Marshaling Data

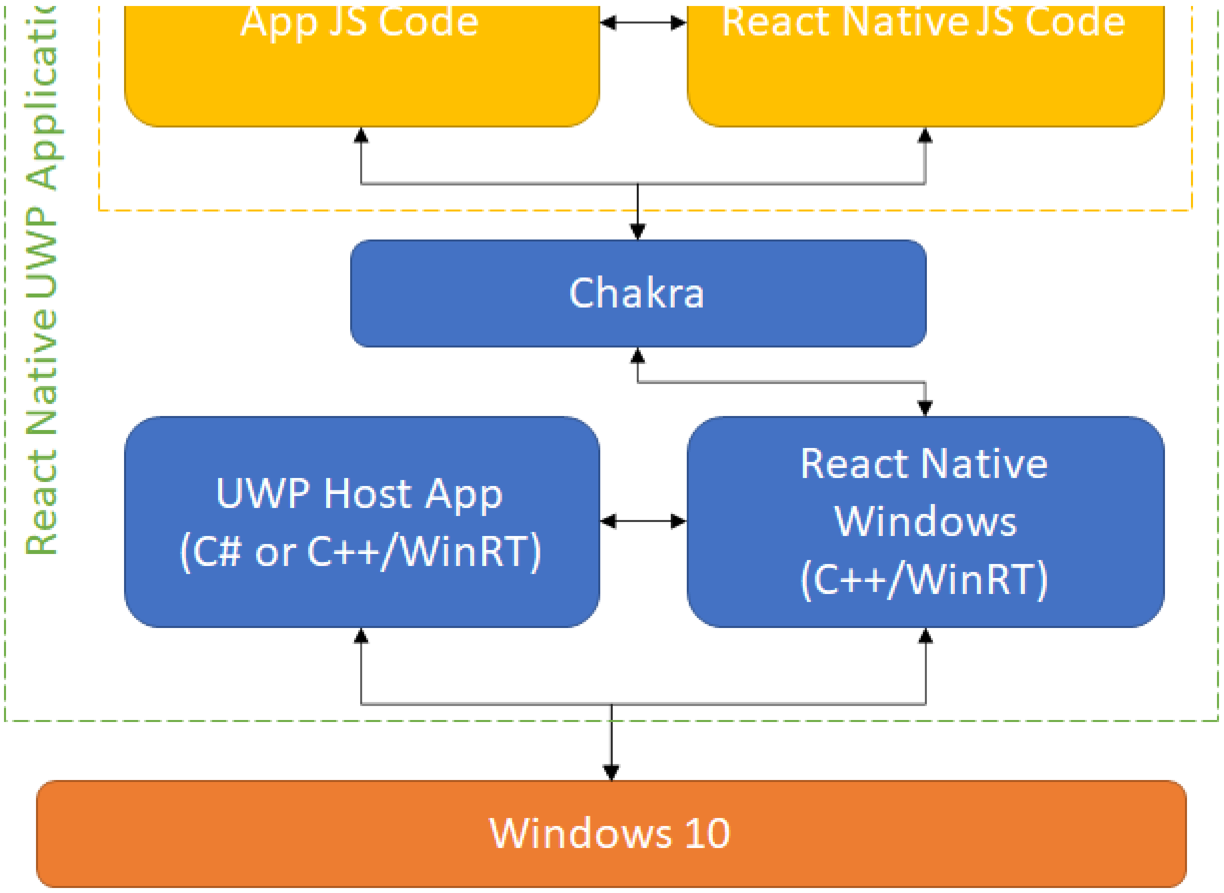
Edit

This documentation and the underlying platform code is a work in progress.

Overview

React Native applications are composed of multiple components and layers, some of which have boundaries that require the marshaling of data. This document intends to cover how this data marshaling occurs in React Native Windows.

