Flutter · Following

What's new in Flutter 3.38

Write less, see more, build faster

14 min read · 2 days ago

Kevin Chisholm Follow

Listen Share More

Introduction

Welcome back to our regularly schedule quarterly release, Flutter 3.38. This update is all about boosting your productivity and refining the developer experience with dot shorthands and updates to Widget Previews. Thanks to our community, this

release includes 825 total commits from 145 unique contributors, with 37 being first time contributors. Let's dig into what's in this release.

Dot shorthands

Write more concise Dart code! We're excited to announce a new Dart feature — **dot shorthands**! Shorthands reduces boilerplate by letting you omit types that Dart can infer.

For example, you can use shorthands to write `.start` instead of `MainAxisAlignment.start` .

```
// With shorthands
Column(
  mainAxisAlignment: .start,
  crossAxisAlignment: .center,
  children: [ /* ... */ ],
),

// Without shorthands
Column(
  mainAxisAlignment: MainAxisAlignment.start,
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [ /* … */ ],
),
```

This also works for named constructors! You can write `.all` instead of `EdgeInsets.all` :

```
Padding(
  padding: .all(8.0),
  child: Text('Hello world'),
),
```

This feature is on by default in Dart 3.10 and Flutter 3.38. For more information, check out the dot shorthands page on dart.dev. You can also read about this and more (like Dart hooks!) in the Dart 3.10 release blog post.

Web

Web development configuration files

The `flutter run` command now supports a configuration file for web settings. This enables you to specify host, port, certificate, and header information in a `web_dev_config.yaml` file at the root of your project. Check the file in so everyone on your team debugs with the same settings. For more information, visit setting up a web development configuration file.

Web development proxy settings

Along with existing command line flags, the web dev config file also supports new proxy settings. Proxy settings make it possible to forward requests to configured paths to another server. This makes it easier to develop a web client that connects to dynamic endpoints on the same host.

Details about proxy settings are also at setting up a web development configuration file.

Expanded support for hot reload on the web

Stateful hot reload is now enabled by default when running with `-d web-server` and you open the link to your Flutter application in a browser. This even works with multiple browsers connected at the same time.

As with `-d chrome`, this feature can be temporarily disabled using the `--no-web-experimental-hot-reload` flag. The ability to disable the feature will be removed in a future release so, if you encounter problems in your development workflow, please file a bug using Dart's web hot reload issue template. For more information, see the hot reload on the web documentation.

Framework

This release includes a number of powerful new capabilities and refinements across the framework, giving developers more granular control over advanced UI, navigation, and platform interactions.

Developers now have more power when creating pop-ups, dialogs, and other floating UI elements with `OverlayPortal`. It is now possible to render a child in any `Overlay` up the widget tree using `OverlayPortal.overlayChildLayoutBuilder` (#174239), making it easier to show an app-wide notification or other UI that needs to escape the layout constraints of its parent widget. The underlying Overlay.of method was also made more robust and efficient (#174315).

For a more modern Android navigation experience, predictive back route transitions are now enabled by default in `MaterialApp` (#173860). When a user performs the back gesture, they now see a preview of the home screen as the current route animates away. In addition, the default page transition has been updated to `FadeForwardsPageTransitionsBuilder` from `ZoomPageTransitionsBuilder` to reflect native behavior.

This release also deepens desktop integration. On Windows, developers can now access a list of connected displays and query detailed properties for each, such as resolution, refresh rate, and physical size (#164460). This enables the creation of applications with sophisticated window management features.

Finally, the framework itself is now more resilient. Errors that occur in widget lifecycle callbacks (such as `didUpdateWidget`) are now handled more gracefully, preventing them from causing cascading failures in the element tree (#173148). `ResizeImage` now correctly implements equality, which makes image caching and comparison more predictable by ensuring that identical `ResizeImage` providers are treated the same (#172643).

On the web, UI polish continues with a fix to `RSuperellipse` that prevents rendering errors when corner radii are larger than the widget itself (#172254), instead, such cases will be treated to produce the pill shape as expected.

For international users, detecting the browser's preferred locale is now more robust. The engine now uses the standard `Intl.Locale` web API to parse browser languages, replacing the previous manual and more fragile implementation (#172964). This change leads to more reliable locale detection and a better experience for a global audience.

An Android-specific bug (#171973) has been resolved, primarily affecting Samsung devices with hardware keyboards. Previously, after a user interacted with a `TextField`, the Android Input Method Editor (IME) could get stuck in a stale state. This caused the IME to mistakenly intercept "Enter" or "Space" key presses, preventing non-text widgets like a `Checkbox` or `Radio` button from receiving the event. The fix ensures that the `InputMethodManager` is correctly reset when the text connection closes, clearing the IME's stale state and restoring predictable hardware keyboard interaction for users.

Material and Cupertino updates

The Material and Cupertino libraries continue to evolve with a focus on API consistency and polished user experiences. This release brings a major API migration, new widget capabilities, and numerous refinements that make building beautiful, functional UIs more straightforward.

Building on the deprecation of `MaterialState`, this release continues the internal migration to the more unified `WidgetState`. This provides a consistent, expressive way to define a widget's appearance across different interaction states, such as pressed, hovered, or disabled, and requires no change for existing apps. This migration has been applied to a wide range of widgets and their themes, including `IconButton`, `ElevatedButton`, `Checkbox`, and `Switch` (#173893). The new API also adds power and flexibility; for instance, `IconButton` now includes a `statesController` property (#169821) that allows for programmatic control over its visual states, opening the door to more custom and interactive designs.

This release also introduces several new features and convenience APIs. The `Badge.count` constructor now includes a `maxCount` parameter (#171054) to easily cap the displayed count (for example, showing "99+" instead of "100").

For finer-grained gesture control, the `InkWell` widget now features an `onLongPressUp` callback (#173221), useful for triggering actions that should only complete when the user lifts their finger.

The Cupertino library also continues its journey toward better iOS fidelity. The `CupertinoSlidingSegmentedControl` adds an `isMomentary` property (#164262) to allow the control to trigger actions without persisting a selection. To better match native iOS behavior, the `CupertinoSheet` now features a subtle "stretch" effect when dragged upward while fully expanded (#168547).

Finally, this release is packed with refinements that polish the behavior of core components. Highlights include a fix for `DropdownMenuFormField` to correctly clear its text field when a form is reset (#174937) and updates to `SegmentedButton` to improve

focus handling (#173953) and ensure its border correctly reflects the widget's state (#172754).

Decoupling Material and Cupertino

We've been doing a lot of planning for decoupling the Material and Cupertino libraries from the framework. The following list includes several discussions around some recently published design docs.

Improving the release process for flutter/packages, which will include Material and Cupertino after decoupling.

