what are all these methods in expo fonts `import \* as Fonts from 'expo-font';

Fonts.FontDisplay

Fonts.isLoaded

Fonts.isLoading

Fonts.loadAsync

Fonts.processFontFamily

Fonts.unloadAllAsync

Fonts.unloadAsync`

and can u provide an example about each of them

ChatGPT

2 / 2

The expo-font library in Expo allows you to work with custom fonts in your React Native applications. Here's a brief explanation of each of the methods you mentioned along with examples:

Fonts.FontDisplay:

This is an enum that defines the font display strategy. It can have one of the following values:

auto

block

fallback

optional

swap

Example:

javascript

Copy code

import { Fonts } from 'expo-font';

const fontDisplayStrategy = Fonts.FontDisplay.auto;

Fonts.isLoaded:

A function that checks whether the specified font is loaded or not.

Example:

javascript

Copy code

import { Fonts } from 'expo-font';

const isFontLoaded = Fonts.isLoaded('YourFontName');

Fonts.isLoading:

A function that checks whether the specified font is still loading or not.

Example:

javascript

Copy code

import { Fonts } from 'expo-font';

const isFontLoading = Fonts.isLoading('YourFontName');

Fonts.loadAsync:

A method for loading custom fonts asynchronously. It returns a Promise.

Example:

javascript

Copy code

import { Fonts } from 'expo-font';

async function loadFonts() {

await Fonts.loadAsync({

'YourFontName': require('./path/to/YourFontFile.ttf'),

});

}

Fonts.processFontFamily:

A function that returns a processed font family string that can be used in styles.

Example:

javascript

Copy code

import { Fonts } from 'expo-font';

const fontFamily = Fonts.processFontFamily({ 'YourFontName': 'Your-Font' });

Fonts.unloadAllAsync:

A method that unloads all the fonts that have been loaded using Fonts.loadAsync.

Example:

javascript

Copy code

import { Fonts } from 'expo-font';

async function unloadAllFonts() {

await Fonts.unloadAllAsync();

}

Fonts.unloadAsync:

A method that unloads a specific font that has been loaded using Fonts.loadAsync.

Example:

javascript

Copy code

import { Fonts } from 'expo-font';

async function unloadFont() {

await Fonts.unloadAsync('YourFontName');

}

These methods are used to manage custom fonts in your Expo application, allowing you to load and use fonts in a more controlled and efficient manner. Remember to replace 'YourFontName' and 'YourFontFile.ttf' with the actual font name and file path in your application.

User

i didnt understand what Fonts.FontDisplay do and what its enums do `auto

block

fallback

optional

swap`

ChatGPT

Fonts.FontDisplay is an enum that defines the font display strategy. Font display strategies dictate how a web font (or any font) should be displayed or rendered while it is being loaded or if there are issues loading it. These strategies are relevant when working with web fonts in web development, and they can also be used in React Native applications when you want to define how a custom font should be handled during loading.

Here's what each of the enum values means:

auto:

The default behavior. The browser or rendering engine chooses the most suitable strategy based on the context. It often means fonts are initially hidden until they are fully loaded. This is the most common behavior for fonts.

block:

The browser or rendering engine hides text using the web font until the font is fully loaded. This can prevent the flash of unstyled text (FOUT) but may lead to a blank space where the text should be until the font is loaded.

fallback:

The browser or rendering engine uses a fallback font (like a system font) to display the text until the web font is loaded. This avoids FOUT and ensures that text is immediately visible but might not match the final font.

optional:

Indicates that the font is not essential for rendering the content, and the browser may choose not to load it at all. This can be useful for decorative fonts or icons that are not critical to the page's content.

swap:

The browser or rendering engine displays text using a fallback font until the web font is loaded, and then it swaps in the web font. This provides a balance between avoiding FOUT and ensuring that the final font is used when available.

