Rollout

The Ultimate Feature Flag Guide

What They Are, How to Use Them, and How to Get Started

INTRO

In this guide, as the title indicates, we're going to talk all about the feature flag. What are feature flags? How do you use them? How do you get started? I'll cover all of this and a lot more besides.

The interesting thing about feature flags is that they're a deceptively simple topic. You can answer the question "what is a feature flag" by writing a few sentences or by writing a book. Depending on the context, either would be appropriate. So to cover all those bases, let's start off as simple as possible and dive progressively into more detail from there.

WHAT ARE FEATURE FLAGS? THE SIMPLEST DEFINITION THAT COULD POSSIBLY WORK

In software, a flag is “one or more bits used to store binary values.” So it’s a Boolean that can either be true or false. And you’d check it with an if statement. In the same context as software, a feature is a chunk of functionality that delivers some kind of value.

Thus, a feature flag, in the simplest terms, is an if statement surrounding some chunk of functionality in your software.

There is, of course, a whole lot more nuance to things than that. Otherwise, I wouldn’t be writing an entire comprehensive guide on the subject. But if you came here looking for a quick definition, that’s it.

FEATURE FLAGS: THE SIMPLEST EXAMPLE THAT COULD POSSIBLY WORK

To drive the simple definition home, let’s take a look at a simple example in pseudo-code below:

```
if(configFile["IsHoliday"] == true) {
  writeGreetingMessage("Happy holidays!");
}
```

We have three pretty basic things going on here, but all of them are important.

1. We’re reaching outside of the running software to a file to get some configuration information.
2. If the configuration information says that it’s a holiday, then we get at the feature in question.
3. The feature in question is to show a holiday greeting.

This is a dead simple feature flag. If it’s some kind of holiday, show a user greeting message wishing them happy holidays. Otherwise, don’t.

FEATURE FLAGS: A SLIGHTLY MORE FORMAL DEFINITION

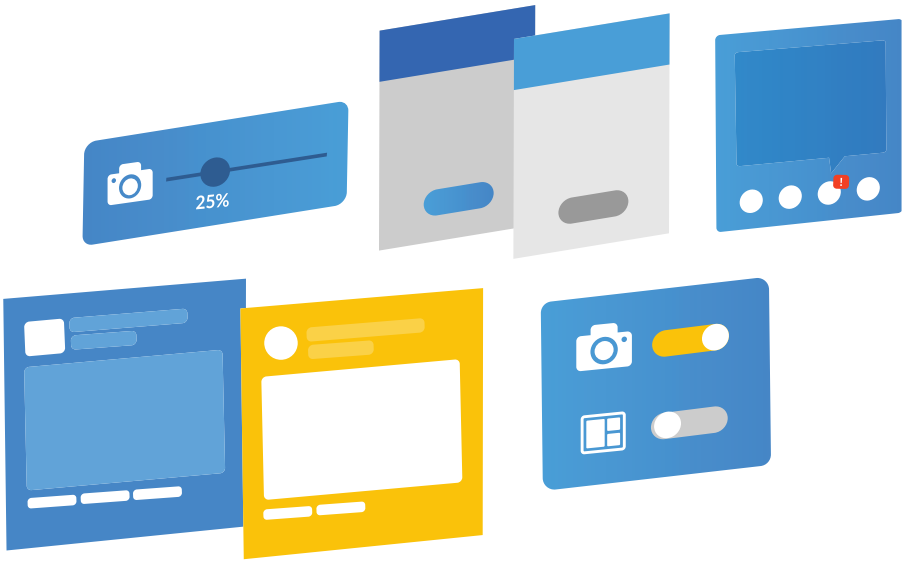
You'll notice that something crept into the example that I didn't mention in the initial, dead-simple definition. I talked about reading something in from a file. To understand why, let's expand on the definition a little.

A feature flag is a way to change your software's functionality without changing and re-deploying your code.

