Flutter Material Theme System Updates

SUMMARY

Update the Material library classes related to themes, to make them easier to understand and use.

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

OBJECTIVE

This document describes changes to the Material library theme classes. We've considered making sweeping updates to the theme classes before, however we were stymied by the need to remain backwards compatible. We're going to try again, given the new Flutter breaking change policy and the tolerance for breaking changes that was shown in recent Flutter developer surveys.

Here is what we're trying to accomplish by updating Material's theme support.

Migrate to the latest Material Design terms and parameters

Of course we've never stopped migrating Flutter, however in the past we've been particularly cautious about changing APIs to match the latest Material terminology. Among the more difficult changes we're planning now: limiting the theme dependencies of each component to the overall theme's color scheme, text theme, and one component specific theme. We also plan to move to the latest text style names in the text theme.

Easy to configure the appearance of all components of a particular type

This is the final push in an ongoing effort. A theme will be defined for each significant Material component type and each of those themes will be a property of the overall theme.

Easy to explain a component's visuals in terms of the themes

The relationship between a component's visual properties and the themes should be clear from the component's API doc. Similarly the impact of each visual property in a theme should be clear from the theme's API doc. New Flutter components do this better, see for example ToggleButtons and ToggleButtonsTheme. Older components are sometimes relatively difficult to understand, see for example the Material button widgets. In addition to improving documentation, we're going to regularize the APIs of the existing themes, so that they all conform to a consistent, easily understood, pattern.

OVERVIEW

The following subsections motivate and describe two large projects that collectively address the goals listed above. Both projects include breaking changes. The projects are covered in greater depth in the "Design" section that follows.

Component Theme Normalization

The overall goal of this project is to "normalize" the definitions of all of the existing component themes and to add new normalized component themes for components that don't already have them.

How Did We Get Here

In late 2015, the ThemeData class had 5 named parameters and 15 properties. There was no overall growth plan.

```
class ThemeData {
  ThemeData({
    ThemeBrightness brightness: ThemeBrightness.light,
    Map<int, Color> primarySwatch,
    Color accentColor,
    this.accentColorBrightness: ThemeBrightness.dark,
    TextTheme text
  })
  final ThemeBrightness brightness;
  final Color canvasColor;
  final Color cardColor;
  final Color dividerColor;
  final Color hintColor;
  final Color highlightColor;
  final Color selectedColor;
  final double hintOpacity;
  final TextTheme text;
  Color get primaryColor
  TextTheme get primaryTextTheme
  IconThemeData get primaryIconTheme
  Color accentColor
  ThemeBrightness accentColorBrightness;
}
```

Now, about four years later, ThemeData has 64 named parameters and 62 properties.

Of the 62 properties, 13 are generic values (they appear to apply to all widgets). For example focusColor and hoverColor apply to nearly all of the components, although only when the components are part of apps running on desktop platforms.

Another 21 properties are values that sound generic but only apply to a few components. For example canvasColor is used for the background color in the Material, ProgressIndicator, OutlineButton, and Dropdown menu widgets. ThemeData's backgroundColor property is used for the background color in the LinearProgressIndicator, the date and time pickers, the Stepper widget, and the SnackBar widget.

And finally 28 of the properties are complex theme values. About 20 of the complex theme values apply to just one type of Material component (so they are "component themes"). For example the toggleButtonsTheme property applies to ToggleButtons. Some of the old theme valued properties see very limited use. For example the accentIconTheme property is only used by FloatingActionButton, and only its color attribute is used there.

The Interdependent ThemeData Problem

Complex dependencies are another aspect of the overall Material theme's properties that can make them difficult to understand and apply. Here's just a small fragment of the ThemeData constructor.

```
final bool isDark = brightness == Brightness.dark;
primarySwatch ??= Colors.blue;
primaryColor ??= isDark ? Colors.grey[900] : primarySwatch;
primaryColorBrightness ??= estimateBrightnessForColor(primaryColor);
primaryColorLight ??= isDark ? Colors.grey[500] : primarySwatch[100];
primaryColorDark ??= isDark ? Colors.black : primarySwatch[700];
toggleableActiveColor ??= isDark ? Colors.tealAccent[200] : (accentColor ?? primarySwatch[600]);
accentColor ??= isDark ? Colors.tealAccent[200] : primarySwatch[500];
canvasColor ??= isDark ? Colors.grey[850] : Colors.grey[50];
scaffoldBackgroundColor ??= canvasColor;
bottomAppBarColor ??= isDark ? Colors.grey[800] : Colors.white;
cardColor ??= isDark ? Colors.grey[800] : Colors.white;
dividerColor ??= isDark ? const Color(0x1FFFFFFF) : const Color(0x1F000000);
```

The origin of this implementation was a desire to *just* configure the theme to match the original Material spec. As the importance of making custom themes grew, we began to move away from adding individual generic properties with default values, and towards component-specific "theme" valued properties, whose own properties were default null. The responsibility for computing default values for visual properties moved from the themes to the components.

