API and Configuration

Available imports

- `Router`
- `Scene`
- `Tabs`
- `Tabbed Scene`
- `Drawer`
- `Modal`
- `Lightbox`
- `Actions`
- `ActionConst`

Router:

Property	Type	Default	Descripti
children		必要	页面根组件
`wrapBy`	`Function`		允许集成诸如Redux（connect）和Mobx
`sceneStyle`	`Style`		适用于所有场景的Style（可选）
`backAndroidHandler`	`Function`		允许在Android中自定义控制返回按钮（可 ckHandler

Scene:

此路由器的最重要的组件， 所有 `<Scene>` 组件必须要有一个唯一的 `key` 。父节点 `<Scene>` 不能将 `component` 作为 `prop`， 因为它将作为其子节点的分组组件。

Property	Type	Default	
`key`	`string`	`required`	将用于标识
`component`	`React.Component`	`semi-required`	要显示的组

`back`	`boolean`	`false`	如果是 `tru`
`backButtonImage`	`string`		设置返回按
`backButtonTintColor`	`string`		自定义后退
`init`	`boolean`	`false`	如果是 `tru`
`clone`	`boolean`	`false`	标有 `clone`
`contentComponent`	`React.Component`		Componer
`drawer`	`boolean`	`false`	load child s
`failure`	`Function`		If `on` retur
`backTitle`	`string`		Specifies t
`headerMode`	`string`	`float`	Specifies h top and an screen has This is a co
`hideNavBar`	`boolean`	`false`	hide the na
`hideTabBar`	`boolean`	`false`	hide the ta
`hideBackImage`	`boolean`	`false`	hide back i
`initial`	`boolean`	`false`	Set to `tru`
`leftButtonImage`	`Image`		Image to s
`leftButtonTextStyle`	`Style`		Style appli
`modal`	`boolean`	`false`	Defines sc animation.
`navBar`	`React.Component`		Optional R
`navBarButtonColor`	`string`		Set the col
`navigationBarStyle`	`Style`		Style appli
`navigationBarTitleImage`	`Object`		The `Image` navbar
`navigationBarTitleImageStyle`	`object`		Styles to a

`navTransparent`	`boolean`	`false`	nav bar ba
`on`	`Function`		aka `onEnt`
`onEnter`	`Function`		Called whe scenes wit
`onExit`	`Function`		Called whe supported
`onLeft`	`Function`		Called whe
`onRight`	`Function`		Called whe
`renderTitle`	`React.Component`		React com
`renderLeftButton`	`React.Component`		React com
`renderRightButton`	`React.Component`		React com
`renderBackButton`	`React.Component`		React com
`rightButtonImage`	`Image`		Image to s
`rightButtonTextStyle`	`Style`		Style appli
`success`	`Function`		If `on` retur
`tabs`	`boolean`	`false`	load child
`title`	`string`		Text to be
`titleStyle`	`Style`		Style appli
`type`	`string`	`push`	Optional ty scene
all other props			Any other

Tabs (`<Tabs>` or `<Scene tabs>`)

Can use all `props` listed above in `<Scene>` as `<Tabs>` is syntatic sugar for `<Scene tabs={true}>` .

Property	Type	Default	Description

`wrap`	`boolean`	`true`	Wrap each scene with own navbar automatically (if it is not another container).
`activeBackgroundColor`	`string`		Specifies the active background color for the tab in focus
`activeTintColor`	`string`		Specifies the active tint color for tabbar icons
`inactiveBackgroundColor`	`string`		Specifies the inactive background color for the tabs not in focus
`inactiveTintColor`	`string`		Specifies the inactive tint color for tabbar icons
`labelStyle`	`object`		Overrides the styles for the tab label
`lazy`	`boolean`	`false`	Won't render/mount the tab scene until the tab is active
`tabBarComponent`	`React.Component`		React component to render custom tab bar
`tabBarPosition`	`string`		Specifies tabbar position. Defaults to `bottom` on iOS and `top` on Android.
`tabBarStyle`	`object`		Override the tabbar styles
`tabStyle`	`object`		Override the style for an individual tab of the tabbar
`showLabel`	`boolean`	`true`	Boolean to show or not the tabbar icons labels

`swipeEnabled`	`boolean`	`true`	Enable or disable swiping tabs.
`tabBarOnPress`	`function`		Custom tab bar icon press.
`backToInitial`	`boolean`	`false`	Back to initial screen on focused tab if tab icon was tapped.

Stack (`<Stack>`)

A component to group Scenes together for its own stack based navigation. Using this will create a separate havigator for this stack, so expect two navbars to appear unless you add `hideNavBar`.

Tab Scene (child `<Scene>` within `Tabs`)

A `Scene` that is a direct child of `Tabs` and can use all `props` listed above in `Scene`,

Property	Type	Default	Description
`icon`	`component`	`undefined`	a React Native component to place as a tab icon
`tabBarLabel`	`string`		The string to override a tab label

Drawer (`<Drawer>` or `<Scene drawer>`)

Can use all `prop` as listed in `Scene` as `<Drawer>`, syntatic sugar for `<Scene drawer={true}>`

Property	Type	Default	Description
`drawerImage`	`Image`		Image to substitute drawer 'hamburger' icon, you have to set it together with `drawer` prop

`drawerIcon`	`React.Component`		Arbitrary component to be used for drawer 'hamburger' icon, you have to set it together with `drawer` prop
`hideDrawerButton`	`boolean`	`false`	Boolean to show or not the drawerImage or drawerIcon
`drawerPosition`	`string`	Determines whether the drawer is on the right or the left. Keywords accepted are `right` and `left`	
`drawerWidth`	`number`		The width, in pixels, of the drawer (optional)

Modals (`<Modal>` or `<Scene modal>`)

To implement a modal, you must use `<Modal>` as the root scene in your `Router`. The `Modal` will render the first scene (should be your true root scene) normally, and all following To display a modal use `<Modal>` as root renderer, so it will render the first element as `normal` scene and all others as popups (when they are pushed).

