# GraphQL Specification Contribution Guide

GraphQL is still an evolving language. This repository contains the

specification text as well as Pull Requests with suggested improvements and

contributions.

Contributions that do not change the interpretation of the spec but instead

improve legibility, fix editorial errors, clear up ambiguity and improve

examples are encouraged and are often merged by a spec editor with

little process.

However, contributions that \_do\_ meaningfully change the interpretation of the

spec must follow an RFC (Request For Comments) process led by a \*champion\*

through a series of \*stages\* intended to improve \*visibility\*, allow for

\*discussion\* to reach the best solution, and arrive at \*consensus\*. This process

becomes ever more important as GraphQL's community broadens.

When proposing or weighing-in on any issue or pull request, consider the

[Code of Conduct](https://github.com/graphql/foundation/blob/master/CODE-OF-CONDUCT.md)

to better understand expected and unacceptable behavior.

## Contributing to GraphQL Libraries

A common point of confusion for those who wish to contribute to GraphQL is where

to start. In fact, you may have found yourself here after attempting to make an

improvement to a GraphQL library. Should a new addition be made to the GraphQL

spec first or a GraphQL library first? Admittedly, this can become a bit of a

[chicken-or-egg](https://en.wikipedia.org/wiki/Chicken\_or\_the\_egg) dilemma.

GraphQL libraries seek to be "spec compliant", which means they discourage

changes that cause them to behave differently from the spec as written. However,

they also encourage pull requests for changes that accompany an RFC \*proposal\*

or RFC \*draft\*. In fact, a spec contribution RFC won't be \*accepted\* until it

has experience being implemented in a GraphQL library.

To allow a library to remain spec compliant while also implementing \*proposals\*

and \*drafts\*, the library's maintainers may request that these new features are

disabled by default with opt-in option flags or they may simply wait to merge a

well-tested pull request until the spec proposal is \*accepted\*.

## Guiding Principles

GraphQL's evolution is guided by a few principles. Suggested contributions

should use these principles to guide the details of an RFC and decisions to

move forward. See editor Lee Byron talk about

[guiding principles at GraphQL Europe 2017](https://youtu.be/mePT9MNTM98?t=17m9s).

\* \*\*Backwards compatibility\*\*

Once a query is written, it should always mean the same thing and return the

same shaped result. Future changes should not change the meaning of existing

schema or queries or in any other way cause an existing compliant GraphQL

service to become non-compliant for prior versions of the spec.

\* \*\*Performance is a feature\*\*

GraphQL typically avoids syntax or behaviors that could jeopardize runtime

efficiency, or that make demands of GraphQL services which cannot efficiently

be fulfilled.

\* \*\*Favor no change\*\*

As GraphQL is implemented in over a dozen languages under the collaboration

of hundreds of individuals, incorporating any change has a high cost.

Accordingly, proposed changes must meet a very high bar of added value.

The burden of proof is on the contributor to illustrate this value.

\* \*\*Enable new capabilities motivated by real use cases\*\*

Every change should intend on unlocking a real and reasonable use case. Real

examples are always more compelling than theoretical ones, and common

scenarios are more compelling than rare ones. RFCs should do more than offer

a different way to reach an already achievable outcome.

\* \*\*Simplicity and consistency over expressiveness and terseness\*\*

Plenty of behaviors and patterns found in other languages are intentionally

absent from GraphQL. "Possible but awkward" is often favored over more complex

alternatives. Simplicity (e.g. fewer concepts) is more important than

expressing more sophisticated ideas or writing less.

\* \*\*Preserve option value\*\*

It's hard to know what the future brings; whenever possible, decisions should

be made that allow for more options in the future. Sometimes this is

unintuitive: spec rules often begin more strict than necessary with a future

option to loosen when motivated by a real use case.

\* \*\*Understandability is just as important as correctness\*\*

The GraphQL spec, despite describing technical behavior, is intended to be

read by people. Use natural tone and include motivation and examples.

## RFC Contribution Champions

Contributing to GraphQL requires a lot of dedicated work. To set clear

expectations and provide accountability, each proposed RFC (request for

comments) must have a \*champion\* who is responsible for addressing feedback and

completing next steps. An RFC may have multiple \*champions\*. The spec editors

are not responsible for completing RFCs which lack a \*champion\* (though an

editor may be a \*champion\* for an RFC).

An RFC which does not have a \*champion\* may not progress through stages, and can

become stale. Stale proposals may be picked up by a new \*champion\* or may

be \*rejected\*.

## RFC Contribution Stages

RFCs are guided by a \*champion\* through a series of stages: \*strawman\*,

\*proposal\*, \*draft\*, and \*accepted\* (or \*rejected\*), each of which has suggested

entrance criteria and next steps detailed below. RFCs typically advance one

stage at a time, but may advance multiple stages at a time. Stage

advancements typically occur during

[Working Group](https://github.com/graphql/graphql-wg) meetings, but may also

occur on GitHub.

In general, it's preferable to start with a pull request so that we can best

evaluate the RFC in detail. However, starting with an issue is also permitted if

the full details are not worked out.

All RFCs start as either a \*strawman\* or \*proposal\*.

