app

> *Control your application's event lifecycle.*

Process: Main

The following example shows how to quit the application when the last window is closed:

```
const { app } = require('electron')
app.on('window-all-closed', () => {
  app.quit()
})
```

Events

The `app` object emits the following events:

Event: 'will-finish-launching'

Emitted when the application has finished basic startup. On Windows and Linux, the `will-finish-launching` event is the same as the `ready` event; on macOS, this event represents the `applicationWillFinishLaunching` notification of `NSApplication` . You would usually set up listeners for the `open-file` and `open-url` events here, and start the crash reporter and auto updater.

In most cases, you should do everything in the `ready` event handler.

Event: 'ready'

Returns:

- `event` Event
- `launchInfo` Record<string, any> | NotificationResponse *macOS*

Emitted once, when Electron has finished initializing. On macOS, `launchInfo` holds the `userInfo` of the `NSUserNotification` or information from `UNNotificationResponse` that was used to open the application, if it was launched from Notification Center. You can also call `app.isReady()` to check if this event has already fired and `app.whenReady()` to get a Promise that is fulfilled when Electron is initialized.

Event: 'window-all-closed'

Emitted when all windows have been closed.

If you do not subscribe to this event and all windows are closed, the default behavior is to quit the app; however, if you subscribe, you control whether the app quits or not. If the user pressed `Cmd + Q` , or the developer called `app.quit()` , Electron will first try to close all the windows and then emit the `will-quit` event, and in this case the `window-all-closed` event would not be emitted.

Event: 'before-quit'

Returns:

- `event` Event

Emitted before the application starts closing its windows. Calling `event.preventDefault()` will prevent the default behavior, which is terminating the application.

Note: If application quit was initiated by `autoUpdater.quitAndInstall()`, then `before-quit` is emitted *after* emitting `close` event on all windows and closing them.

Note: On Windows, this event will not be emitted if the app is closed due to a shutdown/restart of the system or a user logout.

Event: 'will-quit'

Returns:

- `event` Event

Emitted when all windows have been closed and the application will quit. Calling `event.preventDefault()` will prevent the default behavior, which is terminating the application.

See the description of the `window-all-closed` event for the differences between the `will-quit` and `window-all-closed` events.

Note: On Windows, this event will not be emitted if the app is closed due to a shutdown/restart of the system or a user logout.

Event: 'quit'

Returns:

- `event` Event
- `exitCode` Integer

Emitted when the application is quitting.

Note: On Windows, this event will not be emitted if the app is closed due to a shutdown/restart of the system or a user logout.

Event: 'open-file' *macOS*

Returns:

- `event` Event
- `path` string

Emitted when the user wants to open a file with the application. The `open-file` event is usually emitted when the application is already open and the OS wants to reuse the application to open the file. `open-file` is also emitted when a file is dropped onto the dock and the application is not yet running. Make sure to listen for the `open-file` event very early in your application startup to handle this case (even before the `ready` event is emitted).

You should call `event.preventDefault()` if you want to handle this event.

On Windows, you have to parse `process.argv` (in the main process) to get the filepath.

Event: 'open-url' *macOS*

Returns:

- `event` Event

- `url` string

Emitted when the user wants to open a URL with the application. Your application's `Info.plist` file must define the URL scheme within the `CFBundleURLTypes` key, and set `NSPrincipalClass` to `AtomApplication`.

You should call `event.preventDefault()` if you want to handle this event.

Event: 'activate' *macOS*

Returns:

- `event` Event
- `hasVisibleWindows` boolean

Emitted when the application is activated. Various actions can trigger this event, such as launching the application for the first time, attempting to re-launch the application when it's already running, or clicking on the application's dock or taskbar icon.

Event: 'did-become-active' *macOS*

Returns:

- `event` Event

Emitted when mac application become active. Difference from `activate` event is that `did-become-active` is emitted every time the app becomes active, not only when Dock icon is clicked or application is re-launched.

Event: 'continue-activity' *macOS*

Returns:

- `event` Event
- `type` string - A string identifying the activity. Maps to [`NSUserActivity.activityType`](#).
- `userInfo` unknown - Contains app-specific state stored by the activity on another device.
- `details` Object
  - `webpageURL` string (optional) - A string identifying the URL of the webpage accessed by the activity on another device, if available.

Emitted during [Handoff](#) when an activity from a different device wants to be resumed. You should call `event.preventDefault()` if you want to handle this event.

A user activity can be continued only in an app that has the same developer Team ID as the activity's source app and that supports the activity's type. Supported activity types are specified in the app's `Info.plist` under the `NSUserActivityTypes` key.

Event: 'will-continue-activity' *macOS*

Returns:

- `event` Event
- `type` string - A string identifying the activity. Maps to [`NSUserActivity.activityType`](#).

Emitted during [Handoff](#) before an activity from a different device wants to be resumed. You should call `event.preventDefault()` if you want to handle this event.

Event: 'continue-activity-error' *macOS*

Returns:

- `event` Event
- `type` string - A string identifying the activity. Maps to [`NSUserActivity.activityType`]().
- `error` string - A string with the error's localized description.

Emitted during [Handoff]() when an activity from a different device fails to be resumed.

Event: 'activity-was-continued' *macOS*

Returns:

- `event` Event
- `type` string - A string identifying the activity. Maps to [`NSUserActivity.activityType`]().
- `userInfo` unknown - Contains app-specific state stored by the activity.

Emitted during [Handoff]() after an activity from this device was successfully resumed on another one.

Event: 'update-activity-state' *macOS*

Returns:

- `event` Event
- `type` string - A string identifying the activity. Maps to [`NSUserActivity.activityType`]().
- `userInfo` unknown - Contains app-specific state stored by the activity.

Emitted when [Handoff]() is about to be resumed on another device. If you need to update the state to be transferred, you should call `event.preventDefault()` immediately, construct a new `userInfo` dictionary and call `app.updateCurrentActivity()` in a timely manner. Otherwise, the operation will fail and `continue-activity-error` will be called.

Event: 'new-window-for-tab' *macOS*

Returns:

- `event` Event

Emitted when the user clicks the native macOS new tab button. The new tab button is only visible if the current `BrowserWindow` has a `tabbingIdentifier`

Event: 'browser-window-blur'

Returns:

- `event` Event
- `window` [BrowserWindow]()

Emitted when a [browserWindow]() gets blurred.