- Status: Decided
- Ideas: First-Party Package Release Strategy
- Decided on: Batch Release One Pager (PUBLICLY SHARED)

Colors and dot shorthands

- Status: Decided
- A Basic Color Set for Flutter (PUBLICLY SHARED)

Decoupling tests

- Status: In progress
- Decoupling Framework Tests (PUBLICLY SHARED)
- https://github.com/flutter/flutter/issues/177028

Text

- Status: In discussion
- Flutter Decoupling Design From Text (PUBLICLY SHARED)

Scrolling: more robust and predictable slivers

This release brings a number of fixes that make building complex scrolling layouts, especially those using `SliverMainAxisGroup` and `SliverCrossAxisGroup`, more robust and predictable.

Developers using these widgets to group multiple slivers will find that gesture handling is now more reliable. Hit-testing for taps and other pointer events on slivers within these groups is now correctly calculated, ensuring that user interactions behave as expected (#174265).

Several other fixes contribute to more accurate scrolling behavior within a `SliverMainAxisGroup`. Over-scrolling issues when using a pinned header are resolved (#173349), calling `showOnScreen` to reveal a sliver now works correctly (#171339), and the internal scroll offset calculation is more precise (#174369).

For developers building custom scroll views, the new `SliverGrid.list` constructor (#173925) offers a cleaner way to create a grid from a simple list of children.

This release also improves focus navigation for keyboard and D-pad users in complex layouts. In nested scroll views with different scroll axes (such as a vertical list of horizontal carousels), directional focus navigation is now more predictable, preventing the focus from jumping unexpectedly between sections (#172875).

Accessibility: a more inclusive experience for all users

Making applications accessible to all users is a cornerstone of the Flutter framework. This release continues that commitment by giving developers more programmatic control, improving the experience for international users, and polishing the accessibility of core widgets.

For developers building complex applications, this release introduces the ability to turn on accessibility by default on iOS by using `WidgetsFlutterBinding.instance.ensureSemantics` (#174163). Debugging accessibility issues is now easier, as `debugDumpSemanticsTree` includes additional text input validation results information to help diagnose problems more quickly (#174677).

For advanced accessibility in sliver-based scrolling views, the new `SliverSemantics` widget (#167300) is now available. Much like the existing `Semantics` widget, developers can use `SliverSemantics` inside a `CustomScrollView` to annotate portions of their sliver tree with specific semantic information. This is particularly useful for annotating headers, assigning semantics roles, and adding descriptive labels to slivers for screen readers, providing a more understandable and accessible experience for users.

Finally, the accessibility of core widgets continues to be refined. The `CupertinoExpansionTile` is now accessible by default (#174480), and the `AutoComplete` widget now announces the status of search results to the user (#173480). Other improvements, such as larger touch targets in the `TimePicker` (#170060), contribute to a more accessible out-of-the-box experience.

iOS

We are happy to confirm that Flutter fully supports the latest platform releases: iOS 26, Xcode 26, and macOS 26, all of which were released in September. This ensures you can immediately begin developing and testing your apps on Apple's newest operating systems and tooling.

You might have already noticed significant quality-of-life improvement for iOS developers shipped in the last release in Flutter, addressing a long-standing user annoyance: the requirement for the Xcode application to launch automatically when running Flutter apps on physical devices using `flutter run`. We've introduced a new deployment method using the Xcode 26 command line tool, `devicectl`, for app installation, launch, and debugging. This transition eliminates the need to invoke the Xcode application during deployment, and relies solely on command-line Xcode build tools in most cases. If you see problems, you can disable this deployment method with `flutter config --no-enable-lldb-debugging`, and please file an issue to let us know!

Previously, this feature relied on Xcode automation which became unstable and flakey on Xcode 26, especially when running consecutive commands. If you are now developing for the newest Apple releases, we strongly recommend updating your version of Flutter to 3.38 or higher.

UIScene Lifecycle Migration

Flutter 3.38 includes essential support for the Apple-mandated UIScene lifecycle. This is a critical, proactive update following Apple's announcement at WWDC25: "In the release following iOS 26, any UIKit app built with the latest SDK will be required to use the UIScene life cycle, otherwise it will not launch."

To ensure your iOS Flutter applications remain compatible and launch successfully on future iOS releases, a migration is required.

Migrating Flutter applications

All existing iOS Flutter applications must migrate to the new lifecycle. You have two paths to complete this migration:

1. Manual migration: Follow the manual migration instructions provided on Flutter's website.
2. Automatic migration (Experimental): Enable an experimental feature to handle the migration automatically. This will be enabled by default in a future release. Run the following command:

```
flutter config --enable-uiscene-migration
```

Migrating Flutter plugins

Flutter plugins that rely on application lifecycle events must be updated to use the UIScene lifecycle events. Plugin developers should refer to the migration guide. Plugins that have not migrated will display warnings in a future release.

Migrating embedded Flutter (Optional)

For projects that embed Flutter within a native host application, migrating is optional but highly recommended. Adopting Flutter's new UIScene APIs using the add to app migration guide, enables scene lifecycle events for your plugins, ensuring compatibility with the Flutter ecosystem

Android

16KB page size compatibility

Upgrading to Flutter 3.38 is essential preparation for Google Play's 16 KB page size compatibility requirement. Starting November 1, 2025, apps targeting Android 15 and higher must support 16 KB pages. This change ensures your app runs correctly on high-RAM devices, offering performance benefits like up to 30% faster launches. Flutter 3.38 updates the default Android ndkVersion to NDK r28, the minimum required for native code to achieve the proper alignment for 16 KB support.

Memory fixes

Flutter 3.38 fixes a significant memory leak impacting all Flutter apps on Android. The issue (introduced in 3.29.0) occurred when Activities were destroyed upon exit, as configured by developer settings or when Activities were killed by the system due to low memory.

Android dependency updates

It can often be a challenge to figure out the right combination of versions for Android dependencies that will work for your app, including Gradle, the Android Gradle Plugin (AGP), the Kotlin Gradle Plugin (KGP), Java, and others. For the Flutter 3.38 release, we tested and confirmed compatibility in our continuous integration (CI) environment with the following set of Android dependency versions:

- **Java 17**: required minimum version for Android development in Flutter 3.38.
- **KGP 2.2.20**: maximum known and supported Kotlin Gradle Plugin version for the tooling.
- **AGP 8.11.1**: the newest Android Gradle Plugin version compatible with KGP 2.2.20.
- **Gradle 8.14**: this version works with the chosen versions of Java, KGP, and AGP. Note that Gradle 8.13 is the minimum version required for AGP 8.11.1.