Both definitions are true, but this one generalizes the situation a little more. At its core, a feature flag does mean putting if statements around pockets of source code. But the reasoning for it runs a little deeper. The reasoning is that you want the flexibility to change your software's behavior at runtime.

Going back to our example, that's why the bit about reading from a file was important. If we'd just defined and read from a local variable, we'd have the if statement and the gated functionality, but we wouldn't have the capability to change things on the fly. That capability is critical.

If you think of what we're doing in the example, that makes sense. The idea here is that we've got a piece of software running somewhere. We want to be able, on the fly, to decide that it's a holiday and greet the users accordingly. But you don't want to deploy your code every time there's a holiday and then re-deploy when the holiday is over. That's kind of insane. So you contrive of a way to toggle this functionality on and off without re-deploying.



A FEATURE FLAG BY ANY OTHER NAME: FLIPPERS, TOGGLES, CONTROLS, OH MY

I'm about to go more depth into feature flags, but before I do, it's important to clear something up. You've probably seen other names for feature flags. I know that I've run across some in my travels, such as the following:

- Feature toggles
- Feature flippers
- Feature controls
- Rollout flags

There are probably more besides.

While you might see some nuanced semantic arguments that these are slightly different concepts, for our purposes here, I would use all of these interchangeably. So if you came here looking for information about toggles or flippers, you've come to the right place, even though I'll call them feature flags throughout the post.

FACEBOOK AND PIONEERS OF PRODUCTION FLEXIBILITY

Though it's recently been in the news for less favorable reasons lately, Facebook is perhaps best known in the software world for pioneering deployment techniques. It built a massive social network starting more than a decade ago, and its uptime requirements and scale were such that traditional site maintenance approaches weren't good enough. In other words, even 10 years ago, you never saw this message:

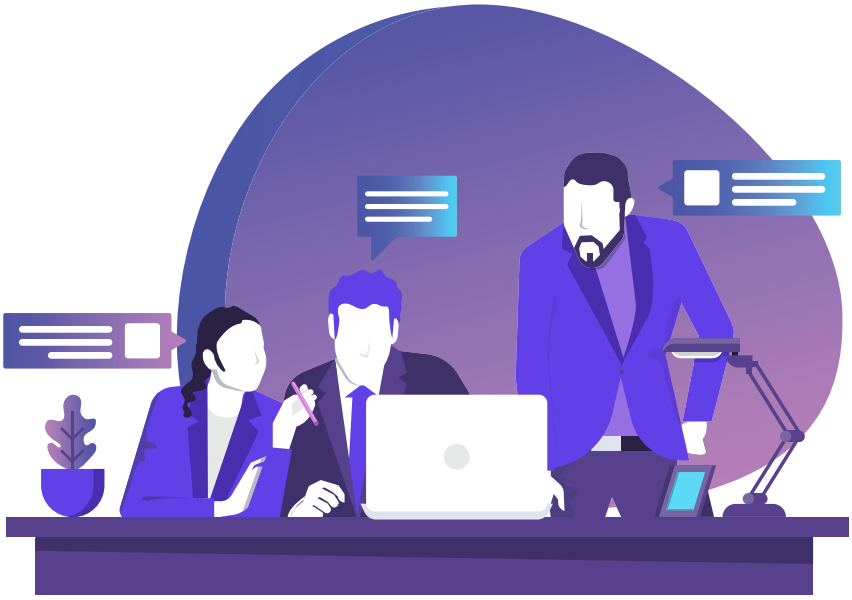
> *Facebook is going to be offline overnight next Thursday so that we can deploy our exciting V3.0 of Facebook!*

Instead, Facebook just quietly rolled out a never-ending stream of updates without fanfare. Day to day, the site changed in subtle ways, adding and refining functionality. But it was never an event.

Behind the scenes, this was a mean feat of engineering. Facebook refined and operationalized deployments to the point that this paper mentions new engineers deploying code to production during their first day or two on the job. Other tech titans developed similar deployment capabilities as well, setting the standard for modern deployment maturity.