Event: 'browser-window-focus'

Returns:

- `event` Event

- `window` BrowserWindow

Emitted when a browserWindow gets focused.

Event: 'browser-window-created'

Returns:

- `event` Event
- `window` BrowserWindow

Emitted when a new browserWindow is created.

Event: 'web-contents-created'

Returns:

- `event` Event
- `webContents` WebContents

Emitted when a new webContents is created.

Event: 'certificate-error'

Returns:

- `event` Event
- `webContents` WebContents
- `url` string
- `error` string - The error code
- `certificate` Certificate
- `callback` Function
  - `isTrusted` boolean - Whether to consider the certificate as trusted
- `isMainFrame` boolean

Emitted when failed to verify the `certificate` for `url`, to trust the certificate you should prevent the default behavior with `event.preventDefault()` and call `callback(true)`.

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

app.on('certificate-error', (event, webContents, url, error, certificate, callback)
=> {
  if (url === 'https://github.com') {
    // Verification logic.
    event.preventDefault()
    callback(true)
  } else {
    callback(false)
  }
})
```

Event: 'select-client-certificate'

Returns:

- `event` Event
- `webContents` WebContents
- `url` URL
- `certificateList` Certificate[]
- `callback` Function
  - `certificate` Certificate (optional)

Emitted when a client certificate is requested.

The `url` corresponds to the navigation entry requesting the client certificate and `callback` can be called with an entry filtered from the list. Using `event.preventDefault()` prevents the application from using the first certificate from the store.

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

app.on('select-client-certificate', (event, webContents, url, list, callback) => {
  event.preventDefault()
  callback(list[0])
})
```

Event: 'login'

Returns:

- `event` Event
- `webContents` WebContents
- `authenticationResponseDetails` Object
  - `url` URL
- `authInfo` Object
  - `isProxy` boolean
  - `scheme` string
  - `host` string
  - `port` Integer
  - `realm` string
- `callback` Function
  - `username` string (optional)
  - `password` string (optional)

Emitted when `webContents` wants to do basic auth.

The default behavior is to cancel all authentications. To override this you should prevent the default behavior with `event.preventDefault()` and call `callback(username, password)` with the credentials.

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

app.on('login', (event, webContents, details, authInfo, callback) => {
  event.preventDefault()
  callback('username', 'secret')
})
```

If `callback` is called without a username or password, the authentication request will be cancelled and the authentication error will be returned to the page.

Event: 'gpu-info-update'

Emitted whenever there is a GPU info update.

Event: 'gpu-process-crashed' *Deprecated*

Returns:

- `event` Event
- `killed` boolean

Emitted when the GPU process crashes or is killed.

Deprecated: This event is superceded by the `child-process-gone` event which contains more information about why the child process disappeared. It isn't always because it crashed. The `killed` boolean can be replaced by checking `reason === 'killed'` when you switch to that event.

Event: 'renderer-process-crashed' *Deprecated*

Returns:

- `event` Event
- `webContents` WebContents
- `killed` boolean

Emitted when the renderer process of `webContents` crashes or is killed.

Deprecated: This event is superceded by the `render-process-gone` event which contains more information about why the render process disappeared. It isn't always because it crashed. The `killed` boolean can be replaced by checking `reason === 'killed'` when you switch to that event.

Event: 'render-process-gone'

Returns:

- `event` Event
- `webContents` WebContents
- `details` Object
  - `reason` string - The reason the render process is gone. Possible values:
    - `clean-exit` - Process exited with an exit code of zero
    - `abnormal-exit` - Process exited with a non-zero exit code
    - `killed` - Process was sent a SIGTERM or otherwise killed externally
    - `crashed` - Process crashed
    - `oom` - Process ran out of memory
    - `launch-failed` - Process never successfully launched
    - `integrity-failure` - Windows code integrity checks failed
  - `exitCode` Integer - The exit code of the process, unless `reason` is `launch-failed`, in which case `exitCode` will be a platform-specific launch failure error code.

Emitted when the renderer process unexpectedly disappears. This is normally because it was crashed or killed.

Event: 'child-process-gone'

Returns:

- `event` Event
- `details` Object
  - `type` string - Process type. One of the following values:
    - `Utility`
    - `Zygote`
    - `Sandbox helper`
    - `GPU`
    - `Pepper Plugin`
    - `Pepper Plugin Broker`
    - `Unknown`
  - `reason` string - The reason the child process is gone. Possible values:
    - `clean-exit` - Process exited with an exit code of zero
    - `abnormal-exit` - Process exited with a non-zero exit code
    - `killed` - Process was sent a SIGTERM or otherwise killed externally
    - `crashed` - Process crashed
    - `oom` - Process ran out of memory
    - `launch-failed` - Process never successfully launched
    - `integrity-failure` - Windows code integrity checks failed
  - `exitCode` number - The exit code for the process (e.g. status from waitpid if on posix, from GetExitCodeProcess on Windows).
  - `serviceName` string (optional) - The non-localized name of the process.
  - `name` string (optional) - The name of the process. Examples for utility: `Audio Service`, `Content Decryption Module Service`, `Network Service`, `Video Capture`, etc.

Emitted when the child process unexpectedly disappears. This is normally because it was crashed or killed. It does not include renderer processes.

Event: 'accessibility-support-changed' *macOS Windows*

Returns:

- `event` Event
- `accessibilitySupportEnabled` boolean - `true` when Chrome's accessibility support is enabled, `false` otherwise.

Emitted when Chrome's accessibility support changes. This event fires when assistive technologies, such as screen readers, are enabled or disabled. See https://www.chromium.org/developers/design-documents/accessibility for more details.

Event: 'session-created'

Returns:

- `session` Session

Emitted when Electron has created a new `session`.

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

app.on('session-created', (session) => {
  console.log(session)
})
```

Event: 'second-instance'

Returns:

- `event` Event
- `argv` string[] - An array of the second instance's command line arguments
- `workingDirectory` string - The second instance's working directory
- `additionalData` unknown - A JSON object of additional data passed from the second instance
- `ackCallback` unknown - A function that can be used to send data back to the second instance

This event will be emitted inside the primary instance of your application when a second instance has been executed and calls `app.requestSingleInstanceLock()`.

