Custom Responsive Widgets Documentation (cwp.dart)

This document details the custom responsive widgets and utilities located in your cwp.dart file. These tools are designed to simplify building adaptive user interfaces in Flutter, ensuring your app looks and functions great across various screen sizes and orientations.

1. Core Concepts

Your cwp.dart file is built upon a few foundational concepts that enable responsive design.

1.1 DeviceType Enum

This enumeration categorizes the current device into distinct types based on its screen width and orientation.

- mobile: Screen width less than 600 logical pixels, portrait orientation.
- mobileLandscape: Screen width less than 600 logical pixels, landscape orientation.
- tablet: Screen width between 600 and 1000 logical pixels, portrait orientation.
- tabletLandscape: Screen width between 600 and 1000 logical pixels, landscape orientation.
- desktop: Screen width 1000 logical pixels or greater.

1.2 Breakpoints

These are the pixel thresholds that define the different DeviceType categories:

- **Mobile Breakpoint**: 600.0 logical pixels.
- **Tablet Breakpoint**: 1000.0 logical pixels.

You can modify these _mobileBreakpoint and _tabletBreakpoint constants at the top of your cwp.dart file if your design requires different device categorization.

1.3 getDeviceType(BuildContext context)

This is a utility function that determines and returns the current DeviceType based on the MediaQuery of the provided context. It's used internally by many of your responsive widgets.

1.4 DeviceTypeExtension on BuildContext (New Feature)

This extension makes checking the current device type and orientation much more convenient within your widgets. Instead of calling getDeviceType(context) repeatedly, you can now use direct getters on your BuildContext.

- context.baseDeviceCategory: Returns the base DeviceType (mobile, tablet, desktop) ignoring orientation.
- context.orientation: Returns the current screen Orientation.

- context.isMobile: Returns true if the device is a mobile in portrait.
- context.isMobileLandscape: Returns true if the device is a mobile in landscape.
- context.isTablet: Returns true if the device is a tablet in portrait.
- context.isTabletLandscape: Returns true if the device is a tablet in landscape.
- context.isDesktop: Returns true if the device is a desktop.

Usage Example:

```
if (context.isDesktop) {

  // Build desktop-specific UI

} else if (context.isMobile) {

  // Build mobile-specific UI

}
```

1.5 ResponsiveValue<T> Utility (Corrected & Enhanced)

This is a **core utility** that allows you to define different values for a property (of any type T) across various device types and orientations. It simplifies the process of applying responsive sizing, spacing, or counts.

- **Purpose**: To retrieve a value that changes based on the screen's characteristics without writing repetitive switch statements.
- **Properties**:
    - mobile, mobileLandscape, tablet, tabletLandscape, desktop: Values of type T for each responsive category.
- **Method**:
    - getValue(BuildContext context): Returns the appropriate value based on the current device and orientation. It includes a fallback mechanism: if a specific value (e.g., tablet) is null, it will fall back to the next available smaller device value (e.g., mobile).

Usage Example:

```
// Define how a margin should change per device

final EdgeInsets horizontalMargin = ResponsiveValue<EdgeInsets>(

  mobile: const EdgeInsets.symmetric(horizontal: 16.0),

  tablet: const EdgeInsets.symmetric(horizontal: 32.0),
```

```
  desktop: const EdgeInsets.symmetric(horizontal: 64.0),
).getValue(context);

// Apply it
Padding(
  padding: horizontalMargin,
  child: Text('Content'),
);
```

2. Widget Reference

Here are the details for each custom responsive widget in your `cwp.dart` file.

2.1 ResponsiveVisibility

Conditionally shows or hides its child widget based on the current `DeviceType` and orientation. When hidden, the widget is completely removed from the widget tree.

