Alchemer Mobile React Native Integration Reference

Getting started

For most React Native apps, installation should be as simple as installing the `apptentive-react-native` module.

```
npm install apptentive-react-native --save
```

Android System Requirements (6+)

- `minSDK` version: >= 21 (Android 5.0)
- `compileSDK` version: 31 to 33
- Gradle version: >= 7.0.0
    - `classpath 'com.android.tools.build:gradle:7.0.0'`

Support Material Dialog Colors

Our native app relies on Google's Material Design library to give more and better styling options to our interactions. To successfully support Android Material dialog colors on React Native you can either have your native Android theme extend a Material theme or simply add this code block to your `styles.xml` file.

```
<style name="ApptentiveThemeOverride">
 <item name="colorSurface">?android:colorBackground</item>
 <item name="colorOnSurface">?colorOnBackground</item>
</style>
```

Usage

Create one app for each supported platform in your Alchemer Mobile Dashboard (i.e. one Android app and one iOS app if you support both platforms that Alchemer Mobile supports). Then navigate to the API & Development section under the Settings tab for each of your apps, and note the Alchemer Mobile (formerly Apptentive) App Key and Alchemer Mobile (formerly Apptentive) App Signature.

> **Credentials:**
>
> Ensure you have separate credentials for each platform.
> Supporting both platforms with one set of credentials is not supported.

Then in your `App.js` file, add code to register the Alchemer Mobile SDK:

```
import { Apptentive, ApptentiveConfiguration } from "apptentive-react-native";

const credentials = Platform.select({
  ios: {
    apptentiveKey: "<YOUR_IOS_APP_KEY>",
    apptentiveSignature: "<YOUR_IOS_APP_SIGNATURE>"
  },
  android: {
    apptentiveKey: "<YOUR_ANDROID_APP_KEY>",
    apptentiveSignature: "<YOUR_ANDROID_APP_SIGNATURE>"
  }
});

export default class App extends Component {
  componentDidMount() {
    const configuration = new ApptentiveConfiguration(
      credentials.apptentiveKey,
      credentials.apptentiveSignature
    );
    // Override log level for debugging (optional)
    configuration.logLevel = 'verbose';
    Apptentive.register(configuration);
  }
}
```

Or in your `App.tsx` file:

```
import React, { useEffect } from 'react';
import { Apptentive, ApptentiveConfiguration } from "apptentive-react-native";

const App = () => {
  const credentials = Platform.select({
    ios: {
      apptentiveKey: "<YOUR_IOS_APP_KEY>",
      apptentiveSignature: "<YOUR_IOS_APP_SIGNATURE>"
    },
    android: {
      apptentiveKey: "<YOUR_ANDROID_APP_KEY>",
      apptentiveSignature: "<YOUR_ANDROID_APP_SIGNATURE>"
    }
  })

  useEffect(() => {
    const configuration = new ApptentiveConfiguration(
      credentials.apptentiveKey,
      credentials.apptentiveSignature
    )

    // Override log level for debugging (optional)
    configuration.logLevel = 'verbose'
    configuration.shouldSanitizeLogMessages = false

    Apptentive.register(configuration)
  }, []);
}
```

If you are not using the `styles.xml` file in your React Native app, we recommend setting `shouldInheritAppTheme` in your Configuration to `false` for well styled interactions out of the box.

To see a full list of `ApptentiveConfiguration` items for Android check here

Dependencies

The React Native plugin for both iOS and Android includes the native SDKs for the respective platforms as transitive dependencies that exist outside of the npm ecosystem (i.e. they are not listed in `package.json`)

The native Android SDK dependency is typically specified down to a particular patch version, for example `6.0.3` (in `android/build.gradle`).

The native iOS SDK is typically specified as some minimum version, up to but not including the next minor version, for example `~> 6.1.0` (in `apptentive-react-native.podspec`).

This means that customers can update the iOS native SDK dependency by running:

```
cd ios
pod update ApptentiveKit
```

...in their project directory.

Events

Events record user interaction. You can use them to determine if and when your dashboard Interactions will be shown to your customer. In your app, trigger an Event with the `Apptentive.engage()` method. This will record the event, and then check to see if any Interactions targeted to that Event are allowed to be displayed, based on the logic you set up on the Alchemer Mobile Dashboard.

```
Apptentive.engage(event).then((result: boolean) => {
 if (result) console.log(`Interaction shown for event: ${event}`);
 else console.log(`Interaction NOT shown for event: ${event}`);
});
```

You can add an Event almost anywhere in your app, just remember that if you want to show an Interaction at that Event, it needs to be a place where launching an Activity will not cause a problem in your app. Some places that may cause problems are right before an Activity change or when your app is in the background.

Now that you have your Events, to test Interactions such as the Love Dialog or Rating Dialog, follow the steps outlined in our native testing guides: Android | iOS

Message Center

See: How to Use Message Center

Showing Message Center

With the Alchemer Mobile Message Center, your customers can send feedback, and you can reply, all without making them leave the app. Handling support inside the app will increase the number of support messages received and ensure a better customer experience.

