Flutter
Eliminating nullOk Parameters

SUMMARY

Now that non-nullability is being enabled, the static functions which take a nullOk parameter don't make much sense anymore. This document proposes removing them.

Author: Greg Spencer (gspencergoog)
Go Link: flutter.dev/go/eliminating-nullok-parameters
Created: 10/2020 / **Last updated:** 10/2020

OBJECTIVE

To eliminate `nullOk` parameters to help with API sanity in the face of null safety, and make all `of` static accessors return non-nullable results.

BACKGROUND

Before non-nullability was the default, it was useful to have a toggle that swapped between throwing an exception and returning null, and it wasn't confusing, since every variable was nullable.

Now that non-nullability is the default, it is desirable to have the default return a non-nullable value, since saying `MediaQuery.of(context, nullOk: false)` and then being still required to apply a `!` after that call feels awkward. The goal of this design proposal is to eliminate that awkwardness in the API.

OVERVIEW

The gist of this proposal is to remove the `nullOk` parameter for the APIs that use

them, and add an additional API that prepends “maybe” to the name of the function that performs a similar function, but instead of throwing an exception, will return null if the sought-after result isn’t available.

Related issue: flutter/flutter#67163

One reason to eliminate this now rather than later, is that once the null-safety changes are mainstream, changing this will require the removal of many `!` operators.

DETAILED DESIGN/DISCUSSION

API with `nullOk`

The APIs in question are:

1. `MediaQuery.of`
2. `Navigator.of`
3. `ScaffoldMessenger.of`
4. `Scaffold.of`
5. `Router.of`
6. `Localizations.localeOf`
7. `FocusTraversalOrder.of`
8. `FocusTraversalGroup.of`
9. `Focus.of`
10. `Shortcuts.of`
11. `Actions.handler`
12. `Actions.find`
13. `Actions.invoke`
14. `AnimatedList.of`
15. `SliverAnimatedList.of`
16. `CupertinoDynamicColor.resolve`
17. `CupertinoDynamicColor.resolveFrom`
    a. This one is a little weird, since it already returns a non-nullable value.
18. `CupertinoUserInterfaceLevel.of`
19. `CupertinoTheme.brightnessOf`
20. `CupertinoThemeData.resolveFrom`
21. `NoDefaultCupertinoThemeData.resolveFrom`
22. `CupertinoTextThemeData.resolveFrom`
23. `MaterialBasedCupertinoThemeData.resolveFrom`

All of these have a `nullOk` parameter and (currently) a nullable return type. They would be converted to a non-nullable return type, and a sibling member function would be created with `maybe` prepended to the name. The `nullOk` parameter would

be removed, which is the breaking change here.

Alternatives

- Instead of modifying the functions listed above, we could arrive at new names for each of them, and deprecate the originals, eventually removing them. This might make for a less breaking migration. Some alternatives for the names would be `within`, `ofContext`, or `resolveFromTheme`. These are less desirable names than `of` or `resolveFrom`, since they aren't as straightforward to read and understand, but they would work.
- The `nullOk` parameter could be marked deprecated, and an assert added that would assert if `nullOk` was passed with `true` as a value, describing the correct API to use. The problem with this is that instead of a compile-time error, you get a runtime error, which triggers only if it gets called, so it could be missed.

Other API

The following `of` methods will be changed to return a non-null value. In cases where that non-null value is not available, they'll throw a descriptive assertion error describing why the value in question is not available (similarly to `debugCheckHasDirectionality`). If needed for a particular use case, we will add a `maybeOf` static method that returns null instead of throwing.

- `MaterialLocalizations.of`
- `CupertinoLocalizations.of`
- `WidgetsLocalizations.of`
  - All three are changing to never return null. A `maybeOf` method is not needed.
- `Directionality.of`
  - Changing to never return null, adding a `maybeOf` method.
- `Theme.of`
  - This never returns null unless `shadowThemeOnly` is set. We will remove that in favor of the `InheritedTheme.capture` API, which is more reliable. A `maybeOf` method is not needed.

TESTING PLAN

Run all existing Flutter and customer tests to verify compliance and compilation.

MIGRATION PLAN

The migration plan would probably include having a tool to allow users to mechanically fix their usages of the functions to use their nullable equivalents where nullability is needed, and the non-nullable one everywhere else,

automatically eliminating any `!` operators that are already added.

In addition, a migration plan will be published to describe migration (manual and automatic, if any).

Customer tests would be migrated as part of the implementation of this change.

Migrating these changes is fairly mechanical. Calls like this:

```
MediaQueryData? nonnullableData = MediaQuery.of(context); // nullOk = false
MediaQueryData? nullableData = MediaQuery.of(context, nullOk: true);
print('${nonnullableData!.devicePixelRatio}'); // Notice the necessary !
print('${nullableData?.devicePixelRatio ?? 'unknown'}');
```

Would convert to:

```
MediaQueryData nonnullableData = MediaQuery.of(context);
MediaQueryData? nullableData = MediaQuery.maybeOf(context);
print('${nonnullableData.devicePixelRatio}'); // No more !
print('${nullableData?.devicePixelRatio ?? 'unknown'}');
```