To ensure that your application seamlessly across Flutter releases, we strongly encourage you to use the API level variables vended by the Flutter SDK in your build files. The configured values for this release are:

- `flutter.compileSdkVersion` (API 36)
- `flutter.targetSdkVersion` (API 36)
- `flutter.minSdkVersion` (API 24) or higher

Engine

Performance overlay

The performance overlay has been refactored to be more efficient, reducing its rendering time on both Skia and Impeller backends. This means you get more accurate performance data with less overhead. (#176364)

Vulkan and OpenGL ES

Numerous fixes and improvements to the Vulkan and OpenGL ES backends improve stability and performance on a wider range of devices. This includes better handling of pipeline caches (#176322), fence waiters (#173085), and image layout transitions (#173884).

Renderer unification

Work continues to unify the CanvasKit and Skwasm renderers. This release includes significant refactoring to share more code between the two, which will lead to a more consistent experience and faster development in the future (#174588).

Thread merging

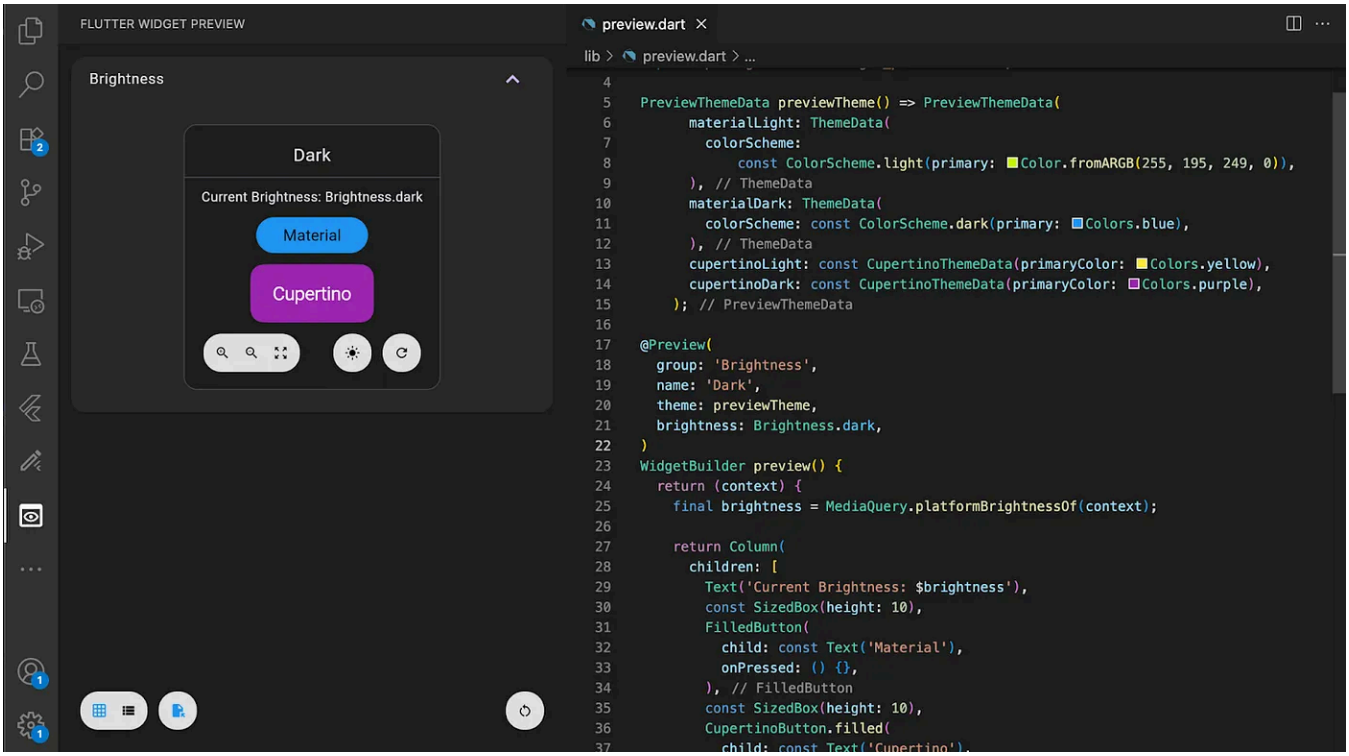
The ability to opt-out of thread merging has been removed from iOS and Android. For more information, check out the great thread merge video.

DevTools and IDEs

Experimental Widget Previews — Updates

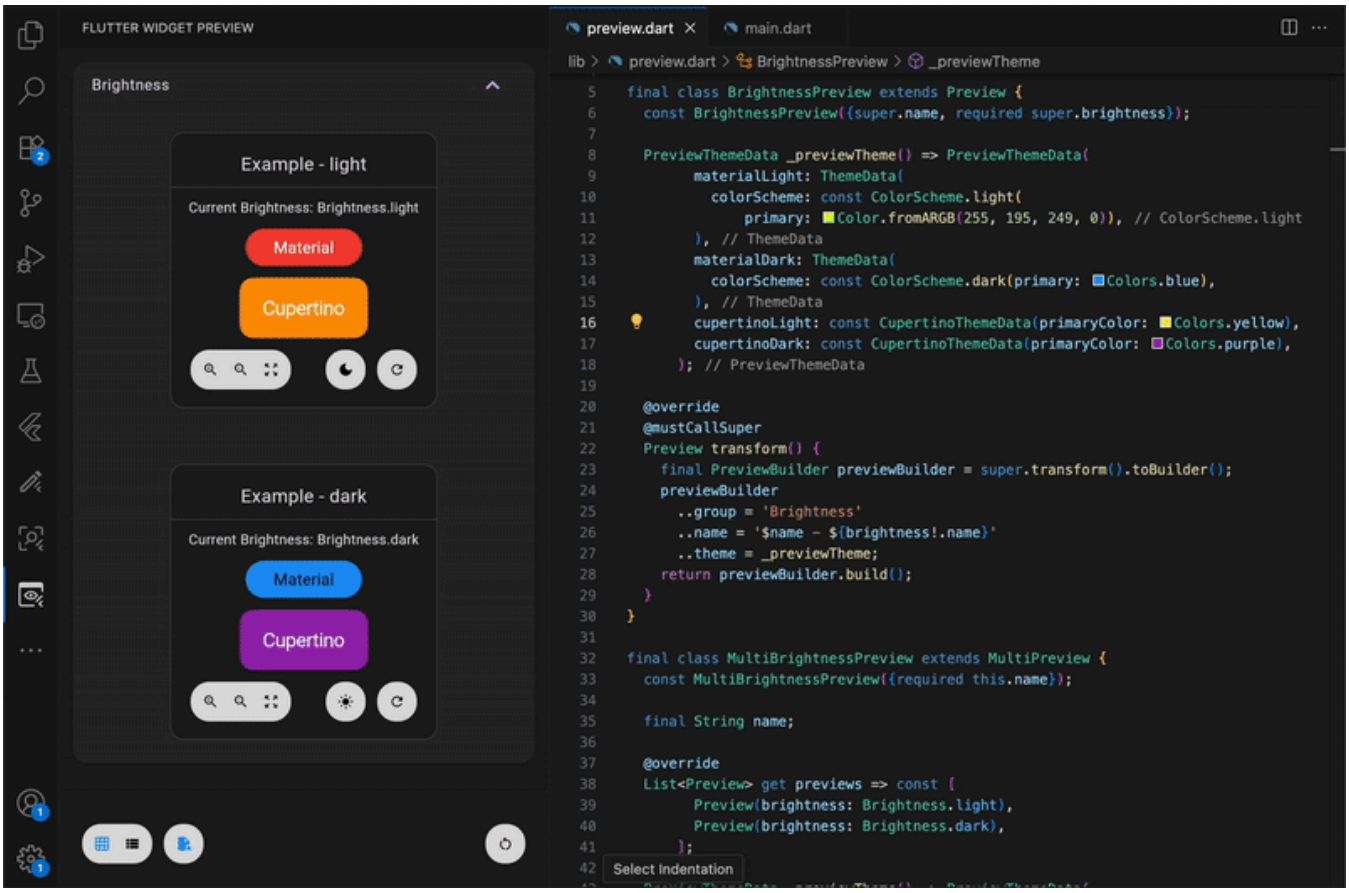
Flutter 3.35 introduced Widget Previews, an experimental feature ready for early feedback from the community. The Flutter 3.38 release brings significant improvements to Widget Previews, including:

