Why I'm leaving Elm

by Luke Plant

Posted in: Elm, Web development — April 4, 2020 14:28

Over the past year or so, I've reluctantly come to the conclusion I need to leave Elm and migrate to some other language (most likely Bucklescript via philip2), and I definitely cannot recommend it to anyone else. This post is about my reasons for that, which are mostly about the way in which the leadership behave.

I'm not going to talk about the good points of Elm as a technology. You can read them other places, and they will probably be true. Elm (up to version 0.18, at least) has been a joy to work with, and it will be hard to replace. I should also note "Your Mileage May Vary" etc. It would be entirely possible to use Elm and find it adequate for your needs, and therefore never bump into the things I hit very quickly.

Contents

Aims and audience

This post is for people who are assessing Elm, and may have something of benefit for people in other communities, especially leaders.

If you are a part of the Elm core team, you might want to skip this post for the sake of your mental health. I have tried harder than in the past to be fair and reasoned, but it's not likely to be easy reading.

On the other hand, if people from within the Elm community have had the courage to ask you to read this, then maybe you should. I also think that the decisions and behaviour that I criticise here are actually causing you (the core Elm team) far more stress than is necessary, and changes would be very good for your mental well-being.

Entitlement and expectations

Since this post will mainly be complaints and criticism, I thought it worth addressing whether that is ever appropriate or not in Open Source.

As a current maintainer of a handful of small Open Source projects, and having put in a lot of hours on some bigger ones, I'm well aware of the unwarranted sense of entitlement that many people have when it comes to interacting with Open Source projects. I agree this is a problem, but I don't think that just because you are not being paid by people that they can have zero expectations about how you will behave.

The first reason for this is that general ethical standards (including honesty and fairness) apply even if people are not paying you.

The second is that if you advertise something as Open Source, there is a common set of assumptions about what that means, some of which are explicit in accepted definitions of the term.

The third is that the Golden Rule applies. I think it is safe to say that every producer of Open Source software is also a massive consumer, and the Open Source movement is only possible because there are certain standards of behaviour which are adhered to.

I think the degree to which we are responsible to other people who are not paying us does depend on a number of things. The benefit we derive from running an Open Source project is one factor. The number of people who will be affected by our decisions is another. And the contributions we received from people is a third, whether those contributions are code or not.

In some cases, the authors, maintainers and sponsors of a project stand to gain a lot from the success of that project, especially when the project functions as a platform of some kind. This is why it is usually not hard for authors of thriving languages to find sponsors.

And decisions made in popular languages can damage or sink someone's project, someone's career or even someone's livelihood, for those who have made a gamble on investing in that language.

For example, I'm well aware that my career has been hugely helped by being part of the Django core team, often in ways that don't really feel fair. And changes we make to Django can affect a lot of people, for better or worse. This doesn't mean that people can come along and demand any change they want and we have to do it, but it does mean that we ought to exercise a certain amount of care and responsibility.

If the author(s) of a language think they can do whatever they want because they are providing the code for free, they must remember the army of people who have provided high quality libraries for free, who have done countless hours of bug reporting for free, done beta testing for free, done usability testing for free, — all at high personal risk, especially when they are early adopters — and, by no means least, provided the most effective kind of advertising and marketing entirely for free: personal, word of mouth recommendation and promotion.

This means that even if a single person wrote every line of code of a language's compiler themselves, when it comes to thinking about the project, **they need to think of themselves as stewards and not owners**.

So this post is not about me demanding missing features in Elm (although certain features do affect the story), it's about the wider issues of how you treat people, and the expectations that we can have in these areas.

Some of my comments are on the level of personal preference or technical needs when it comes to what I'm looking for in Open Source projects, and others are definitely on the level of the ethics of how I think people should behave and be treated, both in general and in Open Source projects specifically.

Now, onto my reasons:

I messed up with my interactions with the core team

One big factor in why I'm leaving the Elm community is that I messed up majorly in my interactions with the Elm core team.

At the beginning it wasn't entirely my fault — I was all set to do a positive “Show and Tell” post on Elm Discourse, when the core devs announced a change that would really hurt my project (along with many other people's). So my introduction instead turned into something a lot more confrontational, so I feel slightly a victim of bad timing, though of course I accept responsibility for my own words.

Since then I've not done a whole much better, although I have been trying.

The story was this: I needed to internationalise my app, and, after years of seeing different solutions, particularly through the eyes of Django, and thinking about some of their issues, I found Mozilla's awesome Fluent project, and settled on that as a solution. I also had a look at all the existing Elm solutions I could find, and found them wanting — most of them don't even attempt to solve the problems that Fluent solves.

It turns out that Fluent can be used in a really elegant way in Elm, by compiling FTL files to Elm source code, and I created elm-fluent which implements this (complementing django-ftl and fluent-compiler that I also created for my Python backend).

Like many other Elm projects that need i18n features, elm-fluent needs an official wrapper for Intl (the initial version uses a non-official wrapper, which is impossible from Elm 0.19 onwards). I attempted to get things going regarding an official wrapper, but it went nowhere. The topic of i18n came up again later on the Elm Discourse, and I contributed, but the thread was shut down in what I considered a really unhelpful way. Rather than starting a new thread to argue, I thought building something useful would be a better and more persuasive idea.

But before I was ready with elm-fluent, Elm 0.19 was out with the restriction on native modules that made elm-fluent impossible (until an official wrapper for Intl is released). Rather than announce a project that was effectively dead in the water, only supporting an old Elm release, I started looking for ways round the restrictions.

That led me eventually to some private emails with Richard Feldman. Unfortunately I messed these up too. The discussion was about native modules, and I used some of the arguments you'll find below, but looking back at my email I'm ashamed — I used even less tact and care than I've managed here, and certainly less than was necessary. Although at multiple points I did ascribe good motivations to the Elm core team, I somehow succeeded in persuading Richard that I considered the core team to be bad faith actors, and that was the end of our communication. I apologise to the rest of the Elm community for the way I allowed my frustration to get the better of me and squandered that opportunity.

In the Elm community, your hope of being useful depends on relationships, and not offending people, and I'm pretty sure I've blown that.

So, if you are looking for a reason to dismiss this post as the rantings of someone who just can't get along with people very well, it's right here. However, I have never had these kind of problems with any other Open Source community, so I don't think it is just me.

If you are thinking of getting involved in the Elm community, take a lesson from this. There are other things about the community that make me pretty uncomfortable about being part of it, but if I hadn't messed up here I might have considered persevering a little longer.

Technical limitations

The technical limitation that is really killing me on my usage of Elm is the restriction on native modules. For those not in the know, native modules in Elm allow you to write part of an Elm module in Javascript. This feature is absolutely necessary in Elm and is used by many of the core libraries, and a bunch of other libraries in elm-community.

