Starlark Build Configuration + Feature Flags

Author: Marissa Staib (mstaib@google.com) with substantial help from Julie Xia (juliexxia@google.com)

Overview

config_feature_flag is effectively a specialized Starlark build setting from before Starlark build settings existed. They are much more similar than they are different, but they are internally handled very differently. Having two concepts which are handled differently but have the same purpose adds complexity to the codebase and to users. This document covers the design points required for unifying the two.

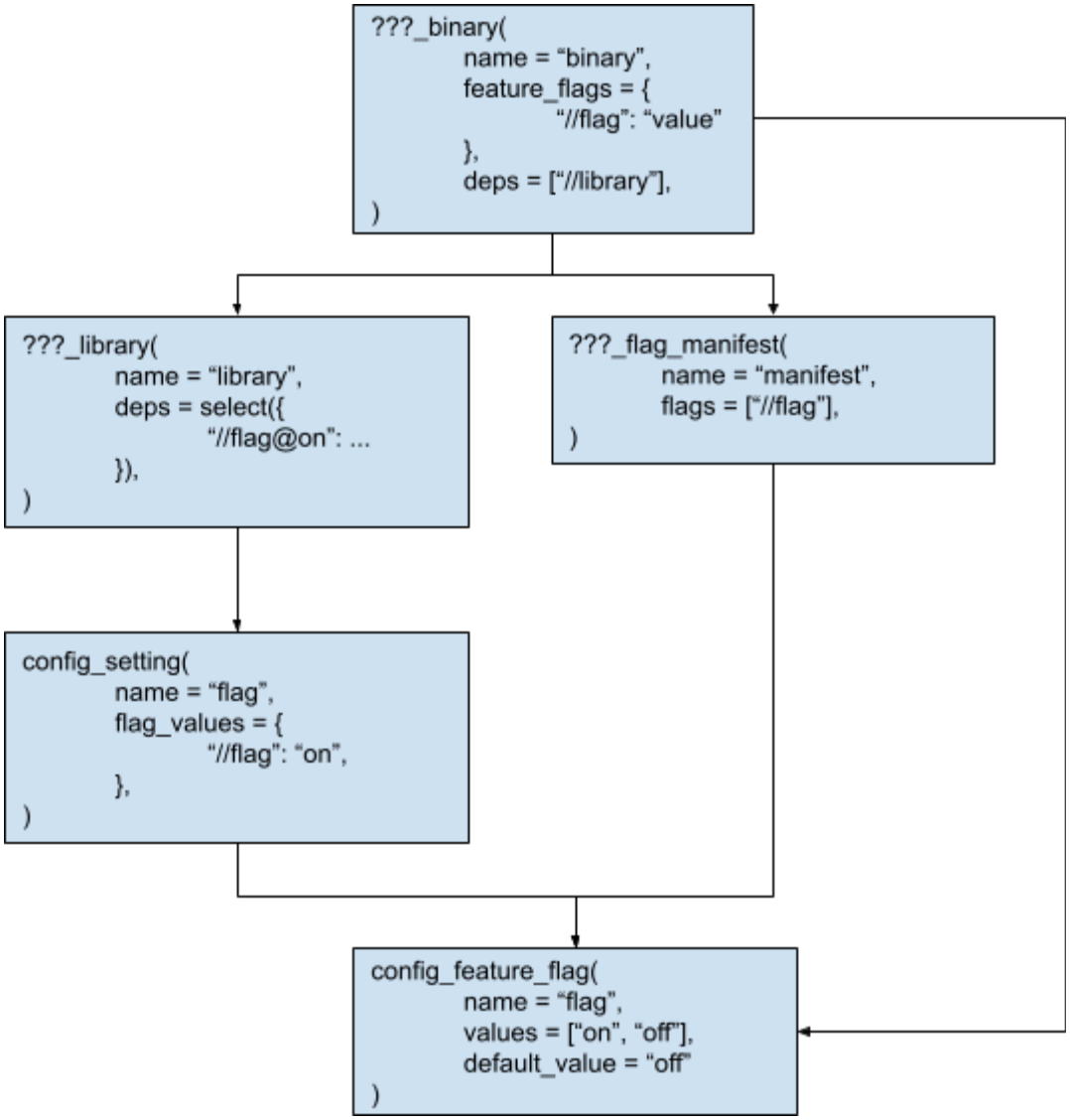
Background

config_feature_flag is an experimental feature which allows applications to modify the way their libraries build, or equivalently, allows libraries to expose the ability to tweak the structure of their build.

A library defines one or more feature flags (for example, a flag which determines whether certain encryption algorithms will be compiled in to an encryption library, or a flag which enables debug logging, or a flag which enables a new view). Each feature flag has a set of allowed values (all strings - for example, “on” and “off”) and a default value which is one of the allowed values (for example, “off”). The library can use config_setting to select on the value of this feature flag, or read its value (for example, to produce a manifest) via a Starlark provider.

