EXPERT SUPPORT
Understanding technology and writing about it since 1990

The Ideal API Reference

Table of Contents

Why an API Requires Excellent Documentation—2

Terminology—2

Qualities of an Ideal API Reference—2

Complete Content—3

Accurate, Clear, and Unambiguous Content—6

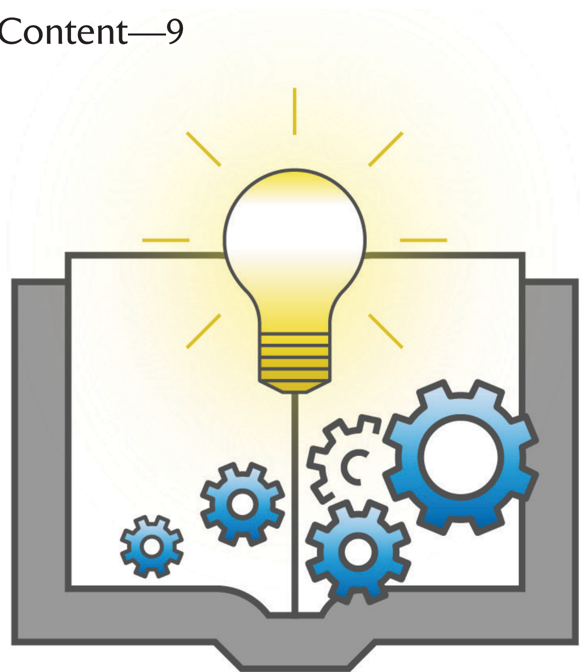
Consistent Content—7

Well-Structured Content—9

Well-Organized and Easy-to-Navigate Content—9

Things to Remember—11

What's Next—12



Definitions, suggestions, and examples to help you create an excellent API reference manual

This article is part of a series of articles on technical writing that Expert Support hosts on its website, expertsupport.com.
Expert Support is located in Los Altos, California and supplies contract technical writers to the computer and software industry.

© 2020 Expert Support Inc. All rights reserved.

Why an API Requires Excellent Documentation

"An application programming interface (API) is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software. An API may be for a web-based system, operating system, database system, computer hardware, or software library." – from Wikipedia

API providers publish APIs to make life easier for other software developers. As Wikipedia suggests, APIs—as well as similar elements, such as libraries, frameworks, and software development kits (SDKs)—are produced for interfacing with programming languages, databases, operating systems, and user applications. Web, mobile, and desktop applications increasingly use APIs, libraries, and frameworks to reduce the development effort. The goal is to significantly decrease the time and effort it takes to develop complicated and sophisticated software. But do APIs deliver on their promise? Sometimes.

To reap the promised benefits, API users need complete, clear, consistent, accurate, unambiguous, and easy-to-navigate documentation. Those are the qualities of excellent API documentation!

When done right, API documentation has several important benefits for both the API user and the API provider:

- Improves the success rate of the API users.
- Speeds deployment of software based on the API, which helps the API user and can potentially help the API provider as well.
- Increases API user satisfaction with the API.
- Increases adoption of (and therefore revenue associated with) the API.
- Reduces support costs for the API provider.

The best API documentation covers a wealth of subjects from specifics about the structure of the API language, to how it should be used, to what kind of environment it runs in. API documentation suites often include the following docs (among others):

- Technical overview (an introduction to the API)
- Installation guide (often with information about setup and configuration)
- Getting started guide (usually contains a Hello World tutorial to introduce and test the programming workflow)
- Additional tutorials
- Cookbooks and recipes
- Software design guide (a.k.a., developer guide) with complete conceptual material and context, guiding the API user's thinking during application design and coding
- Build, test, integrate, and deploy guide
- ***API reference*** (detailed syntax and context for each element of the API)

For information about the ***ideal API documentation suite***, check out this article.

All these documents can provide significant benefits for the API user, giving them a positive experience with the API, and helping them successfully develop usable and maintainable software. In this article, we're going to focus on the **API reference**.

A thorough, well-written API reference is ***the key document*** for an API provider to offer. Without it, API users struggle to work with the API, because they don't have the necessary information to apply the API within their own software.

There are a lot of API reference manuals out there, but only a few are really good, in our opinion. We thought that it would be helpful to be clear about what ***qualities and characteristics make an ideal API reference***, with ***a few hints about how*** to create one along the way.

