Linking

`Linking` gives you a general interface to interact with both incoming and outgoing app links.

Every Link (URL) has a URL Scheme, some websites are prefixed with `https://` or `http://` and the `http` is the URL Scheme. Let's call it scheme for short.

In addition to `https`, you're likely also familiar with the `mailto` scheme. When you open a link with the mailto scheme, your operating system will open an installed mail application. Similarly, there are schemes for making phone calls and sending SMS. Read more about built-in URL schemes below.

Like using the mailto scheme, it's possible to link to other applications by using custom url schemes. For example, when you get a **Magic Link** email from Slack, the **Launch Slack** button is an anchor tag with an href that looks something like: `slack://secret/magic-login/other-secret`. Like with Slack, you can tell the operating system that you want to handle a custom scheme. When the Slack app opens, it receives the URL that was used to open it. This is often referred to as deep linking. Read more about how to get the deep link into your app.

Custom URL scheme isn't the only way to open your application on mobile. You don't want to use a custom URL scheme in links in the email because then the links would be broken on desktop. Instead, you want to use a regular `https` links such as `https://www.myapp.io/records/1234546`. and on mobile you want that link open your app. Android calls it **Deep Links** (Universal Links - iOS).

Built-in URL Schemes

As mentioned in the introduction, there are some URL schemes for core functionality that exist on every platform. The following is a non-exhaustive list, but covers the most commonly used schemes.

SCHEME	DESCRIPTION	IOS	ANDROID

SCHEME	DESCRIPTION	IOS	ANDROID
`tel`	Open phone app, eg: tel:+123456789	✅	✅
`sms`	Open SMS app, eg: sms:+123456789	✅	✅
`https` / `http`	Open web browser app, eg: https://expo.io	✅	✅

Enabling Deep Links

> ### Projects with Native Code Only
>
> The following section only applies to projects with native code exposed. If you are using the managed Expo workflow, see the guide on Linking in the Expo documentation for the appropriate alternative.

If you want to enable deep links in your app, please read the below guide:

Android iOS

> For instructions on how to add support for deep linking on Android, refer to Enabling Deep Links for App Content - Add Intent Filters for Your Deep Links.

If you wish to receive the intent in an existing instance of MainActivity, you may set the `launchMode` of MainActivity to `singleTask` in `AndroidManifest.xml`. See `<activity>` documentation for more information.

```
<activity
  android:name=".MainActivity"
  android:launchMode="singleTask">
```

Handling Deep Links

There are two ways to handle URLs that open your app.

1. If the app is already open, the app is foregrounded and a Linking 'url' event is fired

You can handle these events with `Linking.addEventListener('url', callback)` - it calls `callback({url})` with the linked URL

2. If the app is not already open, it is opened and the url is passed in as the initialURL

You can handle these events with `Linking.getInitialURL()` - it returns a Promise that resolves to the URL, if there is one.

Example

Open Links and Deep Links (Universal Links)

TypeScript JavaScript

Linking Example

Expo

```
import React, {useCallback} from 'react';
import {Alert, Button, Linking, StyleSheet, View} from
'react-native';

const supportedURL = 'https://google.com';

const unsupportedURL = 'slack://open?team=123456';

type OpenURLButtonProps = {
  url: string;
  children: string;
};

const OpenURLButton = ({url, children}: OpenURLButtonProps)
=> {
  const handlePress = useCallback(async () => {
    // Checking if the link is supported for links with
custom URL scheme.
    const supported = await Linking.canOpenURL(url);

    if (supported) {
      // Opening the link with some app, if the URL scheme
is "http" the web link should be opened
      // by some browser in the mobile
      await Linking.openURL(url);
    } else {
```

Download Expo Go and scan the QR code to get started.

Connected devices 0

Log in or set a Device ID to open this Snack from Expo Go on your device or simulator.

Preview My Device iOS Android

Open Custom Settings

TypeScript JavaScript

Linking Example

Expo

```
import React, {useCallback} from 'react';
import {Button, Linking, StyleSheet, View} from 'react-native';

type OpenSettingsButtonProps = {
  children: string;
};

const OpenSettingsButton = ({children}: OpenSettingsButtonProps) => {
  const handlePress = useCallback(async () => {
    // Open the custom settings if the app has one
    await Linking.openSettings();
  }, []);

  return <Button title={children} onPress={handlePress} />;
};

const App = () => {
  return (
    <View style={styles.container}>
      <OpenSettingsButton>Open Settings</OpenSettingsButton>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    --
```

