Flutter
Updating the Material Buttons and their Themes

SUMMARY

Update the Material library button widgets and themes, to make them easier to understand and use.

Author: Hans Muller (@hansmuller)
Go Link: flutter.dev/go/material-button-system-updates
Created: April 2020 / **Last updated:** April 2020

The changes defined in this document have landed in Flutter, see #59702. The class originally called "ContainedButton" in this proposal and #59702, has been renamed ElevatedButton, see #61262. The "Migrating to the New Material Buttons" explains how to convert existing applications to the new button widgets.

OBJECTIVE

Make it easy to configure the visual properties of the standard Material button widgets using the overall application Theme and component-specific themes. Simple tasks, like configuring the text color for all TextButtons, should be simple. Complex tasks, like overriding the text color of OutlinedButtons when they're hovered or focused, should be possible.

BACKGROUND

The "Buttons" section of flutter.dev/go/material-theme-system-updates discusses existing problems with configuring material buttons using the app's overall Theme and ButtonTheme et al. To summarize: it's difficult to explain how the overall Theme and its ButtonTheme contribute to button defaults, and it's particularly difficult to apply these themes to produce even simple changes.

Issue #19623 is a good example. Developers were unable to use the themes to specify the text color of RaisedButtons or FlatButtons and they were unable to divine how one might do so from the API docs.

The root of this problem is that the Theme's "grab bag" properties, like disabledColor and buttonColor, its textTheme's button TextStyle, and its ButtonTheme, all contribute to the final color value. Over the years, evolving requirements and backwards compatibility have made a thicket of the interdependencies between these properties. On top of that, ButtonTheme's getTextColor() method produces a value that additionally factors in the following:

- The button's disabled/pressed/hovered/focused state
- The overall Theme's brightness
- The type of the button. FlatButton, RaisedButton, and OutlineButton have different default text colors.
- The ButtonTheme’s ButtonTextTheme which is not a theme and doesn't just affect button text.

The final hurdle is ButtonBar. Widgets that incorporate a ButtonBar, like AlertDialog or DatePicker, wrap their button children with a ButtonTheme that specifies ButtonTextTheme.primary (not a TextTheme), which effectively overrides the button's text color.

It’s these problems, and others like them, that have driven the decision to create a new set of widgets and themes alongside the old ones, rather than trying to evolve the existing ones.

Glossary

- **ColorScheme** - A set of eleven colors that all of the default Material component colors are based on. Apps can define “dark” and “light” color schemes.
- **MaterialStateProperty<T>** - an object that can be “resolved” to a value of type T given a set of MaterialStates. The existing button widgets keep track of their state and can resolve MaterialStateProperty values. For example, one can create a color that resolves to grey when the button is disabled, blue when it’s pressed, and dark blue when it’s enabled and focused or hovered.
- **MaterialStateColor** - a convenience Color subclass that is-a MaterialStateProperty. Can be used as a value of Color parameters and properties. Used to preserve backwards compatible APIs that were originally defined in terms of Color, but whose implementation also handles resolving MaterialStateProperties.

OVERVIEW

It would not be practical to evolve the existing button and button theme systems in place with "soft" breaking changes. This proposal is about adding a new system that addresses the problems with the old one. Over time the old system will be deprecated and then, in the distant future, removed.

This is not a breaking change. The semantics of FlatButton, OutlineButton, RaisedButton, ButtonBar, ButtonBarTheme, and ButtonTheme will not change.

The current Material Design spec uses different names for the basic button types. New button widgets and themes with names that match the current spec will be added to the Material library. Eventually the old widgets and themes will be deprecated.

Old Widget	Old Theme	New Widget	New Theme
FlatButton	ButtonTheme	TextButton	TextButtonTheme
RaisedButton	ButtonTheme	ElevatedButton	ElevatedButtonTheme
OutlineButton	ButtonTheme	OutlinedButton	OutlinedButtonTheme

The existing ButtonTheme can be configured with MaterialStateProperties. The new widgets and themes will support MaterialStateProperties in widget constructor parameters and in theme properties. The button implementations will allow MaterialStateProperty values that resolve to null, to defer, just as null widget parameters defer to theme properties and null theme properties defer to the widget's default.

Because most buttons have most visual properties in common, we'll introduce a new *ButtonStyle* class that contains all of the common properties. ButtonStyle is analogous to TextStyle. Most ButtonStyle properties will be MaterialStateProperties.

All of a button's default colors will depend on the Theme's ColorScheme which has dark and light variants. All of a button's default colors are defined exclusively by the button's widget class and will not depend on the overall Theme's brightness.