Although the newest Material components and their themes conform to the new approach, there is considerable technical debt. Not all of the new component themes have been defined consistently. There was a certain amount of "evolution" in their design, as they were added over a considerable period of time. Thankfully the design has stabilized and in addition to moving most of the theme's remaining generic properties to component themes, we're going to "normalize" the existing ones.

Related issues: #22913, #9148.

Normalized Component Themes

A normalized component theme is one that applies to just one type of Material component. It represents a collection of visual parameters, all of which are null by default. The component's widgets configure their visuals in terms of widget parameters (also null by default), and then component theme properties, and then the overall theme's text theme and color scheme. The properties of the text theme and color scheme are guaranteed to have non-null values.

The ToggleButtons (implementation) widget is a good example of a component and its normalized theme. See ToggleButtonsTheme, an inherited widget that's configured with a ToggleButtonsThemeData, as well as its test.

Here's an outline of the pattern for a normalized theme in terms of a component called "Foo" that's represented by a widget of the same name.

- InheritedWidget FooTheme defines the Foo component theme. FooTheme's constructor has a FooThemeData parameter called data.
- FooThemeData defines the theme's visual attributes. The class extends Diagnosticable, it has a const constructor, and its properties are null by default.
- ThemeData has a FooThemeData property called fooTheme. Its value is const FooThemeData() by default and it cannot be null.
- FooTheme.of(context) returns the nearest FooTheme's data, otherwise it returns Theme.of(context).fooTheme.
- Foo widgets compute defaults at build time based on (in priority order): Foo properties, FooTheme.of(context), Theme.of(context).colorScheme and Theme.of(context).textTheme, internal defaults.

This Foo example is probably more helpful than the previous outline.

ThemeData Cleanup

We'd like to gradually revise the ThemeData class until its contents are essentially just a TextTheme, a ColorScheme, and a set of normalized component themes. Doing so would make it much easier to configure custom themes because that's all components (will) depend on. We also plan to move to the latest text style names in the text theme, so that developers will have an easier time with UI specs that have been written with the current terminology.

Another advantage of the work outlined here is that it will mesh nicely with Material Design tools, like the Material Theme Builder.

Related issues: #36624, #65782, #66060.

Rename the TextTheme TextStyles

The names of the 13 text styles in the TextTheme class are based on the original Material spec. In early 2018 a new set of names were introduced. It was particularly difficult to adopt the implied API changes because among the text styles with new names was ‘body1’, which had been renamed ‘body2’, and ‘body2’ which of course was now ‘body1’.

We took a run at making the changes in October 2018 (see #22330). In the end we decided to just document the mapping from the old names to the new ones.

We’ve decided to finally tackle the problem, see #45745.

Deprecate/remove the accent properties

The Material Design spec no longer includes a color called ‘accent’. The ‘secondary’ color scheme color is roughly equivalent. The overall theme’s accent properties are rarely used within Flutter itself. For example the accentIconTheme is only used by FloatingActionButton and in that case, it’s only used (in one expression) for the sake of backwards compatibility. The accentColorBrightness property is only used by GridTile. We plan to replace component dependencies on the accent properties with dependencies on component themes, and the overall theme’s color scheme and text theme.

```
Color accentColor;
Brightness accentColorBrightness;
TextTheme accentTextTheme;
IconThemeData accentIconTheme;
```

The process for removing these properties will take a considerable amount of time because they’ve been around since early versions of Flutter.

1. Remove dependencies - but not definitions - from Flutter.
2. When all of the accent properties dependencies have been removed, send a breaking change announcement.
3. Deprecate the properties and constructor parameters.
4. Remove the deprecated properties and constructor parameters.

A little bit of work has been done on step 1, see #46923.

This proposal has been mirrored on GitHub: #56918.

Compute brightness, primaryColorBrightness, accentColorBrightness

Currently ThemeData has two obviously conflicting brightness values, ThemeData.brightness and ThemeData.colorScheme.brightness. These values are redundant and they are not kept in sync.