`argv` is an Array of the second instance's command line arguments, and `workingDirectory` is its current working directory. Usually applications respond to this by making their primary window focused and non-minimized.

Note: If the second instance is started by a different user than the first, the `argv` array will not include the arguments.

Note: `ackCallback` allows the user to send data back to the second instance during the `app.requestSingleInstanceLock()` flow. This callback can be used for cases where the second instance needs to obtain additional information from the first instance before quitting.

Currently, the limit on the message size is kMaxMessageLength, or around 32kB. To be safe, keep the amount of data passed to 31kB at most.

In order to call the callback, `event.preventDefault()` must be called, first. If the callback is not called in either case, `null` will be sent back. If `event.preventDefault()` is not called, but `ackCallback` is called by the user in the event, then the behaviour is undefined.

This event is guaranteed to be emitted after the `ready` event of `app` gets emitted.

Note: Extra command line arguments might be added by Chromium, such as `--original-process-start-time`.

Event: 'first-instance-ack'

Returns:

- `event` Event
- `additionalData` unknown - A JSON object of additional data passed from the first instance, in response to the first instance's `second-instance` event.

This event will be emitted within the second instance during the call to `app.requestSingleInstanceLock()`, when the first instance calls the `ackCallback` provided by the `second-instance` event handler.

Methods

The `app` object has the following methods:

Note: Some methods are only available on specific operating systems and are labeled as such.

`app.quit()`

Try to close all windows. The `before-quit` event will be emitted first. If all windows are successfully closed, the `will-quit` event will be emitted and by default the application will terminate.

This method guarantees that all `beforeunload` and `unload` event handlers are correctly executed. It is possible that a window cancels the quitting by returning `false` in the `beforeunload` event handler.

`app.exit([exitCode])`

- `exitCode` Integer (optional)

Exits immediately with `exitCode`. `exitCode` defaults to 0.

All windows will be closed immediately without asking the user, and the `before-quit` and `will-quit` events will not be emitted.

`app.relaunch([options])`

- `options` Object (optional)
  - `args` string[] (optional)
  - `execPath` string (optional)

Relaunches the app when current instance exits.

By default, the new instance will use the same working directory and command line arguments with current instance. When `args` is specified, the `args` will be passed as command line arguments instead. When `execPath` is specified, the `execPath` will be executed for relaunch instead of current app.

Note that this method does not quit the app when executed, you have to call `app.quit` or `app.exit` after calling `app.relaunch` to make the app restart.

When `app.relaunch` is called for multiple times, multiple instances will be started after current instance exited.

An example of restarting current instance immediately and adding a new command line argument to the new instance:

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

app.relaunch({ args: process.argv.slice(1).concat(['--relaunch']) })
app.exit(0)
```

`app.isReady()`

Returns `boolean` - `true` if Electron has finished initializing, `false` otherwise. See also `app.whenReady()`.

`app.whenReady()`

Returns `Promise<void>` - fulfilled when Electron is initialized. May be used as a convenient alternative to checking `app.isReady()` and subscribing to the `ready` event if the app is not ready yet.

`app.focus([options])`

- `options` Object (optional)
  - `steal` boolean *macOS* - Make the receiver the active app even if another app is currently active.

On Linux, focuses on the first visible window. On macOS, makes the application the active app. On Windows, focuses on the application's first window.

You should seek to use the `steal` option as sparingly as possible.

`app.hide()` *macOS*

Hides all application windows without minimizing them.

`app.isHidden()` *macOS*

Returns `boolean` - `true` if the application—including all of its windows—is hidden (e.g. with `Command-H`), `false` otherwise.

`app.show()` *macOS*

Shows application windows after they were hidden. Does not automatically focus them.

`app.setAppLogsPath([path])`

- `path` string (optional) - A custom path for your logs. Must be absolute.

Sets or creates a directory your app's logs which can then be manipulated with `app.getPath()` or `app.setPath(pathName, newPath)`.

Calling `app.setAppLogsPath()` without a `path` parameter will result in this directory being set to `~/Library/Logs/YourAppName` on *macOS*, and inside the `userData` directory on *Linux* and *Windows*.

`app.getAppPath()`

Returns `string` - The current application directory.

`app.getPath(name)`

- `name` string - You can request the following paths by the name:
  - `home` User's home directory.
  - `appData` Per-user application data directory, which by default points to:
    - `%APPDATA%` on Windows
    - `$XDG_CONFIG_HOME` or `~/.config` on Linux
    - `~/Library/Application Support` on macOS
  - `userData` The directory for storing your app's configuration files, which by default it is the `appData` directory appended with your app's name.
  - `temp` Temporary directory.
  - `exe` The current executable file.
  - `module` The `libchromiumcontent` library.

  - `desktop` The current user's Desktop directory.
  - `documents` Directory for a user's "My Documents".
  - `downloads` Directory for a user's downloads.
  - `music` Directory for a user's music.
  - `pictures` Directory for a user's pictures.
  - `videos` Directory for a user's videos.
  - `recent` Directory for the user's recent files (Windows only).
  - `logs` Directory for your app's log folder.
  - `crashDumps` Directory where crash dumps are stored.

Returns `string` - A path to a special directory or file associated with `name`. On failure, an `Error` is thrown.

If `app.getPath('logs')` is called without called `app.setAppLogsPath()` being called first, a default log directory will be created equivalent to calling `app.setAppLogsPath()` without a `path` parameter.

`app.getFileIcon(path[, options])`

- `path` string
- `options` Object (optional)
  - `size` string
    - `small` - 16x16
    - `normal` - 32x32
    - `large` - 48x48 on *Linux*, 32x32 on *Windows*, unsupported on *macOS*.

Returns `Promise<NativeImage>` - fulfilled with the app's icon, which is a NativeImage.

Fetches a path's associated icon.

On *Windows*, there a 2 kinds of icons:

- Icons associated with certain file extensions, like `.mp3`, `.png`, etc.
- Icons inside the file itself, like `.exe`, `.dll`, `.ico`.

On *Linux* and *macOS*, icons depend on the application associated with file mime type.

`app.setPath(name, path)`

- `name` string
- `path` string

Overrides the `path` to a special directory or file associated with `name`. If the path specifies a directory that does not exist, an `Error` is thrown. In that case, the directory should be created with `fs.mkdirSync` or similar.

You can only override paths of a `name` defined in `app.getPath`.

By default, web pages' cookies and caches will be stored under the `userData` directory. If you want to change this location, you have to override the `userData` path before the `ready` event of the `app` module is emitted.

`app.getVersion()`

Returns `string` - The version of the loaded application. If no version is found in the application's `package.json` file, the version of the current bundle or executable is returned.

`app.getName()`

Returns `string` - The current application's name, which is the name in the application's `package.json` file.

Usually the `name` field of `package.json` is a short lowercase name, according to the npm modules spec. You should usually also specify a `productName` field, which is your application's full capitalized name, and which will be preferred over `name` by Electron.

`app.setName(name)`

- `name` string

Overrides the current application's name.

Note: This function overrides the name used internally by Electron; it does not affect the name that the OS uses.

`app.getLocale()`

Returns `string` - The current application locale, fetched using Chromium's `l10n_util` library. Possible return values are documented here.

To set the locale, you'll want to use a command line switch at app startup, which may be found here.

Note: When distributing your packaged app, you have to also ship the `locales` folder.

Note: On Windows, you have to call it after the `ready` events gets emitted.

`app.getLocaleCountryCode()`

Returns `string` - User operating system's locale two-letter ISO 3166 country code. The value is taken from native OS APIs.

Note: When unable to detect locale country code, it returns empty string.

`app.addRecentDocument(path)` *macOS Windows*

- `path` string

Adds `path` to the recent documents list.

This list is managed by the OS. On Windows, you can visit the list from the task bar, and on macOS, you can visit it from dock menu.

`app.clearRecentDocuments()` *macOS Windows*

Clears the recent documents list.

`app.setAsDefaultProtocolClient(protocol[, path, args])`

- `protocol` string - The name of your protocol, without `://`. For example, if you want your app to handle `electron://` links, call this method with `electron` as the parameter.
- `path` string (optional) *Windows* - The path to the Electron executable. Defaults to `process.execPath`
- `args` string[] (optional) *Windows* - Arguments passed to the executable. Defaults to an empty array

Returns `boolean` - Whether the call succeeded.

Sets the current executable as the default handler for a protocol (aka URI scheme). It allows you to integrate your app deeper into the operating system. Once registered, all links with `your-protocol://` will be opened with the current executable. The whole link, including protocol, will be passed to your application as a parameter.

Note: On macOS, you can only register protocols that have been added to your app's `info.plist`, which cannot be modified at runtime. However, you can change the file during build time via Electron Forge, Electron Packager, or by editing `info.plist` with a text editor. Please refer to Apple's documentation for details.

Note: In a Windows Store environment (when packaged as an `appx`) this API will return `true` for all calls but the registry key it sets won't be accessible by other applications. In order to register your Windows Store application as a default protocol handler you must declare the protocol in your manifest.

The API uses the Windows Registry and `LSSetDefaultHandlerForURLScheme` internally.

`app.removeAsDefaultProtocolClient(protocol[, path, args])` *macOS Windows*

- `protocol` string - The name of your protocol, without `://`.
- `path` string (optional) *Windows* - Defaults to `process.execPath`
- `args` string[] (optional) *Windows* - Defaults to an empty array

Returns `boolean` - Whether the call succeeded.

This method checks if the current executable as the default handler for a protocol (aka URI scheme). If so, it will remove the app as the default handler.

`app.isDefaultProtocolClient(protocol[, path, args])`

- `protocol` string - The name of your protocol, without `://`.
- `path` string (optional) *Windows* - Defaults to `process.execPath`
- `args` string[] (optional) *Windows* - Defaults to an empty array

Returns `boolean` - Whether the current executable is the default handler for a protocol (aka URI scheme).

Note: On macOS, you can use this method to check if the app has been registered as the default protocol handler for a protocol. You can also verify this by checking `~/Library/Preferences/com.apple.LaunchServices.plist` on the macOS machine. Please refer to Apple's documentation for details.

The API uses the Windows Registry and `LSCopyDefaultHandlerForURLScheme` internally.

`app.getApplicationNameForProtocol(url)`

- `url` string - a URL with the protocol name to check. Unlike the other methods in this family, this accepts an entire URL, including `://` at a minimum (e.g. `https://`).

Returns `string` - Name of the application handling the protocol, or an empty string if there is no handler. For instance, if Electron is the default handler of the URL, this could be `Electron` on Windows and Mac. However, don't rely on the precise format which is not guaranteed to remain unchanged. Expect a different format on Linux, possibly with a `.desktop` suffix.

This method returns the application name of the default handler for the protocol (aka URI scheme) of a URL.

`app.getApplicationInfoForProtocol(url)` *macOS Windows*

- `url` string - a URL with the protocol name to check. Unlike the other methods in this family, this accepts an entire URL, including `://` at a minimum (e.g. `https://`).

Returns `Promise<Object>` - Resolve with an object containing the following:

- `icon` NativeImage - the display icon of the app handling the protocol.
- `path` string - installation path of the app handling the protocol.
- `name` string - display name of the app handling the protocol.

This method returns a promise that contains the application name, icon and path of the default handler for the protocol (aka URI scheme) of a URL.

`app.setUserTasks(tasks)` *Windows*

- `tasks` Task[] - Array of `Task` objects

Adds `tasks` to the Tasks category of the Jump List on Windows.

`tasks` is an array of `Task` objects.

Returns `boolean` - Whether the call succeeded.

Note: If you'd like to customize the Jump List even more use `app.setJumpList(categories)` instead.

`app.getJumpListSettings()` *Windows*

Returns `Object`:

- `minItems` Integer - The minimum number of items that will be shown in the Jump List (for a more detailed description of this value see the MSDN docs).
- `removedItems` JumpListItem[] - Array of `JumpListItem` objects that correspond to items that the user has explicitly removed from custom categories in the Jump List. These items must not be re-added to the Jump List in the **next** call to `app.setJumpList()`, Windows will not display any custom category that contains any of the removed items.

`app.setJumpList(categories)` *Windows*

- `categories` JumpListCategory[] | `null` - Array of `JumpListCategory` objects.

Sets or removes a custom Jump List for the application, and returns one of the following strings:

- `ok` - Nothing went wrong.
- `error` - One or more errors occurred, enable runtime logging to figure out the likely cause.
- `invalidSeparatorError` - An attempt was made to add a separator to a custom category in the Jump List. Separators are only allowed in the standard `Tasks` category.
- `fileTypeRegistrationError` - An attempt was made to add a file link to the Jump List for a file type the app isn't registered to handle.
- `customCategoryAccessDeniedError` - Custom categories can't be added to the Jump List due to user privacy or group policy settings.

If `categories` is `null` the previously set custom Jump List (if any) will be replaced by the standard Jump List for the app (managed by Windows).

Note: If a `JumpListCategory` object has neither the `type` nor the `name` property set then its `type` is assumed to be `tasks`. If the `name` property is set but the `type` property is omitted then the `type` is assumed to be `custom`.

Note: Users can remove items from custom categories, and Windows will not allow a removed item to be added back into a custom category until **after** the next successful call to `app.setJumpList(categories)`. Any attempt to re-add a removed item to a custom category earlier than that will result in the entire custom category being omitted from the Jump List. The list of removed items can be obtained using `app.getJumpListSettings()`.

Note: The maximum length of a Jump List item's `description` property is 260 characters. Beyond this limit, the item will not be added to the Jump List, nor will it be displayed.

Here's a very simple example of creating a custom Jump List:

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

app.setJumpList([
  {
    type: 'custom',
    name: 'Recent Projects',
    items: [
      { type: 'file', path: 'C:\\Projects\\project1.proj' },
      { type: 'file', path: 'C:\\Projects\\project2.proj' }
    ]
  },
  { // has a name so `type` is assumed to be "custom"
    name: 'Tools',
    items: [
      {
        type: 'task',
        title: 'Tool A',
        program: process.execPath,
        args: '--run-tool-a',
        icon: process.execPath,
        iconIndex: 0,
        description: 'Runs Tool A'
      },
      {
        type: 'task',
        title: 'Tool B',
        program: process.execPath,
        args: '--run-tool-b',
        icon: process.execPath,
        iconIndex: 0,
        description: 'Runs Tool B'
      }
    ]
  },
  { type: 'frequent' },
  { // has no name and no type so `type` is assumed to be "tasks"
    items: [
      {
        type: 'task',
        title: 'New Project',
        program: process.execPath,
        args: '--new-project',
        description: 'Create a new project.'
      },
      { type: 'separator' },
      {
        type: 'task',
        title: 'Recover Project',
        program: process.execPath,
        args: '--recover-project',
        description: 'Recover Project'
      }
    ]
  }
])
```