Although the button widgets are visually similar there are important differences. Some of them are obvious, e.g. ElevatedButton's background is the color scheme's primary color and TextButton's background is transparent. Some are subtle, e.g. ElevatedButton's overlay color is the color scheme's primary color with 24% opacity and TextButton is the same color with 12% opacity.

To configure a button in terms of the color scheme colors that it depends on, each of the button classes includes a static method that returns a ButtonStyle given one or more color scheme colors. This method essentially exposes the button's visual defaults and is the preferred basis for simple customizations.

DETAILED DESIGN/DISCUSSION

ButtonStyle

This is a new class. It is a simple container for the properties that most buttons have in common. In addition to reducing the size of the APIs for the button widgets and their themes, the ButtonStyle class makes it possible to define methods that produce or operate on this collection of properties as a whole.

```
class ButtonStyle {
  const ButtonStyle({
    this.textStyle,
    this.backgroundColor,
    this.foregroundColor,
    this.overlayColor,
    this.shadowColor,
    this.elevation,
    this.padding,
    this.minimumSize,
    this.side,
    this.shape,
    this.mouseCursor,
    this.visualDensity,
    this.tapTargetSize,
    this.animationDuration,
    This.enableFeedback,
  });

  final MaterialStateProperty<TextStyle> textStyle;
  final MaterialStateProperty<Color> backgroundColor;
  final MaterialStateProperty<Color> foregroundColor;
  final MaterialStateProperty<Color> overlayColor;
  final MaterialStateProperty<Color> shadowColor;
  final MaterialStateProperty<double> elevation;
  final MaterialStateProperty<EdgeInsetsGeometry> padding;
  final MaterialStateProperty<Size> minimumSize;
  final MaterialStateProperty<BorderSide> side;
  final MaterialStateProperty<ShapeBorder> shape;
  final MaterialStateProperty<MouseCursor> mouseCursor;
  final VisualDensity visualDensity;
  final MaterialTapTargetSize tapTargetSize;
  final Duration animationDuration;
  final bool enableFeedback;

  // ... copyWith(), merge()
}
```

Each of ButtonStyle’s MaterialStateProperty attributes resolves to a primitive value based on the button’s state. This is essential because the Material spec and the Google Material variant of the Material spec, define visual property values in terms of a color scheme, a text theme, *and* the button’s state.

All of the ButtonStyle properties are null by default. The button widgets themselves are responsible for defining comprehensive defaults for all of relevant style properties.

A ButtonStyle property can override a value for just one state or all of them. For example to create an ElevatedButton whose background color is the color scheme’s primary color with 50% opacity, but only when the button is pressed, one could write:

```
ElevatedButton(
  style: ButtonStyle(
    backgroundColor: MaterialStateProperty.resolveWith<Color>(
      (Set<MaterialState> states) {
        if (states.contains(MaterialState.pressed))
          return Theme.of(context).colorScheme.primary.withOpacity(0.5);
        return null; // default to the component's default
      },
    ),
  ),
)
```

In this case the background color for all other button states would fallback to the ElevatedButton’s default values.

Configuring a ButtonStyle directly enables one to very precisely control the button’s visual attributes for all states. This level of control is typically required when a custom “branded” look and feel is desirable. However, in many cases it’s useful to make relatively sweeping changes based on a few initial parameters. The button styleFrom() methods enable such sweeping changes.

Button styleFrom() methods

As noted in the overview, one of the basic failings of the existing system is that it’s not clear how to configure the color of text buttons. We’ll refer to this color as the button’s *foregroundColor* and it will be used to render the button’s text and icon descendants. One of the reasons this is trickier than it seems, is that there are other colors that - by default - depend on the foregroundColor: the ink color for the button’s ripple (splash), the button’s focus overlay color, and the button’s hover overlay color.

That said, most developers would prefer to just override the default foreground color and have the dependent colors computed for them. The TextButton API doc will explain that the button's foreground color depends on the color scheme's primary color. TextButton will provide a static *styleFrom* method that computes a ButtonStyle given a primary color:

```
ThemeData(
  textButtonTheme: TextButtonThemeData(
    style: TextButton.styleFrom(primary: Colors.green)
  ),
)
```

The ElevatedButton API doc will explain that the button's foreground color depends on the color scheme's onPrimary color and its background depends on the primary color. So to override the foreground and background and all of the dependent colors:

```
ThemeData(
  elevatedButtonTheme: ElevatedButtonThemeData(
    style: ElevatedButton.styleFrom(
      onPrimary: red,
      primary: Colors.green,
    ),
  ),
)
```

To additionally specify the disabled color, which is documented as depending on the color scheme's onSurface color:

```
ThemeData(
  elevatedButtonTheme: ElevatedButtonThemeData(
    style: ElevatedButton.styleFrom(
      onPrimary: red,
      primary: Colors.green,
      onSurface: Colors.blue
    ),
  ),
)
```

The Button Classes

The button classes themselves have new names and a single ButtonStyle parameter instead of a collection of individual color/shape/etc parameters. In other respects they are essentially the same as the original button classes.