- **Purpose**: To render specific UI elements only on certain device types or orientations (e.g., a bottom navigation bar on mobile only).
- **Properties**: `child` (required `Widget`), `showForMobile`, `showForMobileLandscape`, `showForTablet`, `showForTabletLandscape`, `showForDesktop` (all `bool`, with defaults).
- **Usage Example**:

```
ResponsiveVisibility(
  showForDesktop: true,
  showForMobile: false,
  showForTablet: false,
  child: Text('This banner is only visible on desktop.'),
)
```

-

2.2 ResponsiveSizedBox

Sizes its child either to a percentage of the screen dimensions or to a fixed size.

- **Purpose**: To create flexible boxes whose dimensions scale with the screen or maintain specific pixel sizes.
- **Properties**: child (required Widget), widthPercentage, heightPercentage (doubles 0.0-1.0), fixedWidth, fixedHeight (doubles, override percentages).
- **Usage Example**:

```
ResponsiveSizedBox(

  widthPercentage: 0.8, // 80% of screen width

  heightPercentage: 0.3, // 30% of screen height

  child: Container(color: Colors.blue, child: Center(child: Text('80% Width, 30% Height'))),

)
```

-

2.3 CustomPadding

Applies padding to its child widget using a flexible property system.

- **Purpose**: Provides a more declarative way to specify padding compared to standard EdgeInsets, though it doesn't directly use ResponsiveValue for breakpoints within itself.
- **Properties**: child (required Widget), all, horizontal, vertical, top, bottom, left, right (all double?).
- **Property Priority**: all > (horizontal OR vertical) > individual sides (top, bottom, left, right).
- **Usage Example**:

```
CustomPadding(

  all: 16.0,

  child: Text('Padded equally on all sides.'),

)


CustomPadding(
```

```
  horizontal: 24.0,
  top: 10.0,
  child: Text('Padded horizontally and on top.'),
)
```

-

2.4 ResponsiveText

Displays text with a font size that adapts to the DeviceType and orientation.

- **Purpose**: Ensures text readability and aesthetic consistency across different screen sizes without manual MediaQuery checks for each text widget.
- **Properties**: text (required String), textStyle (TextStyle?), textAlign (TextAlign?), mobileFontSize, mobileLandscapeFontSize, tabletFontSize, tabletLandscapeFontSize, desktopFontSize (all double?).
- **Usage Example**:

```
ResponsiveText(
  text: 'Adaptive Title for All Screens',
  mobileFontSize: 18.0,
  tabletFontSize: 22.0,
  desktopFontSize: 28.0,
  textStyle: TextStyle(fontWeight: FontWeight.bold, color: Colors.deepPurple),
  textAlign: TextAlign.center,
)
```

-

2.5 ResponsiveConstraintBox

Applies minimum and maximum width/height constraints to its child based on the DeviceType and orientation.

- **Purpose**: Prevents widgets from becoming too small or too large, ensuring they stay within desirable bounds on different screens.

- **Properties**: child (required Widget), and various minWidth..., maxWidth..., minHeight..., maxHeight... properties for each DeviceType and orientation.
- **Usage Example**:

```
ResponsiveConstraintBox(

  maxWidthDesktop: 800.0, // Max width 800px on desktop

  minWidthMobile: 300.0,  // Min width 300px on mobile

  maxHeightTabletLandscape: 400.0, // Max height 400px on tablet landscape

  child: Container(color: Colors.green, child: Text('Constrained Box')),

)
```

-

2.6 ResponsiveSpacer

Provides dynamic spacing between widgets, adapting its width or height based on the DeviceType and orientation.

- **Purpose**: To manage responsive gaps in Row or Column layouts.
- **Properties**: widthMobile, heightMobile, etc. (doubles) for different DeviceType and orientations.
- **Usage Example**:

```
Row(

  children: [

    Text('Item A'),

    ResponsiveSpacer(widthMobile: 10.0, widthDesktop: 30.0), // Smaller horizontal gap on mobile, larger on desktop

    Text('Item B'),

  ],

)

Column(

  children: [
```

```
  Text('Header'),
  ResponsiveSpacer(heightMobile: 5.0, heightTablet: 15.0), // Vertical gap
  Text('Content'),
 ],
)
```

-

2.7 ResponsiveBuilder

A powerful generic widget that provides the current DeviceType and Orientation directly to its builder function.

