[Docs](https://electron.atom.io/docs/) [Blog](https://electron.atom.io/blog/) [Community](https://electron.atom.io/community/) [Apps](https://electron.atom.io/apps/) [Userland](https://electron.atom.io/userland/) [Releases](https://electron.atom.io/releases/) [Contact](https://electron.atom.io/contact/)

Electron Documentation1.7.6

[Docs](https://electron.atom.io/docs/) / All

Accelerator

定义快捷键以便更快的操作应用

Define keyboard shortcuts.

Accelerators are Strings that can contain multiple modifiers and key codes, combined by the + character, and are used to define keyboard shortcuts throughout your application.

Examples:

* CommandOrControl+A
* CommandOrControl+Shift+Z

Shortcuts are registered with the [globalShortcut](https://electron.atom.io/docs/api/global-shortcut) module using the [register](https://electron.atom.io/docs/api/global-shortcut#globalshortcutregisteraccelerator-callback) method, i.e.

**const** {app, globalShortcut} = require('electron')

app.on('ready', () => {

*// Register a 'CommandOrControl+Y' shortcut listener.*

globalShortcut.register('CommandOrControl+Y', () => {

*// Do stuff when Y and either Command/Control is pressed.*

})

})

设置快捷键：

Electron的快捷键是通过[globalShortcut](https://electron.atom.io/docs/api/global-shortcut) 模块来注册的，在app的ready阶段可以通过模块globalShortcut的register方法注册快捷键,快捷键的含义是当某些键被按下时触发的事件。

Platform notice

On Linux and Windows, the Command key does not have any effect so use CommandOrControl which represents Command on macOS and Control on Linux and Windows to define some accelerators.

Use Alt instead of Option. The Option key only exists on macOS, whereas the Alt key is available on all platforms.

The Super key is mapped to the Windows key on Windows and Linux and Cmd on macOS.

Available modifiers

* Command (or Cmd for short)
* Control (or Ctrl for short)
* CommandOrControl (or CmdOrCtrl for short)
* Alt
* Option
* AltGr
* Shift
* Super

可用的改变，即某些命令组合的缩写形式：

Cmd-Command

Ctrl-Control

CmdOrCtrl-CommandOrControl

Available key codes

* 0 to 9
* A to Z
* F1 to F24
* Punctuations like ~, !, @, #, $, etc.
* Plus
* Space
* Tab
* Backspace
* Delete
* Insert
* Return (or Enter as alias)
* Up, Down, Left and Right
* Home and End
* PageUp and PageDown
* Escape (or Esc for short)
* VolumeUp, VolumeDown and VolumeMute
* MediaNextTrack, MediaPreviousTrack, MediaStop and MediaPlayPause
* PrintScreen

可以使用的键：

数字：0-9

字母：A-Z

F1-F24

~ ！ @ # $等等

加号：+

空格：

Tab键

后退键

删除键

插入键

回车键

上 下 左 右

Home键 end 键

上一页pageup 下一页 pagedown

Esc

音量控制键等等

app

Control your application’s event lifecycle.

app（名字可以自定义）用来控制应用的事件的生命周期，整个app就是类似事件驱动

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

主进程，运行main.js文件（文件名可以自定义）

The following example shows how to quit the application when the last window is closed（下面的例子展示了当窗口被关闭的时候应用退出）:

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

app.on('window-all-closed', () => { //这里添加了一个监听函数

//ES6写法，可以使用一个function代替，当事件被触发时就会调用该监听函数

app.quit()

})

Events

The app object emits the following events（app对象触发以下事件）:

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 just do everything in the ready event handler.

will-finish-launching事件当应用启动基本完成时触发

在windows和linux上该事件等同于ready事件。在mac os中有意义。具体参看ready事件

Event: ‘ready’

Returns:

* launchInfo Object macOS

Emitted when Electron has finished initializing. On macOS, launchInfo holds the userInfo of the NSUserNotification that was used to open the application, if it was launched from Notification Center. You can call app.isReady() to check if this event has already fired.

ready当应用启动基本完成时触发ready事件

返回launchInfo对象

可以调用方法app.isReady()来查看事件是否应被触发

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.

window-all-closed当所有的窗口都被关闭的时候触发

如果没有注册监听该事件，则当所有的串口关闭的时候，应用汇默认退出

如果注册监听该事件，则app的退出由开发人员控制

如果用户按下了ctrl+q，或者开发者调用了app.quit函数，则electron将会先尝试关闭所有的窗口然后触发will-quit事件，这时window-all-closed事件将不会被触发

Event: ‘before-quit’

Returns:

* event Event

Emitted before the application starts closing its windows. Calling event.preventDefault() will prevent the default behaviour, 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.

before-quit事件在应用开始关闭它的窗口之前触发

返回Event对象

调用event.preventDefault将会阻止终止应用的默认行为

注意：如果应用的退出被autoUpdater.quitAndInstall()发起。则before-quit事件将会在所有windows的close事件被触发并且关闭了所有的windows之后触发

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 behaviour, 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.

will-quit事件当所有的windows被关闭了之后被触发，并且应用将会退出

调用event.preventDefault()方法将会组织这个终止应用的默认行为。Will-quit和window-all-closed事件的不同请查看window-all-closed事件的描述

Event: ‘quit’

Returns:

* event Event
* exitCode Integer

Emitted when the application is quitting.

quit事件当应用退出时触发

返回Event对象和整型的退出码

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: ‘continue-activity’ macOS

Returns:

* event Event
* type String - A string identifying the activity. Maps to [NSUserActivity.activityType](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSUserActivity_Class/index.html#//apple_ref/occ/instp/NSUserActivity/activityType).
* userInfo Object - Contains app-specific state stored by the activity on another device.

Emitted during [Handoff](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/Handoff/HandoffFundamentals/HandoffFundamentals.html) 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: ‘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](https://electron.atom.io/docs/api/browser-window) gets blurred.

browser-window-blur事件当[browserWindow](https://electron.atom.io/docs/api/browser-window)被blurred后触发

返回Event对象和BrowserWindow窗口对象

Event: ‘browser-window-focus’

Returns:

* event Event
* window BrowserWindow

Emitted when a [browserWindow](https://electron.atom.io/docs/api/browser-window) gets focused.

Event: ‘browser-window-created’

Returns:

* event Event
* window BrowserWindow

Emitted when a new [browserWindow](https://electron.atom.io/docs/api/browser-window) is created.

Event: ‘web-contents-created’

Returns:

* event Event
* webContents WebContents

Emitted when a new [webContents](https://electron.atom.io/docs/api/web-contents) is created.

Event: ‘certificate-error’

Returns:

* event Event
* webContents [WebContents](https://electron.atom.io/docs/api/web-contents)
* url String
* error String - The error code
* certificate [Certificate](https://electron.atom.io/docs/api/structures/certificate)
* callback Function
  + isTrusted Boolean - Whether to consider the certificate as trusted

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](https://electron.atom.io/docs/api/web-contents)
* url URL
* certificateList [Certificate[]](https://electron.atom.io/docs/api/structures/certificate)
* callback Function
  + certificate [Certificate](https://electron.atom.io/docs/api/structures/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])

})

Select-client-certificate事件当一个客户端认证被要求时触发，即当某个连接的客户端需要认证时触发该事件

返回：该event事件对象，webContents对象，url，认证列表，回调函数（可选认证的参数）

url对应的是要求client认证的导航入口；callback回调函数会筛选认证列表

使用event.preventDefault()函数可以阻止应用使用认证列表的第一个认证的默认行为

Event: ‘login’

Returns:

* event Event
* webContents [WebContents](https://electron.atom.io/docs/api/web-contents)
* request Object
  + method String
  + url URL
  + referrer URL
* authInfo Object
  + isProxy Boolean
  + scheme String
  + host String
  + port Integer
  + realm String
* callback Function
  + username String
  + password String

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, request, authInfo, callback) => {

event.preventDefault()

callback('username', 'secret')

})

Login事件当webContents需要基本的认证信息时触发。

返回该event对象；webContents对象；request对象，request对象包含的属性有：method,url,referrer-前一个请求连接；authInfo对象，authInfo对象包含的属性有isProxy，scheme，host，port,realm；callback函数，callback函数包含的参数有username，password

该事件的默认行为是取消所有的认证，为了阻止这个默认行为需要调用event.preventDefault()函数并且使用认证信息调用回调函数callback(username, password)。

Event: ‘gpu-process-crashed’

Returns:

* event Event
* killed Boolean

Emitted when the gpu process crashes or is killed.

gpu-process-crashed事件当gpu进程崩溃或者被杀死的时候被触发

返回：event事件对象，killed是否被杀死

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.

accessibility-support-changed事件当chrome的可达性支持改变后被触发

返回：event事件对象；accessibilitySupportEnabled 即可达性支持是否被启用

当辅助的技术如屏幕渲染被启用或者禁用时被触发。

Methods

The app object has the following methods:

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

app对象包含以下方法：

注意：一些方法只在特定的操作系统上可用，这些方法已经做了相应的标记

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.

该方法尝试关闭所有的窗口，该方法首先触发before-quit事件，如果所有的窗口被成功关闭则will-quit事件被默认触发，然后默认应用会终止。

这个方法保证了所有的beforeunload和unload时间呗正确的执行。窗口可以再解决beforeunload事件的时候通过返回false来取消退出

app.exit([exitCode])

* exitCode Integer (optional)

Exits immediately with exitCode. exitCode defaults to 0.

All windows will be closed immediately without asking 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 execPathis 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.

app.focus()

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.

app.hide() macOS

Hides all application windows without minimizing them.

app.show() macOS

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

app.getAppPath()

Returns String - The current application directory.

app.getPath(name)

* name String

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

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.
* pepperFlashSystemPlugin Full path to the system version of the Pepper Flash plugin.

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

* path String
* options Object (optional)
  + size String
    - small - 16x16
    - normal - 32x32
    - large - 48x48 on Linux, 32x32 on Windows, unsupported on macOS.
* callback Function
  + error Error
  + icon [NativeImage](https://electron.atom.io/docs/api/native-image)

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, the directory will be created by this method. On failure an Error is thrown.

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.jsonfile, 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 lowercased 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.

app.getLocale()

Returns String - The current application locale. Possible return values are documented [here](https://electron.atom.io/docs/api/locales).

**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.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]) macOS Windows

* protocol String - The name of your protocol, without ://. If you want your app to handle electron://links, call this method with electron as the parameter.
* 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 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.

On Windows you can provide optional parameters path, the path to your executable, and args, an array of arguments to be passed to your executable when it launches.

**Note:** On macOS, you can only register protocols that have been added to your app’s info.plist, which can not be modified at runtime. You can however change the file with a simple text editor or script during build time. Please refer to [Apple’s documentation](https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html#//apple_ref/doc/uid/TP40009249-102207-TPXREF115) for details.

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]) 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

This method checks if the current executable is the default handler for a protocol (aka URI scheme). If so, it will return true. Otherwise, it will return false.

**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](https://developer.apple.com/library/mac/documentation/Carbon/Reference/LaunchServicesReference/#//apple_ref/c/func/LSCopyDefaultHandlerForURLScheme) for details.

The API uses the Windows Registry and LSCopyDefaultHandlerForURLScheme internally.

app.setUserTasks(tasks) Windows

* tasks [Task[]](https://electron.atom.io/docs/api/structures/task) - Array of Task objects

Adds tasks to the [Tasks](http://msdn.microsoft.com/en-us/library/windows/desktop/dd378460(v=vs.85).aspx#tasks) category of the JumpList on Windows.

tasks is an array of [Task](https://electron.atom.io/docs/api/structures/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](https://msdn.microsoft.com/en-us/library/windows/desktop/dd378398(v=vs.85).aspx)).
* removedItems [JumpListItem[]](https://electron.atom.io/docs/api/structures/jump-list-item) - 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[]](https://electron.atom.io/docs/api/structures/jump-list-category) or 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().

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.makeSingleInstance(callback)

* callback Function
  + argv String[] - An array of the second instance’s command line arguments
  + workingDirectory String - The second instance’s working directory

Returns Boolean.

This method makes your application a Single Instance Application - instead of allowing multiple instances of your app to run, this will ensure that only a single instance of your app is running, and other instances signal this instance and exit.

callback will be called by the first instance with callback(argv, workingDirectory) when a second instance has been executed. 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.

The callback is guaranteed to be executed after the ready event of app gets emitted.

This method returns false if your process is the primary instance of the application and your app should continue loading. And returns true if your process has sent its parameters to another instance, and you should immediately quit.

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

**const** isSecondInstance = app.makeSingleInstance((commandLine, workingDirectory) => {

*// Someone tried to run a second instance, we should focus our window.*

**if** (myWindow) {

**if** (myWindow.isMinimized()) myWindow.restore()

myWindow.focus()

}

})

**if** (isSecondInstance) {

app.quit()

}

*// Create myWindow, load the rest of the app, etc...*

app.on('ready', () => {

})

app.releaseSingleInstance()

Releases all locks that were created by makeSingleInstance. 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](https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSUserActivity_Class/index.html#//apple_ref/occ/instp/NSUserActivity/activityType).
* userInfo Object - 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](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/Handoff/HandoffFundamentals/HandoffFundamentals.html) to another device afterward.

app.getCurrentActivityType() macOS

Returns String - The type of the currently running activity.

app.setAppUserModelId(id) Windows

* id String

Changes the [Application User Model ID](https://msdn.microsoft.com/en-us/library/windows/desktop/dd378459(v=vs.85).aspx) to id.

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](https://code.google.com/p/chromium/codesearch#chromium/src/net/base/net_error_list.h).

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 behaviour.

This method can only be called before app is ready.

app.getAppMemoryInfo() Deprecated

Returns [ProcessMetric[]](https://electron.atom.io/docs/api/structures/process-metric): Array of ProcessMetric objects that correspond to memory and cpu usage statistics of all the processes associated with the app. **Note:** This method is deprecated, use app.getAppMetrics() instead.

app.getAppMetrics()

Returns [ProcessMetric[]](https://electron.atom.io/docs/api/structures/process-metric): Array of ProcessMetric objects that correspond to memory and cpu usage statistics of all the processes associated with the app.

app.getGpuFeatureStatus()

Returns [GPUFeatureStatus](https://electron.atom.io/docs/api/structures/gpu-feature-status) - The Graphics Feature Status from chrome://gpu/.

app.setBadgeCount(count) Linux macOS

* count Integer

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 the existence of a .desktop file to work, for more information please read [Desktop Environment Integration](https://electron.atom.io/docs/tutorial/desktop-environment-integration#unity-launcher-shortcuts-linux).

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 - true if the app is set to open as hidden at login. This setting is only supported on macOS.
* wasOpenedAtLogin Boolean - true if the app was opened at login automatically. This setting is only supported on macOS.
* wasOpenedAsHidden Boolean - 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 only supported on macOS.
* restoreState Boolean - 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 only supported on macOS.

**Note:** This API has no effect on [MAS builds](https://electron.atom.io/docs/tutorial/mac-app-store-submission-guide).

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) - true to open the app as hidden. Defaults to false. The user can edit this setting from the System Preferences so app.getLoginItemStatus().wasOpenedAsHidden should be checked when the app is opened to know the current value. This setting is only supported on macOS.
  + 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.

Set the app’s login item settings.

To work with Electron’s autoUpdater on Windows, which uses [Squirrel](https://github.com/Squirrel/Squirrel.Windows), 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"`

]

})

**Note:** This API has no effect on [MAS builds](https://electron.atom.io/docs/tutorial/mac-app-store-submission-guide).

app.isAccessibilitySupportEnabled() macOS Windows

Returns Boolean - true if Chrome’s accessibility support is enabled, false otherwise. This API will return trueif 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.setAboutPanelOptions(options) macOS

* options Object
  + applicationName String (optional) - The app’s name.
  + applicationVersion String (optional) - The app’s version.
  + copyright String (optional) - Copyright information.
  + credits String (optional) - Credit information.
  + version String (optional) - The app’s build version number.

Set the about panel options. This will override the values defined in the app’s .plist file. See the [Apple docs](https://developer.apple.com/reference/appkit/nsapplication/1428479-orderfrontstandardaboutpanelwith?language=objc) for more details.

app.commandLine.appendSwitch(switch[, value])

* switch String - A command-line switch
* value String (optional) - A value for the given switch

Append a switch (with optional value) to Chromium’s command line.

**Note:** This will not affect process.argv, and is mainly used by developers to control some low-level Chromium behaviors.

app.commandLine.appendArgument(value)

* value String - The argument to append to the command line

Append an argument to Chromium’s command line. The argument will be quoted correctly.

**Note:** This will not affect process.argv.

app.enableMixedSandbox() Experimental macOS Windows

Enables mixed sandbox mode on the app.

This method can only be called before app is ready.

app.dock.bounce([type]) macOS

* type String (optional) - Can be critical or informational. The default is informational

When critical is passed, the dock icon will bounce until either the application becomes active or the request is canceled.

When informational is passed, the dock icon will bounce for one second. However, the request remains active until either the application becomes active or the request is canceled.

Returns Integer an ID representing the request.

app.dock.cancelBounce(id) macOS

* id Integer

Cancel the bounce of id.

app.dock.downloadFinished(filePath) macOS

* filePath String

Bounces the Downloads stack if the filePath is inside the Downloads folder.

app.dock.setBadge(text) macOS

* text String

Sets the string to be displayed in the dock’s badging area.

app.dock.getBadge() macOS

Returns String - The badge string of the dock.

app.dock.hide() macOS

Hides the dock icon.

app.dock.show() macOS

Shows the dock icon.

app.dock.isVisible() macOS

Returns Boolean - Whether the dock icon is visible. The app.dock.show() call is asynchronous so this method might not return true immediately after that call.

app.dock.setMenu(menu) macOS

* menu [Menu](https://electron.atom.io/docs/api/menu)

Sets the application’s [dock menu](https://developer.apple.com/library/mac/documentation/Carbon/Conceptual/customizing_docktile/concepts/dockconcepts.html#//apple_ref/doc/uid/TP30000986-CH2-TPXREF103).

app.dock.setIcon(image) macOS

* image ([NativeImage](https://electron.atom.io/docs/api/native-image) | String)

Sets the image associated with this dock icon.

autoUpdater

Enable apps to automatically update themselves.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

The autoUpdater module provides an interface for the [Squirrel](https://github.com/Squirrel) framework.

You can quickly launch a multi-platform release server for distributing your application by using one of these projects:

* [nuts](https://github.com/GitbookIO/nuts): *A smart release server for your applications, using GitHub as a backend. Auto-updates with Squirrel (Mac & Windows)*
* [electron-release-server](https://github.com/ArekSredzki/electron-release-server): *A fully featured, self-hosted release server for electron applications, compatible with auto-updater*
* [squirrel-updates-server](https://github.com/Aluxian/squirrel-updates-server): *A simple node.js server for Squirrel.Mac and Squirrel.Windows which uses GitHub releases*
* [squirrel-release-server](https://github.com/Arcath/squirrel-release-server): *A simple PHP application for Squirrel.Windows which reads updates from a folder. Supports delta updates.*

Platform notices

Though autoUpdater provides a uniform API for different platforms, there are still some subtle differences on each platform.

macOS

On macOS, the autoUpdater module is built upon [Squirrel.Mac](https://github.com/Squirrel/Squirrel.Mac), meaning you don’t need any special setup to make it work. For server-side requirements, you can read [Server Support](https://github.com/Squirrel/Squirrel.Mac#server-support). Note that [App Transport Security](https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW35) (ATS) applies to all requests made as part of the update process. Apps that need to disable ATS can add the NSAllowsArbitraryLoads key to their app’s plist.

**Note:** Your application must be signed for automatic updates on macOS. This is a requirement of Squirrel.Mac.

Windows

On Windows, you have to install your app into a user’s machine before you can use the autoUpdater, so it is recommended that you use the [electron-winstaller](https://github.com/electron/windows-installer), [electron-forge](https://github.com/electron-userland/electron-forge) or the [grunt-electron-installer](https://github.com/electron/grunt-electron-installer) package to generate a Windows installer.

When using [electron-winstaller](https://github.com/electron/windows-installer) or [electron-forge](https://github.com/electron-userland/electron-forge) make sure you do not try to update your app [the first time it runs](https://github.com/electron/windows-installer#handling-squirrel-events) (Also see [this issue for more info](https://github.com/electron/electron/issues/7155)). It’s also recommended to use [electron-squirrel-startup](https://github.com/mongodb-js/electron-squirrel-startup) to get desktop shortcuts for your app.

The installer generated with Squirrel will create a shortcut icon with an [Application User Model ID](https://msdn.microsoft.com/en-us/library/windows/desktop/dd378459(v=vs.85).aspx) in the format of com.squirrel.PACKAGE\_ID.YOUR\_EXE\_WITHOUT\_DOT\_EXE, examples are com.squirrel.slack.Slack and com.squirrel.code.Code. You have to use the same ID for your app with app.setAppUserModelId API, otherwise Windows will not be able to pin your app properly in task bar.

Unlike Squirrel.Mac, Windows can host updates on S3 or any other static file host. You can read the documents of [Squirrel.Windows](https://github.com/Squirrel/Squirrel.Windows) to get more details about how Squirrel.Windows works.

Linux

There is no built-in support for auto-updater on Linux, so it is recommended to use the distribution’s package manager to update your app.

Events

The autoUpdater object emits the following events:

Event: ‘error’

Returns:

* error Error

Emitted when there is an error while updating.

Event: ‘checking-for-update’

Emitted when checking if an update has started.

Event: ‘update-available’

Emitted when there is an available update. The update is downloaded automatically.

Event: ‘update-not-available’

Emitted when there is no available update.

Event: ‘update-downloaded’

Returns:

* event Event
* releaseNotes String
* releaseName String
* releaseDate Date
* updateURL String

Emitted when an update has been downloaded.

On Windows only releaseName is available.

Methods

The autoUpdater object has the following methods:

autoUpdater.setFeedURL(url[, requestHeaders])

* url String
* requestHeaders Object macOS (optional) - HTTP request headers.

Sets the url and initialize the auto updater.

autoUpdater.getFeedURL()

Returns String - The current update feed URL.

autoUpdater.checkForUpdates()

Asks the server whether there is an update. You must call setFeedURL before using this API.

autoUpdater.quitAndInstall()

Restarts the app and installs the update after it has been downloaded. It should only be called after update-downloaded has been emitted.

**Note:** autoUpdater.quitAndInstall() will close all application windows first and only emit before-quit event on app after that. This is different from the normal quit event sequence.

Class: BrowserView

Create and control views.

**Note:** The BrowserView API is currently experimental and may change or be removed in future Electron releases.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

A BrowserView can be used to embed additional web content into a BrowserWindow. It is like a child window, except that it is positioned relative to its owning window. It is meant to be an alternative to the webview tag.

Example

*// In the main process.*

**const** {BrowserView, BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({width: 800, height: 600})

win.on('closed', () => {

win = null

})

**let** view = **new** BrowserView({

webPreferences: {

nodeIntegration: false

}

})

win.setBrowserView(view)

view.setBounds({ x: 0, y: 0, width: 300, height: 300 })

view.webContents.loadURL('https://electron.atom.io')

new BrowserView([options]) Experimental

* options Object (optional)
  + webPreferences Object (optional) - See [BrowserWindow](https://electron.atom.io/docs/api/browser-window).

Static Methods

BrowserView.fromId(id)

* id Integer

Returns BrowserView - The view with the given id.

Instance Properties

Objects created with new BrowserView have the following properties:

view.webContents Experimental

A [WebContents](https://electron.atom.io/docs/api/web-contents) object owned by this view.

view.id Experimental

A Integer representing the unique ID of the view.

Instance Methods

Objects created with new BrowserView have the following instance methods:

view.setAutoResize(options) Experimental

* options Object
  + width Boolean - If true, the view’s width will grow and shrink together with the window. false by default.
  + height Boolean - If true, the view’s height will grow and shrink together with the window. false by default.

view.setBounds(bounds) Experimental

* bounds [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)

Resizes and moves the view to the supplied bounds relative to the window.

view.setBackgroundColor(color) Experimental

* color String - Color in #aarrggbb or #argb form. The alpha channel is optional.

Class: BrowserWindowProxy

Manipulate the child browser window

Process: [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

The BrowserWindowProxy object is returned from window.open and provides limited functionality with the child window.

Instance Methods

The BrowserWindowProxy object has the following instance methods:

win.blur()

Removes focus from the child window.

win.close()

Forcefully closes the child window without calling its unload event.

win.eval(code)

* code String

Evaluates the code in the child window.

win.focus()

Focuses the child window (brings the window to front).

win.print()

Invokes the print dialog on the child window.

win.postMessage(message, targetOrigin)

* message String
* targetOrigin String

Sends a message to the child window with the specified origin or \* for no origin preference.

In addition to these methods, the child window implements window.opener object with no properties and a single method.

Instance Properties

The BrowserWindowProxy object has the following instance properties:

win.closed

A Boolean that is set to true after the child window gets closed.

BrowserWindow

Create and control browser windows.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

*// In the main process.*

**const** {BrowserWindow} = require('electron')

*// Or use `remote` from the renderer process.*

*// const {BrowserWindow} = require('electron').remote*

**let** win = **new** BrowserWindow({width: 800, height: 600})

win.on('closed', () => {

win = null

})

*// Load a remote URL*

win.loadURL('https://github.com')

*// Or load a local HTML file*

win.loadURL(`file:*//${\_\_dirname}/app/index.html`)*

Frameless window

To create a window without chrome, or a transparent window in arbitrary shape, you can use the [Frameless Window](https://electron.atom.io/docs/api/frameless-window) API.

Showing window gracefully

When loading a page in the window directly, users may see the page load incrementally, which is not a good experience for a native app. To make the window display without visual flash, there are two solutions for different situations.

Using ready-to-show event

While loading the page, the ready-to-show event will be emitted when the renderer process has rendered the page for the first time if the window has not been shown yet. Showing the window after this event will have no visual flash:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({show: false})

win.once('ready-to-show', () => {

win.show()

})

This event is usually emitted after the did-finish-load event, but for pages with many remote resources, it may be emitted before the did-finish-load event.

Setting backgroundColor

For a complex app, the ready-to-show event could be emitted too late, making the app feel slow. In this case, it is recommended to show the window immediately, and use a backgroundColor close to your app’s background:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({backgroundColor: '#2e2c29'})

win.loadURL('https://github.com')

Note that even for apps that use ready-to-show event, it is still recommended to set backgroundColor to make app feel more native.

Parent and child windows

By using parent option, you can create child windows:

**const** {BrowserWindow} = require('electron')

**let** top = **new** BrowserWindow()

**let** child = **new** BrowserWindow({parent: top})

child.show()

top.show()

The child window will always show on top of the top window.

Modal windows

A modal window is a child window that disables parent window, to create a modal window, you have to set both parent and modal options:

**const** {BrowserWindow} = require('electron')

**let** child = **new** BrowserWindow({parent: top, modal: true, show: false})

child.loadURL('https://github.com')

child.once('ready-to-show', () => {

child.show()

})

Page visibility

The [Page Visibility API](https://developer.mozilla.org/en-US/docs/Web/API/Page_Visibility_API) works as follows:

* On all platforms, the visibility state tracks whether the window is hidden/minimized or not.
* Additionally, on macOS, the visibility state also tracks the window occlusion state. If the window is occluded (i.e. fully covered) by another window, the visibility state will be hidden. On other platforms, the visibility state will be hidden only when the window is minimized or explicitly hidden with win.hide().
* If a BrowserWindow is created with show: false, the initial visibility state will be visible despite the window actually being hidden.
* If backgroundThrottling is disabled, the visibility state will remain visible even if the window is minimized, occluded, or hidden.

It is recommended that you pause expensive operations when the visibility state is hidden in order to minimize power consumption.

Platform notices

* On macOS modal windows will be displayed as sheets attached to the parent window.
* On macOS the child windows will keep the relative position to parent window when parent window moves, while on Windows and Linux child windows will not move.
* On Windows it is not supported to change parent window dynamically.
* On Linux the type of modal windows will be changed to dialog.
* On Linux many desktop environments do not support hiding a modal window.

Class: BrowserWindow

Create and control browser windows.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

BrowserWindow is an [EventEmitter](http://nodejs.org/api/events.html#events_class_events_eventemitter).

It creates a new BrowserWindow with native properties as set by the options.

new BrowserWindow([options])

* options Object (optional)
  + width Integer (optional) - Window’s width in pixels. Default is 800.
  + height Integer (optional) - Window’s height in pixels. Default is 600.
  + x Integer (optional) (**required** if y is used) - Window’s left offset from screen. Default is to center the window.
  + y Integer (optional) (**required** if x is used) - Window’s top offset from screen. Default is to center the window.
  + useContentSize Boolean (optional) - The width and height would be used as web page’s size, which means the actual window’s size will include window frame’s size and be slightly larger. Default is false.
  + center Boolean (optional) - Show window in the center of the screen.
  + minWidth Integer (optional) - Window’s minimum width. Default is 0.
  + minHeight Integer (optional) - Window’s minimum height. Default is 0.
  + maxWidth Integer (optional) - Window’s maximum width. Default is no limit.
  + maxHeight Integer (optional) - Window’s maximum height. Default is no limit.
  + resizable Boolean (optional) - Whether window is resizable. Default is true.
  + movable Boolean (optional) - Whether window is movable. This is not implemented on Linux. Default is true.
  + minimizable Boolean (optional) - Whether window is minimizable. This is not implemented on Linux. Default is true.
  + maximizable Boolean (optional) - Whether window is maximizable. This is not implemented on Linux. Default is true.
  + closable Boolean (optional) - Whether window is closable. This is not implemented on Linux. Default is true.
  + focusable Boolean (optional) - Whether the window can be focused. Default is true. On Windows setting focusable: false also implies setting skipTaskbar: true. On Linux setting focusable: falsemakes the window stop interacting with wm, so the window will always stay on top in all workspaces.
  + alwaysOnTop Boolean (optional) - Whether the window should always stay on top of other windows. Default is false.
  + fullscreen Boolean (optional) - Whether the window should show in fullscreen. When explicitly set to false the fullscreen button will be hidden or disabled on macOS. Default is false.
  + fullscreenable Boolean (optional) - Whether the window can be put into fullscreen mode. On macOS, also whether the maximize/zoom button should toggle full screen mode or maximize window. Default is true.
  + skipTaskbar Boolean (optional) - Whether to show the window in taskbar. Default is false.
  + kiosk Boolean (optional) - The kiosk mode. Default is false.
  + title String (optional) - Default window title. Default is "Electron".
  + icon ([NativeImage](https://electron.atom.io/docs/api/native-image) | String) (optional) - The window icon. On Windows it is recommended to use ICOicons to get best visual effects, you can also leave it undefined so the executable’s icon will be used.
  + show Boolean (optional) - Whether window should be shown when created. Default is true.
  + frame Boolean (optional) - Specify false to create a [Frameless Window](https://electron.atom.io/docs/api/frameless-window). Default is true.
  + parent BrowserWindow (optional) - Specify parent window. Default is null.
  + modal Boolean (optional) - Whether this is a modal window. This only works when the window is a child window. Default is false.
  + acceptFirstMouse Boolean (optional) - Whether the web view accepts a single mouse-down event that simultaneously activates the window. Default is false.
  + disableAutoHideCursor Boolean (optional) - Whether to hide cursor when typing. Default is false.
  + autoHideMenuBar Boolean (optional) - Auto hide the menu bar unless the Alt key is pressed. Default is false.
  + enableLargerThanScreen Boolean (optional) - Enable the window to be resized larger than screen. Default is false.
  + backgroundColor String (optional) - Window’s background color as Hexadecimal value, like #66CD00 or #FFF or #80FFFFFF (alpha is supported). Default is #FFF (white).
  + hasShadow Boolean (optional) - Whether window should have a shadow. This is only implemented on macOS. Default is true.
  + darkTheme Boolean (optional) - Forces using dark theme for the window, only works on some GTK+3 desktop environments. Default is false.
  + transparent Boolean (optional) - Makes the window [transparent](https://electron.atom.io/docs/api/frameless-window). Default is false.
  + type String (optional) - The type of window, default is normal window. See more about this below.
  + titleBarStyle String (optional) - The style of window title bar. Default is default. Possible values are:
    - default - Results in the standard gray opaque Mac title bar.
    - hidden - Results in a hidden title bar and a full size content window, yet the title bar still has the standard window controls (“traffic lights”) in the top left.
    - hidden-inset - Deprecated, use hiddenInset instead.
    - hiddenInset - Results in a hidden title bar with an alternative look where the traffic light buttons are slightly more inset from the window edge.
    - customButtonsOnHover Boolean (optional) - Draw custom close, minimize, and full screen buttons on macOS frameless windows. These buttons will not display unless hovered over in the top left of the window. These custom buttons prevent issues with mouse events that occur with the standard window toolbar buttons. **Note:** This option is currently experimental.
  + fullscreenWindowTitle Boolean (optional) - Shows the title in the tile bar in full screen mode on macOS for all titleBarStyle options. Default is false.
  + thickFrame Boolean (optional) - Use WS\_THICKFRAME style for frameless windows on Windows, which adds standard window frame. Setting it to false will remove window shadow and window animations. Default is true.
  + vibrancy String (optional) - Add a type of vibrancy effect to the window, only on macOS. Can be appearance-based, light, dark, titlebar, selection, menu, popover, sidebar, medium-light or ultra-dark.
  + zoomToPageWidth Boolean (optional) - Controls the behavior on macOS when option-clicking the green stoplight button on the toolbar or by clicking the Window > Zoom menu item. If true, the window will grow to the preferred width of the web page when zoomed, false will cause it to zoom to the width of the screen. This will also affect the behavior when calling maximize() directly. Default is false.
  + tabbingIdentifier String (optional) - Tab group name, allows opening the window as a native tab on macOS 10.12+. Windows with the same tabbing identifier will be grouped together. This also adds a native new tab button to your window’s tab bar and allows your app and window to receive the new-window-for-tab event.
  + webPreferences Object (optional) - Settings of web page’s features.
    - devTools Boolean (optional) - Whether to enable DevTools. If it is set to false, can not use BrowserWindow.webContents.openDevTools() to open DevTools. Default is true.
    - nodeIntegration Boolean (optional) - Whether node integration is enabled. Default is true.
    - nodeIntegrationInWorker Boolean (optional) - Whether node integration is enabled in web workers. Default is false. More about this can be found in [Multithreading](https://electron.atom.io/docs/tutorial/multithreading).
    - preload String (optional) - Specifies a script that will be loaded before other scripts run in the page. This script will always have access to node APIs no matter whether node integration is turned on or off. The value should be the absolute file path to the script. When node integration is turned off, the preload script can reintroduce Node global symbols back to the global scope. See example [here](https://electron.atom.io/docs/api/process#event-loaded).
    - sandbox Boolean (optional) - If set, this will sandbox the renderer associated with the window, making it compatible with the Chromium OS-level sandbox and disabling the Node.js engine. This is not the same as the nodeIntegration option and the APIs available to the preload script are more limited. Read more about the option [here](https://electron.atom.io/docs/api/sandbox-option). **Note:** This option is currently experimental and may change or be removed in future Electron releases.
    - session [Session](https://electron.atom.io/docs/api/session#class-session) (optional) - Sets the session used by the page. Instead of passing the Session object directly, you can also choose to use the partition option instead, which accepts a partition string. When both session and partition are provided, session will be preferred. Default is the default session.
    - partition String (optional) - Sets the session used by the page according to the session’s partition string. If partition starts with persist:, the page will use a persistent session available to all pages in the app with the same partition. If there is no persist: prefix, the page will use an in-memory session. By assigning the same partition, multiple pages can share the same session. Default is the default session.
    - zoomFactor Number (optional) - The default zoom factor of the page, 3.0 represents 300%. Default is 1.0.
    - javascript Boolean (optional) - Enables JavaScript support. Default is true.
    - webSecurity Boolean (optional) - When false, it will disable the same-origin policy (usually using testing websites by people), and set allowRunningInsecureContent to true if this options has not been set by user. Default is true.
    - allowRunningInsecureContent Boolean (optional) - Allow an https page to run JavaScript, CSS or plugins from http URLs. Default is false.
    - images Boolean (optional) - Enables image support. Default is true.
    - textAreasAreResizable Boolean (optional) - Make TextArea elements resizable. Default is true.
    - webgl Boolean (optional) - Enables WebGL support. Default is true.
    - webaudio Boolean (optional) - Enables WebAudio support. Default is true.
    - plugins Boolean (optional) - Whether plugins should be enabled. Default is false.
    - experimentalFeatures Boolean (optional) - Enables Chromium’s experimental features. Default is false.
    - experimentalCanvasFeatures Boolean (optional) - Enables Chromium’s experimental canvas features. Default is false.
    - scrollBounce Boolean (optional) - Enables scroll bounce (rubber banding) effect on macOS. Default is false.
    - blinkFeatures String (optional) - A list of feature strings separated by ,, like CSSVariables,KeyboardEventKey to enable. The full list of supported feature strings can be found in the [RuntimeEnabledFeatures.json5](https://cs.chromium.org/chromium/src/third_party/WebKit/Source/platform/RuntimeEnabledFeatures.json5?l=62) file.
    - disableBlinkFeatures String (optional) - A list of feature strings separated by ,, like CSSVariables,KeyboardEventKey to disable. The full list of supported feature strings can be found in the [RuntimeEnabledFeatures.json5](https://cs.chromium.org/chromium/src/third_party/WebKit/Source/platform/RuntimeEnabledFeatures.json5?l=62) file.
    - defaultFontFamily Object (optional) - Sets the default font for the font-family.
      * standard String (optional) - Defaults to Times New Roman.
      * serif String (optional) - Defaults to Times New Roman.
      * sansSerif String (optional) - Defaults to Arial.
      * monospace String (optional) - Defaults to Courier New.
      * cursive String (optional) - Defaults to Script.
      * fantasy String (optional) - Defaults to Impact.
    - defaultFontSize Integer (optional) - Defaults to 16.
    - defaultMonospaceFontSize Integer (optional) - Defaults to 13.
    - minimumFontSize Integer (optional) - Defaults to 0.
    - defaultEncoding String (optional) - Defaults to ISO-8859-1.
    - backgroundThrottling Boolean (optional) - Whether to throttle animations and timers when the page becomes background. This also affects the [Page Visibility API][#page-visibility]. Defaults to true.
    - offscreen Boolean (optional) - Whether to enable offscreen rendering for the browser window. Defaults to false. See the [offscreen rendering tutorial](https://electron.atom.io/docs/tutorial/offscreen-rendering) for more details.
    - contextIsolation Boolean (optional) - Whether to run Electron APIs and the specified preload script in a separate JavaScript context. Defaults to false. The context that the preload script runs in will still have full access to the document and window globals but it will use its own set of JavaScript builtins (Array, Object, JSON, etc.) and will be isolated from any changes made to the global environment by the loaded page. The Electron API will only be available in the preload script and not the loaded page. This option should be used when loading potentially untrusted remote content to ensure the loaded content cannot tamper with the preload script and any Electron APIs being used. This option uses the same technique used by [Chrome Content Scripts](https://developer.chrome.com/extensions/content_scripts#execution-environment). You can access this context in the dev tools by selecting the ‘Electron Isolated Context’ entry in the combo box at the top of the Console tab. **Note:**This option is currently experimental and may change or be removed in future Electron releases.
    - nativeWindowOpen Boolean (optional) - Whether to use native window.open(). Defaults to false. **Note:** This option is currently experimental.
    - webviewTag Boolean (optional) - Whether to enable the [<webview> tag](https://electron.atom.io/docs/api/webview-tag). Defaults to the value of the nodeIntegration option. **Note:** The preload script configured for the <webview> will have node integration enabled when it is executed so you should ensure remote/untrusted content is not able to create a <webview> tag with a possibly malicious preload script. You can use the will-attach-webviewevent on [webContents](https://electron.atom.io/docs/api/web-contents) to strip away the preload script and to validate or alter the <webview>’s initial settings.

When setting minimum or maximum window size with minWidth/maxWidth/ minHeight/maxHeight, it only constrains the users. It won’t prevent you from passing a size that does not follow size constraints to setBounds/setSize or to the constructor of BrowserWindow.

The possible values and behaviors of the type option are platform dependent. Possible values are:

* On Linux, possible types are desktop, dock, toolbar, splash, notification.
* On macOS, possible types are desktop, textured.
  + The textured type adds metal gradient appearance (NSTexturedBackgroundWindowMask).
  + The desktop type places the window at the desktop background window level (kCGDesktopWindowLevel - 1). Note that desktop window will not receive focus, keyboard or mouse events, but you can use globalShortcut to receive input sparingly.
* On Windows, possible type is toolbar.

Instance Events

Objects created with new BrowserWindow emit the following events:

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

Event: ‘page-title-updated’

Returns:

* event Event
* title String

Emitted when the document changed its title, calling event.preventDefault() will prevent the native window’s title from changing.

Event: ‘close’

Returns:

* event Event

Emitted when the window is going to be closed. It’s emitted before the beforeunload and unload event of the DOM. Calling event.preventDefault() will cancel the close.

Usually you would want to use the beforeunload handler to decide whether the window should be closed, which will also be called when the window is reloaded. In Electron, returning any value other than undefined would cancel the close. For example:

window.onbeforeunload = (e) => {

console.log('I do not want to be closed')

*// Unlike usual browsers that a message box will be prompted to users, returning*

*// a non-void value will silently cancel the close.*

*// It is recommended to use the dialog API to let the user confirm closing the*

*// application.*

e.returnValue = false

}

Event: ‘closed’

Emitted when the window is closed. After you have received this event you should remove the reference to the window and avoid using it any more.

Event: ‘session-end’ Windows

Emitted when window session is going to end due to force shutdown or machine restart or session log off.

Event: ‘unresponsive’

Emitted when the web page becomes unresponsive.

Event: ‘responsive’

Emitted when the unresponsive web page becomes responsive again.

Event: ‘blur’

Emitted when the window loses focus.

Event: ‘focus’

Emitted when the window gains focus.

Event: ‘show’

Emitted when the window is shown.

Event: ‘hide’

Emitted when the window is hidden.

Event: ‘ready-to-show’

Emitted when the web page has been rendered (while not being shown) and window can be displayed without a visual flash.

Event: ‘maximize’

Emitted when window is maximized.

Event: ‘unmaximize’

Emitted when the window exits from a maximized state.

Event: ‘minimize’

Emitted when the window is minimized.

Event: ‘restore’

Emitted when the window is restored from a minimized state.

Event: ‘resize’

Emitted when the window is being resized.

Event: ‘move’

Emitted when the window is being moved to a new position.

**Note**: On macOS this event is just an alias of moved.

Event: ‘moved’ macOS

Emitted once when the window is moved to a new position.

Event: ‘enter-full-screen’

Emitted when the window enters a full-screen state.

Event: ‘leave-full-screen’

Emitted when the window leaves a full-screen state.

Event: ‘enter-html-full-screen’

Emitted when the window enters a full-screen state triggered by HTML API.

Event: ‘leave-html-full-screen’

Emitted when the window leaves a full-screen state triggered by HTML API.

Event: ‘app-command’ Windows

Returns:

* event Event
* command String

Emitted when an [App Command](https://msdn.microsoft.com/en-us/library/windows/desktop/ms646275(v=vs.85).aspx) is invoked. These are typically related to keyboard media keys or browser commands, as well as the “Back” button built into some mice on Windows.

Commands are lowercased, underscores are replaced with hyphens, and the APPCOMMAND\_ prefix is stripped off. e.g. APPCOMMAND\_BROWSER\_BACKWARD is emitted as browser-backward.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.on('app-command', (e, cmd) => {

*// Navigate the window back when the user hits their mouse back button*

**if** (cmd === 'browser-backward' && win.webContents.canGoBack()) {

win.webContents.goBack()

}

})

Event: ‘scroll-touch-begin’ macOS

Emitted when scroll wheel event phase has begun.

Event: ‘scroll-touch-end’ macOS

Emitted when scroll wheel event phase has ended.

Event: ‘scroll-touch-edge’ macOS

Emitted when scroll wheel event phase filed upon reaching the edge of element.

Event: ‘swipe’ macOS

Returns:

* event Event
* direction String

Emitted on 3-finger swipe. Possible directions are up, right, down, left.

Event: ‘sheet-begin’ macOS

Emitted when the window opens a sheet.

Event: ‘sheet-end’ macOS

Emitted when the window has closed a sheet.

Event: ‘new-window-for-tab’ macOS

Emitted when the native new tab button is clicked.

Static Methods

The BrowserWindow class has the following static methods:

BrowserWindow.getAllWindows()

Returns BrowserWindow[] - An array of all opened browser windows.

BrowserWindow.getFocusedWindow()

Returns BrowserWindow - The window that is focused in this application, otherwise returns null.

BrowserWindow.fromWebContents(webContents)

* webContents [WebContents](https://electron.atom.io/docs/api/web-contents)

Returns BrowserWindow - The window that owns the given webContents.

BrowserWindow.fromId(id)

* id Integer

Returns BrowserWindow - The window with the given id.

BrowserWindow.addExtension(path)

* path String

Adds Chrome extension located at path, and returns extension’s name.

The method will also not return if the extension’s manifest is missing or incomplete.

**Note:** This API cannot be called before the ready event of the app module is emitted.

BrowserWindow.removeExtension(name)

* name String

Remove a Chrome extension by name.

**Note:** This API cannot be called before the ready event of the app module is emitted.

BrowserWindow.getExtensions()

Returns Object - The keys are the extension names and each value is an Object containing name and versionproperties.

**Note:** This API cannot be called before the ready event of the app module is emitted.

BrowserWindow.addDevToolsExtension(path)

* path String

Adds DevTools extension located at path, and returns extension’s name.

The extension will be remembered so you only need to call this API once, this API is not for programming use. If you try to add an extension that has already been loaded, this method will not return and instead log a warning to the console.

The method will also not return if the extension’s manifest is missing or incomplete.

**Note:** This API cannot be called before the ready event of the app module is emitted.

BrowserWindow.removeDevToolsExtension(name)

* name String

Remove a DevTools extension by name.

**Note:** This API cannot be called before the ready event of the app module is emitted.

BrowserWindow.getDevToolsExtensions()

Returns Object - The keys are the extension names and each value is an Object containing name and versionproperties.

To check if a DevTools extension is installed you can run the following:

**const** {BrowserWindow} = require('electron')

**let** installed = BrowserWindow.getDevToolsExtensions().hasOwnProperty('devtron')

console.log(installed)

**Note:** This API cannot be called before the ready event of the app module is emitted.

Instance Properties

Objects created with new BrowserWindow have the following properties:

**const** {BrowserWindow} = require('electron')

*// In this example `win` is our instance*

**let** win = **new** BrowserWindow({width: 800, height: 600})

win.loadURL('https://github.com')

win.webContents

A WebContents object this window owns. All web page related events and operations will be done via it.

See the [webContents documentation](https://electron.atom.io/docs/api/web-contents) for its methods and events.

win.id

A Integer representing the unique ID of the window.

Instance Methods

Objects created with new BrowserWindow have the following instance methods:

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

win.destroy()

Force closing the window, the unload and beforeunload event won’t be emitted for the web page, and closeevent will also not be emitted for this window, but it guarantees the closed event will be emitted.

win.close()

Try to close the window. This has the same effect as a user manually clicking the close button of the window. The web page may cancel the close though. See the [close event](https://electron.atom.io/docs/all/#event-close).

win.focus()

Focuses on the window.

win.blur()

Removes focus from the window.

win.isFocused()

Returns Boolean - Whether the window is focused.

win.isDestroyed()

Returns Boolean - Whether the window is destroyed.

win.show()

Shows and gives focus to the window.

win.showInactive()

Shows the window but doesn’t focus on it.

win.hide()

Hides the window.

win.isVisible()

Returns Boolean - Whether the window is visible to the user.

win.isModal()

Returns Boolean - Whether current window is a modal window.

win.maximize()

Maximizes the window. This will also show (but not focus) the window if it isn’t being displayed already.

win.unmaximize()

Unmaximizes the window.

win.isMaximized()

Returns Boolean - Whether the window is maximized.

win.minimize()

Minimizes the window. On some platforms the minimized window will be shown in the Dock.

win.restore()

Restores the window from minimized state to its previous state.

win.isMinimized()

Returns Boolean - Whether the window is minimized.

win.setFullScreen(flag)

* flag Boolean

Sets whether the window should be in fullscreen mode.

win.isFullScreen()

Returns Boolean - Whether the window is in fullscreen mode.

win.setAspectRatio(aspectRatio[, extraSize]) macOS

* aspectRatio Float - The aspect ratio to maintain for some portion of the content view.
* extraSize [Size](https://electron.atom.io/docs/api/structures/size) - The extra size not to be included while maintaining the aspect ratio.

This will make a window maintain an aspect ratio. The extra size allows a developer to have space, specified in pixels, not included within the aspect ratio calculations. This API already takes into account the difference between a window’s size and its content size.

Consider a normal window with an HD video player and associated controls. Perhaps there are 15 pixels of controls on the left edge, 25 pixels of controls on the right edge and 50 pixels of controls below the player. In order to maintain a 16:9 aspect ratio (standard aspect ratio for HD @1920x1080) within the player itself we would call this function with arguments of 16/9 and [ 40, 50 ]. The second argument doesn’t care where the extra width and height are within the content view–only that they exist. Just sum any extra width and height areas you have within the overall content view.

win.previewFile(path[, displayName]) macOS

* path String - The absolute path to the file to preview with QuickLook. This is important as Quick Look uses the file name and file extension on the path to determine the content type of the file to open.
* displayName String (optional) - The name of the file to display on the Quick Look modal view. This is purely visual and does not affect the content type of the file. Defaults to path.

Uses [Quick Look](https://en.wikipedia.org/wiki/Quick_Look) to preview a file at a given path.

win.closeFilePreview() macOS

Closes the currently open [Quick Look](https://en.wikipedia.org/wiki/Quick_Look) panel.

win.setBounds(bounds[, animate])

* bounds [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)
* animate Boolean (optional) macOS

Resizes and moves the window to the supplied bounds

win.getBounds()

Returns [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)

win.setContentBounds(bounds[, animate])

* bounds [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)
* animate Boolean (optional) macOS

Resizes and moves the window’s client area (e.g. the web page) to the supplied bounds.

win.getContentBounds()

Returns [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)

win.setSize(width, height[, animate])

* width Integer
* height Integer
* animate Boolean (optional) macOS

Resizes the window to width and height.

win.getSize()

Returns Integer[] - Contains the window’s width and height.

win.setContentSize(width, height[, animate])

* width Integer
* height Integer
* animate Boolean (optional) macOS

Resizes the window’s client area (e.g. the web page) to width and height.

win.getContentSize()

Returns Integer[] - Contains the window’s client area’s width and height.

win.setMinimumSize(width, height)

* width Integer
* height Integer

Sets the minimum size of window to width and height.

win.getMinimumSize()

Returns Integer[] - Contains the window’s minimum width and height.

win.setMaximumSize(width, height)

* width Integer
* height Integer

Sets the maximum size of window to width and height.

win.getMaximumSize()

Returns Integer[] - Contains the window’s maximum width and height.

win.setResizable(resizable)

* resizable Boolean

Sets whether the window can be manually resized by user.

win.isResizable()

Returns Boolean - Whether the window can be manually resized by user.

win.setMovable(movable) macOS Windows

* movable Boolean

Sets whether the window can be moved by user. On Linux does nothing.

win.isMovable() macOS Windows

Returns Boolean - Whether the window can be moved by user.

On Linux always returns true.

win.setMinimizable(minimizable) macOS Windows

* minimizable Boolean

Sets whether the window can be manually minimized by user. On Linux does nothing.

win.isMinimizable() macOS Windows

Returns Boolean - Whether the window can be manually minimized by user

On Linux always returns true.

win.setMaximizable(maximizable) macOS Windows

* maximizable Boolean

Sets whether the window can be manually maximized by user. On Linux does nothing.

win.isMaximizable() macOS Windows

Returns Boolean - Whether the window can be manually maximized by user.

On Linux always returns true.

win.setFullScreenable(fullscreenable)

* fullscreenable Boolean

Sets whether the maximize/zoom window button toggles fullscreen mode or maximizes the window.

win.isFullScreenable()

Returns Boolean - Whether the maximize/zoom window button toggles fullscreen mode or maximizes the window.

win.setClosable(closable) macOS Windows

* closable Boolean

Sets whether the window can be manually closed by user. On Linux does nothing.

win.isClosable() macOS Windows

Returns Boolean - Whether the window can be manually closed by user.

On Linux always returns true.

win.setAlwaysOnTop(flag[, level][, relativeLevel])

* flag Boolean
* level String (optional) macOS - Values include normal, floating, torn-off-menu, modal-panel, main-menu, status, pop-up-menu, screen-saver, and  (Deprecated). The default is floating. See the [macOS docs](https://developer.apple.com/reference/appkit/nswindow/1664726-window_levels)for more details.
* relativeLevel Integer (optional) macOS - The number of layers higher to set this window relative to the given level. The default is 0. Note that Apple discourages setting levels higher than 1 above screen-saver.

Sets whether the window should show always on top of other windows. After setting this, the window is still a normal window, not a toolbox window which can not be focused on.

win.isAlwaysOnTop()

Returns Boolean - Whether the window is always on top of other windows.

win.center()

Moves window to the center of the screen.

win.setPosition(x, y[, animate])

* x Integer
* y Integer
* animate Boolean (optional) macOS

Moves window to x and y.

win.getPosition()

Returns Integer[] - Contains the window’s current position.

win.setTitle(title)

* title String

Changes the title of native window to title.

win.getTitle()

Returns String - The title of the native window.

