Cross Platform Implementation

CAUTION

This document refers to the architecture of the new renderer, Fabric, that is in active roll-out.

The React Native renderer utilizes a core render implementation to be shared across platforms

In the previous render system of React Native, the **React Shadow Tree**, layout logic, and **View Flattening** algorithm were implemented once for each platform. The current renderer was designed to be a cross-platform solution by sharing a core C++ implementation.

The React Native team intends to incorporate an animation system into the render system and also extend the React Native render system to new platforms such as Windows, and operating systems in game consoles, televisions, and more.

Leveraging C++ for the core render system introduces several advantages. A single implementation reduces the cost of development and maintenance. It improves the performance of creating React Shadow Trees and layout calculation because the overhead of integrating Yoga with the renderer is minimized on Android (i.e. no more JNI for Yoga). Finally, the memory footprint of each React Shadow Node is smaller in C++ than it would be if allocated from Kotlin or Swift.

The team is also leveraging C++ features that enforce immutability to ensure there are no issues related to concurrent access to shared but not protected resources.

It is important to recognize that the renderer use case for Android still incurs the cost of JNI for two primary use cases:

- Layout calculation of complex views (e.g. `Text`, `TextInput`, etc.) requires sending props over JNI.

- The mount phase requires sending mutation operations over JNI.

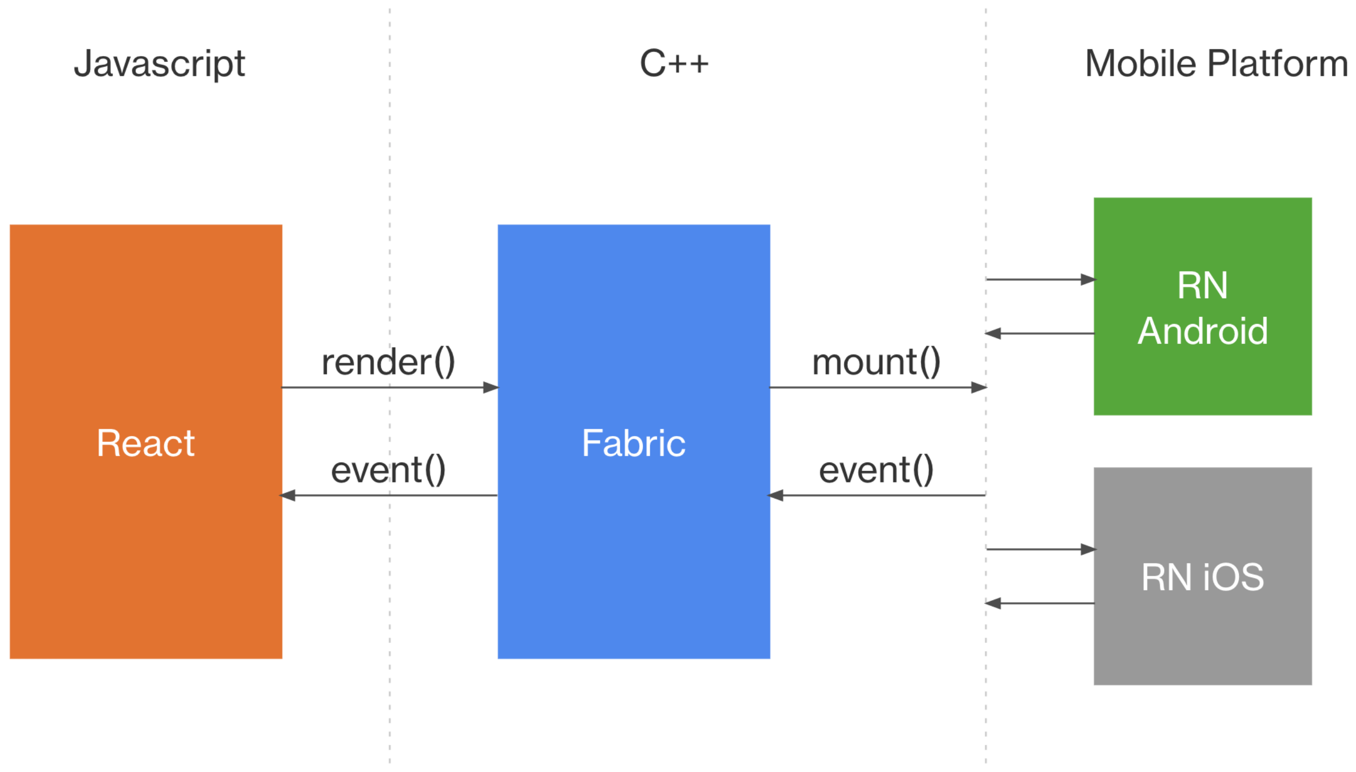
The team is exploring replacing `ReadableMap` with a new mechanism to serialize data using `ByteBuffer` to reduce overhead of JNI. Our goal is to reduce overhead of JNI by 35–50%.

The renderer provides two sides of its C++ APIs:

- **(i)** to communicate with React
- **(ii)** to communicate with the host platform

For **(i)**, React communicates with the renderer to **render** a React Tree and to "listen" for **events** (e.g. `onLayout`, `onKeyPress`, touch, etc).

For **(ii)**, the React Native renderer communicates with the host platform to mount host views on the screen (create, insert, update or delete of host views) and it listens for **events** that are generated by the user on the host platform.



Is this page useful? 👍 👎

Edit this page

*Last updated on **Mar 10, 2022***