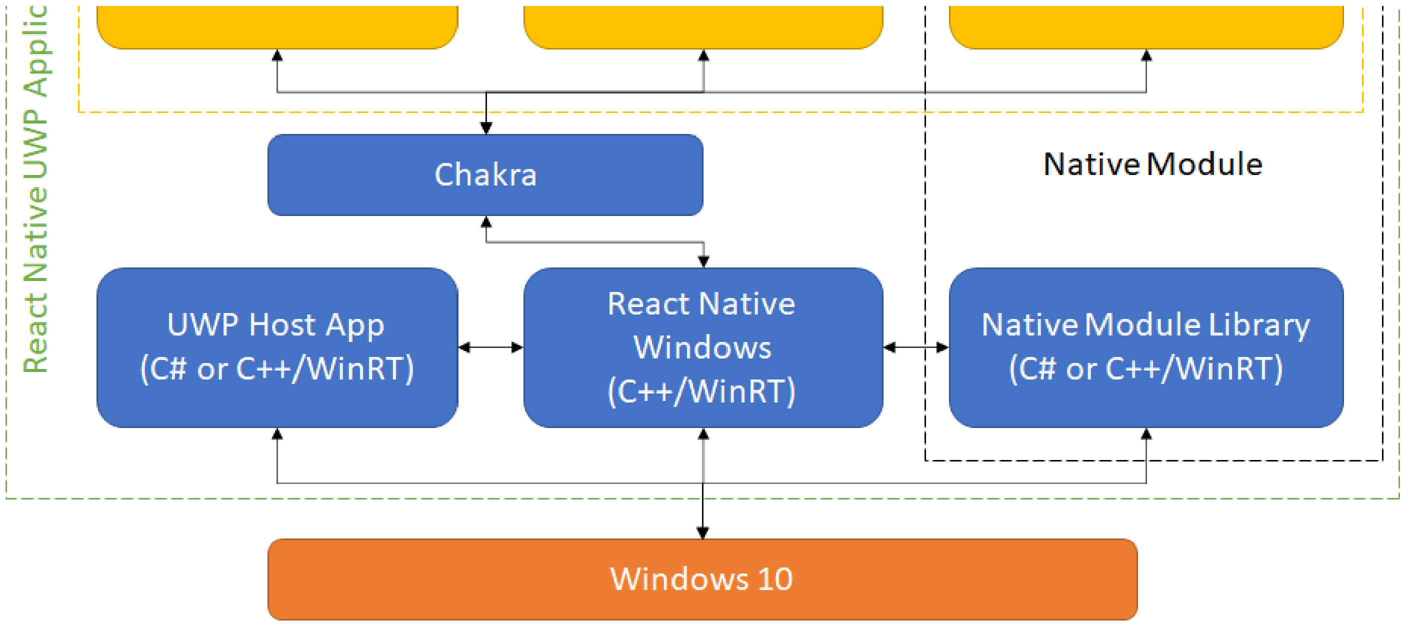
Here's the layout of a minimal React Native Windows UWP application:


React Native UWP Applicatio
App JS Code
React Native JS Code
Chakra
UWP Host App (C# or C++/WinRT)
React Native Windows (C++/WinRT)
Windows 10

The first boundary is perhaps the most obvious - the boundary between the JavaScript VM (Chakra in this case), hosting the bundled JS code of the running application, and the native code of React Native Windows. Across this boundary, JS objects are marshaled into the native code as `folly::dynamic` objects, and vise-versa.

This all happens within `Microsoft.ReactNative` , the compiled library of React Native Windows code. Within the library, the C++ folly::dynamic objects are the primary mechanism for working with JS data.

Now, here's the layout of a React Native Windows UWP application that uses an external Native Module:



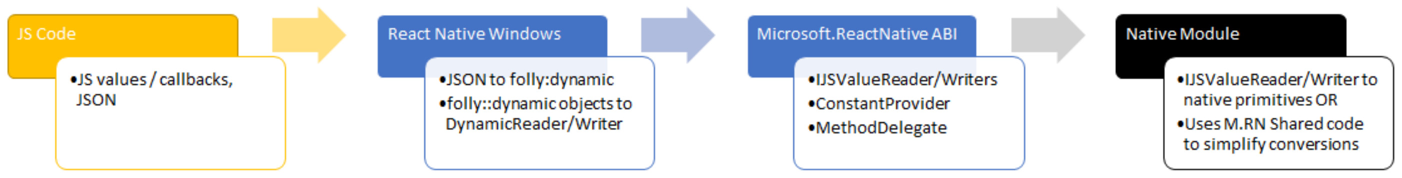
> With external native modules, we mean both those defined in stand-alone Windows Runtime Component libraries and those defined in the host UWP application.

Since we are dealing with a UWP application and need to support external native modules written in both C# and C++/WinRT, the `Microsoft.ReactNative` library is a Windows Runtime Component. This means a WinRT ABI surface, and as such, external native modules interact with React Native Windows across a WinRT boundary.

A lot of work has gone into designing an ABI surface that is as fast and future-proof as possible, especially around the marshaling of JS data.

So after the JS objects have been marshaled into `folly::dynamic` objects internally, we have to further marshal those objects across the WinRT ABI boundary. We do this serially via the high-performance `IJSValueReader` and `IJSValueWriter` interfaces. These interfaces let us (de)serialize data across the WinRT boundary without heap allocations and in a fast, minimal, and future-proof way.

While you can manually use the `IJSValueReader` and `IJSValueWriter` interfaces, we also provide two shared projects, `Microsoft.ReactNative.SharedManaged` for C# and


JS Code
•JS values / callbacks, JSON
React Native Windows
•JSON to folly:dynamic
•folly::dynamic objects to DynamicReader/Writer
Microsoft.ReactNative ABI
•IJSValueReader/Writers
•ConstantProvider
•MethodDelegate
Native Module
•IJSValueReader/Writer to native primitives OR
•Uses M.RN Shared code to simplify conversions

Examples

For examples of using data automatically marshaled into both static and dynamic native types, see the `DataMarshalingExamples` module within the Native Module Sample in `microsoft/react-native-windows-samples`. Implementations for both C# and C++/WinRT are provided.

For further examples of using the dynamic `JSValue` type, see Using `JSValue`.

For examples of marshaling data manually with `IJSValueReader` and `IJSValueWriter`, see Native Modules (Advanced).

‹ Choosing C++ or C# for native code

Using Asynchronous Windows APIs ›