**Note:** The title of web page can be different from the title of the native window.

win.setSheetOffset(offsetY[, offsetX]) macOS

* offsetY Float
* offsetX Float (optional)

Changes the attachment point for sheets on macOS. By default, sheets are attached just below the window frame, but you may want to display them beneath a HTML-rendered toolbar. For example:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

**let** toolbarRect = document.getElementById('toolbar').getBoundingClientRect()

win.setSheetOffset(toolbarRect.height)

win.flashFrame(flag)

* flag Boolean

Starts or stops flashing the window to attract user’s attention.

win.setSkipTaskbar(skip)

* skip Boolean

Makes the window not show in the taskbar.

win.setKiosk(flag)

* flag Boolean

Enters or leaves the kiosk mode.

win.isKiosk()

Returns Boolean - Whether the window is in kiosk mode.

win.getNativeWindowHandle()

Returns Buffer - The platform-specific handle of the window.

The native type of the handle is HWND on Windows, NSView\* on macOS, and Window (unsigned long) on Linux.

win.hookWindowMessage(message, callback) Windows

* message Integer
* callback Function

Hooks a windows message. The callback is called when the message is received in the WndProc.

win.isWindowMessageHooked(message) Windows

* message Integer

Returns Boolean - true or false depending on whether the message is hooked.

win.unhookWindowMessage(message) Windows

* message Integer

Unhook the window message.

win.unhookAllWindowMessages() Windows

Unhooks all of the window messages.

win.setRepresentedFilename(filename) macOS

* filename String

Sets the pathname of the file the window represents, and the icon of the file will show in window’s title bar.

win.getRepresentedFilename() macOS

Returns String - The pathname of the file the window represents.

win.setDocumentEdited(edited) macOS

* edited Boolean

Specifies whether the window’s document has been edited, and the icon in title bar will become gray when set to true.

win.isDocumentEdited() macOS

Returns Boolean - Whether the window’s document has been edited.

win.focusOnWebView()

win.blurWebView()

win.capturePage([rect, ]callback)

* rect [Rectangle](https://electron.atom.io/docs/api/structures/rectangle) (optional) - The bounds to capture
* callback Function
  + image [NativeImage](https://electron.atom.io/docs/api/native-image)

Same as webContents.capturePage([rect, ]callback).

win.loadURL(url[, options])

* url String
* options Object (optional)
  + httpReferrer String (optional) - A HTTP Referrer url.
  + userAgent String (optional) - A user agent originating the request.
  + extraHeaders String (optional) - Extra headers separated by “\n”
  + postData ([UploadRawData[]](https://electron.atom.io/docs/api/structures/upload-raw-data) | [UploadFile[]](https://electron.atom.io/docs/api/structures/upload-file) | [UploadFileSystem[]](https://electron.atom.io/docs/api/structures/upload-file-system) | [UploadBlob[]](https://electron.atom.io/docs/api/structures/upload-blob)) - (optional)
  + baseURLForDataURL String (optional) - Base url (with trailing path separator) for files to be loaded by the data url. This is needed only if the specified url is a data url and needs to load other files.

Same as webContents.loadURL(url[, options]).

The url can be a remote address (e.g. http://) or a path to a local HTML file using the file:// protocol.

To ensure that file URLs are properly formatted, it is recommended to use Node’s [url.format](https://nodejs.org/api/url.html#url_url_format_urlobject) method:

**let** url = require('url').format({

protocol: 'file',

slashes: true,

pathname: require('path').join(\_\_dirname, 'index.html')

})

win.loadURL(url)

You can load a URL using a POST request with URL-encoded data by doing the following:

win.loadURL('http://localhost:8000/post', {

postData: [{

type: 'rawData',

bytes: Buffer.from('hello=world')

}],

extraHeaders: 'Content-Type: application/x-www-form-urlencoded'

})

win.reload()

Same as webContents.reload.

win.setMenu(menu) Linux Windows

* menu Menu | null

Sets the menu as the window’s menu bar, setting it to null will remove the menu bar.

win.setProgressBar(progress[, options])

* progress Double
* options Object (optional)
  + mode String Windows - Mode for the progress bar. Can be none, normal, indeterminate, error, or paused.

Sets progress value in progress bar. Valid range is [0, 1.0].

Remove progress bar when progress < 0; Change to indeterminate mode when progress > 1.

On Linux platform, only supports Unity desktop environment, you need to specify the \*.desktop file name to desktopName field in package.json. By default, it will assume app.getName().desktop.

On Windows, a mode can be passed. Accepted values are none, normal, indeterminate, error, and paused. If you call setProgressBar without a mode set (but with a value within the valid range), normal will be assumed.

win.setOverlayIcon(overlay, description) Windows

* overlay [NativeImage](https://electron.atom.io/docs/api/native-image) - the icon to display on the bottom right corner of the taskbar icon. If this parameter is null, the overlay is cleared
* description String - a description that will be provided to Accessibility screen readers

Sets a 16 x 16 pixel overlay onto the current taskbar icon, usually used to convey some sort of application status or to passively notify the user.

win.setHasShadow(hasShadow) macOS

* hasShadow Boolean

Sets whether the window should have a shadow. On Windows and Linux does nothing.

win.hasShadow() macOS

Returns Boolean - Whether the window has a shadow.

On Windows and Linux always returns true.

win.setThumbarButtons(buttons) Windows

* buttons [ThumbarButton[]](https://electron.atom.io/docs/api/structures/thumbar-button)

Returns Boolean - Whether the buttons were added successfully

Add a thumbnail toolbar with a specified set of buttons to the thumbnail image of a window in a taskbar button layout. Returns a Boolean object indicates whether the thumbnail has been added successfully.

The number of buttons in thumbnail toolbar should be no greater than 7 due to the limited room. Once you setup the thumbnail toolbar, the toolbar cannot be removed due to the platform’s limitation. But you can call the API with an empty array to clean the buttons.

The buttons is an array of Button objects:

* Button Object
  + icon [NativeImage](https://electron.atom.io/docs/api/native-image) - The icon showing in thumbnail toolbar.
  + click Function
  + tooltip String (optional) - The text of the button’s tooltip.
  + flags String[] (optional) - Control specific states and behaviors of the button. By default, it is ['enabled'].

The flags is an array that can include following Strings:

* enabled - The button is active and available to the user.
* disabled - The button is disabled. It is present, but has a visual state indicating it will not respond to user action.
* dismissonclick - When the button is clicked, the thumbnail window closes immediately.
* nobackground - Do not draw a button border, use only the image.
* hidden - The button is not shown to the user.
* noninteractive - The button is enabled but not interactive; no pressed button state is drawn. This value is intended for instances where the button is used in a notification.

win.setThumbnailClip(region) Windows

* region [Rectangle](https://electron.atom.io/docs/api/structures/rectangle) - Region of the window

Sets the region of the window to show as the thumbnail image displayed when hovering over the window in the taskbar. You can reset the thumbnail to be the entire window by specifying an empty region: {x: 0, y: 0,width: 0, height: 0}.

win.setThumbnailToolTip(toolTip) Windows

* toolTip String

Sets the toolTip that is displayed when hovering over the window thumbnail in the taskbar.

win.setAppDetails(options) Windows

* options Object
  + appId String (optional) - Window’s [App User Model ID](https://msdn.microsoft.com/en-us/library/windows/desktop/dd391569(v=vs.85).aspx). It has to be set, otherwise the other options will have no effect.
  + appIconPath String (optional) - Window’s [Relaunch Icon](https://msdn.microsoft.com/en-us/library/windows/desktop/dd391573(v=vs.85).aspx).
  + appIconIndex Integer (optional) - Index of the icon in appIconPath. Ignored when appIconPath is not set. Default is 0.
  + relaunchCommand String (optional) - Window’s [Relaunch Command](https://msdn.microsoft.com/en-us/library/windows/desktop/dd391571(v=vs.85).aspx).
  + relaunchDisplayName String (optional) - Window’s [Relaunch Display Name](https://msdn.microsoft.com/en-us/library/windows/desktop/dd391572(v=vs.85).aspx).

Sets the properties for the window’s taskbar button.

**Note:** relaunchCommand and relaunchDisplayName must always be set together. If one of those properties is not set, then neither will be used.

win.showDefinitionForSelection() macOS

Same as webContents.showDefinitionForSelection().

win.setIcon(icon) Windows Linux

* icon [NativeImage](https://electron.atom.io/docs/api/native-image)

Changes window icon.

win.setAutoHideMenuBar(hide)

* hide Boolean

Sets whether the window menu bar should hide itself automatically. Once set the menu bar will only show when users press the single Alt key.

If the menu bar is already visible, calling setAutoHideMenuBar(true) won’t hide it immediately.

win.isMenuBarAutoHide()

Returns Boolean - Whether menu bar automatically hides itself.

win.setMenuBarVisibility(visible) Windows Linux

* visible Boolean

Sets whether the menu bar should be visible. If the menu bar is auto-hide, users can still bring up the menu bar by pressing the single Alt key.

win.isMenuBarVisible()

Returns Boolean - Whether the menu bar is visible.

win.setVisibleOnAllWorkspaces(visible)

* visible Boolean

Sets whether the window should be visible on all workspaces.

**Note:** This API does nothing on Windows.

win.isVisibleOnAllWorkspaces()

Returns Boolean - Whether the window is visible on all workspaces.

**Note:** This API always returns false on Windows.

win.setIgnoreMouseEvents(ignore)

* ignore Boolean

Makes the window ignore all mouse events.

All mouse events happened in this window will be passed to the window below this window, but if this window has focus, it will still receive keyboard events.

win.setContentProtection(enable) macOS Windows

* enable Boolean

Prevents the window contents from being captured by other apps.

On macOS it sets the NSWindow’s sharingType to NSWindowSharingNone. On Windows it calls SetWindowDisplayAffinity with WDA\_MONITOR.

win.setFocusable(focusable) Windows

* focusable Boolean

Changes whether the window can be focused.

win.setParentWindow(parent) Linux macOS

* parent BrowserWindow

Sets parent as current window’s parent window, passing null will turn current window into a top-level window.

win.getParentWindow()

Returns BrowserWindow - The parent window.

win.getChildWindows()

Returns BrowserWindow[] - All child windows.

win.setAutoHideCursor(autoHide) macOS

* autoHide Boolean

Controls whether to hide cursor when typing.

win.setVibrancy(type) macOS

* type String - Can be appearance-based, light, dark, titlebar, selection, menu, popover, sidebar, medium-light or ultra-dark. See the [macOS documentation](https://developer.apple.com/reference/appkit/nsvisualeffectview?language=objc) for more details.

Adds a vibrancy effect to the browser window. Passing null or an empty string will remove the vibrancy effect on the window.

win.setTouchBar(touchBar) macOS Experimental

* touchBar TouchBar

Sets the touchBar layout for the current window. Specifying null or undefined clears the touch bar. This method only has an effect if the machine has a touch bar and is running on macOS 10.12.1+.

**Note:** The TouchBar API is currently experimental and may change or be removed in future Electron releases.

win.setBrowserView(browserView) Experimental

* browserView [BrowserView](https://electron.atom.io/docs/api/browser-view)

**Note:** The BrowserView API is currently experimental and may change or be removed in future Electron releases.

Supported Chrome Command Line Switches

Command line switches supported by Electron.

You can use [app.commandLine.appendSwitch](https://electron.atom.io/docs/api/app#appcommandlineappendswitchswitch-value) to append them in your app’s main script before the [ready](https://electron.atom.io/docs/api/app#event-ready) event of the [app](https://electron.atom.io/docs/api/app) module is emitted:

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

app.commandLine.appendSwitch('remote-debugging-port', '8315')

app.commandLine.appendSwitch('host-rules', 'MAP \* 127.0.0.1')

app.on('ready', () => {

*// Your code here*

})

–ignore-connections-limit=domains

Ignore the connections limit for domains list separated by ,.

–disable-http-cache

Disables the disk cache for HTTP requests.

–disable-http2

Disable HTTP/2 and SPDY/3.1 protocols.

–inspect=port and –inspect-brk=port

Debug-related flags, see the [Debugging the Main Process](https://electron.atom.io/docs/tutorial/debugging-main-process) guide for details.

–remote-debugging-port=port

Enables remote debugging over HTTP on the specified port.

–disk-cache-size=size

Forces the maximum disk space to be used by the disk cache, in bytes.

–js-flags=flags

Specifies the flags passed to the Node JS engine. It has to be passed when starting Electron if you want to enable the flags in the main process.

$ electron --js-flags="--harmony\_proxies --harmony\_collections" your-app

See the [Node documentation](https://nodejs.org/api/cli.html) or run node --help in your terminal for a list of available flags. Additionally, run node --v8-options to see a list of flags that specifically refer to Node’s V8 JavaScript engine.

–proxy-server=address:port

Use a specified proxy server, which overrides the system setting. This switch only affects requests with HTTP protocol, including HTTPS and WebSocket requests. It is also noteworthy that not all proxy servers support HTTPS and WebSocket requests.

–proxy-bypass-list=hosts

Instructs Electron to bypass the proxy server for the given semi-colon-separated list of hosts. This flag has an effect only if used in tandem with --proxy-server.

For example:

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

app.commandLine.appendSwitch('proxy-bypass-list', '<local>;\*.google.com;\*foo.com;1.2.3.4:5678')

Will use the proxy server for all hosts except for local addresses (localhost, 127.0.0.1 etc.), google.comsubdomains, hosts that contain the suffix foo.com and anything at 1.2.3.4:5678.

–proxy-pac-url=url

Uses the PAC script at the specified url.

–no-proxy-server

Don’t use a proxy server and always make direct connections. Overrides any other proxy server flags that are passed.

–host-rules=rules

A comma-separated list of rules that control how hostnames are mapped.

For example:

* MAP \* 127.0.0.1 Forces all hostnames to be mapped to 127.0.0.1
* MAP \*.google.com proxy Forces all google.com subdomains to be resolved to “proxy”.
* MAP test.com [::1]:77 Forces “test.com” to resolve to IPv6 loopback. Will also force the port of the resulting socket address to be 77.
* MAP \* baz, EXCLUDE www.google.com Remaps everything to “baz”, except for “www.google.com”.

These mappings apply to the endpoint host in a net request (the TCP connect and host resolver in a direct connection, and the CONNECT in an HTTP proxy connection, and the endpoint host in a SOCKS proxy connection).

–host-resolver-rules=rules

Like --host-rules but these rules only apply to the host resolver.

–auth-server-whitelist=url

A comma-separated list of servers for which integrated authentication is enabled.

For example:

--auth-server-whitelist='\*example.com, \*foobar.com, \*baz'

then any url ending with example.com, foobar.com, baz will be considered for integrated authentication. Without \* prefix the url has to match exactly.

–auth-negotiate-delegate-whitelist=url

A comma-separated list of servers for which delegation of user credentials is required. Without \* prefix the url has to match exactly.

–ignore-certificate-errors

Ignores certificate related errors.

–ppapi-flash-path=path

Sets the path of the pepper flash plugin.

–ppapi-flash-version=version

Sets the version of the pepper flash plugin.

–log-net-log=path

Enables net log events to be saved and writes them to path.

–disable-renderer-backgrounding

Prevents Chromium from lowering the priority of invisible pages’ renderer processes.

This flag is global to all renderer processes, if you only want to disable throttling in one window, you can take the hack of [playing silent audio](https://github.com/atom/atom/pull/9485/files).

–enable-logging

Prints Chromium’s logging into console.

This switch can not be used in app.commandLine.appendSwitch since it is parsed earlier than user’s app is loaded, but you can set the ELECTRON\_ENABLE\_LOGGING environment variable to achieve the same effect.

–v=log\_level

Gives the default maximal active V-logging level; 0 is the default. Normally positive values are used for V-logging levels.

This switch only works when --enable-logging is also passed.

–vmodule=pattern

Gives the per-module maximal V-logging levels to override the value given by --v. E.g. my\_module=2,foo\*=3would change the logging level for all code in source files my\_module.\* and foo\*.\*.

Any pattern containing a forward or backward slash will be tested against the whole pathname and not just the module. E.g. \*/foo/bar/\*=2 would change the logging level for all code in the source files under a foo/bardirectory.

This switch only works when --enable-logging is also passed.

Class: ClientRequest

Make HTTP/HTTPS requests.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

ClientRequest implements the [Writable Stream](https://nodejs.org/api/stream.html#stream_writable_streams) interface and is therefore an [EventEmitter](https://nodejs.org/api/events.html#events_class_eventemitter).

new ClientRequest(options)

* options (Object | String) - If options is a String, it is interpreted as the request URL. If it is an object, it is expected to fully specify an HTTP request via the following properties:
  + method String (optional) - The HTTP request method. Defaults to the GET method.
  + url String (optional) - The request URL. Must be provided in the absolute form with the protocol scheme specified as http or https.
  + session Object (optional) - The [Session](https://electron.atom.io/docs/api/session) instance with which the request is associated.
  + partition String (optional) - The name of the [partition](https://electron.atom.io/docs/api/session) with which the request is associated. Defaults to the empty string. The session option prevails on partition. Thus if a session is explicitly specified, partition is ignored.
  + protocol String (optional) - The protocol scheme in the form ‘scheme:’. Currently supported values are ‘http:’ or ‘https:’. Defaults to ‘http:’.
  + host String (optional) - The server host provided as a concatenation of the hostname and the port number ‘hostname:port’
  + hostname String (optional) - The server host name.
  + port Integer (optional) - The server’s listening port number.
  + path String (optional) - The path part of the request URL.
  + redirect String (optional) - The redirect mode for this request. Should be one of follow, error or manual. Defaults to follow. When mode is error, any redirection will be aborted. When mode is manualthe redirection will be deferred until [request.followRedirect](https://electron.atom.io/docs/all/#requestfollowRedirect) is invoked. Listen for the [redirect](https://electron.atom.io/docs/all/#event-redirect) event in this mode to get more details about the redirect request.

options properties such as protocol, host, hostname, port and path strictly follow the Node.js model as described in the [URL](https://nodejs.org/api/url.html) module.

For instance, we could have created the same request to ‘github.com’ as follows:

const request = net.request({

method: 'GET',

protocol: 'https:',

hostname: 'github.com',

port: 443,

path: '/'

})

Instance Events

Event: ‘response’

Returns:

* response IncomingMessage - An object representing the HTTP response message.

Event: ‘login’

Returns:

* authInfo Object
  + isProxy Boolean
  + scheme String
  + host String
  + port Integer
  + realm String
* callback Function
  + username String
  + password String

Emitted when an authenticating proxy is asking for user credentials.

The callback function is expected to be called back with user credentials:

* username String
* password String

request.on('login', (authInfo, callback) => {

callback('username', 'password')

})

Providing empty credentials will cancel the request and report an authentication error on the response object:

request.on('response', (response) => {

console.log(`STATUS: ${response.statusCode}`);

response.on('error', (error) => {

console.log(`ERROR: ${JSON.stringify(error)}`)

})

})

request.on('login', (authInfo, callback) => {

callback()

})

Event: ‘finish’

Emitted just after the last chunk of the request’s data has been written into the request object.

Event: ‘abort’

Emitted when the request is aborted. The abort event will not be fired if the request is already closed.

Event: ‘error’

Returns:

* error Error - an error object providing some information about the failure.

Emitted when the net module fails to issue a network request. Typically when the request object emits an error event, a close event will subsequently follow and no response object will be provided.

Event: ‘close’

Emitted as the last event in the HTTP request-response transaction. The close event indicates that no more events will be emitted on either the request or response objects.

Event: ‘redirect’

Returns:

* statusCode Integer
* method String
* redirectUrl String
* responseHeaders Object

Emitted when there is redirection and the mode is manual. Calling [request.followRedirect](https://electron.atom.io/docs/all/#requestfollowRedirect) will continue with the redirection.

Instance Properties

request.chunkedEncoding

A Boolean specifying whether the request will use HTTP chunked transfer encoding or not. Defaults to false. The property is readable and writable, however it can be set only before the first write operation as the HTTP headers are not yet put on the wire. Trying to set the chunkedEncoding property after the first write will throw an error.

Using chunked encoding is strongly recommended if you need to send a large request body as data will be streamed in small chunks instead of being internally buffered inside Electron process memory.

Instance Methods

request.setHeader(name, value)

* name String - An extra HTTP header name.
* value Object - An extra HTTP header value.

Adds an extra HTTP header. The header name will issued as it is without lowercasing. It can be called only before first write. Calling this method after the first write will throw an error. If the passed value is not a String, its toString() method will be called to obtain the final value.

request.getHeader(name)

* name String - Specify an extra header name.

Returns Object - The value of a previously set extra header name.

request.removeHeader(name)

* name String - Specify an extra header name.

Removes a previously set extra header name. This method can be called only before first write. Trying to call it after the first write will throw an error.

request.write(chunk[, encoding][, callback])

* chunk (String | Buffer) - A chunk of the request body’s data. If it is a string, it is converted into a Buffer using the specified encoding.
* encoding String (optional) - Used to convert string chunks into Buffer objects. Defaults to ‘utf-8’.
* callback Function (optional) - Called after the write operation ends.

callback is essentially a dummy function introduced in the purpose of keeping similarity with the Node.js API. It is called asynchronously in the next tick after chunk content have been delivered to the Chromium networking layer. Contrary to the Node.js implementation, it is not guaranteed that chunk content have been flushed on the wire before callback is called.

Adds a chunk of data to the request body. The first write operation may cause the request headers to be issued on the wire. After the first write operation, it is not allowed to add or remove a custom header.

request.end([chunk][, encoding][, callback])

* chunk (String | Buffer) (optional)
* encoding String (optional)
* callback Function (optional)

Sends the last chunk of the request data. Subsequent write or end operations will not be allowed. The finishevent is emitted just after the end operation.

request.abort()

Cancels an ongoing HTTP transaction. If the request has already emitted the close event, the abort operation will have no effect. Otherwise an ongoing event will emit abort and close events. Additionally, if there is an ongoing response object,it will emit the aborted event.

request.followRedirect()

Continues any deferred redirection request when the redirection mode is manual.

clipboard

Perform copy and paste operations on the system clipboard.

Process: [Main](https://electron.atom.io/docs/glossary#main-process), [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

The following example shows how to write a string to the clipboard:

**const** {clipboard} = require('electron')

clipboard.writeText('Example String')

On X Window systems, there is also a selection clipboard. To manipulate it you need to pass selection to each method:

**const** {clipboard} = require('electron')

clipboard.writeText('Example String', 'selection')

console.log(clipboard.readText('selection'))

Methods

The clipboard module has the following methods:

**Note:** Experimental APIs are marked as such and could be removed in future.

clipboard.readText([type])

* type String (optional)

Returns String - The content in the clipboard as plain text.

clipboard.writeText(text[, type])

* text String
* type String (optional)

Writes the text into the clipboard as plain text.

clipboard.readHTML([type])

* type String (optional)

Returns String - The content in the clipboard as markup.

clipboard.writeHTML(markup[, type])

* markup String
* type String (optional)

Writes markup to the clipboard.

clipboard.readImage([type])

* type String (optional)

Returns [NativeImage](https://electron.atom.io/docs/api/native-image) - The image content in the clipboard.

clipboard.writeImage(image[, type])

* image [NativeImage](https://electron.atom.io/docs/api/native-image)
* type String (optional)

Writes image to the clipboard.

clipboard.readRTF([type])

* type String (optional)

Returns String - The content in the clipboard as RTF.

clipboard.writeRTF(text[, type])

* text String
* type String (optional)

Writes the text into the clipboard in RTF.

clipboard.readBookmark() macOS Windows

Returns Object:

* title String
* url String

Returns an Object containing title and url keys representing the bookmark in the clipboard. The title and url values will be empty strings when the bookmark is unavailable.

clipboard.writeBookmark(title, url[, type]) macOS Windows

* title String
* url String
* type String (optional)

Writes the title and url into the clipboard as a bookmark.

**Note:** Most apps on Windows don’t support pasting bookmarks into them so you can use clipboard.write to write both a bookmark and fallback text to the clipboard.

clipboard.write({

text: 'https://electron.atom.io',

bookmark: 'Electron Homepage'

})

clipboard.readFindText() macOS

Returns String - The text on the find pasteboard. This method uses synchronous IPC when called from the renderer process. The cached value is reread from the find pasteboard whenever the application is activated.

clipboard.writeFindText(text) macOS

* text String

Writes the text into the find pasteboard as plain text. This method uses synchronous IPC when called from the renderer process.

clipboard.clear([type])

* type String (optional)

Clears the clipboard content.

clipboard.availableFormats([type])

* type String (optional)

Returns String[] - An array of supported formats for the clipboard type.

clipboard.has(format[, type]) Experimental

* format String
* type String (optional)

Returns Boolean - Whether the clipboard supports the specified format.

**const** {clipboard} = require('electron')

console.log(clipboard.has('<p>selection</p>'))

clipboard.read(format) Experimental

* format String

Returns String - Reads format type from the clipboard.

clipboard.readBuffer(format) Experimental

* format String

Returns Buffer - Reads format type from the clipboard.

clipboard.writeBuffer(format, buffer[, type]) Experimental

* format String
* buffer Buffer
* type String (optional)

Writes the buffer into the clipboard as format.

clipboard.write(data[, type])

* data Object
  + text String (optional)
  + html String (optional)
  + image [NativeImage](https://electron.atom.io/docs/api/native-image) (optional)
  + rtf String (optional)
  + bookmark String (optional) - The title of the url at text.
* type String (optional)

**const** {clipboard} = require('electron')

clipboard.write({text: 'test', html: '<b>test</b>'})

Writes data to the clipboard.

contentTracing

Collect tracing data from Chromium’s content module for finding performance bottlenecks and slow operations.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

This module does not include a web interface so you need to open chrome://tracing/ in a Chrome browser and load the generated file to view the result.

**Note:** You should not use this module until the ready event of the app module is emitted.

**const** {app, contentTracing} = require('electron')

app.on('ready', () => {

**const** options = {

categoryFilter: '\*',

traceOptions: 'record-until-full,enable-sampling'

}

contentTracing.startRecording(options, () => {

console.log('Tracing started')

setTimeout(() => {

contentTracing.stopRecording('', (path) => {

console.log('Tracing data recorded to ' + path)

})

}, 5000)

})

})

Methods

The contentTracing module has the following methods:

contentTracing.getCategories(callback)

* callback Function
  + categories String[]

Get a set of category groups. The category groups can change as new code paths are reached.

Once all child processes have acknowledged the getCategories request the callback is invoked with an array of category groups.

contentTracing.startRecording(options, callback)

* options Object
  + categoryFilter String
  + traceOptions String
* callback Function

Start recording on all processes.

Recording begins immediately locally and asynchronously on child processes as soon as they receive the EnableRecording request. The callback will be called once all child processes have acknowledged the startRecording request.

categoryFilter is a filter to control what category groups should be traced. A filter can have an optional -prefix to exclude category groups that contain a matching category. Having both included and excluded category patterns in the same list is not supported.

Examples:

* test\_MyTest\*,
* test\_MyTest\*,test\_OtherStuff,
* "-excluded\_category1,-excluded\_category2

traceOptions controls what kind of tracing is enabled, it is a comma-delimited list. Possible options are:

* record-until-full
* record-continuously
* trace-to-console
* enable-sampling
* enable-systrace

The first 3 options are trace recording modes and hence mutually exclusive. If more than one trace recording modes appear in the traceOptions string, the last one takes precedence. If none of the trace recording modes are specified, recording mode is record-until-full.

The trace option will first be reset to the default option (record\_mode set to record-until-full, enable\_sampling and enable\_systrace set to false) before options parsed from traceOptions are applied on it.

contentTracing.stopRecording(resultFilePath, callback)

* resultFilePath String
* callback Function
  + resultFilePath String

Stop recording on all processes.

Child processes typically cache trace data and only rarely flush and send trace data back to the main process. This helps to minimize the runtime overhead of tracing since sending trace data over IPC can be an expensive operation. So, to end tracing, we must asynchronously ask all child processes to flush any pending trace data.

Once all child processes have acknowledged the stopRecording request, callback will be called with a file that contains the traced data.

Trace data will be written into resultFilePath if it is not empty or into a temporary file. The actual file path will be passed to callback if it’s not null.

contentTracing.startMonitoring(options, callback)

* options Object
  + categoryFilter String
  + traceOptions String
* callback Function

Start monitoring on all processes.

Monitoring begins immediately locally and asynchronously on child processes as soon as they receive the startMonitoring request.

Once all child processes have acknowledged the startMonitoring request the callback will be called.

contentTracing.stopMonitoring(callback)

* callback Function

Stop monitoring on all processes.

Once all child processes have acknowledged the stopMonitoring request the callback is called.

contentTracing.captureMonitoringSnapshot(resultFilePath, callback)

* resultFilePath String
* callback Function
  + resultFilePath String

Get the current monitoring traced data.

Child processes typically cache trace data and only rarely flush and send trace data back to the main process. This is because it may be an expensive operation to send the trace data over IPC and we would like to avoid unneeded runtime overhead from tracing. So, to end tracing, we must asynchronously ask all child processes to flush any pending trace data.

Once all child processes have acknowledged the captureMonitoringSnapshot request the callback will be called with a file that contains the traced data.

contentTracing.getTraceBufferUsage(callback)

* callback Function
  + value Number
  + percentage Number

Get the maximum usage across processes of trace buffer as a percentage of the full state. When the TraceBufferUsage value is determined the callback is called.

Class: Cookies

Query and modify a session’s cookies.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

Instances of the Cookies class are accessed by using cookies property of a Session.

For example:

**const** {session} = require('electron')

*// Query all cookies.*

session.defaultSession.cookies.get({}, (error, cookies) => {

console.log(error, cookies)

})

*// Query all cookies associated with a specific url.*

session.defaultSession.cookies.get({url: 'http://www.github.com'}, (error, cookies) => {

console.log(error, cookies)

})

*// Set a cookie with the given cookie data;*

*// may overwrite equivalent cookies if they exist.*

**const** cookie = {url: 'http://www.github.com', name: 'dummy\_name', value: 'dummy'}

session.defaultSession.cookies.set(cookie, (error) => {

**if** (error) console.error(error)

})

Instance Events

The following events are available on instances of Cookies:

Event: ‘changed’

* event Event
* cookie [Cookie](https://electron.atom.io/docs/api/structures/cookie) - The cookie that was changed
* cause String - The cause of the change with one of the following values:
  + explicit - The cookie was changed directly by a consumer’s action.
  + overwrite - The cookie was automatically removed due to an insert operation that overwrote it.
  + expired - The cookie was automatically removed as it expired.
  + evicted - The cookie was automatically evicted during garbage collection.
  + expired-overwrite - The cookie was overwritten with an already-expired expiration date.
* removed Boolean - true if the cookie was removed, false otherwise.

Emitted when a cookie is changed because it was added, edited, removed, or expired.

Instance Methods

The following methods are available on instances of Cookies:

cookies.get(filter, callback)

* filter Object
  + url String (optional) - Retrieves cookies which are associated with url. Empty implies retrieving cookies of all urls.
  + name String (optional) - Filters cookies by name.
  + domain String (optional) - Retrieves cookies whose domains match or are subdomains of domains
  + path String (optional) - Retrieves cookies whose path matches path.
  + secure Boolean (optional) - Filters cookies by their Secure property.
  + session Boolean (optional) - Filters out session or persistent cookies.
* callback Function
  + error Error
  + cookies [Cookie[]](https://electron.atom.io/docs/api/structures/cookie) - an array of cookie objects.

Sends a request to get all cookies matching details, callback will be called with callback(error, cookies) on complete.

cookies.set(details, callback)

* details Object
  + url String - The url to associate the cookie with.
  + name String (optional) - The name of the cookie. Empty by default if omitted.
  + value String (optional) - The value of the cookie. Empty by default if omitted.
  + domain String (optional) - The domain of the cookie. Empty by default if omitted.
  + path String (optional) - The path of the cookie. Empty by default if omitted.
  + secure Boolean (optional) - Whether the cookie should be marked as Secure. Defaults to false.
  + httpOnly Boolean (optional) - Whether the cookie should be marked as HTTP only. Defaults to false.
  + expirationDate Double (optional) - The expiration date of the cookie as the number of seconds since the UNIX epoch. If omitted then the cookie becomes a session cookie and will not be retained between sessions.
* callback Function
  + error Error

Sets a cookie with details, callback will be called with callback(error) on complete.

cookies.remove(url, name, callback)

* url String - The URL associated with the cookie.
* name String - The name of cookie to remove.
* callback Function

Removes the cookies matching url and name, callback will called with callback() on complete.

cookies.flushStore(callback)

* callback Function

Writes any unwritten cookies data to disk.

crashReporter

Submit crash reports to a remote server.

Process: [Main](https://electron.atom.io/docs/glossary#main-process), [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

The following is an example of automatically submitting a crash report to a remote server:

**const** {crashReporter} = require('electron')

crashReporter.start({

productName: 'YourName',

companyName: 'YourCompany',

submitURL: 'https://your-domain.com/url-to-submit',

uploadToServer: true

})

For setting up a server to accept and process crash reports, you can use following projects:

* [socorro](https://github.com/mozilla/socorro)
* [mini-breakpad-server](https://github.com/electron/mini-breakpad-server)

Crash reports are saved locally in an application-specific temp directory folder. For a productName of YourName, crash reports will be stored in a folder named YourName Crashes inside the temp directory. You can customize this temp directory location for your app by calling the app.setPath('temp', '/my/custom/temp') API before starting the crash reporter.

Methods

The crashReporter module has the following methods:

crashReporter.start(options)

* options Object
  + companyName String (optional)
  + submitURL String - URL that crash reports will be sent to as POST.
  + productName String (optional) - Defaults to app.getName().
  + uploadToServer Boolean (optional) - Whether crash reports should be sent to the server Default is true.
  + ignoreSystemCrashHandler Boolean (optional) - Default is false.
  + extra Object (optional) - An object you can define that will be sent along with the report. Only string properties are sent correctly. Nested objects are not supported and the property names and values must be less than 64 characters long.

You are required to call this method before using any other crashReporter APIs and in each process (main/renderer) from which you want to collect crash reports. You can pass different options to crashReporter.start when calling from different processes.

**Note** Child processes created via the child\_process module will not have access to the Electron modules. Therefore, to collect crash reports from them, use process.crashReporter.start instead. Pass the same options as above along with an additional one called crashesDirectory that should point to a directory to store the crash reports temporarily. You can test this out by calling process.crash() to crash the child process.

**Note:** To collect crash reports from child process in Windows, you need to add this extra code as well. This will start the process that will monitor and send the crash reports. Replace submitURL, productName and crashesDirectory with appropriate values.

**Note:** If you need send additional/updated extra parameters after your first call start you can call setExtraParameter on macOS or call start again with the new/updated extra parameters on Linux and Windows.

**const** args = [

`--reporter-url=${submitURL}`,

`--application-name=${productName}`,

`--crashes-directory=${crashesDirectory}`

]

**const** env = {

ELECTRON\_INTERNAL\_CRASH\_SERVICE: 1

}

spawn(process.execPath, args, {

env: env,

detached: true

})

**Note:** On macOS, Electron uses a new crashpad client for crash collection and reporting. If you want to enable crash reporting, initializing crashpad from the main process using crashReporter.start is required regardless of which process you want to collect crashes from. Once initialized this way, the crashpad handler collects crashes from all processes. You still have to call crashReporter.start from the renderer or child process, otherwise crashes from them will get reported without companyName, productName or any of the extra information.

crashReporter.getLastCrashReport()

Returns [CrashReport](https://electron.atom.io/docs/api/structures/crash-report):

Returns the date and ID of the last crash report. If no crash reports have been sent or the crash reporter has not been started, null is returned.

crashReporter.getUploadedReports()

Returns [CrashReport[]](https://electron.atom.io/docs/api/structures/crash-report):

Returns all uploaded crash reports. Each report contains the date and uploaded ID.

crashReporter.getUploadToServer() Linux macOS

Returns Boolean - Whether reports should be submitted to the server. Set through the start method or setUploadToServer.

**Note:** This API can only be called from the main process.

crashReporter.setUploadToServer(uploadToServer) Linux macOS

* uploadToServer Boolean macOS - Whether reports should be submitted to the server

This would normally be controlled by user preferences. This has no effect if called before start is called.

**Note:** This API can only be called from the main process.

crashReporter.setExtraParameter(key, value) macOS

* key String - Parameter key, must be less than 64 characters long.
* value String - Parameter value, must be less than 64 characters long. Specifying null or undefined will remove the key from the extra parameters.

Set an extra parameter to be sent with the crash report. The values specified here will be sent in addition to any values set via the extra option when start was called. This API is only available on macOS, if you need to add/update extra parameters on Linux and Windows after your first call to start you can call start again with the updated extra options.

Crash Report Payload

The crash reporter will send the following data to the submitURL as a multipart/form-data POST:

* ver String - The version of Electron.
* platform String - e.g. ‘win32’.
* process\_type String - e.g. ‘renderer’.
* guid String - e.g. ‘5e1286fc-da97-479e-918b-6bfb0c3d1c72’
* \_version String - The version in package.json.
* \_productName String - The product name in the crashReporter options object.
* prod String - Name of the underlying product. In this case Electron.
* \_companyName String - The company name in the crashReporter options object.
* upload\_file\_minidump File - The crash report in the format of minidump.
* All level one properties of the extra object in the crashReporter options object.

Class: Debugger

An alternate transport for Chrome’s remote debugging protocol.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

Chrome Developer Tools has a [special binding](https://developer.chrome.com/devtools/docs/debugger-protocol) available at JavaScript runtime that allows interacting with pages and instrumenting them.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

**try** {

win.webContents.**debugger**.attach('1.1')

} **catch** (err) {

console.log('Debugger attach failed : ', err)

}

win.webContents.**debugger**.on('detach', (event, reason) => {

console.log('Debugger detached due to : ', reason)

})

win.webContents.**debugger**.on('message', (event, method, params) => {

**if** (method === 'Network.requestWillBeSent') {

**if** (params.request.url === 'https://www.github.com') {

win.webContents.**debugger**.detach()

}

}

})

win.webContents.**debugger**.sendCommand('Network.enable')

Instance Methods

debugger.attach([protocolVersion])

* protocolVersion String (optional) - Requested debugging protocol version.

Attaches the debugger to the webContents.

debugger.isAttached()

Returns Boolean - Whether a debugger is attached to the webContents.

debugger.detach()

Detaches the debugger from the webContents.

debugger.sendCommand(method[, commandParams, callback])

* method String - Method name, should be one of the methods defined by the remote debugging protocol.
* commandParams Object (optional) - JSON object with request parameters.
* callback Function (optional) - Response
  + error Object - Error message indicating the failure of the command.
  + result Any - Response defined by the ‘returns’ attribute of the command description in the remote debugging protocol.

Send given command to the debugging target.

Instance Events

Event: ‘detach’

* event Event
* reason String - Reason for detaching debugger.

Emitted when debugging session is terminated. This happens either when webContents is closed or devtools is invoked for the attached webContents.

Event: ‘message’

* event Event
* method String - Method name.
* params Object - Event parameters defined by the ‘parameters’ attribute in the remote debugging protocol.

Emitted whenever debugging target issues instrumentation event.

desktopCapturer

Access information about media sources that can be used to capture audio and video from the desktop using the [navigator.mediaDevices.getUserMedia](https://developer.mozilla.org/en/docs/Web/API/MediaDevices/getUserMedia) API.

Process: [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

The following example shows how to capture video from a desktop window whose title is Electron:

*// In the renderer process.*

**const** {desktopCapturer} = require('electron')

desktopCapturer.getSources({types: ['window', 'screen']}, (error, sources) => {

**if** (error) **throw** error

**for** (**let** i = 0; i < sources.length; ++i) {

**if** (sources[i].name === 'Electron') {

navigator.mediaDevices.getUserMedia({

audio: false,

video: {

mandatory: {

chromeMediaSource: 'desktop',

chromeMediaSourceId: sources[i].id,

minWidth: 1280,

maxWidth: 1280,

minHeight: 720,

maxHeight: 720

}

}

}, handleStream, handleError)

**return**

}

}

})

**function** handleStream (stream) {

document.querySelector('video').src = URL.createObjectURL(stream)

}

**function** handleError (e) {

console.log(e)

}

To capture video from a source provided by desktopCapturer the constraints passed to [navigator.mediaDevices.getUserMedia](https://developer.mozilla.org/en/docs/Web/API/MediaDevices/getUserMedia) must include chromeMediaSource: 'desktop', and audio: false.

To capture both audio and video from the entire desktop the constraints passed to [navigator.mediaDevices.getUserMedia](https://developer.mozilla.org/en/docs/Web/API/MediaDevices/getUserMedia) must include chromeMediaSource: 'desktop', for both audio and video, but should not include a chromeMediaSourceId constraint.

**const** constraints = {

audio: {

mandatory: {

chromeMediaSource: 'desktop'

}

},

video: {

mandatory: {

chromeMediaSource: 'desktop'

}

}

}

Methods

The desktopCapturer module has the following methods:

desktopCapturer.getSources(options, callback)

* options Object
  + types String[] - An array of Strings that lists the types of desktop sources to be captured, available types are screen and window.
  + thumbnailSize [Size](https://electron.atom.io/docs/api/structures/size) (optional) - The size that the media source thumbnail should be scaled to. Default is 150 x 150.
* callback Function
  + error Error
  + sources [DesktopCapturerSource[]](https://electron.atom.io/docs/api/structures/desktop-capturer-source)

Starts gathering information about all available desktop media sources, and calls callback(error, sources)when finished.

sources is an array of [DesktopCapturerSource](https://electron.atom.io/docs/api/structures/desktop-capturer-source) objects, each DesktopCapturerSource represents a screen or an individual window that can be captured.

dialog

Display native system dialogs for opening and saving files, alerting, etc.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

An example of showing a dialog to select multiple files and directories:

**const** {dialog} = require('electron')

console.log(dialog.showOpenDialog({properties: ['openFile', 'openDirectory', 'multiSelections']}))

The Dialog is opened from Electron’s main thread. If you want to use the dialog object from a renderer process, remember to access it using the remote:

**const** {dialog} = require('electron').remote

console.log(dialog)

Methods

The dialog module has the following methods:

dialog.showOpenDialog([browserWindow, ]options[, callback])

* browserWindow BrowserWindow (optional)
* options Object
  + title String (optional)
  + defaultPath String (optional)
  + buttonLabel String (optional) - Custom label for the confirmation button, when left empty the default label will be used.
  + filters [FileFilter[]](https://electron.atom.io/docs/api/structures/file-filter) (optional)
  + properties String[] (optional) - Contains which features the dialog should use. The following values are supported:
    - openFile - Allow files to be selected.
    - openDirectory - Allow directories to be selected.
    - multiSelections - Allow multiple paths to be selected.
    - showHiddenFiles - Show hidden files in dialog.
    - createDirectory - Allow creating new directories from dialog. macOS
    - promptToCreate - Prompt for creation if the file path entered in the dialog does not exist. This does not actually create the file at the path but allows non-existent paths to be returned that should be created by the application. Windows
    - noResolveAliases - Disable the automatic alias (symlink) path resolution. Selected aliases will now return the alias path instead of their target path. macOS
    - treatPackageAsDirectory - Treat packages, such as .app folders, as a directory instead of a file. macOS
  + message String (optional) macOS - Message to display above input boxes.
* callback Function (optional)
  + filePaths String[] - An array of file paths chosen by the user

Returns String[], an array of file paths chosen by the user, if the callback is provided it returns undefined.

The browserWindow argument allows the dialog to attach itself to a parent window, making it modal.

The filters specifies an array of file types that can be displayed or selected when you want to limit the user to a specific type. For example:

{

filters: [

{name: 'Images', extensions: ['jpg', 'png', 'gif']},

{name: 'Movies', extensions: ['mkv', 'avi', 'mp4']},

{name: 'Custom File Type', extensions: ['as']},

{name: 'All Files', extensions: ['\*']}

]

}

The extensions array should contain extensions without wildcards or dots (e.g. 'png' is good but '.png' and '\*.png' are bad). To show all files, use the '\*' wildcard (no other wildcard is supported).

If a callback is passed, the API call will be asynchronous and the result will be passed via callback(filenames)

**Note:** On Windows and Linux an open dialog can not be both a file selector and a directory selector, so if you set properties to ['openFile', 'openDirectory'] on these platforms, a directory selector will be shown.

dialog.showSaveDialog([browserWindow, ]options[, callback])

* browserWindow BrowserWindow (optional)
* options Object
  + title String (optional)
  + defaultPath String (optional) - Absolute directory path, absolute file path, or file name to use by default.
  + buttonLabel String (optional) - Custom label for the confirmation button, when left empty the default label will be used.
  + filters [FileFilter[]](https://electron.atom.io/docs/api/structures/file-filter) (optional)
  + message String (optional) macOS - Message to display above text fields.
  + nameFieldLabel String (optional) macOS - Custom label for the text displayed in front of the filename text field.
  + showsTagField Boolean (optional) macOS - Show the tags input box, defaults to true.
* callback Function (optional)
  + filename String

Returns String, the path of the file chosen by the user, if a callback is provided it returns undefined.

The browserWindow argument allows the dialog to attach itself to a parent window, making it modal.

The filters specifies an array of file types that can be displayed, see dialog.showOpenDialog for an example.

If a callback is passed, the API call will be asynchronous and the result will be passed via callback(filename)

dialog.showMessageBox([browserWindow, ]options[, callback])

* browserWindow BrowserWindow (optional)
* options Object
  + type String (optional) - Can be "none", "info", "error", "question" or "warning". On Windows, "question" displays the same icon as "info", unless you set an icon using the "icon" option. On macOS, both "warning" and "error" display the same warning icon.
  + buttons String[] (optional) - Array of texts for buttons. On Windows, an empty array will result in one button labeled “OK”.
  + defaultId Integer (optional) - Index of the button in the buttons array which will be selected by default when the message box opens.
  + title String (optional) - Title of the message box, some platforms will not show it.
  + message String - Content of the message box.
  + detail String (optional) - Extra information of the message.
  + checkboxLabel String (optional) - If provided, the message box will include a checkbox with the given label. The checkbox state can be inspected only when using callback.
  + checkboxChecked Boolean (optional) - Initial checked state of the checkbox. false by default.
  + icon [NativeImage](https://electron.atom.io/docs/api/native-image) (optional)
  + cancelId Integer (optional) - The index of the button to be used to cancel the dialog, via the Esc key. By default this is assigned to the first button with “cancel” or “no” as the label. If no such labeled buttons exist and this option is not set, 0 will be used as the return value or callback response. This option is ignored on Windows.
  + noLink Boolean (optional) - On Windows Electron will try to figure out which one of the buttons are common buttons (like “Cancel” or “Yes”), and show the others as command links in the dialog. This can make the dialog appear in the style of modern Windows apps. If you don’t like this behavior, you can set noLink to true.
  + normalizeAccessKeys Boolean (optional) - Normalize the keyboard access keys across platforms. Default is false. Enabling this assumes & is used in the button labels for the placement of the keyboard shortcut access key and labels will be converted so they work correctly on each platform, & characters are removed on macOS, converted to \_ on Linux, and left untouched on Windows. For example, a button label of Vie&w will be converted to Vie\_w on Linux and View on macOS and can be selected via Alt-Won Windows and Linux.
* callback Function (optional)
  + response Number - The index of the button that was clicked
  + checkboxChecked Boolean - The checked state of the checkbox if checkboxLabel was set. Otherwise false.

Returns Integer, the index of the clicked button, if a callback is provided it returns undefined.

Shows a message box, it will block the process until the message box is closed. It returns the index of the clicked button.

The browserWindow argument allows the dialog to attach itself to a parent window, making it modal.

If a callback is passed, the dialog will not block the process. The API call will be asynchronous and the result will be passed via callback(response).

dialog.showErrorBox(title, content)

* title String - The title to display in the error box
* content String - The text content to display in the error box

Displays a modal dialog that shows an error message.

This API can be called safely before the ready event the app module emits, it is usually used to report errors in early stage of startup. If called before the app readyevent on Linux, the message will be emitted to stderr, and no GUI dialog will appear.

dialog.showCertificateTrustDialog([browserWindow, ]options, callback) macOSWindows

* browserWindow BrowserWindow (optional)
* options Object
  + certificate [Certificate](https://electron.atom.io/docs/api/structures/certificate) - The certificate to trust/import.
  + message String - The message to display to the user.
* callback Function

On macOS, this displays a modal dialog that shows a message and certificate information, and gives the user the option of trusting/importing the certificate. If you provide a browserWindow argument the dialog will be attached to the parent window, making it modal.

On Windows the options are more limited, due to the Win32 APIs used:

* The message argument is not used, as the OS provides its own confirmation dialog.
* The browserWindow argument is ignored since it is not possible to make this confirmation dialog modal.

Sheets

On macOS, dialogs are presented as sheets attached to a window if you provide a BrowserWindow reference in the browserWindow parameter, or modals if no window is provided.

You can call BrowserWindow.getCurrentWindow().setSheetOffset(offset) to change the offset from the window frame where sheets are attached.

Class: DownloadItem

Control file downloads from remote sources.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

DownloadItem is an EventEmitter that represents a download item in Electron. It is used in will-download event of Session class, and allows users to control the download item.

*// In the main process.*

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.webContents.session.on('will-download', (event, item, webContents) => {

*// Set the save path, making Electron not to prompt a save dialog.*

item.setSavePath('/tmp/save.pdf')

item.on('updated', (event, state) => {

**if** (state === 'interrupted') {

console.log('Download is interrupted but can be resumed')

} **else** **if** (state === 'progressing') {

**if** (item.isPaused()) {

console.log('Download is paused')

} **else** {

console.log(`Received bytes: ${item.getReceivedBytes()}`)

}

}

})

item.once('done', (event, state) => {

**if** (state === 'completed') {

console.log('Download successfully')

} **else** {

console.log(`Download failed: ${state}`)

}

})

})

Instance Events

Event: ‘updated’

Returns:

* event Event
* state String

Emitted when the download has been updated and is not done.

The state can be one of following:

* progressing - The download is in-progress.
* interrupted - The download has interrupted and can be resumed.

Event: ‘done’

Returns:

* event Event
* state String

Emitted when the download is in a terminal state. This includes a completed download, a cancelled download (via downloadItem.cancel()), and interrupted download that can’t be resumed.

The state can be one of following:

* completed - The download completed successfully.
* cancelled - The download has been cancelled.
* interrupted - The download has interrupted and can not resume.

Instance Methods

The downloadItem object has the following methods:

downloadItem.setSavePath(path)

* path String - Set the save file path of the download item.

The API is only available in session’s will-download callback function. If user doesn’t set the save path via the API, Electron will use the original routine to determine the save path(Usually prompts a save dialog).

downloadItem.getSavePath()

Returns String - The save path of the download item. This will be either the path set via downloadItem.setSavePath(path) or the path selected from the shown save dialog.

downloadItem.pause()

Pauses the download.

downloadItem.isPaused()

Returns Boolean - Whether the download is paused.

downloadItem.resume()

Resumes the download that has been paused.

**Note:** To enable resumable downloads the server you are downloading from must support range requests and provide both Last-Modified and ETag header values. Otherwise resume() will dismiss previously received bytes and restart the download from the beginning.

downloadItem.canResume()

Resumes Boolean - Whether the download can resume.

downloadItem.cancel()

Cancels the download operation.

downloadItem.getURL()

Returns String - The origin url where the item is downloaded from.

downloadItem.getMimeType()

Returns String - The files mime type.

downloadItem.hasUserGesture()

Returns Boolean - Whether the download has user gesture.

downloadItem.getFilename()

Returns String - The file name of the download item.

**Note:** The file name is not always the same as the actual one saved in local disk. If user changes the file name in a prompted download saving dialog, the actual name of saved file will be different.

downloadItem.getTotalBytes()

Returns Integer - The total size in bytes of the download item.

If the size is unknown, it returns 0.

downloadItem.getReceivedBytes()

Returns Integer - The received bytes of the download item.

downloadItem.getContentDisposition()

Returns String - The Content-Disposition field from the response header.

downloadItem.getState()

Returns String - The current state. Can be progressing, completed, cancelled or interrupted.

**Note:** The following methods are useful specifically to resume a cancelled item when session is restarted.

downloadItem.getURLChain()

Returns String[] - The complete url chain of the item including any redirects.

downloadItem.getLastModifiedTime()

Returns String - Last-Modified header value.

downloadItem.getETag()

Returns String - ETag header value.

downloadItem.getStartTime()

Returns Double - Number of seconds since the UNIX epoch when the download was started.

Environment Variables

Control application configuration and behavior without changing code.

Certain Electron behaviors are controlled by environment variables because they are initialized earlier than the command line flags and the app’s code.

POSIX shell example:

$ export ELECTRON\_ENABLE\_LOGGING=true

$ electron

Windows console example:

> set ELECTRON\_ENABLE\_LOGGING=true

> electron

Production Variables

The following environment variables are intended primarily for use at runtime in packaged Electron applications.

