Adapting your app for iPhone Duo

A practical checklist for making your iOS app work on both displays, every fold angle, and Split View. Apple's official guidance, turned into a working method, with my app **Tiles** as the running example.

STEP 1
Build & test
Latest Xcode, Duo simulator, every pose.

→

STEP 2
Resize
Size classes, safe areas, no fixed widths.

→

STEP 3
Bars
System toolbars that move to the side.

→

STEP 4
Fold
Keep content off the hinge and camera.

README · WHO THIS GUIDE IS FOR

- **Beginners:** learn what's different about Duo and which three or four habits cover most of the work.
- **Technical readers:** get the API names (`ReservedRegion`, `ArrangementView`, toolbar visibility priorities) and the traps that are easy to miss.

Sources: Apple's *Designing for iPhone Duo* (HIG) and *Preparing your app for iPhone Duo* (developer docs), published September 9, 2026. This guide is a summary, not a replacement. Read both pages, because details may change.

01 · THE BIG IDEA

Same app, new sizes

You are not designing a new app for a new device. You're still designing for iPhone. What changes is that your app can now appear at many sizes.

Outer display
Device closed. Wide and short compared with other iPhones.

Inner display
Device open. A large, roughly tablet-like canvas.

Partially folded
The hinge splits the inner display into two usable halves.

Split View
Two apps sharing the inner display.

Know the hardware

PART	WHAT TO KNOW
Outer camera	Sits in the corner and is **always visible**. It expands into the Dynamic Island for Live Activities.
Inner camera	Hidden behind the display. Only occupies space **when the camera is active**.
Hinge / fold	Only matters when **partially open**. Divides the inner display into multiple usable regions.
Poses	Closed, tent, open wide, partially folded like a book, open tall, folded like a laptop.

THE RULE · DON'T DESIGN PER POSE

Use **size classes**. A compact-width layout for the outer display and a regular-width layout for the inner display cover every pose. Let the existing layout expand into the space rather than reinventing the app each time it resizes. If your app uses system components and is built to resize, most of this is automatic.

02 · STEP ZERO

Build, then test

Before you change anything, see what Duo does to your app today.

STEP 1
Rebuild
Use the latest Xcode. Built with Xcode 26 or earlier, your app won't extend under the status bar and camera.

→

STEP 2
Run on Duo
Use the Duo simulator or a device. Device Hub in Xcode previews each pose.

→

STEP 3
Walk it all
Open, close, fold partially, and rotate in each pose. Check every view, sheet, and popover.

While you test, look for four things

1. **Views that don't resize well** in each orientation and pose.
2. **Bar presentation.** How navigation bars, toolbars, and tab bars appear on the side.
3. **Sheets and popovers** that land awkwardly when you fold or unfold.
4. **Buttons or content** sitting in the fold or under the camera.

TILES STORY · FIRST LAUNCH

The first launch on the Duo simulator didn't look good. A control sat under the camera and system controls, **blocking one of the most important buttons in the app**. Fixing that was the first real task.

03 · LAYOUT

Make it resizable

This is the foundation. Do it first and a lot of the rest comes free.

DO

- Prefer system containers: split views, tab bars, navigation stacks, arrangement views.
- Lay out with **size classes, layout margins, and safe area insets**.
- Size views **relative to their container**.
- Calculate layout from your **scene or containing view's bounds**, not screen dimensions.
- In UIKit, adopt Auto Layout and observe `horizontalSizeClass` and `verticalSizeClass` through automatic trait tracking.

DON'T

- Use fixed widths or anything tied to one display.
- Use `UIDevice.userInterfaceIdiom` or `UIInterfaceOrientation` for layout decisions in UIKit.
- Assume a screen size. Your app can be any size now.

Keep the experience consistent across displays

Same functionality, same state, same information hierarchy. The inner display can show an **additional level of hierarchy** if it suits your content.

EXAMPLE · APPLE MAIL

Closed, you see either the list *or* an email. Open, you see both side by side. Nothing was removed, the inner display just shows one more level.

If your app already works on iPad and Mac, or has been prepared for iPhone Mirroring, you're most of the way there.

