did to make their API docs a success.

Results:

Docs needs of developers:

- up to date
- right version
- searchable and scannable
- understand scope quickly
- colour-coded blocks of code
- working examples
- direct linking to parts of the docs
-

Too many user needs, no product answers them all. Decide by what they would build for GDS inside use, iterations on their own tool.

GDS needs:

- version control
- self-hosting at gov.uk
- auto-testing of code samples
- consistent one look for all GDS docs, a tool that fits both API docs and general manuals
- use GovSpeak, a GDS-flavoured markdown: warning callouts, numbered lists etc.
-

MVP just out (see on GitHub), next step is to meet the tech writer user needs not yet incorporated.

Further needs:

- separate product for Verify, that is not API documentation
- a slackbot for reviews
- markdown or not (new team-members joining)
- openAPI specification (ongoing)
- moving the various API references from Gelato over to their new product

Prioritising is a challenge.

Rosalie's presentation

The Art of Documentation and Readme.md

Ben Hall
Ocelot Uproar

Whole new ecosystems arise around new technologies, which means we all have a lot to learn. "Documentation is helping developers to have more comfort with how they approach new technologies."

What is the art of documentation? The **user journey** begins long before landing on the docs site. How can we make the **learning journey be more integrated into the product experience**, to make it less explicitly a task? We need to find the developer's **flow channel** with our documentation: not too challenging so we do not trigger anxiety, but not too simple either because then it would be boring. Research results show that teaching others/**immediate use** gives the highest retention rates when learning.

Stage 1: Discovery. Where users decide wether to use the product. At the starting point users don't know anything of the product, so we need to be **very clear and concise with the opening tagline**. Generally build on experiences to express what's happening within the product. (Example intro lines from: Kotlin, Calico, Kubernetes) No need for an outrageous markting pitch at this point.

Stage 2: Getting started The user has a dream of how the product will make their life perfect, so we want to prove that. Good example: Stripe. **Build up confidence and trust that the product will solve the user's problems**. Problem: **no interactive possibility on mobile/iPad**, they would loose out on the journey.

Good example of learning while starting: OpenShift Origin, GO.
Unintentional **blockers**:

- broken/incomplete samples are very frustrating
- source-code-only releases make it much harder to start
- video content: too deep or too wide explanation takes out of the flow and even a small change needs a complete update
- jumping over the gettings started steps

Stage 3: Problem solving Show a prototype for what can be achieved, give **working examples**. Give **seamleass options for adjustments** in the code snippets so the exploration flow is not broken. Be very straightforward. Catch people on their leaning journey with these good practices before they become disillusioned of the product.

Stage 4: Guidance

- Tutorials, how-to guides, discussions.
- Take the user to be an expert.
- Good example is WeaveWorks, Twilio. Take users to the most relevant documentation with seamless choices.
- Slack: you can edit/modify code snippets and get a preview what the result would look like.
- Community: show how they use the product and involve them to

catch the long tail. See Digital Ocean.

Stage 5: References **Readme.md is your gateway to the product**, it sets the tone and show what the product is good for. Most importantly, **tell why the project exists at all**.

Building community: Need to include the **contributor guidelines, license and where to discuss** the project. It is possible to build a community around the documentation itself. Good examples: Kubernetes sig-docs, Docker docs hackathon. **Contribution has to be simple**. Github is not too friendly for small changes, but you can work around that.

Ben's presentation

Opening a Door to a Sleeping Castle

Jaroslaw Machaň
Erste Bank

A talk about the APIs built in Ceska Bank (Erste Bank in Czech Republic) and the story behind. How do banks go about in the digital transformation age? An API platform then an API economy has to be built for a bank to survive the coming years.

With 10k employees and thousands of processes behind, change is hard and slow. "In every big corporation there must be some islands of positive deviation who could break this."

The main motivation to build an API platform wasn't PSD2 but the developers (internal and 3rd party). Need place for rapid prototyping. 2

years ago switched to agile dev cycles.

Prerequisites for being ready for PSD2:

- Long-term based high quality relationship with outside community.
- State of the art applications, comply to world wide standards (not local).

Ceska Bank's Dev portal:

- Focus on the developer experience.
- Single page application, showing the 6 APIs meant for 3rd party developers' use.
- API documentation written in the Apiary platform.
- First bank in the world to release SDKs for their APIs.
- Keys via registration or use the sandbox.
- Sharing and co-operating with community.

Created Gustav in 2015 to show developers where to start with Ceska Bank APIs. Still working, open source banking app.
Q&A with interesting background stories.
"You have to challenge your business."

Jaroslav's presentation

Documenting GraphQL at Shopify

Andrew Johnston
Shopify

At a company like Shopify, API management doesn't know what the clients exactly want. They know that clients search for products, but a product has several properties that the clients may or may not need in their specific case. GraphQL helps Shopify to only return the data that is needed.

They use **Jekyll static site generator for building up their documentation**, in three main steps:

1. **Remove the old docs** before running the build task.
2. Run a **query on the GraphQL endpoint using `graphql-docs`** for receiving the actual schema.
3. **Generate HTML files** from the schema. The files are structured into folders representing the navigation. The changes in documentation can be seen using `git diff`.

Although the queries are self-documenting (based on the nature of GraphQL), **technical writers have key roles in the doc generation procedure**, such as:

- complementing the raw GraphQL schema with descriptions (Facebook's Graph API documentation was a great example for their writers),
- writing educational materials for user onboarding.

Need **detailed conceptual docs** to become familiar with the specific business domain, and to build up some confidence making queries and mutations. Once you are there however, you don't need reference docs anymore.