ThemeData's constructor uses the brightness value to configure many of the grab-bag colors (see "Deprecate/move/remove the 21 grab-bag properties"). The brightness is also used to make the color of all the textTheme's text styles black for Brightness.light, white for Brightness.dark.

We are evolving the Theme's colorScheme as the primary source of default material components colors as well the overall theme's brightness.

This PR changes ThemeData.brightness to be a simple getter that returns colorScheme.brightness:#56956. Specifying a brightness value is still allowed; it's applied to the theme's colorScheme. Likewise for the brightness copyWith parameter.

The values of the other two brightness properties are computed if they're not specified.

```
primaryColorBrightness ??= estimateBrightnessForColor(primaryColor);
accentColorBrightness ??= estimateBrightnessForColor(accentColor);
```

Specifying these values requires the developer to understand how they'll be used (what components depend on them), which is unlikely. Specifying these values implies that there isn't a more direct way to configure the dependent components, which is also unlikely. It would be simpler if they were read-only, i.e. they were not ThemeData constructor parameters.

The following sections describe how primaryColorBrightness and accentColorBrightness are currently used and why they shouldn’t be.

AppBar

AppBar and AppBarTheme have a brightness parameter which can be used to override the theme's primaryColorBrightness value. The brightness value is only used once, by the build method, to decide if the system overlays (status bar, navigation bar) should be light or dark.

```
final Brightness brightness = widget.brightness
  ?? appBarTheme.brightness
  ?? theme.primaryColorBrightness;
final SystemUiOverlayStyle overlayStyle = brightness == Brightness.dark
  ? SystemUiOverlayStyle.light
  : SystemUiOverlayStyle.dark;
```

The AppBar API probably should have provided an explicit overlayStyle parameter. It’s unlikely a developer would intentionally create a ThemeData that specified a primaryColorBrightness that was inconsistent with the primaryColor, just to override the AppBar’s overlay style.

TextField

On iOS, the keyboard's appearance can be modified a little with the keyboardAppearance parameter, https://api.flutter.dev/flutter/material/TextField/keyboardAppearance.html which defaults to the value of primaryColorBrightness. Overriding the Theme's primaryColorBrightness isn't a sound way to get this effect since developers are unlikely to be aware of all of its implications.

SearchDelegate

Defines the appearance of the page shown by showSearchPage(). The SearchDelegate's appBarTheme method returns a ThemeData (rather than an AppBarTheme and see https://github.com/flutter/flutter/issues/45498). By default it returns a copy of the current theme with primaryColor set to white, and with primaryColorBrightness set to Brightness.light. This example demonstrates why it's useful for ThemeData.copyWith() to include the primaryColorBrightness property: copyWith doesn't update dependent properties, so if the primaryColor is changed, then primaryColorBrightness must be changed too. If primaryColorBrightness was always computed, this wouldn't be necessary.

GridTileBar

This class is very rarely used. Internally, it creates a Theme that overrides both accentColor and accentColorBrightness. As noted above, if the accent (secondary) color's brightness was computed, this would not be necessary.

```
final ThemeData darkTheme = ThemeData(
  brightness: Brightness.dark,
  accentColor: theme.accentColor,
  accentColorBrightness: theme.accentColorBrightness,
);
```

Deprecate/move/remove the 21 grab-bag properties

Many of the colors in this section have been around for years and most of them are only used by a handful of components. In most cases the colors should migrate to new or existing component themes.

A considerable amount of TBD analysis will be needed for each property. It's possible that some/all of these colors will never be completely removed or deprecated.

```
Color backgroundColor;
Color bottomAppBarColor;
Color canvasColor;
Color cardColor;
Color cursorColor;
Color dialogBackgroundColor;
Color disabledColor;
Color dividerColor;
Color errorColor;
Color highlightColor;
Color hintColor;
Color indicatorColor;
Color unselectedWidgetColor;
Color buttonColor;
Color secondaryHeaderColor;
Color textSelectionColor;
Color textSelectionHandleColor;
Color scaffoldBackgroundColor;
Color selectedRowColor;
Color splashColor;
Color toggleableActiveColor;
```

Related issue: #33257.

DETAILED DESIGN

Component Theme Normalization Project

This subsection provides some additional detail about how the theme API will change per component theme normalization. It is not a comprehensive list of every change, it's just a more detailed outline of the work. As the work proceeds, component by component, issues will be created that cover all of the details.

The table below lists all 28 of the Material components or component categories, as they appear in the current Material spec. Currently, the components correspond to about 90 Material library widgets, and about 20 component themes. The linked sections that follow summarize the work needed to normalize all of their component themes.