GOOGLE\_API\_KEY

Electron includes a hardcoded API key for making requests to Google’s geocoding webservice. Because this API key is included in every version of Electron, it often exceeds its usage quota. To work around this, you can supply your own Google API key in the environment. Place the following code in your main process file, before opening any browser windows that will make geocoding requests:

process.env.GOOGLE\_API\_KEY = 'YOUR\_KEY\_HERE'

For instructions on how to acquire a Google API key, visit [this page](https://www.chromium.org/developers/how-tos/api-keys).

By default, a newly generated Google API key may not be allowed to make geocoding requests. To enable geocoding requests, visit [this page](https://console.developers.google.com/apis/api/geolocation/overview).

ELECTRON\_NO\_ASAR

Disables ASAR support. This variable is only supported in forked child processes and spawned child processes that set ELECTRON\_RUN\_AS\_NODE.

ELECTRON\_RUN\_AS\_NODE

Starts the process as a normal Node.js process.

ELECTRON\_NO\_ATTACH\_CONSOLE Windows

Don’t attach to the current console session.

ELECTRON\_FORCE\_WINDOW\_MENU\_BAR Linux

Don’t use the global menu bar on Linux.

Development Variables

The following environment variables are intended primarily for development and debugging purposes.

ELECTRON\_ENABLE\_LOGGING

Prints Chrome’s internal logging to the console.

ELECTRON\_LOG\_ASAR\_READS

When Electron reads from an ASAR file, log the read offset and file path to the system tmpdir. The resulting file can be provided to the ASAR module to optimize file ordering.

ELECTRON\_ENABLE\_STACK\_DUMPING

Prints the stack trace to the console when Electron crashes.

This environment variable will not work if the crashReporter is started.

ELECTRON\_DEFAULT\_ERROR\_MODE Windows

Shows the Windows’s crash dialog when Electron crashes.

This environment variable will not work if the crashReporter is started.

File Object

Use the HTML5 File API to work natively with files on the filesystem.

The DOM’s File interface provides abstraction around native files in order to let users work on native files directly with the HTML5 file API. Electron has added a path attribute to the File interface which exposes the file’s real path on filesystem.

Example of getting a real path from a dragged-onto-the-app file:

<div id="holder">

Drag your file here

</div>

<script>

document.addEventListener('drop', **function** (e) {

e.preventDefault();

e.stopPropagation();

**for** (**let** f of e.dataTransfer.files) {

console.log('File(s) you dragged here: ', f.path)

}

});

document.addEventListener('dragover', **function** (e) {

e.preventDefault();

e.stopPropagation();

});

</script>

Frameless Window

Open a window without toolbars, borders, or other graphical “chrome”.

A frameless window is a window that has no [chrome](https://developer.mozilla.org/en-US/docs/Glossary/Chrome), the parts of the window, like toolbars, that are not a part of the web page. These are options on the [BrowserWindow](https://electron.atom.io/docs/api/browser-window) class.

Create a frameless window

To create a frameless window, you need to set frame to false in [BrowserWindow](https://electron.atom.io/docs/api/browser-window)’s options:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({width: 800, height: 600, frame: false})

win.show()

Alternatives on macOS

On macOS 10.9 Mavericks and newer, there’s an alternative way to specify a chromeless window. Instead of setting frame to false which disables both the titlebar and window controls, you may want to have the title bar hidden and your content extend to the full window size, yet still preserve the window controls (“traffic lights”) for standard window actions. You can do so by specifying the titleBarStyle option:

hidden

Results in a hidden title bar and a full size content window, yet the title bar still has the standard window controls (“traffic lights”) in the top left.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({titleBarStyle: 'hidden'})

win.show()

hiddenInset

Results in a hidden title bar with an alternative look where the traffic light buttons are slightly more inset from the window edge.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({titleBarStyle: 'hiddenInset'})

win.show()

customButtonsOnHover

Uses custom drawn close, miniaturize, and fullscreen buttons that display when hovering in the top left of the window. These custom buttons prevent issues with mouse events that occur with the standard window toolbar buttons. This option is only applicable for frameless windows.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({titleBarStyle: 'customButtonsOnHover', frame: false})

win.show()

Transparent window

By setting the transparent option to true, you can also make the frameless window transparent:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({transparent: true, frame: false})

win.show()

Limitations

* You can not click through the transparent area. We are going to introduce an API to set window shape to solve this, see [our issue](https://github.com/electron/electron/issues/1335) for details.
* Transparent windows are not resizable. Setting resizable to true may make a transparent window stop working on some platforms.
* The blur filter only applies to the web page, so there is no way to apply blur effect to the content below the window (i.e. other applications open on the user’s system).
* On Windows operating systems, transparent windows will not work when DWM is disabled.
* On Linux users have to put --enable-transparent-visuals --disable-gpu in the command line to disable GPU and allow ARGB to make transparent window, this is caused by an upstream bug that [alpha channel doesn’t work on some NVidia drivers](https://code.google.com/p/chromium/issues/detail?id=369209) on Linux.
* On Mac the native window shadow will not be shown on a transparent window.

Click-through window

To create a click-through window, i.e. making the window ignore all mouse events, you can call the [win.setIgnoreMouseEvents(ignore)](https://electron.atom.io/docs/api/browser-window#winsetignoremouseeventsignore) API:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.setIgnoreMouseEvents(true)

Draggable region

By default, the frameless window is non-draggable. Apps need to specify -webkit-app-region: drag in CSS to tell Electron which regions are draggable (like the OS’s standard titlebar), and apps can also use -webkit-app-region: no-drag to exclude the non-draggable area from the draggable region. Note that only rectangular shapes are currently supported.

Note: -webkit-app-region: drag is known to have problems while the developer tools are open. See this [GitHub issue](https://github.com/electron/electron/issues/3647) for more information including a workaround.

To make the whole window draggable, you can add -webkit-app-region: drag as body’s style:

<body style="-webkit-app-region: drag">

</body>

And note that if you have made the whole window draggable, you must also mark buttons as non-draggable, otherwise it would be impossible for users to click on them:

button {

-webkit-app-region: no-drag;

}

If you’re setting just a custom titlebar as draggable, you also need to make all buttons in titlebar non-draggable.

Text selection

In a frameless window the dragging behaviour may conflict with selecting text. For example, when you drag the titlebar you may accidentally select the text on the titlebar. To prevent this, you need to disable text selection within a draggable area like this:

**.titlebar** {

-webkit-user-select: none;

-webkit-app-region: drag;

}

Context menu

On some platforms, the draggable area will be treated as a non-client frame, so when you right click on it a system menu will pop up. To make the context menu behave correctly on all platforms you should never use a custom context menu on draggable areas.

globalShortcut

Detect keyboard events when the application does not have keyboard focus.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

The globalShortcut module can register/unregister a global keyboard shortcut with the operating system so that you can customize the operations for various shortcuts.

**Note:** The shortcut is global; it will work even if the app does not have the keyboard focus. You should not use this module until the ready event of the app module is emitted.

**const** {app, globalShortcut} = require('electron')

app.on('ready', () => {

*// Register a 'CommandOrControl+X' shortcut listener.*

**const** ret = globalShortcut.register('CommandOrControl+X', () => {

console.log('CommandOrControl+X is pressed')

})

**if** (!ret) {

console.log('registration failed')

}

*// Check whether a shortcut is registered.*

console.log(globalShortcut.isRegistered('CommandOrControl+X'))

})

app.on('will-quit', () => {

*// Unregister a shortcut.*

globalShortcut.unregister('CommandOrControl+X')

*// Unregister all shortcuts.*

globalShortcut.unregisterAll()

})

Methods

The globalShortcut module has the following methods:

globalShortcut.register(accelerator, callback)

* accelerator [Accelerator](https://electron.atom.io/docs/api/accelerator)
* callback Function

Registers a global shortcut of accelerator. The callback is called when the registered shortcut is pressed by the user.

When the accelerator is already taken by other applications, this call will silently fail. This behavior is intended by operating systems, since they don’t want applications to fight for global shortcuts.

globalShortcut.isRegistered(accelerator)

* accelerator [Accelerator](https://electron.atom.io/docs/api/accelerator)

Returns Boolean - Whether this application has registered accelerator.

When the accelerator is already taken by other applications, this call will still return false. This behavior is intended by operating systems, since they don’t want applications to fight for global shortcuts.

globalShortcut.unregister(accelerator)

* accelerator [Accelerator](https://electron.atom.io/docs/api/accelerator)

Unregisters the global shortcut of accelerator.

globalShortcut.unregisterAll()

Unregisters all of the global shortcuts.

Class: IncomingMessage

Handle responses to HTTP/HTTPS requests.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

IncomingMessage implements the [Readable Stream](https://nodejs.org/api/stream.html#stream_readable_streams) interface and is therefore an [EventEmitter](https://nodejs.org/api/events.html#events_class_eventemitter).

Instance Events

Event: ‘data’

Returns:

* chunk Buffer - A chunk of response body’s data.

The data event is the usual method of transferring response data into applicative code.

Event: ‘end’

Indicates that response body has ended.

Event: ‘aborted’

Emitted when a request has been canceled during an ongoing HTTP transaction.

Event: ‘error’

Returns:

error Error - Typically holds an error string identifying failure root cause.

Emitted when an error was encountered while streaming response data events. For instance, if the server closes the underlying while the response is still streaming, an error event will be emitted on the response object and a close event will subsequently follow on the request object.

Instance Properties

An IncomingMessage instance has the following readable properties:

response.statusCode

An Integer indicating the HTTP response status code.

response.statusMessage

A String representing the HTTP status message.

response.headers

An Object representing the response HTTP headers. The headers object is formatted as follows:

* All header names are lowercased.
* Each header name produces an array-valued property on the headers object.
* Each header value is pushed into the array associated with its header name.

response.httpVersion

A String indicating the HTTP protocol version number. Typical values are ‘1.0’ or ‘1.1’. Additionally httpVersionMajor and httpVersionMinor are two Integer-valued readable properties that return respectively the HTTP major and minor version numbers.

response.httpVersionMajor

An Integer indicating the HTTP protocol major version number.

response.httpVersionMinor

An Integer indicating the HTTP protocol minor version number.

ipcMain

Communicate asynchronously from the main process to renderer processes.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

The ipcMain module is an instance of the [EventEmitter](https://nodejs.org/api/events.html#events_class_eventemitter) class. When used in the main process, it handles asynchronous and synchronous messages sent from a renderer process (web page). Messages sent from a renderer will be emitted to this module.

Sending Messages

It is also possible to send messages from the main process to the renderer process, see [webContents.send](https://electron.atom.io/docs/api/web-contents#webcontentssendchannel-arg1-arg2-) for more information.

* When sending a message, the event name is the channel.
* To reply to a synchronous message, you need to set event.returnValue.
* To send an asynchronous message back to the sender, you can use event.sender.send(...).

An example of sending and handling messages between the render and main processes:

*// In main process.*

**const** {ipcMain} = require('electron')

ipcMain.on('asynchronous-message', (event, arg) => {

console.log(arg) *// prints "ping"*

event.sender.send('asynchronous-reply', 'pong')

})

ipcMain.on('synchronous-message', (event, arg) => {

console.log(arg) *// prints "ping"*

event.returnValue = 'pong'

})

*// In renderer process (web page).*

**const** {ipcRenderer} = require('electron')

console.log(ipcRenderer.sendSync('synchronous-message', 'ping')) *// prints "pong"*

ipcRenderer.on('asynchronous-reply', (event, arg) => {

console.log(arg) *// prints "pong"*

})

ipcRenderer.send('asynchronous-message', 'ping')

Methods

The ipcMain module has the following method to listen for events:

ipcMain.on(channel, listener)

* channel String
* listener Function

Listens to channel, when a new message arrives listener would be called with listener(event, args...).

ipcMain.once(channel, listener)

* channel String
* listener Function

Adds a one time listener function for the event. This listener is invoked only the next time a message is sent to channel, after which it is removed.

ipcMain.removeListener(channel, listener)

* channel String
* listener Function

Removes the specified listener from the listener array for the specified channel.

ipcMain.removeAllListeners([channel])

* channel String

Removes listeners of the specified channel.

Event object

The event object passed to the callback has the following methods:

event.returnValue

Set this to the value to be returned in a synchronous message.

event.sender

Returns the webContents that sent the message, you can call event.sender.send to reply to the asynchronous message, see [webContents.send](https://electron.atom.io/docs/api/web-contents#webcontentssendchannel-arg1-arg2-) for more information.

ipcRenderer

Communicate asynchronously from a renderer process to the main process.

Process: [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

The ipcRenderer module is an instance of the [EventEmitter](https://nodejs.org/api/events.html#events_class_eventemitter) class. It provides a few methods so you can send synchronous and asynchronous messages from the render process (web page) to the main process. You can also receive replies from the main process.

See [ipcMain](https://electron.atom.io/docs/api/ipc-main) for code examples.

Methods

The ipcRenderer module has the following method to listen for events and send messages:

ipcRenderer.on(channel, listener)

* channel String
* listener Function

Listens to channel, when a new message arrives listener would be called with listener(event, args...).

ipcRenderer.once(channel, listener)

* channel String
* listener Function

Adds a one time listener function for the event. This listener is invoked only the next time a message is sent to channel, after which it is removed.

ipcRenderer.removeListener(channel, listener)

* channel String
* listener Function

Removes the specified listener from the listener array for the specified channel.

ipcRenderer.removeAllListeners([channel])

* channel String (optional)

Removes all listeners, or those of the specified channel.

ipcRenderer.send(channel[, arg1][, arg2][, ...])

* channel String
* ...args any[]

Send a message to the main process asynchronously via channel, you can also send arbitrary arguments. Arguments will be serialized in JSON internally and hence no functions or prototype chain will be included.

The main process handles it by listening for channel with ipcMain module.

ipcRenderer.sendSync(channel[, arg1][, arg2][, ...])

* channel String
* ...args any[]

Returns any - The value sent back by the [ipcMain](https://electron.atom.io/docs/api/ipc-main) handler.

Send a message to the main process synchronously via channel, you can also send arbitrary arguments. Arguments will be serialized in JSON internally and hence no functions or prototype chain will be included.

The main process handles it by listening for channel with ipcMain module, and replies by setting event.returnValue.

**Note:** Sending a synchronous message will block the whole renderer process, unless you know what you are doing you should never use it.

ipcRenderer.sendToHost(channel[, arg1][, arg2][, ...])

* channel String
* ...args any[]

Like ipcRenderer.send but the event will be sent to the <webview> element in the host page instead of the main process.

Locales

Locale values returned by app.getLocale().

Electron uses Chromium’s l10n\_util library to fetch the locale. Possible values are listed below:

| **Language Code** | **Language Name** |
| --- | --- |
| af | Afrikaans |
| am | Amharic |
| ar | Arabic |
| az | Azerbaijani |
| be | Belarusian |
| bg | Bulgarian |
| bh | Bihari |
| bn | Bengali |
| br | Breton |
| bs | Bosnian |
| ca | Catalan |
| co | Corsican |
| cs | Czech |
| cy | Welsh |
| da | Danish |
| de | German |
| de-AT | German (Austria) |
| de-CH | German (Switzerland) |
| de-DE | German (Germany) |
| el | Greek |
| en | English |
| en-AU | English (Australia) |
| en-CA | English (Canada) |
| en-GB | English (UK) |
| en-NZ | English (New Zealand) |
| en-US | English (US) |
| en-ZA | English (South Africa) |
| eo | Esperanto |
| es | Spanish |
| es-419 | Spanish (Latin America) |
| et | Estonian |
| eu | Basque |
| fa | Persian |
| fi | Finnish |
| fil | Filipino |
| fo | Faroese |
| fr | French |
| fr-CA | French (Canada) |
| fr-CH | French (Switzerland) |
| fr-FR | French (France) |
| fy | Frisian |
| ga | Irish |
| gd | Scots Gaelic |
| gl | Galician |
| gn | Guarani |
| gu | Gujarati |
| ha | Hausa |
| haw | Hawaiian |
| he | Hebrew |
| hi | Hindi |
| hr | Croatian |
| hu | Hungarian |
| hy | Armenian |
| ia | Interlingua |
| id | Indonesian |
| is | Icelandic |
| it | Italian |
| it-CH | Italian (Switzerland) |
| it-IT | Italian (Italy) |
| ja | Japanese |
| jw | Javanese |
| ka | Georgian |
| kk | Kazakh |
| km | Cambodian |
| kn | Kannada |
| ko | Korean |
| ku | Kurdish |
| ky | Kyrgyz |
| la | Latin |
| ln | Lingala |
| lo | Laothian |
| lt | Lithuanian |
| lv | Latvian |
| mk | Macedonian |
| ml | Malayalam |
| mn | Mongolian |
| mo | Moldavian |
| mr | Marathi |
| ms | Malay |
| mt | Maltese |
| nb | Norwegian (Bokmal) |
| ne | Nepali |
| nl | Dutch |
| nn | Norwegian (Nynorsk) |
| no | Norwegian |
| oc | Occitan |
| om | Oromo |
| or | Oriya |
| pa | Punjabi |
| pl | Polish |
| ps | Pashto |
| pt | Portuguese |
| pt-BR | Portuguese (Brazil) |
| pt-PT | Portuguese (Portugal) |
| qu | Quechua |
| rm | Romansh |
| ro | Romanian |
| ru | Russian |
| sd | Sindhi |
| sh | Serbo-Croatian |
| si | Sinhalese |
| sk | Slovak |
| sl | Slovenian |
| sn | Shona |
| so | Somali |
| sq | Albanian |
| sr | Serbian |
| st | Sesotho |
| su | Sundanese |
| sv | Swedish |
| sw | Swahili |
| ta | Tamil |
| te | Telugu |
| tg | Tajik |
| th | Thai |
| ti | Tigrinya |
| tk | Turkmen |
| to | Tonga |
| tr | Turkish |
| tt | Tatar |
| tw | Twi |
| ug | Uighur |
| uk | Ukrainian |
| ur | Urdu |
| uz | Uzbek |
| vi | Vietnamese |
| xh | Xhosa |
| yi | Yiddish |
| yo | Yoruba |
| zh | Chinese |
| zh-CN | Chinese (Simplified) |
| zh-TW | Chinese (Traditional) |
| zu | Zulu |

Class: MenuItem

Add items to native application menus and context menus.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

See [Menu](https://electron.atom.io/docs/api/menu) for examples.

new MenuItem(options)

* options Object
  + click Function (optional) - Will be called with click(menuItem, browserWindow, event) when the menu item is clicked.
    - menuItem MenuItem
    - browserWindow BrowserWindow
    - event Event
  + role String (optional) - Define the action of the menu item, when specified the click property will be ignored. See [roles](https://electron.atom.io/docs/all/#roles).
  + type String (optional) - Can be normal, separator, submenu, checkbox or radio.
  + label String - (optional)
  + sublabel String - (optional)
  + accelerator [Accelerator](https://electron.atom.io/docs/api/accelerator) (optional)
  + icon ([NativeImage](https://electron.atom.io/docs/api/native-image) | String) (optional)
  + enabled Boolean (optional) - If false, the menu item will be greyed out and unclickable.
  + visible Boolean (optional) - If false, the menu item will be entirely hidden.
  + checked Boolean (optional) - Should only be specified for checkbox or radio type menu items.
  + submenu (MenuItemConstructorOptions[] | Menu) (optional) - Should be specified for submenu type menu items. If submenu is specified, the type: 'submenu' can be omitted. If the value is not a Menu then it will be automatically converted to one using Menu.buildFromTemplate.
  + id String (optional) - Unique within a single menu. If defined then it can be used as a reference to this item by the position attribute.
  + position String (optional) - This field allows fine-grained definition of the specific location within a given menu.

Roles

Roles allow menu items to have predefined behaviors.

It is best to specify role for any menu item that matches a standard role, rather than trying to manually implement the behavior in a click function. The built-in role behavior will give the best native experience.

The label and accelerator values are optional when using a role and will default to appropriate values for each platform.

The role property can have following values:

* undo
* redo
* cut
* copy
* paste
* pasteandmatchstyle
* selectall
* delete
* minimize - Minimize current window
* close - Close current window
* quit- Quit the application
* reload - Reload the current window
* forcereload - Reload the current window ignoring the cache.
* toggledevtools - Toggle developer tools in the current window
* togglefullscreen- Toggle full screen mode on the current window
* resetzoom - Reset the focused page’s zoom level to the original size
* zoomin - Zoom in the focused page by 10%
* zoomout - Zoom out the focused page by 10%
* editMenu - Whole default “Edit” menu (Undo, Copy, etc.)
* windowMenu - Whole default “Window” menu (Minimize, Close, etc.)

The following additional roles are available on macOS:

* about - Map to the orderFrontStandardAboutPanel action
* hide - Map to the hide action
* hideothers - Map to the hideOtherApplications action
* unhide - Map to the unhideAllApplications action
* startspeaking - Map to the startSpeaking action
* stopspeaking - Map to the stopSpeaking action
* front - Map to the arrangeInFront action
* zoom - Map to the performZoom action
* window - The submenu is a “Window” menu
* help - The submenu is a “Help” menu
* services - The submenu is a “Services” menu

When specifying a role on macOS, label and accelerator are the only options that will affect the menu item. All other options will be ignored.

Instance Properties

The following properties are available on instances of MenuItem:

menuItem.enabled

A Boolean indicating whether the item is enabled, this property can be dynamically changed.

menuItem.visible

A Boolean indicating whether the item is visible, this property can be dynamically changed.

menuItem.checked

A Boolean indicating whether the item is checked, this property can be dynamically changed.

A checkbox menu item will toggle the checked property on and off when selected.

A radio menu item will turn on its checked property when clicked, and will turn off that property for all adjacent items in the same menu.

You can add a click function for additional behavior.

menuItem.label

A String representing the menu items visible label

menuItem.click

A Function that is fired when the MenuItem receives a click event

Class: Menu

Create native application menus and context menus.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

new Menu()

Creates a new menu.

Static Methods

The menu class has the following static methods:

Menu.setApplicationMenu(menu)

* menu Menu

Sets menu as the application menu on macOS. On Windows and Linux, the menu will be set as each window’s top menu.

Passing null will remove the menu bar on Windows and Linux but has no effect on macOS.

**Note:** This API has to be called after the ready event of app module.

Menu.getApplicationMenu()

Returns Menu - The application menu, if set, or null, if not set.

**Note:** The returned Menu instance doesn’t support dynamic addition or removal of menu items. [Instance properties](https://electron.atom.io/docs/all/#instance-properties) can still be dynamically modified.

Menu.sendActionToFirstResponder(action) macOS

* action String

Sends the action to the first responder of application. This is used for emulating default macOS menu behaviors. Usually you would just use the [role](https://electron.atom.io/docs/api/menu-item#roles) property of a [MenuItem](https://electron.atom.io/docs/api/menu-item).

See the [macOS Cocoa Event Handling Guide](https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/EventOverview/EventArchitecture/EventArchitecture.html#//apple_ref/doc/uid/10000060i-CH3-SW7) for more information on macOS’ native actions.

Menu.buildFromTemplate(template)

* template MenuItemConstructorOptions[]

Returns Menu

Generally, the template is just an array of options for constructing a [MenuItem](https://electron.atom.io/docs/api/menu-item). The usage can be referenced above.

You can also attach other fields to the element of the template and they will become properties of the constructed menu items.

Instance Methods

The menu object has the following instance methods:

menu.popup([browserWindow, options])

* browserWindow BrowserWindow (optional) - Default is the focused window.
* options Object (optional)
  + x Number (optional) - Default is the current mouse cursor position. Must be declared if y is declared.
  + y Number (optional) - Default is the current mouse cursor position. Must be declared if x is declared.
  + async Boolean (optional) - Set to true to have this method return immediately called, false to return after the menu has been selected or closed. Defaults to false.
  + positioningItem Number (optional) macOS - The index of the menu item to be positioned under the mouse cursor at the specified coordinates. Default is -1.

Pops up this menu as a context menu in the browserWindow.

menu.closePopup([browserWindow])

* browserWindow BrowserWindow (optional) - Default is the focused window.

Closes the context menu in the browserWindow.

menu.append(menuItem)

* menuItem MenuItem

Appends the menuItem to the menu.

menu.insert(pos, menuItem)

* pos Integer
* menuItem MenuItem

Inserts the menuItem to the pos position of the menu.

Instance Properties

menu objects also have the following properties:

menu.items

A MenuItem[] array containing the menu’s items.

Each Menu consists of multiple [MenuItem](https://electron.atom.io/docs/api/menu-item)s and each MenuItem can have a submenu.

Examples

The Menu class is only available in the main process, but you can also use it in the render process via the [remote](https://electron.atom.io/docs/api/remote)module.

Main process

An example of creating the application menu in the main process with the simple template API:

**const** {app, Menu} = require('electron')

**const** template = [

{

label: 'Edit',

submenu: [

{role: 'undo'},

{role: 'redo'},

{type: 'separator'},

{role: 'cut'},

{role: 'copy'},

{role: 'paste'},

{role: 'pasteandmatchstyle'},

{role: 'delete'},

{role: 'selectall'}

]

},

{

label: 'View',

submenu: [

{role: 'reload'},

{role: 'forcereload'},

{role: 'toggledevtools'},

{type: 'separator'},

{role: 'resetzoom'},

{role: 'zoomin'},

{role: 'zoomout'},

{type: 'separator'},

{role: 'togglefullscreen'}

]

},

{

role: 'window',

submenu: [

{role: 'minimize'},

{role: 'close'}

]

},

{

role: 'help',

submenu: [

{

label: 'Learn More',

click () { require('electron').shell.openExternal('https://electron.atom.io') }

}

]

}

]

**if** (process.platform === 'darwin') {

template.unshift({

label: app.getName(),

submenu: [

{role: 'about'},

{type: 'separator'},

{role: 'services', submenu: []},

{type: 'separator'},

{role: 'hide'},

{role: 'hideothers'},

{role: 'unhide'},

{type: 'separator'},

{role: 'quit'}

]

})

*// Edit menu*

template[1].submenu.push(

{type: 'separator'},

{

label: 'Speech',

submenu: [

{role: 'startspeaking'},

{role: 'stopspeaking'}

]

}

)

*// Window menu*

template[3].submenu = [

{role: 'close'},

{role: 'minimize'},

{role: 'zoom'},

{type: 'separator'},

{role: 'front'}

]

}

**const** menu = Menu.buildFromTemplate(template)

Menu.setApplicationMenu(menu)

Render process

Below is an example of creating a menu dynamically in a web page (render process) by using the [remote](https://electron.atom.io/docs/api/remote)module, and showing it when the user right clicks the page:

*<!-- index.html -->*

<script>

**const** {remote} = require('electron')

**const** {Menu, MenuItem} = remote

**const** menu = **new** Menu()

menu.append(**new** MenuItem({label: 'MenuItem1', click() { console.log('item 1 clicked') }}))

menu.append(**new** MenuItem({type: 'separator'}))

menu.append(**new** MenuItem({label: 'MenuItem2', type: 'checkbox', checked: true}))

window.addEventListener('contextmenu', (e) => {

e.preventDefault()

menu.popup(remote.getCurrentWindow())

}, false)

</script>

Notes on macOS Application Menu

macOS has a completely different style of application menu from Windows and Linux. Here are some notes on making your app’s menu more native-like.

Standard Menus

On macOS there are many system-defined standard menus, like the Services and Windows menus. To make your menu a standard menu, you should set your menu’s role to one of the following and Electron will recognize them and make them become standard menus:

* window
* help
* services

Standard Menu Item Actions

macOS has provided standard actions for some menu items, like About xxx, Hide xxx, and Hide Others. To set the action of a menu item to a standard action, you should set the role attribute of the menu item.

Main Menu’s Name

On macOS the label of the application menu’s first item is always your app’s name, no matter what label you set. To change it, modify your app bundle’s Info.plist file. See [About Information Property List Files](https://developer.apple.com/library/ios/documentation/general/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html) for more information.

Setting Menu for Specific Browser Window (Linux Windows)

The [setMenu method](https://github.com/electron/electron/blob/master/docs/api/browser-window.md#winsetmenumenu-linux-windows) of browser windows can set the menu of certain browser windows.

Menu Item Position

You can make use of position and id to control how the item will be placed when building a menu with Menu.buildFromTemplate.

The position attribute of MenuItem has the form [placement]=[id], where placement is one of before, after, or endof and id is the unique ID of an existing item in the menu:

* before - Inserts this item before the id referenced item. If the referenced item doesn’t exist the item will be inserted at the end of the menu.
* after - Inserts this item after id referenced item. If the referenced item doesn’t exist the item will be inserted at the end of the menu.
* endof - Inserts this item at the end of the logical group containing the id referenced item (groups are created by separator items). If the referenced item doesn’t exist, a new separator group is created with the given id and this item is inserted after that separator.

When an item is positioned, all un-positioned items are inserted after it until a new item is positioned. So if you want to position a group of menu items in the same location you only need to specify a position for the first item.

Examples

Template:

[

{label: '4', id: '4'},

{label: '5', id: '5'},

{label: '1', id: '1', position: 'before=4'},

{label: '2', id: '2'},

{label: '3', id: '3'}

]

Menu:

- 1

- 2

- 3

- 4

- 5

Template:

[

{label: 'a', position: 'endof=letters'},

{label: '1', position: 'endof=numbers'},

{label: 'b', position: 'endof=letters'},

{label: '2', position: 'endof=numbers'},

{label: 'c', position: 'endof=letters'},

{label: '3', position: 'endof=numbers'}

]

Menu:

- ---

- a

- b

- c

- ---

- 1

- 2

- 3

nativeImage

Create tray, dock, and application icons using PNG or JPG files.

Process: [Main](https://electron.atom.io/docs/glossary#main-process), [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

In Electron, for the APIs that take images, you can pass either file paths or NativeImage instances. An empty image will be used when null is passed.

For example, when creating a tray or setting a window’s icon, you can pass an image file path as a String:

**const** {BrowserWindow, Tray} = require('electron')

**const** appIcon = **new** Tray('/Users/somebody/images/icon.png')

**let** win = **new** BrowserWindow({icon: '/Users/somebody/images/window.png'})

console.log(appIcon, win)

Or read the image from the clipboard which returns a NativeImage:

**const** {clipboard, Tray} = require('electron')

**const** image = clipboard.readImage()

**const** appIcon = **new** Tray(image)

console.log(appIcon)

Supported Formats

Currently PNG and JPEG image formats are supported. PNG is recommended because of its support for transparency and lossless compression.

On Windows, you can also load ICO icons from file paths. For best visual quality it is recommended to include at least the following sizes in the:

* Small icon
* 16x16 (100% DPI scale)
* 20x20 (125% DPI scale)
* 24x24 (150% DPI scale)
* 32x32 (200% DPI scale)
* Large icon
* 32x32 (100% DPI scale)
* 40x40 (125% DPI scale)
* 48x48 (150% DPI scale)
* 64x64 (200% DPI scale)
* 256x256

Check the *Size requirements* section in [this article](https://msdn.microsoft.com/en-us/library/windows/desktop/dn742485(v=vs.85).aspx).

High Resolution Image

On platforms that have high-DPI support such as Apple Retina displays, you can append @2x after image’s base filename to mark it as a high resolution image.

For example if icon.png is a normal image that has standard resolution, then icon@2x.png will be treated as a high resolution image that has double DPI density.

If you want to support displays with different DPI densities at the same time, you can put images with different sizes in the same folder and use the filename without DPI suffixes. For example:

images/

├── icon.png

├── icon@2x.png

└── icon@3x.png

**const** {Tray} = require('electron')

**let** appIcon = **new** Tray('/Users/somebody/images/icon.png')

console.log(appIcon)

Following suffixes for DPI are also supported:

* @1x
* @1.25x
* @1.33x
* @1.4x
* @1.5x
* @1.8x
* @2x
* @2.5x
* @3x
* @4x
* @5x

Template Image

Template images consist of black and clear colors (and an alpha channel). Template images are not intended to be used as standalone images and are usually mixed with other content to create the desired final appearance.

The most common case is to use template images for a menu bar icon so it can adapt to both light and dark menu bars.

**Note:** Template image is only supported on macOS.

To mark an image as a template image, its filename should end with the word Template. For example:

* xxxTemplate.png
* xxxTemplate@2x.png

Methods

The nativeImage module has the following methods, all of which return an instance of the NativeImage class:

nativeImage.createEmpty()

Returns NativeImage

Creates an empty NativeImage instance.

nativeImage.createFromPath(path)

* path String

Returns NativeImage

Creates a new NativeImage instance from a file located at path. This method returns an empty image if the path does not exist, cannot be read, or is not a valid image.

**const** nativeImage = require('electron').nativeImage

**let** image = nativeImage.createFromPath('/Users/somebody/images/icon.png')

console.log(image)

nativeImage.createFromBuffer(buffer[, options])

* buffer [Buffer](https://nodejs.org/api/buffer.html#buffer_class_buffer)
* options Object (optional)
  + width Integer (optional) - Required for bitmap buffers.
  + height Integer (optional) - Required for bitmap buffers.
  + scaleFactor Double (optional) - Defaults to 1.0.

Returns NativeImage

Creates a new NativeImage instance from buffer.

nativeImage.createFromDataURL(dataURL)

* dataURL String

Returns NativeImage

Creates a new NativeImage instance from dataURL.

Class: NativeImage

Natively wrap images such as tray, dock, and application icons.

Process: [Main](https://electron.atom.io/docs/glossary#main-process), [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

Instance Methods

The following methods are available on instances of the NativeImage class:

image.toPNG([options])

* options Object (optional)
  + scaleFactor Double (optional) - Defaults to 1.0.

Returns Buffer - A [Buffer](https://nodejs.org/api/buffer.html#buffer_class_buffer) that contains the image’s PNG encoded data.

image.toJPEG(quality)

* quality Integer (**required**) - Between 0 - 100.

Returns Buffer - A [Buffer](https://nodejs.org/api/buffer.html#buffer_class_buffer) that contains the image’s JPEG encoded data.

image.toBitmap([options])

* options Object (optional)
  + scaleFactor Double (optional) - Defaults to 1.0.

Returns Buffer - A [Buffer](https://nodejs.org/api/buffer.html#buffer_class_buffer) that contains a copy of the image’s raw bitmap pixel data.

image.toDataURL([options])

* options Object (optional)
  + scaleFactor Double (optional) - Defaults to 1.0.

Returns String - The data URL of the image.

image.getBitmap([options])

* options Object (optional)
  + scaleFactor Double (optional) - Defaults to 1.0.

Returns Buffer - A [Buffer](https://nodejs.org/api/buffer.html#buffer_class_buffer) that contains the image’s raw bitmap pixel data.

The difference between getBitmap() and toBitmap() is, getBitmap() does not copy the bitmap data, so you have to use the returned Buffer immediately in current event loop tick, otherwise the data might be changed or destroyed.

image.getNativeHandle() macOS

Returns Buffer - A [Buffer](https://nodejs.org/api/buffer.html#buffer_class_buffer) that stores C pointer to underlying native handle of the image. On macOS, a pointer to NSImage instance would be returned.

Notice that the returned pointer is a weak pointer to the underlying native image instead of a copy, so you *must*ensure that the associated nativeImage instance is kept around.

image.isEmpty()

Returns Boolean - Whether the image is empty.

image.getSize()

Returns [Size](https://electron.atom.io/docs/api/structures/size)

image.setTemplateImage(option)

* option Boolean

Marks the image as a template image.

image.isTemplateImage()

Returns Boolean - Whether the image is a template image.

image.crop(rect)

* rect [Rectangle](https://electron.atom.io/docs/api/structures/rectangle) - The area of the image to crop

Returns NativeImage - The cropped image.

image.resize(options)

* options Object
  + width Integer (optional) - Defaults to the image’s width.
  + height Integer (optional) - Defaults to the image’s height
  + quality String (optional) - The desired quality of the resize image. Possible values are good, better or best. The default is best. These values express a desired quality/speed tradeoff. They are translated into an algorithm-specific method that depends on the capabilities (CPU, GPU) of the underlying platform. It is possible for all three methods to be mapped to the same algorithm on a given platform.

Returns NativeImage - The resized image.

If only the height or the width are specified then the current aspect ratio will be preserved in the resized image.

image.getAspectRatio()

Returns Float - The image’s aspect ratio.

image.addRepresentation(options)

* options Object
  + scaleFactor Double - The scale factor to add the image representation for.
  + width Integer (optional) - Defaults to 0. Required if a bitmap buffer is specified as buffer.
  + height Integer (optional) - Defaults to 0. Required if a bitmap buffer is specified as buffer.
  + buffer Buffer (optional) - The buffer containing the raw image data.
  + dataURL String (optional) - The data URL containing either a base 64 encoded PNG or JPEG image.

Add an image representation for a specific scale factor. This can be used to explicitly add different scale factor representations to an image. This can be called on empty images.

net

Issue HTTP/HTTPS requests using Chromium’s native networking library

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

The net module is a client-side API for issuing HTTP(S) requests. It is similar to the [HTTP](https://nodejs.org/api/http.html) and [HTTPS](https://nodejs.org/api/https.html) modules of Node.js but uses Chromium’s native networking library instead of the Node.js implementation, offering better support for web proxies.

The following is a non-exhaustive list of why you may consider using the net module instead of the native Node.js modules:

* Automatic management of system proxy configuration, support of the wpad protocol and proxy pac configuration files.
* Automatic tunneling of HTTPS requests.
* Support for authenticating proxies using basic, digest, NTLM, Kerberos or negotiate authentication schemes.
* Support for traffic monitoring proxies: Fiddler-like proxies used for access control and monitoring.

The net module API has been specifically designed to mimic, as closely as possible, the familiar Node.js API. The API components including classes, methods, properties and event names are similar to those commonly used in Node.js.

For instance, the following example quickly shows how the net API might be used:

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

app.on('ready', () => {

**const** {net} = require('electron')

**const** request = net.request('https://github.com')

request.on('response', (response) => {

console.log(`STATUS: ${response.statusCode}`)

console.log(`HEADERS: ${JSON.stringify(response.headers)}`)

response.on('data', (chunk) => {

console.log(`BODY: ${chunk}`)

})

response.on('end', () => {

console.log('No more data in response.')

})

})

request.end()

})

By the way, it is almost identical to how you would normally use the [HTTP](https://nodejs.org/api/http.html)/[HTTPS](https://nodejs.org/api/https.html) modules of Node.js

The net API can be used only after the application emits the ready event. Trying to use the module before the ready event will throw an error.

Methods

The net module has the following methods:

net.request(options)

* options (Object | String) - The ClientRequest constructor options.

Returns [ClientRequest](https://electron.atom.io/docs/api/client-request)

Creates a [ClientRequest](https://electron.atom.io/docs/api/client-request) instance using the provided options which are directly forwarded to the ClientRequest constructor. The net.request method would be used to issue both secure and insecure HTTP requests according to the specified protocol scheme in the options object.

Notification

Create OS desktop notifications

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

Using in the renderer process

If you want to show Notifications from a renderer process you should use the [HTML5 Notification API](https://electron.atom.io/docs/tutorial/notifications)

Class: Notification

Create OS desktop notifications

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

Notification is an [EventEmitter](http://nodejs.org/api/events.html#events_class_events_eventemitter).

It creates a new Notification with native properties as set by the options.

Static Methods

The Notification class has the following static methods:

Notification.isSupported()

Returns Boolean - Whether or not desktop notifications are supported on the current system

new Notification([options]) Experimental

* options Object
  + title String - A title for the notification, which will be shown at the top of the notification window when it is shown
  + subtitle String - (optional) A subtitle for the notification, which will be displayed below the title. macOS
  + body String - The body text of the notification, which will be displayed below the title or subtitle
  + silent Boolean - (optional) Whether or not to emit an OS notification noise when showing the notification
  + icon [NativeImage](https://electron.atom.io/docs/api/native-image) - (optional) An icon to use in the notification
  + hasReply Boolean - (optional) Whether or not to add an inline reply option to the notification. macOS
  + replyPlaceholder String - (optional) The placeholder to write in the inline reply input field. macOS
  + actions [NotificationAction[]](https://electron.atom.io/docs/api/structures/notification-action) - (optional) Actions to add to the notification. Please read the available actions and limitations in the NotificationAction documentation macOS

Instance Events

Objects created with new Notification emit the following events:

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

Event: ‘show’

Returns:

* event Event

Emitted when the notification is shown to the user, note this could be fired multiple times as a notification can be shown multiple times through the show() method.

Event: ‘click’

Returns:

* event Event

Emitted when the notification is clicked by the user.

Event: ‘close’

Returns:

* event Event

Emitted when the notification is closed by manual intervention from the user.

This event is not guarunteed to be emitted in all cases where the notification is closed.

Event: ‘reply’ macOS

Returns:

* event Event
* reply String - The string the user entered into the inline reply field

Emitted when the user clicks the “Reply” button on a notification with hasReply: true.

Event: ‘action’ macOS

Returns:

* event Event
* index Number - The index of the action that was activated

Instance Methods

Objects created with new Notification have the following instance methods:

notification.show()

Immediately shows the notification to the user, please note this means unlike the HTML5 Notification implementation, simply instantiating a new Notification does not immediately show it to the user, you need to call this method before the OS will display it.

powerMonitor

Monitor power state changes.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

You cannot require or use this module until the ready event of the app module is emitted.

For example:

**const** electron = require('electron')

**const** {app} = electron

app.on('ready', () => {

electron.powerMonitor.on('suspend', () => {

console.log('The system is going to sleep')

})

})

Events

The powerMonitor module emits the following events:

Event: ‘suspend’

Emitted when the system is suspending.

Event: ‘resume’

Emitted when system is resuming.

Event: ‘on-ac’ Windows

Emitted when the system changes to AC power.

Event: ‘on-battery’ Windows

Emitted when system changes to battery power.

powerSaveBlocker

Block the system from entering low-power (sleep) mode.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

For example:

**const** {powerSaveBlocker} = require('electron')

**const** id = powerSaveBlocker.start('prevent-display-sleep')

console.log(powerSaveBlocker.isStarted(id))

powerSaveBlocker.stop(id)

Methods

The powerSaveBlocker module has the following methods:

powerSaveBlocker.start(type)

* type String - Power save blocker type.
  + prevent-app-suspension - Prevent the application from being suspended. Keeps system active but allows screen to be turned off. Example use cases: downloading a file or playing audio.
  + prevent-display-sleep - Prevent the display from going to sleep. Keeps system and screen active. Example use case: playing video.

Returns Integer - The blocker ID that is assigned to this power blocker

Starts preventing the system from entering lower-power mode. Returns an integer identifying the power save blocker.

**Note:** prevent-display-sleep has higher precedence over prevent-app-suspension. Only the highest precedence type takes effect. In other words, prevent-display-sleep always takes precedence over prevent-app-suspension.

For example, an API calling A requests for prevent-app-suspension, and another calling B requests for prevent-display-sleep. prevent-display-sleep will be used until B stops its request. After that, prevent-app-suspensionis used.

powerSaveBlocker.stop(id)

* id Integer - The power save blocker id returned by powerSaveBlocker.start.

Stops the specified power save blocker.

powerSaveBlocker.isStarted(id)

* id Integer - The power save blocker id returned by powerSaveBlocker.start.

Returns Boolean - Whether the corresponding powerSaveBlocker has started.

process

Extensions to process object.

Process: [Main](https://electron.atom.io/docs/glossary#main-process), [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

Electron’s process object is extended from the [Node.js process object](https://nodejs.org/api/process.html). It adds the following events, properties, and methods:

Events

Event: ‘loaded’

Emitted when Electron has loaded its internal initialization script and is beginning to load the web page or the main script.

It can be used by the preload script to add removed Node global symbols back to the global scope when node integration is turned off:

*// preload.js*

**const** \_setImmediate = setImmediate

**const** \_clearImmediate = clearImmediate

process.once('loaded', () => {

global.setImmediate = \_setImmediate

global.clearImmediate = \_clearImmediate

})

Properties

process.defaultApp

A Boolean. When app is started by being passed as parameter to the default app, this property is true in the main process, otherwise it is undefined.

process.mas

A Boolean. For Mac App Store build, this property is true, for other builds it is undefined.

process.noAsar

A Boolean that controls ASAR support inside your application. Setting this to true will disable the support for asar archives in Node’s built-in modules.

process.noDeprecation

A Boolean that controls whether or not deprecation warnings are printed to stderr. Setting this to true will silence deprecation warnings. This property is used instead of the --no-deprecation command line flag.

process.resourcesPath

A String representing the path to the resources directory.

process.throwDeprecation

A Boolean that controls whether or not deprecation warnings will be thrown as exceptions. Setting this to truewill throw errors for deprecations. This property is used instead of the --throw-deprecation command line flag.

process.traceDeprecation

A Boolean that controls whether or not deprecations printed to stderr include their stack trace. Setting this to true will print stack traces for deprecations. This property is instead of the --trace-deprecation command line flag.

process.traceProcessWarnings

A Boolean that controls whether or not process warnings printed to stderr include their stack trace. Setting this to true will print stack traces for process warnings (including deprecations). This property is instead of the --trace-warnings command line flag.

process.type

A String representing the current process’s type, can be "browser" (i.e. main process) or "renderer".

process.versions.chrome

A String representing Chrome’s version string.

process.versions.electron

A String representing Electron’s version string.

process.windowsStore

A Boolean. If the app is running as a Windows Store app (appx), this property is true, for otherwise it is undefined.

Methods

The process object has the following methods:

process.crash()

Causes the main thread of the current process crash.

process.getCPUUsage()

Returns [CPUUsage](https://electron.atom.io/docs/api/structures/cpu-usage)

process.getIOCounters() Windows Linux

Returns [IOCounters](https://electron.atom.io/docs/api/structures/io-counters)

process.getProcessMemoryInfo()

Returns Object:

* workingSetSize Integer - The amount of memory currently pinned to actual physical RAM.
* peakWorkingSetSize Integer - The maximum amount of memory that has ever been pinned to actual physical RAM.
* privateBytes Integer - The amount of memory not shared by other processes, such as JS heap or HTML content.
* sharedBytes Integer - The amount of memory shared between processes, typically memory consumed by the Electron code itself

Returns an object giving memory usage statistics about the current process. Note that all statistics are reported in Kilobytes.

process.getSystemMemoryInfo()

Returns Object:

* total Integer - The total amount of physical memory in Kilobytes available to the system.
* free Integer - The total amount of memory not being used by applications or disk cache.
* swapTotal Integer - The total amount of swap memory in Kilobytes available to the system. Windows Linux
* swapFree Integer - The free amount of swap memory in Kilobytes available to the system. Windows Linux

Returns an object giving memory usage statistics about the entire system. Note that all statistics are reported in Kilobytes.

process.hang()

Causes the main thread of the current process hang.

process.setFdLimit(maxDescriptors) macOS Linux

* maxDescriptors Integer

Sets the file descriptor soft limit to maxDescriptors or the OS hard limit, whichever is lower for the current process.

protocol

Register a custom protocol and intercept existing protocol requests.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

An example of implementing a protocol that has the same effect as the file:// protocol:

**const** {app, protocol} = require('electron')

**const** path = require('path')

app.on('ready', () => {

protocol.registerFileProtocol('atom', (request, callback) => {

**const** url = request.url.substr(7)

callback({path: path.normalize(`${\_\_dirname}/${url}`)})

}, (error) => {

**if** (error) console.error('Failed to register protocol')

})

})

**Note:** All methods unless specified can only be used after the ready event of the app module gets emitted.

Methods

The protocol module has the following methods:

protocol.registerStandardSchemes(schemes[, options])

* schemes String[] - Custom schemes to be registered as standard schemes.
* options Object (optional)
  + secure Boolean (optional) - true to register the scheme as secure. Default false.

A standard scheme adheres to what RFC 3986 calls [generic URI syntax](https://tools.ietf.org/html/rfc3986#section-3). For example http and https are standard schemes, while file is not.

Registering a scheme as standard, will allow relative and absolute resources to be resolved correctly when served. Otherwise the scheme will behave like the file protocol, but without the ability to resolve relative URLs.

For example when you load following page with custom protocol without registering it as standard scheme, the image will not be loaded because non-standard schemes can not recognize relative URLs:

<body>

<img src='test.png'>

</body>

Registering a scheme as standard will allow access to files through the [FileSystem API](https://developer.mozilla.org/en-US/docs/Web/API/LocalFileSystem). Otherwise the renderer will throw a security error for the scheme.

By default web storage apis (localStorage, sessionStorage, webSQL, indexedDB, cookies) are disabled for non standard schemes. So in general if you want to register a custom protocol to replace the http protocol, you have to register it as a standard scheme:

**const** {app, protocol} = require('electron')

protocol.registerStandardSchemes(['atom'])

app.on('ready', () => {

protocol.registerHttpProtocol('atom', '...')

})

**Note:** This method can only be used before the ready event of the app module gets emitted.

protocol.registerServiceWorkerSchemes(schemes)

* schemes String[] - Custom schemes to be registered to handle service workers.

protocol.registerFileProtocol(scheme, handler[, completion])

* scheme String
* handler Function
  + request Object
    - url String
    - referrer String
    - method String
    - uploadData [UploadData[]](https://electron.atom.io/docs/api/structures/upload-data)
  + callback Function
    - filePath String (optional)
* completion Function (optional)
  + error Error

Registers a protocol of scheme that will send the file as a response. The handler will be called with handler(request, callback) when a request is going to be created with scheme. completion will be called with completion(null) when scheme is successfully registered or completion(error) when failed.

To handle the request, the callback should be called with either the file’s path or an object that has a pathproperty, e.g. callback(filePath) or callback({path: filePath}).

When callback is called with nothing, a number, or an object that has an error property, the request will fail with the error number you specified. For the available error numbers you can use, please see the [net error list](https://code.google.com/p/chromium/codesearch#chromium/src/net/base/net_error_list.h).

By default the scheme is treated like http:, which is parsed differently than protocols that follow the “generic URI syntax” like file:, so you probably want to call protocol.registerStandardSchemes to have your scheme treated as a standard scheme.

protocol.registerBufferProtocol(scheme, handler[, completion])

* scheme String
* handler Function
  + request Object
    - url String
    - referrer String
    - method String
    - uploadData [UploadData[]](https://electron.atom.io/docs/api/structures/upload-data)
  + callback Function
    - buffer (Buffer | [MimeTypedBuffer](https://electron.atom.io/docs/api/structures/mime-typed-buffer)) (optional)
* completion Function (optional)
  + error Error

Registers a protocol of scheme that will send a Buffer as a response.

The usage is the same with registerFileProtocol, except that the callback should be called with either a Buffer object or an object that has the data, mimeType, and charset properties.

Example:

**const** {protocol} = require('electron')

protocol.registerBufferProtocol('atom', (request, callback) => {

callback({mimeType: 'text/html', data: Buffer.from('<h5>Response</h5>')})

}, (error) => {

**if** (error) console.error('Failed to register protocol')

})

protocol.registerStringProtocol(scheme, handler[, completion])

* scheme String
* handler Function
  + request Object
    - url String
    - referrer String
    - method String
    - uploadData [UploadData[]](https://electron.atom.io/docs/api/structures/upload-data)
  + callback Function
    - data String (optional)
* completion Function (optional)
  + error Error

Registers a protocol of scheme that will send a String as a response.

The usage is the same with registerFileProtocol, except that the callback should be called with either a String or an object that has the data, mimeType, and charset properties.

protocol.registerHttpProtocol(scheme, handler[, completion])

* scheme String
* handler Function
  + request Object
    - url String
    - referrer String
    - method String
    - uploadData [UploadData[]](https://electron.atom.io/docs/api/structures/upload-data)
  + callback Function
    - redirectRequest Object
      * url String
      * method String
      * session Object (optional)
      * uploadData Object (optional)
        + contentType String - MIME type of the content.
        + data String - Content to be sent.
* completion Function (optional)
  + error Error

Registers a protocol of scheme that will send an HTTP request as a response.

The usage is the same with registerFileProtocol, except that the callback should be called with a redirectRequest object that has the url, method, referrer, uploadData and session properties.

By default the HTTP request will reuse the current session. If you want the request to have a different session you should set session to null.

For POST requests the uploadData object must be provided.

protocol.unregisterProtocol(scheme[, completion])

* scheme String
* completion Function (optional)
  + error Error

Unregisters the custom protocol of scheme.

protocol.isProtocolHandled(scheme, callback)

* scheme String
* callback Function
  + error Error

The callback will be called with a boolean that indicates whether there is already a handler for scheme.

protocol.interceptFileProtocol(scheme, handler[, completion])

* scheme String
* handler Function
  + request Object
    - url String
    - referrer String
    - method String
    - uploadData [UploadData[]](https://electron.atom.io/docs/api/structures/upload-data)
  + callback Function
    - filePath String
* completion Function (optional)
  + error Error

Intercepts scheme protocol and uses handler as the protocol’s new handler which sends a file as a response.

protocol.interceptStringProtocol(scheme, handler[, completion])

* scheme String
* handler Function
  + request Object
    - url String
    - referrer String
    - method String
    - uploadData [UploadData[]](https://electron.atom.io/docs/api/structures/upload-data)
  + callback Function
    - data String (optional)
* completion Function (optional)
  + error Error

Intercepts scheme protocol and uses handler as the protocol’s new handler which sends a String as a response.

protocol.interceptBufferProtocol(scheme, handler[, completion])

* scheme String
* handler Function
  + request Object
    - url String
    - referrer String
    - method String
    - uploadData [UploadData[]](https://electron.atom.io/docs/api/structures/upload-data)
  + callback Function
    - buffer Buffer (optional)
* completion Function (optional)
  + error Error

Intercepts scheme protocol and uses handler as the protocol’s new handler which sends a Buffer as a response.

protocol.interceptHttpProtocol(scheme, handler[, completion])

* scheme String
* handler Function
  + request Object
    - url String
    - referrer String
    - method String
    - uploadData [UploadData[]](https://electron.atom.io/docs/api/structures/upload-data)
  + callback Function
    - redirectRequest Object
      * url String
      * method String
      * session Object (optional)
      * uploadData Object (optional)
        + contentType String - MIME type of the content.
        + data String - Content to be sent.
* completion Function (optional)
  + error Error

Intercepts scheme protocol and uses handler as the protocol’s new handler which sends a new HTTP request as a response.

protocol.uninterceptProtocol(scheme[, completion])

* scheme String
* completion Function (optional)
  + error Error

Remove the interceptor installed for scheme and restore its original handler.

remote

Use main process modules from the renderer process.