`app.requestSingleInstanceLock([additionalData])`

- `additionalData` Record<any, any> (optional) - A JSON object containing additional data to send to the first instance.

Returns `boolean`

The return value of this method indicates whether or not this instance of your application successfully obtained the lock. If it failed to obtain the lock, you can assume that another instance of your application is already running with the lock and exit immediately.

I.e. This method returns `true` if your process is the primary instance of your application and your app should continue loading. It returns `false` if your process should immediately quit as it has sent its parameters to another instance that has already acquired the lock.

On macOS, the system enforces single instance automatically when users try to open a second instance of your app in Finder, and the `open-file` and `open-url` events will be emitted for that. However when users start your app in command line, the system's single instance mechanism will be bypassed, and you have to use this method to ensure single instance.

An example of activating the window of primary instance when a second instance starts:

```
const { app } = require('electron')
let myWindow = null

app.on('first-instance-ack', (event, additionalData) => {
  // Print out the ack received from the first instance.
  // Note this event handler must come before the requestSingleInstanceLock call.
  // Expected output: '{"myAckKey":"myAckValue"}'
  console.log(JSON.stringify(additionalData))
})

const additionalData = { myKey: 'myValue' }
const gotTheLock = app.requestSingleInstanceLock(additionalData)

if (!gotTheLock) {
  app.quit()
} else {
  app.on('second-instance', (event, commandLine, workingDirectory, additionalData)
=> {
    // We must call preventDefault if we're sending back data.
    event.preventDefault()
    // Print out data received from the second instance.
    // Expected output: '{"myKey":"myValue"}'
    console.log(JSON.stringify(additionalData))

    // Someone tried to run a second instance, we should focus our window.
    if (myWindow) {
      if (myWindow.isMinimized()) myWindow.restore()
      myWindow.focus()
    }
    const ackData = { myAckKey: 'myAckValue' }
    ackCallback(ackData)
  })

  // Create myWindow, load the rest of the app, etc...
  app.whenReady().then(() => {
    myWindow = createWindow()
  })
}
```

`app.hasSingleInstanceLock()`

Returns `boolean`

This method returns whether or not this instance of your app is currently holding the single instance lock. You can request the lock with `app.requestSingleInstanceLock()` and release with `app.releaseSingleInstanceLock()`

`app.releaseSingleInstanceLock()`

Releases all locks that were created by `requestSingleInstanceLock`. This will allow multiple instances of the application to once again run side by side.

`app.setUserActivity(type, userInfo[, webpageURL])` *macOS*

- `type` string - Uniquely identifies the activity. Maps to [`NSUserActivity.activityType`].
- `userInfo` any - App-specific state to store for use by another device.
- `webpageURL` string (optional) - The webpage to load in a browser if no suitable app is installed on the resuming device. The scheme must be `http` or `https`.

Creates an `NSUserActivity` and sets it as the current activity. The activity is eligible for [Handoff] to another device afterward.

`app.getCurrentActivityType()` *macOS*

Returns `string` - The type of the currently running activity.

`app.invalidateCurrentActivity()` *macOS*

Invalidates the current Handoff user activity.

`app.resignCurrentActivity()` *macOS*

Marks the current Handoff user activity as inactive without invalidating it.

`app.updateCurrentActivity(type, userInfo)` *macOS*

- `type` string - Uniquely identifies the activity. Maps to `NSUserActivity.activityType` .
- `userInfo` any - App-specific state to store for use by another device.

