

systemPreferences

Get system preferences.

Process: [Main](#)

```
const { systemPreferences } = require('electron')
console.log(systemPreferences.isDarkMode())
```

Events

The `systemPreferences` object emits the following events:

Event: 'accent-color-changed' *Windows*

Returns:

- `event` Event
- `newColor` string - The new RGBA color the user assigned to be their system accent color.

Event: 'color-changed' *Windows*

Returns:

- `event` Event

Event: 'inverted-color-scheme-changed' *Windows Deprecated*

Returns:

- `event` Event
- `invertedColorScheme` boolean - `true` if an inverted color scheme (a high contrast color scheme with light text and dark backgrounds) is being used, `false` otherwise.

Deprecated: Should use the new [updated](#) event on the `nativeTheme` module.

Event: 'high-contrast-color-scheme-changed' *Windows Deprecated*

Returns:

- `event` Event
- `highContrastColorScheme` boolean - `true` if a high contrast theme is being used, `false` otherwise.

Deprecated: Should use the new [updated](#) event on the `nativeTheme` module.

Methods

`systemPreferences.isDarkMode()` *macOS Windows Deprecated*

Returns `boolean` - Whether the system is in Dark Mode.

Deprecated: Should use the new [nativeTheme.shouldUseDarkColors](#) API.

`systemPreferences.isSwipeTrackingFromScrollEventsEnabled()` *macOS*

Returns `boolean` - Whether the Swipe between pages setting is on.

`systemPreferences.postNotification(event, userInfo[, deliverImmediately])`

macOS

- `event` `string`
- `userInfo` `Record<string, any>`
- `deliverImmediately` `boolean` (optional) - `true` to post notifications immediately even when the subscribing app is inactive.

Posts `event` as native notifications of macOS. The `userInfo` is an Object that contains the user information dictionary sent along with the notification.

`systemPreferences.postLocalNotification(event, userInfo)` *macOS*

- `event` `string`
- `userInfo` `Record<string, any>`

Posts `event` as native notifications of macOS. The `userInfo` is an Object that contains the user information dictionary sent along with the notification.

`systemPreferences.postWorkspaceNotification(event, userInfo)` *macOS*

- `event` `string`
- `userInfo` `Record<string, any>`

Posts `event` as native notifications of macOS. The `userInfo` is an Object that contains the user information dictionary sent along with the notification.

`systemPreferences.subscribeNotification(event, callback)` *macOS*

- `event` `string`
- `callback` `Function`
 - `event` `string`
 - `userInfo` `Record<string, unknown>`
 - `object` `string`

Returns `number` - The ID of this subscription

Subscribes to native notifications of macOS, `callback` will be called with `callback(event, userInfo)` when the corresponding `event` happens. The `userInfo` is an Object that contains the user information dictionary sent along with the notification. The `object` is the sender of the notification, and only supports `NSString` values for now.

The `id` of the subscriber is returned, which can be used to unsubscribe the `event`.

Under the hood this API subscribes to `NSDistributedNotificationCenter`, example values of `event` are:

- `AppleInterfaceThemeChangedNotification`
- `AppleAquaColorVariantChanged`
- `AppleColorPreferencesChangedNotification`
- `AppleShowScrollBarsSettingChanged`

`systemPreferences.subscribeLocalNotification(event, callback)` *macOS*

- `event` string
- `callback` Function
 - `event` string
 - `userInfo` Record<string, unknown>
 - `object` string

Returns `number` - The ID of this subscription

Same as `subscribeNotification`, but uses `NSNotificationCenter` for local defaults. This is necessary for events such as `NSUserDefaultsDidChangeNotification`.

`systemPreferences.subscribeWorkspaceNotification(event, callback)` *macOS*

- `event` string
- `callback` Function
 - `event` string
 - `userInfo` Record<string, unknown>
 - `object` string

Returns `number` - The ID of this subscription

Same as `subscribeNotification`, but uses `NSWorkspace.sharedWorkspace.notificationCenter`. This is necessary for events such as `NSWorkspaceDidActivateApplicationNotification`.

`systemPreferences.unsubscribeNotification(id)` *macOS*

- `id` Integer

Removes the subscriber with `id`.

`systemPreferences.unsubscribeLocalNotification(id)` *macOS*

- `id` Integer

Same as `unsubscribeNotification`, but removes the subscriber from `NSNotificationCenter`.

`systemPreferences.unsubscribeWorkspaceNotification(id)` *macOS*

- `id` Integer

Same as `unsubscribeNotification`, but removes the subscriber from `NSWorkspace.sharedWorkspace.notificationCenter`.