Process: [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

The remote module provides a simple way to do inter-process communication (IPC) between the renderer process (web page) and the main process.

In Electron, GUI-related modules (such as dialog, menu etc.) are only available in the main process, not in the renderer process. In order to use them from the renderer process, the ipc module is necessary to send inter-process messages to the main process. With the remote module, you can invoke methods of the main process object without explicitly sending inter-process messages, similar to Java’s [RMI](http://en.wikipedia.org/wiki/Java_remote_method_invocation). An example of creating a browser window from a renderer process:

**const** {BrowserWindow} = require('electron').remote

**let** win = **new** BrowserWindow({width: 800, height: 600})

win.loadURL('https://github.com')

**Note:** For the reverse (access the renderer process from the main process), you can use [webContents.executeJavascript](https://electron.atom.io/docs/api/web-contents#contentsexecutejavascriptcode-usergesture-callback).

Remote Objects

Each object (including functions) returned by the remote module represents an object in the main process (we call it a remote object or remote function). When you invoke methods of a remote object, call a remote function, or create a new object with the remote constructor (function), you are actually sending synchronous inter-process messages.

In the example above, both BrowserWindow and win were remote objects and new BrowserWindow didn’t create a BrowserWindow object in the renderer process. Instead, it created a BrowserWindow object in the main process and returned the corresponding remote object in the renderer process, namely the win object.

**Note:** Only [enumerable properties](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Enumerability_and_ownership_of_properties) which are present when the remote object is first referenced are accessible via remote.

**Note:** Arrays and Buffers are copied over IPC when accessed via the remote module. Modifying them in the renderer process does not modify them in the main process and vice versa.

Lifetime of Remote Objects

Electron makes sure that as long as the remote object in the renderer process lives (in other words, has not been garbage collected), the corresponding object in the main process will not be released. When the remote object has been garbage collected, the corresponding object in the main process will be dereferenced.

If the remote object is leaked in the renderer process (e.g. stored in a map but never freed), the corresponding object in the main process will also be leaked, so you should be very careful not to leak remote objects.

Primary value types like strings and numbers, however, are sent by copy.

Passing callbacks to the main process

Code in the main process can accept callbacks from the renderer - for instance the remote module - but you should be extremely careful when using this feature.

First, in order to avoid deadlocks, the callbacks passed to the main process are called asynchronously. You should not expect the main process to get the return value of the passed callbacks.

For instance you can’t use a function from the renderer process in an Array.map called in the main process:

*// main process mapNumbers.js*

exports.withRendererCallback = (mapper) => {

**return** [1, 2, 3].map(mapper)

}

exports.withLocalCallback = () => {

**return** [1, 2, 3].map(x => x + 1)

}

*// renderer process*

**const** mapNumbers = require('electron').remote.require('./mapNumbers')

**const** withRendererCb = mapNumbers.withRendererCallback(x => x + 1)

**const** withLocalCb = mapNumbers.withLocalCallback()

console.log(withRendererCb, withLocalCb)

*// [undefined, undefined, undefined], [2, 3, 4]*

As you can see, the renderer callback’s synchronous return value was not as expected, and didn’t match the return value of an identical callback that lives in the main process.

Second, the callbacks passed to the main process will persist until the main process garbage-collects them.

For example, the following code seems innocent at first glance. It installs a callback for the close event on a remote object:

require('electron').remote.getCurrentWindow().on('close', () => {

*// window was closed...*

})

But remember the callback is referenced by the main process until you explicitly uninstall it. If you do not, each time you reload your window the callback will be installed again, leaking one callback for each restart.

To make things worse, since the context of previously installed callbacks has been released, exceptions will be raised in the main process when the close event is emitted.

To avoid this problem, ensure you clean up any references to renderer callbacks passed to the main process. This involves cleaning up event handlers, or ensuring the main process is explicitly told to deference callbacks that came from a renderer process that is exiting.

Accessing built-in modules in the main process

The built-in modules in the main process are added as getters in the remote module, so you can use them directly like the electron module.

**const** app = require('electron').remote.app

console.log(app)

Methods

The remote module has the following methods:

remote.require(module)

* module String

Returns any - The object returned by require(module) in the main process. Modules specified by their relative path will resolve relative to the entrypoint of the main process.

e.g.

project/

├── main

│   ├── foo.js

│   └── index.js

├── package.json

└── renderer

└── index.js

*// main process: main/index.js*

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

app.on('ready', () => { */\* ... \*/* })

*// some relative module: main/foo.js*

module.exports = 'bar'

*// renderer process: renderer/index.js*

**const** foo = require('electron').remote.require('./foo') *// bar*

remote.getCurrentWindow()

Returns [BrowserWindow](https://electron.atom.io/docs/api/browser-window) - The window to which this web page belongs.

remote.getCurrentWebContents()

Returns [WebContents](https://electron.atom.io/docs/api/web-contents) - The web contents of this web page.

remote.getGlobal(name)

* name String

Returns any - The global variable of name (e.g. global[name]) in the main process.

Properties

remote.process

The process object in the main process. This is the same as remote.getGlobal('process') but is cached.

sandbox Option

Create a browser window with a renderer that can run inside Chromium OS sandbox. With this option enabled, the renderer must communicate via IPC to the main process in order to access node APIs. However, in order to enable the Chromium OS sandbox, electron must be run with the --enable-sandbox command line argument.

One of the key security features of Chromium is that all blink rendering/JavaScript code is executed within a sandbox. This sandbox uses OS-specific features to ensure that exploits in the renderer process cannot harm the system.

In other words, when the sandbox is enabled, the renderers can only make changes to the system by delegating tasks to the main process via IPC. [Here’s](https://www.chromium.org/developers/design-documents/sandbox) more information about the sandbox.

Since a major feature in electron is the ability to run node.js in the renderer process (making it easier to develop desktop applications using web technologies), the sandbox is disabled by electron. This is because most node.js APIs require system access. require() for example, is not possible without file system permissions, which are not available in a sandboxed environment.

Usually this is not a problem for desktop applications since the code is always trusted, but it makes electron less secure than chromium for displaying untrusted web content. For applications that require more security, the sandbox flag will force electron to spawn a classic chromium renderer that is compatible with the sandbox.

A sandboxed renderer doesn’t have a node.js environment running and doesn’t expose node.js JavaScript APIs to client code. The only exception is the preload script, which has access to a subset of the electron renderer API.

Another difference is that sandboxed renderers don’t modify any of the default JavaScript APIs. Consequently, some APIs such as window.open will work as they do in chromium (i.e. they do not return a BrowserWindowProxy).

Example

To create a sandboxed window, simply pass sandbox: true to webPreferences:

**let** win

app.on('ready', () => {

win = **new** BrowserWindow({

webPreferences: {

sandbox: true

}

})

w.loadURL('http://google.com')

})

In the above code the BrowserWindow that was created has node.js disabled and can communicate only via IPC. The use of this option stops electron from creating a node.js runtime in the renderer. Also, within this new window window.open follows the native behaviour (by default electron creates a BrowserWindow and returns a proxy to this via window.open).

It is important to note that this option alone won’t enable the OS-enforced sandbox. To enable this feature, the --enable-sandbox command-line argument must be passed to electron, which will force sandbox: true for all BrowserWindow instances.

To enable OS-enforced sandbox on BrowserWindow or webview process with sandbox:true without causing entire app to be in sandbox, --enable-mixed-sandbox command-line argument must be passed to electron. This option is currently only supported on macOS and Windows.

**let** win

app.on('ready', () => {

*// no need to pass `sandbox: true` since `--enable-sandbox` was enabled.*

win = **new** BrowserWindow()

w.loadURL('http://google.com')

})

Note that it is not enough to call app.commandLine.appendSwitch('--enable-sandbox'), as electron/node startup code runs after it is possible to make changes to chromium sandbox settings. The switch must be passed to electron on the command-line:

electron --enable-sandbox app.js

It is not possible to have the OS sandbox active only for some renderers, if --enable-sandbox is enabled, normal electron windows cannot be created.

If you need to mix sandboxed and non-sandboxed renderers in one application, simply omit the --enable-sandbox argument. Without this argument, windows created with sandbox: true will still have node.js disabled and communicate only via IPC, which by itself is already a gain from security POV.

Preload

An app can make customizations to sandboxed renderers using a preload script. Here’s an example:

**let** win

app.on('ready', () => {

win = **new** BrowserWindow({

webPreferences: {

sandbox: true,

preload: 'preload.js'

}

})

w.loadURL('http://google.com')

})

and preload.js:

*// This file is loaded whenever a javascript context is created. It runs in a*

*// private scope that can access a subset of electron renderer APIs. We must be*

*// careful to not leak any objects into the global scope!*

**const** fs = require('fs')

**const** {ipcRenderer} = require('electron')

*// read a configuration file using the `fs` module*

**const** buf = fs.readFileSync('allowed-popup-urls.json')

**const** allowedUrls = JSON.parse(buf.toString('utf8'))

**const** defaultWindowOpen = window.open

**function** customWindowOpen (url, ...args) {

**if** (allowedUrls.indexOf(url) === -1) {

ipcRenderer.sendSync('blocked-popup-notification', location.origin, url)

**return** null

}

**return** defaultWindowOpen(url, ...args)

}

window.open = customWindowOpen

Important things to notice in the preload script:

* Even though the sandboxed renderer doesn’t have node.js running, it still has access to a limited node-like environment: Buffer, process, setImmediate and require are available.
* The preload script can indirectly access all APIs from the main process through the remote and ipcRenderermodules. This is how fs (used above) and other modules are implemented: They are proxies to remote counterparts in the main process.
* The preload script must be contained in a single script, but it is possible to have complex preload code composed with multiple modules by using a tool like browserify, as explained below. In fact, browserify is already used by electron to provide a node-like environment to the preload script.

To create a browserify bundle and use it as a preload script, something like the following should be used:

browserify preload/index.js \

-x electron \

-x fs \

--insert-global-vars=\_\_filename,\_\_dirname -o preload.js

The -x flag should be used with any required module that is already exposed in the preload scope, and tells browserify to use the enclosing require function for it. --insert-global-vars will ensure that process, Bufferand setImmediate are also taken from the enclosing scope(normally browserify injects code for those).

Currently the require function provided in the preload scope exposes the following modules:

* child\_process
* electron (crashReporter, remote and ipcRenderer)
* fs
* os
* timers
* url

More may be added as needed to expose more electron APIs in the sandbox, but any module in the main process can already be used through electron.remote.require.

Status

Please use the sandbox option with care, as it is still an experimental feature. We are still not aware of the security implications of exposing some electron renderer APIs to the preload script, but here are some things to consider before rendering untrusted content:

* A preload script can accidentaly leak privileged APIs to untrusted code.
* Some bug in V8 engine may allow malicious code to access the renderer preload APIs, effectively granting full access to the system through the remote module.

Since rendering untrusted content in electron is still uncharted territory, the APIs exposed to the sandbox preload script should be considered more unstable than the rest of electron APIs, and may have breaking changes to fix security issues.

One planned enhancement that should greatly increase security is to block IPC messages from sandboxed renderers by default, allowing the main process to explicitly define a set of messages the renderer is allowed to send.

screen

Retrieve information about screen size, displays, cursor position, etc.

Process: [Main](https://electron.atom.io/docs/glossary#main-process), [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

You cannot require or use this module until the ready event of the app module is emitted.

screen is an [EventEmitter](https://nodejs.org/api/events.html#events_class_eventemitter).

**Note:** In the renderer / DevTools, window.screen is a reserved DOM property, so writing let {screen} = require('electron') will not work.

An example of creating a window that fills the whole screen:

**const** electron = require('electron')

**const** {app, BrowserWindow} = electron

**let** win

app.on('ready', () => {

**const** {width, height} = electron.screen.getPrimaryDisplay().workAreaSize

win = **new** BrowserWindow({width, height})

win.loadURL('https://github.com')

})

Another example of creating a window in the external display:

**const** electron = require('electron')

**const** {app, BrowserWindow} = require('electron')

**let** win

app.on('ready', () => {

**let** displays = electron.screen.getAllDisplays()

**let** externalDisplay = displays.find((display) => {

**return** display.bounds.x !== 0 || display.bounds.y !== 0

})

**if** (externalDisplay) {

win = **new** BrowserWindow({

x: externalDisplay.bounds.x + 50,

y: externalDisplay.bounds.y + 50

})

win.loadURL('https://github.com')

}

})

Events

The screen module emits the following events:

Event: ‘display-added’

Returns:

* event Event
* newDisplay [Display](https://electron.atom.io/docs/api/structures/display)

Emitted when newDisplay has been added.

Event: ‘display-removed’

Returns:

* event Event
* oldDisplay [Display](https://electron.atom.io/docs/api/structures/display)

Emitted when oldDisplay has been removed.

Event: ‘display-metrics-changed’

Returns:

* event Event
* display [Display](https://electron.atom.io/docs/api/structures/display)
* changedMetrics String[]

Emitted when one or more metrics change in a display. The changedMetrics is an array of strings that describe the changes. Possible changes are bounds, workArea, scaleFactor and rotation.

Methods

The screen module has the following methods:

screen.getCursorScreenPoint()

Returns [Point](https://electron.atom.io/docs/api/structures/point)

The current absolute position of the mouse pointer.

screen.getMenuBarHeight() macOS

Returns Integer - The height of the menu bar in pixels.

screen.getPrimaryDisplay()

Returns [Display](https://electron.atom.io/docs/api/structures/display) - The primary display.

screen.getAllDisplays()

Returns [Display[]](https://electron.atom.io/docs/api/structures/display) - An array of displays that are currently available.

screen.getDisplayNearestPoint(point)

* point [Point](https://electron.atom.io/docs/api/structures/point)

Returns [Display](https://electron.atom.io/docs/api/structures/display) - The display nearest the specified point.

screen.getDisplayMatching(rect)

* rect [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)

Returns [Display](https://electron.atom.io/docs/api/structures/display) - The display that most closely intersects the provided bounds.

session

Manage browser sessions, cookies, cache, proxy settings, etc.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

The session module can be used to create new Session objects.

You can also access the session of existing pages by using the session property of [WebContents](https://electron.atom.io/docs/api/web-contents), or from the session module.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({width: 800, height: 600})

win.loadURL('http://github.com')

**const** ses = win.webContents.session

console.log(ses.getUserAgent())

Methods

The session module has the following methods:

session.fromPartition(partition[, options])

* partition String
* options Object
  + cache Boolean - Whether to enable cache.

Returns Session - A session instance from partition string. When there is an existing Session with the same partition, it will be returned; otherwise a new Session instance will be created with options.

If partition starts with persist:, the page will use a persistent session available to all pages in the app with the same partition. if there is no persist: prefix, the page will use an in-memory session. If the partition is empty then default session of the app will be returned.

To create a Session with options, you have to ensure the Session with the partition has never been used before. There is no way to change the options of an existing Session object.

Properties

The session module has the following properties:

session.defaultSession

A Session object, the default session object of the app.

Class: Session

Get and set properties of a session.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

You can create a Session object in the session module:

**const** {session} = require('electron')

**const** ses = session.fromPartition('persist:name')

console.log(ses.getUserAgent())

Instance Events

The following events are available on instances of Session:

Event: ‘will-download’

* event Event
* item [DownloadItem](https://electron.atom.io/docs/api/download-item)
* webContents [WebContents](https://electron.atom.io/docs/api/web-contents)

Emitted when Electron is about to download item in webContents.

Calling event.preventDefault() will cancel the download and item will not be available from next tick of the process.

**const** {session} = require('electron')

session.defaultSession.on('will-download', (event, item, webContents) => {

event.preventDefault()

require('request')(item.getURL(), (data) => {

require('fs').writeFileSync('/somewhere', data)

})

})

Instance Methods

The following methods are available on instances of Session:

ses.getCacheSize(callback)

* callback Function
  + size Integer - Cache size used in bytes.

Callback is invoked with the session’s current cache size.

ses.clearCache(callback)

* callback Function - Called when operation is done

Clears the session’s HTTP cache.

ses.clearStorageData([options, callback])

* options Object (optional)
  + origin String - (optional) Should follow window.location.origin’s representation scheme://host:port.
  + storages String[] - (optional) The types of storages to clear, can contain: appcache, cookies, filesystem, indexdb, localstorage, shadercache, websql, serviceworkers
  + quotas String[] - (optional) The types of quotas to clear, can contain: temporary, persistent, syncable.
* callback Function (optional) - Called when operation is done.

Clears the data of web storages.

ses.flushStorageData()

Writes any unwritten DOMStorage data to disk.

ses.setProxy(config, callback)

* config Object
  + pacScript String - The URL associated with the PAC file.
  + proxyRules String - Rules indicating which proxies to use.
  + proxyBypassRules String - Rules indicating which URLs should bypass the proxy settings.
* callback Function - Called when operation is done.

Sets the proxy settings.

When pacScript and proxyRules are provided together, the proxyRules option is ignored and pacScriptconfiguration is applied.

The proxyRules has to follow the rules below:

proxyRules = schemeProxies[";"<schemeProxies>]

schemeProxies = [<urlScheme>"="]<proxyURIList>

urlScheme = "http" | "https" | "ftp" | "socks"

proxyURIList = <proxyURL>[","<proxyURIList>]

proxyURL = [<proxyScheme>"://"]<proxyHost>[":"<proxyPort>]

For example:

* http=foopy:80;ftp=foopy2 - Use HTTP proxy foopy:80 for http:// URLs, and HTTP proxy foopy2:80 for ftp:// URLs.
* foopy:80 - Use HTTP proxy foopy:80 for all URLs.
* foopy:80,bar,direct:// - Use HTTP proxy foopy:80 for all URLs, failing over to bar if foopy:80 is unavailable, and after that using no proxy.
* socks4://foopy - Use SOCKS v4 proxy foopy:1080 for all URLs.
* http=foopy,socks5://bar.com - Use HTTP proxy foopy for http URLs, and fail over to the SOCKS5 proxy bar.com if foopy is unavailable.
* http=foopy,direct:// - Use HTTP proxy foopy for http URLs, and use no proxy if foopy is unavailable.
* http=foopy;socks=foopy2 - Use HTTP proxy foopy for http URLs, and use socks4://foopy2 for all other URLs.

The proxyBypassRules is a comma separated list of rules described below:

* [ URL\_SCHEME "://" ] HOSTNAME\_PATTERN [ ":" <port> ]

Match all hostnames that match the pattern HOSTNAME\_PATTERN.

Examples: “foobar.com”, “*foobar.com”, “*.foobar.com”, “*foobar.com:99”, “https://x.*.y.com:99”

* "." HOSTNAME\_SUFFIX\_PATTERN [ ":" PORT ]

Match a particular domain suffix.

Examples: “.google.com”, “.com”, “http://.google.com”

* [ SCHEME "://" ] IP\_LITERAL [ ":" PORT ]

Match URLs which are IP address literals.

Examples: “127.0.1”, “[0:0::1]”, “[::1]”, “http://[::1]:99”

* IP\_LITERAL "/" PREFIX\_LENGHT\_IN\_BITS

Match any URL that is to an IP literal that falls between the given range. IP range is specified using CIDR notation.

Examples: “192.168.1.1/16”, “fefe:13::abc/33”.

* <local>

Match local addresses. The meaning of <local> is whether the host matches one of: “127.0.0.1”, “::1”, “localhost”.

ses.resolveProxy(url, callback)

* url URL
* callback Function
  + proxy String

Resolves the proxy information for url. The callback will be called with callback(proxy) when the request is performed.

ses.setDownloadPath(path)

* path String - The download location

Sets download saving directory. By default, the download directory will be the Downloads under the respective app folder.

ses.enableNetworkEmulation(options)

* options Object
  + offline Boolean (optional) - Whether to emulate network outage. Defaults to false.
  + latency Double (optional) - RTT in ms. Defaults to 0 which will disable latency throttling.
  + downloadThroughput Double (optional) - Download rate in Bps. Defaults to 0 which will disable download throttling.
  + uploadThroughput Double (optional) - Upload rate in Bps. Defaults to 0 which will disable upload throttling.

Emulates network with the given configuration for the session.

*// To emulate a GPRS connection with 50kbps throughput and 500 ms latency.*

window.webContents.session.enableNetworkEmulation({

latency: 500,

downloadThroughput: 6400,

uploadThroughput: 6400

})

*// To emulate a network outage.*

window.webContents.session.enableNetworkEmulation({offline: true})

ses.disableNetworkEmulation()

Disables any network emulation already active for the session. Resets to the original network configuration.

ses.setCertificateVerifyProc(proc)

* proc Function
  + request Object
    - hostname String
    - certificate [Certificate](https://electron.atom.io/docs/api/structures/certificate)
    - error String - Verification result from chromium.
  + callback Function
    - verificationResult Integer - Value can be one of certificate error codes from [here](https://code.google.com/p/chromium/codesearch#chromium/src/net/base/net_error_list.h). Apart from the certificate error codes, the following special codes can be used.
      * 0 - Indicates success and disables Certificate Transperancy verification.
      * -2 - Indicates failure.
      * -3 - Uses the verification result from chromium.

Sets the certificate verify proc for session, the proc will be called with proc(request, callback) whenever a server certificate verification is requested. Calling callback(0) accepts the certificate, calling callback(-2)rejects it.

Calling setCertificateVerifyProc(null) will revert back to default certificate verify proc.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.webContents.session.setCertificateVerifyProc((request, callback) => {

**const** {hostname} = request

**if** (hostname === 'github.com') {

callback(0)

} **else** {

callback(-2)

}

})

ses.setPermissionRequestHandler(handler)

* handler Function
  + webContents [WebContents](https://electron.atom.io/docs/api/web-contents) - WebContents requesting the permission.
  + permission String - Enum of ‘media’, ‘geolocation’, ‘notifications’, ‘midiSysex’, ‘pointerLock’, ‘fullscreen’, ‘openExternal’.
  + callback Function
    - permissionGranted Boolean - Allow or deny the permission

Sets the handler which can be used to respond to permission requests for the session. Calling callback(true)will allow the permission and callback(false) will reject it.

**const** {session} = require('electron')

session.fromPartition('some-partition').setPermissionRequestHandler((webContents, permission, callback) => {

**if** (webContents.getURL() === 'some-host' && permission === 'notifications') {

**return** callback(false) *// denied.*

}

callback(true)

})

ses.clearHostResolverCache([callback])

* callback Function (optional) - Called when operation is done.

Clears the host resolver cache.

ses.allowNTLMCredentialsForDomains(domains)

* domains String - A comma-seperated list of servers for which integrated authentication is enabled.

Dynamically sets whether to always send credentials for HTTP NTLM or Negotiate authentication.

**const** {session} = require('electron')

*// consider any url ending with `example.com`, `foobar.com`, `baz`*

*// for integrated authentication.*

session.defaultSession.allowNTLMCredentialsForDomains('\*example.com, \*foobar.com, \*baz')

*// consider all urls for integrated authentication.*

session.defaultSession.allowNTLMCredentialsForDomains('\*')

ses.setUserAgent(userAgent[, acceptLanguages])

* userAgent String
* acceptLanguages String (optional)

Overrides the userAgent and acceptLanguages for this session.

The acceptLanguages must a comma separated ordered list of language codes, for example "en-US,fr,de,ko,zh-CN,ja".

This doesn’t affect existing WebContents, and each WebContents can use webContents.setUserAgent to override the session-wide user agent.

ses.getUserAgent()

Returns String - The user agent for this session.

ses.getBlobData(identifier, callback)

* identifier String - Valid UUID.
* callback Function
  + result Buffer - Blob data.

Returns Blob - The blob data associated with the identifier.

ses.createInterruptedDownload(options)

* options Object
  + path String - Absolute path of the download.
  + urlChain String[] - Complete URL chain for the download.
  + mimeType String (optional)
  + offset Integer - Start range for the download.
  + length Integer - Total length of the download.
  + lastModified String - Last-Modified header value.
  + eTag String - ETag header value.
  + startTime Double (optional) - Time when download was started in number of seconds since UNIX epoch.

Allows resuming cancelled or interrupted downloads from previous Session. The API will generate a [DownloadItem](https://electron.atom.io/docs/api/download-item) that can be accessed with the [will-download](https://electron.atom.io/docs/all/#event-will-download) event. The [DownloadItem](https://electron.atom.io/docs/api/download-item) will not have any WebContents associated with it and the initial state will be interrupted. The download will start only when the resume API is called on the [DownloadItem](https://electron.atom.io/docs/api/download-item).

ses.clearAuthCache(options[, callback])

* options ([RemovePassword](https://electron.atom.io/docs/api/structures/remove-password) | [RemoveClientCertificate](https://electron.atom.io/docs/api/structures/remove-client-certificate))
* callback Function (optional) - Called when operation is done

Clears the session’s HTTP authentication cache.

Instance Properties

The following properties are available on instances of Session:

ses.cookies

A [Cookies](https://electron.atom.io/docs/api/cookies) object for this session.

ses.webRequest

A [WebRequest](https://electron.atom.io/docs/api/web-request) object for this session.

ses.protocol

A [Protocol](https://electron.atom.io/docs/api/protocol) object for this session.

**const** {app, session} = require('electron')

**const** path = require('path')

app.on('ready', **function** () {

**const** protocol = session.fromPartition('some-partition').protocol

protocol.registerFileProtocol('atom', **function** (request, callback) {

**var** url = request.url.substr(7)

callback({path: path.normalize(`${\_\_dirname}/${url}`)})

}, **function** (error) {

**if** (error) console.error('Failed to register protocol')

})

})

shell

Manage files and URLs using their default applications.

Process: [Main](https://electron.atom.io/docs/glossary#main-process), [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

The shell module provides functions related to desktop integration.

An example of opening a URL in the user’s default browser:

**const** {shell} = require('electron')

shell.openExternal('https://github.com')

Methods

The shell module has the following methods:

shell.showItemInFolder(fullPath)

* fullPath String

Returns Boolean - Whether the item was successfully shown

Show the given file in a file manager. If possible, select the file.

shell.openItem(fullPath)

* fullPath String

Returns Boolean - Whether the item was successfully opened.

Open the given file in the desktop’s default manner.

shell.openExternal(url[, options, callback])

* url String
* options Object (optional) macOS
  + activate Boolean - true to bring the opened application to the foreground. The default is true.
* callback Function (optional) - If specified will perform the open asynchronously. macOS
  + error Error

Returns Boolean - Whether an application was available to open the URL. If callback is specified, always returns true.

Open the given external protocol URL in the desktop’s default manner. (For example, mailto: URLs in the user’s default mail agent).

shell.moveItemToTrash(fullPath)

* fullPath String

Returns Boolean - Whether the item was successfully moved to the trash

Move the given file to trash and returns a boolean status for the operation.

shell.beep()

Play the beep sound.

shell.writeShortcutLink(shortcutPath[, operation], options) Windows

* shortcutPath String
* operation String (optional) - Default is create, can be one of following:
  + create - Creates a new shortcut, overwriting if necessary.
  + update - Updates specified properties only on an existing shortcut.
  + replace - Overwrites an existing shortcut, fails if the shortcut doesn’t exist.
* options [ShortcutDetails](https://electron.atom.io/docs/api/structures/shortcut-details)

Returns Boolean - Whether the shortcut was created successfully

Creates or updates a shortcut link at shortcutPath.

shell.readShortcutLink(shortcutPath) Windows

* shortcutPath String

Returns [ShortcutDetails](https://electron.atom.io/docs/api/structures/shortcut-details)

Resolves the shortcut link at shortcutPath.

An exception will be thrown when any error happens.

BluetoothDevice Object

* deviceName String
* deviceId String

CertificatePrincipal Object

* commonName String - Common Name
* organizations String[] - Organization names
* organizationUnits String[] - Organization Unit names
* locality String - Locality
* state String - State or province
* country String - Country or region

Certificate Object

* data String - PEM encoded data
* issuer [CertificatePrincipal](https://electron.atom.io/docs/api/structures/certificate-principal) - Issuer principal
* issuerName String - Issuer’s Common Name
* issuerCert Certificate - Issuer certificate (if not self-signed)
* subject [CertificatePrincipal](https://electron.atom.io/docs/api/structures/certificate-principal) - Subject principal
* subjectName String - Subject’s Common Name
* serialNumber String - Hex value represented string
* validStart Number - Start date of the certificate being valid in seconds
* validExpiry Number - End date of the certificate being valid in seconds
* fingerprint String - Fingerprint of the certificate

Cookie Object

* name String - The name of the cookie.
* value String - The value of the cookie.
* domain String (optional) - The domain of the cookie.
* hostOnly Boolean (optional) - Whether the cookie is a host-only cookie.
* path String (optional) - The path of the cookie.
* secure Boolean (optional) - Whether the cookie is marked as secure.
* httpOnly Boolean (optional) - Whether the cookie is marked as HTTP only.
* session Boolean (optional) - Whether the cookie is a session cookie or a persistent cookie with an expiration date.
* expirationDate Double (optional) - The expiration date of the cookie as the number of seconds since the UNIX epoch. Not provided for session cookies.

CPUUsage Object

* percentCPUUsage Number - Percentage of CPU used since the last call to getCPUUsage. First call returns 0.
* idleWakeupsPerSecond Number - The number of average idle cpu wakeups per second since the last call to getCPUUsage. First call returns 0.

CrashReport Object

* date String
* ID Integer

DesktopCapturerSource Object

* id String - The identifier of a window or screen that can be used as a chromeMediaSourceId constraint when calling [navigator.webkitGetUserMedia]. The format of the identifier will be window:XX or screen:XX, where XX is a random generated number.
* name String - A screen source will be named either Entire Screen or Screen <index>, while the name of a window source will match the window title.
* thumbnail [NativeImage](https://electron.atom.io/docs/api/native-image) - A thumbnail image. **Note:** There is no guarantee that the size of the thumbnail is the same as the thumbnailSize specified in the options passed to desktopCapturer.getSources. The actual size depends on the scale of the screen or window.

Display Object

* id Number - Unique identifier associated with the display.
* rotation Number - Can be 0, 90, 180, 270, represents screen rotation in clock-wise degrees.
* scaleFactor Number - Output device’s pixel scale factor.
* touchSupport String - Can be available, unavailable, unknown.
* bounds [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)
* size [Size](https://electron.atom.io/docs/api/structures/size)
* workArea [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)
* workAreaSize [Size](https://electron.atom.io/docs/api/structures/size)

The Display object represents a physical display connected to the system. A fake Display may exist on a headless system, or a Display may correspond to a remote, virtual display.

FileFilter Object

* name String
* extensions String[]

GPUFeatureStatus Object

* 2d\_canvas String - Canvas
* flash\_3d String - Flash
* flash\_stage3d String - Flash Stage3D
* flash\_stage3d\_baseline String - Flash Stage3D Baseline profile
* gpu\_compositing String - Compositing
* multiple\_raster\_threads String - Multiple Raster Threads
* native\_gpu\_memory\_buffers String - Native GpuMemoryBuffers
* rasterization String - Rasterization
* video\_decode String - Video Decode
* video\_encode String - Video Encode
* vpx\_decode String - VPx Video Decode
* webgl String - WebGL
* webgl2 String - WebGL2

Possible values:

* disabled\_software - Software only. Hardware acceleration disabled (yellow)
* disabled\_off - Disabled (red)
* disabled\_off\_ok - Disabled (yellow)
* unavailable\_software - Software only, hardware acceleration unavailable (yellow)
* unavailable\_off - Unavailable (red)
* unavailable\_off\_ok - Unavailable (yellow)
* enabled\_readback - Hardware accelerated but at reduced performance (yellow)
* enabled\_force - Hardware accelerated on all pages (green)
* enabled - Hardware accelerated (green)
* enabled\_on - Enabled (green)
* enabled\_force\_on - Force enabled (green)

IOCounters Object

* readOperationCount Number - The number of I/O read operations.
* writeOperationCount Number - The number of I/O write operations.
* otherOperationCount Number - Then number of I/O other operations.
* readTransferCount Number - The number of I/O read transfers.
* writeTransferCount Number - The number of I/O write transfers.
* otherTransferCount Number - Then number of I/O other transfers.

JumpListCategory Object

* type String (optional) - One of the following:
  + tasks - Items in this category will be placed into the standard Tasks category. There can be only one such category, and it will always be displayed at the bottom of the Jump List.
  + frequent - Displays a list of files frequently opened by the app, the name of the category and its items are set by Windows.
  + recent - Displays a list of files recently opened by the app, the name of the category and its items are set by Windows. Items may be added to this category indirectly using app.addRecentDocument(path).
  + custom - Displays tasks or file links, name must be set by the app.
* name String (optional) - Must be set if type is custom, otherwise it should be omitted.
* items JumpListItem[] (optional) - Array of [JumpListItem](https://electron.atom.io/docs/api/structures/jump-list-item) objects if type is tasks or custom, otherwise it should be omitted.

**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.

JumpListItem Object

* type String (optional) - One of the following:
  + task - A task will launch an app with specific arguments.
  + separator - Can be used to separate items in the standard Tasks category.
  + file - A file link will open a file using the app that created the Jump List, for this to work the app must be registered as a handler for the file type (though it doesn’t have to be the default handler).
* path String (optional) - Path of the file to open, should only be set if type is file.
* program String (optional) - Path of the program to execute, usually you should specify process.execPathwhich opens the current program. Should only be set if type is task.
* args String (optional) - The command line arguments when program is executed. Should only be set if typeis task.
* title String (optional) - The text to be displayed for the item in the Jump List. Should only be set if type is task.
* description String (optional) - Description of the task (displayed in a tooltip). Should only be set if type is task.
* iconPath String (optional) - The absolute path to an icon to be displayed in a Jump List, which can be an arbitrary resource file that contains an icon (e.g. .ico, .exe, .dll). You can usually specify process.execPath to show the program icon.
* iconIndex Number (optional) - The index of the icon in the resource file. If a resource file contains multiple icons this value can be used to specify the zero-based index of the icon that should be displayed for this task. If a resource file contains only one icon, this property should be set to zero.

MemoryInfo Object

* pid Integer - Process id of the process.
* workingSetSize Integer - The amount of memory currently pinned to actual physical RAM.
* peakWorkingSetSize Integer - The maximum amount of memory that has ever been pinned to actual physical RAM. On macOS its value will always be 0.
* privateBytes Integer - The amount of memory not shared by other processes, such as JS heap or HTML content.
* sharedBytes Integer - The amount of memory shared between processes, typically memory consumed by the Electron code itself

Note that all statistics are reported in Kilobytes.

MemoryUsageDetails Object

* count Number
* size Number
* liveSize Number

MimeTypedBuffer Object

* mimeType String - The mimeType of the Buffer that you are sending
* data Buffer - The actual Buffer content

NotificationAction Object

* type String - The type of action, can be button.
* text String - (optional) The label for the given action.

Platform / Action Support

| **Action Type** | **Platform Support** | **Usage of text** | **Default text** | **Limitations** |
| --- | --- | --- | --- | --- |
| button | macOS | Used as the label for the button | “Show” | Maximum of one button, if multiple are provided only the last is used. This action is also incomptible with hasReply and will be ignored if hasReply is true. |

Button support on macOS

In order for extra notification buttons to work on macOS your app must meet the following criteria.

* App is signed
* App has it’s NSUserNotificationAlertStyle set to alert in the info.plist.

If either of these requirements are not met the button simply won’t appear.

Point Object

* x Number
* y Number

PrinterInfo Object

* name String
* description String
* status Number
* isDefault Boolean

Example

Below is an example of some of the additional options that may be set which may be different on each platform.

{

name: 'Zebra\_LP2844',

description: 'Zebra LP2844',

status: 3,

isDefault: false,

options: {

copies: '1',

'device-uri': 'usb://Zebra/LP2844?location=14200000',

finishings: '3',

'job-cancel-after': '10800',

'job-hold-until': 'no-hold',

'job-priority': '50',

'job-sheets': 'none,none',

'marker-change-time': '0',

'number-up': '1',

'printer-commands': 'none',

'printer-info': 'Zebra LP2844',

'printer-is-accepting-jobs': 'true',

'printer-is-shared': 'true',

'printer-location': '',

'printer-make-and-model': 'Zebra EPL2 Label Printer',

'printer-state': '3',

'printer-state-change-time': '1484872644',

'printer-state-reasons': 'offline-report',

'printer-type': '36932',

'printer-uri-supported': 'ipp://localhost/printers/Zebra\_LP2844',

system\_driverinfo: 'Z'

}

}

ProcessMemoryInfo Object

* pid Integer - Process id of the process.
* memory [MemoryInfo](https://electron.atom.io/docs/api/structures/memory-info) - Memory information of the process.

ProcessMetric Object

* pid Integer - Process id of the process.
* type String - Process type (Browser or Tab or GPU etc).
* memory [MemoryInfo](https://electron.atom.io/docs/api/structures/memory-info) - Memory information for the process.
* cpu [CPUUsage](https://electron.atom.io/docs/api/structures/cpu-usage) - CPU usage of the process.

Rectangle Object

* x Number - The x coordinate of the origin of the rectangle (must be an integer)
* y Number - The y coordinate of the origin of the rectangle (must be an integer)
* width Number - The width of the rectangle (must be an integer)
* height Number - The height of the rectangle (must be an integer)

RemoveClientCertificate Object

* type String - clientCertificate.
* origin String - Origin of the server whose associated client certificate must be removed from the cache.

RemovePassword Object

* type String - password.
* origin String (optional) - When provided, the authentication info related to the origin will only be removed otherwise the entire cache will be cleared.
* scheme String (optional) - Scheme of the authentication. Can be basic, digest, ntlm, negotiate. Must be provided if removing by origin.
* realm String (optional) - Realm of the authentication. Must be provided if removing by origin.
* username String (optional) - Credentials of the authentication. Must be provided if removing by origin.
* password String (optional) - Credentials of the authentication. Must be provided if removing by origin.

ScrubberItem Object

* label String - (optional) The text to appear in this item
* icon NativeImage - (optional) The image to appear in this item

SegmentedControlSegment Object

* label String - (optional) The text to appear in this segment
* icon NativeImage - (optional) The image to appear in this segment
* enabled Boolean - (optional) Whether this segment is selectable. Default: true

ShortcutDetails Object

* target String - The target to launch from this shortcut.
* cwd String (optional) - The working directory. Default is empty.
* args String (optional) - The arguments to be applied to target when launching from this shortcut. Default is empty.
* description String (optional) - The description of the shortcut. Default is empty.
* icon String (optional) - The path to the icon, can be a DLL or EXE. icon and iconIndex have to be set together. Default is empty, which uses the target’s icon.
* iconIndex Number (optional) - The resource ID of icon when icon is a DLL or EXE. Default is 0.
* appUserModelId String (optional) - The Application User Model ID. Default is empty.

Size Object

* width Number
* height Number

Task Object

* program String - Path of the program to execute, usually you should specify process.execPath which opens the current program.
* arguments String - The command line arguments when program is executed.
* title String - The string to be displayed in a JumpList.
* description String - Description of this task.
* iconPath String - The absolute path to an icon to be displayed in a JumpList, which can be an arbitrary resource file that contains an icon. You can usually specify process.execPath to show the icon of the program.
* iconIndex Number - The icon index in the icon file. If an icon file consists of two or more icons, set this value to identify the icon. If an icon file consists of one icon, this value is 0.

ThumbarButton Object

* icon [NativeImage](https://electron.atom.io/docs/api/native-image) - The icon showing in thumbnail toolbar.
* click Function
* tooltip String (optional) - The text of the button’s tooltip.
* flags String[] (optional) - Control specific states and behaviors of the button. By default, it is ['enabled'].

The flags is an array that can include following Strings:

* enabled - The button is active and available to the user.
* disabled - The button is disabled. It is present, but has a visual state indicating it will not respond to user action.
* dismissonclick - When the button is clicked, the thumbnail window closes immediately.
* nobackground - Do not draw a button border, use only the image.
* hidden - The button is not shown to the user.
* noninteractive - The button is enabled but not interactive; no pressed button state is drawn. This value is intended for instances where the button is used in a notification.

UploadBlob Object

* type String - blob.
* blobUUID String - UUID of blob data to upload.

UploadData Object

* bytes Buffer - Content being sent.
* file String - Path of file being uploaded.
* blobUUID String - UUID of blob data. Use [ses.getBlobData](https://electron.atom.io/docs/api/session#sesgetblobdataidentifier-callback) method to retrieve the data.

UploadFileSystem Object

* type String - fileSystem.
* filsSystemURL String - FileSystem url to read data for upload.
* offset Integer - Defaults to 0.
* length Integer - Number of bytes to read from offset. Defaults to 0.
* modificationTime Double - Last Modification time in number of seconds sine the UNIX epoch.

UploadFile Object

* type String - file.
* filePath String - Path of file to be uploaded.
* offset Integer - Defaults to 0.
* length Integer - Number of bytes to read from offset. Defaults to 0.
* modificationTime Double - Last Modification time in number of seconds sine the UNIX epoch.

UploadRawData Object

* type String - rawData.
* bytes Buffer - Data to be uploaded.

Synopsis

How to use Node.js and Electron APIs.

All of [Node.js’s built-in modules](https://nodejs.org/api/) are available in Electron and third-party node modules also fully supported as well (including the [native modules](https://electron.atom.io/docs/tutorial/using-native-node-modules)).

Electron also provides some extra built-in modules for developing native desktop applications. Some modules are only available in the main process, some are only available in the renderer process (web page), and some can be used in both processes.

The basic rule is: if a module is [GUI](https://en.wikipedia.org/wiki/Graphical_user_interface) or low-level system related, then it should be only available in the main process. You need to be familiar with the concept of [main process vs. renderer process](https://electron.atom.io/docs/tutorial/quick-start#main-process) scripts to be able to use those modules.

The main process script is just like a normal Node.js script:

**const** {app, BrowserWindow} = require('electron')

**let** win = null

app.on('ready', () => {

win = **new** BrowserWindow({width: 800, height: 600})

win.loadURL('https://github.com')

})

The renderer process is no different than a normal web page, except for the extra ability to use node modules:

***<!DOCTYPE html>***

<html>

<body>

<script>

**const** {app} = require('electron').remote

console.log(app.getVersion())

</script>

</body>

</html>

To run your app, read [Run your app](https://electron.atom.io/docs/tutorial/quick-start#run-your-app).

Destructuring assignment

As of 0.37, you can use [destructuring assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment) to make it easier to use built-in modules.

**const** {app, BrowserWindow} = require('electron')

**let** win

app.on('ready', () => {

win = **new** BrowserWindow()

win.loadURL('https://github.com')

})

If you need the entire electron module, you can require it and then using destructuring to access the individual modules from electron.

**const** electron = require('electron')

**const** {app, BrowserWindow} = electron

**let** win

app.on('ready', () => {

win = **new** BrowserWindow()

win.loadURL('https://github.com')

})

This is equivalent to the following code:

**const** electron = require('electron')

**const** app = electron.app

**const** BrowserWindow = electron.BrowserWindow

**let** win

app.on('ready', () => {

win = **new** BrowserWindow()

win.loadURL('https://github.com')

})

systemPreferences

Get system preferences.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

**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

Returns:

* event Event
* invertedColorScheme Boolean - true if an inverted color scheme, such as a high contrast theme, is being used, false otherwise.

Methods

systemPreferences.isDarkMode() macOS

Returns Boolean - Whether the system is in Dark Mode.

systemPreferences.isSwipeTrackingFromScrollEventsEnabled() macOS

Returns Boolean - Whether the Swipe between pages setting is on.

systemPreferences.postNotification(event, userInfo) macOS

* event String
* userInfo Object

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 Object

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 Object

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 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.unsubscribeNotification(id) macOS

* id Integer

Removes the subscriber with id.

systemPreferences.subscribeLocalNotification(event, callback) macOS

* event String
* callback Function
  + event String
  + userInfo Object

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

systemPreferences.unsubscribeLocalNotification(id) macOS

* id Integer

Same as unsubscribeNotification, but removes the subscriber from NSNotificationCenter.

systemPreferences.getUserDefault(key, type) macOS

* key String
* type String - Can be string, boolean, integer, float, double, url, array, dictionary

Returns any - The value of key in system preferences.

This API uses NSUserDefaults on macOS. Some popular key and types are:

* AppleInterfaceStyle: string
* AppleAquaColorVariant: integer
* AppleHighlightColor: string
* AppleShowScrollBars: string
* NSNavRecentPlaces: array
* NSPreferredWebServices: dictionary
* NSUserDictionaryReplacementItems: array

systemPreferences.setUserDefault(key, type, value) macOS

* key String
* type String - See [getUserDefault][#systempreferencesgetuserdefaultkey-type-macos]
* value String

Set the value of key in system preferences.

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

This API uses NSUserDefaults on macOS. Some popular key and types are:

* ApplePressAndHoldEnabled: boolean

systemPreferences.isAeroGlassEnabled() Windows

Returns Boolean - true if [DWM composition](https://msdn.microsoft.com/en-us/library/windows/desktop/aa969540.aspx) (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')

**let** 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.*

**let** 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

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"*

systemPreferences.getColor(color) Windows

* color String - One of the following values:
  + 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.

Returns String - The system color setting in RGB hexadecimal form (#ABCDEF). See the [Windows docs](https://msdn.microsoft.com/en-us/library/windows/desktop/ms724371(v=vs.85).aspx) for more details.

systemPreferences.isInvertedColorScheme() Windows

Returns Boolean - true if an inverted color scheme, such as a high contrast theme, is active, false otherwise.

Class: TouchBarButton

Create a button in the touch bar for native macOS applications

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBarButton(options) Experimental

* options Object
  + label String (optional) - Button text.
  + backgroundColor String (optional) - Button background color in hex format, i.e #ABCDEF.
  + icon [NativeImage](https://electron.atom.io/docs/api/native-image) (optional) - Button icon.
  + iconPosition String - Can be left, right or overlay.
  + click Function (optional) - Function to call when the button is clicked.

Instance Properties

The following properties are available on instances of TouchBarButton:

touchBarButton.label

A String representing the button’s current text. Changing this value immediately updates the button in the touch bar.

touchBarButton.backgroundColor

A String hex code representing the button’s current background color. Changing this value immediately updates the button in the touch bar.

touchBarButton.icon

A NativeImage representing the button’s current icon. Changing this value immediately updates the button in the touch bar.

Class: TouchBarColorPicker

Create a color picker in the touch bar for native macOS applications

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBarColorPicker(options) Experimental

* options Object
  + availableColors String[] (optional) - Array of hex color strings to appear as possible colors to select.
  + selectedColor String (optional) - The selected hex color in the picker, i.e #ABCDEF.
  + change Function (optional) - Function to call when a color is selected.
    - color String - The color that the user selected from the picker

Instance Properties

The following properties are available on instances of TouchBarColorPicker:

touchBarColorPicker.availableColors

A String[] array representing the color picker’s available colors to select. Changing this value immediately updates the color picker in the touch bar.

touchBarColorPicker.selectedColor

A String hex code representing the color picker’s currently selected color. Changing this value immediately updates the color picker in the touch bar.

Class: TouchBarGroup

Create a group in the touch bar for native macOS applications

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBarGroup(options) Experimental

* options Object
  + items [TouchBar](https://electron.atom.io/docs/api/touch-bar) - Items to display as a group.

Class: TouchBarLabel

Create a label in the touch bar for native macOS applications

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBarLabel(options) Experimental

* options Object
  + label String (optional) - Text to display.
  + textColor String (optional) - Hex color of text, i.e #ABCDEF.

Instance Properties

The following properties are available on instances of TouchBarLabel:

touchBarLabel.label

A String representing the label’s current text. Changing this value immediately updates the label in the touch bar.

touchBarLabel.textColor

A String hex code representing the label’s current text color. Changing this value immediately updates the label in the touch bar.

Class: TouchBarPopover

Create a popover in the touch bar for native macOS applications

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBarPopover(options) Experimental

* options Object
  + label String (optional) - Popover button text.
  + icon [NativeImage](https://electron.atom.io/docs/api/native-image) (optional) - Popover button icon.
  + items [TouchBar](https://electron.atom.io/docs/api/touch-bar) (optional) - Items to display in the popover.
  + showCloseButton Boolean (optional) - true to display a close button on the left of the popover, false to not show it. Default is true.

Instance Properties

The following properties are available on instances of TouchBarPopover:

touchBarPopover.label

A String representing the popover’s current button text. Changing this value immediately updates the popover in the touch bar.

touchBarPopover.icon

A NativeImage representing the popover’s current button icon. Changing this value immediately updates the popover in the touch bar.

Class: TouchBarScrubber

Create a scrubber (a scrollable selector)

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBarScrubber(options) Experimental

* options Object
  + items [ScrubberItem[]](https://electron.atom.io/docs/api/structures/scrubber-item) - An array of items to place in this scrubber
  + select Function - Called when the user taps an item that was not the last tapped item
    - selectedIndex Integer - The index of the item the user selected
  + highlight Function - Called when the user taps any item
    - highlightedIndex Integer - The index of the item the user touched
  + selectedStyle String - Selected item style. Defaults to null.
  + overlayStyle String - Selected overlay item style. Defaults to null.
  + showArrowButtons Boolean - Defaults to false.
  + mode String - Defaults to free.
  + continuous Boolean - Defaults to true.

Instance Properties

The following properties are available on instances of TouchBarScrubber:

touchBarScrubber.items

A ScrubberItem[] array representing the items in this scrubber. Updating this value immediately updates the control in the touch bar. Updating deep properties inside this array **does not update the touch bar**.

touchBarScrubber.selectedStyle

A String representing the style that selected items in the scrubber should have. Updating this value immediately updates the control in the touch bar. Possible values:

* background - Maps to [NSScrubberSelectionStyle roundedBackgroundStyle]
* outline - Maps to [NSScrubberSelectionStyle outlineOverlayStyle]
* null - Actually null, not a string, removes all styles

touchBarScrubber.overlayStyle

A String representing the style that selected items in the scrubber should have. This style is overlayed on top of the scrubber item instead of being placed behind it. Updating this value immediately updates the control in the touch bar. Possible values:

* background - Maps to [NSScrubberSelectionStyle roundedBackgroundStyle]
* outline - Maps to [NSScrubberSelectionStyle outlineOverlayStyle]
* null - Actually null, not a string, removes all styles

touchBarScrubber.showArrowButtons

A Boolean representing whether to show the left / right selection arrows in this scrubber. Updating this value immediately updates the control in the touch bar.

touchBarScrubber.mode

A String representing the mode of this scrubber. Updating this value immediately updates the control in the touch bar. Possible values:

* fixed - Maps to NSScrubberModeFixed
* free - Maps to NSScrubberModeFree

touchBarScrubber.continuous

A Boolean representing whether this scrubber is continuous or not. Updating this value immediately updates the control in the touch bar.

Class: TouchBarSegmentedControl

Create a segmented control (a button group) where one button has a selected state

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBarSegmentedControl(options) Experimental

* options Object
  + segmentStyle String - (optional) Style of the segments:
    - automatic - Default. The appearance of the segmented control is automatically determined based on the type of window in which the control is displayed and the position within the window.
    - rounded - The control is displayed using the rounded style.
    - textured-rounded - The control is displayed using the textured rounded style.
    - round-rect - The control is displayed using the round rect style.
    - textured-square - The control is displayed using the textured square style.
    - capsule - The control is displayed using the capsule style
    - small-square - The control is displayed using the small square style.
    - separated - The segments in the control are displayed very close to each other but not touching.
  + mode String - (optional) The selection mode of the control:
    - single - Default. One item selected at a time, selecting one deselects the previously selected item.
    - multiple - Multiple items can be selected at a time.
    - buttons - Make the segments act as buttons, each segment can be pressed and released but never marked as active.
  + segments [SegmentedControlSegment[]](https://electron.atom.io/docs/api/structures/segmented-control-segment) - An array of segments to place in this control.
  + selectedIndex Integer (optional) - The index of the currently selected segment, will update automatically with user interaction. When the mode is multiple it will be the last selected item.
  + change Function - Called when the user selects a new segment
    - selectedIndex Integer - The index of the segment the user selected.
    - isSelected Boolean - Whether as a result of user selection the segment is selected or not.

Instance Properties

The following properties are available on instances of TouchBarSegmentedControl:

touchBarSegmentedControl.segmentStyle

A String representing the controls current segment style. Updating this value immediately updates the control in the touch bar.

touchBarSegmentedControl.segments

A SegmentedControlSegment[] array representing the segments in this control. Updating this value immediately updates the control in the touch bar. Updating deep properties inside this array **does not update the touch bar**.

touchBarSegmentedControl.selectedIndex

An Integer representing the currently selected segment. Changing this value immediately updates the control in the touch bar. User interaction with the touch bar will update this value automatically.

