Docs   Community   Blog   GitHub   React

Images

EDIT

Static Image Resources

React Native provides a unified way of managing images and other media assets in your iOS and Android apps. To add a static image to your app, place it somewhere in your source code tree and reference it like this:

```
<Image source={require('./my-icon.png')} />
```

The image name is resolved the same way JS modules are resolved. In the example above, the packager will look for `my-icon.png` in the same folder as the component that requires it. Also, if you have `my-icon.ios.png` and `my-icon.android.png`, the packager will pick the correct file for the platform.

You can also use the `@2x` and `@3x` suffixes to provide images for different screen densities. If you have the following file structure:

```
.
├── button.js
└── img
    ├── check@2x.png
    └── check@3x.png
```

...and `button.js` code contains:

```
<Image source={require('./img/check.png')} />
```

...the packager will bundle and serve the image corresponding to device's screen density. For example, `check@2x.png`, will be used on an iPhone 7, while `check@3x.png` will be used on an iPhone 7 Plus or a Nexus 5. If there is no image matching the screen density, option will be selected.

React Native 0.54   Search

1. Same system on iOS and Android.

2. Images live in the same folder as your JavaScript code. Components are self-contained.

3. No global namespace, i.e. you don't have to worry about name collisions.

4. Only the images that are actually used will be packaged into your app.

5. Adding and changing images doesn't require app recompilation, just refresh the simulator as you normally do.

6. The packager knows the image dimensions, no need to duplicate it in the code.

7. Images can be distributed via npm packages.

In order for this to work, the image name in `require` has to be known statically.

```
// GOOD
<Image source={require('./my-icon.png')} />;

// BAD
var icon = this.props.active ? 'my-icon-active' : 'my-icon-inactive';
<Image source={require('./' + icon + '.png')} />;

// GOOD
var icon = this.props.active
  ? require('./my-icon-active.png')
  : require('./my-icon-inactive.png');
<Image source={icon} />;
```

Note that image sources required this way include size (width, height) info for the Image. If you need to scale the image dynamically (i.e. via flex), you may need to manually set `{ width: undefined, height: undefined }` on the style attribute.

Static Non-Image Resources

The `require` syntax described above can be used to statically include audio, video or document files in your project as well. Most common file types are supported including `.mp3`, `.wav`, `.mp4`, `.mov`, `.html` and `.pdf`. See packager defaults for the full list.

currently passed for non-image assets. This limitation doesn't occur for videos that are linked directly into Xcode or the Assets folder for Android.

Images From Hybrid App's Resources

If you are building a hybrid app (some UIs in React Native, some UIs in platform code) you can still use images that are already bundled into the app.

For images included via Xcode asset catalogs or in the Android drawable folder, use the image name without the extension:

```
<Image source={{uri: 'app_icon'}} style={{width: 40, height: 40}} />
```

For images in the Android assets folder, use the `asset:/` scheme:

```
<Image source={{uri: 'asset:/app_icon.png'}} style={{width: 40, height: 40}} />
```

These approaches provide no safety checks. It's up to you to guarantee that those images are available in the application. Also you have to specify image dimensions manually.

Network Images

Many of the images you will display in your app will not be available at compile time, or you will want to load some dynamically to keep the binary size down. Unlike with static resources, *you will need to manually specify the dimensions of your image*. It's highly recommended that you use https as well in order to satisfy App Transport Security requirements on iOS.

```
// GOOD
<Image source={{uri: 'https://facebook.github.io/react/logo-og.png'}}
       style={{width: 400, height: 400}} />

// BAD
<Image source={{uri: 'https://facebook.github.io/react/logo-og.png'}} />
```

Network Requests for Images

```
  source={{
    uri: 'https://facebook.github.io/react/logo-og.png',
    method: 'POST',
    headers: {
      Pragma: 'no-cache',
    },
    body: 'Your Body goes here',
  }}
  style={{width: 400, height: 400}}
/>
```

Uri Data Images

Sometimes, you might be getting encoded image data from a REST API call. You can use the `'data:'` uri scheme to use these images. Same as for network resources, *you will need to manually specify the dimensions of your image*.