04 · VERTICAL CONTROLS

Bars move to the side

On the outer display, toolbars, tab bars, and navigation controls move from the top and bottom to the side. That keeps vertical space for content and controls within reach. On the inner display they stay on the side in landscape and return to horizontal bars in portrait.

SwiftUI

Add `.toolbar { }` to a `NavigationStack` or `NavigationSplitView`.

UIKit

Set toolbar items on a view controller inside a navigation controller. Don't build custom bars from `UIToolbar`, `UINavigationBar`, or `UITabBar`.

If your bars aren't going vertical, you're probably not using the system's bar support.

Some contexts behave differently

CONTEXT	BAR PRESENTATION
Inspectors	Horizontal.
Split views	Horizontal for sidebar and content, vertical for the detail view.
Sheets (outer)	Vertical by default. Opt out with `toolbarVerticalBehavior(_:)` (SwiftUI) or `preferredVerticalBarBehavior` (UIKit).
Sheets (inner)	Horizontal for centered or leading placement, vertical for trailing. Set with `presentationPlacement(_:)` / `preferredPlacement`.

From the edge, top to bottom, the outer display stacks: **Dynamic Island, status bar, toolbar, tab bar.**

05 · TOOLBAR ITEMS

Organize your toolbar

The system decides where each item lands. Your job is to give it the right ingredients.

1. **Primary navigation on top.** Back and Close first, then prominent actions like Done. In SwiftUI use `topBarPinnedTrailing` for Done and `cancellationAction` for a custom Back or Close.
2. **Group related items** with `ToolbarItemGroup` or `UIBarButtonItemGroup`. Don't add spacing yourself, the system adds the gaps.
3. **Set visibility priorities.** Items overflow bottom to top. Use `visibilityPriority` to keep frequent actions (Mail's Compose) and badged status items visible longest.
4. **Use the system overflow menu** (`ToolbarOverflowMenu` , or `additionalOverflowItems`). Move your own overflow actions in, and reserve the ellipsis for overflow only.
5. **Don't override default bar placement.** Familiar positions are part of the platform.

Give every item an icon *and* a title

WHERE THE ITEM ENDS UP	WHAT THE SYSTEM USES
Vertical	Icon
Horizontal	Icon or title (prefers icon)
Overflow menu	Icon and title

TWO TRAPS · NOT SHOWN VERTICALLY

- An item with a **title but no icon**.
- An item built with a **custom view**.

Prefer symbols, always include a title for the overflow menu, and keep text-only buttons to a minimum.

06 · ASYMMETRY

Plan for a lopsided canvas

Controls sit along one edge, so your content area is no longer symmetric. Use safe areas so nothing is covered, including controls on the *opposite* edge when two apps share the display.

SPLIT VIEW · EACH APP USES ITS OWN OUTER EDGE

The left app has controls on the left, the right app on the right. Controls stay on the outside instead of getting lost in the middle. Vertical controls also stay aligned with the hardware, so they hold the same position relative to the camera and stay on the same side in right-to-left languages.

When space is tight: toolbar or tab bar?

Navigation-focused screens

Compress the **toolbar** into the overflow menu so the tab bar and main destinations stay reachable. This is the default.

Task-focused screens

Minimize the **tab bar** so the toolbar actions that complete the task remain visible.

Controlled by `ToolbarVerticalCompressionBehavior` (SwiftUI) and `UIVerticalBarCompressionBehavior` (UIKit).

Smaller details worth getting right

a **Keep controls near their content.** In Mail's inner display, list controls stay above the list pane.

b **Keep relative positions similar across poses** so people don't relearn where things live.

c **Full-width is fine for immersive screens** like Calculator, if nothing hits the Dynamic Island. Let a hero image span wide with `backgroundExtensionEffect()`.

d **Need to know if bars are vertical?** Read `toolbarVerticalEdge` (SwiftUI) or the `verticalBarEdge` trait (UIKit).

07 · RESERVED REGIONS

Dodge the fold

Apple calls the areas your content must avoid or adapt around **reserved regions**. A region can be active or inactive: the fold is active when partially open and inactive when fully open.

REGION	WHEN IT APPLIES
Outer camera	Always there.
Inner camera	Only when active.
Folding region	Only when the device is partially open.