Class: TouchBarSlider

Create a slider in the touch bar for native macOS applications

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBarSlider(options) Experimental

* options Object
  + label String (optional) - Label text.
  + value Integer (optional) - Selected value.
  + minValue Integer (optional) - Minimum value.
  + maxValue Integer (optional) - Maximum value.
  + change Function (optional) - Function to call when the slider is changed.
    - newValue Number - The value that the user selected on the Slider

Instance Properties

The following properties are available on instances of TouchBarSlider:

touchBarSlider.label

A String representing the slider’s current text. Changing this value immediately updates the slider in the touch bar.

touchBarSlider.value

A Number representing the slider’s current value. Changing this value immediately updates the slider in the touch bar.

touchBarSlider.minValue

A Number representing the slider’s current minimum value. Changing this value immediately updates the slider in the touch bar.

touchBarSlider.maxValue

A Number representing the slider’s current maximum value. Changing this value immediately updates the slider in the touch bar.

Class: TouchBarSpacer

Create a spacer between two items in the touch bar for native macOS applications

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBarSpacer(options) Experimental

* options Object
  + size String (optional) - Size of spacer, possible values are:
    - small - Small space between items.
    - large - Large space between items.
    - flexible - Take up all available space.

Class: TouchBar

Create TouchBar layouts for native macOS applications

Process: [Main](https://electron.atom.io/docs/tutorial/quick-start#main-process)

new TouchBar(options) Experimental

* options Object
  + items ([TouchBarButton](https://electron.atom.io/docs/api/touch-bar-button) | [TouchBarColorPicker](https://electron.atom.io/docs/api/touch-bar-color-picker) | [TouchBarGroup](https://electron.atom.io/docs/api/touch-bar-group) | [TouchBarLabel](https://electron.atom.io/docs/api/touch-bar-label) | [TouchBarPopover](https://electron.atom.io/docs/api/touch-bar-popover) | [TouchBarScrubber](https://electron.atom.io/docs/api/touch-bar-scrubber) | [TouchBarSegmentedControl](https://electron.atom.io/docs/api/touch-bar-segmented-control) | [TouchBarSlider](https://electron.atom.io/docs/api/touch-bar-slider) | [TouchBarSpacer](https://electron.atom.io/docs/api/touch-bar-spacer))[]
  + escapeItem ([TouchBarButton](https://electron.atom.io/docs/api/touch-bar-button) | [TouchBarColorPicker](https://electron.atom.io/docs/api/touch-bar-color-picker) | [TouchBarGroup](https://electron.atom.io/docs/api/touch-bar-group) | [TouchBarLabel](https://electron.atom.io/docs/api/touch-bar-label) | [TouchBarPopover](https://electron.atom.io/docs/api/touch-bar-popover) | [TouchBarScrubber](https://electron.atom.io/docs/api/touch-bar-scrubber) | [TouchBarSegmentedControl](https://electron.atom.io/docs/api/touch-bar-segmented-control) | [TouchBarSlider](https://electron.atom.io/docs/api/touch-bar-slider) | [TouchBarSpacer](https://electron.atom.io/docs/api/touch-bar-spacer)) (optional)

Creates a new touch bar with the specified items. Use BrowserWindow.setTouchBar to add the TouchBar to a window.

**Note:** The TouchBar API is currently experimental and may change or be removed in future Electron releases.

**Tip:** If you don’t have a MacBook with Touch Bar, you can use [Touch Bar Simulator](https://github.com/sindresorhus/touch-bar-simulator) to test Touch Bar usage in your app.

Instance Properties

The following properties are available on instances of TouchBar:

touchBar.escapeItem

The TouchBarButton that will replace the “esc” button on the touch bar when set. Setting to null restores the default “esc” button. Changing this value immediately updates the escape item in the touch bar.

Examples

Below is an example of a simple slot machine touch bar game with a button and some labels.

**const** {app, BrowserWindow, TouchBar} = require('electron')

**const** {TouchBarLabel, TouchBarButton, TouchBarSpacer} = TouchBar

**let** spinning = false

*// Reel labels*

**const** reel1 = **new** TouchBarLabel()

**const** reel2 = **new** TouchBarLabel()

**const** reel3 = **new** TouchBarLabel()

*// Spin result label*

**const** result = **new** TouchBarLabel()

*// Spin button*

**const** spin = **new** TouchBarButton({

label: '🎰 Spin',

backgroundColor: '#7851A9',

click: () => {

*// Ignore clicks if already spinning*

**if** (spinning) {

**return**

}

spinning = true

result.label = ''

**let** timeout = 10

**const** spinLength = 4 \* 1000 *// 4 seconds*

**const** startTime = Date.now()

**const** spinReels = () => {

updateReels()

**if** ((Date.now() - startTime) >= spinLength) {

finishSpin()

} **else** {

*// Slow down a bit on each spin*

timeout \*= 1.1

setTimeout(spinReels, timeout)

}

}

spinReels()

}

})

**const** getRandomValue = () => {

**const** values = ['🍒', '💎', '7️⃣', '🍊', '🔔', '⭐', '🍇', '🍀']

**return** values[Math.floor(Math.random() \* values.length)]

}

**const** updateReels = () => {

reel1.label = getRandomValue()

reel2.label = getRandomValue()

reel3.label = getRandomValue()

}

**const** finishSpin = () => {

**const** uniqueValues = **new** Set([reel1.label, reel2.label, reel3.label]).size

**if** (uniqueValues === 1) {

*// All 3 values are the same*

result.label = '💰 Jackpot!'

result.textColor = '#FDFF00'

} **else** **if** (uniqueValues === 2) {

*// 2 values are the same*

result.label = '😍 Winner!'

result.textColor = '#FDFF00'

} **else** {

*// No values are the same*

result.label = '🙁 Spin Again'

result.textColor = null

}

spinning = false

}

**const** touchBar = **new** TouchBar([

spin,

**new** TouchBarSpacer({size: 'large'}),

reel1,

**new** TouchBarSpacer({size: 'small'}),

reel2,

**new** TouchBarSpacer({size: 'small'}),

reel3,

**new** TouchBarSpacer({size: 'large'}),

result

])

**let** window

app.once('ready', () => {

window = **new** BrowserWindow({

frame: false,

titleBarStyle: 'hiddenInset',

width: 200,

height: 200,

backgroundColor: '#000'

})

window.loadURL('about:blank')

window.setTouchBar(touchBar)

})

Running the above example

To run the example above, you’ll need to (assuming you’ve got a terminal open in the dirtectory you want to run the example):

1. Save the above file to your computer as touchbar.js
2. Install Electron via npm install electron
3. Run the example inside Electron: ./node\_modules/.bin/electron touchbar.js

You should then see a new Electron window and the app running in your touch bar (or touch bar emulator).

Class: Tray

Add icons and context menus to the system’s notification area.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

Tray is an [EventEmitter](https://nodejs.org/api/events.html#events_class_eventemitter).

**const** {app, Menu, Tray} = require('electron')

**let** tray = null

app.on('ready', () => {

tray = **new** Tray('/path/to/my/icon')

**const** contextMenu = Menu.buildFromTemplate([

{label: 'Item1', type: 'radio'},

{label: 'Item2', type: 'radio'},

{label: 'Item3', type: 'radio', checked: true},

{label: 'Item4', type: 'radio'}

])

tray.setToolTip('This is my application.')

tray.setContextMenu(contextMenu)

})

**Platform limitations:**

* On Linux the app indicator will be used if it is supported, otherwise GtkStatusIcon will be used instead.
* On Linux distributions that only have app indicator support, you have to install libappindicator1 to make the tray icon work.
* App indicator will only be shown when it has a context menu.
* When app indicator is used on Linux, the click event is ignored.
* On Linux in order for changes made to individual MenuItems to take effect, you have to call setContextMenuagain. For example:

**const** {app, Menu, Tray} = require('electron')

**let** appIcon = null

app.on('ready', () => {

appIcon = **new** Tray('/path/to/my/icon')

**const** contextMenu = Menu.buildFromTemplate([

{label: 'Item1', type: 'radio'},

{label: 'Item2', type: 'radio'}

])

*// Make a change to the context menu*

contextMenu.items[1].checked = false

*// Call this again for Linux because we modified the context menu*

appIcon.setContextMenu(contextMenu)

})

* On Windows it is recommended to use ICO icons to get best visual effects.

If you want to keep exact same behaviors on all platforms, you should not rely on the click event and always attach a context menu to the tray icon.

new Tray(image)

* image ([NativeImage](https://electron.atom.io/docs/api/native-image) | String)

Creates a new tray icon associated with the image.

Instance Events

The Tray module emits the following events:

Event: ‘click’

* event Event
  + altKey Boolean
  + shiftKey Boolean
  + ctrlKey Boolean
  + metaKey Boolean
* bounds [Rectangle](https://electron.atom.io/docs/api/structures/rectangle) - The bounds of tray icon

Emitted when the tray icon is clicked.

Event: ‘right-click’ macOS Windows

* event Event
  + altKey Boolean
  + shiftKey Boolean
  + ctrlKey Boolean
  + metaKey Boolean
* bounds [Rectangle](https://electron.atom.io/docs/api/structures/rectangle) - The bounds of tray icon

Emitted when the tray icon is right clicked.

Event: ‘double-click’ macOS Windows

* event Event
  + altKey Boolean
  + shiftKey Boolean
  + ctrlKey Boolean
  + metaKey Boolean
* bounds [Rectangle](https://electron.atom.io/docs/api/structures/rectangle) - The bounds of tray icon

Emitted when the tray icon is double clicked.

Event: ‘balloon-show’ Windows

Emitted when the tray balloon shows.

Event: ‘balloon-click’ Windows

Emitted when the tray balloon is clicked.

Event: ‘balloon-closed’ Windows

Emitted when the tray balloon is closed because of timeout or user manually closes it.

Event: ‘drop’ macOS

Emitted when any dragged items are dropped on the tray icon.

Event: ‘drop-files’ macOS

* event Event
* files String[] - The paths of the dropped files.

Emitted when dragged files are dropped in the tray icon.

Event: ‘drop-text’ macOS

* event Event
* text String - the dropped text string

Emitted when dragged text is dropped in the tray icon.

Event: ‘drag-enter’ macOS

Emitted when a drag operation enters the tray icon.

Event: ‘drag-leave’ macOS

Emitted when a drag operation exits the tray icon.

Event: ‘drag-end’ macOS

Emitted when a drag operation ends on the tray or ends at another location.

Event: ‘mouse-enter’ macOS

* event Event
  + altKey Boolean
  + shiftKey Boolean
  + ctrlKey Boolean
  + metaKey Boolean
* position [Point](https://electron.atom.io/docs/api/structures/point) - The position of the event

Emitted when the mouse enters the tray icon.

Event: ‘mouse-leave’ macOS

* event Event
  + altKey Boolean
  + shiftKey Boolean
  + ctrlKey Boolean
  + metaKey Boolean
* position [Point](https://electron.atom.io/docs/api/structures/point) - The position of the event

Emitted when the mouse exits the tray icon.

Instance Methods

The Tray class has the following methods:

tray.destroy()

Destroys the tray icon immediately.

tray.setImage(image)

* image ([NativeImage](https://electron.atom.io/docs/api/native-image) | String)

Sets the image associated with this tray icon.

tray.setPressedImage(image) macOS

* image [NativeImage](https://electron.atom.io/docs/api/native-image)

Sets the image associated with this tray icon when pressed on macOS.

tray.setToolTip(toolTip)

* toolTip String

Sets the hover text for this tray icon.

tray.setTitle(title) macOS

* title String

Sets the title displayed aside of the tray icon in the status bar.

tray.setHighlightMode(mode) macOS

* mode String - Highlight mode with one of the following values:
  + selection - Highlight the tray icon when it is clicked and also when its context menu is open. This is the default.
  + always - Always highlight the tray icon.
  + never - Never highlight the tray icon.

Sets when the tray’s icon background becomes highlighted (in blue).

**Note:** You can use highlightMode with a [BrowserWindow](https://electron.atom.io/docs/api/browser-window) by toggling between 'never' and 'always' modes when the window visibility changes.

**const** {BrowserWindow, Tray} = require('electron')

**const** win = **new** BrowserWindow({width: 800, height: 600})

**const** tray = **new** Tray('/path/to/my/icon')

tray.on('click', () => {

win.isVisible() ? win.hide() : win.show()

})

win.on('show', () => {

tray.setHighlightMode('always')

})

win.on('hide', () => {

tray.setHighlightMode('never')

})

tray.displayBalloon(options) Windows

* options Object
  + icon ([NativeImage](https://electron.atom.io/docs/api/native-image) | String) - (optional)
  + title String - (optional)
  + content String - (optional)

Displays a tray balloon.

tray.popUpContextMenu([menu, position]) macOS Windows

* menu Menu (optional)
* position [Point](https://electron.atom.io/docs/api/structures/point) (optional) - The pop up position.

Pops up the context menu of the tray icon. When menu is passed, the menu will be shown instead of the tray icon’s context menu.

The position is only available on Windows, and it is (0, 0) by default.

tray.setContextMenu(menu)

* menu Menu

Sets the context menu for this icon.

tray.getBounds() macOS Windows

Returns [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)

The bounds of this tray icon as Object.

tray.isDestroyed()

Returns Boolean - Whether the tray icon is destroyed.

webContents

Render and control web pages.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

webContents is an [EventEmitter](https://nodejs.org/api/events.html#events_class_eventemitter). It is responsible for rendering and controlling a web page and is a property of the [BrowserWindow](https://electron.atom.io/docs/api/browser-window) object. An example of accessing the webContents object:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({width: 800, height: 1500})

win.loadURL('http://github.com')

**let** contents = win.webContents

console.log(contents)

Methods

These methods can be accessed from the webContents module:

**const** {webContents} = require('electron')

console.log(webContents)

webContents.getAllWebContents()

Returns WebContents[] - An array of all WebContents instances. This will contain web contents for all windows, webviews, opened devtools, and devtools extension background pages.

webContents.getFocusedWebContents()

Returns WebContents - The web contents that is focused in this application, otherwise returns null.

webContents.fromId(id)

* id Integer

Returns WebContents - A WebContents instance with the given ID.

Class: WebContents

Render and control the contents of a BrowserWindow instance.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

Instance Events

Event: ‘did-finish-load’

Emitted when the navigation is done, i.e. the spinner of the tab has stopped spinning, and the onload event was dispatched.

Event: ‘did-fail-load’

Returns:

* event Event
* errorCode Integer
* errorDescription String
* validatedURL String
* isMainFrame Boolean

This event is like did-finish-load but emitted when the load failed or was cancelled, e.g. window.stop() is invoked. The full list of error codes and their meaning is available [here](https://code.google.com/p/chromium/codesearch#chromium/src/net/base/net_error_list.h).

Event: ‘did-frame-finish-load’

Returns:

* event Event
* isMainFrame Boolean

Emitted when a frame has done navigation.

Event: ‘did-start-loading’

Corresponds to the points in time when the spinner of the tab started spinning.

Event: ‘did-stop-loading’

Corresponds to the points in time when the spinner of the tab stopped spinning.

Event: ‘did-get-response-details’

Returns:

* event Event
* status Boolean
* newURL String
* originalURL String
* httpResponseCode Integer
* requestMethod String
* referrer String
* headers Object
* resourceType String

Emitted when details regarding a requested resource are available. status indicates the socket connection to download the resource.

Event: ‘did-get-redirect-request’

Returns:

* event Event
* oldURL String
* newURL String
* isMainFrame Boolean
* httpResponseCode Integer
* requestMethod String
* referrer String
* headers Object

Emitted when a redirect is received while requesting a resource.

Event: ‘dom-ready’

Returns:

* event Event

Emitted when the document in the given frame is loaded.

Event: ‘page-favicon-updated’

Returns:

* event Event
* favicons String[] - Array of URLs

Emitted when page receives favicon urls.

Event: ‘new-window’

Returns:

* event Event
* url String
* frameName String
* disposition String - Can be default, foreground-tab, background-tab, new-window, save-to-disk and other.
* options Object - The options which will be used for creating the new BrowserWindow.
* additionalFeatures String[] - The non-standard features (features not handled by Chromium or Electron) given to window.open().

Emitted when the page requests to open a new window for a url. It could be requested by window.open or an external link like <a target='\_blank'>.

By default a new BrowserWindow will be created for the url.

Calling event.preventDefault() will prevent Electron from automatically creating a new BrowserWindow. If you call event.preventDefault() and manually create a new BrowserWindow then you must set event.newGuest to reference the new BrowserWindow instance, failing to do so may result in unexpected behavior. For example:

myBrowserWindow.webContents.on('new-window', (event, url) => {

event.preventDefault()

**const** win = **new** BrowserWindow({show: false})

win.once('ready-to-show', () => win.show())

win.loadURL(url)

event.newGuest = win

})

Event: ‘will-navigate’

Returns:

* event Event
* url String

Emitted when a user or the page wants to start navigation. It can happen when the window.location object is changed or a user clicks a link in the page.

This event will not emit when the navigation is started programmatically with APIs like webContents.loadURL and webContents.back.

It is also not emitted for in-page navigations, such as clicking anchor links or updating the window.location.hash. Use did-navigate-in-page event for this purpose.

Calling event.preventDefault() will prevent the navigation.

Event: ‘did-navigate’

Returns:

* event Event
* url String

Emitted when a navigation is done.

This event is not emitted for in-page navigations, such as clicking anchor links or updating the window.location.hash. Use did-navigate-in-page event for this purpose.

Event: ‘did-navigate-in-page’

Returns:

* event Event
* url String
* isMainFrame Boolean

Emitted when an in-page navigation happened.

When in-page navigation happens, the page URL changes but does not cause navigation outside of the page. Examples of this occurring are when anchor links are clicked or when the DOM hashchange event is triggered.

Event: ‘will-prevent-unload’

Returns:

* event Event

Emitted when a beforeunload event handler is attempting to cancel a page unload.

Calling event.preventDefault() will ignore the beforeunload event handler and allow the page to be unloaded.

**const** {BrowserWindow, dialog} = require('electron')

**const** win = **new** BrowserWindow({width: 800, height: 600})

win.webContents.on('will-prevent-unload', (event) => {

**const** choice = dialog.showMessageBox(win, {

type: 'question',

buttons: ['Leave', 'Stay'],

title: 'Do you want to leave this site?',

message: 'Changes you made may not be saved.',

defaultId: 0,

cancelId: 1

})

**const** leave = (choice === 0)

**if** (leave) {

event.preventDefault()

}

})

Event: ‘crashed’

Returns:

* event Event
* killed Boolean

Emitted when the renderer process crashes or is killed.

Event: ‘plugin-crashed’

Returns:

* event Event
* name String
* version String

Emitted when a plugin process has crashed.

Event: ‘destroyed’

Emitted when webContents is destroyed.

Event: ‘before-input-event’

Returns:

* event Event
* input Object - Input properties
  + type String - Either keyUp or keyDown
  + key String - Equivalent to [KeyboardEvent.key](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent)
  + code String - Equivalent to [KeyboardEvent.code](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent)
  + isAutoRepeat Boolean - Equivalent to [KeyboardEvent.repeat](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent)
  + shift Boolean - Equivalent to [KeyboardEvent.shiftKey](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent)
  + control Boolean - Equivalent to [KeyboardEvent.controlKey](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent)
  + alt Boolean - Equivalent to [KeyboardEvent.altKey](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent)
  + meta Boolean - Equivalent to [KeyboardEvent.metaKey](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent)

Emitted before dispatching the keydown and keyup events in the page. Calling event.preventDefault will prevent the page keydown/keyup events and the menu shortcuts.

To only prevent the menu shortcuts, use [setIgnoreMenuShortcuts](https://electron.atom.io/docs/all/#contentssetignoremenushortcuts):

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({width: 800, height: 600})

win.webContents.on('before-input-event', (event, input) => {

*// For example, only enable application menu keyboard shortcuts when*

*// Ctrl/Cmd are down.*

win.webContents.setIgnoreMenuShortcuts(!input.control && !input.meta)

})

Event: ‘devtools-opened’

Emitted when DevTools is opened.

Event: ‘devtools-closed’

Emitted when DevTools is closed.

Event: ‘devtools-focused’

Emitted when DevTools is focused / opened.

Event: ‘certificate-error’

Returns:

* event Event
* url String
* error String - The error code
* certificate [Certificate](https://electron.atom.io/docs/api/structures/certificate)
* callback Function
  + isTrusted Boolean - Indicates whether the certificate can be considered trusted

Emitted when failed to verify the certificate for url.

The usage is the same with [the certificate-error event of app](https://electron.atom.io/docs/api/app#event-certificate-error).

Event: ‘select-client-certificate’

Returns:

* event Event
* url URL
* certificateList [Certificate[]](https://electron.atom.io/docs/api/structures/certificate)
* callback Function
  + certificate [Certificate](https://electron.atom.io/docs/api/structures/certificate) - Must be a certificate from the given list

Emitted when a client certificate is requested.

The usage is the same with [the select-client-certificate event of app](https://electron.atom.io/docs/api/app#event-select-client-certificate).

Event: ‘login’

Returns:

* event Event
* request Object
  + method String
  + url URL
  + referrer URL
* authInfo Object
  + isProxy Boolean
  + scheme String
  + host String
  + port Integer
  + realm String
* callback Function
  + username String
  + password String

Emitted when webContents wants to do basic auth.

The usage is the same with [the login event of app](https://electron.atom.io/docs/api/app#event-login).

Event: ‘found-in-page’

Returns:

* event Event
* result Object
  + requestId Integer
  + activeMatchOrdinal Integer - Position of the active match.
  + matches Integer - Number of Matches.
  + selectionArea Object - Coordinates of first match region.
  + finalUpdate Boolean

Emitted when a result is available for [webContents.findInPage] request.

Event: ‘media-started-playing’

Emitted when media starts playing.

Event: ‘media-paused’

Emitted when media is paused or done playing.

Event: ‘did-change-theme-color’

Emitted when a page’s theme color changes. This is usually due to encountering a meta tag:

<meta name='theme-color' content='#ff0000'>

Event: ‘update-target-url’

Returns:

* event Event
* url String

Emitted when mouse moves over a link or the keyboard moves the focus to a link.

Event: ‘cursor-changed’

Returns:

* event Event
* type String
* image NativeImage (optional)
* scale Float (optional) - scaling factor for the custom cursor
* size [Size](https://electron.atom.io/docs/api/structures/size) (optional) - the size of the image
* hotspot [Point](https://electron.atom.io/docs/api/structures/point) (optional) - coordinates of the custom cursor’s hotspot

Emitted when the cursor’s type changes. The type parameter can be default, crosshair, pointer, text, wait, help, e-resize, n-resize, ne-resize, nw-resize, s-resize, se-resize, sw-resize, w-resize, ns-resize, ew-resize, nesw-resize, nwse-resize, col-resize, row-resize, m-panning, e-panning, n-panning, ne-panning, nw-panning, s-panning, se-panning, sw-panning, w-panning, move, vertical-text, cell, context-menu, alias, progress, nodrop, copy, none, not-allowed, zoom-in, zoom-out, grab, grabbing, custom.

If the type parameter is custom, the image parameter will hold the custom cursor image in a NativeImage, and scale, size and hotspot will hold additional information about the custom cursor.

Event: ‘context-menu’

Returns:

* event Event
* params Object
  + x Integer - x coordinate
  + y Integer - y coordinate
  + linkURL String - URL of the link that encloses the node the context menu was invoked on.
  + linkText String - Text associated with the link. May be an empty string if the contents of the link are an image.
  + pageURL String - URL of the top level page that the context menu was invoked on.
  + frameURL String - URL of the subframe that the context menu was invoked on.
  + srcURL String - Source URL for the element that the context menu was invoked on. Elements with source URLs are images, audio and video.
  + mediaType String - Type of the node the context menu was invoked on. Can be none, image, audio, video, canvas, file or plugin.
  + hasImageContents Boolean - Whether the context menu was invoked on an image which has non-empty contents.
  + isEditable Boolean - Whether the context is editable.
  + selectionText String - Text of the selection that the context menu was invoked on.
  + titleText String - Title or alt text of the selection that the context was invoked on.
  + misspelledWord String - The misspelled word under the cursor, if any.
  + frameCharset String - The character encoding of the frame on which the menu was invoked.
  + inputFieldType String - If the context menu was invoked on an input field, the type of that field. Possible values are none, plainText, password, other.
  + menuSourceType String - Input source that invoked the context menu. Can be none, mouse, keyboard, touch, touchMenu.
  + mediaFlags Object - The flags for the media element the context menu was invoked on.
    - inError Boolean - Whether the media element has crashed.
    - isPaused Boolean - Whether the media element is paused.
    - isMuted Boolean - Whether the media element is muted.
    - hasAudio Boolean - Whether the media element has audio.
    - isLooping Boolean - Whether the media element is looping.
    - isControlsVisible Boolean - Whether the media element’s controls are visible.
    - canToggleControls Boolean - Whether the media element’s controls are toggleable.
    - canRotate Boolean - Whether the media element can be rotated.
  + editFlags Object - These flags indicate whether the renderer believes it is able to perform the corresponding action.
    - canUndo Boolean - Whether the renderer believes it can undo.
    - canRedo Boolean - Whether the renderer believes it can redo.
    - canCut Boolean - Whether the renderer believes it can cut.
    - canCopy Boolean - Whether the renderer believes it can copy
    - canPaste Boolean - Whether the renderer believes it can paste.
    - canDelete Boolean - Whether the renderer believes it can delete.
    - canSelectAll Boolean - Whether the renderer believes it can select all.

Emitted when there is a new context menu that needs to be handled.

Event: ‘select-bluetooth-device’

Returns:

* event Event
* devices [BluetoothDevice[]](https://electron.atom.io/docs/api/structures/bluetooth-device)
* callback Function
  + deviceId String

Emitted when bluetooth device needs to be selected on call to navigator.bluetooth.requestDevice. To use navigator.bluetooth api webBluetooth should be enabled. If event.preventDefault is not called, first available device will be selected. callback should be called with deviceId to be selected, passing empty string to callback will cancel the request.

**const** {app, webContents} = require('electron')

app.commandLine.appendSwitch('enable-web-bluetooth')

app.on('ready', () => {

webContents.on('select-bluetooth-device', (event, deviceList, callback) => {

event.preventDefault()

**let** result = deviceList.find((device) => {

**return** device.deviceName === 'test'

})

**if** (!result) {

callback('')

} **else** {

callback(result.deviceId)

}

})

})

Event: ‘paint’

Returns:

* event Event
* dirtyRect [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)
* image [NativeImage](https://electron.atom.io/docs/api/native-image) - The image data of the whole frame.

Emitted when a new frame is generated. Only the dirty area is passed in the buffer.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({webPreferences: {offscreen: true}})

win.webContents.on('paint', (event, dirty, image) => {

*// updateBitmap(dirty, image.getBitmap())*

})

win.loadURL('http://github.com')

Event: ‘devtools-reload-page’

Emitted when the devtools window instructs the webContents to reload

Event: ‘will-attach-webview’

Returns:

* event Event
* webPreferences Object - The web preferences that will be used by the guest page. This object can be modified to adjust the preferences for the guest page.
* params Object - The other <webview> parameters such as the src URL. This object can be modified to adjust the parameters of the guest page.

Emitted when a <webview>’s web contents is being attached to this web contents. Calling event.preventDefault() will destroy the guest page.

This event can be used to configure webPreferences for the webContents of a <webview> before it’s loaded, and provides the ability to set settings that can’t be set via <webview> attributes.

**Note:** The specified preload script option will be appear as preloadURL (not preload) in the webPreferencesobject emitted with this event.

Instance Methods

contents.loadURL(url[, options])

* url String
* options Object (optional)
  + httpReferrer String (optional) - A HTTP Referrer url.
  + userAgent String (optional) - A user agent originating the request.
  + extraHeaders String (optional) - Extra headers separated by “\n”
  + postData ([UploadRawData[]](https://electron.atom.io/docs/api/structures/upload-raw-data) | [UploadFile[]](https://electron.atom.io/docs/api/structures/upload-file) | [UploadFileSystem[]](https://electron.atom.io/docs/api/structures/upload-file-system) | [UploadBlob[]](https://electron.atom.io/docs/api/structures/upload-blob)) - (optional)
  + baseURLForDataURL String (optional) - Base url (with trailing path separator) for files to be loaded by the data url. This is needed only if the specified url is a data url and needs to load other files.

Loads the url in the window. The url must contain the protocol prefix, e.g. the http:// or file://. If the load should bypass http cache then use the pragma header to achieve it.

**const** {webContents} = require('electron')

**const** options = {extraHeaders: 'pragma: no-cache\n'}

webContents.loadURL('https://github.com', options)

contents.downloadURL(url)

* url String

Initiates a download of the resource at url without navigating. The will-download event of session will be triggered.

contents.getURL()

Returns String - The URL of the current web page.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({width: 800, height: 600})

win.loadURL('http://github.com')

**let** currentURL = win.webContents.getURL()

console.log(currentURL)

contents.getTitle()

Returns String - The title of the current web page.

contents.isDestroyed()

Returns Boolean - Whether the web page is destroyed.

contents.focus()

Focuses the web page.

contents.isFocused()

Returns Boolean - Whether the web page is focused.

contents.isLoading()

Returns Boolean - Whether web page is still loading resources.

contents.isLoadingMainFrame()

Returns Boolean - Whether the main frame (and not just iframes or frames within it) is still loading.

contents.isWaitingForResponse()

Returns Boolean - Whether the web page is waiting for a first-response from the main resource of the page.

contents.stop()

Stops any pending navigation.

contents.reload()

Reloads the current web page.

contents.reloadIgnoringCache()

Reloads current page and ignores cache.

contents.canGoBack()

Returns Boolean - Whether the browser can go back to previous web page.

contents.canGoForward()

Returns Boolean - Whether the browser can go forward to next web page.

contents.canGoToOffset(offset)

* offset Integer

Returns Boolean - Whether the web page can go to offset.

contents.clearHistory()

Clears the navigation history.

contents.goBack()

Makes the browser go back a web page.

contents.goForward()

Makes the browser go forward a web page.

contents.goToIndex(index)

* index Integer

Navigates browser to the specified absolute web page index.

contents.goToOffset(offset)

* offset Integer

Navigates to the specified offset from the “current entry”.

contents.isCrashed()

Returns Boolean - Whether the renderer process has crashed.

contents.setUserAgent(userAgent)

* userAgent String

Overrides the user agent for this web page.

contents.getUserAgent()

Returns String - The user agent for this web page.

contents.insertCSS(css)

* css String

Injects CSS into the current web page.

contents.executeJavaScript(code[, userGesture, callback])

* code String
* userGesture Boolean (optional) - Default is false.
* callback Function (optional) - Called after script has been executed.
  + result Any

Returns Promise - A promise that resolves with the result of the executed code or is rejected if the result of the code is a rejected promise.

Evaluates code in page.

In the browser window some HTML APIs like requestFullScreen can only be invoked by a gesture from the user. Setting userGesture to true will remove this limitation.

If the result of the executed code is a promise the callback result will be the resolved value of the promise. We recommend that you use the returned Promise to handle code that results in a Promise.

contents.executeJavaScript('fetch("https://jsonplaceholder.typicode.com/users/1").then(resp => resp.json())', true)

.then((result) => {

console.log(result) *// Will be the JSON object from the fetch call*

})

contents.setIgnoreMenuShortcuts(ignore) Experimental

* ignore Boolean

Ignore application menu shortcuts while this web contents is focused.

contents.setAudioMuted(muted)

* muted Boolean

Mute the audio on the current web page.

contents.isAudioMuted()

Returns Boolean - Whether this page has been muted.

contents.setZoomFactor(factor)

* factor Number - Zoom factor.

Changes the zoom factor to the specified factor. Zoom factor is zoom percent divided by 100, so 300% = 3.0.

contents.getZoomFactor(callback)

* callback Function
  + zoomFactor Number

Sends a request to get current zoom factor, the callback will be called with callback(zoomFactor).

contents.setZoomLevel(level)

* level Number - Zoom level

Changes the zoom level to the specified level. The original size is 0 and each increment above or below represents zooming 20% larger or smaller to default limits of 300% and 50% of original size, respectively.

contents.getZoomLevel(callback)

* callback Function
  + zoomLevel Number

Sends a request to get current zoom level, the callback will be called with callback(zoomLevel).

contents.setZoomLevelLimits(minimumLevel, maximumLevel)

* minimumLevel Number
* maximumLevel Number

**Deprecated:** Call setVisualZoomLevelLimits instead to set the visual zoom level limits. This method will be removed in Electron 2.0.

contents.setVisualZoomLevelLimits(minimumLevel, maximumLevel)

* minimumLevel Number
* maximumLevel Number

Sets the maximum and minimum pinch-to-zoom level.

contents.setLayoutZoomLevelLimits(minimumLevel, maximumLevel)

* minimumLevel Number
* maximumLevel Number

Sets the maximum and minimum layout-based (i.e. non-visual) zoom level.

contents.undo()

Executes the editing command undo in web page.

contents.redo()

Executes the editing command redo in web page.

contents.cut()

Executes the editing command cut in web page.

contents.copy()

Executes the editing command copy in web page.

contents.copyImageAt(x, y)

* x Integer
* y Integer

Copy the image at the given position to the clipboard.

contents.paste()

Executes the editing command paste in web page.

contents.pasteAndMatchStyle()

Executes the editing command pasteAndMatchStyle in web page.

contents.delete()

Executes the editing command delete in web page.

contents.selectAll()

Executes the editing command selectAll in web page.

contents.unselect()

Executes the editing command unselect in web page.

contents.replace(text)

* text String

Executes the editing command replace in web page.

contents.replaceMisspelling(text)

* text String

Executes the editing command replaceMisspelling in web page.

contents.insertText(text)

* text String

Inserts text to the focused element.

contents.findInPage(text[, options])

* text String - Content to be searched, must not be empty.
* options Object (optional)
  + forward Boolean - (optional) Whether to search forward or backward, defaults to true.
  + findNext Boolean - (optional) Whether the operation is first request or a follow up, defaults to false.
  + matchCase Boolean - (optional) Whether search should be case-sensitive, defaults to false.
  + wordStart Boolean - (optional) Whether to look only at the start of words. defaults to false.
  + medialCapitalAsWordStart Boolean - (optional) When combined with wordStart, accepts a match in the middle of a word if the match begins with an uppercase letter followed by a lowercase or non-letter. Accepts several other intra-word matches, defaults to false.

Starts a request to find all matches for the text in the web page and returns an Integer representing the request id used for the request. The result of the request can be obtained by subscribing to [found-in-page](https://electron.atom.io/docs/api/web-contents#event-found-in-page)event.

contents.stopFindInPage(action)

* action String - Specifies the action to take place when ending [webContents.findInPage] request.
  + clearSelection - Clear the selection.
  + keepSelection - Translate the selection into a normal selection.
  + activateSelection - Focus and click the selection node.

Stops any findInPage request for the webContents with the provided action.

**const** {webContents} = require('electron')

webContents.on('found-in-page', (event, result) => {

**if** (result.finalUpdate) webContents.stopFindInPage('clearSelection')

})

**const** requestId = webContents.findInPage('api')

console.log(requestId)

contents.capturePage([rect, ]callback)

* rect [Rectangle](https://electron.atom.io/docs/api/structures/rectangle) (optional) - The area of the page to be captured
* callback Function
  + image [NativeImage](https://electron.atom.io/docs/api/native-image)

Captures a snapshot of the page within rect. Upon completion callback will be called with callback(image). The image is an instance of [NativeImage](https://electron.atom.io/docs/api/native-image) that stores data of the snapshot. Omitting rect will capture the whole visible page.

contents.hasServiceWorker(callback)

* callback Function
  + hasWorker Boolean

Checks if any ServiceWorker is registered and returns a boolean as response to callback.

contents.unregisterServiceWorker(callback)

* callback Function
  + success Boolean

Unregisters any ServiceWorker if present and returns a boolean as response to callback when the JS promise is fulfilled or false when the JS promise is rejected.

contents.getPrinters()

Get the system printer list.

Returns [PrinterInfo[]](https://electron.atom.io/docs/api/structures/printer-info)

contents.print([options])

* options Object (optional)
  + silent Boolean (optional) - Don’t ask user for print settings. Default is false.
  + printBackground Boolean (optional) - Also prints the background color and image of the web page. Default is false.
  + deviceName String (optional) - Set the printer device name to use. Default is ''.

Prints window’s web page. When silent is set to true, Electron will pick the system’s default printer if deviceName is empty and the default settings for printing.

Calling window.print() in web page is equivalent to calling webContents.print({silent: false, printBackground: false, deviceName: ''}).

Use page-break-before: always; CSS style to force to print to a new page.

contents.printToPDF(options, callback)

* options Object
  + marginsType Integer - (optional) Specifies the type of margins to use. Uses 0 for default margin, 1 for no margin, and 2 for minimum margin.
  + pageSize String - (optional) Specify page size of the generated PDF. Can be A3, A4, A5, Legal, Letter, Tabloid or an Object containing height and width in microns.
  + printBackground Boolean - (optional) Whether to print CSS backgrounds.
  + printSelectionOnly Boolean - (optional) Whether to print selection only.
  + landscape Boolean - (optional) true for landscape, false for portrait.
* callback Function
  + error Error
  + data Buffer

Prints window’s web page as PDF with Chromium’s preview printing custom settings.

The callback will be called with callback(error, data) on completion. The data is a Buffer that contains the generated PDF data.

The landscape will be ignored if @page CSS at-rule is used in the web page.

By default, an empty options will be regarded as:

{

marginsType: 0,

printBackground: false,

printSelectionOnly: false,

landscape: false

}

Use page-break-before: always; CSS style to force to print to a new page.

An example of webContents.printToPDF:

**const** {BrowserWindow} = require('electron')

**const** fs = require('fs')

**let** win = **new** BrowserWindow({width: 800, height: 600})

win.loadURL('http://github.com')

win.webContents.on('did-finish-load', () => {

*// Use default printing options*

win.webContents.printToPDF({}, (error, data) => {

**if** (error) **throw** error

fs.writeFile('/tmp/print.pdf', data, (error) => {

**if** (error) **throw** error

console.log('Write PDF successfully.')

})

})

})

contents.addWorkSpace(path)

* path String

Adds the specified path to DevTools workspace. Must be used after DevTools creation:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.webContents.on('devtools-opened', () => {

win.webContents.addWorkSpace(\_\_dirname)

})

contents.removeWorkSpace(path)

* path String

Removes the specified path from DevTools workspace.

contents.openDevTools([options])

* options Object (optional)
  + mode String - Opens the devtools with specified dock state, can be right, bottom, undocked, detach. Defaults to last used dock state. In undocked mode it’s possible to dock back. In detach mode it’s not.

Opens the devtools.

contents.closeDevTools()

Closes the devtools.

contents.isDevToolsOpened()

Returns Boolean - Whether the devtools is opened.

contents.isDevToolsFocused()

Returns Boolean - Whether the devtools view is focused .

contents.toggleDevTools()

Toggles the developer tools.

contents.inspectElement(x, y)

* x Integer
* y Integer

Starts inspecting element at position (x, y).

contents.inspectServiceWorker()

Opens the developer tools for the service worker context.

contents.send(channel[, arg1][, arg2][, ...])

* channel String
* ...args any[]

Send an asynchronous message to renderer process via channel, you can also send arbitrary arguments. Arguments will be serialized in JSON internally and hence no functions or prototype chain will be included.

The renderer process can handle the message by listening to channel with the ipcRenderer module.

An example of sending messages from the main process to the renderer process:

*// In the main process.*

**const** {app, BrowserWindow} = require('electron')

**let** win = null

app.on('ready', () => {

win = **new** BrowserWindow({width: 800, height: 600})

win.loadURL(`file:*//${\_\_dirname}/index.html`)*

win.webContents.on('did-finish-load', () => {

win.webContents.send('ping', 'whoooooooh!')

})

})

*<!-- index.html -->*

<html>

<body>

<script>

require('electron').ipcRenderer.on('ping', (event, message) => {

console.log(message) *// Prints 'whoooooooh!'*

})

</script>

</body>

</html>

contents.enableDeviceEmulation(parameters)

* parameters Object
  + screenPosition String - Specify the screen type to emulate (default: desktop)
    - desktop - Desktop screen type
    - mobile - Mobile screen type
  + screenSize [Size](https://electron.atom.io/docs/api/structures/size) - Set the emulated screen size (screenPosition == mobile)
  + viewPosition [Point](https://electron.atom.io/docs/api/structures/point) - Position the view on the screen (screenPosition == mobile) (default: {x: 0, y: 0})
  + deviceScaleFactor Integer - Set the device scale factor (if zero defaults to original device scale factor) (default: 0)
  + viewSize [Size](https://electron.atom.io/docs/api/structures/size) - Set the emulated view size (empty means no override)
  + fitToView Boolean - Whether emulated view should be scaled down if necessary to fit into available space (default: false)
  + offset [Point](https://electron.atom.io/docs/api/structures/point) - Offset of the emulated view inside available space (not in fit to view mode) (default: {x: 0,y: 0})
  + scale Float - Scale of emulated view inside available space (not in fit to view mode) (default: 1)

Enable device emulation with the given parameters.

contents.disableDeviceEmulation()

Disable device emulation enabled by webContents.enableDeviceEmulation.

contents.sendInputEvent(event)

* event Object
  + type String (**required**) - The type of the event, can be mouseDown, mouseUp, mouseEnter, mouseLeave, contextMenu, mouseWheel, mouseMove, keyDown, keyUp, char.
  + modifiers String[] - An array of modifiers of the event, can include shift, control, alt, meta, isKeypad, isAutoRepeat, leftButtonDown, middleButtonDown, rightButtonDown, capsLock, numLock, left, right.

Sends an input event to the page. **Note:** The BrowserWindow containing the contents needs to be focused for sendInputEvent() to work.

For keyboard events, the event object also have following properties:

* keyCode String (**required**) - The character that will be sent as the keyboard event. Should only use the valid key codes in [Accelerator](https://electron.atom.io/docs/api/accelerator).

For mouse events, the event object also have following properties:

* x Integer (**required**)
* y Integer (**required**)
* button String - The button pressed, can be left, middle, right
* globalX Integer
* globalY Integer
* movementX Integer
* movementY Integer
* clickCount Integer

For the mouseWheel event, the event object also have following properties:

* deltaX Integer
* deltaY Integer
* wheelTicksX Integer
* wheelTicksY Integer
* accelerationRatioX Integer
* accelerationRatioY Integer
* hasPreciseScrollingDeltas Boolean
* canScroll Boolean

contents.beginFrameSubscription([onlyDirty ,]callback)

* onlyDirty Boolean (optional) - Defaults to false
* callback Function
  + frameBuffer Buffer
  + dirtyRect [Rectangle](https://electron.atom.io/docs/api/structures/rectangle)

Begin subscribing for presentation events and captured frames, the callback will be called with callback(frameBuffer, dirtyRect) when there is a presentation event.

The frameBuffer is a Buffer that contains raw pixel data. On most machines, the pixel data is effectively stored in 32bit BGRA format, but the actual representation depends on the endianness of the processor (most modern processors are little-endian, on machines with big-endian processors the data is in 32bit ARGB format).

The dirtyRect is an object with x, y, width, height properties that describes which part of the page was repainted. If onlyDirty is set to true, frameBuffer will only contain the repainted area. onlyDirty defaults to false.

contents.endFrameSubscription()

End subscribing for frame presentation events.

contents.startDrag(item)

* item Object
  + file String or files Array - The path(s) to the file(s) being dragged.
  + icon [NativeImage](https://electron.atom.io/docs/api/native-image) - The image must be non-empty on macOS.

Sets the item as dragging item for current drag-drop operation, file is the absolute path of the file to be dragged, and icon is the image showing under the cursor when dragging.

contents.savePage(fullPath, saveType, callback)

* fullPath String - The full file path.
* saveType String - Specify the save type.
  + HTMLOnly - Save only the HTML of the page.
  + HTMLComplete - Save complete-html page.
  + MHTML - Save complete-html page as MHTML.
* callback Function - (error) => {}.
  + error Error

Returns Boolean - true if the process of saving page has been initiated successfully.

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.loadURL('https://github.com')

win.webContents.on('did-finish-load', () => {

win.webContents.savePage('/tmp/test.html', 'HTMLComplete', (error) => {

**if** (!error) console.log('Save page successfully')

})

})

contents.showDefinitionForSelection() macOS

Shows pop-up dictionary that searches the selected word on the page.

contents.setSize(options)

Set the size of the page. This is only supported for <webview> guest contents.

* options Object
  + normal Object (optional) - Normal size of the page. This can be used in combination with the [disableguestresize](https://electron.atom.io/docs/api/web-view-tag#disableguestresize) attribute to manually resize the webview guest contents.
    - width Integer
    - height Integer

contents.isOffscreen()

Returns Boolean - Indicates whether *offscreen rendering* is enabled.

contents.startPainting()

If *offscreen rendering* is enabled and not painting, start painting.

contents.stopPainting()

If *offscreen rendering* is enabled and painting, stop painting.

contents.isPainting()

Returns Boolean - If *offscreen rendering* is enabled returns whether it is currently painting.

contents.setFrameRate(fps)

* fps Integer

If *offscreen rendering* is enabled sets the frame rate to the specified number. Only values between 1 and 60 are accepted.

contents.getFrameRate()

Returns Integer - If *offscreen rendering* is enabled returns the current frame rate.

contents.invalidate()

Schedules a full repaint of the window this web contents is in.

If *offscreen rendering* is enabled invalidates the frame and generates a new one through the 'paint' event.

contents.getWebRTCIPHandlingPolicy()

Returns String - Returns the WebRTC IP Handling Policy.

contents.setWebRTCIPHandlingPolicy(policy)

* policy String - Specify the WebRTC IP Handling Policy.
  + default - Exposes user’s public and local IPs. This is the default behavior. When this policy is used, WebRTC has the right to enumerate all interfaces and bind them to discover public interfaces.
  + default\_public\_interface\_only - Exposes user’s public IP, but does not expose user’s local IP. When this policy is used, WebRTC should only use the default route used by http. This doesn’t expose any local addresses.
  + default\_public\_and\_private\_interfaces - Exposes user’s public and local IPs. When this policy is used, WebRTC should only use the default route used by http. This also exposes the associated default private address. Default route is the route chosen by the OS on a multi-homed endpoint.
  + disable\_non\_proxied\_udp - Does not expose public or local IPs. When this policy is used, WebRTC should only use TCP to contact peers or servers unless the proxy server supports UDP.

Setting the WebRTC IP handling policy allows you to control which IPs are exposed via WebRTC. See [BrowserLeaks](https://browserleaks.com/webrtc) for more details.

contents.getOSProcessId()

Returns Integer - The pid of the associated renderer process.

Instance Properties

contents.id

A Integer representing the unique ID of this WebContents.

contents.session

A [Session](https://electron.atom.io/docs/api/session) used by this webContents.

contents.hostWebContents

A [WebContents](https://electron.atom.io/docs/api/web-contents) instance that might own this WebContents.

contents.devToolsWebContents

A WebContents of DevTools for this WebContents.

**Note:** Users should never store this object because it may become null when the DevTools has been closed.

contents.debugger

A [Debugger](https://electron.atom.io/docs/api/debugger) instance for this webContents.

webFrame

Customize the rendering of the current web page.

Process: [Renderer](https://electron.atom.io/docs/glossary#renderer-process)

An example of zooming current page to 200%.

**const** {webFrame} = require('electron')

webFrame.setZoomFactor(2)

Methods

The webFrame module has the following methods:

webFrame.setZoomFactor(factor)

* factor Number - Zoom factor.

Changes the zoom factor to the specified factor. Zoom factor is zoom percent divided by 100, so 300% = 3.0.

webFrame.getZoomFactor()

Returns Number - The current zoom factor.

webFrame.setZoomLevel(level)

* level Number - Zoom level

Changes the zoom level to the specified level. The original size is 0 and each increment above or below represents zooming 20% larger or smaller to default limits of 300% and 50% of original size, respectively.

webFrame.getZoomLevel()

Returns Number - The current zoom level.

webFrame.setZoomLevelLimits(minimumLevel, maximumLevel)

* minimumLevel Number
* maximumLevel Number

**Deprecated:** Call setVisualZoomLevelLimits instead to set the visual zoom level limits. This method will be removed in Electron 2.0.

webFrame.setVisualZoomLevelLimits(minimumLevel, maximumLevel)

* minimumLevel Number
* maximumLevel Number

Sets the maximum and minimum pinch-to-zoom level.

webFrame.setLayoutZoomLevelLimits(minimumLevel, maximumLevel)

* minimumLevel Number
* maximumLevel Number

Sets the maximum and minimum layout-based (i.e. non-visual) zoom level.

webFrame.setSpellCheckProvider(language, autoCorrectWord, provider)

* language String
* autoCorrectWord Boolean
* provider Object
  + spellCheck Function - Returns Boolean
    - text String

Sets a provider for spell checking in input fields and text areas.

The provider must be an object that has a spellCheck method that returns whether the word passed is correctly spelled.

An example of using [node-spellchecker](https://github.com/atom/node-spellchecker) as provider:

**const** {webFrame} = require('electron')

webFrame.setSpellCheckProvider('en-US', true, {

spellCheck (text) {

**return** !(require('spellchecker').isMisspelled(text))

}

})

webFrame.registerURLSchemeAsSecure(scheme)

* scheme String

Registers the scheme as secure scheme.

Secure schemes do not trigger mixed content warnings. For example, https and data are secure schemes because they cannot be corrupted by active network attackers.

webFrame.registerURLSchemeAsBypassingCSP(scheme)

* scheme String

Resources will be loaded from this scheme regardless of the current page’s Content Security Policy.

webFrame.registerURLSchemeAsPrivileged(scheme[, options])

* scheme String
* options Object (optional)
  + secure Boolean - (optional) Default true.
  + bypassCSP Boolean - (optional) Default true.
  + allowServiceWorkers Boolean - (optional) Default true.
  + supportFetchAPI Boolean - (optional) Default true.
  + corsEnabled Boolean - (optional) Default true.

Registers the scheme as secure, bypasses content security policy for resources, allows registering ServiceWorker and supports fetch API.

Specify an option with the value of false to omit it from the registration. An example of registering a privileged scheme, without bypassing Content Security Policy:

**const** {webFrame} = require('electron')

webFrame.registerURLSchemeAsPrivileged('foo', { bypassCSP: false })

webFrame.insertText(text)

* text String

Inserts text to the focused element.

webFrame.executeJavaScript(code[, userGesture, callback])

* code String
* userGesture Boolean (optional) - Default is false.
* callback Function (optional) - Called after script has been executed.
  + result Any

Returns Promise - A promise that resolves with the result of the executed code or is rejected if the result of the code is a rejected promise.

Evaluates code in page.

In the browser window some HTML APIs like requestFullScreen can only be invoked by a gesture from the user. Setting userGesture to true will remove this limitation.

webFrame.getResourceUsage()

Returns Object:

* images [MemoryUsageDetails](https://electron.atom.io/docs/api/structures/memory-usage-details)
* cssStyleSheets [MemoryUsageDetails](https://electron.atom.io/docs/api/structures/memory-usage-details)
* xslStyleSheets [MemoryUsageDetails](https://electron.atom.io/docs/api/structures/memory-usage-details)
* fonts [MemoryUsageDetails](https://electron.atom.io/docs/api/structures/memory-usage-details)
* other [MemoryUsageDetails](https://electron.atom.io/docs/api/structures/memory-usage-details)

Returns an object describing usage information of Blink’s internal memory caches.

**const** {webFrame} = require('electron')

console.log(webFrame.getResourceUsage())

This will generate:

{

images: {

count: 22,

size: 2549,

liveSize: 2542

},

cssStyleSheets: { */\* same with "images" \*/* },

xslStyleSheets: { */\* same with "images" \*/* },

fonts: { */\* same with "images" \*/* },

other: { */\* same with "images" \*/* }

}

webFrame.clearCache()

Attempts to free memory that is no longer being used (like images from a previous navigation).

Note that blindly calling this method probably makes Electron slower since it will have to refill these emptied caches, you should only call it if an event in your app has occurred that makes you think your page is actually using less memory (i.e. you have navigated from a super heavy page to a mostly empty one, and intend to stay there).

Class: WebRequest

Intercept and modify the contents of a request at various stages of its lifetime.

Process: [Main](https://electron.atom.io/docs/glossary#main-process)

Instances of the WebRequest class are accessed by using the webRequest property of a Session.

The methods of WebRequest accept an optional filter and a listener. The listener will be called with listener(details) when the API’s event has happened. The details object describes the request. Passing nullas listener will unsubscribe from the event.

The filter object has a urls property which is an Array of URL patterns that will be used to filter out the requests that do not match the URL patterns. If the filter is omitted then all requests will be matched.

For certain events the listener is passed with a callback, which should be called with a response object when listener has done its work.

An example of adding User-Agent header for requests:

**const** {session} = require('electron')

*// Modify the user agent for all requests to the following urls.*

**const** filter = {

urls: ['https://\*.github.com/\*', '\*://electron.github.io']

}

session.defaultSession.webRequest.onBeforeSendHeaders(filter, (details, callback) => {

details.requestHeaders['User-Agent'] = 'MyAgent'

callback({cancel: false, requestHeaders: details.requestHeaders})

})

Instance Methods

The following methods are available on instances of WebRequest:

webRequest.onBeforeRequest([filter, ]listener)

* filter Object
  + urls String[] - Array of URL patterns that will be used to filter out the requests that do not match the URL patterns.
* listener Function
  + details Object
    - id Integer
    - url String
    - method String
    - resourceType String
    - timestamp Double
    - uploadData [UploadData[]](https://electron.atom.io/docs/api/structures/upload-data)
  + callback Function
    - response Object
      * cancel Boolean (optional)
      * redirectURL String (optional) - The original request is prevented from being sent or completed and is instead redirected to the given URL.