The TextButton class is a substitute for FlatButton.

```
class TextButton {
  const TextButton({
    Key key,
    @required VoidCallback onPressed,
    VoidCallback onLongPress,
    ButtonStyle style,
    Clip clipBehavior = Clip.none,
    FocusNode focusNode,
    bool autofocus = false,
    Widget child,
  });

  factory TextButton.icon({
    // Same parameters as TextButton except
    // that child is replaced by icon and label.
    @required Widget icon,
    @required Widget label,
  });

  static ButtonStyle styleFrom({
    Color primary,
    Color onSurface,
  });
}
```

The ElevatedButton is a substitute for RaisedButton. It's API is identical to TextButton except that it's styleFrom() method includes an onPrimary parameter. The ElevatedButton's visual defaults are *not* the same TextButton.

OutlinedButton is a substitute for OutlineButton and it also has the same API as TextButton as well as slightly different defaults for its visual parameters.

The Button Theme Classes

There are three new themes, one for each of the new button classes.

```
class TextButtonTheme extends InheritedTheme {
  const TextButtonTheme({
    Key key,
    @required this.data,
    Widget child,
  }) : assert(data != null), super(key: key, child: child);
  final TextButtonThemeData data;
  static TextButtonThemeData of(BuildContext context);
}
```

```
class TextButtonThemeData with Diagnosticable {
  const TextButtonThemeData({ this.style });
  final ButtonStyle style;
}
```

The theme classes for OutlinedButton and ElevatedButton are the same.

The button themes are “normalized” as defined in flutter.dev/go/material-theme-system-updates. That means that all of the theme properties are null by default, just like the button constructor parameters. Similarly, by default ThemeData.textButtonTheme, ThemeData.elevatedButtonTheme, and ThemeData.outlinedButtonTheme are all null by default.

Button Defaults

The button widget classes are the source of truth about default values. The themes do not define defaults, they just provide a way to override the defaults defined by the button classes themselves. The button classes do not use default constructor parameter values, all constructor parameters are null by default as well.

The button classes, like most “modern” Material widget classes, compute default values based on constructor parameters, theme values, and widget state. The classes document the defaults’ dependencies.

The button implementations will continue to bottom out to the RawMaterialButton class, which is a useful reference for what aspects buttons are configurable at all.

Generally speaking, the approach the widget classes will take to computing the values used to configure the RawMaterialButton they’re building, is the same as it has been:

```
value = widget.value ?? Theme.of(context).theme?.value ?? _defaultValue
```

In other words, if a non-null widget parameter doesn’t exist then defer to the corresponding button theme value, and if a non-null theme value doesn’t exist then defer to widget’s internal default. The internal default value will typically depend on the overall theme’s colorScheme and textTheme.

In practice this is a little more complicated because many widget parameters, theme values and internal default values are MaterialStateProperties. A MaterialStateProperty can resolve to null, which means (as before) that the implementation should defer. Additionally the button’s theme has-a ButtonStyle valued style property, So:

```
value = widget.value?.resolve(states)
  ?? Theme.of(context).theme?.style?.value?.resolve(states)
  ?? _defaultValue.resolve(states);
```

The tables that follow enumerate the default button style values for the new button classes. Rows are ButtonStyle properties, columns are button states. The "active" just means: not one of the other states. There are a few abbreviations to make the tables manageable:

- In the textStyle row, "button" means the button text style from the overall theme's textTheme.
- P8 means EdgeInsets.all(8), P16 means EdgeInsets.all(16).
- RR4 Means a rounded rectangle shape with all corner radii = 4.0. RR4 does not draw its border.
- The minimumSize height values represent the area the button will paint. The button will accept input within the area defined by the overall theme's minimumTapTargetSize, and it will occupy that much space in its parent's layout.

