When we add a new feature to Sass, we want to make sure the feature is

well-designed, clearly specified, feasible to implement, and that it meets the

use-cases it's designed for. Although most features should follow the [full

process][], very small features can follow the [fast-track process][] instead.

[full process]: #process

[fast-track process]: #fast-track

## Process

The process for adding a new feature works as follows:

1. The feature is informally discussed on [the issue tracker][]. Most new

features come directly from use-cases brought up by Sass users, or new CSS

syntax that Sass needs to support. Once the Sass team has agreed that a

feature is desirable, it's marked as [Planned][] and can move to step 2.

[the issue tracker]: https://github.com/sass/sass/issues

[Planned]: https://github.com/sass/sass/labels/Planned

2. A formal proposal is written for the feature, following the format [outlined

below](#proposal). This proposal is sent as a pull request, where the Sass

team will discuss its specifics with the author. If/when everyone agrees on a

first draft, the pull request will be accepted and the feature moves to step

3.

Step 2 is also where issues are opened for each individual implementation to

add the feature. These issues should link to the feature's main issue in the

[sass/sass][] issue tracker, and that issue should link back to the

implementation issues.

[sass/sass]: https://github.com/sass/sass

3. Public comments are solicited for the feature, usually via a tweet from

[@SassCSS][]. If the feature is big enough, a blog post soliciting feedback

may also be written. Then we await comments and iterate on feedback for an

amount of time that varies based on the size of the feature and the amount of

feedback received.

[@SassCSS]: https://twitter.com/SassCSS

As the proposal is updated based on feedback, its draft number should be

increased according to the [versioning policy][] and changes should be logged

in a changelog file named `<proposal>.changes.md`. Once enough time has

elapsed and the Sass team is satisfied that all feedback is addressed, the

feature moves to step 4.

[versioning policy]: README.md#versioning-policy

4. The proposal is marked as accepted and moved into [the `accepted/`

directory][]. \*This doesn't mean that the proposal is immutable\*, but it does

mean that no major changes to its semantics are expected. At this point, it's

time to write [specs][] for the new feature, in tandem with implementing it

in [Dart Sass][] (since it's the reference implementation). Writing the specs

alongside an implementation helps ensure that the specs are accurate and

sensible, and that the implementation is correct.

[the `accepted/` directory]: accepted

[specs]: https://github.com/sass/sass-spec

[Dart Sass]: https://github.com/sass/dart-sass

The new specs should have an `options.yml` file that marks them as TODO for

LibSass, with a reference to its issue for the new feature. For example:

```yaml

---

:todo:

- sass/libsass#2701

```

Once the specs and the implementation are complete, they're sent as pull

requests to [sass-spec][] and [Dart Sass][], respectively. They need to have

special lines in their pull request messages in order to build properly:

[sass-spec]: https://github.com/sass/sass-spec

\* The sass-spec pull request message should include `[skip dart-sass]`. This

will cause it not to run Dart Sass tests, which would otherwise fail

because the implementation of the new feature hasn't landed yet.

\* The Dart Sass pull request's message should link to the sass-spec pull

request (for example, `See sass/sass-spec#1293`). This will cause it to run

against the specs in that pull request and so test your new feature.

Once these pull requests land, the feature moves to step 5.

5. The feature is eventually implemented in LibSass. Once the feature lands in

[the LibSass repo][] it moves to step 6.

[the LibSass repo]: https://github.com/sass/libsass

6. The LibSass devs mark the original issue for the feature as complete for

LibSass and close it out with a comment indicating which LibSass and Node

Sass releases will first support the feature.

## Proposal

A good feature proposal should make it possible for an average Sass user to

understand and discuss the feature and the context around it, and possible for

Sass maintainers to implement consistent and well-defined behavior. The

following outline is designed to make satisfy these needs.

A proposal must include at minimum a Summary and a Syntax \*or\* a Semantics

section. Everything else is optional. Proposals may include additional sections,

or divide a section into sub-sections, as necessary to make it clear and

readable. All proposals should also include tables of contents that link to all

their sections.

Everything in sections that aren't explicitly marked as non-normative should be

construed as part of the specification of the feature. Non-normative notes can

be included inline in normative sections using [blockquotes][].

[blockquotes]: https://daringfireball.net/projects/markdown/syntax#blockquote

See [the `accepted/` directory][] for examples of proposals that have been

accepted.

\* \*\*Background\*\*

This non-normative section describes the broader context for the feature. This

is particularly relevant for changes to existing syntax, and \*especially\* for

backwards-incompatible changes. It should explain Sass's current behavior, the

original reasoning behind that behavior, and why it's insufficient.

See [Plain CSS `min()` and `max()`][min-max background] for a good example of

a Background section.

[min-max background]: https://github.com/sass/sass/blob/master/accepted/min-max.md#background

\* \*\*Summary\*\*

This non-normative section provides a concise, user-friendly summary of the

behavior being proposed. It doesn't need to be fully explicit about every

corner of the feature, it just needs to give users an idea of how it works and

what use-cases it addresses. Code examples are encouraged.

See [Escapes in Identifiers][] for a good example of a Summary section.