Terminology

There's no industry-standard terminology to use for the roles that we need to talk about in this article, so here are the terms we're using:

- **API users:** Software developers who make use of an API
- **API providers:** Organizations that develop/provide an API
- **API designers:** Engineers who design and develop an API
- **API writers:** Technical writers who research and write the API reference

Figure 1 illustrates these roles and the relationships between them. In the figure, we've also included the parallel roles associated with software applications to distinguish them from the API-related roles.

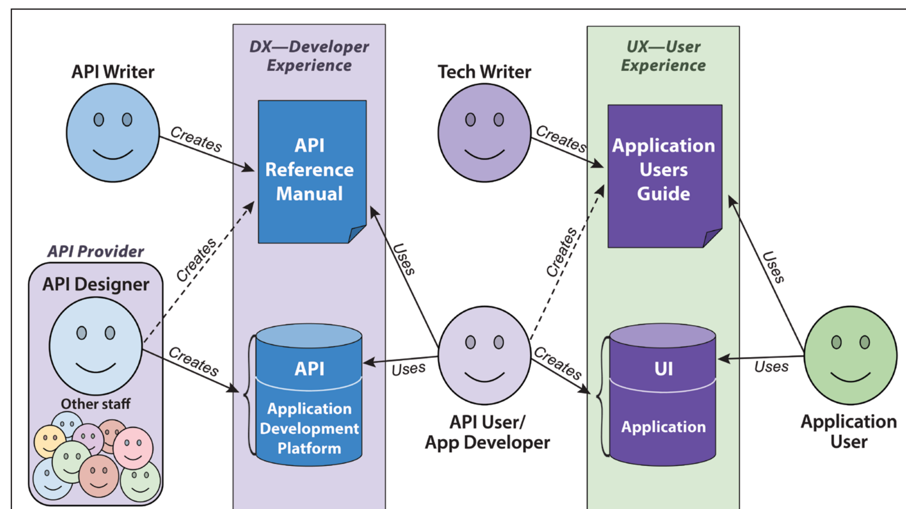


Figure 1. Roles and products associated with application development platforms and applications

© 2020 Expert Support Inc. All rights reserved.

Note that both the API and the API reference manual are part of the developer experience. In parallel, the application's UI and the application user guide are the critical parts of the experience for the application's user.

In the figure above, we intentionally made a distinction between the API writer and the tech writer who produces the application user guide. These two individuals have different kinds of knowledge and skills. Only a small percentage of tech writers are true API writers who are capable of creating an ideal API reference. Nearly all API writers have experience writing code.

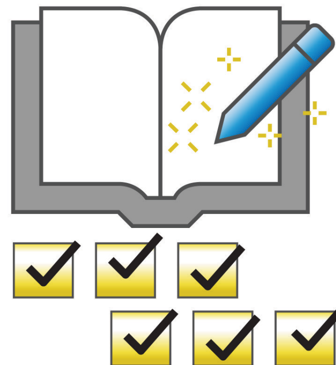
Qualities of an Ideal API Reference

We think that an ideal API reference should be:

- Complete
- Accurate, clear, and unambiguous
- Consistent
- Well structured
- Well organized
- Easy to navigate

With these qualities, a reference provides complete, clear, and concise information that can be easily found by the API user. Although this might seem obvious, getting an API reference to have these qualities isn't easy and is often time consuming. Also, not everyone agrees on the definition of these terms for an API reference, so we want to introduce our definitions.

Below we hope to provide definitions for these qualities, and give you a peek at what they might look like along with a few hints of how you achieve them.



Complete Content

How can you tell if an API reference is complete? Ask this question: ***Can this reference be used as the specifications for the API?*** Or, put another way: ***Can the API provider create or recreate the API from it?*** If the answer is no, the reference likely isn't complete enough. At a minimum, the reference manual should be able to answer the questions in the table below.

For Objects and Classes

- What does the object/class represent?
- What are the attributes of the object/class? What values can those attributes have?
- Are there default values for the attributes?
- Are the attribute values inherited? If so, how?
- What methods/functions are associated with the object/class?
- How is the object/class used in relationship to others?
- Are there specific programming tasks associated with the object/class?
- Where is a real-world code example snippet that shows exactly how to use this object/class?

© 2020 Expert Support Inc. All rights reserved.

