Dark Theme

The Material dark theme system can be used to create a beautiful and functional dark theme for your app, which generally consists of dark background colors and light foreground colors for elements such as text and iconography.

Some of the most common benefits of a dark theme include conserving battery power for devices with OLED screens, reducing eye strain, and facilitating use in low-light environments.

Starting with Android Q, users are now able to switch their device into dark theme via a new system setting, which applies to both the Android system UI and apps running on the device.

Design & API Documentation

- Material Design guidelines: Dark Theme
- Android Q Dark Theme Documentation
- AppCompat DayNight Documentation

Setup

In order to support the dark theme functionality in Android Q, make sure you are depending on the latest `1.1.0` alpha version of the Material Android library, and update your app theme to inherit from `Theme.MaterialComponents.DayNight` (or one of its descendants). E.g.:

res/values/themes.xml

```
<style name="Theme.MyApp" parent="Theme.MaterialComponents.DayNight">
    <!-- ... -->
</style>
```

Alternatively, if you want to define separate `Light` and `Dark` themes for your app, you can inherit from `Theme.MaterialComponents.Light` in the `values` directory, and `Theme.MaterialComponents` in the `values-night` directory. E.g.:

res/values/themes.xml

```
<style name="Theme.MyApp" parent="Theme.MaterialComponents.Light">
    <!-- ... -->
</style>
```

res/values-night/themes.xml

```
<style name="Theme.MyApp" parent="Theme.MaterialComponents">
    <!-- ... -->
</style>
```

The `Theme.MaterialComponents` theme is a static dark theme, whereas `Theme.MaterialComponents.DayNight` is a more dynamic theme which will help facilitate easy switching between your app's `Light` and `Dark` theme. If using a `DayNight` theme, you can define one app theme that references color resources, which can be overridden in the `values-night` directory if needed.

Catalog

To see how Material components adapt in a dark theme, build and run the Catalog app and enable dark theme in one of the following ways:

- Any API Level: Overflow menu on Catalog home screen
- Android Q: `Settings > Display > Dark Theme` (or Dark Theme tile in Notification Tray)
- Android P: `Settings > System > Developer options > Night mode`

Color Palette

At the core of any dark theme is a color palette that uses dark background colors and light foreground colors. The Material `Dark` themes make use of the Material Color System, in order to provide default dark theme values for neutral palette colors such as `colorBackground` and `colorSurface`.

The baseline Material `Dark` theme background and surface colors are dark gray instead of black, which increases visibility for shadows and also reduces eye strain for light text.

The Material `Dark` themes also provide adjusted defaults for the baseline branded palette, including `colorPrimary` and `colorSecondary`. See the Material Dark Theme spec for guidance on how you can adjust your brand colors for dark theme.

Primary vs. Surface Coloring for Large Surfaces

According to the Material Dark Theme spec, large surfaces should not use bright background colors because they can emit too much brightness. When applicable, the Material themes will provide this behavior out-of-the-box, e.g., by having the `Light` theme default to `Widget.MaterialComponents.AppBarLayout.Primary` and the `Dark` theme default to `Widget.MaterialComponents.AppBarLayout.Surface`.

However, there may be some cases where you need to apply this primary vs. surface color swap to some custom UI. For convenience, the Material themes provide a `colorPrimarySurface` attribute, that points to `colorPrimary` in the `Light` theme and `colorSurface` in the `Dark` theme. There is also a corresponding `colorOnPrimarySurface` attribute that can be used for foreground elements such as text and iconography on top of a `colorPrimarySurface` background.

Additionally, the Material Android library provides `PrimarySurface` styles for components that act as large surfaces and commonly use `colorPrimary` for their background in light theme. These styles will automatically switch between the component's primary colored style in light theme and surface colored style in dark theme. E.g.:

- `Widget.MaterialComponents.BottomAppBar.PrimarySurface`
- `Widget.MaterialComponents.BottomNavigationView.PrimarySurface`
- `Widget.MaterialComponents.TabLayout.PrimarySurface`
- `Widget.MaterialComponents.Toolbar.PrimarySurface`

Elevation Overlays

In addition to the color palette adjustments mentioned above, communicating the hierarchy of a UI via elevation requires some dark theme specific considerations.