`systemPreferences.registerDefaults(defaults)` *macOS*

- `defaults` Record<string, string | boolean | number> - a dictionary of (`key`: `value`) user defaults

Add the specified defaults to your application's `NSUserDefaults`.

`systemPreferences.getUserDefault<Type extends keyof UserDefaultTypes>(key, type)` *macOS*

- `key` string

- `type Type` - Can be `string`, `boolean`, `integer`, `float`, `double`, `url`, `array` or `dictionary`.

Returns `UserDefaultTypes[Type]` - The value of `key` in `NSUserDefaults`.

Some popular `key` and `type` s are:

- `AppleInterfaceStyle` : `string`
- `AppleAquaColorVariant` : `integer`
- `AppleHighlightColor` : `string`
- `AppleShowScrollBars` : `string`
- `NSNavRecentPlaces` : `array`
- `NSPreferredWebServices` : `dictionary`
- `NSUserDefaultsDictionaryReplacementItems` : `array`

`systemPreferences.setUserDefault<Type extends keyof UserDefaultTypes>(key, type, value)` *macOS*

- `key` `string`
- `type Type` - Can be `string`, `boolean`, `integer`, `float`, `double`, `url`, `array` or `dictionary`.
- `value` `UserDefaultTypes[Type]`

Set the value of `key` in `NSUserDefaults`.

Note that `type` should match actual type of `value`. An exception is thrown if they don't.

Some popular `key` and `type` s are:

- `ApplePressAndHoldEnabled` : `boolean`

`systemPreferences.removeUserDefault(key)` *macOS*

- `key` `string`

Removes the `key` in `NSUserDefaults`. This can be used to restore the default or global value of a `key` previously set with `setUserDefault`.

`systemPreferences.isAeroGlassEnabled()` *Windows*

Returns `boolean` - `true` if [DWM composition](#) (Aero Glass) is enabled, and `false` otherwise.

An example of using it to determine if you should create a transparent window or not (transparent windows won't work correctly when DWM composition is disabled):

```
const { BrowserWindow, systemPreferences } = require('electron')
const browserOptions = { width: 1000, height: 800 }

// Make the window transparent only if the platform supports it.
if (process.platform !== 'win32' || systemPreferences.isAeroGlassEnabled()) {
  browserOptions.transparent = true
  browserOptions.frame = false
}
```

```
// Create the window.
const win = new BrowserWindow(browserOptions)

// Navigate.
if (browserOptions.transparent) {
  win.loadURL(`file://${__dirname}/index.html`)
} else {
  // No transparency, so we load a fallback that uses basic styles.
  win.loadURL(`file://${__dirname}/fallback.html`)
}
```

`systemPreferences.getAccentColor()` *Windows macOS*

Returns `string` - The users current system wide accent color preference in RGBA hexadecimal form.

```
const color = systemPreferences.getAccentColor() // `aabbccdd`
const red = color.substr(0, 2) // "aa"
const green = color.substr(2, 2) // "bb"
const blue = color.substr(4, 2) // "cc"
const alpha = color.substr(6, 2) // "dd"
```

This API is only available on macOS 10.14 Mojave or newer.

`systemPreferences.getColor(color)` *Windows macOS*

- `color` `string` - One of the following values:
 - On **Windows**:
 - `3d-dark-shadow` - Dark shadow for three-dimensional display elements.
 - `3d-face` - Face color for three-dimensional display elements and for dialog box backgrounds.
 - `3d-highlight` - Highlight color for three-dimensional display elements.
 - `3d-light` - Light color for three-dimensional display elements.
 - `3d-shadow` - Shadow color for three-dimensional display elements.
 - `active-border` - Active window border.
 - `active-caption` - Active window title bar. Specifies the left side color in the color gradient of an active window's title bar if the gradient effect is enabled.
 - `active-caption-gradient` - Right side color in the color gradient of an active window's title bar.
 - `app-workspace` - Background color of multiple document interface (MDI) applications.
 - `button-text` - Text on push buttons.
 - `caption-text` - Text in caption, size box, and scroll bar arrow box.
 - `desktop` - Desktop background color.
 - `disabled-text` - Grayed (disabled) text.
 - `highlight` - Item(s) selected in a control.
 - `highlight-text` - Text of item(s) selected in a control.
 - `hotlight` - Color for a hyperlink or hot-tracked item.
 - `inactive-border` - Inactive window border.