Preview My Device iOS Android

Get the Deep Link

TypeScript JavaScript

Linking Example

Expo

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

const useInitialURL = () => {
  const [url, setUrl] = useState<string | null>(null);
  const [processing, setProcessing] = useState(true);

  useEffect(() => {
    const getUrlAsync = async () => {
      // Get the deep link used to open the app
      const initialUrl = await Linking.getInitialURL();

      // The setTimeout is just for testing purpose
      setTimeout(() => {
        setUrl(initialUrl);
        setProcessing(false);
      }, 1000);
    };

    getUrlAsync();
  }, []);

  return {url, processing};
};
```

Download Expo Go and scan the QR code to get started.

Connected devices 0

Log in or set a Device ID to open this Snack from Expo Go on your device or simulator.

Preview My Device iOS Android

Send Intents (Android)

TypeScript JavaScript

Reference

Methods

addEventListener()

```
static addEventListener(
  type: 'url',
  handler: (event: {url: string}) => void,
): EmitterSubscription;
```

Add a handler to Linking changes by listening to the `url` event type and providing the handler.

`canOpenURL()`

```
static canOpenURL(url: string): Promise<boolean>;
```

Determine whether or not an installed app can handle a given URL.

The method returns a `Promise` object. When it is determined whether or not the given URL can be handled, the promise is resolved and the first parameter is whether or not it can be opened.

The `Promise` will reject on Android if it was impossible to check if the URL can be opened or when targetting Android 11 (SDK 30) if you didn't specify the relevant intent queries in `AndroidManifest.xml`. Similarly on iOS, the promise will reject if you didn't add the specific scheme in the `LSApplicationQueriesSchemes` key inside `Info.plist` (see bellow).

Parameters:

NAME	TYPE	DESCRIPTION
url Required	string	The URL to open.

> For web URLs, the protocol (`"http://"`, `"https://"`) must be set accordingly!

> This method has limitations on iOS 9+. From the official Apple documentation:
>
> - If your app is linked against an earlier version of iOS but is running in iOS 9.0 or later, you can call this method up to 50 times. After reaching that limit, subsequent calls always resolve to `false`. If the user reinstalls or upgrades the app, iOS resets the limit.
>
> As of iOS 9, your app also needs to provide the `LSApplicationQueriesSchemes` key inside `Info.plist` or `canOpenURL()` will always resolve to `false`.

> When targeting Android 11 (SDK 30) you must specify the intents for the schemes you want to handle in `AndroidManifest.xml`. A list of common intents can be found here.
>
> For example to handle `https` schemes the following needs to be added to your manifest:

```
<manifest ...>
    <queries>
        <intent>
            <action android:name="android.intent.action.VIEW" />
            <data android:scheme="https"/>
        </intent>
    </queries>
</manifest>
```

getInitialURL()

```
static getInitialURL(): Promise<string | null>;
```

If the app launch was triggered by an app link, it will give the link url, otherwise it will give `null`.

> To support deep linking on Android, refer http://developer.android.com/training/app-indexing/deep-linking.html#handling-intents

> getInitialURL may return `null` while debugging is enabled. Disable the debugger to ensure it gets passed.

openSettings()

```
static openSettings(): Promise<void>;
```

Open the Settings app and displays the app's custom settings, if it has any.

`openURL()`

```
static openURL(url: string): Promise<any>;
```

Try to open the given `url` with any of the installed apps.

You can use other URLs, like a location (e.g. "geo:37.484847,-122.148386" on Android or "http://maps.apple.com/?ll=37.484847,-122.148386" on iOS), a contact, or any other URL that can be opened with the installed apps.

The method returns a `Promise` object. If the user confirms the open dialog or the url automatically opens, the promise is resolved. If the user cancels the open dialog or there are no registered applications for the url, the promise is rejected.

Parameters:

NAME	TYPE	DESCRIPTION
url Required	string	The URL to open.

> This method will fail if the system doesn't know how to open the specified URL. If you're passing in a non-http(s) URL, it's best to check `canOpenURL()` first.

> For web URLs, the protocol (`"http://"`, `"https://"`) must be set accordingly!

> This method may behave differently in a simulator e.g. `"tel:"` links are not able to be handled in the iOS simulator as there's no access to the dialer app.

`sendIntent()` Android

```
static sendIntent(
  action: string,
  extras?: Array<{key: string; value: string | number | boolean}>,
): Promise<void>;
```

Launch an Android intent with extras.

Parameters:

NAME	TYPE
action Required	string
extras	`Array<{key: string, value: string | number | boolean}>`

Is this page useful? 👍 👎

Edit this page

*Last updated on **Aug 17, 2023***