# Managing Change in the Haskell Ecosystem

Draft proposal, Haskell Foundation Stability Working Group, Dec 2021

*Purpose and audience*

The Haskell ecosystem tries to steer a course between uncontrolled change (which makes it hard to use Haskell in production) and stasis. No one wants stasis, so we must instead try to ameliorate the costs of breaking changes.

The [Stability Working Group of the Haskell Foundation](https://docs.google.com/document/d/13tPObTEVhhjYIDvK9JmyLSq29nnX2879eWwpj50hHOE/edit?usp=sharing) seeks to reduce these costs, by improving communication and other measures.

This document is a constructive first step at a plan for doing so. It is very much a draft. We offer it as a way to focus discussion, not as a finished article. If you have other, or better suggestions, please offer them.

## 1. Principles and Motto

The Haskell Foundation Stability Working Group (SWG) aims to be the package maintainer’s friend. The motto should be *first, do no harm*.

The SWG will collect and maintain a central list of breaking changes to key components of the ecosystem and offer a review service for those changes.

The SWG may publish recommendations, but they will be just recommendations. Decisions about whether or how those recommendations are met will be made elsewhere by whatever process is normally used for the relevant components.

2. A Central issue tracker for breaking changes

The central idea we propose is to maintain a central issue tracker for breaking changes in key components. What is a “breaking change”? See Section 4. What is a “key component”? See Section 3.

The idea is that instead of an un-coordinated torrent of changes, Haskell users will see a predictable and organised stream of changes. The very act of recording them centrally will promote coordination; and the cost of doing so will remind authors that these changes impose costs on users.

All breaking changes (as understood by the [PVP](https://pvp.haskell.org/)) to the key components will be listed in a central issue tracker managed by the SWG, cross linking to the project issue tracker and repository as necessary. Each issue will include:

* a description of the change;
* the version of the component where it is planned to debut with an estimated date;
* the reason for the change and the estimated benefits;
* the expected disruption of the change;
* the mitigation resources: migration guides and tools to help.

The aim is to:

* collect all the changes in a single place,
* promote communication between us all about the merits and timing of these changes.
* (where practical) help schedule the change to minimize disruption.

## 3. Decide the Key Components

If the SWG is to help mitigate the costs of changes, we must establish scope: what components of the Haskell ecosystem are in scope? We propose:

* **GHC itself**. GHC forms the foundation of everything we Haskellers do, so the GHC releases will naturally be included. GHC has a really good review process and track record in managed evolution so the SWG should expect to be making minimal impact here, but the SWG will track any disruptive changes to GHC and try to schedule other changes accordingly.

* **Cabal and stack.**
* **Core Libraries**. The Core Libraries are, by definition, very widely used, and on which many other packages depend. We propose to include the Core Libraries in the SWG’s scope, [currently](https://wiki.haskell.org/Core_Libraries_Committee): array, base, deepseq, directory, filepath, mtl, primitive, process, random, stm, template-haskell, unix, vector, Win32.
* **Other libraries that are widely used**, but that are not currently Core. A starting set could include
  + Cabal
  + aeson
  + cassava
  + containers
  + cryptonite
  + hashable
  + haskell-language-server
  + text
  + time
  + unordered-containers

A component will only be included in the scope of the SWG if their maintainers agree – there is no compulsion.

Disruptive changes of interest to packages outside the scope of the SWG can be noted and published to help with general awareness and scheduling even if no review is being carried out on the change.

4. What is a breaking change?

Changes come in all sorts. They vary on three main axes:

* **How likely it is that the change will break a client.** Many changes could in principle break some clients, but are unlikely to. For example, adding a new export to a package could break a client that just happened to use a function with the same name, but one might expect that to be rare. [Here is another example](https://discourse.haskell.org/t/is-adding-hascallstack-a-breaking-change/3696). In contrast, changing the API of a widely used function is much more disruptive.
* **How much work is involved in fixing the break**. Some changes are easy (even trivial) to adapt to; others are more complicated. For example, adapting to a change might entail adding a hiding clause to a module import -- the work of a moment.
* **How stable the changed feature is.** For example, a new language extension in GHC may be advertised (explicitly or implicitly) as experimental, and its details of a new feature in GHC may change in subsequent releases. But that is to be expected, and in any case the feature cannot be widely used so the impact of change is low. In contrast, changes to longer-established features are much more disruptive.

We cannot give hard and fast rules, but we define a breaking change as one that

* Is likely to break many clients, and
* Concerns a stable feature or API

A change that meets these criteria is “breaking”, even if fixing the break is easy; it’s a task that still has to be done by clients. But the description of the change can clearly state how easy it is to fix.

5. Introduce LTS GHC Releases

Certain GHC releases should be designated as Long Term Support (LTS) releases. This won’t affect the GHC release itself but provides a coordination point for other changes in key packages, with disruptive changes targeting these LTS releases. Prior to the targeted LTS release, the old version remains the preferred release for dependent packages.

This would allow:

· disruptive changes to collect into (relatively) widely spaced releases;

· the pressure on package maintainers to be reduced between LTS releases;

· disruptors to get a clear window to phase in breaking changes.

A couple of scenarios have been sketched out in [this GIST](https://gist.github.com/cdornan/ae2d678fadb261e6aea6d303192bf8be) to illustrate how these LTS designations could work to reduce the pressure on package maintainers in dealing with disruptive changes.

6. Summary

In summary,we are proposing that breaking changes to key components adhere to the following policy:

1. All breaking changes should be registered in the SWG issue tracker and reviewed by the SWG.
2. Where practicable the release containing the change should be scheduled for a future LTS release.