> This is recommended for very small and dynamic images only, like icons in a list from a DB.

```
// include at least width and height!
<Image
  style={{
    width: 51,
    height: 51,
    resizeMode: Image.resizeMode.contain,
  }}
  source={{
    uri:
      'data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAADMAAAAzCAYAAAA6oTAqAAAAEXRFWHRTb2Z0d2FyZQBw
  }}
/>
```

Cache Control (iOS Only)

In some cases you might only want to display an image if it is already in the local cache, i.e. a low resolution placeholder until a higher resolution is available. In other cases you do not care if the

- `reload`: The data for the URL will be loaded from the originating source. No existing cache data should be used to satisfy a URL load request.

- `force-cache`: The existing cached data will be used to satisfy the request, regardless of its age or expiration date. If there is no existing data in the cache corresponding the request, the data is loaded from the originating source.

- `only-if-cached`: The existing cache data will be used to satisfy a request, regardless of its age or expiration date. If there is no existing data in the cache corresponding to a URL load request, no attempt is made to load the data from the originating source, and the load is considered to have failed.

```
<Image
  source={{
    uri: 'https://facebook.github.io/react/logo-og.png',
    cache: 'only-if-cached',
  }}
  style={{width: 400, height: 400}}
/>
```

Local Filesystem Images

See CameraRoll for an example of using local resources that are outside of `Images.xcassets`.

Best Camera Roll Image

iOS saves multiple sizes for the same image in your Camera Roll, it is very important to pick the one that's as close as possible for performance reasons. You wouldn't want to use the full quality 3264x2448 image as source when displaying a 200x200 thumbnail. If there's an exact match, React Native will pick it, otherwise it's going to use the first one that's at least 50% bigger in order to avoid blur when resizing from a close size. All of this is done by default so you don't have to worry about writing the tedious (and error prone) code to do it yourself.

Why Not Automatically Size Everything?

experience.

In React Native this behavior is intentionally not implemented. It is more work for the developer to know the dimensions (or aspect ratio) of the remote image in advance, but we believe that it leads to a better user experience. Static images loaded from the app bundle via the `require('./my-icon.png')` syntax *can be automatically sized* because their dimensions are available immediately at the time of mounting.

For example, the result of `require('./my-icon.png')` might be:

```
{"__packager_asset":true,"uri":"my-icon.png","width":591,"height":573}
```

Source as an object

In React Native, one interesting decision is that the `src` attribute is named `source` and doesn't take a string but an object with a `uri` attribute.

```
<Image source={{uri: 'something.jpg'}} />
```

On the infrastructure side, the reason is that it allows us to attach metadata to this object. For example if you are using `require('./my-icon.png')`, then we add information about its actual location and size (don't rely on this fact, it might change in the future!). This is also future proofing, for example we may want to support sprites at some point, instead of outputting `{uri: ...}`, we can output `{uri: ..., crop: {left: 10, top: 50, width: 20, height: 40}}` and transparently support spriting on all the existing call sites.

On the user side, this lets you annotate the object with useful attributes such as the dimension of the image in order to compute the size it's going to be displayed in. Feel free to use it as your data structure to store more information about your image.

Background Image via Nesting

A common feature request from developers familiar with the web is `background-image`. To handle this use case, you can use the `<ImageBackground>` component, which has the same props as

component when needed.

```
return (
  <ImageBackground source={...}>
    <Text>Inside</Text>
  </ImageBackground>
);
```

iOS Border Radius Styles

Please note that the following corner specific, border radius style properties are currently ignored by iOS's image component:

- `borderTopLeftRadius`

- `borderTopRightRadius`

- `borderBottomLeftRadius`

- `borderBottomRightRadius`

Off-thread Decoding

Image decoding can take more than a frame-worth of time. This is one of the major sources of frame drops on the web because decoding is done in the main thread. In React Native, image decoding is done in a different thread. In practice, you already need to handle the case when the image is not downloaded yet, so displaying the placeholder for a few more frames while it is decoding does not require any code change.

Docs   Community   More

Getting   Who's   Blog
Started   using React Twitter
                  Native?
Tutorial   GitHub

Components Stack   React
and APIs   Overflow

Docs   Community   Blog   GitHub   React

Copyright © 2018 Facebook Inc.