The listener will be called with listener(details, callback) when a request is about to occur.

The uploadData is an array of UploadData objects.

The callback has to be called with an response object.

webRequest.onBeforeSendHeaders([filter, ]listener)

* filter Object
  + urls String[] - Array of URL patterns that will be used to filter out the requests that do not match the URL patterns.
* listener Function

The listener will be called with listener(details, callback) before sending an HTTP request, once the request headers are available. This may occur after a TCP connection is made to the server, but before any http data is sent.

* details Object
  + id Integer
  + url String
  + method String
  + resourceType String
  + timestamp Double
  + requestHeaders Object
* callback Function
  + response Object
    - cancel Boolean (optional)
    - requestHeaders Object (optional) - When provided, request will be made with these headers.

The callback has to be called with an response object.

webRequest.onSendHeaders([filter, ]listener)

* filter Object
  + urls String[] - Array of URL patterns that will be used to filter out the requests that do not match the URL patterns.
* listener Function
  + details Object
    - id Integer
    - url String
    - method String
    - resourceType String
    - timestamp Double
    - requestHeaders Object

The listener will be called with listener(details) just before a request is going to be sent to the server, modifications of previous onBeforeSendHeaders response are visible by the time this listener is fired.

webRequest.onHeadersReceived([filter, ]listener)

* filter Object
  + urls String[] - Array of URL patterns that will be used to filter out the requests that do not match the URL patterns.
* listener Function

The listener will be called with listener(details, callback) when HTTP response headers of a request have been received.

* details Object
  + id String
  + url String
  + method String
  + resourceType String
  + timestamp Double
  + statusLine String
  + statusCode Integer
  + responseHeaders Object
* callback Function
  + response Object
    - cancel Boolean
    - responseHeaders Object (optional) - When provided, the server is assumed to have responded with these headers.
    - statusLine String (optional) - Should be provided when overriding responseHeaders to change header status otherwise original response header’s status will be used.

The callback has to be called with an response object.

webRequest.onResponseStarted([filter, ]listener)

* filter Object
  + urls String[] - Array of URL patterns that will be used to filter out the requests that do not match the URL patterns.
* listener Function
  + details Object
    - id Integer
    - url String
    - method String
    - resourceType String
    - timestamp Double
    - responseHeaders Object
    - fromCache Boolean - Indicates whether the response was fetched from disk cache.
    - statusCode Integer
    - statusLine String

The listener will be called with listener(details) when first byte of the response body is received. For HTTP requests, this means that the status line and response headers are available.

webRequest.onBeforeRedirect([filter, ]listener)

* filter Object
  + urls String[] - Array of URL patterns that will be used to filter out the requests that do not match the URL patterns.
* listener Function
  + details Object
    - id String
    - url String
    - method String
    - resourceType String
    - timestamp Double
    - redirectURL String
    - statusCode Integer
    - ip String (optional) - The server IP address that the request was actually sent to.
    - fromCache Boolean
    - responseHeaders Object

The listener will be called with listener(details) when a server initiated redirect is about to occur.

webRequest.onCompleted([filter, ]listener)

* filter Object
  + urls String[] - Array of URL patterns that will be used to filter out the requests that do not match the URL patterns.
* listener Function
  + details Object
    - id Integer
    - url String
    - method String
    - resourceType String
    - timestamp Double
    - responseHeaders Object
    - fromCache Boolean
    - statusCode Integer
    - statusLine String

The listener will be called with listener(details) when a request is completed.

webRequest.onErrorOccurred([filter, ]listener)

* filter Object
  + urls String[] - Array of URL patterns that will be used to filter out the requests that do not match the URL patterns.
* listener Function
  + details Object
    - id Integer
    - url String
    - method String
    - resourceType String
    - timestamp Double
    - fromCache Boolean
    - error String - The error description.

The listener will be called with listener(details) when an error occurs.

<webview> Tag

Display external web content in an isolated frame and process.

Process: [Renderer](https://electron.atom.io/docs/tutorial/quick-start#renderer-process)

Use the webview tag to embed ‘guest’ content (such as web pages) in your Electron app. The guest content is contained within the webview container. An embedded page within your app controls how the guest content is laid out and rendered.

Unlike an iframe, the webview runs in a separate process than your app. It doesn’t have the same permissions as your web page and all interactions between your app and embedded content will be asynchronous. This keeps your app safe from the embedded content. **Note:** Most methods called on the webview from the host page require a synchronous call to the main process.

Example

To embed a web page in your app, add the webview tag to your app’s embedder page (this is the app page that will display the guest content). In its simplest form, the webview tag includes the src of the web page and css styles that control the appearance of the webview container:

<webview id="foo" src="https://www.github.com/" style="display:inline-flex; width:640px; height:480px"></webview>

If you want to control the guest content in any way, you can write JavaScript that listens for webview events and responds to those events using the webview methods. Here’s sample code with two event listeners: one that listens for the web page to start loading, the other for the web page to stop loading, and displays a “loading…” message during the load time:

<script>

onload = () => {

**const** webview = document.querySelector('webview')

**const** indicator = document.querySelector('.indicator')

**const** loadstart = () => {

indicator.innerText = 'loading...'

}

**const** loadstop = () => {

indicator.innerText = ''

}

webview.addEventListener('did-start-loading', loadstart)

webview.addEventListener('did-stop-loading', loadstop)

}

</script>

CSS Styling Notes

Please note that the webview tag’s style uses display:flex; internally to ensure the child object element fills the full height and width of its webview container when used with traditional and flexbox layouts (since v0.36.11). Please do not overwrite the default display:flex; CSS property, unless specifying display:inline-flex; for inline layout.

webview has issues being hidden using the hidden attribute or using display: none;. It can cause unusual rendering behaviour within its child browserplugin object and the web page is reloaded when the webview is un-hidden. The recommended approach is to hide the webview using visibility: hidden.

<style>

webview {

display:inline-flex;

width:640px;

height:480px;

}

webview**.hide** {

visibility: hidden;

}

</style>

Tag Attributes

The webview tag has the following attributes:

src

<webview src="https://www.github.com/"></webview>

Returns the visible URL. Writing to this attribute initiates top-level navigation.

Assigning src its own value will reload the current page.

The src attribute can also accept data URLs, such as data:text/plain,Hello, world!.

autosize

<webview src="https://www.github.com/" autosize minwidth="576" minheight="432"></webview>

When this attribute is present the webview container will automatically resize within the bounds specified by the attributes minwidth, minheight, maxwidth, and maxheight. These constraints do not impact the webview unless autosize is enabled. When autosize is enabled, the webview container size cannot be less than the minimum values or greater than the maximum.

nodeintegration

<webview src="http://www.google.com/" nodeintegration></webview>

When this attribute is present the guest page in webview will have node integration and can use node APIs like require and process to access low level system resources. Node integration is disabled by default in the guest page.

plugins

<webview src="https://www.github.com/" plugins></webview>

When this attribute is present the guest page in webview will be able to use browser plugins. Plugins are disabled by default.

preload

<webview src="https://www.github.com/" preload="./test.js"></webview>

Specifies a script that will be loaded before other scripts run in the guest page. The protocol of script’s URL must be either file: or asar:, because it will be loaded by require in guest page under the hood.

When the guest page doesn’t have node integration this script will still have access to all Node APIs, but global objects injected by Node will be deleted after this script has finished executing.

**Note:** This option will be appear as preloadURL (not preload) in the webPreferences specified to the will-attach-webview event.

httpreferrer

<webview src="https://www.github.com/" httpreferrer="http://cheng.guru"></webview>

Sets the referrer URL for the guest page.

useragent

<webview src="https://www.github.com/" useragent="Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; AS; rv:11.0) like Gecko"></webview>

Sets the user agent for the guest page before the page is navigated to. Once the page is loaded, use the setUserAgent method to change the user agent.

disablewebsecurity

<webview src="https://www.github.com/" disablewebsecurity></webview>

When this attribute is present the guest page will have web security disabled. Web security is enabled by default.

partition

<webview src="https://github.com" partition="persist:github"></webview>

<webview src="https://electron.atom.io" partition="electron"></webview>

Sets the session used by the page. If partition starts with persist:, the page will use a persistent session available to all pages in the app with the same partition. if there is no persist: prefix, the page will use an in-memory session. By assigning the same partition, multiple pages can share the same session. If the partitionis unset then default session of the app will be used.

This value can only be modified before the first navigation, since the session of an active renderer process cannot change. Subsequent attempts to modify the value will fail with a DOM exception.

allowpopups

<webview src="https://www.github.com/" allowpopups></webview>

When this attribute is present the guest page will be allowed to open new windows. Popups are disabled by default.

webpreferences

<webview src="https://github.com" webpreferences="allowRunningInsecureContent, javascript=no"></webview>

A list of strings which specifies the web preferences to be set on the webview, separated by ,. The full list of supported preference strings can be found in [BrowserWindow](https://electron.atom.io/docs/api/browser-window#new-browserwindowoptions).

The string follows the same format as the features string in window.open. A name by itself is given a trueboolean value. A preference can be set to another value by including an =, followed by the value. Special values yes and 1 are interpreted as true, while no and 0 are interpreted as false.

blinkfeatures

<webview src="https://www.github.com/" blinkfeatures="PreciseMemoryInfo, CSSVariables"></webview>

A list of strings which specifies the blink features to be enabled separated by ,. The full list of supported feature strings can be found in the [RuntimeEnabledFeatures.json5](https://cs.chromium.org/chromium/src/third_party/WebKit/Source/platform/RuntimeEnabledFeatures.json5?l=62) file.

disableblinkfeatures

<webview src="https://www.github.com/" disableblinkfeatures="PreciseMemoryInfo, CSSVariables"></webview>

A list of strings which specifies the blink features to be disabled separated by ,. The full list of supported feature strings can be found in the [RuntimeEnabledFeatures.json5](https://cs.chromium.org/chromium/src/third_party/WebKit/Source/platform/RuntimeEnabledFeatures.json5?l=62) file.

guestinstance

<webview src="https://www.github.com/" guestinstance="3"></webview>

A value that links the webview to a specific webContents. When a webview first loads a new webContents is created and this attribute is set to its instance identifier. Setting this attribute on a new or existing webview connects it to the existing webContents that currently renders in a different webview.

The existing webview will see the destroy event and will then create a new webContents when a new url is loaded.

disableguestresize

<webview src="https://www.github.com/" disableguestresize></webview>

When this attribute is present the webview contents will be prevented from resizing when the webview element itself is resized.

This can be used in combination with [webContents.setSize](https://electron.atom.io/docs/api/web-contents#contentssetsizeoptions) to manually resize the webview contents in reaction to a window size change. This can make resizing faster compared to relying on the webview element bounds to automatically resize the contents.

**const** {webContents} = require('electron')

*// We assume that `win` points to a `BrowserWindow` instance containing a*

*// `<webview>` with `disableguestresize`.*

win.on('resize', () => {

**const** [width, height] = win.getContentSize()

**for** (**let** wc of webContents.getAllWebContents()) {

*// Check if `wc` belongs to a webview in the `win` window.*

**if** (wc.hostWebContents &&

wc.hostWebContents.id === win.webContents.id) {

wc.setSize({

normal: {

width: width,

height: height

}

})

}

}

})

Methods

The webview tag has the following methods:

**Note:** The webview element must be loaded before using the methods.

**Example**

**const** webview = document.querySelector('webview')

webview.addEventListener('dom-ready', () => {

webview.openDevTools()

})

<webview>.loadURL(url[, options])

* url URL
* options Object (optional)
  + httpReferrer String (optional) - A HTTP Referrer url.
  + userAgent String (optional) - A user agent originating the request.
  + extraHeaders String (optional) - Extra headers separated by “\n”
  + postData ([UploadRawData[]](https://electron.atom.io/docs/api/structures/upload-raw-data) | [UploadFile[]](https://electron.atom.io/docs/api/structures/upload-file) | [UploadFileSystem[]](https://electron.atom.io/docs/api/structures/upload-file-system) | [UploadBlob[]](https://electron.atom.io/docs/api/structures/upload-blob)) - (optional)
  + baseURLForDataURL String (optional) - Base url (with trailing path separator) for files to be loaded by the data url. This is needed only if the specified url is a data url and needs to load other files.

Loads the url in the webview, the url must contain the protocol prefix, e.g. the http:// or file://.

<webview>.getURL()

Returns String - The URL of guest page.

<webview>.getTitle()

Returns String - The title of guest page.

<webview>.isLoading()

Returns Boolean - Whether guest page is still loading resources.

<webview>.isWaitingForResponse()

Returns Boolean - Whether the guest page is waiting for a first-response for the main resource of the page.

<webview>.stop()

Stops any pending navigation.

<webview>.reload()

Reloads the guest page.

<webview>.reloadIgnoringCache()

Reloads the guest page and ignores cache.

<webview>.canGoBack()

Returns Boolean - Whether the guest page can go back.

<webview>.canGoForward()

Returns Boolean - Whether the guest page can go forward.

<webview>.canGoToOffset(offset)

* offset Integer

Returns Boolean - Whether the guest page can go to offset.

<webview>.clearHistory()

Clears the navigation history.

<webview>.goBack()

Makes the guest page go back.

<webview>.goForward()

Makes the guest page go forward.

<webview>.goToIndex(index)

* index Integer

Navigates to the specified absolute index.

<webview>.goToOffset(offset)

* offset Integer

Navigates to the specified offset from the “current entry”.

<webview>.isCrashed()

Returns Boolean - Whether the renderer process has crashed.

<webview>.setUserAgent(userAgent)

* userAgent String

Overrides the user agent for the guest page.

<webview>.getUserAgent()

Returns String - The user agent for guest page.

<webview>.insertCSS(css)

* css String

Injects CSS into the guest page.

<webview>.executeJavaScript(code, userGesture, callback)

* code String
* userGesture Boolean - Default false.
* callback Function (optional) - Called after script has been executed.
  + result Any

Evaluates code in page. If userGesture is set, it will create the user gesture context in the page. HTML APIs like requestFullScreen, which require user action, can take advantage of this option for automation.

<webview>.openDevTools()

Opens a DevTools window for guest page.

<webview>.closeDevTools()

Closes the DevTools window of guest page.

<webview>.isDevToolsOpened()

Returns Boolean - Whether guest page has a DevTools window attached.

<webview>.isDevToolsFocused()

Returns Boolean - Whether DevTools window of guest page is focused.

<webview>.inspectElement(x, y)

* x Integer
* y Integer

Starts inspecting element at position (x, y) of guest page.

<webview>.inspectServiceWorker()

Opens the DevTools for the service worker context present in the guest page.

<webview>.setAudioMuted(muted)

* muted Boolean

Set guest page muted.

<webview>.isAudioMuted()

Returns Boolean - Whether guest page has been muted.

<webview>.undo()

Executes editing command undo in page.

<webview>.redo()

Executes editing command redo in page.

<webview>.cut()

Executes editing command cut in page.

<webview>.copy()

Executes editing command copy in page.

<webview>.paste()

Executes editing command paste in page.

<webview>.pasteAndMatchStyle()

Executes editing command pasteAndMatchStyle in page.

<webview>.delete()

Executes editing command delete in page.

<webview>.selectAll()

Executes editing command selectAll in page.

<webview>.unselect()

Executes editing command unselect in page.

<webview>.replace(text)

* text String

Executes editing command replace in page.

<webview>.replaceMisspelling(text)

* text String

Executes editing command replaceMisspelling in page.

<webview>.insertText(text)

* text String

Inserts text to the focused element.

<webview>.findInPage(text[, options])

* text String - Content to be searched, must not be empty.
* options Object (optional)
  + forward Boolean - (optional) Whether to search forward or backward, defaults to true.
  + findNext Boolean - (optional) Whether the operation is first request or a follow up, defaults to false.
  + matchCase Boolean - (optional) Whether search should be case-sensitive, defaults to false.
  + wordStart Boolean - (optional) Whether to look only at the start of words. defaults to false.
  + medialCapitalAsWordStart Boolean - (optional) When combined with wordStart, accepts a match in the middle of a word if the match begins with an uppercase letter followed by a lowercase or non-letter. Accepts several other intra-word matches, defaults to false.

Starts a request to find all matches for the text in the web page and returns an Integer representing the request id used for the request. The result of the request can be obtained by subscribing to [found-in-page](https://electron.atom.io/docs/api/webview-tag#event-found-in-page)event.

<webview>.stopFindInPage(action)

* action String - Specifies the action to take place when ending [<webview>.findInPage](https://electron.atom.io/docs/api/webview-tag#webviewtagfindinpage) request.
  + clearSelection - Clear the selection.
  + keepSelection - Translate the selection into a normal selection.
  + activateSelection - Focus and click the selection node.

Stops any findInPage request for the webview with the provided action.

<webview>.print([options])

* options Object (optional)
  + silent Boolean (optional) - Don’t ask user for print settings. Default is false.
  + printBackground Boolean (optional) - Also prints the background color and image of the web page. Default is false.
  + deviceName String (optional) - Set the printer device name to use. Default is ''.

Prints webview’s web page. Same as webContents.print([options]).

<webview>.printToPDF(options, callback)

* options Object
  + marginsType Integer - (optional) Specifies the type of margins to use. Uses 0 for default margin, 1 for no margin, and 2 for minimum margin.
  + pageSize String - (optional) Specify page size of the generated PDF. Can be A3, A4, A5, Legal, Letter, Tabloid or an Object containing height and width in microns.
  + printBackground Boolean - (optional) Whether to print CSS backgrounds.
  + printSelectionOnly Boolean - (optional) Whether to print selection only.
  + landscape Boolean - (optional) true for landscape, false for portrait.
* callback Function
  + error Error
  + data Buffer

Prints webview’s web page as PDF, Same as webContents.printToPDF(options, callback).

<webview>.capturePage([rect, ]callback)

* rect [Rectangle](https://electron.atom.io/docs/api/structures/rectangle) (optional) - The area of the page to be captured
* callback Function
  + image [NativeImage](https://electron.atom.io/docs/api/native-image)

Captures a snapshot of the webview’s page. Same as webContents.capturePage([rect, ]callback).

<webview>.send(channel[, arg1][, arg2][, ...])

* channel String
* ...args any[]

Send an asynchronous message to renderer process via channel, you can also send arbitrary arguments. The renderer process can handle the message by listening to the channel event with the ipcRenderer module.

See [webContents.send](https://electron.atom.io/docs/api/web-contents#webcontentssendchannel-args) for examples.

<webview>.sendInputEvent(event)

* event Object

Sends an input event to the page.

See [webContents.sendInputEvent](https://electron.atom.io/docs/api/web-contents#webcontentssendinputeventevent) for detailed description of event object.

<webview>.setZoomFactor(factor)

* factor Number - Zoom factor.

Changes the zoom factor to the specified factor. Zoom factor is zoom percent divided by 100, so 300% = 3.0.

<webview>.setZoomLevel(level)

* level Number - Zoom level

Changes the zoom level to the specified level. The original size is 0 and each increment above or below represents zooming 20% larger or smaller to default limits of 300% and 50% of original size, respectively.

<webview>.showDefinitionForSelection() macOS

Shows pop-up dictionary that searches the selected word on the page.

<webview>.getWebContents()

Returns [WebContents](https://electron.atom.io/docs/api/web-contents) - The web contents associated with this webview.

DOM events

The following DOM events are available to the webview tag:

Event: ‘load-commit’

Returns:

* url String
* isMainFrame Boolean

Fired when a load has committed. This includes navigation within the current document as well as subframe document-level loads, but does not include asynchronous resource loads.

Event: ‘did-finish-load’

Fired when the navigation is done, i.e. the spinner of the tab will stop spinning, and the onload event is dispatched.

Event: ‘did-fail-load’

Returns:

* errorCode Integer
* errorDescription String
* validatedURL String
* isMainFrame Boolean

This event is like did-finish-load, but fired when the load failed or was cancelled, e.g. window.stop() is invoked.

Event: ‘did-frame-finish-load’

Returns:

* isMainFrame Boolean

Fired when a frame has done navigation.

Event: ‘did-start-loading’

Corresponds to the points in time when the spinner of the tab starts spinning.

Event: ‘did-stop-loading’

Corresponds to the points in time when the spinner of the tab stops spinning.

Event: ‘did-get-response-details’

Returns:

* status Boolean
* newURL String
* originalURL String
* httpResponseCode Integer
* requestMethod String
* referrer String
* headers Object
* resourceType String

Fired when details regarding a requested resource is available. status indicates socket connection to download the resource.

Event: ‘did-get-redirect-request’

Returns:

* oldURL String
* newURL String
* isMainFrame Boolean

Fired when a redirect was received while requesting a resource.

Event: ‘dom-ready’

Fired when document in the given frame is loaded.

Event: ‘page-title-updated’

Returns:

* title String
* explicitSet Boolean

Fired when page title is set during navigation. explicitSet is false when title is synthesized from file url.

Event: ‘page-favicon-updated’

Returns:

* favicons String[] - Array of URLs.

Fired when page receives favicon urls.

Event: ‘enter-html-full-screen’

Fired when page enters fullscreen triggered by HTML API.

Event: ‘leave-html-full-screen’

Fired when page leaves fullscreen triggered by HTML API.

Event: ‘console-message’

Returns:

* level Integer
* message String
* line Integer
* sourceId String

Fired when the guest window logs a console message.

The following example code forwards all log messages to the embedder’s console without regard for log level or other properties.

**const** webview = document.querySelector('webview')

webview.addEventListener('console-message', (e) => {

console.log('Guest page logged a message:', e.message)

})

Event: ‘found-in-page’

Returns:

* result Object
  + requestId Integer
  + activeMatchOrdinal Integer - Position of the active match.
  + matches Integer - Number of Matches.
  + selectionArea Object - Coordinates of first match region.
  + finalUpdate Boolean

Fired when a result is available for [webview.findInPage](https://electron.atom.io/docs/api/webview-tag#webviewtagfindinpage) request.

**const** webview = document.querySelector('webview')

webview.addEventListener('found-in-page', (e) => {

webview.stopFindInPage('keepSelection')

})

**const** requestId = webview.findInPage('test')

console.log(requestId)

Event: ‘new-window’

Returns:

* url String
* frameName String
* disposition String - Can be default, foreground-tab, background-tab, new-window, save-to-disk and other.
* options Object - The options which should be used for creating the new BrowserWindow.

Fired when the guest page attempts to open a new browser window.

The following example code opens the new url in system’s default browser.

**const** {shell} = require('electron')

**const** webview = document.querySelector('webview')

webview.addEventListener('new-window', (e) => {

**const** protocol = require('url').parse(e.url).protocol

**if** (protocol === 'http:' || protocol === 'https:') {

shell.openExternal(e.url)

}

})

Event: ‘will-navigate’

Returns:

* url String

Emitted when a user or the page wants to start navigation. It can happen when the window.location object is changed or a user clicks a link in the page.

This event will not emit when the navigation is started programmatically with APIs like <webview>.loadURL and <webview>.back.

It is also not emitted during in-page navigation, such as clicking anchor links or updating the window.location.hash. Use did-navigate-in-page event for this purpose.

Calling event.preventDefault() does **NOT** have any effect.

Event: ‘did-navigate’

Returns:

* url String

Emitted when a navigation is done.

This event is not emitted for in-page navigations, such as clicking anchor links or updating the window.location.hash. Use did-navigate-in-page event for this purpose.

Event: ‘did-navigate-in-page’

Returns:

* isMainFrame Boolean
* url String

Emitted when an in-page navigation happened.

When in-page navigation happens, the page URL changes but does not cause navigation outside of the page. Examples of this occurring are when anchor links are clicked or when the DOM hashchange event is triggered.

Event: ‘close’

Fired when the guest page attempts to close itself.

The following example code navigates the webview to about:blank when the guest attempts to close itself.

**const** webview = document.querySelector('webview')

webview.addEventListener('close', () => {

webview.src = 'about:blank'

})

Event: ‘ipc-message’

Returns:

* channel String
* args Array

Fired when the guest page has sent an asynchronous message to embedder page.

With sendToHost method and ipc-message event you can easily communicate between guest page and embedder page:

*// In embedder page.*

**const** webview = document.querySelector('webview')

webview.addEventListener('ipc-message', (event) => {

console.log(event.channel)

*// Prints "pong"*

})

webview.send('ping')

*// In guest page.*

**const** {ipcRenderer} = require('electron')

ipcRenderer.on('ping', () => {

ipcRenderer.sendToHost('pong')

})

Event: ‘crashed’

Fired when the renderer process is crashed.

Event: ‘gpu-crashed’

Fired when the gpu process is crashed.

Event: ‘plugin-crashed’

Returns:

* name String
* version String

Fired when a plugin process is crashed.

Event: ‘destroyed’

Fired when the WebContents is destroyed.

Event: ‘media-started-playing’

Emitted when media starts playing.

Event: ‘media-paused’

Emitted when media is paused or done playing.

Event: ‘did-change-theme-color’

Returns:

* themeColor String

Emitted when a page’s theme color changes. This is usually due to encountering a meta tag:

<meta name='theme-color' content='#ff0000'>

Event: ‘update-target-url’

Returns:

* url String

Emitted when mouse moves over a link or the keyboard moves the focus to a link.

Event: ‘devtools-opened’

Emitted when DevTools is opened.

Event: ‘devtools-closed’

Emitted when DevTools is closed.

Event: ‘devtools-focused’

Emitted when DevTools is focused / opened.

window.open Function

Open a new window and load a URL.

When window.open is called to create a new window in a web page, a new instance of BrowserWindow will be created for the url and a proxy will be returned to window.open to let the page have limited control over it.

The proxy has limited standard functionality implemented to be compatible with traditional web pages. For full control of the new window you should create a BrowserWindow directly.

The newly created BrowserWindow will inherit the parent window’s options by default. To override inherited options you can set them in the features string.

window.open(url[, frameName][, features])

* url String
* frameName String (optional)
* features String (optional)

Returns [BrowserWindowProxy](https://electron.atom.io/docs/api/browser-window-proxy) - Creates a new window and returns an instance of BrowserWindowProxy class.

The features string follows the format of standard browser, but each feature has to be a field of BrowserWindow’s options.

**Notes:**

* Node integration will always be disabled in the opened window if it is disabled on the parent window.
* Context isolation will always be enabled in the opened window if it is enabled on the parent window.
* JavaScript will always be disabled in the opened window if it is disabled on the parent window.
* Non-standard features (that are not handled by Chromium or Electron) given in features will be passed to any registered webContent’s new-window event handler in the additionalFeatures argument.

window.opener.postMessage(message, targetOrigin)

* message String
* targetOrigin String

Sends a message to the parent window with the specified origin or \* for no origin preference.

Using Chrome’s window.open() implementation

If you want to use Chrome’s built-in window.open() implementation, set nativeWindowOpen to true in the webPreferences options object.

Native window.open() allows synchronous access to opened windows so it is convenient choice if you need to open a dialog or a preferences window.

This option can also be set on <webview> tags as well:

<webview webpreferences="nativeWindowOpen=yes"></webview>

The creation of the BrowserWindow is customizable via WebContents’s new-window event.

*// main process*

**const** mainWindow = **new** BrowserWindow({

width: 800,

height: 600,

webPreferences: {

nativeWindowOpen: true

}

})

mainWindow.webContents.on('new-window', (event, url, frameName, disposition, options, additionalFeatures) => {

**if** (frameName === 'modal') {

*// open window as modal*

event.preventDefault()

Object.assign(options, {

modal: true,

parent: mainWindow,

width: 100,

height: 100

})

event.newGuest = **new** BrowserWindow(options)

}

})

*// renderer process (mainWindow)*

**let** modal = window.open('', 'modal')

modal.document.write('<h1>Hello</h1>')

Technical Differences Between Electron and NW.js (formerly node-webkit)

**Note: Electron was previously named Atom Shell.**

Like NW.js, Electron provides a platform to write desktop applications with JavaScript and HTML and has Node integration to grant access to the low level system from web pages.

But there are also fundamental differences between the two projects that make Electron a completely separate product from NW.js:

**1. Entry of Application**

In NW.js the main entry point of an application is a web page. You specify a main page URL in the package.jsonand it is opened in a browser window as the application’s main window.

In Electron, the entry point is a JavaScript script. Instead of providing a URL directly, you manually create a browser window and load an HTML file using the API. You also need to listen to window events to decide when to quit the application.

Electron works more like the Node.js runtime. Electron’s APIs are lower level so you can use it for browser testing in place of [PhantomJS](http://phantomjs.org/).

**2. Build System**

In order to avoid the complexity of building all of Chromium, Electron uses [libchromiumcontent](https://github.com/electron/libchromiumcontent) to access Chromium’s Content API. libchromiumcontent is a single shared library that includes the Chromium Content module and all of its dependencies. Users don’t need a powerful machine to build Electron.

**3. Node Integration**

In NW.js, the Node integration in web pages requires patching Chromium to work, while in Electron we chose a different way to integrate the libuv loop with each platform’s message loop to avoid hacking Chromium. See the [node\_bindings](https://github.com/electron/electron/tree/master/atom/common) code for how that was done.

**4. Multi-context**

If you are an experienced NW.js user, you should be familiar with the concept of Node context and web context. These concepts were invented because of how NW.js was implemented.

By using the [multi-context](http://strongloop.com/strongblog/whats-new-node-js-v0-12-multiple-context-execution/) feature of Node, Electron doesn’t introduce a new JavaScript context in web pages.

Note: NW.js has optionally supported multi-context since 0.13.

Build Instructions (Linux)

Follow the guidelines below for building Electron on Linux.

Prerequisites

* At least 25GB disk space and 8GB RAM.
* Python 2.7.x. Some distributions like CentOS 6.x still use Python 2.6.x so you may need to check your Python version with python -V.
* Node.js. There are various ways to install Node. You can download source code from [nodejs.org](http://nodejs.org/) and compile it. Doing so permits installing Node on your own home directory as a standard user. Or try repositories such as [NodeSource](https://nodesource.com/blog/nodejs-v012-iojs-and-the-nodesource-linux-repositories).
* [clang](https://clang.llvm.org/get_started.html) 3.4 or later.
* Development headers of GTK+ and libnotify.

On Ubuntu, install the following libraries:

$ sudo apt-get install build-essential clang libdbus-1-dev libgtk2.0-dev \

libnotify-dev libgnome-keyring-dev libgconf2-dev \

libasound2-dev libcap-dev libcups2-dev libxtst-dev \

libxss1 libnss3-dev gcc-multilib g++-multilib curl \

gperf bison

On RHEL / CentOS, install the following libraries:

$ sudo yum install clang dbus-devel gtk2-devel libnotify-devel \

libgnome-keyring-devel xorg-x11-server-utils libcap-devel \

cups-devel libXtst-devel alsa-lib-devel libXrandr-devel \

GConf2-devel nss-devel

On Fedora, install the following libraries:

$ sudo dnf install clang dbus-devel gtk2-devel libnotify-devel \

libgnome-keyring-devel xorg-x11-server-utils libcap-devel \

cups-devel libXtst-devel alsa-lib-devel libXrandr-devel \

GConf2-devel nss-devel

Other distributions may offer similar packages for installation via package managers such as pacman. Or one can compile from source code.

Getting the Code

$ git clone https://github.com/electron/electron

Bootstrapping

The bootstrap script will download all necessary build dependencies and create the build project files. You must have Python 2.7.x for the script to succeed. Downloading certain files can take a long time. Notice that we are using ninja to build Electron so there is no Makefile generated.

$ cd electron

$ ./script/bootstrap.py --verbose

Cross compilation

If you want to build for an arm target you should also install the following dependencies:

$ sudo apt-get install libc6-dev-armhf-cross linux-libc-dev-armhf-cross \

g++-arm-linux-gnueabihf

And to cross-compile for arm or ia32 targets, you should pass the --target\_arch parameter to the bootstrap.py script:

$ ./script/bootstrap.py -v --target\_arch=arm

Building

If you would like to build both Release and Debug targets:

$ ./script/build.py

This script will cause a very large Electron executable to be placed in the directory out/R. The file size is in excess of 1.3 gigabytes. This happens because the Release target binary contains debugging symbols. To reduce the file size, run the create-dist.py script:

$ ./script/create-dist.py

This will put a working distribution with much smaller file sizes in the dist directory. After running the create-dist.py script, you may want to remove the 1.3+ gigabyte binary which is still in out/R.

You can also build the Debug target only:

$ ./script/build.py -c D

After building is done, you can find the electron debug binary under out/D.

Cleaning

To clean the build files:

$ npm run clean

To clean only out and dist directories:

$ npm run clean-build

**Note:** Both clean commands require running bootstrap again before building.

Troubleshooting

Error While Loading Shared Libraries: libtinfo.so.5

Prebuilt clang will try to link to libtinfo.so.5. Depending on the host architecture, symlink to appropriate libncurses:

$ sudo ln -s /usr/lib/libncurses.so.5 /usr/lib/libtinfo.so.5

Tests

See [Build System Overview: Tests](https://electron.atom.io/docs/development/build-system-overview#tests)

Advanced topics

The default building configuration is targeted for major desktop Linux distributions. To build for a specific distribution or device, the following information may help you.

Building libchromiumcontent locally

To avoid using the prebuilt binaries of libchromiumcontent, you can build libchromiumcontent locally. To do so, follow these steps:

1. Install [depot\_tools](https://chromium.googlesource.com/chromium/src/+/master/docs/linux_build_instructions.md#Install)
2. Install [additional build dependencies](https://chromium.googlesource.com/chromium/src/+/master/docs/linux_build_instructions.md#Install-additional-build-dependencies)
3. Fetch the git submodules:

$ git submodule update --init --recursive

1. Copy the .gclient config file

$ cp vendor/libchromiumcontent/.gclient .

1. Pass the --build\_libchromiumcontent switch to bootstrap.py script:

$ ./script/bootstrap.py -v --build\_libchromiumcontent

Note that by default the shared\_library configuration is not built, so you can only build Release version of Electron if you use this mode:

$ ./script/build.py -c R

Using system clang instead of downloaded clang binaries

By default Electron is built with prebuilt [clang](https://clang.llvm.org/get_started.html) binaries provided by the Chromium project. If for some reason you want to build with the clang installed in your system, you can call bootstrap.py with --clang\_dir=<path>switch. By passing it the build script will assume the clang binaries reside in <path>/bin/.

For example if you installed clang under /user/local/bin/clang:

$ ./script/bootstrap.py -v --build\_libchromiumcontent --clang\_dir /usr/local

$ ./script/build.py -c R

Using compilers other than clang

To build Electron with compilers like g++, you first need to disable clang with --disable\_clang switch first, and then set CC and CXX environment variables to the ones you want.

For example building with GCC toolchain:

$ env CC=gcc CXX=g++ ./script/bootstrap.py -v --build\_libchromiumcontent --disable\_clang

$ ./script/build.py -c R

Environment variables

Apart from CC and CXX, you can also set following environment variables to custom the building configurations:

* CPPFLAGS
* CPPFLAGS\_host
* CFLAGS
* CFLAGS\_host
* CXXFLAGS
* CXXFLAGS\_host
* AR
* AR\_host
* CC
* CC\_host
* CXX
* CXX\_host
* LDFLAGS

The environment variables have to be set when executing the bootstrap.py script, it won’t work in the build.pyscript.

Build Instructions (macOS)

Follow the guidelines below for building Electron on macOS.

Prerequisites

* macOS >= 10.11.6
* [Xcode](https://developer.apple.com/technologies/tools/) >= 8.2.1
* [node.js](http://nodejs.org/) (external)

If you are using the Python downloaded by Homebrew, you also need to install the following Python modules:

* [pyobjc](https://pythonhosted.org/pyobjc/install.html)

macOS SDK

If you’re simply developing Electron and don’t plan to redistribute your custom Electron build, you may skip this section.

For certain features (e.g. pinch-zoom) to work properly, you must target the macOS 10.10 SDK.

Official Electron builds are built with [Xcode 8.2.1](http://adcdownload.apple.com/Developer_Tools/Xcode_8.2.1/Xcode_8.2.1.xip), which does not contain the 10.10 SDK by default. To obtain it, first download and mount the [Xcode 6.4](http://developer.apple.com/devcenter/download.action?path=/Developer_Tools/Xcode_6.4/Xcode_6.4.dmg) DMG.

Then, assuming that the Xcode 6.4 DMG has been mounted at /Volumes/Xcode and that your Xcode 8.2.1 install is at /Applications/Xcode.app, run:

cp -r /Volumes/Xcode/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.10.sdk /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/

You will also need to enable Xcode to build against the 10.10 SDK:

* Open /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Info.plist
* Set the MinimumSDKVersion to 10.10
* Save the file

Getting the Code

$ git clone https://github.com/electron/electron

Bootstrapping

The bootstrap script will download all necessary build dependencies and create the build project files. Notice that we’re using [ninja](https://ninja-build.org/) to build Electron so there is no Xcode project generated.

$ cd electron

$ ./script/bootstrap.py -v

Building

Build both Release and Debug targets:

$ ./script/build.py

You can also only build the Debug target:

$ ./script/build.py -c D

After building is done, you can find Electron.app under out/D.

32bit Support

Electron can only be built for a 64bit target on macOS and there is no plan to support 32bit macOS in the future.

Cleaning

To clean the build files:

$ npm run clean

To clean only out and dist directories:

$ npm run clean-build

**Note:** Both clean commands require running bootstrap again before building.

Tests

See [Build System Overview: Tests](https://electron.atom.io/docs/development/build-system-overview#tests)

Build Instructions (Windows)

Follow the guidelines below for building Electron on Windows.

Prerequisites

* Windows 7 / Server 2008 R2 or higher
* Visual Studio 2015 Update 3 - [download VS 2015 Community Edition for free](https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx)
* [Python 2.7](http://www.python.org/download/releases/2.7/)
* [Node.js](http://nodejs.org/download/)
* [Git](http://git-scm.com/)
* [Debugging Tools for Windows](https://msdn.microsoft.com/en-us/library/windows/hardware/ff551063.aspx) if you plan on creating a full distribution since symstore.exe is used for creating a symbol store from .pdb files.

If you don’t currently have a Windows installation, [dev.microsoftedge.com](https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/) has timebombed versions of Windows that you can use to build Electron.

Building Electron is done entirely with command-line scripts and cannot be done with Visual Studio. You can develop Electron with any editor but support for building with Visual Studio will come in the future.

**Note:** Even though Visual Studio is not used for building, it’s still **required** because we need the build toolchains it provides.

Getting the Code

$ git clone https://github.com/electron/electron.git

Bootstrapping

The bootstrap script will download all necessary build dependencies and create the build project files. Notice that we’re using ninja to build Electron so there is no Visual Studio project generated.

$ cd electron

$ python script\bootstrap.py -v

Building

Build both Release and Debug targets:

$ python script\build.py

You can also only build the Debug target:

$ python script\build.py -c D

After building is done, you can find electron.exe under out\D (debug target) or under out\R (release target).

32bit Build

To build for the 32bit target, you need to pass --target\_arch=ia32 when running the bootstrap script:

$ python script\bootstrap.py -v --target\_arch=ia32

The other building steps are exactly the same.

Visual Studio project

To generate a Visual Studio project, you can pass the --msvs parameter:

$ python script\bootstrap.py --msvs

Cleaning

To clean the build files:

$ npm run clean

To clean only out and dist directories:

$ npm run clean-build

**Note:** Both clean commands require running bootstrap again before building.

Tests

See [Build System Overview: Tests](https://electron.atom.io/docs/development/build-system-overview#tests)

Troubleshooting

Command xxxx not found

If you encountered an error like Command xxxx not found, you may try to use the VS2015 Command Prompt console to execute the build scripts.

Fatal internal compiler error: C1001

Make sure you have the latest Visual Studio update installed.

Assertion failed: ((handle))->activecnt >= 0

If building under Cygwin, you may see bootstrap.py failed with following error:

Assertion failed: ((handle))->activecnt >= 0, file src\win\pipe.c, line 1430

Traceback (most recent call last):

File "script/bootstrap.py", line 87, in <module>

sys.exit(main())

File "script/bootstrap.py", line 22, in main

update\_node\_modules('.')

File "script/bootstrap.py", line 56, in update\_node\_modules

execute([NPM, 'install'])

File "/home/zcbenz/codes/raven/script/lib/util.py", line 118, in execute

raise e

subprocess.CalledProcessError: Command '['npm.cmd', 'install']' returned non-zero exit status 3

This is caused by a bug when using Cygwin Python and Win32 Node together. The solution is to use the Win32 Python to execute the bootstrap script (assuming you have installed Python under C:\Python27):

$ /cygdrive/c/Python27/python.exe script/bootstrap.py

LNK1181: cannot open input file ‘kernel32.lib’

Try reinstalling 32bit Node.js.

Error: ENOENT, stat ‘C:\Users\USERNAME\AppData\Roaming\npm’

Simply making that directory [should fix the problem](http://stackoverflow.com/a/25095327/102704):

$ mkdir ~\AppData\Roaming\npm

node-gyp is not recognized as an internal or external command

You may get this error if you are using Git Bash for building, you should use PowerShell or VS2015 Command Prompt instead.

Build System Overview

Electron uses [gyp](https://gyp.gsrc.io/) for project generation and [ninja](https://ninja-build.org/) for building. Project configurations can be found in the .gypand .gypi files.

Gyp Files

Following gyp files contain the main rules for building Electron:

* electron.gyp defines how Electron itself is built.
* common.gypi adjusts the build configurations of Node to make it build together with Chromium.
* brightray/brightray.gyp defines how brightray is built and includes the default configurations for linking with Chromium.
* brightray/brightray.gypi includes general build configurations about building.

Component Build

Since Chromium is quite a large project, the final linking stage can take quite a few minutes, which makes it hard for development. In order to solve this, Chromium introduced the “component build”, which builds each component as a separate shared library, making linking very quick but sacrificing file size and performance.

In Electron we took a very similar approach: for Debug builds, the binary will be linked to a shared library version of Chromium’s components to achieve fast linking time; for Release builds, the binary will be linked to the static library versions, so we can have the best possible binary size and performance.

Minimal Bootstrapping

All of Chromium’s prebuilt binaries (libchromiumcontent) are downloaded when running the bootstrap script. By default both static libraries and shared libraries will be downloaded and the final size should be between 800MB and 2GB depending on the platform.

By default, libchromiumcontent is downloaded from Amazon Web Services. If the LIBCHROMIUMCONTENT\_MIRRORenvironment variable is set, the bootstrap script will download from it. [libchromiumcontent-qiniu-mirror](https://github.com/hokein/libchromiumcontent-qiniu-mirror) is a mirror for libchromiumcontent. If you have trouble in accessing AWS, you can switch the download address to it via export LIBCHROMIUMCONTENT\_MIRROR=http://7xk3d2.dl1.z0.glb.clouddn.com/

If you only want to build Electron quickly for testing or development, you can download just the shared library versions by passing the --dev parameter:

$ ./script/bootstrap.py --dev

$ ./script/build.py -c D

Two-Phase Project Generation

Electron links with different sets of libraries in Release and Debug builds. gyp, however, doesn’t support configuring different link settings for different configurations.

To work around this Electron uses a gyp variable libchromiumcontent\_component to control which link settings to use and only generates one target when running gyp.

Target Names

Unlike most projects that use Release and Debug as target names, Electron uses R and D instead. This is because gyp randomly crashes if there is only one Release or Debug build configuration defined, and Electron only has to generate one target at a time as stated above.

This only affects developers, if you are just building Electron for rebranding you are not affected.

Tests

Test your changes conform to the project coding style using:

$ npm run lint

Test functionality using:

$ npm test

Whenever you make changes to Electron source code, you’ll need to re-run the build before the tests:

$ npm run build && npm test

You can make the test suite run faster by isolating the specific test or block you’re currently working on using Mocha’s [exclusive tests](https://mochajs.org/#exclusive-tests) feature. Just append .only to any describe or it function call:

describe.only('some feature', **function** () {

*// ... only tests in this block will be run*

})

Alternatively, you can use mocha’s grep option to only run tests matching the given regular expression pattern:

$ npm test -- --grep child\_process

Tests that include native modules (e.g. runas) can’t be executed with the debug build (see [#2558](https://github.com/electron/electron/issues/2558) for details), but they will work with the release build.

To run the tests with the release build use:

$ npm test -- -R

Chromium Development

A collection of resources for learning about Chromium and tracking its development

* [chromiumdev](https://chromiumdev-slack.herokuapp.com/) on Slack
* [@ChromiumDev](https://twitter.com/ChromiumDev) on Twitter
* [@googlechrome](https://twitter.com/googlechrome) on Twitter
* [Blog](https://blog.chromium.org/)
* [Code Search](https://cs.chromium.org/)
* [Source Code](https://cs.chromium.org/chromium/src/)
* [Development Calendar and Release Info](https://www.chromium.org/developers/calendar)
* [Discussion Groups](http://www.chromium.org/developers/discussion-groups)

See also [V8 Development](https://electron.atom.io/docs/development/v8-development)

Using clang-format on C++ Code

[clang-format](http://clang.llvm.org/docs/ClangFormat.html) is a tool to automatically format C/C++/Objective-C code, so that developers don’t need to worry about style issues during code reviews.

It is highly recommended to format your changed C++ code before opening pull requests, which will save you and the reviewers’ time.

You can install clang-format and git-clang-format via npm install -g clang-format.

To automatically format a file according to Electron C++ code style, simply run clang-format -i path/to/electron/file.cc. It should work on macOS/Linux/Windows.

The workflow to format your changed code:

1. Make codes changes in Electron repository.
2. Run git add your\_changed\_file.cc.
3. Run git-clang-format, and you will probably see modifications in your\_changed\_file.cc, these modifications are generated from clang-format.
4. Run git add your\_changed\_file.cc, and commit your change.
5. Now the branch is ready to be opened as a pull request.

If you want to format the changed code on your latest git commit (HEAD), you can run git-clang-format HEAD~1. See git-clang-format -h for more details.

Editor Integration

You can also integrate clang-format directly into your favorite editors. For further guidance on setting up editor integration, see these pages:

* [Atom](https://atom.io/packages/clang-format)
* [Vim & Emacs](http://clang.llvm.org/docs/ClangFormat.html#vim-integration)
* [Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=xaver.clang-format)

Coding Style

These are the style guidelines for coding in Electron.

You can run npm run lint to show any style issues detected by cpplint and eslint.

C++ and Python

For C++ and Python, we follow Chromium’s [Coding Style](http://www.chromium.org/developers/coding-style). You can use [clang-format](https://electron.atom.io/docs/development/clang-format) to format the C++ code automatically. There is also a script script/cpplint.py to check whether all files conform.

The Python version we are using now is Python 2.7.

The C++ code uses a lot of Chromium’s abstractions and types, so it’s recommended to get acquainted with them. A good place to start is Chromium’s [Important Abstractions and Data Structures](https://www.chromium.org/developers/coding-style/important-abstractions-and-data-structures) document. The document mentions some special types, scoped types (that automatically release their memory when going out of scope), logging mechanisms etc.

JavaScript

* Write [standard](http://npm.im/standard) JavaScript style.
* File names should be concatenated with - instead of \_, e.g. file-name.js rather than file\_name.js, because in [github/atom](https://github.com/github/atom) module names are usually in the module-name form. This rule only applies to .jsfiles.
* Use newer ES6/ES2015 syntax where appropriate
  + [const](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const) for requires and other constants
  + [let](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let) for defining variables
  + [Arrow functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions) instead of function () { }
  + [Template literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals) instead of string concatenation using +

Naming Things

Electron APIs uses the same capitalization scheme as Node.js:

* When the module itself is a class like BrowserWindow, use CamelCase.
* When the module is a set of APIs, like globalShortcut, use mixedCase.
* When the API is a property of object, and it is complex enough to be in a separate chapter like win.webContents, use mixedCase.
* For other non-module APIs, use natural titles, like <webview> Tag or Process Object.

When creating a new API, it is preferred to use getters and setters instead of jQuery’s one-function style. For example, .getText() and .setText(text) are preferred to .text([text]). There is a [discussion](https://github.com/electron/electron/issues/46) on this.

Debugging on Windows

If you experience crashes or issues in Electron that you believe are not caused by your JavaScript application, but instead by Electron itself, debugging can be a little bit tricky, especially for developers not used to native/C++ debugging. However, using Visual Studio, GitHub’s hosted Electron Symbol Server, and the Electron source code, it is fairly easy to enable step-through debugging with breakpoints inside Electron’s source code.

Requirements

* **A debug build of Electron**: The easiest way is usually building it yourself, using the tools and prerequisites listed in the [build instructions for Windows](https://electron.atom.io/docs/development/build-instructions-windows). While you can easily attach to and debug Electron as you can download it directly, you will find that it is heavily optimized, making debugging substantially more difficult: The debugger will not be able to show you the content of all variables and the execution path can seem strange because of inlining, tail calls, and other compiler optimizations.
* **Visual Studio with C++ Tools**: The free community editions of Visual Studio 2013 and Visual Studio 2015 both work. Once installed, [configure Visual Studio to use GitHub’s Electron Symbol server](https://electron.atom.io/docs/development/setting-up-symbol-server). It will enable Visual Studio to gain a better understanding of what happens inside Electron, making it easier to present variables in a human-readable format.
* **ProcMon**: The [free SysInternals tool](https://technet.microsoft.com/en-us/sysinternals/processmonitor.aspx) allows you to inspect a processes parameters, file handles, and registry operations.

Attaching to and Debugging Electron

To start a debugging session, open up PowerShell/CMD and execute your debug build of Electron, using the application to open as a parameter.

$ ./out/D/electron.exe ~/my-electron-app/

Setting Breakpoints

Then, open up Visual Studio. Electron is not built with Visual Studio and hence does not contain a project file - you can however open up the source code files “As File”, meaning that Visual Studio will open them up by themselves. You can still set breakpoints - Visual Studio will automatically figure out that the source code matches the code running in the attached process and break accordingly.

Relevant code files can be found in ./atom/ as well as in Brightray, found in ./brightray/browser and ./brightray/common. If you’re hardcore, you can also debug Chromium directly, which is obviously found in chromium\_src.

Attaching

You can attach the Visual Studio debugger to a running process on a local or remote computer. After the process is running, click Debug / Attach to Process (or press CTRL+ALT+P) to open the “Attach to Process” dialog box. You can use this capability to debug apps that are running on a local or remote computer, debug multiple processes simultaneously.

If Electron is running under a different user account, select the Show processes from all users check box. Notice that depending on how many BrowserWindows your app opened, you will see multiple processes. A typical one-window app will result in Visual Studio presenting you with two Electron.exe entries - one for the main process and one for the renderer process. Since the list only gives you names, there’s currently no reliable way of figuring out which is which.

Which Process Should I Attach to?

Code executed within the main process (that is, code found in or eventually run by your main JavaScript file) as well as code called using the remote (require('electron').remote) will run inside the main process, while other code will execute inside its respective renderer process.

You can be attached to multiple programs when you are debugging, but only one program is active in the debugger at any time. You can set the active program in the Debug Location toolbar or the Processes window.

Using ProcMon to Observe a Process

While Visual Studio is fantastic for inspecting specific code paths, ProcMon’s strength is really in observing everything your application is doing with the operating system - it captures File, Registry, Network, Process, and Profiling details of processes. It attempts to log **all** events occurring and can be quite overwhelming, but if you seek to understand what and how your application is doing to the operating system, it can be a valuable resource.

For an introduction to ProcMon’s basic and advanced debugging features, go check out [this video tutorial](https://channel9.msdn.com/shows/defrag-tools/defrag-tools-4-process-monitor)provided by Microsoft.

Debugging on macOS

If you experience crashes or issues in Electron that you believe are not caused by your JavaScript application, but instead by Electron itself, debugging can be a little bit tricky, especially for developers not used to native/C++ debugging. However, using lldb, and the Electron source code, it is fairly easy to enable step-through debugging with breakpoints inside Electron’s source code.

Requirements

* **A debug build of Electron**: The easiest way is usually building it yourself, using the tools and prerequisites listed in the [build instructions for macOS](https://electron.atom.io/docs/development/build-instructions-osx). While you can easily attach to and debug Electron as you can download it directly, you will find that it is heavily optimized, making debugging substantially more difficult: The debugger will not be able to show you the content of all variables and the execution path can seem strange because of inlining, tail calls, and other compiler optimizations.
* **Xcode**: In addition to Xcode, also install the Xcode command line tools. They include LLDB, the default debugger in Xcode on Mac OS X. It supports debugging C, Objective-C and C++ on the desktop and iOS devices and simulator.

Attaching to and Debugging Electron

To start a debugging session, open up Terminal and start lldb, passing a debug build of Electron as a parameter.

$ lldb ./out/D/Electron.app

(lldb) target create "./out/D/Electron.app"

Current executable set to './out/D/Electron.app' (x86\_64).

Setting Breakpoints

LLDB is a powerful tool and supports multiple strategies for code inspection. For this basic introduction, let’s assume that you’re calling a command from JavaScript that isn’t behaving correctly - so you’d like to break on that command’s C++ counterpart inside the Electron source.