[Escapes in Identifiers]: accepted/identifier-escapes.md#summary

\* \*\*Design Decisions\*\*

This sub-section goes into detail about decisions that were made during the

design of the feature. It should describe alternatives that were considered,

and explain why the final decision was made the way it was.

See [Plain CSS `min()` and `max()`][min-max design] for a good example

of a Design Decisions section.

[min-max design]: accepted/min-max.md#design-decisions

\* \*\*Syntax\*\*

This section describes the syntax of the feature being added, if it adds new

syntax to the language. The syntax should be written in [Backus-Naur form][],

with regular expression-style operators and the convention that nonterminals

are written in capitalized camel-case form. For example:

<x><pre>

\*\*MinMaxExpression\*\* ::= CssMinMax | FunctionExpression

\*\*CssMinMax\*\* ::= ('min(' | 'max(') CalcValue (',' CalcValue)\* ')'

\*\*CalcValue\*\* ::= CalcValue (('+' | '-' | '\*' | '/') CalcValue)+

&#32; | '(' CalcValue ')'

&#32; | ('calc(' | 'env(' | 'var(') InterpolatedDeclarationValue ')'

&#32; | CssMinMax

&#32; | Interpolation

&#32; | Number

</pre></x>

[Backus-Naur form]: https://en.wikipedia.org/wiki/Backus%E2%80%93Naur\_form

Syntax definitions can also refer to productions from CSS specs. The proposal

should link to the specs in question.

See [Range-Context Media Features][] for an good example of a Syntax section.

[Range-Context Media Features]: accepted/media-ranges.md

\* \*\*Semantics\*\*

This section describes the runtime behavior of the new feature. It may be

omitted if the feature only has to do with how the stylesheet is parsed. The

semantics section covers everything about how a stylesheet is evaluated,

including how imports are resolved and the behavior of built-in functions.

See [CSS Imports][css-imports semantics] for a good example of a Semantics section.

[css-imports semantics]: accepted/css-imports.md#semantics

\* \*\*Deprecation Process\*\*

All backwards-incompatible features should go through a deprecation process if

at all possible (see [Dart Sass's compatibility policy][]). This section

describes the details of that process, including what code will produce

deprecation warnings and how those warnings will indicate what the user should

do to make their stylesheet forwards-compatible.

[Dart Sass's compatibility policy]: https://github.com/sass/dart-sass#compatibility-policy

See [CSS Imports][css-imports deprecation] for a good example of a Deprecation

Process section.

[css-imports deprecation]: https://github.com/sass/dart-sass#compatibility-policy

## Fast Track

Some features are too small and too unlikely to be controversial to warrant the

full-fledged proposal process. Features like that can be \*fast-tracked\*, a

process that requires less time and less reviewer energy than the normal flow.

A feature is eligible for fast-tracking if it:

\* Is simple enough that it's unlikely to need to change substantially as a

result of review.

\* Modifies an existing specification in the `spec/` directory. It's fair game

for a new spec to be written or ported from [the `accepted/` directory] in

order for a proposal to be fast-tracked, but that must be done before the

proposal can move to step 2.

\* Requires very little modification of the specification and of the

implementation. Ideally a fast-tracked feature requires very little

modification of the sass-spec repo as well, but this may not always be

feasible for features with many small edge cases or that happen to appear in

many specs.

\* Requires no deprecations and introduces no backwards incompatibilities.

The proposal author makes the initial decision about whether or not to

fast-track a feature. However if anyone (whether they're a member of the Sass

team or just a community member) requests that that feature be moved to the full

process, it must be moved so that it can have a full discussion.

The fast-track process works as follows:

1. The feature is informally discussed on [the issue tracker][]. Once the Sass

team has agreed that a feature is desirable, it's marked as [Planned][] and

can move to step 2.

2. Issues are opened for each individual implementation to add the feature.

These issues should link to the feature's main issue in the [sass/sass][]

issue tracker, and that issue should link back to the implementation issues.

Three pull requests are sent out concurrently.

1. A formal proposal is written for the feature as a pull request to this

repository, where the Sass team will discuss its specifics with the

author. \*Unlike the full proposal process\*, this pull request directly

modifies the appropriate spec in `specs/`.

2. A pull request is sent to [sass-spec][] that adds or updates specs for the

new feature. The new specs should have an `options.yml` file that marks

them as TODO for LibSass, with a reference to its issue for the new

feature. For example:

```yaml

---

:todo:

- sass/libsass#2701

```

This pull request message should include `[skip dart-sass]`. This will

cause it not to run Dart Sass tests, which would otherwise fail because

the implementation of the new feature hasn't landed yet.

3. A pull request is sent to [Dart Sass][] that implements the new feature.

This pull request's message should link to the sass-spec pull request (for

example, `See sass/sass-spec#1293`). This will cause it to run against the

specs in that pull request and so test your new feature.

These pull requests should remain open for at least two full workdays to

ensure any interested parties have a chance to comment on them. After that

point, \*and\* after all three pull requests have been approved by reviewers,

they should be landed simultaneously.

3. The feature is eventually implemented in LibSass. Once the feature lands in

[the LibSass repo][] it moves to step 4.

[the LibSass repo]: https://github.com/sass/libsass

4. The LibSass devs mark the original issue for the feature as complete for

LibSass and close it out with a comment indicating which LibSass and Node

Sass releases will first support the feature.