For Functions, Methods, and Procedures

- What is the purpose of the function/method?
- When should the function/method be used? What is the optimal use of this function/method?
- Are there any related functions/methods that programmers should know about and potentially use with or instead of this one?
- How is the function/method called and what is the syntax of the call?
- How is the function/method used in relation to others?
- What needs to be set up before some code can call this function/method?
- What are all the parameters of this function/method? Where do all of the parameter values come from? For instance, is the input to one function the output from another?
- Does the function/method return some data, message, or status? Is it clear what the return value means?
- Are there side effects?
- What error codes can come from this function/method? What does this "xyz" message or status code mean, and what actions should the user code take when the code is returned?
- Are there rate limits? Pagination?
- Are there any caveats to using this function/method?
- Are there any real-world examples of code that shows exactly how to use this function/method?
- What further warnings, cautions, or notes?
- What user privileges, if any, are needed to use this function?
- Is there a commonly used procedure that needs to be explained to the users? If so, what is the step flow diagram for this procedure with all the items in the right order with the right parameters? Or, is there a real code example available?
- Is there enough information available to understand and explain what the procedure is doing and why?

For Parameters

- What is the purpose of this parameter?
- What are all valid values and their meanings? Default values?
- When and why would someone use these parameters and each of their various values?
- What happens when invalid values are specified?
- What are the minimums and maximums?
- Are there any prohibited values? What are the limitations and assumptions about the character set?
- On what does the value's definition depend: user or user's environment? If user-dependent, how and where is the definition made? If environment-dependent, how is the environment identified and how is the definition made differently for different environments?
- If there's no dependency and the text is free form, what are the constraints in terms of supported and unsupported character sets, invalid characters, and size limits? Is a value user defined, or does it depend on the user's environment.
- Do certain parameter values have an effect on other parameters or commands? For example, if one parameter turns sound off and on, and another parameter sets the volume from 0 to 10, the documentation should show if or how the parameters interact.

Other Items

- What package or library supplies this API?
- Under what other conditions, if any, is use of the API allowed or denied?
- What data types are specific to this API? What are their possible values? Minimums/maximums? When can they be used?
- Are there any data structures that are used globally by the API elements?
- Is localization required, and, if so, how would the application need to account for it and what other impacts does it have?
- Is there an associated database?

© 2020 Expert Support Inc. All rights reserved.

Using our definition of completeness, ***an ideal API reference describes the syntax and function of every element of the API in detail***, even if the description of an element might seem obvious. The reference needs to describe every publicly exported object, class, function, method, property, parameter, data type, and so on. If it's a web-based API, every resource, method, endpoint, request, response, and so on needs to be covered.

In addition, the reference needs to describe every aspect or characteristic of each element, including its intended use. This ensures that nothing is assumed or overlooked, and helps to instill the API user's confidence in the documentation. It's important for API writers to be thorough, even when they think the information is obvious.

Here's an example of the `animate` function from the [Angular API Reference]. This entry includes the element name, what it is, a complete syntax with links to other elements, and detailed descriptions of each parameter and return values. We'll dive into more details later.

Name

animate FUNCTION — Element type

Defines an animation step that combines styling information with timing information. — Brief description

```
animate(timings: string | number, styles: AnimationStyleMetadata |
AnimationKeyframesSequenceMetadata = null): AnimationAnimateMetadata
```

Detailed syntax

Parameters

`timings` (Parameter name)	`string | number` (Parameter data type)	Sets `AnimateTimings` for the parent animation. A string in the format "duration [delay][easing]". • Duration and delay are expressed as a number and optional time unit, such as "1s" or "10ms" for one second and 10 milliseconds, respectively. The default unit is milliseconds. • The easing value controls how the animation accelerates and decelerates during its runtime. Value is one of `ease`, `ease-in`, `ease-out`, `ease-in-out`, or a `cubic-bezier()` function call. If not supplied, no easing is applied. For example, the string "1s 100ms ease-out" specifies a duration of 1000 milliseconds, and delay of 100 ms, and the "ease-out" easing style, which decelerates near the end of the duration. (Parameter detailed description)
`styles`	`AnimationStyleMetadata | AnimationKeyframesSequenceMetadata`	Sets AnimationStyles for the parent animation. A function call to either `style()` or `keyframes()` that returns a collection of CSS style entries to be applied to the parent animation. When null, uses the styles from the destination state. This is useful when describing an animation step that will complete an animation; see "Animating to the final state" in `transitions()`. Optional. Default is `null`.

Returns

`AnimationAnimateMetadata`: An object that encapsulates the animation step. — Return value: data type & description

Figure 2. Example function description from the Angular API Reference

© 2020 Expert Support Inc. All rights reserved.

Completeness means two important things: describing all the elements of the API and describing each element exhaustively.