Previously, there were tight restrictions on native modules uploaded to package.elm-lang.org. But you could work around that for your own needs if necessary using something like elm-github-install. In Elm 0.19, however, the compiler itself limits the feature to certain official libraries — you cannot use it at all in your own projects.

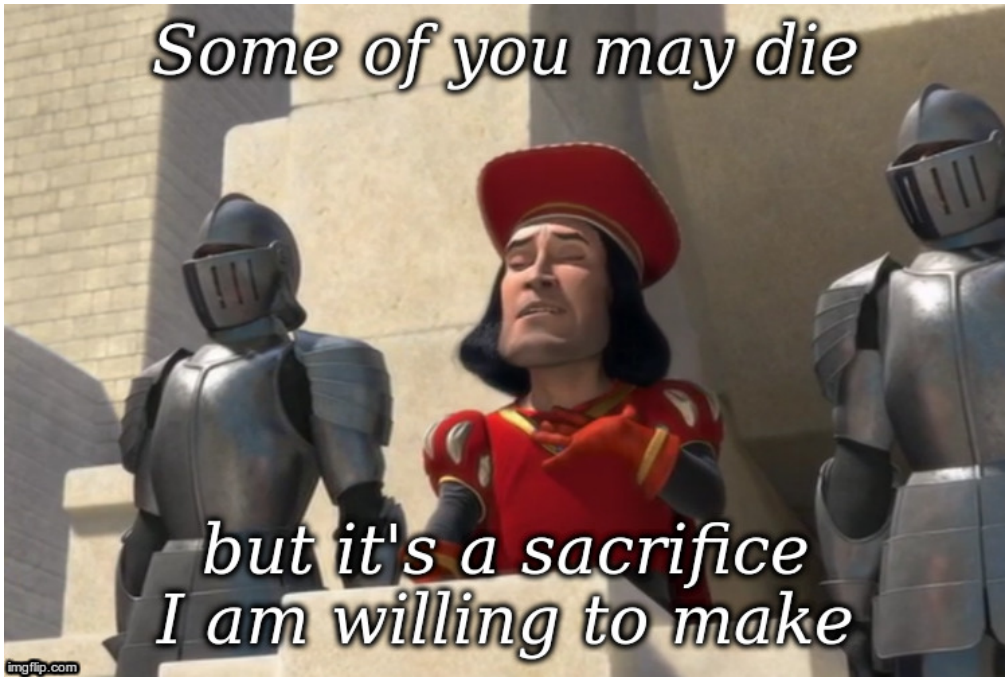
I blogged before about my need for native modules, and since then a bigger thing came up — the need for a Intl wrapper that I mentioned above.

The restriction causes huge problems for lots of other people too. For example, if there is a bug in any core library, or something missing, you just have to wait for the core team to fix it, rather than being able to fix it yourself. You might need a performance fix, which can be done using Javascript but not in Elm (lack of destructive updates makes some things very hard to implement efficiently), and again you will be stuck having to explain to your boss "I know this is possible in Javascript, but we chose Elm and it makes it very hard". The work-arounds are not attractive, and in some cases simply won't work (for example, elm-fluent can't work around the restriction without destroying its design).

In my experience, things like this come up. I didn't know that I was going to need an Intl wrapper, but then it happened. Without the ability to get out of these kind of sticky situations, Elm is not an attractive option to me, and I wouldn't recommend it to other people, on that basis alone.

The most difficult part to accept is that this is crippleware — by which I mean a deliberate limitation that takes additional work by the authors to impose. The feature is still very much there, and very much necessary, just disabled if the compiler can't see that you are contributing to certain projects. For the foreseeable future something like it will be needed, as long as it is possible to distribute core libraries separate from the compiler itself.

I have no doubt that the Elm core team believe this change is in people's best interests. Evan initially expressed a belief that everyone would be able to upgrade to Elm 0.19. However, despite many people indicating that they cannot [1], there has been no reversal of the decision. It appears that they believe long term they are still acting for the best, because forcing this change moves more of the ecosystem to pure Elm code, which Evan believes will benefit the project in the future. There are many holes and problems in this argument, but the attitude of treating existing projects as collateral damage is unacceptable in my book.



Open Source

Elm claims to be Open Source - for example, see the licence on the compiler, and the fact that Evan delivers talks about Elm entitled things like The Hard Parts of Open Source.

But I think this claim is increasingly hard to defend. For me, real Open Source goes beyond a LICENSE file.

Development process

This first point may be more of a personal preference, but for a start, I'd like to see some kind of openness in the development process before I considered something to be Open Source. In Elm, you cannot even see any records of the core team's decision making process, let alone contribute to it (see section below for the latter). In the past, there used to the elm-dev Google group, but it is closed now, and it was never a place where Elm's design was discussed or you could contribute in that way.

Today, there is no forum I can find where I can meaningfully see how Elm's development decisions are made. Discourse is not designed for this kind of interaction at all (e.g. all threads close after 10 days automatically, there is no category for development discussions etc.), and Slack is obviously not appropriate for a multitude of reasons.

Forkability

Another requirement for Open Source in my book is forkability. Having forks *per se* is not necessarily helpful or required, but it should be **possible** to fork or patch it.

As part of my investigations to make it possible for others to try out elm-fluent on Elm 0.19, I considered the possibility of patching Elm (a patch, not a long term fork). I shared this on a thread completely outside the normal Elm forums (on an elm-github-install issue). Richard Feldman

arrived to attempt to shut this down — how he knew about it, or thought he had the right to do this, I have no idea. His comment to me made it clear that I will be *persona non grata* in the Elm community if I patch the Elm compiler.

The earlier versions of his comment were much more strongly worded, for which Richard apologised to me, but the intention of the revised version is the same, and I know where I stand. The core Elm developers deliberately and unnecessarily introduced restrictions into Elm 0.19 in order to stop you using features that they want to reserve for their exclusive use, but if you find and share any workaround for these things that have broken your project, **you** are the one who will be considered an aggressor, the one who has 'attacked' Elm.

Threatening a person with exclusion from a community for attempting to patch the source code is quite antithetical to the spirit of Open Source, as far as I can see. It is the opposite of behaviour you'll see in other Open Source projects. For example, Dropbox created their own Python 2 fork without the smallest thought of needing to apologise for such behaviour, and multiple competing implementations are considered healthy.

An Open Source project should be forkable — the leaders need to accept people's right to do that, and the community also needs to be ready to do it if they decide that the current leadership are not serving their interests. This is a big part of the point of Open Source. For example, people contributed to OpenOffice although it was 'owned' by a company, because the licence meant that if they started throwing their weight around in unwelcome ways (as they did), the community was able to fork it and carry on (as they did). A well run Open Source project will understand that this is an essential feature of Open Source, and welcome it, rather than resort to threats to maintain control.