Updates the current activity if its type matches `type` , merging the entries from `userInfo` into its current `userInfo` dictionary.

`app.setAppUserModelId(id)` *Windows*

- `id` string

Changes the Application User Model ID to `id` .

`app.setActivationPolicy(policy)` *macOS*

- `policy` string - Can be 'regular', 'accessory', or 'prohibited'.

Sets the activation policy for a given app.

Activation policy types:

- 'regular' - The application is an ordinary app that appears in the Dock and may have a user interface.
- 'accessory' - The application doesn’t appear in the Dock and doesn’t have a menu bar, but it may be activated programmatically or by clicking on one of its windows.
- 'prohibited' - The application doesn’t appear in the Dock and may not create windows or be activated.

`app.importCertificate(options, callback)` *Linux*

- `options` Object
  - `certificate` string - Path for the pkcs12 file.
  - `password` string - Passphrase for the certificate.
- `callback` Function
  - `result` Integer - Result of import.

Imports the certificate in pkcs12 format into the platform certificate store. `callback` is called with the `result` of import operation, a value of `0` indicates success while any other value indicates failure according to Chromium net_error_list.

`app.configureHostResolver(options)`

- `options` Object
  - `enableBuiltInResolver` boolean (optional) - Whether the built-in host resolver is used in preference to getaddrinfo. When enabled, the built-in resolver will attempt to use the system's DNS settings to do DNS lookups itself. Enabled by default on macOS, disabled by default on Windows and Linux.

- `secureDnsMode` string (optional) - Can be "off", "automatic" or "secure". Configures the DNS-over-HTTP mode. When "off", no DoH lookups will be performed. When "automatic", DoH lookups will be performed first if DoH is available, and insecure DNS lookups will be performed as a fallback. When "secure", only DoH lookups will be performed. Defaults to "automatic".
- `secureDnsServers` string[] (optional) - A list of DNS-over-HTTP server templates. See RFC8484 § 3 for details on the template format. Most servers support the POST method; the template for such servers is simply a URI. Note that for some DNS providers, the resolver will automatically upgrade to DoH unless DoH is explicitly disabled, even if there are no DoH servers provided in this list.
- `enableAdditionalDnsQueryTypes` boolean (optional) - Controls whether additional DNS query types, e.g. HTTPS (DNS type 65) will be allowed besides the traditional A and AAAA queries when a request is being made via insecure DNS. Has no effect on Secure DNS which always allows additional types. Defaults to true.

Configures host resolution (DNS and DNS-over-HTTPS). By default, the following resolvers will be used, in order:

1. DNS-over-HTTPS, if the DNS provider supports it, then
2. the built-in resolver (enabled on macOS only by default), then
3. the system's resolver (e.g. `getaddrinfo`).

This can be configured to either restrict usage of non-encrypted DNS (`secureDnsMode: "secure"`), or disable DNS-over-HTTPS (`secureDnsMode: "off"`). It is also possible to enable or disable the built-in resolver.

To disable insecure DNS, you can specify a `secureDnsMode` of `"secure"` . If you do so, you should make sure to provide a list of DNS-over-HTTPS servers to use, in case the user's DNS configuration does not include a provider that supports DoH.

```
app.configureHostResolver({
  secureDnsMode: 'secure',
  secureDnsServers: [
    'https://cloudflare-dns.com/dns-query'
  ]
})
```

This API must be called after the `ready` event is emitted.

`app.disableHardwareAcceleration()`

Disables hardware acceleration for current app.

This method can only be called before app is ready.

`app.disableDomainBlockingFor3DAPIs()`

By default, Chromium disables 3D APIs (e.g. WebGL) until restart on a per domain basis if the GPU processes crashes too frequently. This function disables that behavior.

This method can only be called before app is ready.

`app.getAppMetrics()`

Returns `ProcessMetric[]` : Array of `ProcessMetric` objects that correspond to memory and CPU usage statistics of all the processes associated with the app.

`app.getGPUFeatureStatus()`

Returns [`GPUFeatureStatus`](#) - The Graphics Feature Status from `chrome://gpu/`.

Note: This information is only usable after the `gpu-info-update` event is emitted.

`app.getGPUInfo(infoType)`

- `infoType` string - Can be `basic` or `complete`.

Returns `Promise<unknown>`

For `infoType` equal to `complete`: Promise is fulfilled with `Object` containing all the GPU Information as in [chromium's GPUInfo object](#). This includes the version and driver information that's shown on `chrome://gpu` page.

For `infoType` equal to `basic`: Promise is fulfilled with `Object` containing fewer attributes than when requested with `complete`. Here's an example of basic response:

```
{
  auxAttributes:
   {
     amdSwitchable: true,
     canSupportThreadedTextureMailbox: false,
     directComposition: false,
     directRendering: true,
     glResetNotificationStrategy: 0,
     inProcessGpu: true,
     initializationTime: 0,
     jpegDecodeAcceleratorSupported: false,
     optimus: false,
     passthroughCmdDecoder: false,
     sandboxed: false,
     softwareRendering: false,
     supportsOverlays: false,
     videoDecodeAcceleratorFlags: 0
   },
  gpuDevice:
   [{ active: true, deviceId: 26657, vendorId: 4098 },
     { active: false, deviceId: 3366, vendorId: 32902 }],
  machineModelName: 'MacBookPro',
  machineModelVersion: '11.5'
}
```

Using `basic` should be preferred if only basic information like `vendorId` or `driverId` is needed.

`app.setBadgeCount([count])` *Linux macOS*

- `count` Integer (optional) - If a value is provided, set the badge to the provided value otherwise, on macOS, display a plain white dot (e.g. unknown number of notifications). On Linux, if a value is not provided the badge will not display.

Returns `boolean` - Whether the call succeeded.

Sets the counter badge for current app. Setting the count to `0` will hide the badge.

On macOS, it shows on the dock icon. On Linux, it only works for Unity launcher.

Note: Unity launcher requires a `.desktop` file to work. For more information, please read the [Unity integration documentation](#).

`app.getBadgeCount()` *Linux macOS*

Returns `Integer` - The current value displayed in the counter badge.