A **division** is when the fold splits a large view into smaller ones. An **occlusion** is when hardware, like a camera, covers part of your view.

What the system already does

Alerts, context menus, and sheets move around the fold on their own. Split views adjust column widths and margins to match the inner display's symmetry, like Notes evening out its two panes as the device folds.

What you do for custom views

Use the `ReservedRegion` API. SwiftUI: wrap in a `GeometryReader` and call `reservedRegions(kind:options:layoutDirectionBehavior:)` on the proxy. UIKit: call `reservedRegions(kind:options:)` on the view. Inspect each frame and move content out of the way.

Rules of thumb for folding

1. **Prefer containers that adapt automatically** over hand-positioning.
2. **Use an even number of columns in grids** so content divides cleanly down the middle.
3. **Make small adjustments.** Controls that vanish or jump around are hard to track.
4. **Keep nothing important or interactive across the fold.**

08 · ARRANGEMENT VIEWS

One container, two views

An arrangement view holds a **primary** and **secondary** view and arranges them by display size, orientation, and reserved regions. It handles layout only, not navigation.

STYLE	BEHAVIOR	THINK OF IT AS
Split	Primary and secondary share the area. Side by side when wider than tall, stacked when taller than wide.	`HStack` / `VStack`
Overlay	Primary sits on top of secondary when closed or fully open. Partially folded, the views move to either side of the fold.	`ZStack`

COPY-PASTE · SWIFTUI

```
ArrangementView {
    PrimaryView()
} secondary: {
    SecondaryView()
}
.arrangementViewStyle(.split.axes(.horizontal))
```

COPY-PASTE · UIKIT

```
let arrangementVC = UIArrangementViewController()
arrangementVC.setViewController(PrimaryViewController(), for: .primary)
arrangementVC.setViewController(SecondaryViewController(), for: .secondary)
arrangementVC.updateArrangement(.split.axes(.horizontal))
```

Rules

1. **Consider one** when your layout already looks like an `HStack`, `VStack`, or `ZStack` pair. You can restrict the axes, or collapse the secondary view in an overlay.
2. **Keep navigation outside it.** Put navigation split views and tab views around an arrangement view, not inside.
3. **Don't nest it** inside a navigation split view, list, scroll view, or anything that could make part of it unreachable.

09 · SPECIAL CASES

Cameras and games

If your app captures photos or video, or is a game, a few extra rules apply.

APPS WITH A CAMERA

With AVKit or AVFoundation you can now use the **outer-display camera, inner-display camera, and rear camera**. As the device opens, closes, and rotates, your app may switch displays and the active camera may flip direction. Read Apple's *Choosing a camera by the direction it faces*. When the device is fully open and shooting with the rear camera, it can also show content on the outer display. Apple has a separate doc on registering a camera capture accessory for that.

GAMES

- Make the game playable in **every pose**.
- You may lock to portrait or landscape, but **fill the screen** as the pose changes.
- Keep text and control sizes as consistent as possible when resizing.
- Prefer changing the **aspect ratio** over letterboxing or pillarboxing. If you can't avoid it, add artwork to the padding so it still feels full screen.

10 · CASE STUDY

How it played out in Tiles

Here's how the steps above worked in my wealth-building app, Tiles.

OUTER DISPLAY · FOLDED

- Cleaned up corners near the straight hinge edge.
- Moved to system bars so toolbars and navigation sit on the vertical edge.
- Fixed a key button blocked by the camera and system controls by relying on **safe areas** instead of fixed offsets.

INNER DISPLAY · UNFOLDED

- Didn't just stretch the layout to fill the screen. Used **size classes** and tested **Split View**.
- With Tiles on the left side of Split View, controls belong on the **left edge**, the app's outer edge, not floating in the middle.

FOLDING REGION

- Kept important content and interactive elements off the hinge.
- Tiles is built around **mosaic tiles**, so I explored letting tile sizes and arrangement adapt as space changes. The layout reorganizes instead of stretching.

11 · AUDIT PROMPT

Let your AI audit it

A prompt you can adapt for Claude Code or similar. Replace the bracketed part with your own app name.