Discrimination

The Open Source Definition written by OSI, who are generally recognised as some kind of an authority in this area, includes "No Discrimination Against Persons or Groups" and "No Discrimination Against Fields of Endeavor" as items in its list of requirements for an Open Source licence. Technically Elm doesn't fall foul of this, because Elm's licence itself has no restrictions. But to me, it is pretty clear that building discrimination against certain libraries into the compiler (i.e. restriction of features for everything except the core libraries or libraries from certain GitHub organisations, which is what Elm 0.19 attempts to do) is clearly against the spirit of OSI's definition.

In thinking about this, we do have to distinguish between primary and secondary users, whose rights are in tension. For example, if I run some Open Source blogging software, I can blog about any topic I like, and I can also heavily censor/moderate any comment from other users if I so choose. As the primary user, I get to choose how I use the software, and with OSI's definition, a licence is not Open Source if it tries to stop me from doing that on the basis of certain

religious/political views etc. The OSI's definition gives all rights to the primary users, not the secondary users.

So, I am a primary user of the Elm compiler, but a secondary user of package.elm-lang.org. For this reason I have no complaints about the fact that not all users of package.elm-lang.org have the same rights — that some can upload core packages and others cannot.

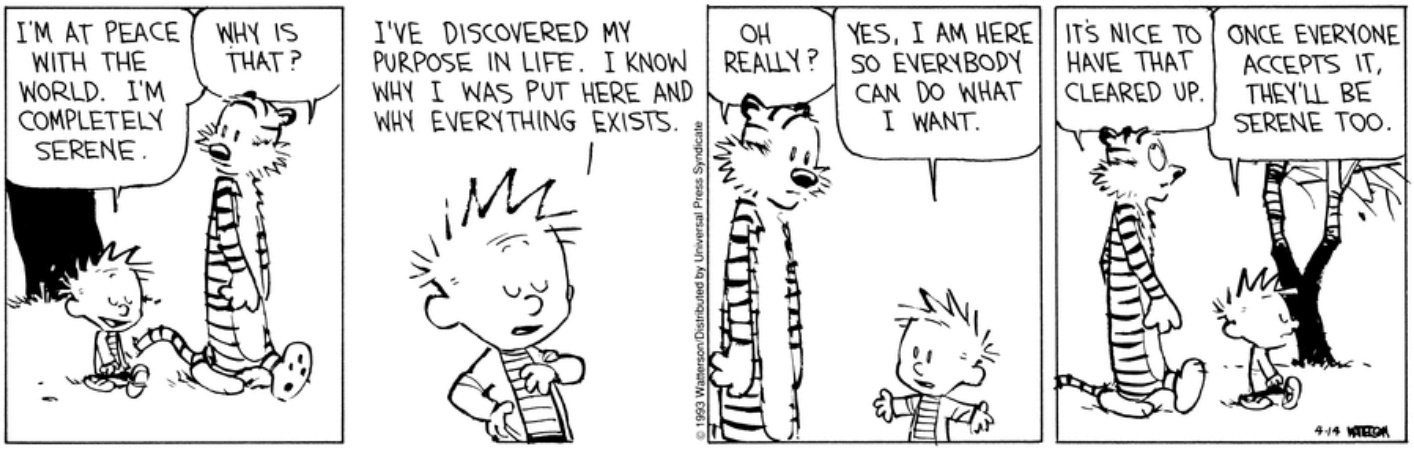
At the same time, we all recognise that when proprietary services give away their client tools as Open Source software, this is really a token gesture, or done just to smooth the wheels for their proprietary service. For example, you would never claim “it's Open Source” as a benefit of using boto (a Python front-end to Amazon Web Services).

So, given that:

- the compiler you get when you install Elm is deliberately crippled (so that you cannot make bindings to browser APIs, for example), despite this being a necessary and present compiler feature,
- the compiler hard codes `https://package.elm-lang.org` as a source of packages, which you have to use to get any functioning program,
- it gives you no ability to configure this source (and in fact you have no ability to turn off connecting to it),
- and you are a secondary user of `package.elm-lang.org` , lacking key rights that other users have,

...then I think it is pretty inescapable that this is an essentially proprietary system. I'm not a Free Software purist type, so I don't think that proprietary software is evil. But if Elm started out as an Open Source project, attracted users to it on that basis, but no longer meets the basic expectations of what those words mean to everyone else, we do have a problem. At the very least, potential users and contributors should be aware of this.

Leadership style



The leadership style in Elm is extremely aggressive and authoritarian.

By that I do not mean impolite or rude. It is almost always very civil. But still ultimately aggressive and controlling.

I gave an example above, which happened even outside the Elm forums. On Elm Discourse itself things are more tightly controlled. Within my first week of starting to contribute I had a post deleted and was blocked from contributing for a week. My offense was that, after the core team had announced their plans to restrict native modules in Elm 0.19, I posted a solution to someone's problem that made use of native modules.

You will find many similar reports across different sites on the internet (and I have more examples below). Sometimes you will find denials or defences from the core developers, but I really don't think you can escape the conclusion that Elm has a very threatening, aggressive leadership style – for example, see the edit at the bottom of this anonymous comment on a Hacker News post:

> There are tons of bugs in 0.18 compiler and 0.18 libraries that will never be fixed. Many of those have / had pull requests by the community members open for more than a year but they were never merged. These are tiny fixes, not 200 lines of code adding a new feature. There's no way to fork a package and apply a patch in 0.19 if the package contains native code. All you can do is report the bug and hope Evan fixes it before your deadline (which ranges from 1 week to 3 years).
>
> Edit: Using a throwaway so I don't get banned from Elm Discourse and /r/elm. I still have production applications using Elm.

In what kind of community is it necessary for people to anonymise their criticism in this way?

Most of the time you won't see this behaviour. If you never have reason to disagree with the leadership, you will likely find them very friendly and helpful. If a leadership style was judged by

how the leaders behave when everyone agrees with them and does what they say, my assessment would be quite different. But of course, that's not how you judge leadership styles.

Why-don't-you-just-ing

I've borrowed this phrase from Evan's talk on The Hard Parts of Open Source.

Evan is complaining about people who make inadequately informed "Why don't you just ..." suggestions that take 20 seconds to make, while it takes 20 hours to explain why the suggestion won't cut it.

Despite knowing how incredibly frustrating this is, Evan and the core Elm team do the same thing to their community, but taken to a whole new level. They do not say "Why don't you just", they say "You must just" — despite not knowing anything about the rest of the project-specific constraints that you are taking into account and couldn't explain — and then they build that opinion into the compiler itself.