🟦	App bars: bottom	🟦	App bars: top	◻	Backdrop
🟩	Banners	🟩	Bottom Navigation	🟩	Buttons
🟦	Buttons: floating action	🟦	Cards	🟩	Chips
🟩	Data tables	🟦	Dialogs	🟩	Dividers
◻	Image Lists	🟦	Lists	■	Menus
🟦	Navigation Drawer	⬜	Pickers	🟩	Progress Indicators
⬜	Checkboxes	⬜	Radio buttons	⬜	Switches
🟦	Sheets: bottom	⬜	Sheets: side	🟩	Sliders
🟩	Snackbars	■	Tabs	■	Text fields
	Tooltips				

◻ Currently there's no Material library component.

⬜ There are no themes for the components in this category yet.

🟦 The theme support needs significant work to make it conformant.

🟦 The theme support needs a modest amount of work to make it conformant.

🟩 The theme support is conformant or very close.

Related issues: #49154, #27979

App bars: bottom

Widget: BottomAppBar.

The widget API docs don't mention BottomAppBarTheme.

BottomAppBarTheme properties are null by default (good), but there's no BottomAppBarThemeData.

ThemeData property is BottomAppBarTheme

Related issues: #48195.

App bars: top

Widgets: AppBar, SliverAppBar, FlexibleSpaceBar.

AppBarTheme properties are null by default (good), but there's no AppBarThemeData.

ThemeData property is AppBarTheme.

Related issues: #47166, #48195, #49430, #50606, #60792

Backdrop

Defined here: https://material.io/components/backdrop/

Although we've built backdrop components for various demo apps - see for example #15579 - currently the Material library doesn't include one and currently we don't have plans to add this widget to the Material library.

Banners

Widget: MaterialBanner

The widget API docs don't mention MaterialBannerTheme.

MaterialBannerTheme and MaterialThemeData are conformant.

Bottom navigation

Widget: BottomNavigationBar

BottomNavigationBar and BottomNavigationBarTheme are conformant. See #54714.

Buttons

A detailed proposal about updating the button widgets and their theme system can be found here: https://flutter.dev/go/material-button-system-updates and an additional GitHub on this same topic here: #54776.

The remainder of this section summarizes the limitations of the origin buttons API.

Most of the issues raised here have been addressed by a new set of button widgets, see #59702.

Widgets: RaisedButton, RawMaterialButton, FlatButton, MaterialButton, IconButton, BackButton, CloseButton, OutlineButton, ToggleButtons, InkResponse, InkWell.

One bright spot: ToggleButtons already has a normalized theme. In fact it can be considered the archetype for normalized component themes.

Related issues: #16488, #19623, #22745, #26180, #27461, #27979, #29556, #29728, #30047, #32020, #37587, #38335, #38602, #38646, #38655, #41627, #44047, #43358, #45849, #46715, #48299, #48299, #51144, #62882.

Updating ButtonTheme

Currently FlatButton, RaisedButton, and OutlineButton are all styled with ButtonTheme. In addition to not conforming to the patterns described in the "Component Theme Normalization" section, there are several problems with the way ButtonTheme currently works.

ButtonTheme's get methods compute the default values for the visual properties of all three types rather than deferring to the components. In some cases they use the type of the get method's MaterialButton parameter to compute the property. This makes it quite difficult to explain the theme, or for apps to configure it.

```
Color getFillColor(MaterialButton button) {
  final Color fillColor = button.enabled ? button.color : button.disabledColor;
  if (fillColor != null)
    return fillColor;

  if (button is FlatButton || button is OutlineButton || button.runtimeType == MaterialButton)
    return null;

  if (button.enabled && button is RaisedButton && _buttonColor != null)
    return _buttonColor;
  switch (getTextTheme(button)) {
    case ButtonTextTheme.normal:
    case ButtonTextTheme.accent:
      return button.enabled ? colorScheme.primary : getDisabledFillColor(button);
    case ButtonTextTheme.primary:
      return button.enabled
        ? _buttonColor ?? colorScheme.primary
        : colorScheme.onSurface.withOpacity(0.12);
  }
}
```

Similarly, it's quite difficult to configure ButtonTheme to *just* change the appearance of one type of button. Like the textColor for FlatButtons or the pressed elevation for RaisedButtons.