- **IDE integration**: Both our VSCode and Intellij / Android Studio plugins are updated with initial support for Widget Previews. You can now view your previews directly within your IDE for a more seamless development experience.

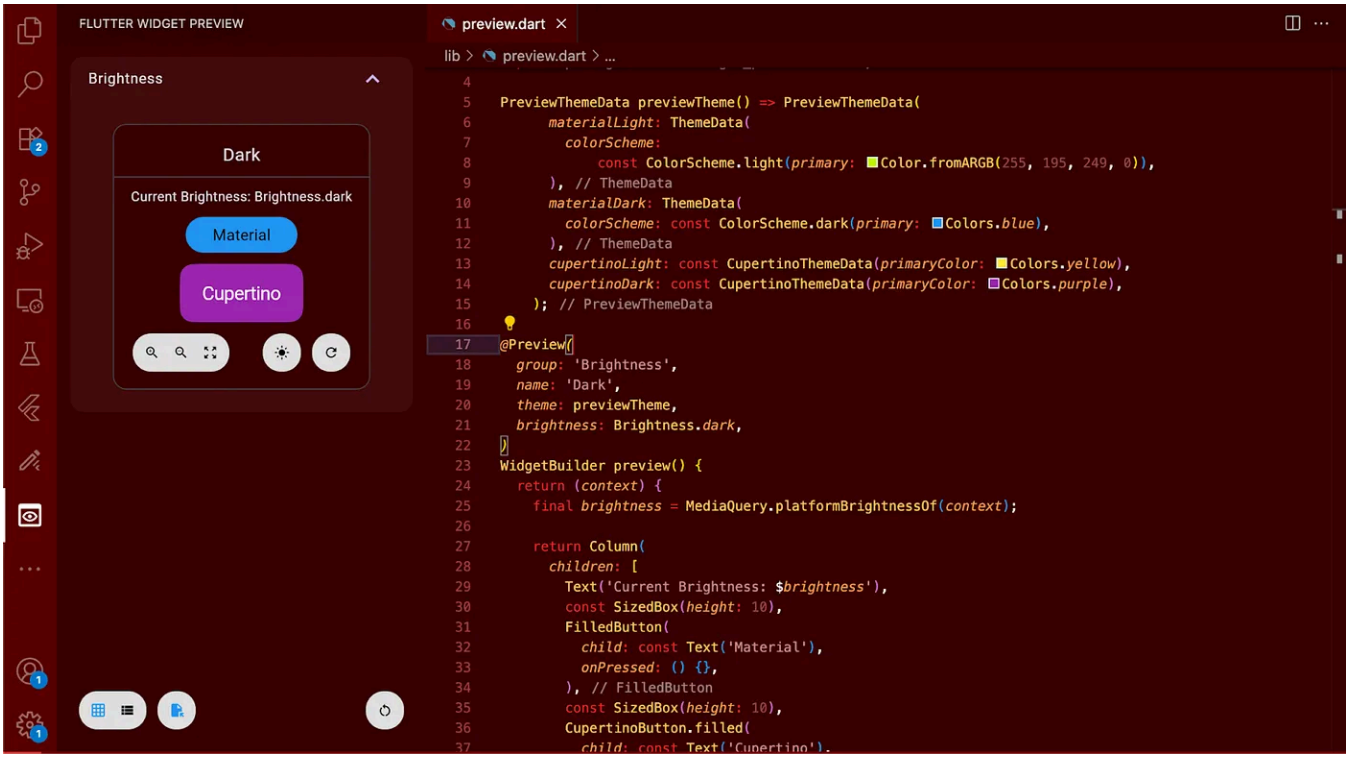


Widget Previews embedded in VSCode.

When used within an IDE, the widget preview environment is configured by default to filter the displayed previews based on the currently selected source file:

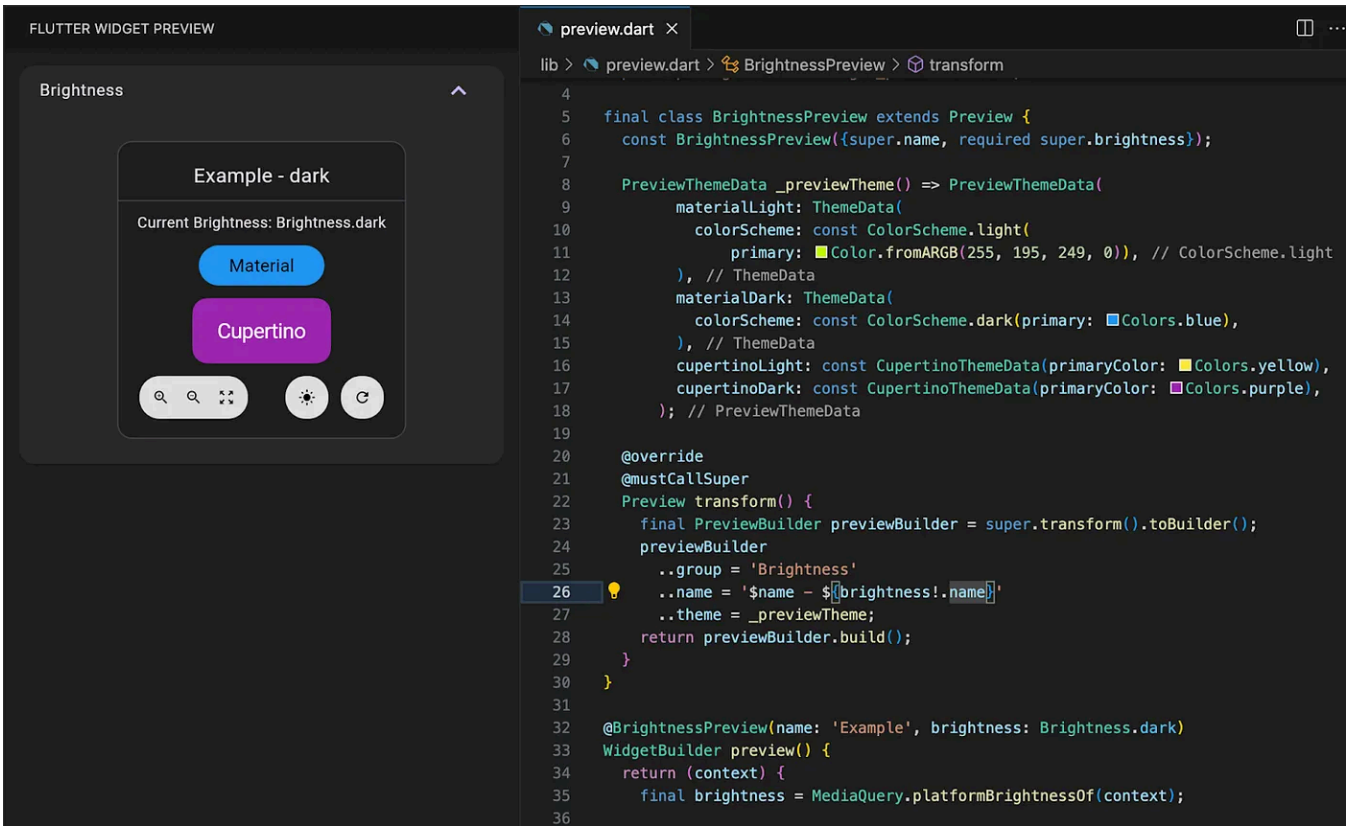


- **Widget preview environment theming and control improvements:** The widget preview environment now supports light and dark modes, as well as custom IDE color schemes to match your development environment. Controls within the widget preview environment have also been adjusted to use less space, leaving more room available for rendering previews.



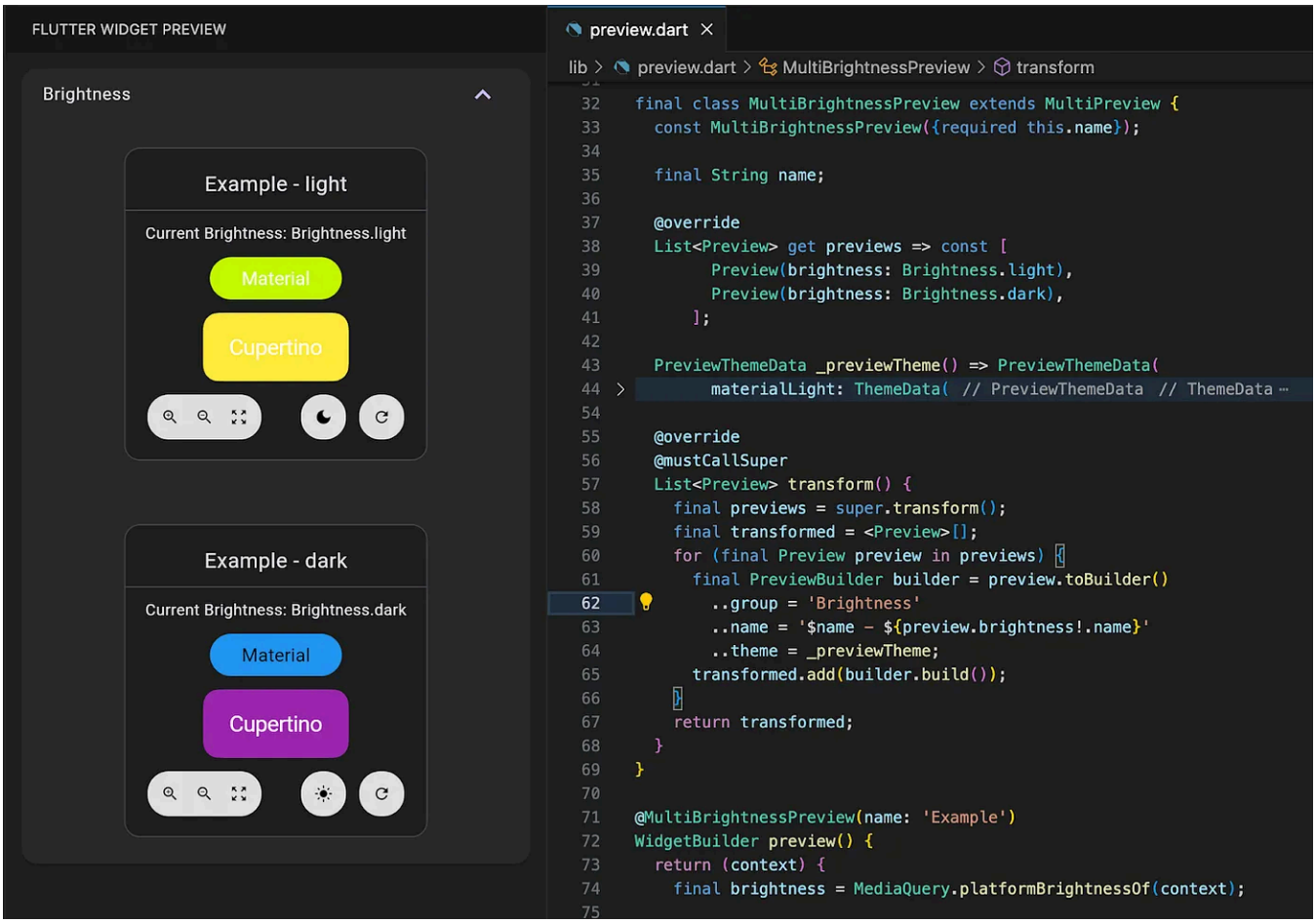
Custom theming support for the Widget Previews environment.

- **Preview extensibility:** The Preview annotation class is no longer marked as final and can now be extended to create custom Preview annotations, allowing for less boilerplate for common preview types.