## Stage 0: \*Strawman\*

An RFC at the \*strawman\* stage captures a described problem or

partially-considered solutions. A \*strawman\* does not need to meet any entrance

criteria. A \*strawman's\* goal is to prove or disprove a problem and guide

discussion towards either rejection or a preferred solution. A \*strawman\* may

be an issue or a pull request (though an illustrative pull request is preferrable).

\*There is no entrance criteria for a Strawman\*

As implied by the name [strawman](https://en.wikipedia.org/wiki/Straw\_man\_proposal),

the goal at this stage is to knock it down (\*reject\*) by considering other

possible related solutions, showing that the motivating problem can be solved

with no change to the specification, or that it is not aligned with the

\*guiding principles\*.

Once determined that the \*strawman\* is compelling, it should seek the entrance

criteria for \*proposal\*.

## Stage 1: \*Proposal\*

An RFC at the \*proposal\* stage is a solution to a problem with enough fidelity

to be discussed in detail. It must be backed by a willing \*champion\*. A

\*proposal\*'s goal is to make a compelling case for acceptance by describing

both the problem and the solution via examples and spec edits. A \*proposal\*

should be a pull request.

\*Entrance criteria:\*

\* Identified \*champion\*

\* Clear explanation of problem and solution

\* Illustrative examples

\* Incomplete spec edits

\* Identification of potential concerns, challenges, and drawbacks

A \*proposal\* is subject to the same discussion as a \*strawman\*: ensuring that it

is well aligned with the \*guiding principles\*, is a problem worth solving, and

is the preferred solution to that problem. A \*champion\* is not expected to have

confidence in every detail at this stage and should instead focus on identifying

and resolving issues and edge-cases. To better understand the technical

ramifications of the \*proposal\*, a \*champion\* is encouraged to implement it in a

GraphQL library.

Most \*proposals\* are expected to evolve or change and may be rejected. Therefore,

it is unwise to rely on a \*proposal\* in a production GraphQL service. GraphQL

libraries \*may\* implement \*proposals\*, though are encouraged to not enable the

\*proposed\* feature without explicit opt-in.

## Stage 2: \*Draft\*

An RFC at the \*draft\* stage is a fully formed solution. There is working group

consensus the problem identified should be solved, and this particular solution

is preferred. A \*draft's\* goal is to precisely and completely describe the

solution and resolve any concerns through library implementations. A \*draft\*

must be a pull request.

\*Entrance criteria:\*

\* Consensus the solution is preferred (typically via Working Group)

\* Resolution of identified concerns and challenges

\* Precisely described with spec edits

\* Compliant implementation in GraphQL.js (might not be merged)

A \*proposal\* becomes a \*draft\* when the set of problems or drawbacks have been

fully considered and accepted or resolved, and the solution is deemed

desirable. A \*draft\*'s goal is to complete final spec edits that are ready to

be merged and implement the \*draft\* in GraphQL libraries along with tests to

gain confidence that the spec text is sufficient.

\*Drafts\* may continue to evolve and change, occasionally dramatically, and are

not guaranteed to be accepted. Therefore, it is unwise to rely on a \*draft\* in a

production GraphQL Service. GraphQL libraries \*should\* implement \*drafts\* to

provide valuable feedback, though are encouraged not to enable the \*draft\*

feature without explicit opt-in when possible.

## Stage 3: \*Accepted\*

An RFC at the \*accepted\* stage is a completed solution. According to a spec

editor it is ready to be merged as-is into the spec document. The RFC is

ready to be deployed in GraphQL libraries. An \*accepted\* RFC must be

implemented in GraphQL.js.

\*Entrance criteria:\*

\* Consensus the solution is complete (via editor or working group)

\* Complete spec edits, including examples and prose

\* Compliant implementation in GraphQL.js (fully tested and merged or ready to merge)

A \*draft\* is \*accepted\* when the working group or editor has been convinced via

implementations and tests that it appropriately handles all edge cases; that the

spec changes not only precisely describe the new syntax and semantics but

include sufficient motivating prose and examples; and that the RFC includes

edits to any other affected areas of the spec. Once \*accepted\*, its \*champion\*

should encourage adoption of the RFC by opening issues or pull requests on other

popular GraphQL libraries.

An \*accepted\* RFC is merged into the GraphQL spec's master branch by an editor

and will be included in the next released revision.

## Stage X: \*Rejected\*

An RFC may be \*rejected\* at any point and for any reason. Most rejections occur

when a \*strawman\* is proven to be unnecessary, is misaligned with the \*guiding

principles\*, or fails to meet the entrance criteria to become a \*proposal\*.

A \*proposal\* may become \*rejected\* for similar reasons as well as if it fails to

reach consensus or loses the confidence of its \*champion\*. Likewise a \*draft\*

may encounter unforeseen issues during implementations which cause it to lose

consensus or the confidence of its \*champion\*.

RFCs which have lost a \*champion\* will not be \*rejected\* immediately, but may

become \*rejected\* if they fail to attract a new \*champion\*.

Once \*rejected\*, an RFC will typically not be reconsidered. Reconsideration is

possible if a \*champion\* believes the original reason for rejection no longer

applies due to new circumstances or new evidence.