Now, if they were simply making design decisions about the language that they felt were right, this would be bearable — we'd just have to disagree about those language features. It is of course the job of language designers to decide which features go in and which get the cut. But in fact they **know** for their own purposes that these solutions do not suffice, and therefore keep the features in, but reserve them for their own use.

Fairness

Distinct from whether you believe in Open Source or not, or whether Elm should be considered Open Source or not, the principle of fairness is an important enough ethical principle that it deserves its own section. The decision to limit certain features to certain people (i.e. GitHub organisations) goes right against this principle.

(I'm deliberately using 'fairness' rather than 'equality', because in some senses it is impossible to treat people equally. If you recognise some people as leaders, or as worth listening to, and others as not, are you treating people equally? If you allow some people to publish standard library code, but not others, is that equality? Clearly not, but you can still treat people fairly without treating them equally).

I'll illustrate with two libraries:

The first is Evan's markdown library. It bundles a minified Javascript library, and uses some kernel code and other internals so that you can use this fast, optimised Markdown parser with an Elm API as a pure function. There is no way that you can characterise this library as merely a "compiler internal" — as Richard Feldman tried to claim was the only valid use case for kernel code.

Since the restriction on kernel code in Elm 0.19, this has been part of `elm-explorations`, and its description says it exists "for historical reasons" — but that is just dodging the issue. You can delete a repo with a few clicks. Why does it exist then? It's not hard to imagine the kind of reasons:

- Evan wanted to write Markdown and convert it to HTML within an Elm app.
- There was a Javascript library marked that did the hard work, and did it well.
- In Elm he wanted a pure function (with signature `String -> Html a`), because there is no reason why it shouldn't be, and that would clearly be the nicest interface. In addition, he wanted to use it in code that would double as demo code, since it is used in elm-lang.org, therefore it needed to clearly demonstrate the Elm architecture, rather than obscure it, as would happen if any other workaround for Javascript interop were used.
- He wanted it soon.
- It would take a long time to rewrite that Javascript code as a pure Elm module.
- Every minute he spent rewriting that working code as pure Elm is time that could be spent doing potentially more valuable things, for himself and for the Elm community.
- And a rewritten version might well be slow and buggy too — the original has been bug fixed and hand-optimized pretty well already, and it is very well known that in some situations pure functional languages can have serious problems producing performant code. Translating optimised JS code into Elm is pretty hard too...
- Maybe Evan didn't actually need to extend it in interesting ways, so for his usages at least its not going to benefit much from an Elm implementation. Therefore someone else, with more interest in an extendable Elm implementation, would probably be in a better position to do the rewrite.
- Of course, there are risks... he might have to fix that bit of native code every time there is a new compiler version...
- But the entire wrapper is pretty small, so even if he had to rewrite it, it's not a huge amount of work (0.15, 0.17, 0.19 part 1, 0.19 part 2).

This is all solid reasoning — and the decision to do this benefits everyone involved. The last thing I want is for Evan to stop doing other important things for Elm so he can rewrite already working code!

What is the problem? The **favouritism and unfairness** that put the needs of the core Elm developers (and their friends — those who can create repos in certain GitHub organisations) above

everyone else. **Because every single reason why Evan may have chosen this course also applies to other developers as well**, with only small modifications. But instead of realising "hmm, sometimes developers are going to benefit from an escape hatch like this, so I shouldn't try to stop them from using it", he locked down the escape hatch for everyone but himself and his friends, and moved the code to `elm-explorations` so that it would continue to work.

In addition to this, other people in the community who have, in the past, chosen to do exactly the same thing as Evan and use kernel code to make an Elm wrapper for a Javascript library, are effectively accused of 'blocking' Elm, of doing what is bad for the community ("Choose not to block. Choose to make a cool thing in Elm"). The hypocrisy here is pretty galling. Ultimately we have to say he is using his position as leader to bully people into writing libraries for his project, even when it is against their interests.

You might argue that we're not all writing demo code to illustrate The Elm Architecture, and that would be right — we are doing something much more important: writing production code. If you think that demo code is more important than production code in terms of needing good architecture, then you have your priorities upside down.

The second library is Evan's elm/parser. This library, in common with many similar libraries from other languages, like Haskell's Parsec, defines a couple of custom operators.

Custom operators are now limited to a few GitHub repos in Elm 0.19 and later. This means that if you want to write your own parsing library, you will have a hard time competing with Evan's.

Why is this important? For one thing a community may well need more than one parsing library. This is not a theoretical concern. As mzero commented:

> Of all the changes in 0.19, this is the one that most hurt my code: I have parser combinator library, and used just two custom operators, for the very reasons that Evan points out in at the top.
>
> Now I learn that elm/parser can, and does, define two operators for parsing, for the same reasons my library had done so. There are indeed times when custom embedded languages with custom operators are worth the mental effort on the programming staff. Parsing is one of them, which Evan acknowledges, and indeed uses in elm/parser.
>
> However, it is not realistic to assume that elm/parser will become the only parsing package we ever need. For one, it only works on String. Parsing over byte arrays is quite common, (and what mine did). Even if elm/parse had been parameterized on the stream type - there are still differing implementation and functionality tradeoffs in

> parsers (backtracking, error tracking, error recovery, etc..) that make different parser libraries useful even they support the same stream type.

So the current policy is bad for the Elm ecosystem, and discriminates against the needs of some users who don't happen to have exactly the same needs as the core Elm team.

But more important to me here is the principle of fairness. It seems that Evan and the core team have forgotten that languages, especially Open Source ones, operate as platforms, and in these platforms contributions from other developers and reputation are critical.

If I am a library author, I need contributions from other developers for my library to be up to scratch. Also, if I can create a popular and well-recognised library, my reputation improves, and I may be able to get sponsors for working on the library, or other work based off my reputation. This is a hugely important part of the currency of Open Source work. (This is perhaps not a sustainable model of funding Open Source, by itself, but that's the way it is right now, and Evan ought to understand this, given that No Red Ink currently sponsors his work on Elm.)

So having a level playing field for library authors is vital. Instead, Evan has reserved some features for himself (and friends). As well as custom operators, `elm/parser` also benefits from kernel code for some performance critical parts. How are you going to compete with this?

Fairness must be a central principle in any Open Source project. But if you choose to model your leadership after a dictatorship, it is critical to pay special attention to it.

If you believe that your own good intentions are enough to stop any abuses of power (which include neglect), then you must be completely ignorant of the human condition, and of why we have concepts like democracy. The fact that both “tyrant” and “despot” originally meant simply “one who rules with absolute power”, without the overtones of cruelty that now exist, ought to be a clue.

It is of course not possible to be perfectly fair, but you can try, and deliberately building discriminatory restrictions into a compiler, as they have done, is going out of your way to be unfair.