Why am I mentioning this in a feature flag guide post? I'm mentioning it because the feature flag was philosophically fundamental to this development.

Feature flags aren't just about toggling random greeting messages. They're part of a much larger important movement in software development. And that's why the definition can be simple.... or really, really complicated.



PRODUCT-FOCUSED USE CASES

Let's talk now about use cases for the feature flag. Although to be accurate, these are really use cases for the broader concept of feature flag management systems, which represent the collection of all feature flags that you use across your application. More on feature flag management in a bit.

Facebook and similar companies didn't realize enormous value out of a random hodgepodge of production flags. They realized enormous value by using feature flags to address these use cases.

Separating Deployment From Feature Roll-out

It might not occur to you immediately, but there are subtly different risks to deploying code and deploying features. For most of us—especially developers that have been in the industry a long time—this seems like a bizarre statement. How can you separate the two? Your features are your code, so it's all the same pile of risk.

Well not really, if you think about it. Deploying code carries what I'll call technical risk. Software might crash or bugs might emerge. Deploying features carries user-related risk. Users might hate the new features, or you might have account management issues, for instance.

With traditional deployments, you just absorb all this risk at once. But feature flag management systems give you the ability to separate these risks, dealing with one at a time. You put the new code into production, see how that goes, and then turn the features on later once they're settled from a technical perspective.

Canary Launches

Speaking of risk mitigation, feature flags let you improve even on the scheme I just mentioned. I talked about deploying the code and then throwing the switch for the new features in one fell swoop. That's fine and good, but you can de-risk even further with a canary release.

With a canary release, you introduce the new code gradually to your user base. The first that see it are the proverbial canaries in the coal mine. If there is an issue, you can find out with only a fraction of the user base seeing it, rather than everyone inundating you at once with support requests. Of course, if everything goes well, you introduce the new functionality to more and more users until everyone has it.

Production Testing

Conventional wisdom has always held that you don't test in production. That's what QA groups, sandbox environments, and everything else internal are for. Those are your dress rehearsals and production is the actual show.

But Facebook, Netflix, and others have turned that on its head as I mentioned earlier. Those companies couldn't possibly recreate their production environments for test, so they have to run QA in production. And in doing so, they've proved that there's an awful lot of value to the activity.

So when using feature flags, you can de-risk deploying functionality whose production behavior is unknown to you. Of course, you always want to test everything you can as early as you can. But it's nice to have options.

Turning Things Off With a Kill Switch

Just as you can use feature flags to increase the number of users that see a feature, you can also go in the opposite direction. You can decrease the number seeing it, including decreasing it immediately to zero with a conceptual kill switch.

Of course, this comes in handy when you've rolled out a misbehaving feature. You can turn it off, let life go back to normal for your users, and then live to fight another day. But it also comes in handy for sunsetting and then decommissioning features as well.

Running Experiments

The last product-focused use case that I'll mention is using feature flags to run production experiments. The easiest one of these to explain is the A/B test.

With an A/B test, the idea is that you go to production with two different versions of something to see which does the best. For instance, perhaps you run an e-commerce site, and you want to see whether a green or red "buy" button results in more likes. You can deploy both buttons and use a feature flag to split your user base in half and see which button performs the best.

You could conceive of all manner of experiments to run like this, for all manner of reasons. The sky is really the limit on how you can use feature flags and the data you gather running experiments to improve your application.

INFRASTRUCTURE AND PROCESS RELATED USE CASES

Feature flag management systems have powerful implications for software products, of course. But they also have uses that are more related to infrastructure and internal process. Let's look at a couple of those.

Migrations

First up, migrations. Are you changing over to using a new version of some web service? Or perhaps you're doing the iconic database migration?