In the context of React Native and the expo-font library, you can use these font display strategies when defining how a custom font should be handled during loading. For example, you might choose Fonts.FontDisplay.block if you want to hide text until the custom font is fully loaded, or you might choose Fonts.FontDisplay.swap if you want to use a fallback font initially and then swap in the custom font when it's loaded.

Here's an example of how to use Fonts.FontDisplay:

javascript

Copy code

import { Fonts } from 'expo-font';

const fontDisplayStrategy = Fonts.FontDisplay.block; // You can change this to any of the enum values.

Save to grepper

The choice of which strategy to use depends on your design and performance considerations for your application.

User

what is better, to use Fonts.loadAsync or use `useFonts` hook

ChatGPT

Whether to use Fonts.loadAsync or the useFonts hook depends on your specific use case and how you want to manage custom fonts in your Expo or React Native application.

Fonts.loadAsync:

Use Fonts.loadAsync when you want to load custom fonts asynchronously before rendering your components. This method is useful when you need to ensure that fonts are loaded and ready for use before any text elements are displayed on the screen.

It allows you to control the loading process, and you can wait for the fonts to be loaded using await before rendering your components.

Useful if you have specific font loading requirements or if you need to load fonts conditionally based on certain app states.

Example:

javascript

Copy code

import { Fonts } from 'expo-font';

async function loadFonts() {

await Fonts.loadAsync({

'YourFontName': require('./path/to/YourFontFile.ttf'),

});

}

useFonts hook (from expo-font or react-native-elements):

Use the useFonts hook when you want to load fonts in a more declarative way within functional components.

It simplifies the process of loading fonts and automatically handles font loading and unloading for you.

Suitable for scenarios where you want to load fonts for a specific component or screen and don't need to manage the loading process explicitly.

Example (using expo-font):

javascript

Copy code

import { useFonts } from 'expo-font';

function MyComponent() {

const [loaded] = useFonts({

'YourFontName': require('./path/to/YourFontFile.ttf'),

});

if (!loaded) {

// Fonts are still loading, you can render a loading indicator here.

return null;

}

return (

// Your component that uses the custom font.

);

}

In summary, if you need fine-grained control over the font loading process and want to ensure fonts are loaded before rendering any components, Fonts.loadAsync is a good choice. On the other hand, if you prefer a more declarative approach and don't need to manage fonts at a global level, using the useFonts hook can make your code cleaner and more concise. The choice depends on your specific project requirements and coding preferences.

User

so we can say `Fonts.loadAsync` we shall use it universally cases and `useFonts` we shall use it Locally cases!

ChatGPT

Yes, you can generalize the use of Fonts.loadAsync and useFonts as follows:

Use Fonts.loadAsync when you want to load fonts universally or at a global level, ensuring that fonts are loaded before any part of your application relies on them. This is useful when you have fonts that are used across multiple screens or components and you want to manage their loading centrally.

Use useFonts when you want to load fonts locally or within specific components. This hook is suitable for cases where fonts are only needed in a particular screen or component and you want a more declarative way to manage their loading without affecting the rest of your application.

The choice between the two methods ultimately depends on your application's architecture and how you want to organize and control the font loading process. Both methods are valid and serve different use cases, so you can choose the one that best fits your project's needs.

User

but this mean that `Fonts.loadAsync` it will be rendered at initial rendering while `useFonts` will be rendered as lazy rendering

ChatGPT

Yes, you are correct in your observation:

Fonts.loadAsync is typically used for initial rendering and ensures that fonts are loaded before any components are rendered. It's a synchronous operation, meaning it blocks the rendering of your components until the fonts are loaded.

useFonts is typically used for lazy rendering, where you can load fonts as needed within specific components. It allows you to continue rendering other parts of your application while fonts are being loaded in the background. useFonts returns a loading state that you can use to conditionally render components or loading indicators until the fonts are ready.

This distinction aligns with the concept of lazy loading, where you load resources like fonts or data only when they are required, rather than loading everything upfront during the initial app load. The choice between these methods depends on your application's requirements and performance considerations.