Haskell Platform Package Guidelines

Mark Lentczner, HP Release Manager
v1 - January 2013

Preamble

So you, or someone else, thinks your package should be part of the Haskell Platform? Since we want to have a broad range of common functionality in the Platform, there's a good chance it should be!

However, being in the Platform isn't all sugar and roses... Being in the platform carries some responsibilities, and guidelines that are somewhat more constraining than if you just carry on developing the package on your own.

Think of it this way: If your package is in the platform, then you'll have many many more clients of your package than you would otherwise. But, these clients won't have carefully chosen your package from all the others on Hackage, they just got it by default. They are counting on it meeting the aims of the Platform.

Chief among these aims is stability: The Platform exists primarily so that for many tasks, programmers can just get on with their task. As such, from release to release, they want the stuff in the Platform to "just work". They don't want some package in the Platform to have a suddenly wholly refactored API

Another chief aim is that your package, and you, need to play well with others: The Platform is released as a single bundle. We all need to work together to ensure that it all works together, and continues to deliver that "just works" experience.

What follows are som guidelines for maintainers of packages that are part of the Platform. These aren't all hard and fast rules, but they are what I think make a reasonable set of responsibilities of being part of the Platform:

API

The API of a Platform package should be more stable than other packages. On hackage, you can just bump the major version when you overhaul your API, and clients are free to keep using the older version. Not so with Platform packages: Developers will often simply wipe out an old install of the Platform for a newer one, and if your package's API changes wholesale, then the version bump is little solace.

This is why we look for packages that have an established API, validated by client usage, and that have shown stability over time.

APIs can evolve, but with high value on compatibility with the older versions. Packages in the Platform should look to support their current API in the two to four year time frame. They should aim to take a year or more to deprecate a commonly used API before removing it.

Building

Packages must build with cabal. The Platform packages are supplied in one variant only, and so packages with build flags, should have a reasonable, inclusive default.

Packages should generate all documentation via Haddock. It is very difficult to manage the building and bundling of non-Haddock docs. The Platform already has some, and we'd prefer if it didn't have any more.

Tests would be highly appreciated. Preferably integrated into cabal's test functionality. The Platform build will be evolving to run these tests during packaging this year.

Code

Target stable GHC. You should expect that the Platform will be released based on a GHC that is up to 9 months old! Don't depend on the latest exotic stuff. Use only tried-and-true GHC extensions, and prefer those that don't require your clients to use them.

Keep it cross-OS. The Platform is released on Linux, Mac OS X, Windows, and in source form. The aim for it to work agnostically. It is fine if you have per-OS conditional code.

Dependencies

The platform is self-contained: Hence, every package that a Platform package depends on, must also be in the Platform.

On hackage, there is often a pull to break packages up into smaller, separable pieces so that client code can rely on just the parts they need. With the Platform, the tension isn't there: Once some functionality is going into the Platform, it really doesn't matter much if it is one package or three smaller ones (other than issues with non-split libraries).

Non-Haskell dependencies are a serious concern. The Platform does not itself include other libraries like GTK or ImageMagic. Hence, any external library that a package depends on must be generally available on all OSes. If it isn't, it makes extra work for packagers to ensure that the proper OS package dependency is set up... and this isn't always easy.

Maintainers

It takes a functional village. Maintainers of packages in the Platfrom should expect to be involved in the month prior to Platform release (twice a year). Input and communication from maintainers is essential to ensure their package compiles, that we know which version to incorporate, and for last minute tweaks against other packages as needed.

Of course, we all have day-jobs and lives beyond our Haskell code, and those usually take precedence. In order that nothing gets badly blocked, communication and mutual trust are key to ensuring we can work together to get the release out.