- `inactive-caption` - Inactive window caption. Specifies the left side color in the color gradient of an inactive window's title bar if the gradient effect is enabled.
- `inactive-caption-gradient` - Right side color in the color gradient of an inactive window's title bar.
- `inactive-caption-text` - Color of text in an inactive caption.
- `info-background` - Background color for tooltip controls.
- `info-text` - Text color for tooltip controls.
- `menu` - Menu background.
- `menu-highlight` - The color used to highlight menu items when the menu appears as a flat menu.
- `menubar` - The background color for the menu bar when menus appear as flat menus.
- `menu-text` - Text in menus.
- `scrollbar` - Scroll bar gray area.
- `window` - Window background.
- `window-frame` - Window frame.
- `window-text` - Text in windows.

◦ On **macOS**

- `alternate-selected-control-text` - The text on a selected surface in a list or table. *deprecated*
- `control-background` - The background of a large interface element, such as a browser or table.
- `control` - The surface of a control.
- `control-text` - The text of a control that isn't disabled.
- `disabled-control-text` - The text of a control that's disabled.
- `find-highlight` - The color of a find indicator.
- `grid` - The gridlines of an interface element such as a table.
- `header-text` - The text of a header cell in a table.
- `highlight` - The virtual light source onscreen.
- `keyboard-focus-indicator` - The ring that appears around the currently focused control when using the keyboard for interface navigation.
- `label` - The text of a label containing primary content.
- `link` - A link to other content.
- `placeholder-text` - A placeholder string in a control or text view.
- `quaternary-label` - The text of a label of lesser importance than a tertiary label such as watermark text.
- `scrubber-textured-background` - The background of a scrubber in the Touch Bar.
- `secondary-label` - The text of a label of lesser importance than a normal label such as a label used to represent a subheading or additional information.
- `selected-content-background` - The background for selected content in a key window or view.
- `selected-control` - The surface of a selected control.
- `selected-control-text` - The text of a selected control.
- `selected-menu-item-text` - The text of a selected menu.
- `selected-text-background` - The background of selected text.
- `selected-text` - Selected text.
- `separator` - A separator between different sections of content.
- `shadow` - The virtual shadow cast by a raised object onscreen.

- `tertiary-label` - The text of a label of lesser importance than a secondary label such as a label used to represent disabled text.
- `text-background` - Text background.
- `text` - The text in a document.
- `under-page-background` - The background behind a document's content.
- `unemphasized-selected-content-background` - The selected content in a non-key window or view.
- `unemphasized-selected-text-background` - A background for selected text in a non-key window or view.
- `unemphasized-selected-text` - Selected text in a non-key window or view.
- `window-background` - The background of a window.
- `window-frame-text` - The text in the window's titlebar area.

Returns `string` - The system color setting in RGB hexadecimal form (`#ABCDEF`). See the [Windows docs](#) and the [macOS docs](#) for more details.

The following colors are only available on macOS 10.14: `find-highlight` , `selected-content-background` , `separator` , `unemphasized-selected-content-background` , `unemphasized-selected-text-background` , and `unemphasized-selected-text` .

`systemPreferences.getSystemColor(color)` *macOS*

- `color` `string` - One of the following values:
 - `blue`
 - `brown`
 - `gray`
 - `green`
 - `orange`
 - `pink`
 - `purple`
 - `red`
 - `yellow`

Returns `string` - The standard system color formatted as `#RRGGBBAA` .

Returns one of several standard system colors that automatically adapt to vibrancy and changes in accessibility settings like 'Increase contrast' and 'Reduce transparency'. See [Apple Documentation](#) for more details.

`systemPreferences.isInvertedColorScheme()` *Windows Deprecated*

Returns `boolean` - `true` if an inverted color scheme (a high contrast color scheme with light text and dark backgrounds) is active, `false` otherwise.

Deprecated: Should use the new [nativeTheme.shouldUseInvertedColorScheme](#) API.

`systemPreferences.isHighContrastColorScheme()` *macOS Windows Deprecated*

Returns `boolean` - `true` if a high contrast theme is active, `false` otherwise.

Deprecated: Should use the new [nativeTheme.shouldUseHighContrastColors](#) API.

systemPreferences.getEffectiveAppearance() *macOS*

Returns `string` - Can be `dark`, `light` or `unknown`.

Gets the macOS appearance setting that is currently applied to your application, maps to [NSApplication.effectiveAppearance](#)

systemPreferences.getAppLevelAppearance() *macOS Deprecated*

Returns `string` | `null` - Can be `dark`, `light` or `unknown`.