Andrew's presentation

How can API documentation be inherently agile?

Jennifer Riggins

Jennifer explores then contradicts that documentation is inherently un-agile: she shows many examples and quotes opinions on how docs fit in the agile circle.

Documentation is the number one thing API consumers want, it is the basis of their decision-making. In the world of micro-services and containers, everything has to work together and although documentation is less likely to happen it becomes more and more necessary. Early-on docs allow prototyping, simpler code and true collaboration. Your documentation is your SEO. It has to say what your API does.

Transformations:

- Worldpay waterfall to agile
- Sendgrid uses modified RICE on their Kanban
- Rob Woodgate's notes on docs being part of definition of done

Advice to tech writers:

- Create a style guide then get the devs document their code.
- Template, checklists.
- Automate.
- Docs have to live alongside the code to stay up to date.

Jennifer's presentation and slides.

EVENTS

DECEMBER 5, 2016

Agile The Docs

Explore the tools, processes and challenges documentarians have in an agile development team
London UK

https://pronovix.com/agile-docs

- **Rosalie Marshall**: Challenges Facing Technical Writing
- **Stella Crowhurst and Jamie Measures**: The Worldpay Shift from User Guides to Experiences
- **Rob Woodgate**: Should Documentation be Part of the Definition of Done?
- **Ellis Pratt**: Agile Authoring Methodology: Learning from Lean
- **Emma Hughes**: Bringing Agile to Technical Documentation at Lavastorm
- **Kristof Van Tomme**: Tempo - an Alternative Reason for Agile
- **Ben Hall**: Lightning Talk on Katacoda
- **Adrian Warman**: Lightning Talk: Documentation Management Practices at IBM

Challenges Facing Technical Writing

Rosalie Marshall
Technical writer at Government Digital Service UK

Early 2014: GDS had many products with documentation but no tech writers. What does good government documentation look like? Shall they use automation tools like Swagger or Raml? They were then still relying on their developers to write their documentation - but developers' priority is the product. Hired/ hiring tech writers.

GDS follows the agile principles but they face challenges: the focus is on the product not the docs.

Challenge #1 is the approach that the product should be so good it should speak for itself, so we need no documentation. To promote docs, show user research that users do struggle and need documentation. Sometimes it's hard to translate user reseach data into a prioritized to-do list.

Challenge #2: the docs writers are only brought in when the product is ready. They loose data from middle of the developement process.

Solving these challenges by:

a/In different groups their tech writers integrate differently PAAS, Ben: treat the docs as developer stories, end-of-week email to update on docs changes. Registers, Jen: not part of the dev cycle, watches out for pieces of work that might impact the docs. Verify and Notify, Cathrine: developer makes the docs changes and tech writer reviews the pull request.

Documentation is not part of definition of done in GDS. Open for opinions on that.

b/ Creating standards to assure users that the product is up-to-date, coming from a trusted resource and is accurate.
User research to create a template to target different users' information needs.
Documentation prototype format for developers within the government.

Keen to learn about how to make GDS content more agile

Rosalie's presentation

The Worldpay Shift from User Guides to Experiences

Stella Crowhurst
Content Strategy Director at Worldpay
Jamie Measures
Information Architect, global eCommerce at Worldpay

How they shifted from waterfall to agile user documentation in the past 2 years.

The too few technical writers would have to sit together with the development teams and keep the documentation always updated, sometimes even write it. 30-40 different projects every year, developer teams all over the world using various methodologies: time consuming and hard to plan.

"Working in an agile environment is tough: developers are very opinionated and sometimes they don't like to have a technical writer sitting there every day and asking questions."

Their key to succes was to really get control of the writing process and how they relate it with agile.

- Be more assertive and own the writing process.
- Inform developers what and when will be needed.
- Write only what is really necessary and focus on the persona.
- When the pressure is really bad, do co-writing sessions and/or workshops.
- Inside surveys, kick-off meetings with the project manager.

Decided to focus on developers' integration documentation, and give FAQ-style docs to support users with a problem.

“One of the ways we gain respect with what we’ve been doing is asking those brilliant questions that tech authors ask. [...] We genuinely influence the user interface design in a positive way. An extra thinking brain in the development cycle of the product, asking those innocent questions that developers tend to tunnel-vision away from.”

Stella’s presentation and slides

Should Documentation be Part of the Definition of Done?

Rob Woodgate

For true scrum cycles we need fully multi-functional teams, which we don’t really have. Also, software engineering is still a linear process and some part of documenting always falls to the end. Rob gives us a list of questions to help make a decision whether to still include documentation in the DoD.
See our blog post and Rob’s own notes and links.

Rob’s presentation and slides

Agile Authoring Methodology: Learning from Lean

Ellis Pratt
Director at Cherryleaf

Being lean is to keep the whole process, from beginning to end as efficient as possible. Not just the software development team. Divide work into single-focus tasks (automation), let anyone be able to stop the process in case of error (autonomation).

Identify waste:

- muda: not adding value to the user. Unnecessary content creation.
- muri: overburdening. Too difficult/too much work for the time given.
- mura: unevenness, waiting around. Delays in approval process.

There is essential waste too, for example quality testing.

Load balancing: get rid of over- and underwork, even out the workload. Bring in subcontractors for example - but they need to know the tools too. Templates help.

Docs as code: act as a content developer. Add the docs tasks to the Kanban board, refer in title to the coding reference numbers. Review and edit as if it's the calibration&defect of code.

Deliverables: just enough documentation at first sprint. You can publish docs in increments.

Optimise the whole: make people aware of the costs of documenting late, where that might generate waste.