Contribution process


Hi! I am Princess Unikitty,
and I welcome you all to Cloud Cuckoo Land!
So there are no signs on anything.
How does anyone know what not to do?
Here in Cloud Cuckoo Land, there are no rules:
There's no government, no baby sitters, no bedtimes,
no frowny faces,
no bushy mustaches,
and no negativity of any kind.
You just said the word "no" like a thousand times.
And there's also no consistency.

Elm has no defined process for contributing to it. The closest you will find is this post on "Building Trust: What Has Worked" by Richard Feldman. This is linked from the official Elm web page in the section regarding contributing to Elm itself.

Instead of a process, you are told you need to build trust with the core developers (which I've spectacularly failed at). "Elm is built on relationships".

I can understand why people think this sounds nice, but in practice it leads to an awful lot of frustration.

Even worse, it means that the top tiers of the community will quickly become an Old Boy's network. This is the opposite of the kind of community that I've enjoyed elsewhere in the Open Source world.

Other Open Source projects have recognised that having a clearly documented contribution process is a good thing, and strive to keep that process working well. On GitHub this has standardised into a `CONTRIBUTING.md` file. Sarah Dresner has an excellent guide on contributing to Open Source projects, but virtually nothing in it applies to Elm.

The core Elm developers have rejected the accumulated wisdom of all these projects and instead put huge walls around contributing. My experience is that these walls, while blocking many people, are especially effective at blocking people who are already in minorities in technology communities. People who are already struggling to find their way in — for example due to being a non-native English speaker, or feeling in a minority for some other reason — will find the Elm contribution non-process an impossible barrier. **In reality there always are processes and rules** (see later), **it's just that the Elm leadership are blind to them. And unwritten rules are a perfect way to discriminate, knowingly or unknowingly.**

All the while, the core team have effectively absolved themselves of any responsibility to potential contributors — where there is no process other than “relationships”, any failure of this process can be blamed on other people failing to build those relationships.

I completely agree that relationships are key to Open Source projects, but relationships are not in themselves a process. In almost every other sphere of life (trading, government, starting a business etc.) if the only answer to the question of how to do anything was “relationships” — in other words, “it's all about who you know” — you would not be optimistic about the ethics of how things were operating.

Meta-process

While I think the Elm contribution non-process is bad, the meta-process — i.e. the “how can we improve the contribution process” conversation — is non-existent, in terms of any community involvement.

Richard's post on Building Trust ends with a “Seeking feedback” section:

> I know people have many opinions on this subject, but I’m not looking for feedback on whether folks agree with this approach of doing things, or for that matter alternative ideas for how things could be done. The feedback I’m looking for is around communication - specifically:
>
> - Which parts of the observations or advice stood out as unclear?

- What's a good place to communicate this, such that it can be discovered by folks who want to to build something that requires this sort of collaboration? (If elm-lang.org or package.elm-lang.org seems best to you, where on those sites would you put it such that beginners don't get sidetracked on it?)

In other words, the core team are very interested in ensuring that you have understood what they have to say about this, but have no interest in whatever you might want to say about it. The rest of the core team's communication sends the same message — there is no meta-process; any comments or ideas from the community about how things could be better are not wanted or appreciated.

A bad process is fixable if there is meta-process, but there is none in Elm. This is in fact why I'm forced to write all of this in a blog post — the Elm core team have clearly communicated that there are no channels within Elm spaces for us to talk about these things.

If I understand rightly, the Elm leadership would agree that there is no community meta-process at all — there isn't intended to be. The leadership believe that not providing any clear process for normal community members to contribute to Elm, or any forum to discuss this, is in the community's best interests, because they will not get benefit from discussion. And I believe this is hubris. That's where we differ.

I've made plenty of mistakes in software development, both on human and technical levels, and no doubt will continue to make them. Maybe writing this blog post is one of them. But hopefully there are still open channels for people to tell me about them — the most dangerous thing you can do is to surround yourself with an atmosphere that shuts down all criticism or suggestions for improvement. And that is exactly the atmosphere that exists in official Elm spaces, as far as I can see.

Communication

I find certain elements about how the core Elm team communicates very hard to stomach.

For example, take for instance Evan's blog post about Elm 0.19. He claims:

> **Every package on package.elm-lang.org is written entirely in Elm**. That means we have a 100% guarantee that there is nothing weird going on in any of your dependencies, so we can cut it up just as easily as your application. If packages contained arbitrary JavaScript code, we would inherit all the same optimization challenges and have to be more conservative.

Notice the emphasis, which is original. And this claim is repeated further down the page:

> Again, this works across the entire Elm ecosystem because every single package on package.elm-lang.org written completely in Elm.

You only have to click a few times on package.elm-lang.org to find some view source links, and discover that of the 6 "popular packages" listed at the top, 5 of them are partly written in Javascript — core, json, browser, url, http.

I think it is just about possible to defend Evan's false statements here (which I reported to Richard Feldman many months ago) as a kind of marketing hyperbole.

For myself, however, I prefer and would expect technical blog posts to have a higher regard for accuracy, especially when you are talking about something that directly relates to a very controversial decision. When you read posts from the core team, instead of being able to take them as a reliable source of information about how Elm works, it becomes more like deciphering political party propaganda.

Or take Evan's post about the removal of custom operators. He starts with a pretty dubious statement:

> Elm 0.19 removes the syntax for user-defined operators.

What he means is that the syntax is still very much there, and very much used by some libraries, but not available for most people. Leaving that aside, let's look at one angle of what follows — those who want to define custom operators for use in maths. Here are some excerpts:

> **Usage in Packages**
>
> Elm has had this feature for a while, so I studied how it has been used so far in packages.
>
> ...<snip>...
>
> 3. Math Operators - Some packages are for math. Vector math. Matrix Math. They generally use operators like `|+|` or `|*|` that match the cultural norms. This is not as common as I would have hoped actually! More math!
>
> ...<snip>...
>
> **Root Design Goal**
>
> Each of these categories stems from a reasonable design goal. I will try to outline the design goal, and then point a path towards acheiving that goal in a nicer way:

...<snip>...

3. I want to do math! I really like this goal. I think languages like Julia have done an excellent job at overloading `+` and `*` in a reasonable way. Their approach is really lovely, but we would have to lose Elm's type system to match them. Point being, rather than making `|+|` and `|*|` as a stopgap, perhaps it is possible to think about the broader question in a comprehensive way. Should there be a way to overload `+` for vector and matrix math? How would that work? How would you multiply a vector by a scalar? Perhaps the best design is to restore user-defined operators for bracketed math operations like `|+|` and `|-|` with certain types? Or maybe it can just be done in a really nice way with a library. Worth exploring!