`app.isUnityRunning()` *Linux*

Returns `boolean` - Whether the current desktop environment is Unity launcher.

`app.getLoginItemSettings([options])` *macOS Windows*

- `options` Object (optional)
  - `path` string (optional) *Windows* - The executable path to compare against. Defaults to `process.execPath`.
  - `args` string[] (optional) *Windows* - The command-line arguments to compare against. Defaults to an empty array.

If you provided `path` and `args` options to `app.setLoginItemSettings`, then you need to pass the same arguments here for `openAtLogin` to be set correctly.

Returns `Object`:

- `openAtLogin` boolean - `true` if the app is set to open at login.
- `openAsHidden` boolean *macOS* - `true` if the app is set to open as hidden at login. This setting is not available on [MAS builds](#).
- `wasOpenedAtLogin` boolean *macOS* - `true` if the app was opened at login automatically. This setting is not available on [MAS builds](#).
- `wasOpenedAsHidden` boolean *macOS* - `true` if the app was opened as a hidden login item. This indicates that the app should not open any windows at startup. This setting is not available on [MAS builds](#).
- `restoreState` boolean *macOS* - `true` if the app was opened as a login item that should restore the state from the previous session. This indicates that the app should restore the windows that were open the last time the app was closed. This setting is not available on [MAS builds](#).
- `executableWillLaunchAtLogin` boolean *Windows* - `true` if app is set to open at login and its run key is not deactivated. This differs from `openAtLogin` as it ignores the `args` option, this property will be true if the given executable would be launched at login with **any** arguments.
- `launchItems` Object[] *Windows*
  - `name` string *Windows* - name value of a registry entry.
  - `path` string *Windows* - The executable to an app that corresponds to a registry entry.
  - `args` string[] *Windows* - the command-line arguments to pass to the executable.
  - `scope` string *Windows* - one of `user` or `machine`. Indicates whether the registry entry is under `HKEY_CURRENT USER` or `HKEY_LOCAL_MACHINE`.
  - `enabled` boolean *Windows* - `true` if the app registry key is startup approved and therefore shows as `enabled` in Task Manager and Windows settings.

`app.setLoginItemSettings(settings)` *macOS Windows*

- `settings` Object
  - `openAtLogin` boolean (optional) - `true` to open the app at login, `false` to remove the app as a login item. Defaults to `false` .
  - `openAsHidden` boolean (optional) *macOS* - `true` to open the app as hidden. Defaults to `false` . The user can edit this setting from the System Preferences so `app.getLoginItemSettings().wasOpenedAsHidden` should be checked when the app is opened to know the current value. This setting is not available on MAS builds.
  - `path` string (optional) *Windows* - The executable to launch at login. Defaults to `process.execPath` .
  - `args` string[] (optional) *Windows* - The command-line arguments to pass to the executable. Defaults to an empty array. Take care to wrap paths in quotes.
  - `enabled` boolean (optional) *Windows* - `true` will change the startup approved registry key and `enable / disable` the App in Task Manager and Windows Settings. Defaults to `true` .
  - `name` string (optional) *Windows* - value name to write into registry. Defaults to the app's AppUserModelId(). Set the app's login item settings.

To work with Electron's `autoUpdater` on Windows, which uses Squirrel, you'll want to set the launch path to Update.exe, and pass arguments that specify your application name. For example:

```
const appFolder = path.dirname(process.execPath)
const updateExe = path.resolve(appFolder, '..', 'Update.exe')
const exeName = path.basename(process.execPath)

app.setLoginItemSettings({
  openAtLogin: true,
  path: updateExe,
  args: [
    '--processStart', `"${exeName}"`,
    '--process-start-args', `"--hidden"`
  ]
})
```

`app.isAccessibilitySupportEnabled()` *macOS Windows*

Returns `boolean` - `true` if Chrome's accessibility support is enabled, `false` otherwise. This API will return `true` if the use of assistive technologies, such as screen readers, has been detected. See https://www.chromium.org/developers/design-documents/accessibility for more details.

`app.setAccessibilitySupportEnabled(enabled)` *macOS Windows*

- `enabled` boolean - Enable or disable accessibility tree rendering

Manually enables Chrome's accessibility support, allowing to expose accessibility switch to users in application settings. See Chromium's accessibility docs for more details. Disabled by default.

This API must be called after the `ready` event is emitted.

Note: Rendering accessibility tree can significantly affect the performance of your app. It should not be enabled by default.

`app.showAboutPanel()`

Show the app's about panel options. These options can be overridden with `app.setAboutPanelOptions(options)` .

`app.setAboutPanelOptions(options)`

- `options` Object
  - `applicationName` string (optional) - The app's name.
  - `applicationVersion` string (optional) - The app's version.
  - `copyright` string (optional) - Copyright information.
  - `version` string (optional) *macOS* - The app's build version number.
  - `credits` string (optional) *macOS Windows* - Credit information.
  - `authors` string[] (optional) *Linux* - List of app authors.
  - `website` string (optional) *Linux* - The app's website.
  - `iconPath` string (optional) *Linux Windows* - Path to the app's icon in a JPEG or PNG file format. On Linux, will be shown as 64x64 pixels while retaining aspect ratio.

Set the about panel options. This will override the values defined in the app's `.plist` file on macOS. See the Apple docs for more details. On Linux, values must be set in order to be shown; there are no defaults.

If you do not set `credits` but still wish to surface them in your app, AppKit will look for a file named "Credits.html", "Credits.rtf", and "Credits.rtfd", in that order, in the bundle returned by the NSBundle class method main. The first file found is used, and if none is found, the info area is left blank. See Apple documentation for more information.

`app.isEmojiPanelSupported()`

Returns `boolean` - whether or not the current OS version allows for native emoji pickers.

`app.showEmojiPanel()` *macOS Windows*

Show the platform's native emoji picker.

`app.startAccessingSecurityScopedResource(bookmarkData)` *mas*

- `bookmarkData` string - The base64 encoded security scoped bookmark data returned by the `dialog.showOpenDialog` or `dialog.showSaveDialog` methods.