Example:
In the example below, the `root` Scene is nested within a `<Modal>`, since it is the first nested `Scene`, it will render normally. If one were to `push` to `statusModal`, `errorModal` or `loginModal`, they will render as a `Modal` and by default will pull up from the bottom of the screen. It is important to note that currently the `Modal` does not allow for transparent backgrounds.

```
//... import components
<Router>
```

```
  <Modal>
    <Scene key="root">
      <Scene key="screen1" initial={true} component={Screen1} />
      <Scene key="screen2" component={Screen2} />
    </Scene>
    <Scene key="statusModal" component={StatusModal} />
    <Scene key="errorModal" component={ErrorModal} />
    <Scene key="loginModal" component={LoginModal} />
  </Modal>
</Router>
```

Lightbox (`<Lightbox>`)

Lightbox is a component used to render a component on top of the current `Scene`. Unlike modal, it will allow for resizing and transparency of the background.

Example:
In the example below, the `root` Scene is nested within a `<Lightbox>`, since it is the first nested `Scene`, it will render normally. If one were to `push` to `loginLightbox`, they will render as a `Lightbox` and by default will lay on top of the current `Scene` allowing for transparent backrounds.

```
//... import components
<Router>
  <Lightbox>
    <Scene key="root">
      <Scene key="screen1" initial={true} component={Screen1} />
      <Scene key="screen2" component={Screen2} />
    </Scene>

    {/* Lightbox components will lay over the screen, allowing transparency*/}
    <Scene key="loginLightbox" component={loginLightbox} />
  </Lightbox>
</Router>
```

Actions

This `Object` is the main utility is to provide navigation features to your application. Assuming your `Router` and `Scenes` are configured properly, use the properties listed below to navigate between scenes. Some offer the added functionality to pass React `props` to the navigated scene.

These can be used directly, for example, `Actions.pop()` will dispatch correspond action written in the source code, or, you can set those constants in scene type, when you do Actions.main(), it will dispatch action according to your scene type or the default one.

Property	Type	Parameters	Description
`[key]`	`Function`	`Object`	The `Actions` object "automagically" uses the `Scene`'s `key` prop in the `Router` to navigate. To navigate to a scene, call `Actions.key()` or `Actions[key].call()`.
`currentScene`	`String`		Returns the current scene that is active
`jump`	`Function`	`(sceneKey: String, props: Object)`	used to switch to a new tab. For `Tabs` only.
`pop`	`Function`		Go back to the previous scene by "popping" the current scene off the nav stack
`popTo`	`Function`	`(sceneKey: String, props: Object)`	Pops the navigation stack until the `Scene` with the specified key is reached.
`push`	`Function`	`(sceneKey: String, props: Object)`	Pushes the scene to the stack, performing a transition to the new scene.
`refresh`	`Function`	`(props: Object)`	Reloads the current scene by loading new `props` into the `Scene`
`replace`	`Function`	`(sceneKey: String, props: Object)`	Pops the current scene from the stack and pushes the new scene to the navigation stack. *No transition will occur.
		`(sceneKey: String,`	Clears the routing stack and pushes the scene into the first index. *No transition*

`reset`	`Function`	`props: Object)`	*will occur.*
`drawerOpen`	`Function`		Opens the `Drawer` if applicable
`drawerClose`	`Function`		Closes the `Drawer` if applicable

ActionConst

Type constants to determine `Scene` transitions, These are **PREFERRED** over typing their values manually as these are subject to change as the project is updated.

Property	Type	Value
`ActionConst.JUMP`	`string`	'REACT_NATIVE_ROUTER_FLUX_JUMP'
`ActionConst.PUSH`	`string`	'REACT_NATIVE_ROUTER_FLUX_PUSH'
`ActionConst.PUSH_OR_POP`	`string`	'REACT_NATIVE_ROUTER_FLUX_PUSH_OR_
`ActionConst.REPLACE`	`string`	'REACT_NATIVE_ROUTER_FLUX_REPLACE'
`ActionConst.BACK`	`string`	'REACT_NATIVE_ROUTER_FLUX_BACK'
`ActionConst.BACK_ACTION`	`string`	'REACT_NATIVE_ROUTER_FLUX_BACK_ACTI
`ActionConst.POP_TO`	`string`	'REACT_NATIVE_ROUTER_FLUX_POP_TO'
`ActionConst.REFRESH`	`string`	'REACT_NATIVE_ROUTER_FLUX_REFRESH'
`ActionConst.RESET`	`string`	'REACT_NATIVE_ROUTER_FLUX_RESET'
`ActionConst.FOCUS`	`string`	'REACT_NATIVE_ROUTER_FLUX_FOCUS'
`ActionConst.BLUR`	`string`	'REACT_NATIVE_ROUTER_FLUX_BLUR'
`ActionConst.ANDROID_BACK`	`string`	'REACT_NATIVE_ROUTER_FLUX_ANDROID_