Do you see how enthusiastic Evan is about mathematics? Look at all those exclamation marks! And look at all those question marks — can you see how open he is to the community and their ideas?

But what actually just happened here?

If you are trying to do maths with Elm, you face some significant problems. First, with only some very basic operator overloading baked into the language (mathematics operators can work with Int or Float types), if you want to do other polymorphic operations using maths Elm isn't much fun. In addition, like most pure functional languages, it presents significant challenges when trying to do high performance work, but Elm is worse than most because there are no escape hatches, and the compiler is still in its infancy when it comes to optimisations (like eliminating intermediate data structures). With Elm 0.18, there were a couple of plus points:

- Although there was no stable interface for importing functions written in Javascript (or compile-to-Javascript languages), there was the unstable interface known as native modules.
- You could write custom operators to make maths-y code at least look a bit nicer and be more readable.

Both of these have been taken away (from you, not from him). Evan says at the top he is going to "point a path towards achieving that goal in a nicer way", but all he actually does for these users is leave them with a bunch of questions, and suggestions that they can do nothing about. "Worth exploring!" sounds like an invitation to explore, but once you've understood the context, Evan is the only person who is actually empowered to do anything.

All of this just leads to the feeling that you are being gas-lighted. Do you think that Evan just added a completely unnecessary restriction that made your life much harder, taking away genuinely useful things for people like you trying to do mathematics, without any compensating improvements? You

must be imagining things — he **likes** math, he's **encouraging** it. “More math!” he said, it's there in black and white! With lots of friendly exclamation marks too!

An analogy

In evaluating the ethical issues I have, I find an analogy to government helpful.

Suppose you have a leader whose job it is to make policies for the people. Whether this country functions as a democracy or not, you cannot expect that everyone in the country will have equal access to the leader. It would be neither desirable nor possible for everyone to be able to influence the leader to the same degree — a trusted advisor or even a spouse is inevitably going to have more influence.

In the Elm community, they recognise this by talking about the need for relationships. They go further, however — they pretty much make it a **policy** that the leader will only listen to a very small group of people. To me, that is pretty unwise, but let's say we can live with that for now.

Suppose, now, the leader is a dictator, and they make laws which give preferential treatment to a group of people, like tax breaks. If we are looking for fairness in this government, such laws are immediately suspect — but could perhaps be justified if the preferential treatment is justified on the basis of the kind of work being done, and not simply certain people (e.g. everyone in the food industry gets certain tax breaks because of some special need in the country, and we can objectively decide who will get the tax break). But if the special treatment is limited to the leader and their friends, then we have definitely lost the plot ethically.

The leaders in our hypothetical dictatorship already have special privilege by virtue of being lawmakers. The laws of the country, along with services provided by the government etc. (which correspond to the source code produced in this analogy) should apply equally to everyone, and whether it is a dictatorship or democracy this rule applies if we believe in fairness.

The beauty of Open Source software is the way in which it can benefit anyone. My human relationships as a software author are inherently limited by human nature — I can't build a relationship of trust with everyone. But the source code I produce does **not** have those limitations. The unavoidable inequality intrinsic to the necessity of human relationships in a leadership structure cannot be used as an excuse for deliberately propagating unfairness by adding discriminatory restrictions into things I produce for people outside of that structure.

Worked example - Intl

To see what all of this looks like in practice, take the issue of needing a wrapper for Intl, which I've mentioned a couple of times above.

This has been a long standing problem for Elm users. Before Elm 0.19 it was solved by being able to write kernel code yourself to wrap `Intl` yourself, or use elm-intl and install it with `elm-github-install`.

Several people tried to do something about this:

- kirchner opened a thread in December 2017, but got no reply from core devs AFAICS.
- I posted about my need for Intl bindings in May 2018. I got a reply from Evan, but he didn't respond to my second post offering to help.
- In July 2018, toastal opened a thread about i18n/l10n, where we eventually learned that the core team have been looking at the issue.

Unfortunately, Luke Westby's response illustrated so many of the things that are wrong with the Elm leadership. I'll quote most of it below:

> toastal:
>
> > what steps are required to get this on the radar of the Elm dev team
>
> It's already on our radar.
>
> toastal:
>
> > it seems many seem to agree that leveraging the browser's Intl feature is the right thing to do
>
> So do we.
>
> toastal:
>
> > a wrapped implementation
>
> Making a wrapper for some browser API and posting it is not the right way to propose something for the platform. We looked at https://github.com/vanwagonet/elm-intl and quickly decided not to consider it further because it just wraps the Intl API instead of serving to explain what an ideal Elm API that uses Intl could be. We think this is a pretty hard question in its own right considering how much stuff Intl actually does.
>
> toastal:
>
> > v0.19 is what is going to kill the current solution (3rd party native code)

> It seems to me like this thread is really about how to do something in 0.19 that uses Native modules. The answer is to use ports or custom elements. This is going to happen for other people's programs and other repos that distribute third party Native code. Intl isn't special in that regard, nor is it uniquely poorly suited for ports and custom elements.
>
> If you want to help the process along while we finish 0.19 the first step is always a literature review. Find ways that other languages and platforms address the specific i18n use cases that you have in mind (Intl does a lot of stuff that we consider distinct APIs), present what you learned and share your sources.

Then immediately after he posted:

> This thread is getting long and fractious. Let's do this:
>
> - Make new threads for specific problems with converting i18n stuff with Intl to use ports and custom elements.
> - Make a new thread for the fluent stuff that @spookylukey is working on, if that's what @spookylukey wants to do. It's definitely very interesting to me!
> - Make a new thread if @toastal or anyone else wants to complete a review of prior work on the subject of i18n APIs that might use Intl in an Elm implementation. Feel free to find me in Slack or send me DMs here if you have questions about what the final product might look like.

And closed the thread, blocking anyone from replying to his comments.

The first major problem I have is the **arrogance** demonstrated. (I'm not picking on Luke Westby here — he is merely demonstrating the attitude of the core team.)

Despite multiple people having demonstrated experience and expressed willingness to help, including myself, it's pretty clear that the Elm core team have consulted with very few, if any, of these people. There is no sense of "we might just possibly benefit from someone else's expertise".

Then, Luke flatly contradicts the other people in the list. They have stated that ports/custom elements are really not suitable for wrapping `Intl`, and he, in effect, says "Yes they are" — but without any proof, despite it being very clear that the other people in the thread are much more experienced in this matter than he is. He's a core dev, so he doesn't need things like reasons, it seems, nor does he need to ask why other people hold their views. He just Knows Better.

He then effectively accuses the people in the thread of being trouble makers: “It seems to me like this thread is really about how to do something in 0.19 that uses Native modules.” – I mean, how dare you bring up yet another piece of evidence that our previous decisions have caused huge problems with no adequate answers.