Can a writing team go too far with completeness? Of course. Remember that an API reference is a reference. This means it's typically used as a random-access document, so every entry must stand alone. Reference entries should describe general usage intention—what it does, what it's for—and should make the context explicit by describing how an element is expected to be used with related API elements and data structures.

Some discussion points belong in other documents in the API documentation suite. For example, long discussions about usage, use cases, or how the API is structured belong in a design or programmers guide, or in special introductory pages or appendices for the API reference.

Accurate, Clear, and Unambiguous Content

While ***complete*** content is essential, if the API user can't understand the content, or worse, if it's inaccurate, the writer might as well have not written it. When we talk about accuracy, we're referring to the correctness of the content. Too often, the content contained in an API reference isn't correct or doesn't work for the user. Sometimes the content is accurate, but unintelligible to anyone other than the person who wrote it, and in other situations, the content can have multiple interpretations, most of which are wrong.

Achieving ***accuracy*** and ***clarity***, and ***avoiding ambiguity*** is difficult, even for good API writers. Typically, issues with accuracy, clarity, and ambiguity arise due to one or more of the following situations:

- Inaccurate, incomplete, or unintelligible API specification (spec)
- Not enough access to the API designers to get content
- Not enough access to the API designers to review documents
- Non-writers writing significant sections of content that ends up in the API reference
- Not enough reviewers outside of the API design team

The following sections provide some guidance on how API providers and API writers can overcome these issues.

Assign a Senior API Writer at the Beginning of the Design Process

An ideal API reference is typically based upon a good spec, so it's important at the beginning of an API development project to ensure that the team develops and maintains a really good specification. To accomplish this, an organization can assign a senior API writer at the beginning of the project (even if only on a part-time basis) to help create/maintain the spec. When there's a professional API writer involved at the beginning, that person can ensure clarity and unambiguity in the specification from the start. The technical writer can also maintain a *technical glossary* (which we talk about later in this article).

Other benefits to having a senior API writer on the API design team include:

- The writer of the API reference may be the first person to consider the entire public API (the layer exposed to the API user) from an API user's perspective and can point out incongruities from that perspective.
- The members of a product team generally specialize in particular parts of the product or areas of functionality, and aren't as familiar with areas outside of their specialty; the API writer has a complete view of the API.
- The API architects are often more concerned with implementation choices than with the completeness and consistency of the public API, and that's exactly what the API writer is paid to worry about.
- Focusing on developing the spec or an API reference gives the API writer an opportunity to share their observations about the API user layer, representing the perspective of the target users. With these observations and perspectives, the API design team can decide how to improve the API design early in the process, when the cost of doing so is relatively low.

Generate Reference Material from Source Code

For efficiency and accuracy, many API development teams generate their reference documentation from comments in source code as a part of the development workflow. API writers achieve the best quality API reference if they can work directly on the comments in the code. When editing generated documents, an API writer does the following:

- Examines the comments for unclear or incomplete descriptions common to initial cycles of API documentation, and adds complete descriptions for elements that the authors considered "self documenting" or obvious.
- Ensures completeness by checking that all input and output values are mentioned and described, and that defaults are provided for optional elements.
- Revises existing comments for consistent style and terminology.

Granting permissions and privileges to API writers for commenting yields high return with low risk. Writers who are capable of describing and explaining APIs are equally capable of both editing comments in code and working with version-control tools. For developers who are concerned about giving API writers access to their source files, some tools limit editing access to only comment lines.

Some API development teams don't allow writers any degree of write access to source code. Instead, the API writer does the documentation work in a separate branch where changes are permitted. In this situation, the team needs to allocate extra resources to perform several follow-up tasks, including:

- Merging comment and other documentation changes into the main line
- Regenerating documents
- Scheduling additional rounds of reviewing

An integrated development environment (IDE) or an editor made code-aware with plugins can have an embedded documentation generation feature or site generator. Examples include Antora for AsciiDoc, JSDoc 3, and the Javadoc Generation wizard in Eclipse.

© 2020 Expert Support Inc. All rights reserved.

Use Other Existing Material as a Resource for Content

When the project is underway, different groups within the API provider (such as the design, development, architecture, and marketing teams) can often provide considerable information about the API, even if it isn't yet in a form that's useful to users. The content tends to be focused on implementation choices and details, rather than the user layer and usage model. Existing material can include:

- Product requirement documents or design documents
- Presentations, ideally with audio and video recordings or detailed speaker notes to provide a full explanation of the slides
- Product/customer support documents or knowledge bases
- Marketing overviews or white papers
- Existing interactive or developer-created documentation, beyond what's in the source code
- Content from important team email streams

From this material, experienced API writers can extract concepts, diagrams, terminology, and possibly even chunks of text and examples. With these elements, writers can fill in some holes in the reference or create a first pass at usage-oriented content. These elements can also form a basis for further questions to ask the subject-matter experts (SMEs).