ButtonThemes include a ButtonTextTheme (a misnomer) valued property, which according to the enum's documentation affects the button's text color.

```
enum ButtonTextTheme {
  normal, /// textColor is black or white depending on [ThemeData.brightness].
  accent, /// textColor is the theme's accent color
  primary, /// textColor is the theme's primary color
}
```

Sadly, it also changes the button's default padding, shape, fill color, splash color, and highlight color. In many of these cases, the way the default is computed also depends on the type of the ButtonTheme getter's button parameter.

All three button classes just configure a RawMaterialButton. A simpler, conforming, approach to defining the ButtonTheme would be to define three themes: FlatButtonTheme, RaisedButtonTheme, OutlineButtonTheme. These themes would - by and large - have the same null-by-default properties:

```
Color color
Color textColor
TextStyle textStyle
Color highlightColor
Color splashColor
double elevation
EdgeInsetsGeometry padding
BoxConstraints constraints
ShapeBorder shape
Duration animationDuration
```

The color, textColor, and elevation properties could be defined with MaterialStates to enable specifying different values for disabled, focused, hovered, pressed.

OutlineButtonTheme would get an additional borderSide property.

Buttons: floating action

Widget: FloatingActionButton.

FloatingActionButtonThemeData is conformant but there's no FloatingActionButtonTheme class. The ThemeData property is a FloatingActionButtonThemeData (good).

Related issues: #28138, #53502.

Cards

Widget: Card.

CardTheme properties are null by default (good), but there's no CardThemeData.

ThemeData property is CardTheme.

Chips

Widgets: Chip, RawChip, InputChip, ChoiceChip, FilterChip, ActionChip.

ChipTheme and ChipThemeData are conformant however the default ChipTheme constructed by ThemeData includes non-null default values.

Data tables

Widgets: DataTable, PaginatedDataTable, DataColumn, DataRow, DataCell, TableRowInkWell, GridTileBar.

DataTableTheme was added in #64316

Dialogs

Widgets: AboutDialog, Dialog, AlertDialog, SimpleDialogOption, SimpleDialog, ButtonBar.

DialogTheme should be called DialogThemeData.

Dividers

Widgets: Divider, VerticalDivider.

The widget API class doc doesn't mention DividerTheme.

DividerTheme and DividerThemeData are conformant.

Image lists

There is no Flutter component yet, see https://material.io/components/image-lists/.

Lists

Widgets: RadioListTile<T>, SwitchListTile, AboutListTile, ExpansionTile, CheckboxListTile, GridTile, ListTile, Scrollbar, ReorderableListView.

ListTileTheme is conformant, but there's no ListTileThemeData and no ThemeData propety. It also serves double duty as the theme for Drawer (by setting style to ListTileStyle.list) which is an odd one-off.

Related issues: #42621, #17635 (comment)

Menus

Widgets: DropdownButton<T>, DropdownButtonFormField<T>, PopupMenuButton<T>, DropdownMenuItem<T>, PopupMenuDivider, PopupMenuItem<T>, CheckedPopupMenuItem<T>.

PopupMenuTheme and PopupMenuThemeData are conformant.

There is no theme for dropdown menus.

Navigation drawer

Widgets: Drawer, UserAccountsDrawerHeader, DrawerHeader.

Currently the drawer's contents are wrapped with a ListTileTheme whose style is ListTileStyle.drawer. It would probably be better to just have a DrawerTheme.

Pickers

Widgets: DayPicker, MonthPicker, YearPicker.

Currently there are no themes for the date and time pickers.

Progress indicators

Widgets: LinearProgressIndicator, CircularProgressIndicator, RefreshProgressIndicator, RefreshIndicator.

ProgressIndicatorTheme was added in #81075.

Selection controls: Checkboxes

Widget: Checkbox.

There is no checkbox theme. Currently the widgets are configured in terms of the overall theme's toggleableActiveColor, unselectedWidgetColor, disabledColor, focusColor, and hoverColor.

Related issues: #53420.

Selection controls: Radio buttons

Widget: RadioButton<T>.

There is no theme for RadioButtons. Currently they're defined in terms of the overall theme's unselectedWidgetColor, disabledColor, toggleableActiveColor, focusColor, hoverColor.

Selection controls: Switches

Widget: Switch.

There is no theme for Switch. Currently some of the switch colors are defined in terms of the overall theme's toggleableActiveColor, hoverColor, focusColor.

Sheets: bottom

Widget: BottomSheet.

BottomSheetThemeData is conformant however there's no BottomSheetTheme.

Sheets: side

Side sheets are just a feature of Scaffold, see https://api.flutter.dev/flutter/material/Scaffold/endDrawer.html

Sliders

Widgets: RangeSlider, Slider.