The next major problem is that he clearly demonstrates **just how little the core team understand “Intl”**. Pretty much everything he said is wrong-headed. [2]

Regarding **process**, he scolds the people who've contributed so far for using the wrong process (“Making a wrapper for some browser API and posting it is not the right way to propose something for the platform.”), and tells them what the right process is, according to him — a literature review, which as far as I can see is completely impractical [3].

So we discover that **there are indeed rules and processes (there always are), but they are unwritten ones that you are just supposed to know**, and you can expect to be chided for not knowing them.

But the biggest problem is that he **immediately closes the thread**. He gives no-one any chance to respond to any of the specific points he makes. He gives a process which is extremely unlikely to go anywhere fast, and **shuts down any meta-process at all**.

This is an extremely arrogant and aggressive way to interact with the community, and gives community members very few good options if we happen to disagree.

You could choose to be silent (as I did on the Elm discourse board — even I have enough tact to realise that immediately opening a thread entitled “Why Everything You Said In The Last Thread Is Wrong” would have been a bad move); or you can be aggressive and confrontational in return (like I have eventually done in this blog post).

The result of all this is entirely predictable — more than 18 months later (at the time of writing), as far as I can see there has been **zero progress** on this front.

And a further consequence of this is that **non-English developers and end users are discriminated against**, due to the difficulty of formatting numbers and dates in correct ways for non-English locales.

I'm not going to say something silly like “Elm is racist” — most programming languages have a very strong bias towards English in multiple ways, and it is perfectly reasonable to make a programming language use the natural language conventions of the authors by default. Nor do I think we need to feel guilty about the huge advantages this gives first language English speakers — the emergence of a common language is a phenomenon that has occurred throughout history, and it benefits everyone.

However, we ought to be **aware** of those advantages — the fact that we have benefited more than others. And if language authors **go out of their way to build completely unnecessary discriminatory policies** into the compiler itself, despite this going against the whole spirit of Open Source, in ways that give additional special advantages (like access to certain compiler features) to people who already have advantages (like the core team), they cannot be surprised when this makes life even harder for minorities, and **they are responsible for these discriminatory effects**, especially once these things have been pointed out to them.

There is no technical reason or excuse for this level of discrimination against non-English internet users in Elm projects. The browser already provides web developers with CLDR data needed for correctly formatting dates and numbers in different locales, and other localization issues, via the Intl APIs. The Elm compiler already has mechanisms for interfacing with browser APIs, and those mechanisms are not going to disappear. It's perfectly acceptable that there is no Elm standard library for doing this, but not acceptable that the Elm core team should attempt to block you from using already existing compiler mechanisms and browser APIs to provide a solution to users.

In this worked example, I've focused on the need for `Intl` wrappers, because it illustrates most of my previous points. However, the same thing will happen many times over with every group whose needs are not adequately represented in the core team — they will suffer the same unfairness and discrimination, even if it is not easy to put a name on that group or identify them.

Conclusion

I do not think I am asking too much of the Elm leadership. The principles I've laid out here I've seen put into practice in every other Open Source community I've worked in. Even in projects using the BDFL model (such as Django at the beginning), I've never expected or seen the behaviour I've seen from the Elm leadership.

Rather, the Elm leadership have gone out their way to ignore normal behaviour in Open Source projects.

By doing so, in addition to abandoning basic principles of fairness, they have also made more work for themselves — only they are able to fix certain kinds of bugs and missing libraries, they have to make many decisions that they may be entirely unqualified to make, and they have blocked many valuable contributions from many people.

The result is a project that I don't feel inclined to contribute to. With any other Open Source project I've ever been involved in, if I work on some bug fix or feature, I have a very high degree of confidence that my contribution, if accepted, would benefit everyone in the community equally — or at least without any deliberate discrimination. In fact, **this is one of the aspects of Open Source that I love and value the most — being able to share work in a way that benefits everyone, without the risk that someone is going to take those things away and use**

them for just a privileged few. With Elm, I don't have this confidence — the leaders have demonstrated that they don't believe in the principles behind such expectations. That's a huge turn-off for me, and I can only hope that other projects do not look to Elm for inspiration on how to run themselves.

Footnotes

1

If you attempt to say that you can't upgrade to Elm 0.19 on Elm forums, you get shouted down by core developers with why-don't-you-just solutions.

It is of course always **possible**, in some form, to upgrade an Elm 0.18 project to Elm 0.19. But if:

- this involves moving quite a bit of logic out of Elm into Javascript
- or it involves a lot of work that results in degrading your app's architecture
- or the process makes you regret that you chose Elm
- or you find that it is actually easier and results in a better structured app if you port your project to a completely different language

...then this can hardly be counted as positive evidence for the claim that “everyone can upgrade”.

As specific examples of projects that cannot be upgraded, here are a few:

- My 7,000 line project.
- elm-fluent, the i18n tool I created.
- Elm 0.19 broke us
- Dark's frontend - Paul Biggar wrote an Elm-to-ReasonML compiler to be able to migrate away. If you are a startup like Dark, the last thing you really want to do is move to another platform **while your team is still actively developing it**, or have to write a new tool just to do that. If that was the easiest way to keep going, then “we couldn't upgrade to 0.19” is a pretty fair assessment.

2

Perhaps I can illustrate the many problems with Luke's post by rewriting it as if it had been about Math instead of Intl, with some small elaborations:

> Making a wrapper for some browser API and posting it is not the right way to propose something for the platform. We looked at the maths functions in elm-core/Basics.js and quickly decided not to consider it further because they just wrap the Math API instead of serving to explain what an ideal Elm API that uses "maths" could be. We think this is a pretty hard question in its own right considering how much stuff `Math` actually does.
>
> For example, look at the docs for sin:
>
> ---
>
> > Figure out the sine given an angle in radians:
> >
> > ```
> > sin (degrees 30)     == 0.49999999999999994
> > sin (turns (1/12))   == 0.49999999999999994
> > sin (radians (pi/6)) == 0.49999999999999994
> > sin (pi/6)           == 0.49999999999999994
> > ```
>
> ---
>
> This doesn't help you know how to use this function practically in an Elm app.
>
> It seems to me like this thread is really about how to do something in 0.19 that uses Native modules. The answer is to use ports or custom elements. This is going to happen for other people's programs and other repos that distribute third party Native code. `Math` isn't special in that regard, nor is it uniquely poorly suited for ports and custom elements.
>
> If you want to help the process along, the first step is always a literature review. Find ways that other languages and platforms address the specific mathematics use cases that you have in mind (`Math` does a lot of stuff that we consider distinct APIs), present what you learned and share your sources.

It's difficult to know where to start, but let's try.

