Building for omni-channel

UDS enables the creation of multi-brand or multi-platform applications and user interfaces. The principal tool for doing this is the UDS base component library built with React Native. UDS base components support a richly customisable theming mechanism. Themability enables the same components to be reused across brands, React Native enables those components to be reused across platforms.

This guide covers using the UDS base components to build a multi-brand multi-platform application, by which we mean an application that shares UI code and logic across multiple brands and platforms.

> INFO
>
> Even though the UDS base components are built with React Native, **the components can be consumed by React "web" applications with no additional setup required**.

Before jumping into this technical guide, we recommend that you first read unlocking omni-channel which is a high level introduction to this topic.

Caveats

It's important to understand the following caveats when building in a multi-platform multi-brand context:

1. **Only UDS base components** When building multi-platform, you will only have access to the components listed in the UDS base component library. Additional components listed in a brand's design system will not be available to you, because they are built for the web using React.

2. **No support for Community Themes** One of the most powerful aspects of the UDS theming system is the ability to define visual "variants" of a component using the `variant` prop. However **there is no requirement that different brand community themes define the same variants**. This means that if you are using multiple themes from different brands, please stick with the **core brand packages** . The variants defined inside

community packages will not necessarily align, e.g. TELUS may have a `high` button variant, whereas Koodo may have a `primary` button variant.

```
import { Button, useTheme } from '@telus-uds/components-base'
/* it is simpler to use the default button */
() => <Button>Click me</Button>

/* ...or you can apply a variant after checking the current theme
*/
const theme = useTheme()
const variant = theme.metadata.name === 'allium' ? { priority:
'high' } : { primary: true }
() => <Button>Click me</Button>
```

> ⚠ CAUTION
>
> Invalid variants are silently ignored.

Installing

UDS is distributed as a series of npm packages, the relevant packages are:

1. `@telus-uds/components-base` containing the set of themable React Native base components
2. `@telus-uds/theme-<brand>` containing a brand theme
3. `@telus-uds/palette-<brand>` containing a brand palette

In addition to the base components, you will need to install a theme for each brand you will support, plus the brand palette for each brand in order to use that brand's fonts and icons. See the UDS fonts documentation for including font assets in your build. For icons you can simply import the icons you want from the relevant brand palette, *e.g.*

```
import { Users } from '@telus-uds/palette-allium/build/web/icons'
// The export is a React (Native) component which takes color and
size props
<Users color={...} size={...} />
```

> INFO
>
> Icon sets are brand-specific, you will need to handle loading the appropriate icons from the currently active brand in your application logic.

Usage

Once you have installed these packages you will be able to consume UDS base components to build React or React Native applications. Entry points are automatically configured in UDS Base for web versus React Native applications so no additional setup is required.

All UDS base components need to be wrapped in a `BaseProvider`. This component handles providing a theme context - you must pass in a default theme. We recommend placing the provider once at the root of your app.

```
import alliumTheme from '@telus-uds/theme-allium'
import koodoTheme from '@telus-uds/theme-koodo'
import { Typography, BaseProvider } from '@telus-uds/components-base'

const theme = /* custom app logic to decide on current theme */

const App = () => (
  <BaseProvider defaultTheme={theme}>
    <Typography>Hello world!</p>
  </BaseProvider>
)
```

Theme switching

Dynamic theme switching can be achieved with the `useTheme` and `useSetTheme` hooks.

```
import { Button, useTheme, useSetTheme } from '@telus-uds/components-base'

const themes = [alliumTheme, koodoTheme, publicTheme]
const Toggle = () => {
  const theme = useTheme()
```

```
  const { setTheme } = useSetTheme()
  const themeIndex = themes.findIndex(t => t === theme)
  const nextTheme = themes[(themeIndex+1) % themes.length]

  return <Button onPress={() => setTheme(nextTheme)}>next
theme</Button>}
}
```

Using a palette standalone

If you wish to consume the design tokens from any brand, you can access them from the brand palette. Here is an example with the TELUS palette `palette-allium`.

```
import palette from '@telus-uds/palette-allium'
const PurpleText = () => <p style={{ color:
palette.color.purpleTelus }}>Purple text</p>
```

Where to get help

The best place to get help is in the UDS community slack channel. Here you can reach out not just to the UDS team, but also to the whole community, to troubleshoot any issues you're having, or make suggestions to help make things better.

If you find a bug while using UDS, you can raise an issue for it.

Edit this page