Gets the macOS appearance setting that you have declared you want for your application, maps to [NSApplication.appearance](#). You can use the `setAppLevelAppearance` API to set this value.

systemPreferences.setAppLevelAppearance(appearance) *macOS Deprecated*

- `appearance` `string` | `null` - Can be `dark` or `light`

Sets the appearance setting for your application, this should override the system default and override the value of `getEffectiveAppearance`.

systemPreferences.canPromptTouchID() *macOS*

Returns `boolean` - whether or not this device has the ability to use Touch ID.

NOTE: This API will return `false` on macOS systems older than Sierra 10.12.2.

systemPreferences.promptTouchID(reason) *macOS*

- `reason` `string` - The reason you are asking for Touch ID authentication

Returns `Promise<void>` - resolves if the user has successfully authenticated with Touch ID.

```
const { systemPreferences } = require('electron')

systemPreferences.promptTouchID('To get consent for a Security-Gated
Thing').then(success => {
  console.log('You have successfully authenticated with Touch ID!')
}).catch(err => {
  console.log(err)
})
```

This API itself will not protect your user data; rather, it is a mechanism to allow you to do so. Native apps will need to set [Access Control Constants](#) like `kSecAccessControlUserPresence` on their keychain entry so that reading it would auto-prompt for Touch ID biometric consent. This could be done with [node-keytar](#), such that one would store an encryption key with `node-keytar` and only fetch it if `promptTouchID()` resolves.

NOTE: This API will return a rejected Promise on macOS systems older than Sierra 10.12.2.

systemPreferences.isTrustedAccessibilityClient(prompt) *macOS*

- `prompt` `boolean` - whether or not the user will be informed via prompt if the current process is untrusted.

Returns `boolean` - `true` if the current process is a trusted accessibility client and `false` if it is not.

`systemPreferences.getMediaAccessStatus(mediaType)` *Windows macOS*

- `mediaType` string - Can be `microphone`, `camera` or `screen`.

Returns `string` - Can be `not-determined`, `granted`, `denied`, `restricted` or `unknown`.

This user consent was not required on macOS 10.13 High Sierra or lower so this method will always return `granted`. macOS 10.14 Mojave or higher requires consent for `microphone` and `camera` access. macOS 10.15 Catalina or higher requires consent for `screen` access.

Windows 10 has a global setting controlling `microphone` and `camera` access for all win32 applications. It will always return `granted` for `screen` and for all media types on older versions of Windows.

`systemPreferences.askForMediaAccess(mediaType)` *macOS*

- `mediaType` string - the type of media being requested; can be `microphone`, `camera`.

Returns `Promise<boolean>` - A promise that resolves with `true` if consent was granted and `false` if it was denied. If an invalid `mediaType` is passed, the promise will be rejected. If an access request was denied and later is changed through the System Preferences pane, a restart of the app will be required for the new permissions to take effect. If access has already been requested and denied, it *must* be changed through the preference pane; an alert will not pop up and the promise will resolve with the existing access status.

Important: In order to properly leverage this API, you [must set](#) the `NSMicrophoneUsageDescription` and `NSCameraUsageDescription` strings in your app's `Info.plist` file. The values for these keys will be used to populate the permission dialogs so that the user will be properly informed as to the purpose of the permission request. See [Electron Application Distribution](#) for more information about how to set these in the context of Electron.

This user consent was not required until macOS 10.14 Mojave, so this method will always return `true` if your system is running 10.13 High Sierra or lower.

`systemPreferences.getAnimationSettings()`

Returns `Object` :

- `shouldRenderRichAnimation` boolean - Returns true if rich animations should be rendered. Looks at session type (e.g. remote desktop) and accessibility settings to give guidance for heavy animations.
- `scrollAnimationsEnabledBySystem` boolean - Determines on a per-platform basis whether scroll animations (e.g. produced by home/end key) should be enabled.
- `prefersReducedMotion` boolean - Determines whether the user desires reduced motion based on platform APIs.

Returns an object with system animation settings.

Properties

`systemPreferences.appLevelAppearance` *macOS*

A `string` property that can be `dark`, `light` or `unknown`. It determines the macOS appearance setting for your application. This maps to values in: [NSApplication.appearance](#). Setting this will override the system default as well as the value of `getEffectiveAppearance`.

Possible values that can be set are `dark` and `light`, and possible return values are `dark`, `light`, and `unknown`.

This property is only available on macOS 10.14 Mojave or newer.

`systemPreferences.effectiveAppearance` **macOS Readonly**

A `string` property that can be `dark`, `light` or `unknown`.

Returns the macOS appearance setting that is currently applied to your application, maps to [NSApplication.effectiveAppearance](#)