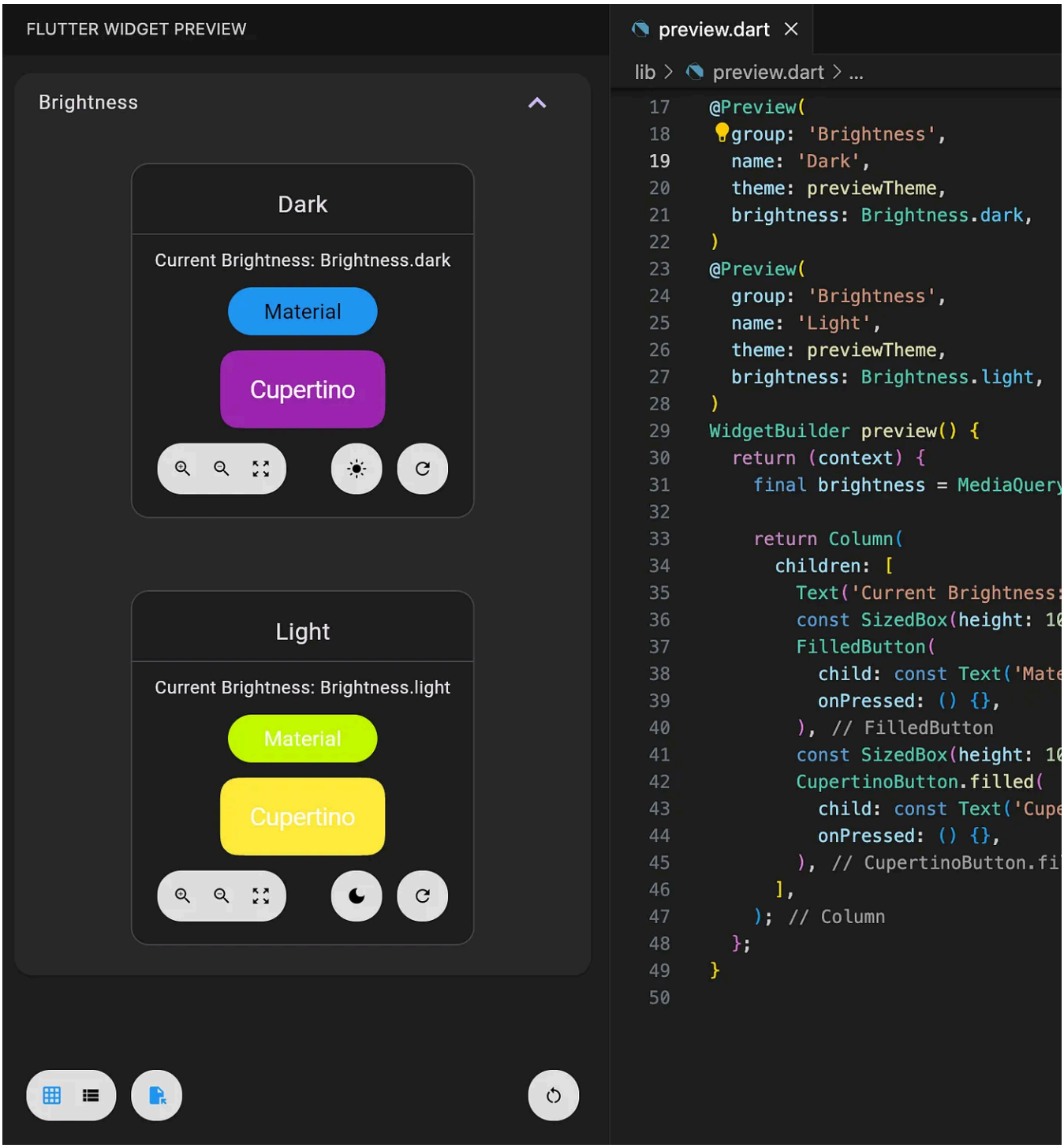


An example of a custom `BrightnessPreview` annotation.

- **MultiPreview support:** A new `MultiPreview` base class allows for creating multiple Preview variations from a single, custom annotation.



- **Preview groups:** A new group parameter in the `Preview` class allows the grouping of related previews.



An example of multiple "Brightness" previews in a preview group.

- **Reduced restrictions around @Preview annotation arguments:** Private constants are now supported as arguments to `Preview` annotations. Function arguments (such as wrapper and theme) are still required to have public, statically accessible names.

Widget Previews is still an experimental feature, and your feedback is critical in shaping its future. The APIs and user experience are not yet stable and will change as we learn from you.

Based on early feedback, more enhancements are planned to improve the widget preview experience, including:

1. **Flutter DevTools Widget Inspector support**: The Widget Inspector is being updated to support inspecting previews within the widget preview environment. We plan to embed the inspector directly within the widget previewer, making it easily accessible regardless of your development environment.
2. **Multi-project support in IDEs:** The widget previewer currently only supports displaying previews contained within a single project or Pub workspace. We're actively investigating options to support IDE sessions with multiple Flutter projects (issue #173550).
3. **Startup performance improvements:** Opportunities for performance improvements are being investigated to reduce initial startup times, including:

- Launching a precompiled widget preview environment after the first run
- Parallelizing preview detection logic to better handle large projects

To get started, check out the documentation and let us know what you think!

- **Read the docs:** Getting Started with Flutter Widget Previews (Experimental)
- **Give feedback**: File issues and feature requests in the Flutter GitHub repository.
- **Learn more**: For a technical deep-dive, see the Flutter Widget Previews Architecture document.

***Important Note**: There's a known issue where the Widget Previewer can crash or stop updating after a* `flutter pub get`*. If you encounter this issue, run* `flutter pub get` *in your project and restart your IDE. See #178317 for details.*

DevTools updates

Flutter 3.38 contains fixes for some of the top pain points users called out in the 2025 DevTools user survey, including:

Network Panel Improvements

- Made it easier to understand when the panel is recording network traffic. (#9495)

- Fixed issues around copy-pasting network requests. (#9472, #9482, #9485, #8588)

Flutter Inspector fixes

- Fixed a bug where selecting a widget sometimes opened the underlying framework source code instead of the user's source code. (#176530)
- Fixed a bug occasionally preventing interaction with the the top buttons in the Inspector panel. (#9327)

Deprecations and breaking changes

This release includes several important deprecations and breaking changes as part of the ongoing effort to modernize and improve the Flutter framework.

Key build and tooling changes have been made that might affect custom build scripts. The `version` file at the root of the Flutter SDK has been removed in favor of a new `flutter.version.json` file located in `bin/cache` (#172793). Additionally, the `AssetManifest.json` file is no longer generated by default (#172594).