Lean and agile methods spotlight process problems, so you do need a commitment to resolving those.

Ellis's presentation and slides

Bringing Agile to Technical Documentation at Lavastorm

Emma Hughes

Technical writer at Lavastorm

There are 2 scrum developer teams at Lavastorm and Emma is the one technical writer.
Tools:

- Legacy docs were all in Word/pdf, switched to MadCap Flare, publishing integrated HTML help
- Integrate with Perforce
- Slack - makes docs life visible, also gives peek into what the developers are up to
- JIRA - task tracking on separate docs Kanban board but the devs mark need for docs-writing on their JIRA tickets
- RealTimeBoard - using the devs' tools

Challenges:

- Balancing her time between processing their legacy documentation and writing up new developments.
- Only attends relevant scrum meetings but checks highlights.
- 2-weeks sprints. Docs are not part of the definition of done, but unofficial agreement on keeping the docs up-to-date.
- Work with UI team to keep ahead.

Emma's presentation

Tempo - an Alternative Reason for Agile

Kristof Van Tomme
CEO at Pronovix

Frequency is a function of time: it is essential for change.
Observe the changes in tempo all around us: teams also have their rythm. Be sensitive to the tempo of the group, you can modulate it to your benefit. (see 'Tempo' from Venkatesh Guru Rao) As in FM, we need a carrier wave in a company: the agile process sets this steady rythm and each smaller team can resonate with it.

Kristof's presentation

Lightning Talk on Katacoda

Ben Hall
Founder of Katacoda, interactive learning environment for software engineers

Katacoda is a technical learning platform directed toward software developers, where the documentation is shifted to very real-world scenario heavy aspects. Start with a quick overview of concept, then drop people into a live developer environment. These are virtual machines

hosted directly in the browser or configured as per the developer's specific needs.
Third parties can embed the developer platform on their sites, and so have a more interactive tutorial where the technical preconfigurations are already done (and thus this initial barrier removed). Developers engage better.
They've built an editor that uses markdown and through a set of selections one can set up the environment to work in - but this was still a disconnect from the browser experience. They now use Github as an easy starting point for documentation (in markdown), this way allowing the devs to use their own usual workflow and tools and thus helping continuous delivery for tech guides.
Grammarly' automated suggestions helps reduce file changes back-and forth and generally reduce the editor's work.

Ben's presentation

Lightning Talk: Documentation Management Practices at IBM

Adrian Warman

Content Choreographer (Information Development and Content Strategy) Lead at IBM UK Cloud Data Services.

Within the software development group at IBM, the various teams use doc tools and techniques that reflect the rhythm of their product development process.

- IBM's old style products with yearly release cycles can make use of traditional documentation tools.

- As IBM acquired Cloudant, they not only got the product but its mindset as well. The documentation is written in markdown and stored on github.com.
- IBM Graph is a hybrid sort of internal project using markdown and github.

Old established processes are very difficult to change and an outside developer is very unlikely to write documentation to an old-style technology. But in a never environment they are very likely to write initial contribution (and some cases full scale contribution) particularly to API reference documentation, less so to user guides. Projects using open technologies were able to get a lot more interaction both internally and externally, with daily or even hourly updates.

IBM's content creation process is changing: "technical writing" turned "information architect" turned"content designer". They receive beta-content from developers, marketing, sales, customers, which they then refine with the UX and the key message in mind.

Important but often omitted: Accessibility testing, video/augmented reality technologies. Written words are only a part of the user experience. In the future: Emphasis on user experience, designing the flow, bringing together the different media types and expectations. Using the AI Watson to real-time analyze the support calls to determine the subject of the conversation and refer the user to the corresponding information resource. The tech writers are only helping Watson to help the user. "As technical writers we are becoming content choreographers, ringmasters, guiding people through a rich customer experience."

Adrian's presentation

Continuous Documentation

Chris Ward
Gregarious mammal
Technical writer at contentful.com

Open source testing tools tips to assist continuous deployment of docs. Chris wrote about these fields in more detail and gave coding snippets too on Codeship's blog, see the links.

Spelling, grammar

Markdown-spellcheck uses open source dictionary files. Adjust your script to your preferences (e.g. report mode, ignore numbers and acronyms). Create a separate repository with the typical custom language (not a real word, but not wrong), so they won't get flagged by the checker. Decide what to let through.

Atom editor with extensions

Write-good gives a report-like output, more of a guidance, hard to automate its evaluation. You can write different tests for it, e.g. weasel-words

Testing codes

- Dredd for API examples (in Blueprint and Swagger/OpenAPI formats),
- Inline code testing is a little harder, Sphinx can do it for Python.

Screenshots automation

- Capture a screenshot along your successful selenium tests.
- Idea: from your examples you could generate a json and make videos at a continuous integration. Works for examples from terminal.

Code examples and live code playgrounds

Languages like JavaScript or Ruby offer interactive possibilities with the documentation. E.g. JSFiddle lets you embed your JS and people can experiment with the output while reading the docs. Autogenerate the code, feed the results to a service with an API and generate the playground dynamically.

Chris's presentation

EVENTS

JUNE 1, 2016

API Documentation at WTD NA 2017

by Laura Vass

https://pronovix.com/blog/api-documentation-wtd-na-2017

The annual North-American Write The Docs conference this May featured 2 API documentation redo presentations:

1. **Lyzi Diamond** described Mapbox (formerly a part of Development Seed) documentation automagic in detail,
2. **Sarah Hersh** talked about the journey that NPR One undertook towards a new task-based approach for their API's developer documentation.

In this post we aim to give you an account of these presentations, plus a little extra takeaway.