Collect Content from Subject-Matter Experts

Subject-matter experts (SMEs) are typically members of the API development team. API architects and developers know the technical architecture and how the API is built, operates, and is intended to be used. SMEs can also be product managers, project managers, and QA and customer support team members who specialize in this product.

Throughout the documentation process (although primarily in the early stages), an API writer has to solicit information from SMEs. The writer's job is to focus on the usage model and to work with the SMEs to provide context, identify and correct inconsistencies, and complete content where a deeper explanation is necessary.

SMEs are often under too much pressure to have a lot of time to talk with writers about the product, so it's essential for the writer to use every minute with an SME wisely. As such, it's critical for the API writer to be prepared before approaching an SME.

In the best of all situations, the writer should have attempted to collect and understand all the available pieces of information listed in Complete Content before interviewing the SME. When possible, an interview with an API SME should be more about getting clarification and filling in the gaps rather than collecting a lot of raw information that can be gleaned from other sources.

API writers need to prepare their questions carefully, and ask them cooly and neutrally. It's also important for writers to be confident enough to ask clarifying questions when something doesn't make sense—it really might not make sense.

Get SMEs to Review the Documentation

It's critical for the API design team (and SMEs in general) to review the API documentation. It's the only way to ensure accuracy and completeness. But getting access to an SME's time is tough. To make it worse, if SMEs are given a lot of documentation to review at one time, their eyes will likely glaze over and they'll stop seeing the errors. Again, there are a number of techniques that help address these issues:

- **Split the review into small chunks.** Try to keep the review time to less than 30 minutes. This might mean that you can only send out one or two functions at a time for review.
- **Make sure that the content is really ready for a review.** Before you send out the content, clean it up a bit. Is it devoid of typos? Does it reflect the recent changes that the writer knows about? Does it use the appropriate terms, formatting, etc.? The cleanliness of the writing is incredibly important, because the writer needs to keep the SME focused on what they need to review—the technical content. They shouldn't be distracted by extraneous errors.
- **Send out a review to one person at a time early in the review process.** Until the primary SME has reviewed the content and deemed it technically correct, don't show it to others.
- **When you get corrections from the reviewer, apply them globally.** Nothing makes SME reviewers more upset than correcting the same error, term, or change multiple times. Note: This might take some thought on the writer's part.
- **Start the review process early!** Don't wait until the last minute when everyone is under the gun to get the product out the door.

Get Outside Reviewers to Review the Documentation

SME reviewers are necessary for getting the technical content accurate. However, you also need reviewers outside the team who aren't familiar with the API reference, as they can point out where the content isn't clear, and give you a user perspective.

These outside reviewers can be customer app developers, sys admins, support personnel, and field engineers. Maybe the writer can use customer support personnel within the organization, or a "friendly" customer (like a Beta tester) who can serve this purpose. In any case, the team needs to find someone outside the development team who doesn't know what the words are supposed to mean. Writers outside the project can potentially be good reviewers, as long as they have the coding skills necessary to understand how to use the API. And also note that, by the end of the project, the writers working on the documentation are too indoctrinated in the API to be objective reviewers.

Consistent Content

Consistency is another important element of a good API reference. Typical API users treat an API reference like a dictionary, where they want to be able to look something up quickly, and get concise information about it. Consistency in terminology, order, format, color, linking, descriptions, and other aspects are critical to the API user's ability to find and understand the information quickly.

© 2020 Expert Support Inc. All rights reserved.

The technical writer needs to make the following things consistent throughout the reference:

- Terminology
- Description style
- Formatting
- Navigation
- Ordering or structure

To achieve consistency, experienced technical writers develop important meta-documents that guide them while writing. We describe the three most important of these documents next.

Glossary

One of the first things that many experienced writers do when they join a project is create a (project or product or technical) glossary. Sound odd? It turns out that different team members, preliminary docs, specs and discussions, proof-of-concept implementations, wire frames, and so on often use different terms to mean the same thing, or ***almost*** the same thing. Or worse, they use the ***same term to mean two different things!*** These inconsistencies make it difficult for API developers to understand what the product does. It also makes it tough for them to ensure that all parts of the API are consistent. Even worse, terminology issues make it difficult for API users to figure out how to use the API from the reference material.