Other notable changes include:

- For more predictable behavior, a SnackBar that includes an action will no longer auto-dismiss (#173084).
- The `OverlayPortal.targetsRootOverlay` constructor is deprecated in favor of the more flexible `OverlayPortal(overlayLocation: OverlayChildLocation.rootOverlay)`.
- Several properties on `CupertinoDynamicColor`, such as `withAlpha` and `withOpacity`, are now deprecated in favor of standard `Color` methods (#171160).
- Flutter 3.38 requires Java 17 as the minimum version for Android, matching the Gradle 8.14 (July 2025 release) minimum requirement.

For more details and migration guidance on these and other changes, check out the breaking changes page.

Outro

Flutter 3.38 is focused on making your day-to-day development faster and more enjoyable. These enhancements aim to streamline the way you build. We are

incredibly grateful for the hard work and feedback from every community member who contributed to this release.

For a comprehensive list of all the changes, be sure to check out the detailed breaking changes and release notes. To get a free boost to your productivity, simply run `flutter upgrade`!

Announcements | Releases | Flutter | Flutter App Development | Release Notes

Following

Published in Flutter

64K followers · Last published 2 days ago

Flutter is Google's UI framework for crafting high-quality native interfaces on iOS, Android, web, and desktop. Flutter works with existing code, is used by developers and organizations around the world, and is free and open source. Learn more at https://flutter.dev

Follow

Written by Kevin Chisholm

13.2K followers · 7 following

Kevin Chisholm is a Technical Program Manager for Dart and Flutter at Google.

Responses (7)

Huong Pham

What are your thoughts?

Michał Nowak
2 days ago

For anyone looking into widget previews. There's a typo in the docs link. The correct one is: https://docs.flutter.dev/tools/widget-previewer

58 1 reply Reply

Anthony Robledo
2 days ago

Amazing, thanks! I want to give a huge shout-out to Rami for filing the original feature request for dot shorthand, and to the community for always pushing towards syntactic excellence!

22 Reply

O

Omar Jadiani
1 day ago

Thank you, Kevin for the amazing articles🙌

1 Reply

See all responses

More from Kevin Chisholm and Flutter

In Flutter by Kevin Chisholm

What's new in Flutter 3.35

Hot Reload, Widget Previews, and More

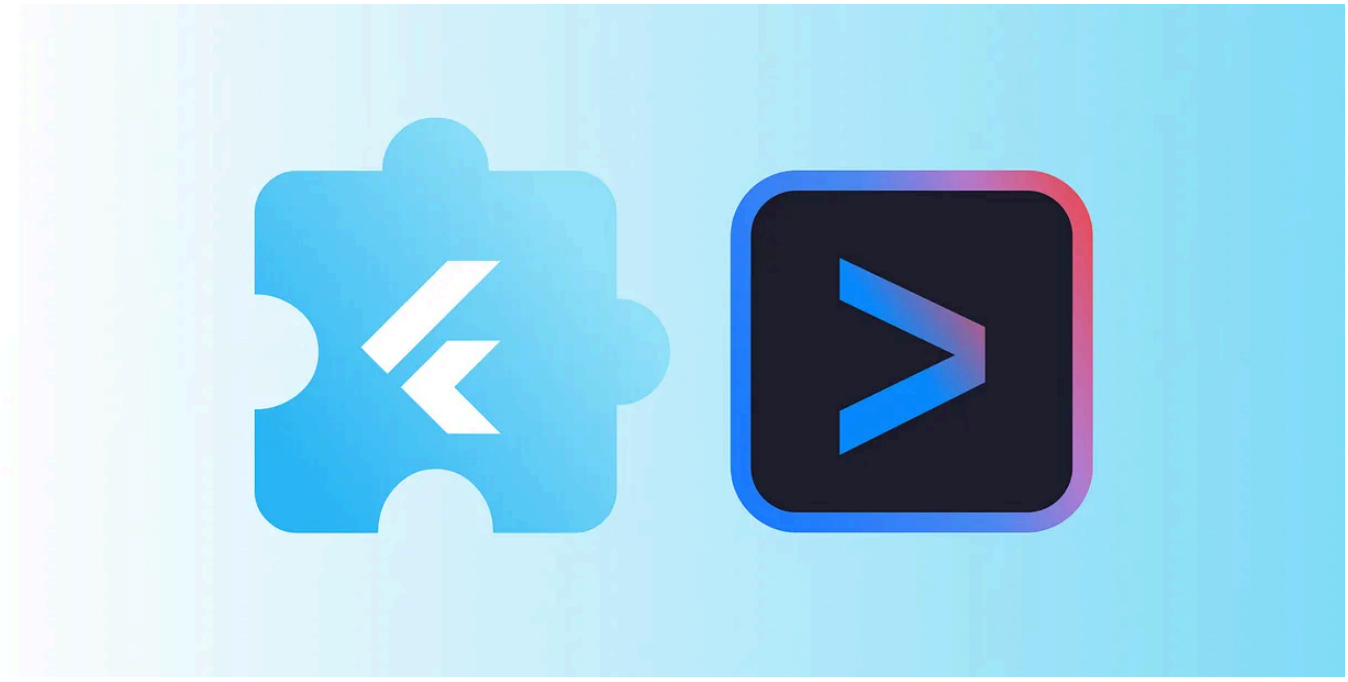
Aug 15 2.3K 23

In Flutter by Emma Twersky

Announcing Flutter 3.38 & Dart 3.10: Building the future of apps

Hi, Flutter community! My name is Emma Twersky, and I am unbelievably excited to be your new lead for Flutter and Dart DevRel. I'm not just...

