NEXT.JS

Menu

App Router > ... > Components > <Script>

<Script>

This API reference will help you understand how to use props available for the Script Component. For features and usage, please see the Optimizing Scripts page.

app/dashboard/page.tsx TypeScript

```tsx
import Script from 'next/script'

export default function Dashboard() {
  return (
    <>
      <Script src="https://example.com/script.js" />
    </>
  )
}
```

Props

Here's a summary of the props available for the Script Component:

Prop	Example	Type	Required
`src`	`src="http://example.com/script"`	String	Required unless inline script is used
`strategy`	`strategy="lazyOnload"`	String	-

Prop	Example	Type	Required
`onLoad`	`onLoad={onLoadFunc}`	Function	-
`onReady`	`onReady={onReadyFunc}`	Function	-
`onError`	`onError={onErrorFunc}`	Function	-

Required Props

The `<Script />` component requires the following properties.

`src`

A path string specifying the URL of an external script. This can be either an absolute external URL or an internal path. The `src` property is required unless an inline script is used.

Optional Props

The `<Script />` component accepts a number of additional properties beyond those which are required.

`strategy`

The loading strategy of the script. There are four different strategies that can be used:

- `beforeInteractive`: Load before any Next.js code and before any page hydration occurs.
- `afterInteractive`: (**default**) Load early but after some hydration on the page occurs.
- `lazyOnload`: Load during browser idle time.
- `worker`: (experimental) Load in a web worker.

`beforeInteractive`

Scripts that load with the `beforeInteractive` strategy are injected into the initial HTML from the server, downloaded before any Next.js module, and executed in the order they are placed before *any* hydration occurs on the page.

Scripts denoted with this strategy are preloaded and fetched before any first-party code, but their execution does not block page hydration from occurring.

`beforeInteractive` scripts must be placed inside the root layout (`app/layout.tsx`) and are designed to load scripts that are needed by the entire site (i.e. the script will load when any page in the application has been loaded server-side).

This strategy should only be used for critical scripts that need to be fetched before any part of the page becomes interactive.

app/layout.tsx TypeScript

```tsx
import Script from 'next/script'

export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="en">
      <body>
        {children}
        <Script
          src="https://example.com/script.js"
          strategy="beforeInteractive"
        />
      </body>
    </html>
  )
}
```

Good to know: Scripts with `beforeInteractive` will always be injected inside the `head` of the HTML document regardless of where it's placed in the component.

Some examples of scripts that should be loaded as soon as possible with `beforeInteractive` include:

- Bot detectors
- Cookie consent managers

`afterInteractive`

Scripts that use the `afterInteractive` strategy are injected into the HTML client-side and will load after some (or all) hydration occurs on the page. **This is the default strategy** of the Script component and should be used for any script that needs to load as soon as possible but not before any first-party Next.js code.

`afterInteractive` scripts can be placed inside of any page or layout and will only load and execute when that page (or group of pages) is opened in the browser.

app/page.js

```
import Script from 'next/script'

export default function Page() {
  return (
    <>
      <Script src="https://example.com/script.js" strategy="afterInteractive" />
    </>
  )
}
```

Some examples of scripts that are good candidates for `afterInteractive` include:

- Tag managers
- Analytics

`lazyOnload`

Scripts that use the `lazyOnload` strategy are injected into the HTML client-side during browser idle time and will load after all resources on the page have been fetched. This strategy should be used for any background or low priority scripts that do not need to load early.

`lazyOnload` scripts can be placed inside of any page or layout and will only load and execute when that page (or group of pages) is opened in the browser.

app/page.js

```js
import Script from 'next/script'

export default function Page() {
  return (
    <>
      <Script src="https://example.com/script.js" strategy="lazyOnload" />
    </>
  )
}
```

Examples of scripts that do not need to load immediately and can be fetched with `lazyOnload` include:

- Chat support plugins
- Social media widgets

`worker`

Warning: The `worker` strategy is not yet stable and does not yet work with the `app` directory. Use with caution.

Scripts that use the `worker` strategy are off-loaded to a web worker in order to free up the main thread and ensure that only critical, first-party resources are processed on it. While this strategy can be used for any script, it is an advanced use case that is not guaranteed to support all third-party scripts.

To use `worker` as a strategy, the `nextScriptWorkers` flag must be enabled in `next.config.js`:

next.config.js

```js
module.exports = {
  experimental: {
    nextScriptWorkers: true,
  },
}
```

`worker` scripts can **only currently be used in the** `pages/` **directory**:

pages/home.tsx TypeScript

```tsx
import Script from 'next/script'

export default function Home() {
  return (
    <>
      <Script src="https://example.com/script.js" strategy="worker" />
    </>
  )
}
```

`onLoad`

Warning: `onLoad` does not yet work with Server Components and can only be used in Client Components. Further, `onLoad` can't be used with `beforeInteractive` – consider using `onReady` instead.

Some third-party scripts require users to run JavaScript code once after the script has finished loading in order to instantiate content or call a function. If you are loading a script with either afterInteractive or lazyOnload as a loading strategy, you can execute code after it has loaded using the onLoad property.

Here's an example of executing a lodash method only after the library has been loaded.

app/page.tsx TypeScript

```tsx
'use client'

import Script from 'next/script'

export default function Page() {
  return (
    <>
      <Script
        src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.20/lodash.min.j
        onLoad={() => {
          console.log(_.sample([1, 2, 3, 4]))
        }}
      />
    </>
  )
}
```

`onReady`

Warning: `onReady` does not yet work with Server Components and can only be used in Client Components.

Some third-party scripts require users to run JavaScript code after the script has finished loading and every time the component is mounted (after a route navigation for example). You can execute code after the script's load event when it first loads and then after every subsequent component re-mount using the onReady property.

Here's an example of how to re-instantiate a Google Maps JS embed every time the component is mounted:

app/page.tsx TypeScript

```tsx
'use client'

import { useRef } from 'react'
import Script from 'next/script'

```

```
export default function Page() {
  const mapRef = useRef()

  return (
    <>
      <div ref={mapRef}></div>
      <Script
        id="google-maps"
        src="https://maps.googleapis.com/maps/api/js"
        onReady={() => {
          new google.maps.Map(mapRef.current, {
            center: { lat: -34.397, lng: 150.644 },
            zoom: 8,
          })
        }}
      />
    </>
  )
}
```

`onError`

Warning: `onError` does not yet work with Server Components and can only be used in Client Components. `onError` cannot be used with the `beforeInteractive` loading strategy.

Sometimes it is helpful to catch when a script fails to load. These errors can be handled with the onError property:

app/page.tsx TypeScript

```tsx
'use client'

import Script from 'next/script'

export default function Page() {
  return (
    <>
      <Script
        src="https://example.com/script.js"
        onError={(e: Error) => {
          console.error('Script failed to load', e)
        }}
      />
    </>
  )
}
```

Version History

Version	Changes
`v13.0.0`	`beforeInteractive` and `afterInteractive` is modified to support `app`.
`v12.2.4`	`onReady` prop added.
`v12.2.2`	Allow `next/script` with `beforeInteractive` to be placed in `_document`.
`v11.0.0`	`next/script` introduced.

Previous
<Link>

Next
File Conventions

Next.js Conf Governance

Previews

Subscribe to our newsletter

Stay updated on new releases and features, guides, and case studies.

you@domain.com Subscribe