An application sets a dictionary mapping the labels of feature flags to their values, and its dependencies are guaranteed to see those values for those feature flags and the default values for all other feature flags, regardless of whether the application is built at the top level or as a dependency of another application which uses feature flags. In other words, all of the feature flags’ values are reset when entering an application which sets feature flags.

Here’s a diagram depicting a basic example of an application with a feature flag.



Starlark build configuration allows for similar capabilities of defining, reading, and writing configuration data, as described in its design document. However, Starlark build configuration allows for significantly more flexibility - more types than just strings, the ability to set individual settings arbitrarily, etc.

Until now, config_feature_flag has been fairly low-profile - its overhead could potentially lead to out-of-memory errors. But with Starlark build configuration approaching, it seems a good time to roll the two together.

Relevant situations

Reading feature flags in transitions

Current state: Feature flags cannot be read by Starlark transitions.
Near term: Feature flags can be read by Starlark transitions by the same mechanisms used to read Starlark build settings (that is, declaring the name of the feature flag in the inputs of the transition)

Transitions don't exist today, so this is a new state anyway. But in the interests of reducing and eventually eliminating the conceptual gap between config_feature_flags and Starlark build settings, making it possible to read feature flags in the same places Starlark build settings can be read is helpful.

Order of rule-class transitions

Current state: Feature flag transitions do not occur on the same rule as a Starlark rule-class transition.
Near term: Starlark rule-class transitions happen before feature flag transitions (that is, they observe the state without the feature flag transition).

There are three options here:

1. The incoming transition on a rule which sets feature flags sees the parent's feature flags. (Starlark transition -> Feature flag transition)
2. The incoming transition on a rule which sets feature flags sees the dependency's feature flags. (Feature flag transition -> Starlark transition)
3. Incoming transitions on rules which set feature flags can't read feature flags through the configuration, even if other rules can.

Putting the feature flag transition first (option 2) means that the feature flags on the rule the transition is applied to can be observed both through the attributes and through the transition's setting map.

Option 3 requires additional work, specially preventing rules from reading feature flags when they set them. This is doable, and has some similarities to how Starlark transitions are not allowed to look at attributes which are configured by build settings changed by that transition. And it can expand into option 1 later.

Option 1 allows for reading of the parent's feature flags (not permitted by either of the other two options). This allows additional flexibility and provides consistency with most incoming transitions, which read the configuration received from the parent (potentially with the effects of an attribute transition).

Writing feature flags in transitions

Current state: Feature flags can only be written all at once (clearing all previous feature flag values) at the application level
Near term: Feature flags can be written individually in Starlark build transitions or all at once using the feature flag transition
Someday?: Groups of Starlark build configuration can be reset for hermetic builds of a given rule, and feature flags are just one such group.

Feature flags initially only supported being set as a group for an application. This is safer for many uses - for example, in Android, having a change. However, with the advent of Starlark build configuration, this is something that cannot be controlled by feature flags alone. Rules must be prepared for the possibility that somewhere in their transitive dependencies, they may encounter multiple configurations of the same target - or that they may themselves appear multiple times in the transitive dependencies of some other target. Keeping feature flags separate and different in this way doesn't necessarily grant benefit.

The ability to clear all feature flags is still a useful distinction, and possibly something that should be generalized in some way.

Defining

Current state: Feature flags are defined with config_feature_flag rules. Starlark build settings are defined with custom rule classes which were defined with special parameters to the rule function.
Near term: No change.
Someday?: The config_feature_flag rule class becomes a Starlark build setting rule class defined in some common place.

It would be nice to define config_feature_flag just as a normal Starlark build setting like any other, removing the need for any remaining special treatment, but to preserve the functionality of feature flags, some level of special treatment is still needed in the short term.

Internals

Current state: Feature flags are stored in a map in ConfigFeatureFlagConfiguration; Starlark build settings are stored in a map in BuildOptions
New state: Feature flags and Starlark build settings alike are stored in the same BuildOptions map

Merging these internals is one of the main benefits of combining these two concepts. This intrinsically permits most of the changes above, and also enables the ability to share infrastructural benefits that are applied to one or the other. For example, presently feature flags

have support for “tagged trimming”, where users must name the feature flags used by a target and its transitive dependencies (saving memory and avoiding duplication in the general case where targets do not need to be duplicated because of a change to a feature flag, while still allowing for targeted duplication where necessary), but no such trimming exists for Starlark build configuration. Similarly, support for automatic trimming of Starlark build configuration is coming, but it wouldn’t intrinsically support feature flags. Merging the two makes it easier to share benefits and keep the two concepts similar so as to eventually merge their user-interface representations as well.