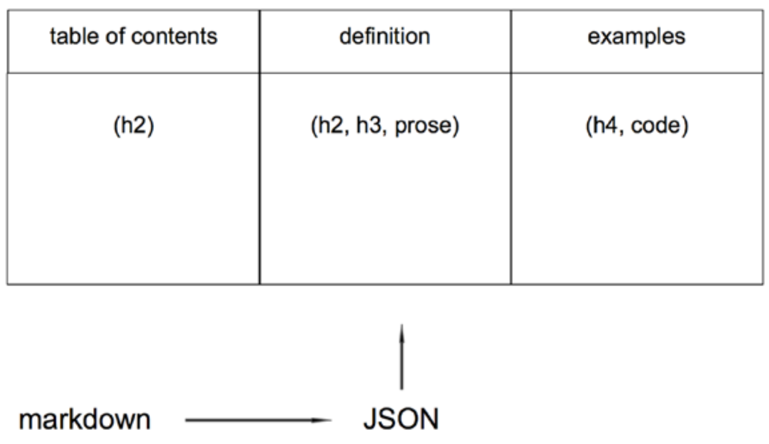
Lyzi Diamond: Testing: it's not just for code anymore

Lyzi Diamond from Mapbox showcased the use of docbox on their own documentation, through remark and retext. Lyzi showed why it works for them to test everything.
The story started in 2015, December, when Mapbox had a confusing TOC for their developer docs. They wanted the **docs to be consistent, complete and correct**, and contribution had to be possible.

The inner workings of the automation process

Their API documentation is in markdown, in a custom specified format which then gets rendered, tested, lintered into a human-readable output.

1. The first batch of transformation happens with remark, a markdown parser that builds a syntax tree from markdown files. Based on predefinition, each syntax is transformed to be rendered in the left, right or middle column.



docbox (Lyzi Diamond, Mapbox)

Remark supports many linters and it also has built-in code generation. Remark is a free and open-source tool, well supported and documented.

2. The second batch of automagic comes via Mapbox's version of retext, they use it to enforce linguistic correctness and consistency. Retext is an ecosystem of plugins that can do much automagic exactly to your custom specifications. For example: you can lint for simple typos or search for words on your no-list.

Overall perks for the whole team

The enforced structure and linters highlight inconsistencies in the original documentation and stimulate the various groups and roles in the company to talk with each other. Not only outsiders contribute: also insider non-coders now have a reference framework for pointing out when and how an API differs from the others on their platform. Furthermore: **the rigorous documentation workflow brought group-wide understanding** with it and asks for consistent decision-making and cooperation.
We got the directions towards the sets of tests (content-test, copy-cop, Hemingway app) and can start using them on our own projects. Thank you!

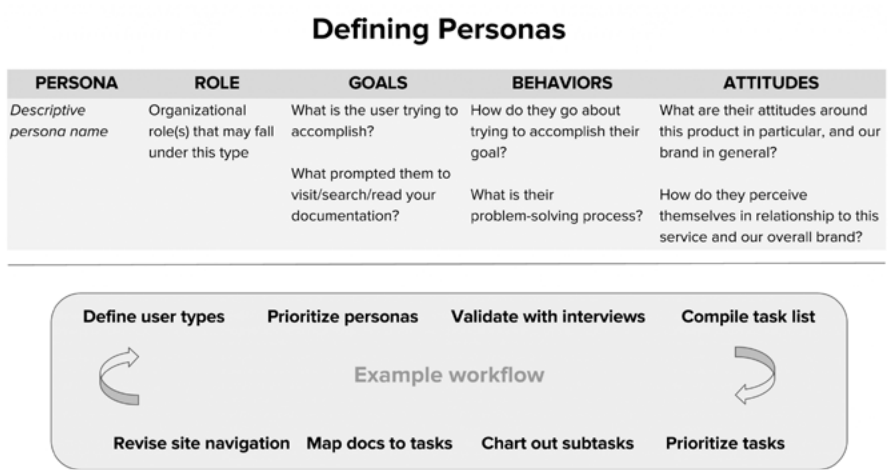
Sarah Hersh: Start with the tasks, not the endpoints

NPR One needed a developer center to enable the extension of the NPR One experience to new platforms and audiences, furthering NPR's original goal of a more informed public.

Sarah illustrated task-based docs with a metaphor of the self-serving coffee shop: one conference attendee goes into a coffee bar in search of that first cup in the morning, potentially jetlagged. What they find inside is no staff but a wall of documentation: confusing and just too much information. What the customer would need at this point is to find clear steps getting her task done. The point is, **accurate and relevant doesn't equal helpful**.

How do we define tasks? When you consider what the developer can do with your API, then the endpoints are the means and not the goal. Do the tasks align with the user, the API-owner's goals and priorities? Before anything else, your team must be clear on why you give access to the API. At NPR One, they defined developer personas:

- What is the persona's role in their organization?
- What are they trying to accomplish? What do they want to find in the docs?
- How do they usually go towards this goal? Does this persona go through a typical problem-solving process?
- Do they have any known attitude/relationship towards the product or the brand in general?