First of all, you would conclude that the (imaginary) author of these comments doesn't know much about maths. It doesn't inspire confidence, and you probably wouldn't want this person in charge of mathematics APIs in your language.

Probably the biggest problem is the failure to understand that these maths functions are **primitives**. They are so widely used and needed, there is no single use case you should be thinking about.

Of course, at the same time, some of these functions many people will never use at all (when was the last time you used trig functions in a web app?). They are indeed distinct APIs, which the inexperienced will look at with bafflement.

But the people who need them know that they need them, and know how to use them. They don't want then wrapped up in some specific functionality that uses sin/cos/tan to draw triangles etc. — they just want the raw, primitive function.

We don't need to do a literature review to decide how to write the wrapper for `Math.sin`, `Math.cos` etc., because, given the other decisions already made in Elm (regarding how numbers map onto Javascript builtins etc.), there are extremely few sensible options.

Because these functions are basic, pure functions from which you want to build up other things, using Math functions via ports or custom elements would be horrendous, and would cause huge damage to your ability to use the Elm Architecture, because both ports and custom elements have virtually zero ability to compose. It's impossible that the person who wrote the imaginary comment above could have tried to use `Math` in Elm using the methods they described — the contortions would have killed them.

This analogy applies almost exactly to `Intl`.

Given the other decisions about how Elm renders pages (e.g. it at least gives you the option to manually build up HTML DOM fragments), a low-level wrapping of `Intl` that exposes its functions essentially as primitives is the only kind of wrapping that makes sense for Elm as a built-in. Other tools that build on that, such as `elm-fluent`, would then be possible, in the same way that you have things like elm-ui building on elm-html, but you'd need an extremely high level UI library to get rid of the need for exposing low-level wrappers on `Intl`, way higher than the style that Elm supports.

The pure functions in `Intl` make extremely bad candidates for use via ports [4], and would completely destroy use cases like elm-fluent that need to build up more pure functions out of these primitives.

In my tool elm-fluent, I used the wrapper that the core team have dismissed, elm-intl, and, contrary to their uninformed assessment, it was exactly what I wanted — better, in fact, than I imagined. I knew exactly how to use it, because I understand the i18n primitives and how they fit together.

---

[3]

Following on from the analogy above, we can see why the approach of doing literature review that the core team insist on is not really appropriate:

- the topic of localization is huge, just like 'maths'. There are many, many things to consider. Fluent is perhaps one of the most holistic solutions I've seen (combines usage of PluralForms, NumberFormat and DateFormat), but it really covers only one aspect of l10n, and you would still sometimes need access to the underlying primitives it wraps, in addition to the primitives it doesn't wrap at all. No-one could hope to do a literature review on the topic of "localization" and come up with anything helpful.

- It kind of doesn't matter, because we've agreed that we can't "roll our own" due to practical considerations. This means we are very much constrained to the API that `Intl` provides. We can write a more Elm-ish wrapper, but this is exactly what `elm-intl` does — I found it to be ideal for my purposes in elm-fluent, despite not being involved in the design of it at all — and yet the core team have rejected that approach.

- And it's a good thing that we are constrained in this way. The `Intl` modules are modern, recently designed APIs. Look at the people involved and you'll see their level of expertise, which the Elm core team can't compete with. Throwing away their expertise because we think we can do better would be a mistake.

My only conclusion is that the core Elm team have no idea what they are talking about when it comes to the `Intl` APIs. **That is, of course, not a problem, nor anything to be embarrassed about**. There are vast numbers of APIs in the software world; no-one can be an expert on all of them. But if you have a leadership style that consists of secret discussions that no-one outside the group can read, let alone contribute to, and if you ignore the actual experience of people in your community, post diktats to public discussions and then immediately close them down because you think you detected a whiff of dissent, without giving anyone the chance to enlighten you… **then we do have a major problem**.

4

As an example of trying to use a pair of ports for a pure function, let's take the very first sample you find in the Elm Guide. This sample uses `String.fromInt` , which happens to work reasonably well for presenting numbers to English speakers, as does `String.fromFloat` but in many situations these aren't anything close to "good enough" for many people. We want to use Intl.NumberFormat.format, and we are told that ports is the way to do it. (There is also the solution of custom elements; see comment below.)

Original code:

```
import Browser
import Html exposing (Html, button, div, text)
import Html.Events exposing (onClick)

main =
  Browser.sandbox { init = 0, update = update, view = view }

type Msg = Increment | Decrement

update msg model =
  case msg of
    Increment ->
      model + 1

    Decrement ->
      model - 1

view model =
  div []
    [ button [ onClick Decrement ] [ text "-" ]
    , div [] [ text (String.fromInt model) ]
    , button [ onClick Increment ] [ text "+" ]
    ]
```

If you want to experience the pain of actually rewriting this using ports for localizing numbers via Intl.NumberFormat, I will leave that to you, as I couldn't face it myself, but in short, you will find:

- the model will need to store intermediate results which are needed by the view. It will also become redundant (it has de-normalized a piece of information) and be capable of being inconsistent with itself (and in fact it is guaranteed to be at least momentarily inconsistent with itself every time it is updated).
- our update function will need to trigger commands to generate those intermediate results, according to the needs of the view function.
- our view function will now depend on the update function having done part of its job for it.

In other words, the update function and view function have become tightly coupled to each other, and every part of the Elm architecture has been seriously damaged — the model, the update function and the view are now a mess.

Oh, and this isn't just a one-off cost — for each different function from `Intl` you want to use, you have to add all this stuff every time you **use** that function — additional fields to the model etc., or more complicated data structures. Plus, this is a simple, easy example, where inputs and outputs are Elm primitives, and nothing is deeply layered. In a typical app it is going to be much, much worse.

(Yes, a custom element could have worked here, but in many cases it won't, e.g. if you wanted to use a properly localized number in an attribute (such as `aria-label`) of some other element, or you want to use it in an update function. They also work badly for numbers that you want embedded into sentences e.g. if you are trying to create something like elm-fluent or any of the other i18n solutions people have created in the Elm world. And a custom element is a ridiculous amount of work for such a tiny thing.)

If you think `Intl` can be used very well via ports, then you are saying that `String.fromInt` can used very well via ports. In fact, that's what you are forcing the non-English speaking world to do if it wants to use Elm. The very first code sample in the Elm Guide shows just how bad it is.

You may also like: §

- Two experiences with Elm
- Announcement: Django Views - The Right Way
- Test smarter, not harder
- 6 digit OTP for Two Factor Auth (2FA) is brute-forceable in 3 days
- WordPress 4.7.2 post mortem
- A simple password-less, email-only login system
- How to learn Django without installing anything

Comments §

Comments should load when you scroll to here...