TextButton

Defaults depend on the overall Theme's textTheme.button TextStyle and its colorScheme primary and onSurface colors.

	active	hovered	focused	pressed	disabled
textStyle	button				
backgroundColor	transparent				
foregroundColor	primary				onSurface(38%)
overlayColor		primary(4%)	primary(12%)	primary(12%)	
elevation	0				
padding	P8				
minimumSize	64x36				
shape	RR4				

ElevatedButton

Defaults depend on the overall Theme's textTheme.button TextStyle and its colorScheme primary, onPrimary, and onSurface colors.

	active	hovered	focused	pressed	disabled
textStyle	button				

backgroundColor	primary				onSurface(12%)
foregroundColor	onPrimary				onSurface(38%)
overlayColor		onPrimary(8%)	onPrimary(24%)	onPrimary(24%)	
elevation	2	4	4	8	0
padding	P16				
minimumSize	64x36				
shape	RR4				

OutlinedButton

Defaults depend on the overall Theme's textTheme.button TextStyle and its colorScheme primary, and onSurface colors.

The value of OBRR is rounded rectangle shape with all corner radii = 4.0 and a onSurface(12%) border with width 1.

	active	hovered	focused	pressed	disabled
textStyle	button				
backgroundColor	transparent				
foregroundColor	primary				onSurface(38%)
overlayColor		primary(4%)	primary(12%)	primary(12%)	
elevation	0				
padding	P16				
minimumSize	64x36				
shape	OBRR				

What's Being Left Behind

The new classes outlined in this document leave behind a considerable swath of the existing API. The old API will eventually be deprecated, but it will not change.

Button Widgets

The existing FlatButton, RaisedButton, and OutlineButton classes have a cloud of color/elevation and other constructor parameters that enable on to configure their appearance. This proposal introduces new replacement button classes : TextButton, ElevatedButton, and OutlinedButton.

In large part, the existing buttons classes have so many constructor parameters because they encode state in the parameter names. RaisedButton is typical:

```
ButtonTextTheme textTheme,
Color textColor
Color disabledTextColor
Color color,
Color disabledColor
Color focusColor
Color hoverColor
Color highlightColor
Color splashColor
Brightness colorBrightness
double elevation,
double focusElevation
double hoverElevation
double highlightElevation
double disabledElevation
EdgeInsetsGeometry padding
VisualDensity visualDensity
ShapeBorder shape
```

There are many problems with this, for example:

- It doesn't scale well. Additional state-qualified parameters must be added for every possible state or combination of states that a basic parameter might depend on.
- Many of the parameters are not represented directly by ButtonTheme.
- Applying the same set of parameter values to more than one button is messy.
- It's not clear how to handle light or dark modes. The colorBrightness parameter only applies to the button's textColor, and only if the textColor parameter is unspecified.

The ButtonStyle class encapsulates all of these parameters and uses MaterialStateProperty to avoid the explosion of state-qualified parameters.

ButtonStyle leaves the archaic (ButtonTextTheme) textTheme and colorBrightness parameters behind. There's also no longer a need for separate splash and highlight colors. Although InkWell will continue to support both, ButtonStyle only needs one. It's called overlayColor as in the current Material spec.

ButtonStyle does not support defining dark or light (brightness) dependent values. Applications that define custom themes will continue to have to create one ThemeData for each brightness, with different ButtonStyle values for each one.

ThemeData colors and themes

Overrides of the default visual attributes of the existing FlatButton, RaisedButton, and OutlinedButton classes can be defined with ButtonTheme as well as a grab-bag of ThemeData properties.

The new button classes do not depend on the Theme's buttonTheme.

The following ThemeData colors do not contribute to the visual defaults for the new button classes. The comments very briefly cover how the color was used by FlatButton, RaisedButton, OutlineButton. Some of the colors are listed as "no longer used" because, although the buttons don't depend on them anymore, they have depended on them in the past.

```
buttonColor        // RaisedButton background
primaryColor       // Documented but not used
accentColor        // Documented but not used
canvasColor        // OutlineButton background if elevation > 0
focusColor         // All buttons
hoverColor         // All buttons
highlightColor     // All buttons
splashColor        // All buttons except FlatButton...
disabledColor      // No longer used
```

The following component theme valued ThemeData properties do not contribute to the visual defaults for the new button classes. Some of them are rarely used anywhere within the Material library however, most of them, at one time or another, have seemed as though they should contribute to the buttons' appearance.

```
buttonTheme        // All buttons
primaryTextTheme   // No longer used
accentTextTheme    // No longer used
iconTheme          // No longer used
primaryIconTheme   // No long used
accentIconTheme    // No longer used
```