It's critical for the writing team or the API design team to create and maintain an internal *technical glossary*. The design team uses this document to standardize their naming conventions, while the writing team needs this glossary to ensure that terms are used consistently throughout the API documentation suite. We suggest that the glossary includes these types of terms:

- Terms that are specific to the product
- Common words that have a specific meaning in the product, such as a group, person, reader, staff, or user
- Terms that are overloaded with multiple meanings and uses within the industry

We recommend eventually publishing the basic glossary terms publicly so that users who are learning about the API can understand specific terms. For an example, visit the Angular Glossary.

Document Style Guide

Similarly, the writing team should create a style guide for writing the API reference manual content. This guide sets standards that help the API user understand what they're looking at, how to find it, and how to use it.

A style guide should cover as many areas as possible related to creating the documentation, including:

- Grammatical styles and choices, such as whether to use the serial comma, or when to hyphenate terms
- Preferred tone of document, formal or friendly
- Word choices, such as how the writer refers to the reader, how things are capitalized, and standard abbreviations
- Product name and branding rules
- Formatting conventions, such as how and when to use headings, how to format syntax descriptions, how to format code examples, and what links should look like
- Conventions for documentation elements, such as tables, lists, bullets, and quotes
- What colors, fonts, and icons to use to highlight different objects under specific situations

Large companies often have an internal style guide, although it's typically geared toward user or marketing documentation. If there's no existing style guide for technical documents or APIs, the technical writing team should create one to set standards for the project. If the documentation is open source, this guide should be made available to contributors. For an example, take a look at the Red Hat Style Guide or the Angular Style Guide.

Coding Style Guide

When writing API documentation, it's important to have a coding style guide for both the technical writers and API users. This style guide is especially useful for creating code examples within the documentation. We can't stress enough how important it is for the documentation to have examples that are considered ***good code.*** API users are notorious for copying code straight out of the documentation. If the examples aren't clean, bad code can proliferate around the community, eventually causing support headaches or a bad reputation for the API.

Requests

A REST API request is represented within WordPress by an instance of the `WP_REST_Request` class, which is used to store and retrieve information for the current request. A `WP_REST_Request` object is automatically generated when you make an HTTP request to a registered API route. The data specified in this object (derived from the route URI or the JSON payload sent as a part of the request) determines what response you will get back out of the API.

Requests are usually submitted remotely via HTTP but may also be made internally from PHP within WordPress plugin or theme code. There are a lot of neat things you can do using this class, detailed further elsewhere in the handbook.

Figure 3. An example glossary term from the REST API

© 2020 Expert Support Inc. All rights reserved.

The ***coding style guide*** needs to cover naming conventions, preferred coding practices, maybe even formatting conventions. If possible, this guide should cover healthy coding practices that the engineering team wants all API users to adopt.

Ideally, this meta-document should be part of the published documentation set. For an example, see the [coding style guide for Angular]().

Well-Structured Content

OK, let's say that the prose of an organization's ideal API reference is complete, accurate, clear… just perfect. But does the document have a **clear and logical structure** that enables API users to guess where things are located? API users make use of reference documents in different ways, according to their current task and personal preferences. As appropriate to the language, some are interested in finding a function, class, method, endpoint, or object that does what they want. Some want the short answer about what a class or method does, and how to use it. Some API users need to check a detail about syntax as they use a function in their code. Most just look for a snippet of code to copy and paste.

Each element type in the API should have a defined structure and that structure should be used consistently, so users can find what they need. For example, the entry for an API function should be able to answer the questions listed in Complete Content.

Some elements have more content than others, and some require several examples. The key thing is the structure, and that structure depends on what the API user needs to know. One of the API writer's tasks is to create a template that suits the needs of the API they're working with.

The figure below shows an example template for a function.

```
[name or action]
Purpose
    …
Syntax
    ….
Description
    …
Access Control (or Scope)
    …
Parameters
    (Subheads for different types of parameters; e.g.,
    for endpoints, might be URL, Header, and data parameters)
    …
Returns
    …
Errors
    …
Example
    …
Related functions
    …
```

Figure 4. A template for a function description

Well-Organized and Easy-to-Navigate Content

Document content can be accurate, consistent, and well written, but if the reader can't find it, it might as well not be there. So, part of the job of technical writers is to make things **easy to find**. There are a number of writing techniques that can help the reader:

- Organize the content well so that element lookup is easy.
- Potentially provide several organizations for the different kinds and levels of users.
- Include hyperlinks to other API elements and related content.
- Use names and terminology that aid the reader when searching the document.
- Employ multiple structures or documents that organize the content for easy access, such as a quick reference guide.

