Platform-Specific Code

When building a cross-platform app, you'll want to re-use as much code as possible. Scenarios may arise where it makes sense for the code to be different, for example you may want to implement separate visual components for Android and iOS.

React Native provides two ways to organize your code and separate it by platform:

- Using the `Platform` module.
- Using platform-specific file extensions.

Certain components may have properties that work on one platform only. All of these props are annotated with `@platform` and have a small badge next to them on the website.

Platform module

React Native provides a module that detects the platform in which the app is running. You can use the detection logic to implement platform-specific code. Use this option when only small parts of a component are platform-specific.

```
import {Platform, StyleSheet} from 'react-native';

const styles = StyleSheet.create({
  height: Platform.OS === 'ios' ? 200 : 100,
});
```

`Platform.OS` will be `ios` when running on iOS and `android` when running on Android.

There is also a `Platform.select` method available, that given an object where keys can be one of `'ios' | 'android' | 'native' | 'default'`, returns the most fitting value for the platform you are currently running on. That is, if you're running on a phone, `ios` and `android` keys will take preference. If those are not specified, `native` key will be used and then the `default` key.

```
import {Platform, StyleSheet} from 'react-native';

const styles = StyleSheet.create({
  container: {
    flex: 1,
    ...Platform.select({
      ios: {
        backgroundColor: 'red',
      },
      android: {
        backgroundColor: 'green',
      },
      default: {
        // other platforms, web for example
        backgroundColor: 'blue',
      },
    }),
  },
});
```

This will result in a container having `flex: 1` on all platforms, a red background color on iOS, a green background color on Android, and a blue background color on other platforms.

Since it accepts `any` value, you can also use it to return platform-specific components, like below:

```
const Component = Platform.select({
  ios: () => require('ComponentIOS'),
  android: () => require('ComponentAndroid'),
})();

<Component />;
```

```
const Component = Platform.select({
  native: () => require('ComponentForNative'),
  default: () => require('ComponentForWeb'),
})();

<Component />;
```

Detecting the Android version

On Android, the `Platform` module can also be used to detect the version of the Android Platform in which the app is running:

```
import {Platform} from 'react-native';

if (Platform.Version === 25) {
  console.log('Running on Nougat!');
}
```

Note: `Version` is set to the Android API version not the Android OS version. To find a mapping please refer to Android Version History.

Detecting the iOS version

On iOS, the `Version` is a result of `-[UIDevice systemVersion]`, which is a string with the current version of the operating system. An example of the system version is "10.3". For example, to detect the major version number on iOS:

```
import {Platform} from 'react-native';

const majorVersionIOS = parseInt(Platform.Version, 10);
if (majorVersionIOS <= 9) {
  console.log('Work around a change in behavior');
}
```

Platform-specific extensions

When your platform-specific code is more complex, you should consider splitting the code out into separate files. React Native will detect when a file has a `.ios.` or `.android.` extension and load the relevant platform file when required from other components.

For example, say you have the following files in your project:

```
BigButton.ios.js
BigButton.android.js
```

You can then import the component as follows:

```
import BigButton from './BigButton';
```

React Native will automatically pick up the right file based on the running platform.

Native-specific extensions (i.e. sharing code with NodeJS and Web)

You can also use the `.native.js` extension when a module needs to be shared between NodeJS/Web and React Native but it has no Android/iOS differences. This is especially useful for projects that have common code shared among React Native and ReactJS.

For example, say you have the following files in your project:

```
Container.js # picked up by webpack, Rollup or any other Web bundler
Container.native.js # picked up by the React Native bundler for both Android and iOS
(Metro)
```

You can still import it without the `.native` extension, as follows:

```
import Container from './Container';
```

Pro tip: Configure your Web bundler to ignore `.native.js` extensions in order to avoid having unused code in your production bundle, thus reducing the final bundle size.

Is this page useful?

Edit this page

Last updated on ***Aug 24, 2023***