- **Purpose**: When ResponsiveVisibility isn't enough, and you need to build entirely different widget trees, apply complex logic, or pass device-specific values to children based on the responsive context.
- **Properties**: builder (required ResponsiveWidgetBuilder).
- **Usage Example**:

```
ResponsiveBuilder(
 builder: (context, deviceType, orientation) {
  if (deviceType == DeviceType.desktop) {
   return const Text('This is your desktop experience!');
  } else if (deviceType == DeviceType.mobile && orientation == Orientation.portrait) {
   return const Text('Mobile portrait view.');
  } else {
   return const Text('Tablet or Mobile Landscape view.');
  }
 },
)
```

-

2.8 `ResponsiveLayoutGrid` (Corrected & Enhanced)

Creates a `GridView` whose number of columns adapts dynamically to the `DeviceType` and orientation.

- **Purpose**: To build responsive grid layouts common in dashboards, galleries, or product listings, ensuring optimal column count per device.
- **Properties**: `children` (required `List<Widget>`), `columns` (required `ResponsiveValue<int>` - defining column counts for each device type), `mainAxisSpacing`, `crossAxisSpacing`, `childAspectRatio`.
- **Usage Example**:

```
ResponsiveLayoutGrid(
  columns: const ResponsiveValue<int>(
    mobile: 1,           // 1 column on mobile portrait
    mobileLandscape: 2,  // 2 columns on mobile landscape
    tablet: 2,           // 2 columns on tablet portrait
    tabletLandscape: 3,  // 3 columns on tablet landscape
    desktop: 4,          // 4 columns on desktop
  ),
  mainAxisSpacing: 10.0,
  crossAxisSpacing: 10.0,
  children: [
    // Your grid items here (e.g., ProductCard, DashboardMetric)
    Container(color: Colors.red, height: 100),
    Container(color: Colors.green, height: 100),
    Container(color: Colors.blue, height: 100),
  ],
)
```

-

2.9 `CustomPageView` (New Feature)

A responsive `PageView` that simplifies configuring its `viewportFraction` and overall dimensions based on device type.

- **Purpose**: Ideal for creating dynamic carousels or swipeable sections where you want to show a different number of partial items on screen based on device size.
- **Properties**: Standard `PageView` properties (`controller`, `itemBuilder`, `itemCount`, etc.), plus `viewportFraction` (`ResponsiveValue<double>`), `widthPercentage` (`ResponsiveValue<double>?`), `heightPercentage` (`ResponsiveValue<double>?`).
- **Factory Constructor**: `CustomPageView.responsive()` provides convenient defaults for common responsive carousel patterns.
- **Usage Example**:

```
CustomPageView.responsive(
  itemBuilder: (context, index) {
    return Container(
      margin: const EdgeInsets.symmetric(horizontal: 8.0),
      color: Colors.primaries[index % Colors.primaries.length],
      child: Center(
        child: Text('Page ${index + 1}', style: const TextStyle(fontSize: 24, color: Colors.white)),
      ),
    );
  },
  itemCount: 10,
  // Default responsive behavior:
  // mobile: 1.0 viewport (full page)
  // tablet: 0.8 viewport (shows part of next page)
  // desktop: 0.5 viewport (shows two pages)
)
```

-

2.10 `ResponsiveIcon` (New Feature)

Displays an `Icon` whose `size` automatically changes based on the `DeviceType` and `Orientation`.

- **Purpose**: Ensures icons are always appropriately sized for readability and aesthetics across all devices.
- **Properties**: `icon` (required `IconData`), `size` (required `ResponsiveValue<double>`), `color` (`Color?`).
- **Usage Example**:

```
ResponsiveIcon(

  icon: Icons.star,

  size: const ResponsiveValue<double>(

    mobile: 24.0,

    tablet: 32.0,

    desktop: 40.0,

  ),

  color: Colors.amber,

)
```

-

2.11 `ResponsiveButton` (New Feature)

A versatile button widget (defaulting to `ElevatedButton`) that automatically adjusts its `padding` and internal `textStyle` based on the device.