Provide Well-Organized Content

The structure of individual elements and the overall structure of the API reference are both critical. The reference's structure/table of contents (TOC), whether in a paper book, a PDF, or a web page, needs to be organized to help API users easily find what they need. Ease of use and search optimization are the primary goals when designing the navigation organization.

To organize the top-level topics of an API reference, the technical writer must first assess the many ways to organize the elements. Common strategies are:

- Alphabetical order
- Categorical groups
    - o Hierarchies, such as classes and methods in each class
    - o Concepts
    - o Actions
    - o Processes
    - o By function
- Order of use (task orientation)

The choice is determined by the content, the intended audience, the size of the API, and what kind of companion material is available. For example, a task-oriented approach to organization is appropriate for a user guide or cookbook document, which is read sequentially or in sections. However, this approach is rarely appropriate for an API reference, which readers typically access when searching for specific language elements.

The simplest organization is alphabetical, which is often used when the API is very small. Note that many providers of more complicated APIs organize first by function and then in alphabetical order. For example, the Okta API Reference content is organized at the highest level by function (mappings, policy, schemas). Then the next level is organized by categorical groups (objects, event types, operations), and the specific elements of each group are listed alphabetically.

© 2020 Expert Support Inc. All rights reserved.

There's no one way to approach this high-level organization. It depends on the language and the use of the language. In fact, some API providers list the elements in multiple TOCs, because experienced and inexperienced API users look things up differently.

The ideal API reference provides alternative means for looking up information so that readers with different levels of understanding, different perspectives, different questions, and different needs can still find what they need quickly. Whatever primary scheme is used to organize the reference, an ideal API reference always provides at least one other approach and may provide several others.

Add Hyperlinks Everywhere

Hyperlinking is critical for an ideal API reference. The Angular API Reference for `AnimationMetaData` (Figure 5) provides links for every API element listed on the page. There are 16 links just in this section.

These links are more than just a good idea. If the API user sees something that they need to know more about, the writer can't expect them to go back to the TOC to look for it. In the Angular API Reference, they just need to click the link. And, from our perspective, in the best case, the link opens in another window so the user can have both topics open at the same time.

AnimationMetadata INTERFACE

Base for animation data structures.

```
interface AnimationMetadata {
  type: AnimationMetadataType
}
```

Child interfaces

- AnimationAnimateChildMetadata
- AnimationAnimateMetadata
- AnimationAnimateRefMetadata
- AnimationGroupMetadata
- AnimationKeyframesSequenceMetadata
- AnimationQueryMetadata
- AnimationReferenceMetadata
- AnimationSequenceMetadata
- AnimationStaggerMetadata
- AnimationStateMetadata
- AnimationStyleMetadata
- AnimationTransitionMetadata
- AnimationTriggerMetadata

Links

Properties

Property	Description
type: AnimationMetadataType	

Figure 5. Hyperlinks in an online API reference

Use Names and Terminology that Assist Search

When API languages are complex, there are often thousands of pages associated with the API reference, one for each element, plus a bunch of other related information. The API reference content itself must help readers find specific information to answer their questions quickly.

Memorable and logical naming conventions help readers remember the right terms to use for searches. In the Angular API Reference, nearly all the functions associated with a different functionality start with a keyword for that functionality. For example, nearly all the functions associated with animations, start with Animate. This convention gives the API user a place to start when looking for a function.

Employ other Useful Structures/Documents for Organizing Content

Beyond basic search functionality and the overall API reference TOC structure, there are other meta-documents or meta-pages that can really help the API user.

API quick reference—It's surprising how often API users just need to look up the name of an element and its syntax. An API quick reference that lists the element name, a single line of syntax, and a very short description in a table can be very supportive to API users. Many API users keep this page open on their desktops. Several API providers have even created API quick reference wall charts, which are popular among API users.

© 2020 Expert Support Inc. All rights reserved.