Defining Personas

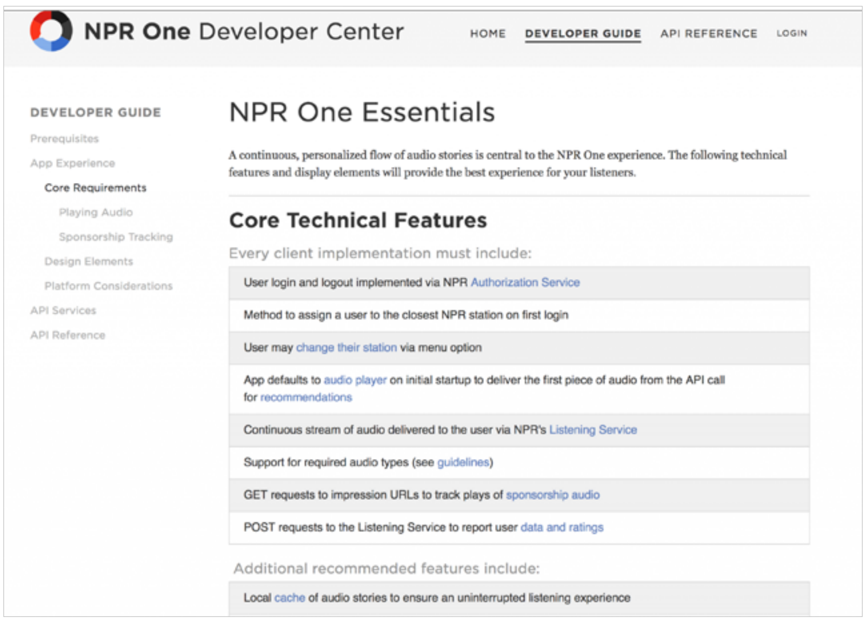
PERSONA	ROLE	GOALS	BEHAVIORS	ATTITUDES
Descriptive persona name	Organizational role(s) that may fall under this type	What is the user trying to accomplish? What prompted them to visit/search/read your documentation?	How do they go about trying to accomplish their goal? What is their problem-solving process?	What are their attitudes around this product in particular, and our brand in general? How do they perceive themselves in relationship to this service and our overall brand?

Defining Personas (NPR One)

Ultimately, **we need to know what developers are hoping to walk away with**.

NPR One's UX research and logged data analysis helped identify the common pain points, e.g. developers complained that they had to visit multiple pages before finding all the information needed.

NPR One prioritized the defined tasks. The team then decided to highlight the most common, so called umbrella tasks. They put the API reference as a separate main menu item for the more experienced developers but kept the rest of the documentation in a classic, shallow navigation structure under developer guide.



NPR One Essentials

A continuous, personalized flow of audio stories is central to the NPR One experience. The following technical features and display elements will provide the best experience for your listeners.

Core Technical Features

Every client implementation must include:

- User login and logout implemented via NPR Authorization Service
- Method to assign a user to the closest NPR station on first login
- User may change their station via menu option
- App defaults to audio player on initial startup to deliver the first piece of audio from the API call for recommendations
- Continuous stream of audio delivered to the user via NPR's Listening Service
- Support for required audio types (see guidelines)
- GET requests to impression URLs to track plays of sponsorship audio
- POST requests to the Listening Service to report user data and ratings

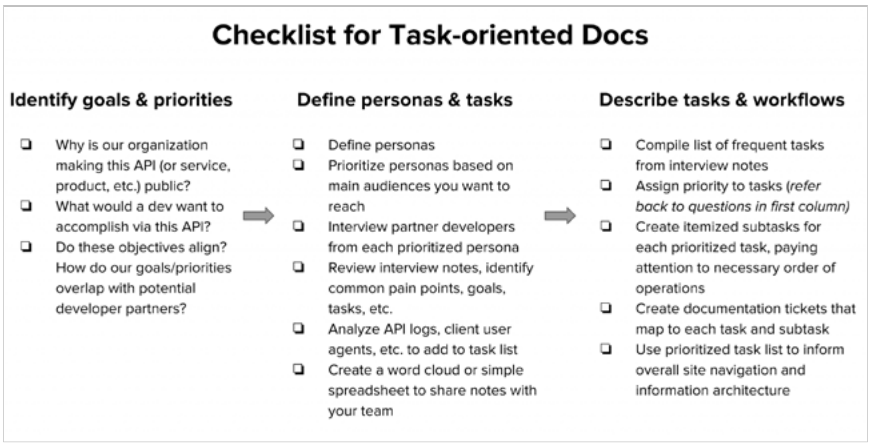
Additional recommended features include:

- Local cache of audio stories to ensure an uninterrupted listening experience

Developer Center (NPR One)

The new developer center received very positive feedback. As a result, the company introduced the task-based approach to other areas, such as the newsletter - which yielded 40% higher opening rates. *The idea is to focus on what the reader might accomplish with the new feature rather than just presenting the feature as a new one.*

Sarah provided us with a checklist for making our docs task-oriented, and a rinse-and-repeat example workflow to get it done.



Checklist for Task-oriented Docs

Identify goals & priorities

- ❑ Why is our organization making this API (or service, product, etc.) public?
- ❑ What would a dev want to accomplish via this API?
- ❑ Do these objectives align? How do our goals/priorities overlap with potential developer partners?

Define personas & tasks

- ❑ Define personas
- ❑ Prioritize personas based on main audiences you want to reach
- ❑ Interview partner developers from each prioritized persona
- ❑ Review interview notes, identify common pain points, goals, tasks, etc.
- ❑ Analyze API logs, client user agents, etc. to add to task list
- ❑ Create a word cloud or simple spreadsheet to share notes with your team

Describe tasks & workflows

- ❑ Compile list of frequent tasks from interview notes
- ❑ Assign priority to tasks *(refer back to questions in first column)*
- ❑ Create itemized subtasks for each prioritized task, paying attention to necessary order of operations
- ❑ Create documentation tickets that map to each task and subtask
- ❑ Use prioritized task list to inform overall site navigation and information architecture

Checklist for Task-orientated Docs (NPR One)

Andrea Longo showed the sysadmin’s perspective on software use and incident management, and although she was talking of software in general, one of her points is very relevant to API documentation: it is not only the end product (versions) we need to document. The reasons and data for our decisions along development can be equally important, at least for internal use.