- **Purpose**: Maintains consistent touch targets and text legibility for buttons on any screen size, enhancing overall UI/UX. Uses `WidgetStateProperty` for modern Flutter versions.
- **Properties**: `child` (required `Widget`), `onPressed` (`VoidCallback?`), `style` (`ButtonStyle?`), `padding` (`ResponsiveValue<EdgeInsetsGeometry>?`), `textStyle` (`ResponsiveValue<TextStyle>?`).
- **Usage Example**:

```
ResponsiveButton(

  onPressed: () { /* Handle tap */ },
```

```
  child: const Text('Tap Me'),
  padding: const ResponsiveValue(
    mobile: EdgeInsets.symmetric(horizontal: 20, vertical: 10),
    tablet: EdgeInsets.symmetric(horizontal: 30, vertical: 15),
    desktop: EdgeInsets.symmetric(horizontal: 40, vertical: 20),
  ),
  textStyle: const ResponsiveValue(
    mobile: TextStyle(fontSize: 16),
    desktop: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
  ),
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.blue,
    foregroundColor: Colors.white,
  ),
)
```

-

2.12 `ResponsiveIconButton` (New Feature)

An `IconButton` whose `iconSize` and `padding` adapt automatically based on the device type.

- **Purpose**: Ensures icon buttons are visually consistent and easy to tap on all devices.
- **Properties**: `onPressed` (`VoidCallback?`), `icon` (required `IconData`), `iconSize` (required `ResponsiveValue<double>`), `padding` (`ResponsiveValue<EdgeInsetsGeometry>?`), `color` (`Color?`), `tooltip` (`String?`).
- **Usage Example**:

```
ResponsiveIconButton(
  onPressed: () { /* Handle tap */ },
  icon: Icons.settings,
```

```
  iconSize: const ResponsiveValue<double>(
    mobile: 24.0,
    tablet: 28.0,
    desktop: 32.0,
  ),
  color: Colors.grey[700],
  tooltip: 'App Settings',
)
```

-

3. General Usage and Modification

3.1 Applying the Widgets to Your Project

1. **Save the file**: Ensure the entire code provided is saved as `cwp.dart` in your project's root directory (outside the `lib` folder), as per your project structure.
2. **Import**: In any `.dart` file where you want to use these widgets (e.g., `home_page.dart`, `product_detail_page.dart`), add the import at the top:

```
import 'package:your_app_name/../cwp.dart'; // Adjust 'your_app_name' to your actual project name
```

3.
4. **Ensure `MaterialApp`**: Always ensure your application's root widget is a `MaterialApp` (or `WidgetsApp`). All these responsive widgets rely on `MediaQuery` which is provided by `MaterialApp`.

3.2 Modifying Breakpoints

To adjust the screen width thresholds for `mobile` and `tablet` devices:

1. Open your `cwp.dart` file.
2. Locate the constants `_mobileBreakpoint` and `_tabletBreakpoint`.
3. Change their `double` values to your desired pixel widths. For example:

```
const double _mobileBreakpoint = 650.0; // New mobile breakpoint
const double _tabletBreakpoint = 1100.0; // New tablet breakpoint
```

4.
5. All widgets in your app that use these responsive utilities will automatically adapt to the new breakpoints.

3.3 Extending Functionality

You can easily extend these utilities:

- **Add more `DeviceType` categories**: If you need more granular control (e.g., `largeTablet`, `smallDesktop`), add them to the `DeviceType` enum and update `getDeviceType` and `ResponsiveValue` accordingly.
- **Create new responsive widgets**: Use `ResponsiveValue<T>` as the foundation to create custom widgets that respond to specific properties, just like `ResponsiveIcon` or `ResponsiveButton` do for `size` or `padding`.

By leveraging this comprehensive set of responsive widgets, you can build beautiful, adaptable Flutter applications with cleaner, more maintainable code!