Traditionally, IT organizations in these scenarios tend to make use of the dreaded forklift upgrade. They rewrite large swaths of the code to use the new external dependency, then they do a one-time migration that sure better not fail because a rollback will be painful at best and impossible at worst.

Feature flags can get you away from this nerve-wracking approach. You can just start building calls to the new service or database right into your existing application and deploy them with no fanfare. Then you can discretely test out the new connection at some low-risk time to see if it's working.

By the time you're ready to completely migrate, the code will have been in production for a long time and you'll have already tested it. Flipping the switch will be a non-event.

Developer Collaboration: Feature Branches or Feature Flags?

Finally, there's the software development process issue of feature branches versus feature flags. You can watch a webinar and read extensively about this idea here. But the short form is that these represent two different ways that the software development team can approach collaboration.

Using feature branching, teams work with their own copies of the code and generally merge all changes together less frequently, effectively deferring complexity. Using feature flags, it's easier to turn off features that others are working on, allowing all developers to integrate their code more continuously with the master branch.

Feature branching comes with downsides and risks, so many teams view the use of feature flags as a preferable alternative.

A MORE REAL-WORLD KIND OF EXAMPLE

With an understanding of feature flag use cases, let's now expand our toy example to be slightly more representative of the real world. Imagine an e-commerce application. It's not just going to have a single place to greet users, and it's not going to make the decision to do so based on the contents of some one-off file.

Instead, you'll probably have a construct like this for the top of a page:

```
if(feature.isActive("holiday-greeting")) {
  print("Happy holidays," + user.name + "!");
}
```

And then, perhaps something like this at the bottom:

```
if(feature.isActive("holiday-greeting")) {
  printLink("Click to see more holiday daeals", "~/holidayDeals");
}
```

In other words, when you decide that it's holiday time, it's likely that you're going to have multiple places that behave differently in holiday mode. Maybe you'll have dozens. You're not going to want to define a different feature flag for each of them or even to re-hash the same implementation in each place.

And then there's the determining factor for turning on the feature flag. Do you want the "holiday-greeting" feature to be activated by a manual setting that someone controls? Or do you want it to activate automatically on certain dates at certain times? Or some combination of both?

In the real world, the "how" of feature flag management involves some planning and design. These are not trivial decisions, and they aggregate as your application becomes more complex.

ANATOMY OF A FEATURE FLAG

To understand what's going on here, let's take a look at the anatomy of a feature flag management scheme in a bit more detail. As I said at the outset, you can define it extremely simply, or you can get really complex. These terms will help you understand what's happening at a more complex level.

- **Toggle point**—in our example, each occurrence of a check for feature.isActive("holiday-greeting") represents a single toggle point. Since a feature is rarely just a few linear lines of code, turning a feature on and off can require many, many toggle points.
- **Toggle router**—the feature.isActive(string) method represents the toggle router. A toggle router maps the toggle points to the state of the feature flag. This is how you maintain a coherent, single point of knowledge for the state of the feature across many toggle points.
- **Toggle context**—the toggle context represents contextual information that the router takes into account when computing the feature's state. For instance, I mentioned that a "holiday-greeting" feature might depend on the date. Current date is an example of context. Other examples might include the logged in user, the geolocation information of the user, a referring URL, etc.
- **Toggle configuration**—in addition to ambient context, you can also control the results of the toggle router for a feature on the basis of simple configuration. In our example, you might have an ambient context that turns the holiday greeting on, but you also have the ability to manually turn it off.

HOW TO GET STARTED: IMPLEMENTATION

You might, at this point, be thinking that this seems like over-engineering just to turn one little thing on or off in your application. After all, maybe you just came here looking for the simplest of primers and read on simply for some bonus information and because you found me witty.

Do you really need to understand all this?

The answer is, not at first. But you will before you know it.

Think of the idea of application logging. If you were brand new to programming or writing some kind of toy implementation, it might actually be easier to just to use your language's file API to dump some random text to a file than it would be to install and configure a full-blown logging framework.