Shadows are less effective in an app using a dark theme, because they will have less contrast with the dark background colors and will appear to be less visible. In order to compensate for this, Material surfaces in a dark theme become lighter at higher elevations, when they are closer to the implied light source.

This is accomplished via elevation overlays, which are semi-transparent white (`colorOnSurface`) overlays that are conceptually placed on top of the surface color. The semi-transparent alpha percentage is calculated using an equation based on elevation, which results in higher alpha percentages at higher elevations, and therefore lighter surfaces.

Note: we avoid overdraw with the elevation overlays by calculating a composite blend of the surface color with the overlay color and using that as the surface's background, instead of drawing another layer to the canvas.

Affected Components

The following is a list of Material components that support elevation overlays in dark theme, because they use `colorSurface` and can be elevated:

- Top App Bar
- Bottom App Bar
- Bottom Navigation
- Tabs
- Card
- Dialog
- Menu
- Bottom Sheet
- Navigation Drawer
- Switch

Theme Attributes

In order to facilitate some orchestration around the elevation overlays, we have the following theme attributes:

Attribute Name	Description	Default Value
`elevationOverlayEnabled`	Whether the elevation overlay functionality is enabled.	`false` in `Light` themes, `true` in `Da`
`elevationOverlayColor`	The color used for the elevation overlays, applied at an alpha based on elevation.	`colorOnSurface`

Note: If inheriting from the `Theme.MaterialComponents` theme or a descendant, you most likely do not have to set these attributes yourself because the Material themes already set up the above defaults.

Custom Views & Non-Material Components

If you would like to apply dark theme elevation overlays to your custom views or any non-Material views that are elevated surfaces, then you can use the `MaterialShapeDrawable` or `ElevationOverlayProvider` APIs.

MaterialShapeDrawable

The key to supporting elevation overlays in a custom view is creating a `MaterialShapeDrawable` with the overlay support enabled via `MaterialShapeDrawable#createWithElevationOverlay`, and setting it as the background of your view.

Next, override the `View#setElevation` method and forward the elevation passed in to your `MaterialShapeDrawable` background's `setElevation` method.

`MaterialShapeDrawable` is the preferred approach for custom views because it will keep track of the elevation value for you and factor that in to the overlay any time elevation changes, and you don't have to worry about incorrectly compounding the overlays multiple times.

ElevationOverlayProvider

If you have a case where the elevation value is more static and you would like to get the corresponding dark theme overlay color (perhaps to color an existing view), then you can use `ElevationOverlayProvider`.

If elevation overlays are enabled at the theme level, the `ElevationOverlayProvider#compositeOverlayWithThemeSurfaceColorIfNeeded` method will return `colorSurface` with the overlay color blended in at an alpha level based on the elevation passed in. Otherwise, it will simply return `colorSurface`, so that you can use the result of this method in both `Light` and `Dark` themes without needing any additional orchestration.

If you need to blend the overlays with an arbitrary color or an adjusted surface color, or get access to lower level values such as the overlay alpha percentages, take a look at the other `ElevationOverlayProvider` methods including `compositeOverlayIfNeeded`, `compositeOverlay`, and `calculateOverlayAlpha`.

Absolute Elevation

When calculating the elevation overlay alpha percentage, Material components factor in the absolute elevation of their parent view. This is because the distance from the light source is the driving factor behind elevation overlays.

If you need to factor in absolute elevation in a custom view that supports overlays, then you can use the `MaterialShapeUtils#setParentAbsoluteElevation` methods when using a `MaterialShapeDrawable` background. For example:

```
@Override
protected void onAttachedToWindow() {
  super.onAttachedToWindow();

  MaterialShapeUtils.setParentAbsoluteElevation(this);
}
```

Alternatively, you could use the `ElevationOverlayProvider` composite methods that take in a `View` parameter or the `getParentAbsoluteElevation` method.

Note: This means that you should consider accessibility contrast ratios for text and iconography, when deeply nesting elevated Material components and views that support elevation overlays.