StyleSheet

A StyleSheet is an abstraction similar to CSS StyleSheets

StyleSheet **Expo**

```
import React from 'react';
import {StyleSheet, Text, View} from 'react-native';

const App = () => (
  <View style={styles.container}>
    <Text style={styles.title}>React Native</Text>
  </View>
);

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 24,
    backgroundColor: '#eaeaea',
  },
  title: {
    marginTop: 16,
    paddingVertical: 8,
    borderWidth: 4,
    borderColor: '#20232a',
    borderRadius: 6,
    backgroundColor: '#61dafb',
    color: '#20232a',
    textAlign: 'center',
    fontSize: 30,
    fontWeight: 'bold',
```

Preview My Device iOS Android Web

Code quality tips:

- By moving styles away from the render function, you're making the code easier to understand.
- Naming the styles is a good way to add meaning to the low level components in the render function.

Reference

Methods

`compose()`

```
static compose(style1: Object, style2: Object): Object | Object[];
```

Combines two styles such that `style2` will override any styles in `style1`. If either style is falsy, the other one is returned without allocating an array, saving allocations and maintaining reference equality for PureComponent checks.

Compose

Expo

```
import React from 'react';
import {StyleSheet, Text, View} from 'react-native';

const App = () => (
  <View style={container}>
    <Text style={text}>React Native</Text>
  </View>
);

const page = StyleSheet.create({
  container: {
    flex: 1,
    padding: 24,
    backgroundColor: '#fff',
  },
  text: {
    fontSize: 30,
    color: '#000',
  },
});

const lists = StyleSheet.create({
  listContainer: {
    flex: 1,
    backgroundColor: '#61dafb',
  },
```

Preview My Device iOS Android Web

`create()`

```
static create(styles: Object): Object;
```

Creates a StyleSheet style reference from the given object.

`flatten()`

```
static flatten(style: Object[]): Object;
```

Flattens an array of style objects, into one aggregated style object. Alternatively, this method can be used to lookup IDs, returned by `StyleSheet.register`.

> **NOTE:** Exercise caution as abusing this can tax you in terms of optimizations. IDs enable optimizations through the bridge and memory in general. Referring to style objects directly will deprive you of these optimizations.

Flatten

Expo

```
import React from 'react';
import {StyleSheet, Text, View} from 'react-native';

const App = () => (
  <View style={page.container}>
    <Text style={flattenStyle}>React Native</Text>
    <Text>Flatten Style</Text>
    <Text style={page.code}>{JSON.stringify(flattenStyle,
null, 2)}</Text>
  </View>
);

const page = StyleSheet.create({
  container: {
    flex: 1,
    padding: 24,
    alignItems: 'center',
  },
  text: {
    color: '#000',
    fontSize: 14,
    fontWeight: 'bold',
  },
  code: {
    marginTop: 12,
    padding: 12,
```

Preview My Device iOS Android Web

This method internally uses `StyleSheetRegistry.getStyleByID(style)` to resolve style objects represented by IDs. Thus, an array of style objects (instances of `StyleSheet.create()`), are individually resolved to, their respective objects, merged as one and then returned. This also explains the alternative use.

`setStyleAttributePreprocessor()`

> **WARNING: EXPERIMENTAL.** Breaking changes will probably happen a lot and will not be reliably announced. The whole thing might be deleted, who knows? Use at your own risk.

```
static setStyleAttributePreprocessor(
  property: string,
  process: (propValue: any) => any,
);
```

Sets a function to use to pre-process a style property value. This is used internally to process color and transform values. You should not use this unless you really know what you are doing and have exhausted other options.

Properties

`absoluteFill`

A very common pattern is to create overlays with position absolute and zero positioning (`position: 'absolute', left: 0, right: 0, top: 0, bottom: 0`), so `absoluteFill` can be used for convenience and to reduce duplication of these repeated styles. If you want, absoluteFill can be used to create a customized entry in a StyleSheet, e.g.:

absoluteFill Expo

```
import React from 'react';
import {StyleSheet, Text, View} from 'react-native';

const App = () => (
  <View style={styles.container}>
    <View style={styles.box1}>
      <Text style={styles.text}>1</Text>
    </View>
    <View style={[styles.box2, StyleSheet.absoluteFill]}>
      <Text style={styles.text}>2</Text>
    </View>
    <View style={styles.box3}>
      <Text style={styles.text}>3</Text>
    </View>
  </View>
);

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  box1: {
    position: 'absolute',
    top: 40,
    left: 40,
    width: 100,
```

Preview My Device iOS Android Web

`absoluteFillObject`

Sometimes you may want `absoluteFill` but with a couple tweaks - `absoluteFillObject` can be used to create a customized entry in a `StyleSheet`, e.g.:

`hairlineWidth`

This is defined as the width of a thin line on the platform. It can be used as the thickness of a border or division between two elements. Example:

This constant will always be a round number of pixels (so a line defined by it can look crisp) and will try to match the standard width of a thin line on the underlying platform. However, you should not rely on it being a constant size, because on different platforms and screen densities its value may be calculated differently.

A line with hairline width may not be visible if your simulator is downscaled.

Is this page useful?

Edit this page

Last updated on ***Jun 21, 2023***