SliderTheme and SliderThemeData are conformant.

Snackbars

Widgets: SnackBarAction, SnackBar.

SnackBarTheme and SnackBarThemeData are conformant.

There's an open RFE, see #46599.

Tabs

Widgets: Tab, TabBar, TabBarView, TabPageSelectorIndicator, TabPageSelector.

TabBarTheme should be called TabBarThemeData, etc.

Related issues: #69162.

Text fields

Widgets: TextField, InputDecorator, SelectableText, TextFormField, TextSelectionToolbar.

Related issues: #21385, #33183, #31169, #28228, #41382, #48287, #48882

Update InputDecorationTheme, InputDecorator

InputDecorationTheme should be called InputDecorationThemeData, etc, per "Normalized Component Themes" above.

InputDecorator uses ThemeData primaryColor, accentColor, disabledColor, errorColor, hintColor, iconTheme.color.

Current values for disabledColor are defined by InputDecorator.

```
// dark theme: 10% white (enabled), 5% white (disabled)
// light theme: 4% black (enabled), 2% black (disabled)
const Color darkEnabled = Color(0x1AFFFFFF);
const Color darkDisabled = Color(0x0DFFFFFF);
const Color lightEnabled = Color(0x0A000000);
const Color lightDisabled = Color(0x05000000);
```

Current values for errorColor and hintColor are defined by ThemeData.

```
errorColor ??= Colors.red[700];
hintColor ??= isDark ? const Color(0x80FFFFFF) : const Color(0x8A000000);
```

The default input color for input decorator icons is defined like this:

```
switch (themeData.brightness) {
  case Brightness.dark:
    return Colors.white70;
  case Brightness.light:
    return Colors.black45;
  default:
    return themeData.iconTheme.color;
}
```

These dependencies should be updated:

primaryColor => colorScheme.primary
accentColor => colorScheme.secondary
disabledColor => colorScheme.onSurface.withOpacity(0.38)
errorColor => colorScheme.error
iconColor => colorScheme.onSurface.withOpacity(0.80)

Add TextSelectionTheme

Please see flutter.dev/go/text-selection-theme, the design doc that supersedes the material in this section. The remainder of this section should be viewed as historical background.

The default appearance of the text selection (handles, cursor, highlight) is intended to match the platform. The material spec does not cover text selection, however both Android and iOS specs, as well as Angular Material (though not listed in spec) match the border color by defaulting to the primary color. The handle color on mobile is seen to default to the cursor color. In addition, also on mobile, the highlight color seems to generally be based on primary with an opacity of 20-45% depending on the color, though there are many exceptions.

The Material text selection toolbar, text selection handles and menu are defined by materialTextSelectionControls (in text_selection.dart).

The toolbar is just a row of FlatButtons, which inherit the current button theme. A TextSelectionTheme class would enable configuring the appearance of FlatButtons in the toolbar.

Currently, materialTextSelectionControls depends on ThemeData.textSelectionHandleColor:

```
textSelectionHandleColor ??= isDark ? Colors.tealAccent[400] : primarySwatch[300];
```

The Material SelectableText widget defines the appearance of selected text. Currently it depends on 2 ThemeData properties:

```
textSelectionColor ??= isDark ? accentColor : primarySwatch[200];
cursorColor = cursorColor ?? const Color.fromRGBO(66, 133, 244, 1.0);
```

The cursor's width radius can not be configured with ThemeData, they default to 2 and const Radius.circular(2) respectively.

These classes should defer to a new TextSelectionTheme.

Related issues: #42122, #39387, #37550, #46238, #61227.
Related proposal: Change default textSelectionHandleColor to accentColor

Update TextField

TextField depends on ThemeData errorColor, cursorColor, and textSelectionColor. These dependencies should be updated as outlined in the previous sections.

The new text field theme, TextInputTheme, is #61008.

Related issues: #44100.

Tooltips

Widget: Tooltip.

TooltipTheme and TooltipThemeData are conformant.

Other Material Components

The Material library contains several components that are no longer defined by the Material spec. Most of them can still be found here: https://material.io/archive/guidelines/components/.

There are no themes for these components.

Stepper

Widgets: Stepper, Step.

Expansion panel

Widgets: ExpansionPanel, ExpansionPanelRadio, ExpansionPanelList.

Mergeable Material

Widgets: MergeableMaterial, MaterialSlice, MaterialGap.

SearchBar

TBD and highly requested, see #9784, #17119.