Returns `Function` - This function **must** be called once you have finished accessing the security scoped file. If you do not remember to stop accessing the bookmark, kernel resources will be leaked and your app will lose its ability to reach outside the sandbox completely, until your app is restarted.

```
// Start accessing the file.
const stopAccessingSecurityScopedResource =
app.startAccessingSecurityScopedResource(data)
// You can now access the file outside of the sandbox 🎉

// Remember to stop accessing the file once you've finished with it.
stopAccessingSecurityScopedResource()
```

Start accessing a security scoped resource. With this method Electron applications that are packaged for the Mac App Store may reach outside their sandbox to access files chosen by the user. See Apple's documentation for a description of how this system works.

`app.enableSandbox()`

Enables full sandbox mode on the app. This means that all renderers will be launched sandboxed, regardless of the value of the `sandbox` flag in WebPreferences.

This method can only be called before app is ready.

`app.isInApplicationsFolder()` *macOS*

Returns `boolean` - Whether the application is currently running from the systems Application folder. Use in combination with `app.moveToApplicationsFolder()`

`app.moveToApplicationsFolder([options])` *macOS*

- `options` Object (optional)
    - `conflictHandler` Function<boolean> (optional) - A handler for potential conflict in move failure.
        - `conflictType` string - The type of move conflict encountered by the handler; can be `exists` or `existsAndRunning`, where `exists` means that an app of the same name is present in the Applications directory and `existsAndRunning` means both that it exists and that it's presently running.

Returns `boolean` - Whether the move was successful. Please note that if the move is successful, your application will quit and relaunch.

No confirmation dialog will be presented by default. If you wish to allow the user to confirm the operation, you may do so using the `dialog` API.

NOTE: This method throws errors if anything other than the user causes the move to fail. For instance if the user cancels the authorization dialog, this method returns false. If we fail to perform the copy, then this method will throw an error. The message in the error should be informative and tell you exactly what went wrong.

By default, if an app of the same name as the one being moved exists in the Applications directory and is *not* running, the existing app will be trashed and the active app moved into its place. If it *is* running, the pre-existing running app will assume focus and the previously active app will quit itself. This behavior can be changed by providing the optional conflict handler, where the boolean returned by the handler determines whether or not the move conflict is resolved with default behavior. i.e. returning `false` will ensure no further action is taken, returning `true` will result in the default behavior and the method continuing.

For example:

```
app.moveToApplicationsFolder({
  conflictHandler: (conflictType) => {
    if (conflictType === 'exists') {
      return dialog.showMessageBoxSync({
        type: 'question',
        buttons: ['Halt Move', 'Continue Move'],
        defaultId: 0,
        message: 'An app of this name already exists'
      }) === 1
    }
  }
})
```

Would mean that if an app already exists in the user directory, if the user chooses to 'Continue Move' then the function would continue with its default behavior and the existing app will be trashed and the active app moved into its place.

`app.isSecureKeyboardEntryEnabled()` *macOS*

Returns `boolean` - whether `Secure Keyboard Entry` is enabled.

By default this API will return `false` .

`app.setSecureKeyboardEntryEnabled(enabled)` *macOS*

- `enabled` boolean - Enable or disable `Secure Keyboard Entry`

Set the `Secure Keyboard Entry` is enabled in your application.

By using this API, important information such as password and other sensitive information can be prevented from being intercepted by other processes.

See Apple's documentation for more details.

Note: Enable `Secure Keyboard Entry` only when it is needed and disable it when it is no longer needed.

Properties

`app.accessibilitySupportEnabled` *macOS Windows*

A `boolean` property that's `true` if Chrome's accessibility support is enabled, `false` otherwise. This property will be `true` if the use of assistive technologies, such as screen readers, has been detected. Setting this property to `true` manually enables Chrome's accessibility support, allowing developers to expose accessibility switch to users in application settings.

See Chromium's accessibility docs for more details. Disabled by default.

This API must be called after the `ready` event is emitted.

Note: Rendering accessibility tree can significantly affect the performance of your app. It should not be enabled by default.

`app.applicationMenu`

A `Menu | null` property that returns `Menu` if one has been set and `null` otherwise. Users can pass a Menu to set this property.

`app.badgeCount` *Linux macOS*

An `Integer` property that returns the badge count for current app. Setting the count to `0` will hide the badge.

On macOS, setting this with any nonzero integer shows on the dock icon. On Linux, this property only works for Unity launcher.

Note: Unity launcher requires a `.desktop` file to work. For more information, please read the Unity integration documentation.

Note: On macOS, you need to ensure that your application has the permission to display notifications for this property to take effect.

`app.commandLine` *Readonly*

A `CommandLine` object that allows you to read and manipulate the command line arguments that Chromium uses.

`app.dock` *macOS Readonly*

A `Dock` `| undefined` object that allows you to perform actions on your app icon in the user's dock on macOS.

`app.isPackaged` *Readonly*

A `boolean` property that returns `true` if the app is packaged, `false` otherwise. For many apps, this property can be used to distinguish development and production environments.

`app.name`

A `string` property that indicates the current application's name, which is the name in the application's `package.json` file.

Usually the `name` field of `package.json` is a short lowercase name, according to the npm modules spec. You should usually also specify a `productName` field, which is your application's full capitalized name, and which will be preferred over `name` by Electron.

`app.userAgentFallback`

A `string` which is the user agent string Electron will use as a global fallback.

This is the user agent that will be used when no user agent is set at the `webContents` or `session` level. It is useful for ensuring that your entire app has the same user agent. Set to a custom value as early as possible in your app's initialization to ensure that your overridden value is used.

`app.runningUnderRosettaTranslation` *macOS Readonly Deprecated*

A `boolean` which when `true` indicates that the app is currently running under the Rosetta Translator Environment.

You can use this property to prompt users to download the arm64 version of your application when they are running the x64 version under Rosetta incorrectly.

Deprecated: This property is superceded by the `runningUnderARM64Translation` property which detects when the app is being translated to ARM64 in both macOS and Windows.

`app.runningUnderARM64Translation` *Readonly macOS Windows*

A `boolean` which when `true` indicates that the app is currently running under an ARM64 translator (like the macOS Rosetta Translator Environment or Windows WOW).

You can use this property to prompt users to download the arm64 version of your application when they are running the x64 version under Rosetta incorrectly.