2d ago 549 5

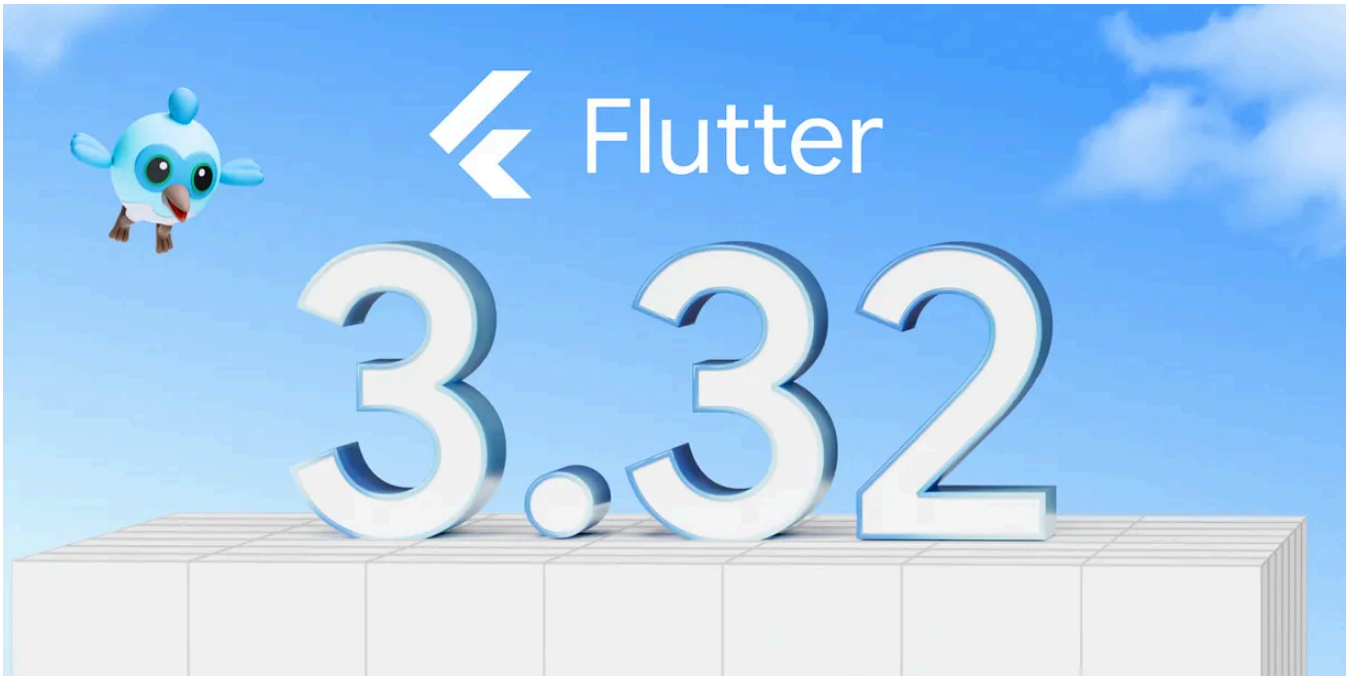


In Flutter by John Ryan

Meet the Flutter Extension for Gemini CLI

Build high quality, feature rich apps with the new Flutter Extension for Gemini CLI

Oct 8 636 6



In Flutter by Kevin Chisholm

What's new in Flutter 3.32

Hot reload on web, native fidelity, and deeper integrations

May 21 3.2K 35

See all from Kevin Chisholm

See all from Flutter

Recommended from Medium

Yuri Novicow

Does Google plan to abandon Flutter, and should we worry?

Every few months, we have this same conversation about Google potentially killing Flutter. But this time there's actually some stuff worth...

3d ago 126 2

Seungchul Jeff Ha

Stop Wasting Hours on Flutter Builds!

Learn how to build Flutter apps with separate dev and prod environments using .env files. Automate APK, AAB, and IPA builds in minutes!

Nov 7 20 2

In Flutter by Emma Twersky

Announcing Flutter 3.38 & Dart 3.10: Building the future of apps

Hi, Flutter community! My name is Emma Twersky, and I am unbelievably excited to be your new lead for Flutter and Dart DevRel. I'm not just...

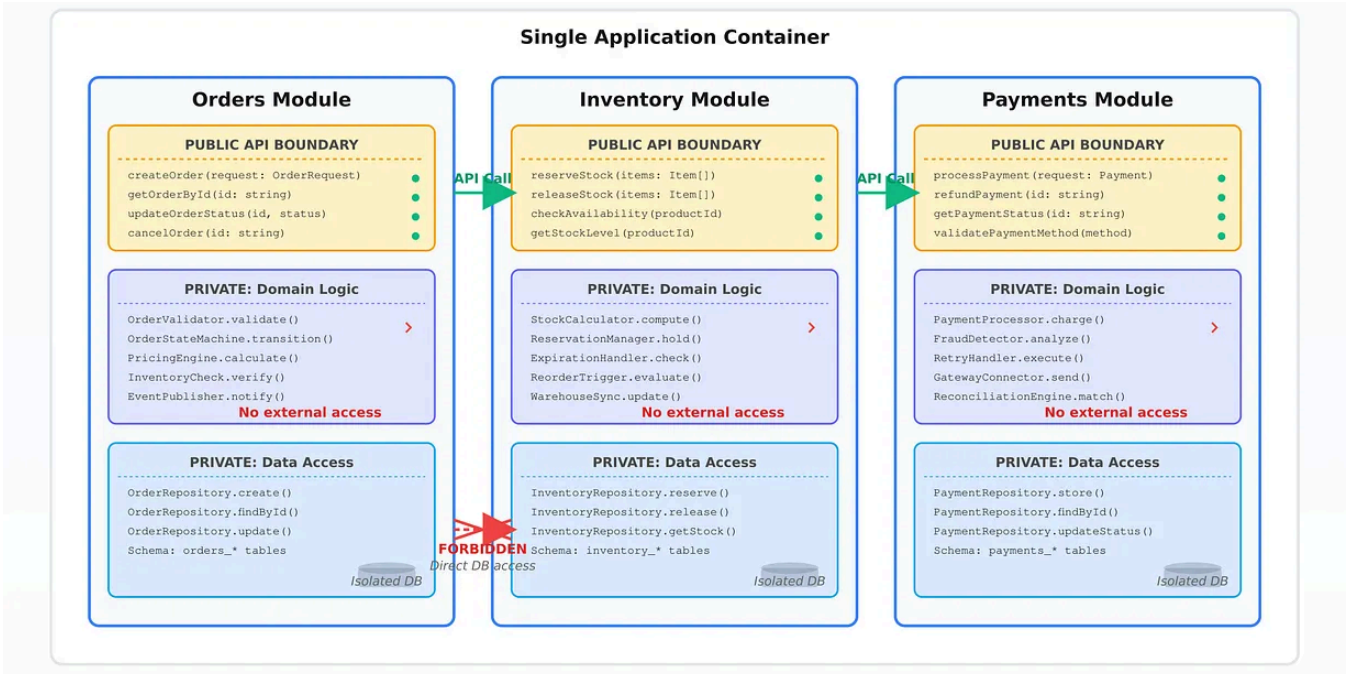
2d ago 549 5

In Easy Flutter by Jack Henry

9 Dart and Flutter Functionalities Every Senior Dev Uses in Every Project

Because surviving production bugs isn't luck, it's strategy.

Nov 6 15 3



The Atomic Architect

Architecture Patterns That Actually Scale In 2025: The Only Three You Need

Last month, a company I advised shut down.

6d ago 772 24

In Stackademic by Mobile App Developer

RIP Flutter? Apple's iOS 26 (Liquid Glass) Just Changed the Game—What Happened to Cross-Platform

"Flutter is dead." "React Native is done." "Apple killed cross-platform development."

Jul 14 187 12

See more recommendations