Relevant code files can be found in ./atom/ as well as in Brightray, found in ./brightray/browser and ./brightray/common. If you’re hardcore, you can also debug Chromium directly, which is obviously found in chromium\_src.

Let’s assume that you want to debug app.setName(), which is defined in browser.cc as Browser::SetName(). Set the breakpoint using the breakpoint command, specifying file and line to break on:

(lldb) breakpoint set --file browser.cc --line 117

Breakpoint 1: where = Electron Framework`atom::Browser::SetName(std::\_\_1::basic\_string<char, std::\_\_1::char\_traits<char>, std::\_\_1::allocator<char> > const&) + 20 at browser.cc:118, address = 0x000000000015fdb4

Then, start Electron:

(lldb) run

The app will immediately be paused, since Electron sets the app’s name on launch:

(lldb) run

Process 25244 launched: '/Users/fr/Code/electron/out/D/Electron.app/Contents/MacOS/Electron' (x86\_64)

Process 25244 stopped

**\*** thread *#1: tid = 0x839a4c, 0x0000000100162db4 Electron Framework`atom::Browser::SetName(this=0x0000000108b14f20, name="Electron") + 20 at browser.cc:118, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1*

frame *#0: 0x0000000100162db4 Electron Framework`atom::Browser::SetName(this=0x0000000108b14f20, name="Electron") + 20 at browser.cc:118*

115 }

116

117 void Browser::SetName(const std::string& name) {

-> 118 name\_override\_ = name;

119 }

120

121 int Browser::GetBadgeCount() {

(lldb)

To show the arguments and local variables for the current frame, run frame variable (or fr v), which will show you that the app is currently setting the name to “Electron”.

(lldb) frame variable

(atom::Browser **\***) this = 0x0000000108b14f20

(const string &) name = "Electron": {

[...]

}

To do a source level single step in the currently selected thread, execute step (or s). This would take you into name\_override\_.empty(). To proceed and do a step over, run next (or n).

(lldb) step

Process 25244 stopped

**\*** thread *#1: tid = 0x839a4c, 0x0000000100162dcc Electron Framework`atom::Browser::SetName(this=0x0000000108b14f20, name="Electron") + 44 at browser.cc:119, queue = 'com.apple.main-thread', stop reason = step in*

frame *#0: 0x0000000100162dcc Electron Framework`atom::Browser::SetName(this=0x0000000108b14f20, name="Electron") + 44 at browser.cc:119*

116

117 void Browser::SetName(const std::string& name) {

118 name\_override\_ = name;

-> 119 }

120

121 int Browser::GetBadgeCount() {

122 **return** badge\_count\_;

To finish debugging at this point, run process continue. You can also continue until a certain line is hit in this thread (thread until 100). This command will run the thread in the current frame till it reaches line 100 in this frame or stops if it leaves the current frame.

Now, if you open up Electron’s developer tools and call setName, you will once again hit the breakpoint.

Further Reading

LLDB is a powerful tool with a great documentation. To learn more about it, consider Apple’s debugging documentation, for instance the [LLDB Command Structure Reference](https://developer.apple.com/library/mac/documentation/IDEs/Conceptual/gdb_to_lldb_transition_guide/document/lldb-basics.html#//apple_ref/doc/uid/TP40012917-CH2-SW2) or the introduction to [Using LLDB as a Standalone Debugger](https://developer.apple.com/library/mac/documentation/IDEs/Conceptual/gdb_to_lldb_transition_guide/document/lldb-terminal-workflow-tutorial.html).

You can also check out LLDB’s fantastic [manual and tutorial](http://lldb.llvm.org/tutorial.html), which will explain more complex debugging scenarios.

Releasing

This document describes the process for releasing a new version of Electron.

Create a temporary branch

Create a new branch from master. Name it release or anything you like.

Note: If you are creating a backport release, you’ll check out 1-6-x, 1-7-x, etc instead of master.

git checkout master

git pull

git checkout -b release

This branch is created as a precaution to prevent any merged PRs from sneaking into a release between the time the temporary release branch is created and the CI builds are complete.

Check for extant drafts

The upload script [looks for an existing draft release](https://github.com/electron/electron/blob/7961a97d7ddbed657c6c867cc8426e02c236c077/script/upload.py#L173-L181). To prevent your new release from clobbering an existing draft, check [the releases page](https://github.com/electron/electron/releases) and make sure there are no drafts.

Bump the version

Run the bump-version script, passing major, minor, or patch as an argument:

npm run bump-version -- patch

git push origin HEAD

This will bump the version number in several files. See [this bump commit](https://github.com/electron/electron/commit/78ec1b8f89b3886b856377a1756a51617bc33f5a) for an example.

Most releases will be patch level. Upgrades to Chrome or other major changes should use minor. For more info, see [electron-versioning](https://electron.atom.io/docs/tutorial/electron-versioning).

Wait for builds 

The presence of the word [Bump](https://github.com/electron/electron/blob/7961a97d7ddbed657c6c867cc8426e02c236c077/script/cibuild-linux#L3-L6) in the commit message created by the bump-version script will [trigger the release process](https://github.com/electron/electron/blob/7961a97d7ddbed657c6c867cc8426e02c236c077/script/cibuild#L82-L96).

To monitor the build progress, see the following pages:

* [208.52.191.140:8080/view/All/builds](http://208.52.191.140:8080/view/All/builds) for Mac and Windows
* [jenkins.githubapp.com/label/chromium/](https://jenkins.githubapp.com/label/chromium/) for Linux

Compile release notes

Writing release notes is a good way to keep yourself busy while the builds are running. For prior art, see existing releases on [the releases page](https://github.com/electron/electron/releases).

Tips:

* Each listed item should reference a PR on electron/electron, not an issue, nor a PR from another repo like libcc.
* No need to use link markup when referencing PRs. Strings like #123 will automatically be converted to links on github.com.
* To see the version of Chromium, V8, and Node in every version of Electron, visit [atom.io/download/electron/index.json](https://atom.io/download/electron/index.json).

Patch releases

For a patch release, use the following format:

## Bug Fixes

\* Fixed a cross-platform thing. #123

### Linux

\* Fixed a Linux thing. #123

### macOS

\* Fixed a macOS thing. #123

### Windows

\* Fixed a Windows thing. #1234

## API Changes

\* Changed a thing. #123

### Linux

\* Changed a Linux thing. #123

### macOS

\* Changed a macOS thing. #123

### Windows

\* Changed a Windows thing. #123

Minor releases

For a minor release (which is normally a Chromium update, and possibly also a Node update), e.g. 1.8.0, use this format:

\*\*Note:\*\* This is a beta release. This is the first release running on upgraded versions of Chrome/Node.js/V8 and most likely will have have some instability and/or regressions.

Please file new issues for any bugs you find in it.

This release is published to [npm](https://www.npmjs.com/package/electron) under the `beta` tag and can be installed via `npm install electron@beta`.

## Upgrades

- Upgraded from Chrome `oldVersion` to `newVersion`. #123

- Upgraded from Node `oldVersion` to `newVersion`. #123

- Upgraded from v8 `oldVersion` to `newVersion`. #9116

## Other Changes

- Some other change. #123

Edit the release draft

1. Visit [the releases page](https://github.com/electron/electron/releases) and you’ll see a new draft release with placeholder release notes.
2. Edit the release and add release notes.
3. Ensure the prerelease checkbox is checked. This should happen automatically for Electron versions >=1.7
4. Click ‘Save draft’. **Do not click ‘Publish release’!**
5. Wait for all builds to pass before proceeding.

Merge temporary branch

Merge the temporary back into master, without creating a merge commit:

git merge release master --no-commit

git push origin master

If this fails, rebase with master and rebuild:

git pull

git checkout release

git rebase master

git push origin HEAD

Run local debug build

Run local debug build to verify that you are actually building the version you want. Sometimes you thought you were doing a release for a new version, but you’re actually not.

npm run build

npm start

Verify the window is displaying the current updated version.

Set environment variables

You’ll need to set the following environment variables to publish a release. Ask another team member for these credentials.

* ELECTRON\_S3\_BUCKET
* ELECTRON\_S3\_ACCESS\_KEY
* ELECTRON\_S3\_SECRET\_KEY
* ELECTRON\_GITHUB\_TOKEN - A personal access token with “repo” scope.

You will only need to do this once.

Publish the release

This script will download the binaries and generate the node headers and the .lib linker used on Windows by node-gyp to build native modules.

npm run release

Note: Many distributions of Python still ship with old HTTPS certificates. You may see a InsecureRequestWarning, but it can be disregarded.

Delete the temporary branch

git checkout master

git branch -D release *# delete local branch*

git push origin :release *# delete remote branch*

Promoting a release on npm

New releases are published to npm with the beta tag. Every release should eventually get promoted to stable unless there’s a good reason not to.

Releases are normally given around two weeks in the wild before being promoted. Before promoting a release, check to see if there are any bug reports against that version, e.g. issues labeled with version/1.7.x.

It’s also good to ask users in Slack if they’re using the beta versions successfully.

To see what’s beta and stable at any given time:

$ npm dist-tag ls electron

beta: 1.7.5

latest: 1.6.11

To promote a beta version to stable (aka latest):

npm dist-tag add electron@1.2.3 latest

Setting Up Symbol Server in Debugger

Debug symbols allow you to have better debugging sessions. They have information about the functions contained in executables and dynamic libraries and provide you with information to get clean call stacks. A Symbol Server allows the debugger to load the correct symbols, binaries and sources automatically without forcing users to download large debugging files. The server functions like [Microsoft’s symbol server](http://support.microsoft.com/kb/311503) so the documentation there can be useful.

Note that because released Electron builds are heavily optimized, debugging is not always easy. The debugger will not be able to show you the content of all variables and the execution path can seem strange because of inlining, tail calls, and other compiler optimizations. The only workaround is to build an unoptimized local build.

The official symbol server URL for Electron is https://electron-symbols.githubapp.com. You cannot visit this URL directly, you must add it to the symbol path of your debugging tool. In the examples below, a local cache directory is used to avoid repeatedly fetching the PDB from the server. Replace c:\code\symbols with an appropriate cache directory on your machine.

Using the Symbol Server in Windbg

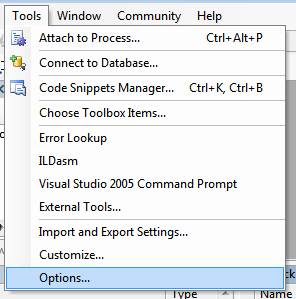
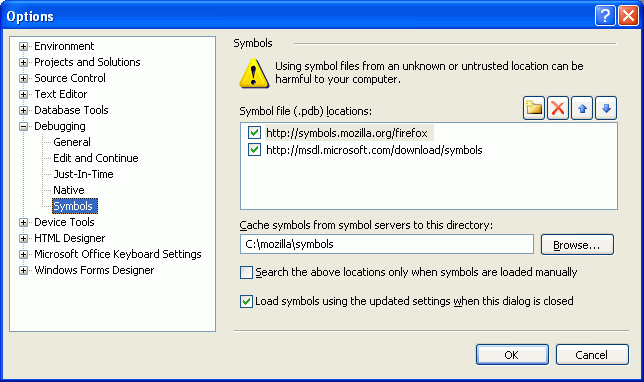
The Windbg symbol path is configured with a string value delimited with asterisk characters. To use only the Electron symbol server, add the following entry to your symbol path (**Note:** you can replace c:\code\symbolswith any writable directory on your computer, if you’d prefer a different location for downloaded symbols):

SRV\*c:\code\symbols\\*https://electron-symbols.githubapp.com

Set this string as \_NT\_SYMBOL\_PATH in the environment, using the Windbg menus, or by typing the .sympathcommand. If you would like to get symbols from Microsoft’s symbol server as well, you should list that first:

SRV\*c:\code\symbols\\*http://msdl.microsoft.com/download/symbols;SRV\*c:\code\symbols\\*https://electron-symbols.githubapp.com

Using the symbol server in Visual Studio

Troubleshooting: Symbols will not load

Type the following commands in Windbg to print why symbols are not loading:

> !sym noisy

> .reload /f electron.exe

Source Code Directory Structure

The source code of Electron is separated into a few parts, mostly following Chromium on the separation conventions.

You may need to become familiar with [Chromium’s multi-process architecture](http://dev.chromium.org/developers/design-documents/multi-process-architecture) to understand the source code better.

Structure of Source Code

Electron

├── atom/ - C++ source code.

| ├── app/ - System entry code.

| ├── browser/ - The frontend including the main window, UI, and all of the

| | main process things. This talks to the renderer to manage web pages.

| | ├── ui/ - Implementation of UI stuff for different platforms.

| | | ├── cocoa/ - Cocoa specific source code.

| | | ├── win/ - Windows GUI specific source code.

| | | └── x/ - X11 specific source code.

| | ├── api/ - The implementation of the main process APIs.

| | ├── net/ - Network related code.

| | ├── mac/ - Mac specific Objective-C source code.

| | └── resources/ - Icons, platform-dependent files, etc.

| ├── renderer/ - Code that runs in renderer process.

| | └── api/ - The implementation of renderer process APIs.

| └── common/ - Code that used by both the main and renderer processes,

| including some utility functions and code to integrate node's message

| loop into Chromium's message loop.

| └── api/ - The implementation of common APIs, and foundations of

| Electron's built-in modules.

├── chromium\_src/ - Source code that copied from Chromium.

├── default\_app/ - The default page to show when Electron is started without

| providing an app.

├── docs/ - Documentations.

├── lib/ - JavaScript source code.

| ├── browser/ - Javascript main process initialization code.

| | └── api/ - Javascript API implementation.

| ├── common/ - JavaScript used by both the main and renderer processes

| | └── api/ - Javascript API implementation.

| └── renderer/ - Javascript renderer process initialization code.

| └── api/ - Javascript API implementation.

├── spec/ - Automatic tests.

├── electron.gyp - Building rules of Electron.

└── common.gypi - Compiler specific settings and building rules for other

components like `node` and `breakpad`.

Structure of Other Directories

* **script** - Scripts used for development purpose like building, packaging, testing, etc.
* **tools** - Helper scripts used by gyp files, unlike script, scripts put here should never be invoked by users directly.
* **vendor** - Source code of third party dependencies, we didn’t use third\_party as name because it would confuse it with the same directory in Chromium’s source code tree.
* **node\_modules** - Third party node modules used for building.
* **out** - Temporary output directory of ninja.
* **dist** - Temporary directory created by script/create-dist.py script when creating a distribution.
* **external\_binaries** - Downloaded binaries of third-party frameworks which do not support building with gyp.

Keeping Git Submodules Up to Date

The Electron repository has a few vendored dependencies, found in the [/vendor](https://github.com/electron/electron/tree/master/vendor) directory. Occasionally you might see a message like this when running git status:

$ git status

modified: vendor/libchromiumcontent (new commits)

modified: vendor/node (new commits)

To update these vendored dependencies, run the following command:

git submodule update --init --recursive

If you find yourself running this command often, you can create an alias for it in your ~/.gitconfig file:

[alias]

su = submodule update --init --recursive

Upgrading Chrome Checklist

This document is meant to serve as an overview of what steps are needed on each Chrome upgrade in Electron.

These are things to do in addition to updating the Electron code for any Chrome/Node API changes.

* Verify the new Chrome version is available from https://github.com/zcbenz/chromium-source-tarball/releases
* Update the VERSION file at the root of the electron/libchromiumcontent repository
* Update the CLANG\_REVISION in script/update-clang.sh to match the version Chrome is using in libchromiumcontent/src/tools/clang/scripts/update.py
* Upgrade vendor/node to the Node release that corresponds to the v8 version being used in the new Chrome release. See the v8 versions in Node on https://nodejs.org/en/download/releases for more details
* Upgrade vendor/crashpad for any crash reporter changes needed
* Upgrade vendor/depot\_tools for any build tools changes needed
* Update the libchromiumcontent SHA-1 to download in script/lib/config.py
* Open a pull request on electron/libchromiumcontent with the changes
* Open a pull request on electron/electron with the changes
  + This should include upgrading the submodules in vendor/ as needed
* Verify debug builds succeed on:
  + macOS
  + 32-bit Windows
  + 64-bit Window
  + 32-bit Linux
  + 64-bit Linux
  + ARM Linux
* Verify release builds succeed on:
  + macOS
  + 32-bit Windows
  + 64-bit Window
  + 32-bit Linux
  + 64-bit Linux
  + ARM Linux
* Verify tests pass on:
  + macOS
  + 32-bit Windows
  + 64-bit Window
  + 32-bit Linux
  + 64-bit Linux
  + ARM Linux

Verify ffmpeg Support

Electron ships with a version of ffmpeg that includes proprietary codecs by default. A version without these codecs is built and distributed with each release as well. Each Chrome upgrade should verify that switching this version is still supported.

You can verify Electron’s support for multiple ffmpeg builds by loading the following page. It should work with the default ffmpeg library distributed with Electron and not work with the ffmpeg library built without proprietary codecs.

***<!DOCTYPE html>***

<html>

<head>

<meta charset="utf-8">

<title>Proprietary Codec Check</title>

</head>

<body>

<p>Checking if Electron is using proprietary codecs by loading video from http://www.quirksmode.org/html5/videos/big\_buck\_bunny.mp4</p>

<p id="outcome"></p>

<video style="display:none" src="http://www.quirksmode.org/html5/videos/big\_buck\_bunny.mp4" autoplay></video>

<script>

**const** video = document.querySelector('video')

video.addEventListener('error', ({target}) => {

**if** (target.error.code === target.error.MEDIA\_ERR\_SRC\_NOT\_SUPPORTED) {

document.querySelector('#outcome').textContent = 'Not using proprietary codecs, video emitted source not supported error event.'

} **else** {

document.querySelector('#outcome').textContent = `Unexpected error: ${target.error.code}`

}

})

video.addEventListener('playing', () => {

document.querySelector('#outcome').textContent = 'Using proprietary codecs, video started playing.'

})

</script>

</body>

</html>

Links

* [Chrome Release Schedule](https://www.chromium.org/developers/calendar)

V8 Development

A collection of resources for learning and using V8

* [V8 Tracing](https://github.com/v8/v8/wiki/Tracing-V8)
* [V8 Profiler](https://github.com/v8/v8/wiki/V8-Profiler) - Profiler combinations which are useful for profiling: --prof, --trace-ic, --trace-opt, --trace-deopt, --print-bytecode, --print-opt-code
* [V8 Interpreter Design](https://docs.google.com/document/d/11T2CRex9hXxoJwbYqVQ32yIPMh0uouUZLdyrtmMoL44/edit?ts=56f27d9d#heading=h.6jz9dj3bnr8t)
* [Optimizing compiler](https://github.com/v8/v8/wiki/TurboFan)
* [V8 GDB Debugging](https://github.com/v8/v8/wiki/GDB-JIT-Interface)

See also [Chromium Development](https://electron.atom.io/docs/development/chromium-development)

Electron FAQ

Why am I having trouble installing Electron?

When running npm install electron, some users occasionally encounter installation errors.

In almost all cases, these errors are the result of network problems and not actual issues with the electron npm package. Errors like ELIFECYCLE, EAI\_AGAIN, ECONNRESET, and ETIMEDOUT are all indications of such network problems. The best resolution is to try switching networks, or just wait a bit and try installing again.

You can also attempt to download Electron directly from [electron/electron/releases](https://github.com/electron/electron/releases) if installing via npm is failing.

When will Electron upgrade to latest Chrome?

The Chrome version of Electron is usually bumped within one or two weeks after a new stable Chrome version gets released. This estimate is not guaranteed and depends on the amount of work involved with upgrading.

Only the stable channel of Chrome is used. If an important fix is in beta or dev channel, we will back-port it.

For more information, please see the [security introduction](https://electron.atom.io/docs/tutorial/security).

When will Electron upgrade to latest Node.js?

When a new version of Node.js gets released, we usually wait for about a month before upgrading the one in Electron. So we can avoid getting affected by bugs introduced in new Node.js versions, which happens very often.

New features of Node.js are usually brought by V8 upgrades, since Electron is using the V8 shipped by Chrome browser, the shiny new JavaScript feature of a new Node.js version is usually already in Electron.

How to share data between web pages?

To share data between web pages (the renderer processes) the simplest way is to use HTML5 APIs which are already available in browsers. Good candidates are [Storage API](https://developer.mozilla.org/en-US/docs/Web/API/Storage), [localStorage](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage), [sessionStorage](https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage), and [IndexedDB](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API).

Or you can use the IPC system, which is specific to Electron, to store objects in the main process as a global variable, and then to access them from the renderers through the remote property of electron module:

*// In the main process.*

global.sharedObject = {

someProperty: 'default value'

}

*// In page 1.*

require('electron').remote.getGlobal('sharedObject').someProperty = 'new value'

*// In page 2.*

console.log(require('electron').remote.getGlobal('sharedObject').someProperty)

My app’s window/tray disappeared after a few minutes.

This happens when the variable which is used to store the window/tray gets garbage collected.

If you encounter this problem, the following articles may prove helpful:

* [Memory Management](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management)
* [Variable Scope](https://msdn.microsoft.com/library/bzt2dkta(v=vs.94).aspx)

If you want a quick fix, you can make the variables global by changing your code from this:

**const** {app, Tray} = require('electron')

app.on('ready', () => {

**const** tray = **new** Tray('/path/to/icon.png')

tray.setTitle('hello world')

})

to this:

**const** {app, Tray} = require('electron')

**let** tray = null

app.on('ready', () => {

tray = **new** Tray('/path/to/icon.png')

tray.setTitle('hello world')

})

I can not use jQuery/RequireJS/Meteor/AngularJS in Electron.

Due to the Node.js integration of Electron, there are some extra symbols inserted into the DOM like module, exports, require. This causes problems for some libraries since they want to insert the symbols with the same names.

To solve this, you can turn off node integration in Electron:

*// In the main process.*

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({

webPreferences: {

nodeIntegration: false

}

})

win.show()

But if you want to keep the abilities of using Node.js and Electron APIs, you have to rename the symbols in the page before including other libraries:

<head>

<script>

window.nodeRequire = require;

**delete** window.require;

**delete** window.exports;

**delete** window.module;

</script>

<script type="text/javascript" src="jquery.js"></script>

</head>

require('electron').xxx is undefined.

When using Electron’s built-in module you might encounter an error like this:

> require('electron').webFrame.setZoomFactor(1.0)

Uncaught TypeError: Cannot read property 'setZoomLevel' of undefined

This is because you have the [npm electron module](https://www.npmjs.com/package/electron) installed either locally or globally, which overrides Electron’s built-in module.

To verify whether you are using the correct built-in module, you can print the path of the electron module:

console.log(require.resolve('electron'))

and then check if it is in the following form:

"/path/to/Electron.app/Contents/Resources/atom.asar/renderer/api/lib/exports/electron.js"

If it is something like node\_modules/electron/index.js, then you have to either remove the npm electronmodule, or rename it.

npm uninstall electron

npm uninstall -g electron

However if you are using the built-in module but still getting this error, it is very likely you are using the module in the wrong process. For example electron.app can only be used in the main process, while electron.webFrameis only available in renderer processes.

Glossary

This page defines some terminology that is commonly used in Electron development.

ASAR

ASAR stands for Atom Shell Archive Format. An [asar](https://github.com/electron/asar) archive is a simple tar-like format that concatenates files into a single file. Electron can read arbitrary files from it without unpacking the whole file.

The ASAR format was created primarily to improve performance on Windows… TODO

Brightray

Brightray [was](https://github.com/electron-archive/brightray) a static library that made [libchromiumcontent](https://electron.atom.io/docs/all/#libchromiumcontent) easier to use in applications. It is now deprecated and has been merged into Electron’s codebase.

CRT

The C Run-time Library (CRT) is the part of the C++ Standard Library that incorporates the ISO C99 standard library. The Visual C++ libraries that implement the CRT support native code development, and both mixed native and managed code, and pure managed code for .NET development.

DMG

An Apple Disk Image is a packaging format used by macOS. DMG files are commonly used for distributing application “installers”. [electron-builder](https://github.com/electron-userland/electron-builder) supports dmg as a build target.

IME

Input Method Editor. A program that allows users to enter characters and symbols not found on their keyboard. For example, this allows users of Latin keyboards to input Chinese, Japanese, Korean and Indic characters.

IPC

IPC stands for Inter-Process Communication. Electron uses IPC to send serialized JSON messages between the [main](https://electron.atom.io/docs/all/#main-process) and [renderer](https://electron.atom.io/docs/all/#renderer-process) processes.

libchromiumcontent

A shared library that includes the [Chromium Content module](https://www.chromium.org/developers/content-module) and all its dependencies (e.g., Blink, [V8](https://electron.atom.io/docs/all/#v8), etc.). Also referred to as “libcc”.

* [github.com/electron/libchromiumcontent](https://github.com/electron/libchromiumcontent)

main process

The main process, commonly a file named main.js, is the entry point to every Electron app. It controls the life of the app, from open to close. It also manages native elements such as the Menu, Menu Bar, Dock, Tray, etc. The main process is responsible for creating each new renderer process in the app. The full Node API is built in.

Every app’s main process file is specified in the main property in package.json. This is how electron . knows what file to execute at startup.

See also: [process](https://electron.atom.io/docs/all/#process), [renderer process](https://electron.atom.io/docs/all/#renderer-process)

MAS

Acronym for Apple’s Mac App Store. For details on submitting your app to the MAS, see the [Mac App Store Submission Guide](https://electron.atom.io/docs/tutorial/mac-app-store-submission-guide).

native modules

Native modules (also called [addons](https://nodejs.org/api/addons.html) in Node.js) are modules written in C or C++ that can be loaded into Node.js or Electron using the require() function, and used just as if they were an ordinary Node.js module. They are used primarily to provide an interface between JavaScript running in Node.js and C/C++ libraries.

Native Node modules are supported by Electron, but since Electron is very likely to use a different V8 version from the Node binary installed in your system, you have to manually specify the location of Electron’s headers when building native modules.

See also [Using Native Node Modules](https://electron.atom.io/docs/tutorial/using-native-node-modules).

NSIS

Nullsoft Scriptable Install System is a script-driven Installer authoring tool for Microsoft Windows. It is released under a combination of free software licenses, and is a widely-used alternative to commercial proprietary products like InstallShield. [electron-builder](https://github.com/electron-userland/electron-builder) supports NSIS as a build target.

OSR

Off-screen rendering.

process

A process is an instance of a computer program that is being executed. Electron apps that make use of the [main](https://electron.atom.io/docs/all/#main-process)and one or many [renderer](https://electron.atom.io/docs/all/#renderer-process) process are actually running several programs simultaneously.

In Node.js and Electron, each running process has a process object. This object is a global that provides information about, and control over, the current process. As a global, it is always available to applications without using require().

See also: [main process](https://electron.atom.io/docs/all/#main-process), [renderer process](https://electron.atom.io/docs/all/#renderer-process)

renderer process

The renderer process is a browser window in your app. Unlike the main process, there can be multiple of these and each is run in a separate process. They can also be hidden.

In normal browsers, web pages usually run in a sandboxed environment and are not allowed access to native resources. Electron users, however, have the power to use Node.js APIs in web pages allowing lower level operating system interactions.

See also: [process](https://electron.atom.io/docs/all/#process), [main process](https://electron.atom.io/docs/all/#main-process)

Squirrel

Squirrel is an open-source framework that enables Electron apps to update automatically as new versions are released. See the [autoUpdater](https://electron.atom.io/docs/api/auto-updater) API for info about getting started with Squirrel.

userland

This term originated in the Unix community, where “userland” or “userspace” referred to programs that run outside of the operating system kernel. More recently, the term has been popularized in the Node and npm community to distinguish between the features available in “Node core” versus packages published to the npm registry by the much larger “user” community.

Like Node, Electron is focused on having a small set of APIs that provide all the necessary primitives for developing multi-platform desktop applications. This design philosophy allows Electron to remain a flexible tool without being overly prescriptive about how it should be used. Userland enables users to create and share tools that provide additional functionality on top of what is available in “core”.

V8

V8 is Google’s open source JavaScript engine. It is written in C++ and is used in Google Chrome. V8 can run standalone, or can be embedded into any C++ application.

Electron builds V8 as part of Chromium and then points Node to that V8 when building it.

V8’s version numbers always correspond to those of Google Chrome. Chrome 59 includes V8 5.9, Chrome 58 includes V8 5.8, etc.

* [developers.google.com/v8](https://developers.google.com/v8)
* [nodejs.org/api/v8.html](https://nodejs.org/api/v8.html)
* [docs/development/v8-development.md](https://electron.atom.io/docs/development/v8-development)

webview

webview tags are used to embed ‘guest’ content (such as external web pages) in your Electron app. They are similar to iframes, but differ in that each webview runs in a separate process. It doesn’t have the same permissions as your web page and all interactions between your app and embedded content will be asynchronous. This keeps your app safe from the embedded content.

About Electron

[Electron](https://electron.atom.io/) is an open source library developed by GitHub for building cross-platform desktop applications with HTML, CSS, and JavaScript. Electron accomplishes this by combining [Chromium](https://www.chromium.org/Home) and [Node.js](https://nodejs.org/) into a single runtime and apps can be packaged for Mac, Windows, and Linux.

Electron began in 2013 as the framework on which [Atom](https://atom.io/), GitHub’s hackable text editor, would be built. The two were open sourced in the Spring of 2014.

It has since become a popular tool used by open source developers, startups, and established companies. [See who is building on Electron](https://electron.atom.io/apps/).

Read on to learn more about the contributors and releases of Electron or get started building with Electron in the [Quick Start Guide](https://electron.atom.io/docs/tutorial/quick-start).

Core Team and Contributors

Electron is maintained by a team at GitHub as well as a group of [active contributors](https://github.com/electron/electron/graphs/contributors) from the community. Some of the contributors are individuals and some work at larger companies who are developing on Electron. We’re happy to add frequent contributors to the project as maintainers. Read more about [contributing to Electron](https://github.com/electron/electron/blob/master/CONTRIBUTING.md).

Releases

[Electron releases](https://github.com/electron/electron/releases) frequently. We release when there are significant bug fixes, new APIs or are updating versions of Chromium or Node.js.

Updating Dependencies

Electron’s version of Chromium is usually updated within one or two weeks after a new stable Chromium version is released, depending on the effort involved in the upgrade.

When a new version of Node.js is released, Electron usually waits about a month before upgrading in order to bring in a more stable version.

In Electron, Node.js and Chromium share a single V8 instance—usually the version that Chromium is using. Most of the time this *just works* but sometimes it means patching Node.js.

Versioning

Due to the hard dependency on Node.js and Chromium, Electron is in a tricky versioning position and [does not follow semver](http://semver.org/). You should therefore always reference a specific version of Electron. [Read more about Electron’s versioning](https://electron.atom.io/docs/tutorial/electron-versioning/) or see the [versions currently in use](https://electron.atom.io/#electron-versions).

LTS

Long term support of older versions of Electron does not currently exist. If your current version of Electron works for you, you can stay on it for as long as you’d like. If you want to make use of new features as they come in you should upgrade to a newer version.

A major update came with version v1.0.0. If you’re not yet using this version, you should [read more about the v1.0.0 changes](https://electron.atom.io/blog/2016/05/11/electron-1-0).

Core Philosophy

In order to keep Electron small (file size) and sustainable (the spread of dependencies and APIs) the project limits the scope of the core project.

For instance, Electron uses just the rendering library from Chromium rather than all of Chromium. This makes it easier to upgrade Chromium but also means some browser features found in Google Chrome do not exist in Electron.

New features added to Electron should primarily be native APIs. If a feature can be its own Node.js module, it probably should be. See the [Electron tools built by the community](https://electron.atom.io/community).

History

Below are milestones in Electron’s history.

| **:calendar:** | **:tada:** |
| --- | --- |
| **April 2013** | [Atom Shell is started](https://github.com/electron/electron/commit/6ef8875b1e93787fa9759f602e7880f28e8e6b45). |
| **May 2014** | [Atom Shell is open sourced](http://blog.atom.io/2014/05/06/atom-is-now-open-source.html). |
| **April 2015** | [Atom Shell is re-named Electron](https://github.com/electron/electron/pull/1389). |
| **May 2016** | [Electron releases v1.0.0](https://electron.atom.io/blog/2016/05/11/electron-1-0). |
| **May 2016** | [Electron apps compatible with Mac App Store](https://electron.atom.io/docs/tutorial/mac-app-store-submission-guide). |
| **August 2016** | [Windows Store support for Electron apps](https://electron.atom.io/docs/tutorial/windows-store-guide). |

Accessibility

Making accessible applications is important and we’re happy to introduce new functionality to [Devtron](https://electron.atom.io/devtron) and [Spectron](https://electron.atom.io/spectron) that gives developers the opportunity to make their apps better for everyone.

Accessibility concerns in Electron applications are similar to those of websites because they’re both ultimately HTML. With Electron apps, however, you can’t use the online resources for accessibility audits because your app doesn’t have a URL to point the auditor to.

These new features bring those auditing tools to your Electron app. You can choose to add audits to your tests with Spectron or use them within DevTools with Devtron. Read on for a summary of the tools or checkout our [accessibility documentation](https://electron.atom.io/docs/tutorial/accessibility) for more information.

Spectron

In the testing framework Spectron, you can now audit each window and <webview> tag in your application. For example:

app.client.auditAccessibility().then(**function** (audit) {

**if** (audit.failed) {

console.error(audit.message)

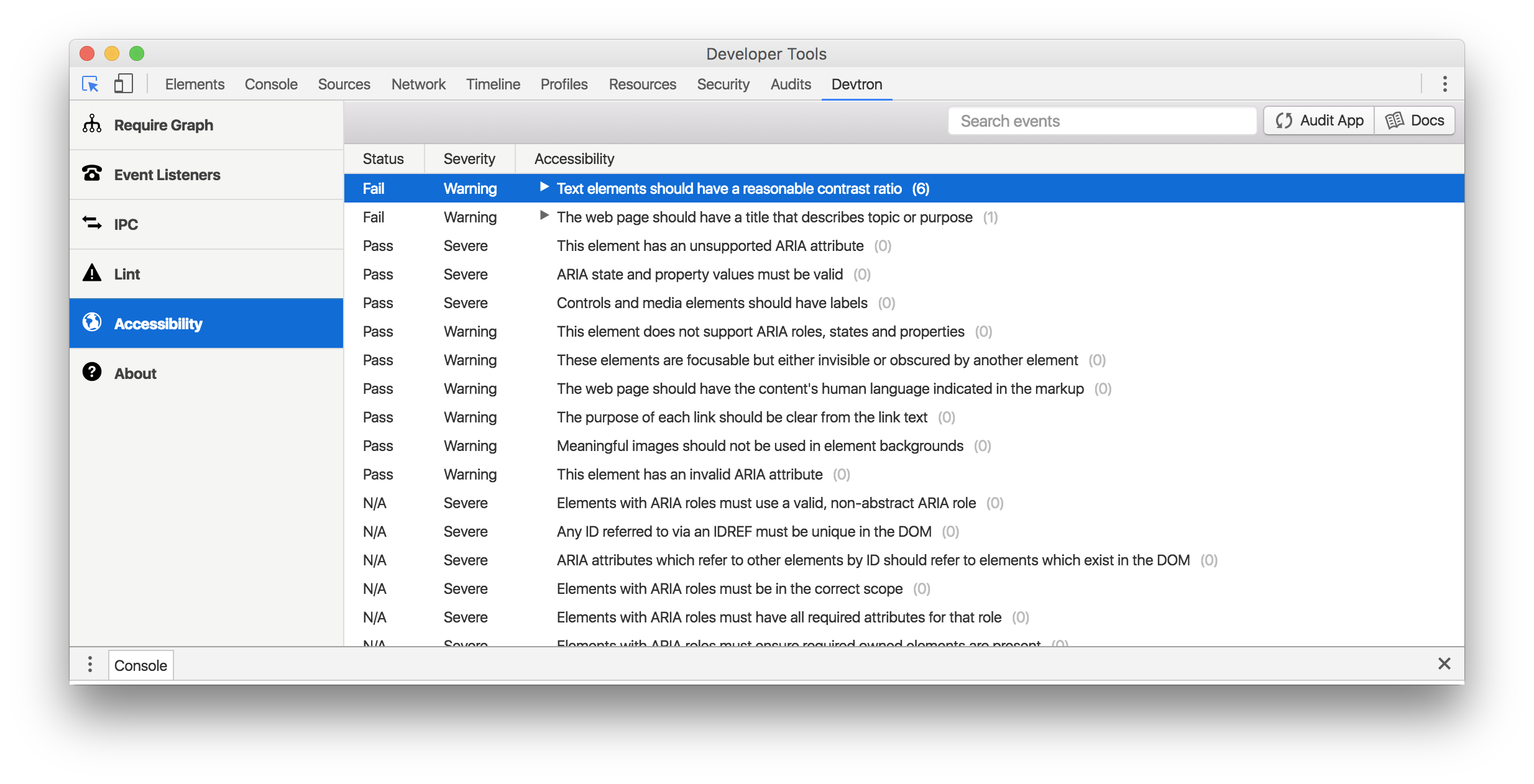
}

})

You can read more about this feature in [Spectron’s documentation](https://github.com/electron/spectron#accessibility-testing).

Devtron

In Devtron, there is a new accessibility tab which will allow you to audit a page in your app, sort and filter the results.



Both of these tools are using the [Accessibility Developer Tools](https://github.com/GoogleChrome/accessibility-developer-tools) library built by Google for Chrome. You can learn more about the accessibility audit rules this library uses on that [repository’s wiki](https://github.com/GoogleChrome/accessibility-developer-tools/wiki/Audit-Rules).

If you know of other great accessibility tools for Electron, add them to the [accessibility documentation](https://electron.atom.io/docs/tutorial/accessibility) with a pull request.

Application Distribution

To distribute your app with Electron, you need to download Electron’s [prebuilt binaries](https://github.com/electron/electron/releases). Next, the folder containing your app should be named app and placed in Electron’s resources directory as shown in the following examples. Note that the location of Electron’s prebuilt binaries is indicated with electron/ in the examples below.

On macOS:

electron/Electron.app/Contents/Resources/app/

├── package.json

├── main.js

└── index.html

On Windows and Linux:

electron/resources/app

├── package.json

├── main.js

└── index.html

Then execute Electron.app (or electron on Linux, electron.exe on Windows), and Electron will start as your app. The electron directory will then be your distribution to deliver to final users.

Packaging Your App into a File

Apart from shipping your app by copying all of its source files, you can also package your app into an [asar](https://github.com/electron/asar)archive to avoid exposing your app’s source code to users.

To use an asar archive to replace the app folder, you need to rename the archive to app.asar, and put it under Electron’s resources directory like below, and Electron will then try to read the archive and start from it.

On macOS:

electron/Electron.app/Contents/Resources/

└── app.asar

On Windows and Linux:

electron/resources/

└── app.asar

More details can be found in [Application packaging](https://electron.atom.io/docs/tutorial/application-packaging).

Rebranding with Downloaded Binaries

After bundling your app into Electron, you will want to rebrand Electron before distributing it to users.

Windows

You can rename electron.exe to any name you like, and edit its icon and other information with tools like [rcedit](https://github.com/atom/rcedit).

macOS

You can rename Electron.app to any name you want, and you also have to rename the CFBundleDisplayName, CFBundleIdentifier and CFBundleName fields in the following files:

* Electron.app/Contents/Info.plist
* Electron.app/Contents/Frameworks/Electron Helper.app/Contents/Info.plist

You can also rename the helper app to avoid showing Electron Helper in the Activity Monitor, but make sure you have renamed the helper app’s executable file’s name.

The structure of a renamed app would be like:

MyApp.app/Contents

├── Info.plist

├── MacOS/

│   └── MyApp

└── Frameworks/

├── MyApp Helper EH.app

| ├── Info.plist

| └── MacOS/

|    └── MyApp Helper EH

├── MyApp Helper NP.app

| ├── Info.plist

| └── MacOS/

|    └── MyApp Helper NP

└── MyApp Helper.app

├── Info.plist

└── MacOS/

   └── MyApp Helper

Linux

You can rename the electron executable to any name you like.

Packaging Tools

Apart from packaging your app manually, you can also choose to use third party packaging tools to do the work for you:

* [electron-forge](https://github.com/electron-userland/electron-forge)
* [electron-builder](https://github.com/electron-userland/electron-builder)
* [electron-packager](https://github.com/electron-userland/electron-packager)

Rebranding by Rebuilding Electron from Source

It is also possible to rebrand Electron by changing the product name and building it from source. To do this you need to modify the atom.gyp file and have a clean rebuild.

Creating a Custom Electron Fork

Creating a custom fork of Electron is almost certainly not something you will need to do in order to build your app, even for “Production Level” applications. Using a tool such as electron-packager or electron-forge will allow you to “Rebrand” Electron without having to do these steps.

You need to fork Electron when you have custom C++ code that you have patched directly into Electron, that either cannot be upstreamed, or has been rejected from the official version. As maintainers of Electron, we very much would like to make your scenario work, so please try as hard as you can to get your changes into the official version of Electron, it will be much much easier on you, and we appreciate your help.

Creating a Custom Release with surf-build

1. Install [Surf](https://github.com/surf-build/surf), via npm: npm install -g surf-build@latest
2. Create a new S3 bucket and create the following empty directory structure:
3. - atom-shell/
4. - symbols/
5. - dist/
6. Set the following Environment Variables:

* ELECTRON\_GITHUB\_TOKEN - a token that can create releases on GitHub
* ELECTRON\_S3\_ACCESS\_KEY, ELECTRON\_S3\_BUCKET, ELECTRON\_S3\_SECRET\_KEY - the place where you’ll upload node.js headers as well as symbols
* ELECTRON\_RELEASE - Set to true and the upload part will run, leave unset and surf-build will just do CI-type checks, appropriate to run for every pull request.
* CI - Set to true or else it will fail
* GITHUB\_TOKEN - set it to the same as ELECTRON\_GITHUB\_TOKEN
* SURF\_TEMP - set to C:\Temp on Windows to prevent path too long issues
* TARGET\_ARCH - set to ia32 or x64

1. In script/upload.py, you *must* set ELECTRON\_REPO to your fork (MYORG/electron), especially if you are a contributor to Electron proper.
2. surf-build -r https://github.com/MYORG/electron -s YOUR\_COMMIT -n 'surf-PLATFORM-ARCH'
3. Wait a very, very long time for the build to complete.

Application Packaging

To mitigate [issues](https://github.com/joyent/node/issues/6960) around long path names on Windows, slightly speed up require and conceal your source code from cursory inspection, you can choose to package your app into an [asar](https://github.com/electron/asar) archive with little changes to your source code.

Generating asar Archive

An [asar](https://github.com/electron/asar) archive is a simple tar-like format that concatenates files into a single file. Electron can read arbitrary files from it without unpacking the whole file.

Steps to package your app into an asar archive:

1. Install the asar Utility

$ npm install -g asar

2. Package with asar pack

$ asar pack your-app app.asar

Using asar Archives

In Electron there are two sets of APIs: Node APIs provided by Node.js and Web APIs provided by Chromium. Both APIs support reading files from asar archives.

Node API

With special patches in Electron, Node APIs like fs.readFile and require treat asar archives as virtual directories, and the files in it as normal files in the filesystem.

For example, suppose we have an example.asar archive under /path/to:

$ asar list /path/to/example.asar

/app.js

/file.txt

/dir/module.js

/static/index.html

/static/main.css

/static/jquery.min.js

Read a file in the asar archive:

**const** fs = require('fs')

fs.readFileSync('/path/to/example.asar/file.txt')

List all files under the root of the archive:

**const** fs = require('fs')

fs.readdirSync('/path/to/example.asar')

Use a module from the archive:

require('/path/to/example.asar/dir/module.js')

You can also display a web page in an asar archive with BrowserWindow:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow({width: 800, height: 600})

win.loadURL('file:///path/to/example.asar/static/index.html')

Web API

In a web page, files in an archive can be requested with the file: protocol. Like the Node API, asar archives are treated as directories.

For example, to get a file with $.get:

<script>

**let** $ = require('./jquery.min.js')

$.get('file:///path/to/example.asar/file.txt', (data) => {

console.log(data)

})

</script>

Treating an asar Archive as a Normal File

For some cases like verifying the asar archive’s checksum, we need to read the content of an asar archive as a file. For this purpose you can use the built-in original-fs module which provides original fs APIs without asar support:

**const** originalFs = require('original-fs')

originalFs.readFileSync('/path/to/example.asar')

You can also set process.noAsar to true to disable the support for asar in the fs module:

**const** fs = require('fs')

process.noAsar = true

fs.readFileSync('/path/to/example.asar')

Limitations of the Node API

Even though we tried hard to make asar archives in the Node API work like directories as much as possible, there are still limitations due to the low-level nature of the Node API.

Archives Are Read-only

The archives can not be modified so all Node APIs that can modify files will not work with asar archives.

Working Directory Can Not Be Set to Directories in Archive

Though asar archives are treated as directories, there are no actual directories in the filesystem, so you can never set the working directory to directories in asar archives. Passing them as the cwd option of some APIs will also cause errors.

Extra Unpacking on Some APIs

Most fs APIs can read a file or get a file’s information from asar archives without unpacking, but for some APIs that rely on passing the real file path to underlying system calls, Electron will extract the needed file into a temporary file and pass the path of the temporary file to the APIs to make them work. This adds a little overhead for those APIs.

APIs that requires extra unpacking are:

* child\_process.execFile
* child\_process.execFileSync
* fs.open
* fs.openSync
* process.dlopen - Used by require on native modules

Fake Stat Information of fs.stat

The Stats object returned by fs.stat and its friends on files in asar archives is generated by guessing, because those files do not exist on the filesystem. So you should not trust the Stats object except for getting file size and checking file type.

Executing Binaries Inside asar Archive

There are Node APIs that can execute binaries like child\_process.exec, child\_process.spawn and child\_process.execFile, but only execFile is supported to execute binaries inside asar archive.

This is because exec and spawn accept command instead of file as input, and commands are executed under shell. There is no reliable way to determine whether a command uses a file in asar archive, and even if we do, we can not be sure whether we can replace the path in command without side effects.

Adding Unpacked Files in asar Archive

As stated above, some Node APIs will unpack the file to filesystem when calling, apart from the performance issues, it could also lead to false alerts of virus scanners.

To work around this, you can unpack some files creating archives by using the --unpack option, an example of excluding shared libraries of native modules is:

$ asar pack app app.asar --unpack **\***.node

After running the command, apart from the app.asar, there is also an app.asar.unpacked folder generated which contains the unpacked files, you should copy it together with app.asar when shipping it to users.

Debugging the Main Process in node-inspector

[node-inspector](https://github.com/node-inspector/node-inspector) provides a familiar DevTools GUI that can be used in Chrome to debug Electron’s main process, however, because node-inspector relies on some native Node modules they must be rebuilt to target the version of Electron you wish to debug. You can either rebuild the node-inspector dependencies yourself, or let [electron-inspector](https://github.com/enlight/electron-inspector) do it for you, both approaches are covered in this document.

**Note**: At the time of writing the latest release of node-inspector (0.12.8) can’t be rebuilt to target Electron 1.3.0 or later without patching one of its dependencies. If you use electron-inspector it will take care of this for you.

Use electron-inspector for Debugging

1. Install the [node-gyp required tools](https://github.com/nodejs/node-gyp#installation)

2. Install [electron-rebuild](https://github.com/electron/electron-rebuild), if you haven’t done so already.

npm install electron-rebuild --save-dev

3. Install [electron-inspector](https://github.com/enlight/electron-inspector)

npm install electron-inspector --save-dev

4. Start Electron

Launch Electron with the --debug switch:

electron --debug=5858 your/app

or, to pause execution on the first line of JavaScript:

electron --debug-brk=5858 your/app

5. Start electron-inspector

On macOS / Linux:

node\_modules/.bin/electron-inspector

On Windows:

node\_modules\\.bin\\electron-inspector

electron-inspector will need to rebuild node-inspector dependencies on the first run, and any time you change your Electron version. The rebuild process may require an internet connection to download Node headers and libs, and may take a few minutes.

6. Load the debugger UI

Open http://127.0.0.1:8080/debug?ws=127.0.0.1:8080&port=5858 in the Chrome browser. You may have to click pause if starting with --debug-brk to force the UI to update.

Use node-inspector for Debugging

1. Install the [node-gyp required tools](https://github.com/nodejs/node-gyp#installation)

2. Install [node-inspector](https://github.com/node-inspector/node-inspector)

$ npm install node-inspector

3. Install [node-pre-gyp](https://github.com/mapbox/node-pre-gyp)

$ npm install node-pre-gyp

4. Recompile the node-inspector v8 modules for Electron

**Note:** Update the target argument to be your Electron version number

$ node\_modules/.bin/node-pre-gyp --target=1.2.5 --runtime=electron --fallback-to-build --directory node\_modules/v8-debug/ --dist-url=https://atom.io/download/atom-shell reinstall

$ node\_modules/.bin/node-pre-gyp --target=1.2.5 --runtime=electron --fallback-to-build --directory node\_modules/v8-profiler/ --dist-url=https://atom.io/download/atom-shell reinstall

See also [How to install native modules](https://electron.atom.io/docs/tutorial/using-native-node-modules#how-to-install-native-modules).

5. Enable debug mode for Electron

You can either start Electron with a debug flag like:

$ electron --debug=5858 your/app

or, to pause your script on the first line:

$ electron --debug-brk=5858 your/app

6. Start the [node-inspector](https://github.com/node-inspector/node-inspector) server using Electron

$ ELECTRON\_RUN\_AS\_NODE=true path/to/electron.exe node\_modules/node-inspector/bin/inspector.js

7. Load the debugger UI

Open http://127.0.0.1:8080/debug?ws=127.0.0.1:8080&port=5858 in the Chrome browser. You may have to click pause if starting with --debug-brk to see the entry line.

Debugging the Main Process in VSCode

1. Open an Electron project in VSCode.

$ git clone git@github.com:electron/electron-quick-start.git

$ code electron-quick-start

2. Add a file .vscode/launch.json with the following configuration:

{

"version": "0.2.0",

"configurations": [

{

"name": "Debug Main Process",

"type": "node",

"request": "launch",

"cwd": "${workspaceRoot}",

"runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/electron",

"windows": {

"runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/electron.cmd"

},

"args" : ["."]

}

]

}

**Note:** For Windows, use "${workspaceRoot}/node\_modules/.bin/electron.cmd" for runtimeExecutable.

3. Debugging

Set some breakpoints in main.js, and start debugging in the [Debug View](https://code.visualstudio.com/docs/editor/debugging). You should be able to hit the breakpoints.

Here is a pre-configured project that you can download and directly debug in VSCode: https://github.com/octref/vscode-electron-debug/tree/master/electron-quick-start

Debugging the Main Process

The DevTools in an Electron browser window can only debug JavaScript that’s executed in that window (i.e. the web pages). To debug JavaScript that’s executed in the main process you will need to use an external debugger and launch Electron with the --inspect or --inspect-brk switch.

Command Line Switches

Use one of the following command line switches to enable debugging of the main process:

--inspect=[port]

Electron will listen for V8 inspector protocol messages on the specified port, an external debugger will need to connect on this port. The default port is 5858.

electron --inspect=5858 your/app

--inspect-brk=[port]

Like --inspector but pauses execution on the first line of JavaScript.

External Debuggers

You will need to use a debugger that supports the V8 inspector protocol.

* Connect Chrome by visiting chrome://inspect and selecting to inspect the launched Electron app present there.
* [Debugging the Main Process in VSCode](https://electron.atom.io/docs/tutorial/debugging-main-process-vscode)

Desktop Environment Integration

Different operating systems provide different features for integrating desktop applications into their desktop environments. For example, on Windows, applications can put shortcuts in the JumpList of task bar, and on Mac, applications can put a custom menu in the dock menu.

This guide explains how to integrate your application into those desktop environments with Electron APIs.

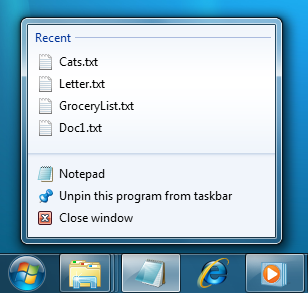
Notifications

See [Notifications](https://electron.atom.io/docs/tutorial/notifications)

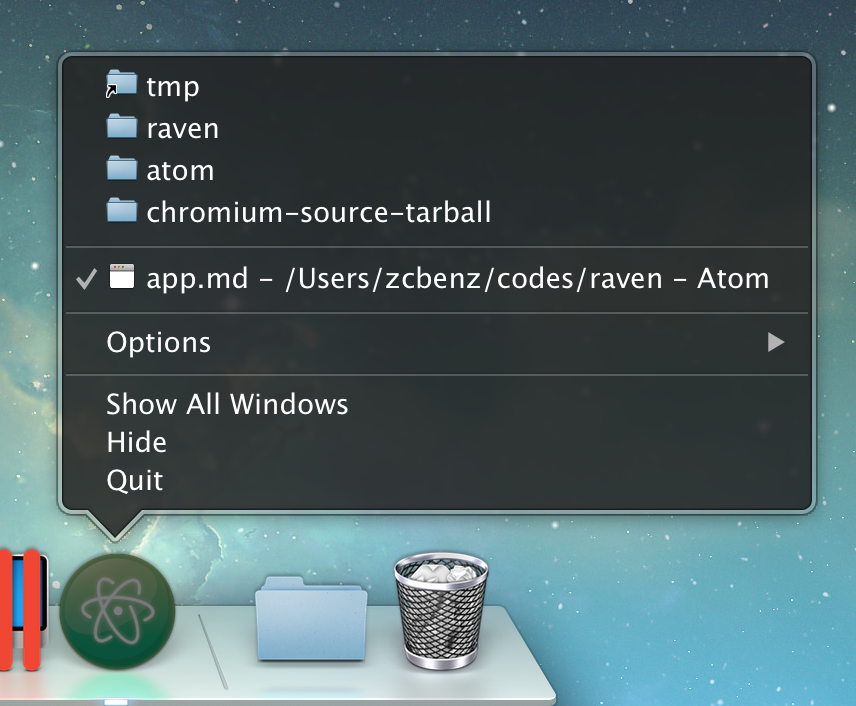
Recent documents (Windows & macOS)

Windows and macOS provide easy access to a list of recent documents opened by the application via JumpList or dock menu, respectively.