Many other areas were covered by the speakers, the video recordings are on youtube.

EVENTS

AUGUST 7, 2017

DevRelCon Beijing 2017

by Katalin Nagygyörgy

https://pronovix.com/blog/devrelcon-beijing-2017

Seven years ago I had the chance to visit China, and I had a great time exploring the cultural heritage, large cities and gastronomy. So when my talk got accepted for the first DevRelCon in Asia, I was excited to see China again — this time from a professional point of view. My plane landed in a dust storm so Beijing hid its face while I went to the hotel and took a nap to help with my jet lag. I woke up in the early afternoon in a different place: the storm cleared and left a sunny sky behind.

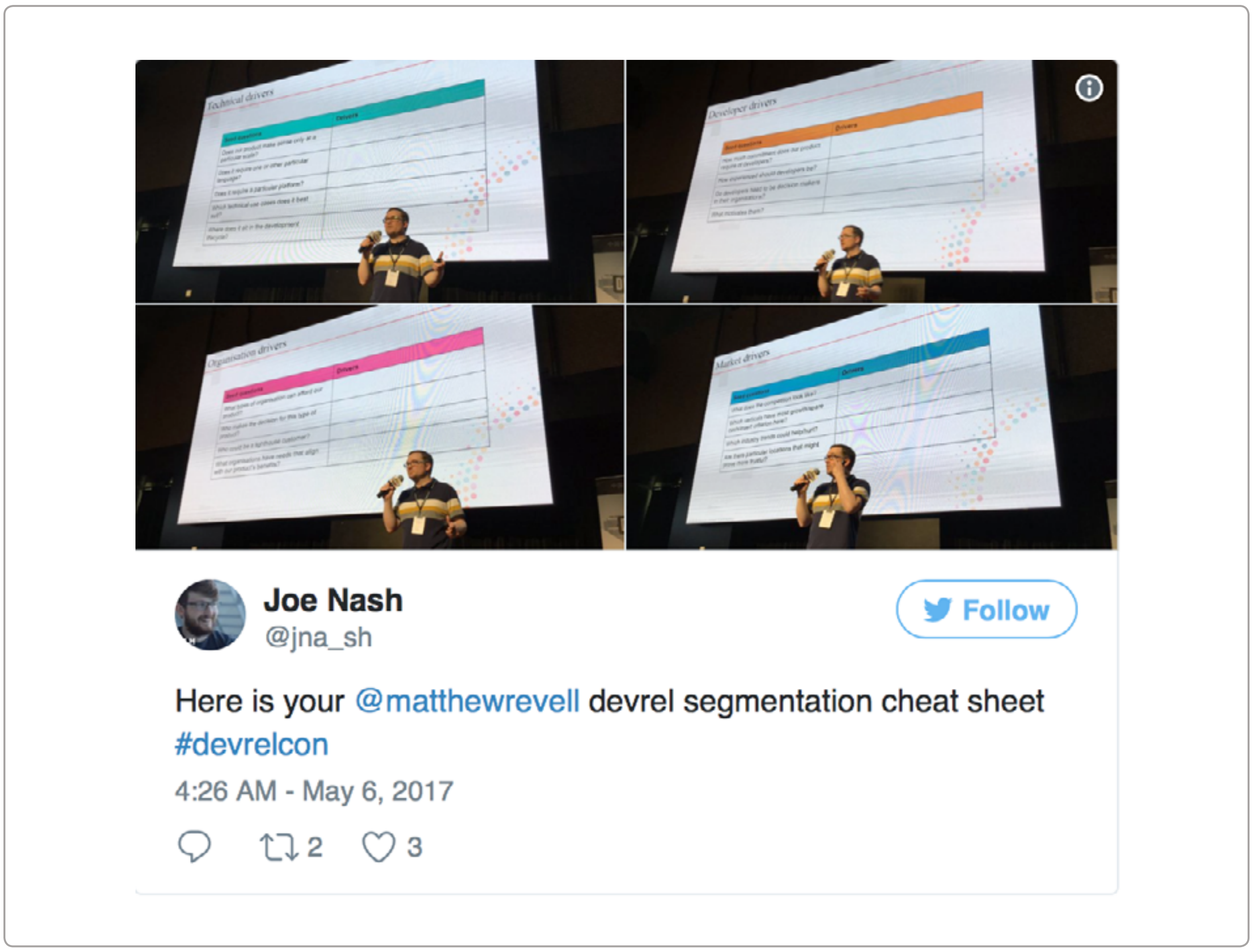
We had the opportunity to meet the speakers and the main organizers, and to check out the event venue in MeePark before the conference which actually took place in the well known 798 Arts District **of the city**. This place is an innovation area for artists and tech people in former factory buildings, so it provided a perfect space for sharing and discussing trending IT topics.

Three main topics stood out for me in the presentations, the first about developer relations in general, the second about the importance of open source projects and the third about the significance of trust building.

Dev Rel trends in the East and in the West

The **essence of developer relationship building** in areas like communication, outreach, dev characteristics and talent acquisition was an important topic throughout the day.
The lineup started with the inspiring ideas of Matthew Revell – the founder of DevRelCon – about effective developer outreach strategy. He introduced a way to segment the developer audience so that we can specifically target each group. The next talk by He Lishi was a nice addition with practical dev evangelism tips for sales and marketing purposes. Atsushi Nakatsugawa focused on problems that a company should solve before starting to do devrel. These talks provided a great

basis to the panel discussion about trends all over the world.