COPY-PASTE · CLAUDE CODE PROMPT

```
I'm adapting [APP NAME] for iPhone Duo. Read Apple's HIG page
"Designing for iPhone Duo" and the developer doc "Preparing your app
for iPhone Duo" first.

Then audit my project for:
1. Fixed widths, screen-size checks, or UIDevice idiom/orientation checks
2. Custom bars not built on NavigationStack / NavigationSplitView toolbars
3. Toolbar items with a title but no icon, or built from custom views
4. Grids or tile layouts with an odd column count
5. Interactive content that could land in the folding region or under
   the camera

For each finding, show the file, explain the problem, and propose the
smallest fix. Don't redesign anything I haven't asked about.
```

Why this prompt works

a **It reads the source first.** The AI works from Apple's current guidance, not from memory.

b **Five concrete checks** map to the traps in this guide, so findings are specific.

c **"Smallest fix"** keeps it from redesigning screens you didn't ask about.

12 · THE CHECKLIST

Before you ship

Work through this once per screen.

Setup

- [ ] Built with the latest Xcode
- [ ] Tested in the Duo simulator, every pose and rotation

Layout

- [ ] No fixed widths or display-specific assumptions
- [ ] Driven by size classes, margins, safe areas
- [ ] Calculations use scene or container bounds
- [ ] Same functionality and state on both displays

Bars

- [ ] Built on system navigation containers
- [ ] Back/Close first, prominent actions next
- [ ] Items grouped, not manually spaced
- [ ] Visibility priorities set
- [ ] Every item has an icon and a title
- [ ] Overflow actions in the system menu
- [ ] Toolbar vs. tab bar compression chosen per screen

Fold, camera & Split View

- [ ] Nothing important or tappable in the fold
- [ ] Custom views use `ReservedRegion`
- [ ] Grids use an even number of columns
- [ ] Layout changes on fold are small
- [ ] Controls sit on the app's outer edge
- [ ] Content isn't covered by opposite-edge controls

13 · QUICK REFERENCE

SwiftUI vs. UIKit

The API for each task, side by side.

TASK	SWIFTUI	UIKIT
Resize by size class	`horizontalSizeClass`	Automatic trait tracking
Safe areas	`GeometryProxy.safeAreaInsets`	`UIView.safeAreaInsets`
Reserved regions	`GeometryProxy.reservedRegions(...)`	`UIView.reservedRegions(...)`
Arrangement view	`ArrangementView`	`UIArrangementViewController`
Split view	`NavigationSplitView`	`UISplitViewController`
Group toolbar items	`ToolbarItemGroup`	`UIBarButtonItemGroup`
Visibility priority	`visibilityPriority(_:)`	`UIBarButtonItem.visibilityPriority`
Overflow menu	`ToolbarOverflowMenu`	`additionalOverflowItems`
Compression behavior	`ToolbarVerticalCompressionBehavior`	`UIVerticalBarCompressionBehavior`
Opt out of vertical sheet bars	`toolbarVerticalBehavior(_:)`	`preferredVerticalBarBehavior`
Detect vertical bars	`toolbarVerticalEdge`	`verticalBarEdge` trait
Hero image under bar	`backgroundExtensionEffect()`	`UIBackgroundExtensionView`

14 · RECAP

What you learned

Four habits cover most of the work.

1 **Build and test first.** Latest Xcode, Duo simulator, every pose, every sheet.

2 **Make it resizable.** Size classes, safe areas, container bounds. No fixed widths.

3 **Let the system draw the bars.** Icon plus title on every item, priorities for the actions that matter.

4 **Keep the fold and camera clear.** Nothing important across the hinge, even column counts, small adjustments.

Further reading

RESOURCE	WHERE
Designing for iPhone Duo	Apple HIG · developer.apple.com/design/human-interface-guidelines/designing-for-iphone-duo
Preparing your app for iPhone Duo	Apple developer docs · developer.apple.com/documentation/TechnologyOverviews/preparing-your-app-for-iphone-duo
Layout and Designing for iOS	Apple HIG · developer.apple.com/design/human-interface-guidelines
Screenshot specifications	App Store Connect help. Worth checking before you ship.

This guide is part of the **AI Coding Guides** series, covering the full path from ideation to App Store launch using AI-assisted development.