NgModules	`import { NgModule } from '@angular/core';`
`@NgModule({ declarations: ..., imports: ..., exports: ..., providers: ..., bootstrap: ...}) class MyModule {}`	Defines a module that contains components, directives, pipes, and providers.
`declarations: [MyRedComponent, MyBlueComponent, MyDatePipe]`	List of components, directives, and pipes that belong to this module.
`imports: [BrowserModule, SomeOtherModule]`	List of modules to import into this module. Everything from the imported modules is available to `declarations` of this module.
`exports: [MyRedComponent, MyDatePipe]`	List of components, directives, and pipes visible to modules that import this module.
`providers: [MyService, { provide: ... }]`	List of dependency injection providers visible both to the contents of this module and to importers of this module.
`entryComponents: [SomeComponent, OtherComponent]`	List of components not referenced in any reachable template, for example dynamically created from code.
`bootstrap: [MyAppComponent]`	List of components to bootstrap when this module is bootstrapped.

Figure 6. A small part of the Angular Quick Reference

Glossary—Beyond establishing terms, a glossary should provide links to related API elements that help the less experienced API user.

Index (for print or PDF delivery)—Sometimes indexes are really helpful. With the use of search on web pages, indexes are rarely needed; however, if an organization still provides paper versions, an index is critical, and if an organization provides PDFs, it's still often worth the effort.

The differing styles of organization inherent in each type of reference provide a variety of ways for readers to find information: alphabetic in the TOC and glossary, by category in the lists, and by actions or operations in the index.

When deciding what to include in a TOC, glossary, quick reference, or index, it's important to come from the API user's perspective. What tasks can a programmer perform with this API? What information do they need to accomplish a particular task? How are they going to look for it?

Things to Remember

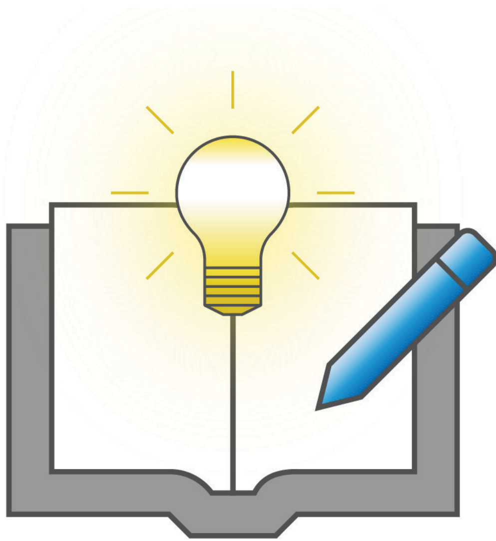
The cornerstone of an effective API documentation suite is the API reference. An organization with a small, simple API might not need an ideal API reference, because something basic is sufficient. However, if an organization has a large, complex API and the reference doesn't meet the qualities of the *ideal API reference*, their API users will likely suffer and might be unsuccessful using the API. The final result is that no matter how great it is, the API won't get adopted the way the organization had hoped.

There's a strong relationship between good API design and good API documentation. For best results, create the documentation plan and involve an information architect (or senior technical writer) early in the API development cycle. This strategy helps the API design team create a better API that's not only easier to document, but also easier to understand, use, and maintain.

Many of the techniques that make an API reference clear, complete, and easy to use should be considered in the design of the API itself. As API architects and designers work with information architects and experienced writers to create excellent documentation, they typically uncover inconsistencies of naming or usage that can cause significant confusion and make learning difficult. Things that are hard to describe are also hard to use, and identifying those pain points early benefits everyone.

Good API documentation doesn't write itself. It takes teamwork consisting of the API design team and trained technical writers that know how to give it the qualities of the ideal API documentation suite. In addition, an ideal API reference isn't easy to write. Those writers who can master the ideas presented here, also need to master the tools and techniques that make these projects successful.

© 2020 Expert Support Inc. All rights reserved.



What's Next

We hope this paper helps you understand more about the elements needed to create a high-quality API reference for your API. Let us know if you found this article useful, and what you particularly liked about it as well as where you'd like to see more explanation or assistance. We expect to update this document with additional information, and we'd love to hear from you so we can focus our efforts where they might matter most.

This paper is the first in a series that will cover, in time, how to evaluate and effectively produce technical communications for software developers, systems administrators, end users, and others. If you have specific challenges or ideas for how we should prioritize these projects, we'd love to hear from you.

And of course, if you need help with specific technical communications projects or staffing, please do let us know. We'd love to explore how we might support your efforts directly.

This article is the result of a collaborative effort among Expert Support Staff, with significant contributions from these Senior Technical Writers: Judy Bogart, Denny Brown, Jan Clayton, Ellen Levy Finch, Carli Scott, and Eric Wenburg. As such, this work represents the collective expertise of many of the world's best technical writers assembled at Expert Support, and working for demanding clients in Silicon Valley over the last several decades.

© 2020 Expert Support Inc. All rights reserved.