Joe Nash
@jna_sh

Follow

Here is your @matthewrevell devrel segmentation cheat sheet #devrelcon

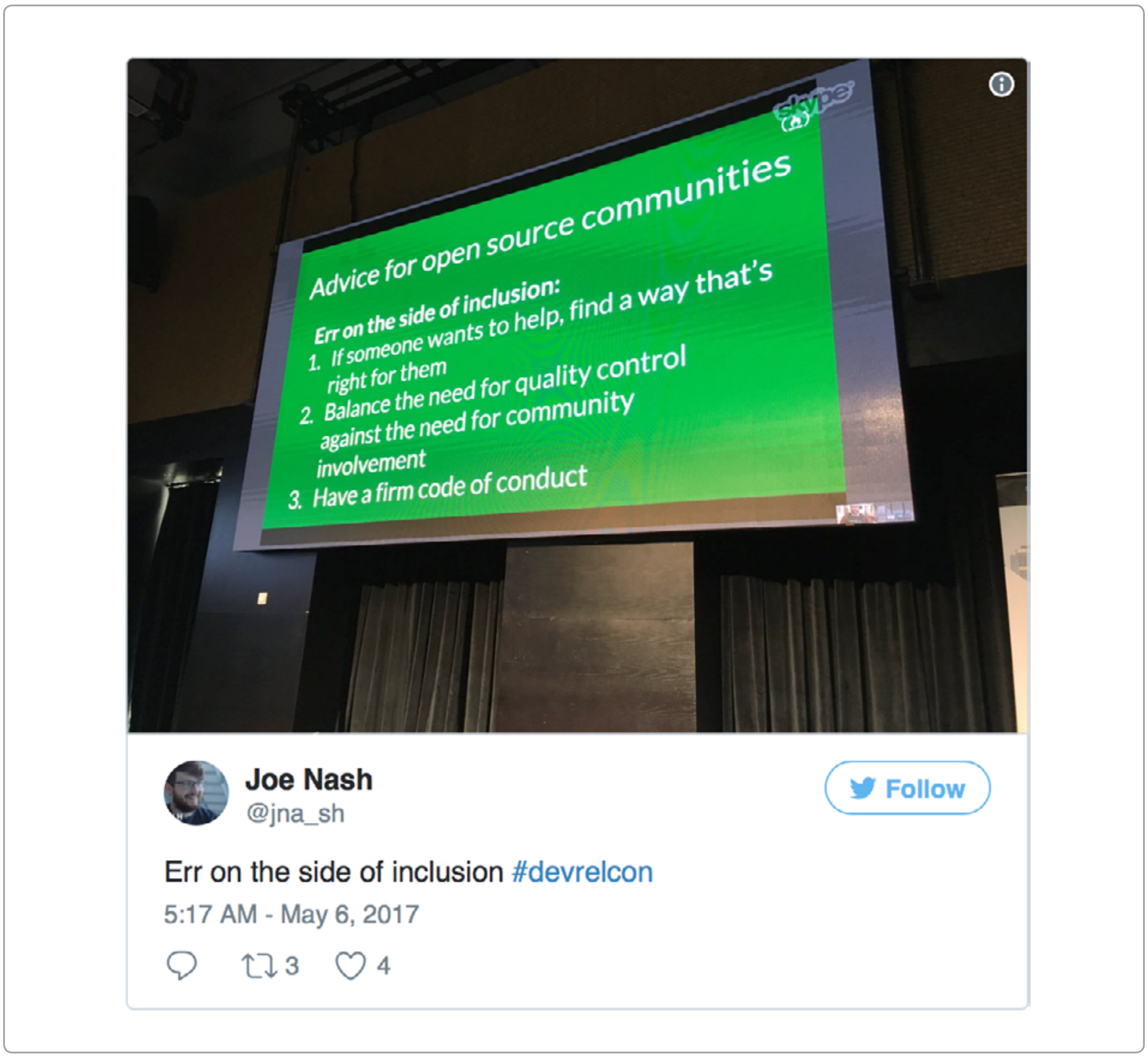
4:26 AM - May 6, 2017

2 3

Other speakers focused on community building and engagement. Yin Ming introduced his community and explained how you can identify what developers like and how to find the best communication formats for them. Jiang Tao on the other hand, spoke about the importance of developer community and talent acquisition and why this matters when building AIs. It was also nice to see how GitHub's student program provides opportunities for students so they can learn faster how the dev world works via Joe Nash.

Why open source projects are important

Although it was surprising to see so much attention was dedicated to open source, it was interesting to see how the conference devoted time to introduce different viewpoints on this topic, and on its role in strengthening the local dev community. Quincy Larson, the founder of freeCodeCamp, joined the event remotely and shared his insights about how building open source communities is helping society at large. From the Chinese point of view Li Jiansheng spoke about the current status of open source communities, and why those are important in a developer's life. Fang Qunjie then explained how devrel can help build better open platforms for communities. Ding Qi's presentation was a great live example which introduced the open source strategy of Alibaba and how it helped the platform succeed.