But how long would that last? Would you hold out until you had to think of log levels? Different styles of appender? Multi-threading? Sooner or later, it'd become more painful to keep rolling it yourself than it would be to go back and use an established solution.

So it goes with feature flag management. If you're new to it and dabbling, I'd recommending implementing it manually. You'll develop an understanding of how it works and make better decisions later. But if you find yourself back here, looking up these definitions with an eye to rolling your own extensive implementation, don't.

Mature, third-party feature flag management systems exist, and you should use them.

FEATURE FLAGS AND TECHNICAL DEBT

One of the historical knocks on the use of feature flags is that they create technical debt. This is an understandable sentiment since, implemented by hand, they can lead to massive, ad hoc tangles of conditional logic in your codebase. And that can be downright nasty.

This is one of the reasons that I advocate picking a third-party management system. These systems can actually save you from technical debt, even as your own, home-rolled solution can cause it. But even with such a system, you have to be careful.

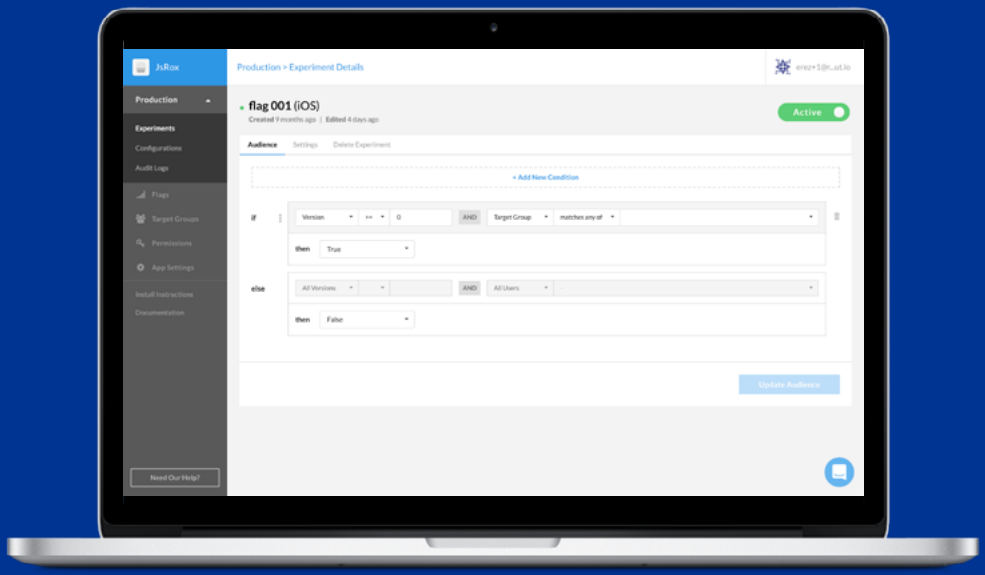
In general, codebases tend toward entropy—they tend to rot unless you actively curate them. It's no different with your implementation of feature flag management. Do your best to group toggle points as close together as possible, rather than having features sprawl all over the application. Adhering to the SOLID principles will help with this, as will keeping your code (and feature flag logic) DRY. And make sure to ruthlessly cull outdated toggle points and routers and to plan to retire your feature flags.

RESOURCE LINKS (LANGUAGES)

That should give you plenty to get started—the what, the why, and the how of feature flags, from the simple to the complex. I'll close by offering some detailed resources on top of the ones we've linked to throughout the post. These are detailed, tech stack-specific guides to getting started with feature flags:

- A guide to Android feature flags
- A guide to JavaScript feature flags
- A guide to NodeJS feature flags

Rollout



Rollout is an advanced feature management solution that gives engineering and product teams feature control, post-deployment. It's the most effective way to roll out new features to the right audience while protecting customers from failure and improving KPI's.

For more information:

LEARN MORE

visit https://rollout.io

EMAIL US

email us at support@rollout.io