Message Center lets customers see all the messages they have sent you, read all of your replies, and even send images that may help debug issues.

Find a place in your app where you can add a button that opens Message Center. Your settings page is a good place.

```
<Button title="Message Center" onPress={() => Apptentive.presentMessageCenter()} />
```

To add a check to see if Message Center is available first, use the `Apptentive.canShowMessageCenter` function:

```
...

const [showMessageCenter, setShowMessageCenter] = useState(false);

useEffect(() => {
  Apptentive.canShowMessageCenter().then((canShow: boolean) =>
    setShowMessageCenter(canShow));
}, []);

...

{
  showMessageCenter &&
    <Button title="Message Center" onPress={() => Apptentive.presentMessageCenter()} />
}
```

Unread Message Count

To check the unread message count at any time you can use the `Apptentive.getUnreadMessageCount` function:

```
const [unreadMessageCount, setUnreadMessageCount] = useState(0);

useEffect(() => {
  Apptentive.getUnreadMessageCount().then((count: number) => {
    setUnreadMessageCount(count);
  });
}, []);
```

Alternatively, you can listen for unread message count changes using the `Apptentive.onUnreadMessageCountChanged` callback.

```
 const [unreadMessageCount, setUnreadMessageCount] = useState(0);

useEffect(() => {
  Apptentive.onUnreadMessageCountChanged = (count: number) => {
    setUnreadMessageCount(count);
    return {};
  };
  return () => {};
}, []);
```

Custom Data

You can send Custom Data associated with the device or the person using the app. This is useful for sending user IDs, user attributes (e.g., loyalty status, and other information that helps you support your users better. Custom Data can also be used for configuring when Interactions will run. The calls return a `Promise` and are `.then`-able. You can add custom data of type `string`, `number`, or `boolean`.

```
 // Add Device custom data
Apptentive.addCustomDeviceData('debug', true);
Apptentive.addCustomDeviceData('build', 1);
Apptentive.addCustomDeviceData('version', 'v1.34');

// Remove a Device custom data key/value pair
Apptentive.removeCustomDeviceData('version');


// Add Person custom data
Apptentive.addCustomPersonData('paid', true);
Apptentive.addCustomPersonData('level', 3);
Apptentive.addCustomPersonData('status', 'PRO');

// Remove a Person custom data key/value pair
Apptentive.removeCustomPersonData('status');
```

Push Notifications

Alchemer Mobile iOS can send push notifications to ensure your customers see your replies to their feedback in Message Center.

iOS

On iOS, you'll need to follow Apple's instructions on adding Push capability to your app.

You will need to export your push certificate and key in `.p12` format and upload it to the Integrations section of the Settings tab in your Alchemer Mobile dashboard under "Apptentive Push". You can find more information on this process in the Push Notifications section of our iOS Integration Reference.

You will then edit your `AppDelegate.m` file. First import the Alchemer Mobile SDK at the top level of this file:

```
@import Apptentive;
```

Then add the following methods to your App Delegate class:

```
- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData *
)deviceToken
{
    // Register for Apptentive's push service:
    [Apptentive.shared setPushNotificationIntegration:ApptentivePushProviderApptentive withDeviceToken:dev
iceToken];

    // Uncomment if using PushNotificationsIOS module:
    //[RCTPushNotificationManager didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
}

- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo fetchC
ompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
{
    // Forward the notification to the Apptentive SDK:
    BOOL handledByApptentive = [Apptentive.shared didReceiveRemoteNotification:userInfo fetchCompletion
Handler:completionHandler];

    // Be sure your code calls the completion handler if you expect to receive non-Apptentive push notifications
.
    if (!handledByApptentive) {
        // ...handle the push notification
        // ...and call the completion handler:
        completionHandler(UIBackgroundFetchResultNewData);

        // Uncomment if using PushNotificationIOS module (and remove the above call to `completionHandler`):
        //[RCTPushNotificationManager didReceiveRemoteNotification:userInfo fetchCompletionHandler:completi
onHandler];
    }
}

- (void)application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification *)notification
{
    // Forward the notification to the Apptentive SDK:
    BOOL handledByApptentive = [Apptentive.shared didReceiveLocalNotification:notification fromViewControl
ler:self.window.rootViewController];

    // Uncomment if using PushNotificationIOS module:
    //if (!handledByApptentive) {
    //    [RCTPushNotificationManager didReceiveLocalNotification:notification];
    //}
}
```

Alchemer Mobile's push services work well alongside other push notification services, such as those handled by the PushNotificationIOS React Native module. Note that you will have to implement the handful of additional methods listed in the documentation in your App Delegate to support this module.

Digital Support for International Customers

Alchemer Digital supports global teams by offering both United States (US) and European Union (EU) data center environments. Check out the guidelines here.

Related Articles