Advice for open source communities

Err on the side of inclusion:

1. If someone wants to help, find a way that's right for them
2. Balance the need for quality control against the need for community involvement
3. Have a firm code of conduct

Joe Nash
@jna_sh

Follow

Err on the side of inclusion #devrelcon

5:17 AM - May 6, 2017

3 4

How to build trust

Building trust emerged as the third large topic, and was introduced through **two different models**: Phil Leggetter talked about the AAARRRP model, an upgraded version of the startup community's Pirate Metrics. He begins his model with Awareness and ends it with Product.
I gave a talk about the importance of trust in developer portals and explained how you can build trust and successfully strengthen your relationship with developers. Typically developer portals have two conflicting objectives. On the one hand, every developer portal aims to delight developers with a great product experience and to convince them to register as early as possible. On the other hand, companies want only trustworthy people to use their products, they want to avoid misuse. To create a balance between these two objectives it is important to understand the structure of trust and how to apply it to products.

Phil Leggetter
@leggetter

Trust. A recurring theme today at #DevRelCon This time it is @NagygyorgyKata #DevRelConBeijing
11:41 AM - May 6, 2017

2 3 15

Huge community outreach

Besides the 200 participants, the event was livestreamed to developers at venues in five different cities around China, making this a huge conference. The organizers Hoopy and the Chinese partner DevEco did an amazing job, reaching out to so many people at the same time. The whole event was well-organised and highly professional. In the end, we all got a nice plaque — a great memory keepsake.

Authors

Cristiano Betta

Cristiano Betta is a Developer Experience designer who helps companies small and large to improve their developer onboarding, activation, and support. He likes to look at great developer onboarding flows, analysing and documenting the best practices and pitfalls of common design practices. Although he has over 15 years of development experience, he believes that at the core we're all beginners at some things, and documentation and onboarding should reflect that notion.

In the past he's worked as a contractor, startup founder, event organiser, and developer advocate at Braintree/PayPal. Don't buy him coffee, just get him a matcha green tea instead.

Dezső Biczó

Dezső is a Senior Software Engineer and Lead Architect at Pronovix. He wanted to have a computer from a very young age — not for playing games, but to do programming and other cool stuff. He started learning web programming at high school where he met his mentor László Csécsy (boobaa) who introduced him to Drupal. He earned a BSc degree in Bachelor of Business Information Technology and later an MSc degree in Software Engineering at the University of Szeged in Hungary. Thanks to his enthusiasm for computers and programming he is always ready to improve his skills, and can quickly learn new languages and technologies. Nowadays he is building complex Developer Portals for Pronovix customers and contributing to our Drupal 8 Developer Portal distribution as an architect.

Kathleen De Roo

As a copywriter and member of the content team at Pronovix, Kathleen is responsible for writing, reviewing and editing website copy and blog posts, mainly on the many aspects of developer portal documentation. She's got an interest in information architecture. She holds master's degrees in history and in archiving / records management.

Adam Duvander

Adam DuVander loves APIs and the people behind them. He helps Zapier's API partners integrate with its automation platform to empower their shared customers. Previously he led developer relations for database API Orchestrate (acquired by CenturyLink Cloud) and developer communications at SendGrid. He spent four years as a writer, editor, and analyst for ProgrammableWeb, journal of the API economy. Way back when he wrote the book on mapping APIs, Map Scripting 101, and covered developer topics for Wired.

James Higginbotham

James Higginbotham is an API and microservice architecture coach. He provides API strategy, design and documentation services to enterprise IT and software-centric businesses. You can learn more about how James can help by visiting his website.

Diána Lakatos

Diána is a Senior Technical Writer at Pronovix. She is specialized in API documentation, topic-based authoring, and contextual help solutions. She writes, edits and reviews software documentation, website copy, user documents, and publications. She also enjoys working as a Program Monitor for NHK World TV and Arirang TV.

Adam Locke

Adam Locke is a technical writer, mountain biker, bookworm, and craft beer nerd. He and his wife live in the East End of Pittsburgh, PA with their elderly dog.

Katalin Nagygyörgy

Kata is a User Experience Researcher at Pronovix. She collaborates with developer portal customers to define architectural decisions based on their business strategy. She also provides them with possibilities to make sure the product communication resonates with current and future users, mostly developers.
As a psychologist, she is passionate about research methodologies and gamification. She has a PhD in psychology for which she did research into the motivational background of online gaming.

Kitti Radovics

Kitti started to work at Pronovix in June 2015 as an iTrainee, and this was the first time she met with web programming. During the trainee programme, she evolved a lot on the field of theming and Drupal site building with the assistance of her colleagues. Now she is a Front-End Developer and Site Builder at Pronovix and she is also responsible for coordinating internal projects. Since January 2017, she is also the Product Owner of our Drupal 8 Developer Portal project.

Stephanie Steinhardt

Stephanie studied technical documentation with a diploma in software documentation. Since 2005 self-employed as a freelance technical writer with focus on authoring e-training content. Since December 2014 also research assistant at the “Expert-oriented Optimization of Software Developer Documentation” project at the University of Applied Sciences in Merseburg.

Kristof Van Tomme

Kristof Van Tomme is an open source strategist and architect. He is the CEO and co-founder of Pronovix. He’s got a degree in bioengineering and is a regular speaker at technology conferences. For a few years now he’s been building bridges between the documentation and Drupal community. He shares his time between Belgium where he lives, Hungary where Pronovix has its office, and London where he started the local Write the Docs meetup, the group that initiated recent conferences, like Versioning the Docs, Agile the Docs and API the Docs.

Laura Vass

Laura founded Pronovix with Kristof, while still working in toxicogenomics R&D back in 2005. As co-owner she overviews the company's financials and leads the content team. In daily operations she is mostly active as writer/ editor for website copy, articles and blog posts, marketing copy and user guides. Proud contributing writer of the Drupal 8 end user documentation. She is also a zentangles-evangelist, eliminating imaginary bottlenecks to creative freedom.