**JumpList:**



**Application dock menu:**



To add a file to recent documents, you can use the [app.addRecentDocument](https://electron.atom.io/docs/api/app#appaddrecentdocumentpath-os-x-windows) API:

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

app.addRecentDocument('/Users/USERNAME/Desktop/work.type')

And you can use [app.clearRecentDocuments](https://electron.atom.io/docs/api/app#appclearrecentdocuments-os-x-windows) API to empty the recent documents list:

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

app.clearRecentDocuments()

Windows Notes

In order to be able to use this feature on Windows, your application has to be registered as a handler of the file type of the document, otherwise the file won’t appear in JumpList even after you have added it. You can find everything on registering your application in [Application Registration](http://msdn.microsoft.com/en-us/library/windows/desktop/ee872121(v=vs.85).aspx).

When a user clicks a file from the JumpList, a new instance of your application will be started with the path of the file added as a command line argument.

macOS Notes

When a file is requested from the recent documents menu, the open-file event of app module will be emitted for it.

Custom Dock Menu (macOS)

macOS enables developers to specify a custom menu for the dock, which usually contains some shortcuts for commonly used features of your application:

**Dock menu of Terminal.app:**



To set your custom dock menu, you can use the app.dock.setMenu API, which is only available on macOS:

**const** {app, Menu} = require('electron')

**const** dockMenu = Menu.buildFromTemplate([

{label: 'New Window', click () { console.log('New Window') }},

{label: 'New Window with Settings',

submenu: [

{label: 'Basic'},

{label: 'Pro'}

]

},

{label: 'New Command...'}

])

app.dock.setMenu(dockMenu)

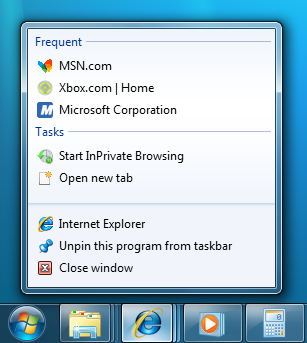
User Tasks (Windows)

On Windows you can specify custom actions in the Tasks category of JumpList, as quoted from MSDN:

Applications define tasks based on both the program’s features and the key things a user is expected to do with them. Tasks should be context-free, in that the application does not need to be running for them to work. They should also be the statistically most common actions that a normal user would perform in an application, such as compose an email message or open the calendar in a mail program, create a new document in a word processor, launch an application in a certain mode, or launch one of its subcommands. An application should not clutter the menu with advanced features that standard users won’t need or one-time actions such as registration. Do not use tasks for promotional items such as upgrades or special offers.

It is strongly recommended that the task list be static. It should remain the same regardless of the state or status of the application. While it is possible to vary the list dynamically, you should consider that this could confuse the user who does not expect that portion of the destination list to change.

**Tasks of Internet Explorer:**



Unlike the dock menu in macOS which is a real menu, user tasks in Windows work like application shortcuts such that when user clicks a task, a program will be executed with specified arguments.

To set user tasks for your application, you can use [app.setUserTasks](https://electron.atom.io/docs/api/app#appsetusertaskstasks-windows) API:

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

app.setUserTasks([

{

program: process.execPath,

arguments: '--new-window',

iconPath: process.execPath,

iconIndex: 0,

title: 'New Window',

description: 'Create a new window'

}

])

To clean your tasks list, just call app.setUserTasks with an empty array:

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

app.setUserTasks([])

The user tasks will still show even after your application closes, so the icon and program path specified for a task should exist until your application is uninstalled.

Thumbnail Toolbars

On Windows you can add a thumbnail toolbar with specified buttons in a taskbar layout of an application window. It provides users a way to access to a particular window’s command without restoring or activating the window.

From MSDN, it’s illustrated:

This toolbar is simply the familiar standard toolbar common control. It has a maximum of seven buttons. Each button’s ID, image, tooltip, and state are defined in a structure, which is then passed to the taskbar. The application can show, enable, disable, or hide buttons from the thumbnail toolbar as required by its current state.

For example, Windows Media Player might offer standard media transport controls such as play, pause, mute, and stop.

**Thumbnail toolbar of Windows Media Player:**



You can use [BrowserWindow.setThumbarButtons](https://electron.atom.io/docs/api/browser-window#winsetthumbarbuttonsbuttons-windows-7) to set thumbnail toolbar in your application:

**const** {BrowserWindow} = require('electron')

**const** path = require('path')

**let** win = **new** BrowserWindow({

width: 800,

height: 600

})

win.setThumbarButtons([

{

tooltip: 'button1',

icon: path.join(\_\_dirname, 'button1.png'),

click () { console.log('button1 clicked') }

},

{

tooltip: 'button2',

icon: path.join(\_\_dirname, 'button2.png'),

flags: ['enabled', 'dismissonclick'],

click () { console.log('button2 clicked.') }

}

])

To clean thumbnail toolbar buttons, just call BrowserWindow.setThumbarButtons with an empty array:

**const** {BrowserWindow} = require('electron')

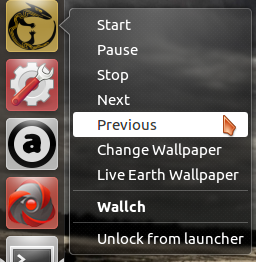
**let** win = **new** BrowserWindow()

win.setThumbarButtons([])

Unity Launcher Shortcuts (Linux)

In Unity, you can add custom entries to its launcher via modifying the .desktop file, see [Adding Shortcuts to a Launcher](https://help.ubuntu.com/community/UnityLaunchersAndDesktopFiles#Adding_shortcuts_to_a_launcher).

**Launcher shortcuts of Audacious:**



Progress Bar in Taskbar (Windows, macOS, Unity)

On Windows a taskbar button can be used to display a progress bar. This enables a window to provide progress information to the user without the user having to switch to the window itself.

On macOS the progress bar will be displayed as a part of the dock icon.

The Unity DE also has a similar feature that allows you to specify the progress bar in the launcher.

**Progress bar in taskbar button:**



To set the progress bar for a Window, you can use the [BrowserWindow.setProgressBar](https://electron.atom.io/docs/api/browser-window#winsetprogressbarprogress) API:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.setProgressBar(0.5)

Icon Overlays in Taskbar (Windows)

On Windows a taskbar button can use a small overlay to display application status, as quoted from MSDN:

Icon overlays serve as a contextual notification of status, and are intended to negate the need for a separate notification area status icon to communicate that information to the user. For instance, the new mail status in Microsoft Outlook, currently shown in the notification area, can now be indicated through an overlay on the taskbar button. Again, you must decide during your development cycle which method is best for your application. Overlay icons are intended to supply important, long-standing status or notifications such as network status, messenger status, or new mail. The user should not be presented with constantly changing overlays or animations.

**Overlay on taskbar button:**

Overlay on taskbar button

To set the overlay icon for a window, you can use the [BrowserWindow.setOverlayIcon](https://electron.atom.io/docs/api/browser-window#winsetoverlayiconoverlay-description-windows-7) API:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.setOverlayIcon('path/to/overlay.png', 'Description for overlay')

Flash Frame (Windows)

On Windows you can highlight the taskbar button to get the user’s attention. This is similar to bouncing the dock icon on macOS. From the MSDN reference documentation:

Typically, a window is flashed to inform the user that the window requires attention but that it does not currently have the keyboard focus.

To flash the BrowserWindow taskbar button, you can use the [BrowserWindow.flashFrame](https://electron.atom.io/docs/api/browser-window#winflashframeflag) API:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.once('focus', () => win.flashFrame(false))

win.flashFrame(true)

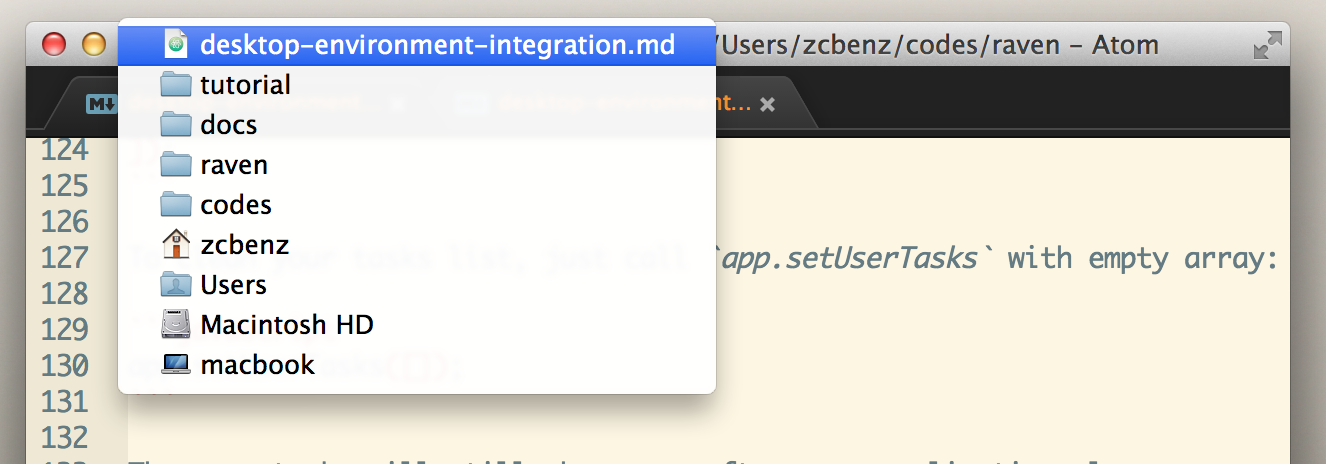
Don’t forget to call the flashFrame method with false to turn off the flash. In the above example, it is called when the window comes into focus, but you might use a timeout or some other event to disable it.

Represented File of Window (macOS)

On macOS a window can set its represented file, so the file’s icon can show in the title bar and when users Command-Click or Control-Click on the title a path popup will show.

You can also set the edited state of a window so that the file icon can indicate whether the document in this window has been modified.

**Represented file popup menu:**



To set the represented file of window, you can use the [BrowserWindow.setRepresentedFilename](https://electron.atom.io/docs/api/browser-window#winsetrepresentedfilenamefilename-os-x) and [BrowserWindow.setDocumentEdited](https://electron.atom.io/docs/api/browser-window#winsetdocumenteditededited-os-x) APIs:

**const** {BrowserWindow} = require('electron')

**let** win = **new** BrowserWindow()

win.setRepresentedFilename('/etc/passwd')

win.setDocumentEdited(true)

Dragging files out of the window

For certain kinds of apps that manipulate on files, it is important to be able to drag files from Electron to other apps. To implement this feature in your app, you need to call webContents.startDrag(item) API on ondragstartevent.

In web page:

<a href="#" id="drag">item</a>

<script type="text/javascript" charset="utf-8">

document.getElementById('drag').ondragstart = (event) => {

event.preventDefault()

ipcRenderer.send('ondragstart', '/path/to/item')

}

</script>

In the main process:

**const** {ipcMain} = require('electron')

ipcMain.on('ondragstart', (event, filePath) => {

event.sender.startDrag({

file: filePath,

icon: '/path/to/icon.png'

})

})

DevTools Extension

Electron supports the [Chrome DevTools Extension](https://developer.chrome.com/extensions/devtools), which can be used to extend the ability of devtools for debugging popular web frameworks.

How to load a DevTools Extension

This document outlines the process for manually loading an extension. You may also try [electron-devtools-installer](https://github.com/GPMDP/electron-devtools-installer), a third-party tool that downloads extensions directly from the Chrome WebStore.

To load an extension in Electron, you need to download it in Chrome browser, locate its filesystem path, and then load it by calling the BrowserWindow.addDevToolsExtension(extension) API.

Using the [React Developer Tools](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi) as example:

1. Install it in Chrome browser.
2. Navigate to chrome://extensions, and find its extension ID, which is a hash string like fmkadmapgofadopljbjfkapdkoienihi.
3. Find out filesystem location used by Chrome for storing extensions:
   * on Windows it is %LOCALAPPDATA%\Google\Chrome\User Data\Default\Extensions;
   * on Linux it could be:
     + ~/.config/google-chrome/Default/Extensions/
     + ~/.config/google-chrome-beta/Default/Extensions/
     + ~/.config/google-chrome-canary/Default/Extensions/
     + ~/.config/chromium/Default/Extensions/
   * on macOS it is ~/Library/Application Support/Google/Chrome/Default/Extensions.
4. Pass the location of the extension to BrowserWindow.addDevToolsExtension API, for the React Developer Tools, it is something like: ~/Library/Application Support/Google/Chrome/Default/Extensions/fmkadmapgofadopljbjfkapdkoienihi/0.15.0\_0

**Note:** The BrowserWindow.addDevToolsExtension API cannot be called before the ready event of the app module is emitted.

The name of the extension is returned by BrowserWindow.addDevToolsExtension, and you can pass the name of the extension to the BrowserWindow.removeDevToolsExtension API to unload it.

Supported DevTools Extensions

Electron only supports a limited set of chrome.\* APIs, so some extensions using unsupported chrome.\* APIs for chrome extension features may not work. Following Devtools Extensions are tested and guaranteed to work in Electron:

* [Ember Inspector](https://chrome.google.com/webstore/detail/ember-inspector/bmdblncegkenkacieihfhpjfppoconhi)
* [React Developer Tools](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi)
* [Backbone Debugger](https://chrome.google.com/webstore/detail/backbone-debugger/bhljhndlimiafopmmhjlgfpnnchjjbhd)
* [jQuery Debugger](https://chrome.google.com/webstore/detail/jquery-debugger/dbhhnnnpaeobfddmlalhnehgclcmjimi)
* [AngularJS Batarang](https://chrome.google.com/webstore/detail/angularjs-batarang/ighdmehidhipcmcojjgiloacoafjmpfk)
* [Vue.js devtools](https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd)
* [Cerebral Debugger](http://www.cerebraljs.com/documentation/the_debugger)
* [Redux DevTools Extension](https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd)

What should I do if a DevTools Extension is not working?

First please make sure the extension is still being maintained, some extensions can not even work for recent versions of Chrome browser, and we are not able to do anything for them.

Then file a bug at Electron’s issues list, and describe which part of the extension is not working as expected.

Electron Versioning

If you’ve been using Node and npm for a while, you are probably aware of [Semantic Versioning](http://semver.org/), or SemVer for short. It’s a convention for specifying version numbers for software that helps communicate intentions to the users of your software.

Overview of Semantic Versioning

Semantic versions are always made up of three numbers:

major.minor.patch

Semantic version numbers are bumped (incremented) using the following rules:

* **Major** is for changes that break backwards compatibility.
* **Minor** is for new features that don’t break backwards compatibility.
* **Patch** is for bug fixes and other minor changes.

A simple mnemonic for remembering this scheme is as follows:

breaking.feature.fix

Electron Versioning

Due to its dependency on Node and Chromium, it is not possible for the Electron project to adhere to a SemVer policy. **You should therefore always reference a specific version of Electron.**

Electron version numbers are bumped using the following rules:

* **Major** is for breaking changes in Electron’s API. If you upgrade from 0.37.0 to 1.0.0, you will have to make changes to your app.
* **Minor** is for major Chrome and minor Node upgrades, or significant Electron changes. If you upgrade from 1.5.0 to 1.6.0, your app is supposed to still work, but you might have to work around small changes.
* **Patch** is for new features and bug fixes. If you upgrade from 1.6.2 to 1.6.3, your app will continue to work as-is.

We recommend that you set a fixed version when installing Electron from npm:

npm install electron --save-exact --save-dev

The --save-exact flag will add electron to your package.json file without using a ^ or ~, e.g. 1.6.2 instead of ^1.6.2. This practice ensures that all upgrades of Electron are a manual operation made by you, the developer.

Alternatively, you can use the ~ prefix in your SemVer range, like ~1.6.2. This will lock your major and minor version, but allow new patch versions to be installed.

Installation

Tips for installing Electron

To install prebuilt Electron binaries, use [npm](https://docs.npmjs.com/). The preferred method is to install Electron as a development dependency in your app:

npm install electron --save-dev --save-exact

The --save-exact flag is recommended as Electron does not follow semantic versioning. See the [versioning doc](https://electron.atom.io/docs/tutorial/electron-versioning/)for info on how to manage Electron versions in your apps.

Global Installation

You can also install the electron command globally in your $PATH:

npm install electron -g

Customization

If you want to change the architecture that is downloaded (e.g., ia32 on an x64 machine), you can use the --arch flag with npm install or set the npm\_config\_arch environment variable:

npm install --arch=ia32 electron

In addition to changing the architecture, you can also specify the platform (e.g., win32, linux, etc.) using the --platform flag:

npm install --platform=win32 electron

Proxies

If you need to use an HTTP proxy you can [set these environment variables](https://github.com/request/request/tree/f0c4ec061141051988d1216c24936ad2e7d5c45d#controlling-proxy-behaviour-using-environment-variables).

Troubleshooting

When running npm install electron, some users occasionally encounter installation errors.

In almost all cases, these errors are the result of network problems and not actual issues with the electron npm package. Errors like ELIFECYCLE, EAI\_AGAIN, ECONNRESET, and ETIMEDOUT are all indications of such network problems. The best resolution is to try switching networks, or just wait a bit and try installing again.

You can also attempt to download Electron directly from [electron/electron/releases](https://github.com/electron/electron/releases) if installing via npm is failing.

If installation fails with an EACCESS error you may need to [fix your npm permissions](https://docs.npmjs.com/getting-started/fixing-npm-permissions).

If the above error persists, the [unsafe-perm](https://docs.npmjs.com/misc/config#unsafe-perm) flag may need to be set to true:

sudo npm install electron --unsafe-perm=true

On slower networks, it may be advisable to use the --verbose flag in order to show download progress:

npm install --verbose electron

If you need to force a re-download of the asset and the SHASUM file set the force\_no\_cache enviroment variable to true.

Keyboard Shortcuts

Configure local and global keyboard shortcuts

Local Shortcuts

You can use the [Menu](https://electron.atom.io/docs/api/menu) module to configure keyboard shortcuts that will be triggered only when the app is focused. To do so, specify an [accelerator](https://electron.atom.io/docs/api/accelerator) property when creating a [MenuItem](https://electron.atom.io/docs/api/menu-item).

**const** {Menu, MenuItem} = require('electron')

**const** menu = **new** Menu()

menu.append(**new** MenuItem({

label: 'Print',

accelerator: 'CmdOrCtrl+P',

click: () => { console.log('time to print stuff') }

}))

It’s easy to configure different key combinations based on the user’s operating system.

{

accelerator: process.platform === 'darwin' ? 'Alt+Cmd+I' : 'Ctrl+Shift+I'

}

Global Shortcuts

You can use the [globalShortcut](https://electron.atom.io/docs/api/global-shortcut) module to detect keyboard events even when the application does not have keyboard focus.

**const** {app, globalShortcut} = require('electron')

app.on('ready', () => {

globalShortcut.register('CommandOrControl+X', () => {

console.log('CommandOrControl+X is pressed')

})

})

Shortcuts within a BrowserWindow

If you want to handle keyboard shortcuts for a [BrowserWindow](https://electron.atom.io/docs/api/browser-window), you can use the keyup and keydown event listeners on the window object inside the renderer process.

window.addEventListener('keyup', doSomething, true)

Note the third parameter true which means the listener will always receive key presses before other listeners so they can’t have stopPropagation() called on them.

The [before-input-event](https://electron.atom.io/docs/tutorial/web-contents#event-before-input-event) event is emitted before dispatching keydown and keyup events in the page. It can be used to catch and handle custom shortcuts that are not visible in the menu.

If you don’t want to do manual shortcut parsing there are libraries that do advanced key detection such as [mousetrap](https://github.com/ccampbell/mousetrap).

Mousetrap.bind('4', () => { console.log('4') })

Mousetrap.bind('?', () => { console.log('show shortcuts!') })

Mousetrap.bind('esc', () => { console.log('escape') }, 'keyup')

*// combinations*

Mousetrap.bind('command+shift+k', () => { console.log('command shift k') })

*// map multiple combinations to the same callback*

Mousetrap.bind(['command+k', 'ctrl+k'], () => {

console.log('command k or control k')

*// return false to prevent default behavior and stop event from bubbling*

**return** false

})

*// gmail style sequences*

Mousetrap.bind('g i', () => { console.log('go to inbox') })

Mousetrap.bind('\* a', () => { console.log('select all') })

*// konami code!*

Mousetrap.bind('up up down down left right left right b a enter', () => {

console.log('konami code')

})

Mac App Store Submission Guide

Since v0.34.0, Electron allows submitting packaged apps to the Mac App Store (MAS). This guide provides information on: how to submit your app and the limitations of the MAS build.

**Note:** Submitting an app to Mac App Store requires enrolling [Apple Developer Program](https://developer.apple.com/support/compare-memberships/), which costs money.

How to Submit Your App

The following steps introduce a simple way to submit your app to Mac App Store. However, these steps do not ensure your app will be approved by Apple; you still need to read Apple’s [Submitting Your App](https://developer.apple.com/library/mac/documentation/IDEs/Conceptual/AppDistributionGuide/SubmittingYourApp/SubmittingYourApp.html) guide on how to meet the Mac App Store requirements.

Get Certificate

To submit your app to the Mac App Store, you first must get a certificate from Apple. You can follow these [existing guides](https://github.com/nwjs/nw.js/wiki/Mac-App-Store-%28MAS%29-Submission-Guideline#first-steps) on web.

Get Team ID

Before signing your app, you need to know the Team ID of your account. To locate your Team ID, Sign in to [Apple Developer Center](https://developer.apple.com/account/), and click Membership in the sidebar. Your Team ID appears in the Membership Information section under the team name.

Sign Your App

After finishing the preparation work, you can package your app by following [Application Distribution](https://electron.atom.io/docs/tutorial/application-distribution), and then proceed to signing your app.

First, you have to add a ElectronTeamID key to your app’s Info.plist, which has your Team ID as value:

<plist version="1.0">

<dict>

...

<key>ElectronTeamID</key>

<string>TEAM\_ID</string>

</dict>

</plist>

Then, you need to prepare two entitlements files.

child.plist:

***<?xml version="1.0" encoding="UTF-8"?>***

***<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">***

<plist version="1.0">

<dict>

<key>com.apple.security.app-sandbox</key>

<true/>

<key>com.apple.security.inherit</key>

<true/>

</dict>

</plist>

parent.plist:

***<?xml version="1.0" encoding="UTF-8"?>***

***<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">***

<plist version="1.0">

<dict>

<key>com.apple.security.app-sandbox</key>

<true/>

<key>com.apple.security.application-groups</key>

<string>TEAM\_ID.your.bundle.id</string>

</dict>

</plist>

You have to replace TEAM\_ID with your Team ID, and replace your.bundle.id with the Bundle ID of your app.

And then sign your app with the following script:

*#!/bin/bash*

*# Name of your app.*

APP="YourApp"

*# The path of your app to sign.*

APP\_PATH="/path/to/YourApp.app"

*# The path to the location you want to put the signed package.*

RESULT\_PATH="~/Desktop/$APP.pkg"

*# The name of certificates you requested.*

APP\_KEY="3rd Party Mac Developer Application: Company Name (APPIDENTITY)"

INSTALLER\_KEY="3rd Party Mac Developer Installer: Company Name (APPIDENTITY)"

*# The path of your plist files.*

CHILD\_PLIST="/path/to/child.plist"

PARENT\_PLIST="/path/to/parent.plist"

FRAMEWORKS\_PATH="$APP\_PATH/Contents/Frameworks"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/Electron Framework.framework/Versions/A/Electron Framework"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/Electron Framework.framework/Versions/A/Libraries/libnode.dylib"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/Electron Framework.framework"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/$APP Helper.app/Contents/MacOS/$APP Helper"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/$APP Helper.app/"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/$APP Helper EH.app/Contents/MacOS/$APP Helper EH"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/$APP Helper EH.app/"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/$APP Helper NP.app/Contents/MacOS/$APP Helper NP"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$FRAMEWORKS\_PATH/$APP Helper NP.app/"

codesign -s "$APP\_KEY" -f --entitlements "$CHILD\_PLIST" "$APP\_PATH/Contents/MacOS/$APP"

codesign -s "$APP\_KEY" -f --entitlements "$PARENT\_PLIST" "$APP\_PATH"

productbuild --component "$APP\_PATH" /Applications --sign "$INSTALLER\_KEY" "$RESULT\_PATH"

If you are new to app sandboxing under macOS, you should also read through Apple’s [Enabling App Sandbox](https://developer.apple.com/library/ios/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/EnablingAppSandbox.html) to have a basic idea, then add keys for the permissions needed by your app to the entitlements files.

Apart from manually signing your app, you can also choose to use the [electron-osx-sign](https://github.com/electron-userland/electron-osx-sign) module to do the job.

Sign Native Modules

Native modules used in your app also need to be signed. If using electron-osx-sign, be sure to include the path to the built binaries in the argument list:

electron-osx-sign YourApp.app YourApp.app/Contents/Resources/app/node\_modules/nativemodule/build/release/nativemodule

Also note that native modules may have intermediate files produced which should not be included (as they would also need to be signed). If you use [electron-packager](https://github.com/electron-userland/electron-packager) before version 8.1.0, add --ignore=.+\.o$ to your build step to ignore these files. Versions 8.1.0 and later ignores those files by default.

Upload Your App

After signing your app, you can use Application Loader to upload it to iTunes Connect for processing, making sure you have [created a record](https://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/CreatingiTunesConnectRecord.html) before uploading.

Submit Your App for Review

After these steps, you can [submit your app for review](https://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/SubmittingTheApp.html).

Limitations of MAS Build

In order to satisfy all requirements for app sandboxing, the following modules have been disabled in the MAS build:

* crashReporter
* autoUpdater

and the following behaviors have been changed:

* Video capture may not work for some machines.
* Certain accessibility features may not work.
* Apps will not be aware of DNS changes.
* APIs for launching apps at login are disabled. See https://github.com/electron/electron/issues/7312#issuecomment-249479237

Also, due to the usage of app sandboxing, the resources which can be accessed by the app are strictly limited; you can read [App Sandboxing](https://developer.apple.com/app-sandboxing/) for more information.

Additional Entitlements

Depending on which Electron APIs your app uses, you may need to add additional entitlements to your parent.plist file to be able to use these APIs from your app’s Mac App Store build.

Network Access

Enable outgoing network connections to allow your app to connect to a server:

<key>com.apple.security.network.client</key>

<true/>

Enable incoming network connections to allow your app to open a network listening socket:

<key>com.apple.security.network.server</key>

<true/>

See the [Enabling Network Access documentation](https://developer.apple.com/library/ios/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/EnablingAppSandbox.html#//apple_ref/doc/uid/TP40011195-CH4-SW9) for more details.

dialog.showOpenDialog

<key>com.apple.security.files.user-selected.read-only</key>

<true/>

See the [Enabling User-Selected File Access documentation](https://developer.apple.com/library/mac/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/EnablingAppSandbox.html#//apple_ref/doc/uid/TP40011195-CH4-SW6) for more details.

dialog.showSaveDialog

<key>com.apple.security.files.user-selected.read-write</key>

<true/>

See the [Enabling User-Selected File Access documentation](https://developer.apple.com/library/mac/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/EnablingAppSandbox.html#//apple_ref/doc/uid/TP40011195-CH4-SW6) for more details.

Known issues

shell.openItem(filePath)

This will fail when the app is signed for distribution in the Mac App Store. Subscribe to [#9005](https://github.com/electron/electron/issues/9005) for updates.

Workaround

shell.openExternal('file://' + filePath) will open the file in the default application as long as the extension is associated with an installed app.

Cryptographic Algorithms Used by Electron

Depending on the country and region you are located, Mac App Store may require documenting the cryptographic algorithms used in your app, and even ask you to submit a copy of U.S. Encryption Registration (ERN) approval.

Electron uses following cryptographic algorithms:

* AES - [NIST SP 800-38A](http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf), [NIST SP 800-38D](http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf), [RFC 3394](http://www.ietf.org/rfc/rfc3394.txt)
* HMAC - [FIPS 198-1](http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf)
* ECDSA - ANS X9.62–2005
* ECDH - ANS X9.63–2001
* HKDF - [NIST SP 800-56C](http://csrc.nist.gov/publications/nistpubs/800-56C/SP-800-56C.pdf)
* PBKDF2 - [RFC 2898](https://tools.ietf.org/html/rfc2898)
* RSA - [RFC 3447](http://www.ietf.org/rfc/rfc3447)
* SHA - [FIPS 180-4](http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf)
* Blowfish - https://www.schneier.com/cryptography/blowfish/
* CAST - [RFC 2144](https://tools.ietf.org/html/rfc2144), [RFC 2612](https://tools.ietf.org/html/rfc2612)
* DES - [FIPS 46-3](http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf)
* DH - [RFC 2631](https://tools.ietf.org/html/rfc2631)
* DSA - [ANSI X9.30](http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+X9.30-1%3A1997)
* EC - [SEC 1](http://www.secg.org/sec1-v2.pdf)
* IDEA - “On the Design and Security of Block Ciphers” book by X. Lai
* MD2 - [RFC 1319](http://tools.ietf.org/html/rfc1319)
* MD4 - [RFC 6150](https://tools.ietf.org/html/rfc6150)
* MD5 - [RFC 1321](https://tools.ietf.org/html/rfc1321)
* MDC2 - [ISO/IEC 10118-2](https://www.openssl.org/docs/manmaster/crypto/mdc2.html)
* RC2 - [RFC 2268](https://tools.ietf.org/html/rfc2268)
* RC4 - [RFC 4345](https://tools.ietf.org/html/rfc4345)
* RC5 - http://people.csail.mit.edu/rivest/Rivest-rc5rev.pdf
* RIPEMD - [ISO/IEC 10118-3](http://webstore.ansi.org/RecordDetail.aspx?sku=ISO%2FIEC%2010118-3:2004)

On how to get the ERN approval, you can reference the article: [How to legally submit an app to Apple’s App Store when it uses encryption (or how to obtain an ERN)](https://carouselapps.com/2015/12/15/legally-submit-app-apples-app-store-uses-encryption-obtain-ern/).

Multithreading

With [Web Workers](https://developer.mozilla.org/en/docs/Web/API/Web_Workers_API/Using_web_workers), it is possible to run JavaScript in OS-level threads.

Multi-threaded Node.js

It is possible to use Node.js features in Electron’s Web Workers, to do so the nodeIntegrationInWorker option should be set to true in webPreferences.

**let** win = **new** BrowserWindow({

webPreferences: {

nodeIntegrationInWorker: true

}

})

The nodeIntegrationInWorker can be used independent of nodeIntegration, but sandbox must not be set to true.

Available APIs

All built-in modules of Node.js are supported in Web Workers, and asar archives can still be read with Node.js APIs. However none of Electron’s built-in modules can be used in a multi-threaded environment.

Native Node.js modules

Any native Node.js module can be loaded directly in Web Workers, but it is strongly recommended not to do so. Most existing native modules have been written assuming single-threaded environment, using them in Web Workers will lead to crashes and memory corruptions.

Note that even if a native Node.js module is thread-safe it’s still not safe to load it in a Web Worker because the process.dlopen function is not thread safe.

The only way to load a native module safely for now, is to make sure the app loads no native modules after the Web Workers get started.

process.dlopen = () => {

**throw** **new** Error('Load native module is not safe')

}

**let** worker = **new** Worker('script.js')

Notifications (Windows, Linux, macOS)

All three operating systems provide means for applications to send notifications to the user. Electron conveniently allows developers to send notifications with the [HTML5 Notification API](https://notifications.spec.whatwg.org/), using the currently running operating system’s native notification APIs to display it.

**Note:** Since this is an HTML5 API it is only available in the renderer process. If you want to show Notifications in the main process please check out the [Notification](https://electron.atom.io/docs/api/notification) module.

**let** myNotification = **new** Notification('Title', {

body: 'Lorem Ipsum Dolor Sit Amet'

})

myNotification.onclick = () => {

console.log('Notification clicked')

}

While code and user experience across operating systems are similar, there are subtle differences.

Windows

* On Windows 10, notifications “just work”.
* On Windows 8.1 and Windows 8, a shortcut to your app, with an [Application User Model ID][app-user-model-id], must be installed to the Start screen. Note, however, that it does not need to be pinned to the Start screen.
* On Windows 7, notifications work via a custom implementation which visually resembles the native one on newer systems.

Furthermore, in Windows 8, the maximum length for the notification body is 250 characters, with the Windows team recommending that notifications should be kept to 200 characters. That said, that limitation has been removed in Windows 10, with the Windows team asking developers to be reasonable. Attempting to send gigantic amounts of text to the API (thousands of characters) might result in instability.

Advanced Notifications

Later versions of Windows allow for advanced notifications, with custom templates, images, and other flexible elements. To send those notifications (from either the main process or the renderer process), use the userland module [electron-windows-notifications](https://github.com/felixrieseberg/electron-windows-notifications), which uses native Node addons to send ToastNotification and TileNotification objects.

While notifications including buttons work with just electron-windows-notifications, handling replies requires the use of [electron-windows-interactive-notifications](https://github.com/felixrieseberg/electron-windows-interactive-notifications), which helps with registering the required COM components and calling your Electron app with the entered user data.

Quiet Hours / Presentation Mode

To detect whether or not you’re allowed to send a notification, use the userland module [electron-notification-state](https://github.com/felixrieseberg/electron-notification-state).

This allows you to determine ahead of time whether or not Windows will silently throw the notification away.

macOS

Notifications are straight-forward on macOS, but you should be aware of [Apple’s Human Interface guidelines regarding notifications](https://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/OSXHIGuidelines/NotificationCenter.html).

Note that notifications are limited to 256 bytes in size and will be truncated if you exceed that limit.

Advanced Notifications

Later versions of macOS allow for notifications with an input field, allowing the user to quickly reply to a notification. In order to send notifications with an input field, use the userland module [node-mac-notifier](https://github.com/CharlieHess/node-mac-notifier).

Do not disturb / Session State

To detect whether or not you’re allowed to send a notification, use the userland module [electron-notification-state](https://github.com/felixrieseberg/electron-notification-state).

This will allow you to detect ahead of time whether or not the notification will be displayed.

Linux

Notifications are sent using libnotify which can show notifications on any desktop environment that follows [Desktop Notifications Specification][notification-spec], including Cinnamon, Enlightenment, Unity, GNOME, KDE.

Offscreen Rendering

Offscreen rendering lets you obtain the content of a browser window in a bitmap, so it can be rendered anywhere, for example on a texture in a 3D scene. The offscreen rendering in Electron uses a similar approach than the [Chromium Embedded Framework](https://bitbucket.org/chromiumembedded/cef) project.

Two modes of rendering can be used and only the dirty area is passed in the 'paint' event to be more efficient. The rendering can be stopped, continued and the frame rate can be set. The specified frame rate is a top limit value, when there is nothing happening on a webpage, no frames are generated. The maximum frame rate is 60, because above that there is no benefit, just performance loss.

**Note:** An offscreen window is always created as a [Frameless Window](https://electron.atom.io/docs/api/frameless-window).

Two modes of rendering

GPU accelerated

GPU accelerated rendering means that the GPU is used for composition. Because of that the frame has to be copied from the GPU which requires more performance, thus this mode is quite a bit slower than the other one. The benefit of this mode that WebGL and 3D CSS animations are supported.

Software output device

This mode uses a software output device for rendering in the CPU, so the frame generation is much faster, thus this mode is preferred over the GPU accelerated one.

To enable this mode GPU acceleration has to be disabled by calling the [app.disableHardwareAcceleration()](https://electron.atom.io/docs/api/app#appdisablehardwareacceleration) API.

Usage

**const** {app, BrowserWindow} = require('electron')

app.disableHardwareAcceleration()

**let** win

app.once('ready', () => {

win = **new** BrowserWindow({

webPreferences: {

offscreen: true

}

})

win.loadURL('http://github.com')

win.webContents.on('paint', (event, dirty, image) => {

*// updateBitmap(dirty, image.getBitmap())*

})

win.webContents.setFrameRate(30)

})

Online/Offline Event Detection

Online and offline event detection can be implemented in the renderer process using standard HTML5 APIs, as shown in the following example.

*main.js*

**const** {app, BrowserWindow} = require('electron')

**let** onlineStatusWindow

app.on('ready', () => {

onlineStatusWindow = **new** BrowserWindow({ width: 0, height: 0, show: false })

onlineStatusWindow.loadURL(`file:*//${\_\_dirname}/online-status.html`)*

})

*online-status.html*

***<!DOCTYPE html>***

<html>

<body>

<script>

**const** alertOnlineStatus = () => {

window.alert(navigator.onLine ? 'online' : 'offline')

}

window.addEventListener('online', alertOnlineStatus)

window.addEventListener('offline', alertOnlineStatus)

alertOnlineStatus()

</script>

</body>

</html>

There may be instances where you want to respond to these events in the main process as well. The main process however does not have a navigator object and thus cannot detect these events directly. Using Electron’s inter-process communication utilities, the events can be forwarded to the main process and handled as needed, as shown in the following example.

*main.js*

**const** {app, BrowserWindow, ipcMain} = require('electron')

**let** onlineStatusWindow

app.on('ready', () => {

onlineStatusWindow = **new** BrowserWindow({ width: 0, height: 0, show: false })

onlineStatusWindow.loadURL(`file:*//${\_\_dirname}/online-status.html`)*

})

ipcMain.on('online-status-changed', (event, status) => {

console.log(status)

})

*online-status.html*

***<!DOCTYPE html>***

<html>

<body>

<script>

**const** {ipcRenderer} = require('electron')

**const** updateOnlineStatus = () => {

ipcRenderer.send('online-status-changed', navigator.onLine ? 'online' : 'offline')

}

window.addEventListener('online', updateOnlineStatus)

window.addEventListener('offline', updateOnlineStatus)

updateOnlineStatus()

</script>

</body>

</html>

**NOTE:** If Electron is not able to connect to a local area network (LAN) or a router, it is considered offline; all other conditions return true. So while you can assume that Electron is offline when navigator.onLine returns a false value, you cannot assume that a true value necessarily means that Electron can access the internet. You could be getting false positives, such as in cases where the computer is running a virtualization software that has virtual ethernet adapters that are always “connected.” Therefore, if you really want to determine the internet access status of Electron, you should develop additional means for checking.

Planned Breaking API Changes

The following list includes the APIs that will be removed in Electron 2.0.

There is no timetable for when this release will occur but deprecation warnings will be added at least 90 days beforehand.

app

*// Deprecated*

app.getAppMemoryInfo()

*// Replace with*

app.getAppMetrics()

BrowserWindow

*// Deprecated*

**let** optionsA = {webPreferences: {blinkFeatures: ''}}

**let** windowA = **new** BrowserWindow(optionsA)

*// Replace with*

**let** optionsB = {webPreferences: {enableBlinkFeatures: ''}}

**let** windowB = **new** BrowserWindow(optionsB)

*// Deprecated*

**let** optionsA = {titleBarStyle: 'hidden-inset'}

**let** windowA = **new** BrowserWindow(optionsA)

*// Replace with*

**let** optionsB = {titleBarStyle: 'hiddenInset'}

**let** windowB = **new** BrowserWindow(optionsB)

clipboard

*// Deprecated*

clipboard.readRtf()

*// Replace with*

clipboard.readRTF()

*// Deprecated*

clipboard.writeRtf()

*// Replace with*

clipboard.writeRTF()

*// Deprecated*

clipboard.readHtml()

*// Replace with*

clipboard.readHTML()

*// Deprecated*

clipboard.writeHtml()

*// Replace with*

clipboard.writeHTML()

crashReporter

*// Deprecated*

crashReporter.start({

companyName: 'Crashly',

submitURL: 'https://crash.server.com',

autoSubmit: true

})

*// Replace with*

crashReporter.start({

companyName: 'Crashly',

submitURL: 'https://crash.server.com',

uploadToServer: true

})

menu

*// Deprecated*

menu.popup(browserWindow, 100, 200, 2)

*// Replace with*

menu.popup(browserWindow, {x: 100, y: 200, positioningItem: 2})

nativeImage

*// Deprecated*

nativeImage.toPng()

*// Replace with*

nativeImage.toPNG()

*// Deprecated*

nativeImage.toJpeg()

*// Replace with*

nativeImage.toJPEG()

*// Deprecated*

nativeImage.createFromBuffer(buffer, 1.0)

*// Replace with*

nativeImage.createFromBuffer(buffer, {

scaleFactor: 1.0

})

process

*// Deprecated*

process.versions['atom-shell']

*// Replace with*

process.versions.electron

* process.versions.electron and process.version.chrome will be made read-only properties for consistency with the other process.versions properties set by Node.

session

*// Deprecated*

ses.setCertificateVerifyProc(**function** (hostname, certificate, callback) {

callback(true)

})

*// Replace with*

ses.setCertificateVerifyProc(**function** (request, callback) {

callback(0)

})

Tray

*// Deprecated*

tray.setHighlightMode(true)

*// Replace with*

tray.setHighlightMode('on')

*// Deprecated*

tray.setHighlightMode(false)

*// Replace with*

tray.setHighlightMode('off')

webContents

*// Deprecated*

webContents.openDevTools({detach: true})

*// Replace with*

webContents.openDevTools({mode: 'detach'})

*// Deprecated*

webContents.setZoomLevelLimits(1, 2)

*// Replace with*

webContents.setVisualZoomLevelLimits(1, 2)

webFrame

*// Deprecated*

webFrame.setZoomLevelLimits(1, 2)

*// Replace with*

webFrame.setVisualZoomLevelLimits(1, 2)

*// Deprecated*

webFrame.registerURLSchemeAsSecure('app')

*// Replace with*

protocol.registerStandardSchemes(['app'], {secure: true})

*// Deprecated*

webFrame.registerURLSchemeAsPrivileged('app', {secure: true})

*// Replace with*

protocol.registerStandardSchemes(['app'], {secure: true})

<webview>

*// Deprecated*

webview.setZoomLevelLimits(1, 2)

*// Replace with*

webview.setVisualZoomLevelLimits(1, 2)

Node Headers URL

This is the URL specified as disturl in a .npmrc file or as the --dist-url command line flag when building native Node modules.

Deprecated: https://atom.io/download/atom-shell

Replace with: https://atom.io/download/electron

Duplicate ARM Assets

Each Electron release includes two identical ARM builds with slightly different filenames, like electron-v1.7.3-linux-arm.zip and electron-v1.7.3-linux-armv7l.zip. The asset with the v7l prefix was added to clarify to users which ARM version it supports, and to disambiguate it from future armv6l and arm64 assets that may be produced.

The file *without the prefix* is still being published to avoid breaking any setups that may be consuming it. Starting at 2.0, the un-prefixed file will no longer be published.

For details, see [6986](https://github.com/electron/electron/pull/6986) and [7189](https://github.com/electron/electron/pull/7189).

FIXME comments

The FIXME string is used in code comments to denote things that should be fixed for the 2.0 release. See https://github.com/electron/electron/search?q=fixme

Quick Start

Electron enables you to create desktop applications with pure JavaScript by providing a runtime with rich native (operating system) APIs. You could see it as a variant of the Node.js runtime that is focused on desktop applications instead of web servers.

This doesn’t mean Electron is a JavaScript binding to graphical user interface (GUI) libraries. Instead, Electron uses web pages as its GUI, so you could also see it as a minimal Chromium browser, controlled by JavaScript.

Main Process

In Electron, the process that runs package.json’s main script is called **the main process**. The script that runs in the main process can display a GUI by creating web pages.

Renderer Process

Since Electron uses Chromium for displaying web pages, Chromium’s multi-process architecture is also used. Each web page in Electron runs in its own process, which is called **the renderer process**.

In normal browsers, web pages usually run in a sandboxed environment and are not allowed access to native resources. Electron users, however, have the power to use Node.js APIs in web pages allowing lower level operating system interactions.

Differences Between Main Process and Renderer Process

The main process creates web pages by creating BrowserWindow instances. Each BrowserWindow instance runs the web page in its own renderer process. When a BrowserWindow instance is destroyed, the corresponding renderer process is also terminated.

The main process manages all web pages and their corresponding renderer processes. Each renderer process is isolated and only cares about the web page running in it.

In web pages, calling native GUI related APIs is not allowed because managing native GUI resources in web pages is very dangerous and it is easy to leak resources. If you want to perform GUI operations in a web page, the renderer process of the web page must communicate with the main process to request that the main process perform those operations.

In Electron, we have several ways to communicate between the main process and renderer processes. Like [ipcRenderer](https://electron.atom.io/docs/api/ipc-renderer) and [ipcMain](https://electron.atom.io/docs/api/ipc-main) modules for sending messages, and the [remote](https://electron.atom.io/docs/api/remote) module for RPC style communication. There is also an FAQ entry on [how to share data between web pages](https://electron.atom.io/docs/faq#how-to-share-data-between-web-pages).

Write your First Electron App

Generally, an Electron app is structured like this:

your-app/

├── package.json

├── main.js

└── index.html

The format of package.json is exactly the same as that of Node’s modules, and the script specified by the mainfield is the startup script of your app, which will run the main process. An example of your package.json might look like this:

{

"name" : "your-app",

"version" : "0.1.0",

"main" : "main.js"

}

**Note**: If the main field is not present in package.json, Electron will attempt to load an index.js.

The main.js should create windows and handle system events, a typical example being:

**const** {app, BrowserWindow} = require('electron')

**const** path = require('path')

**const** url = require('url')

*// Keep a global reference of the window object, if you don't, the window will*

*// be closed automatically when the JavaScript object is garbage collected.*

**let** win

**function** createWindow () {

*// Create the browser window.*

win = **new** BrowserWindow({width: 800, height: 600})

*// and load the index.html of the app.*

win.loadURL(url.format({

pathname: path.join(\_\_dirname, 'index.html'),

protocol: 'file:',

slashes: true

}))

*// Open the DevTools.*

win.webContents.openDevTools()

*// Emitted when the window is closed.*

win.on('closed', () => {

*// Dereference the window object, usually you would store windows*

*// in an array if your app supports multi windows, this is the time*

*// when you should delete the corresponding element.*

win = null

})

}

*// This method will be called when Electron has finished*

*// initialization and is ready to create browser windows.*

*// Some APIs can only be used after this event occurs.*

app.on('ready', createWindow)

*// Quit when all windows are closed.*

app.on('window-all-closed', () => {

*// On macOS it is common for applications and their menu bar*

*// to stay active until the user quits explicitly with Cmd + Q*

**if** (process.platform !== 'darwin') {

app.quit()

}

})

app.on('activate', () => {

*// On macOS it's common to re-create a window in the app when the*

*// dock icon is clicked and there are no other windows open.*

**if** (win === null) {

createWindow()

}

})

*// In this file you can include the rest of your app's specific main process*

*// code. You can also put them in separate files and require them here.*

Finally the index.html is the web page you want to show:

***<!DOCTYPE html>***

<html>

<head>

<meta charset="UTF-8">

<title>Hello World!</title>

</head>

<body>

<h1>Hello World!</h1>

We are using node <script>document.write(process.versions.node)</script>,

Chrome <script>document.write(process.versions.chrome)</script>,

and Electron <script>document.write(process.versions.electron)</script>.

</body>

</html>

Run your app

Once you’ve created your initial main.js, index.html, and package.json files, you’ll probably want to try running your app locally to test it and make sure it’s working as expected.

electron

[electron](https://github.com/electron-userland/electron-prebuilt) is an npm module that contains pre-compiled versions of Electron.

If you’ve installed it globally with npm, then you will only need to run the following in your app’s source directory:

electron .

If you’ve installed it locally, then run:

macOS / Linux

$ ./node\_modules/.bin/electron .

Windows

$ .\node\_modules\.bin\electron .

Manually Downloaded Electron Binary

If you downloaded Electron manually, you can also use the included binary to execute your app directly.

macOS

$ ./Electron.app/Contents/MacOS/Electron your-app/

Linux

$ ./electron/electron your-app/

Windows

$ .\electron\electron.exe your-app\

Electron.app here is part of the Electron’s release package, you can download it from [here](https://github.com/electron/electron/releases).

Run as a distribution

After you’re done writing your app, you can create a distribution by following the [Application Distribution](https://electron.atom.io/docs/tutorial/application-distribution) guide and then executing the packaged app.

Try this Example

Clone and run the code in this tutorial by using the [electron/electron-quick-start](https://github.com/electron/electron-quick-start) repository.

**Note**: Running this requires [Git](https://git-scm.com/) and [Node.js](https://nodejs.org/en/download/) (which includes [npm](https://npmjs.org/)) on your system.

*# Clone the repository*

$ git clone https://github.com/electron/electron-quick-start

*# Go into the repository*

$ cd electron-quick-start

*# Install dependencies*

$ npm install

*# Run the app*

$ npm start

For more example apps, see the [list of boilerplates](https://electron.atom.io/community/#boilerplates) created by the awesome electron community.

REPL

Read-Eval-Print-Loop (REPL) is a simple, interactive computer programming environment that takes single user inputs (i.e. single expressions), evaluates them, and returns the result to the user.

The repl module provides a REPL implementation that can be accessed using:

* Assuming you have electron or electron-prebuilt installed as a local project dependency:
* ./node\_modules/.bin/electron --interactive
* Assuming you have electron or electron-prebuilt installed globally:
* electron --interactive

This only creates a REPL for the main process. You can use the Console tab of the Dev Tools to get a REPL for the renderer processes.

**Note:** electron --interactive is not available on Windows.

More information can be found in the [Node.js REPL docs](https://nodejs.org/dist/latest/docs/api/repl.html).

Security, Native Capabilities, and Your Responsibility

As web developers, we usually enjoy the strong security net of the browser - the risks associated with the code we write are relatively small. Our websites are granted limited powers in a sandbox, and we trust that our users enjoy a browser built by a large team of engineers that is able to quickly respond to newly discovered security threats.

When working with Electron, it is important to understand that Electron is not a web browser. It allows you to build feature-rich desktop applications with familiar web technologies, but your code wields much greater power. JavaScript can access the filesystem, user shell, and more. This allows you to build high quality native applications, but the inherent security risks scale with the additional powers granted to your code.

With that in mind, be aware that displaying arbitrary content from untrusted sources poses a severe security risk that Electron is not intended to handle. In fact, the most popular Electron apps (Atom, Slack, Visual Studio Code, etc) display primarily local content (or trusted, secure remote content without Node integration) – if your application executes code from an online source, it is your responsibility to ensure that the code is not malicious.

Reporting Security Issues

For information on how to properly disclose an Electron vulnerability, see [SECURITY.md](https://github.com/electron/electron/tree/master/SECURITY.md)

Chromium Security Issues and Upgrades

While Electron strives to support new versions of Chromium as soon as possible, developers should be aware that upgrading is a serious undertaking - involving hand-editing dozens or even hundreds of files. Given the resources and contributions available today, Electron will often not be on the very latest version of Chromium, lagging behind by either days or weeks.

We feel that our current system of updating the Chromium component strikes an appropriate balance between the resources we have available and the needs of the majority of applications built on top of the framework. We definitely are interested in hearing more about specific use cases from the people that build things on top of Electron. Pull requests and contributions supporting this effort are always very welcome.

Ignoring Above Advice

A security issue exists whenever you receive code from a remote destination and execute it locally. As an example, consider a remote website being displayed inside a browser window. If an attacker somehow manages to change said content (either by attacking the source directly, or by sitting between your app and the actual destination), they will be able to execute native code on the user’s machine.

 Under no circumstances should you load and execute remote code with Node integration enabled. Instead, use only local files (packaged together with your application) to execute Node code. To display remote content, use the webview tag and make sure to disable the nodeIntegration.

Checklist

This is not bulletproof, but at the least, you should attempt the following:

* Only display secure (https) content
* Disable the Node integration in all renderers that display remote content (setting nodeIntegration to falsein webPreferences)
* Enable context isolation in all renderers that display remote content (setting contextIsolation to true in webPreferences)
* Use ses.setPermissionRequestHandler() in all sessions that load remote content
* Do not disable webSecurity. Disabling it will disable the same-origin policy.
* Define a [Content-Security-Policy](http://www.html5rocks.com/en/tutorials/security/content-security-policy/) , and use restrictive rules (i.e. script-src 'self')
* [Override and disable eval](https://github.com/nylas/N1/blob/0abc5d5defcdb057120d726b271933425b75b415/static/index.js#L6-L8) , which allows strings to be executed as code.
* Do not set allowRunningInsecureContent to true.
* Do not enable experimentalFeatures or experimentalCanvasFeatures unless you know what you’re doing.
* Do not use blinkFeatures unless you know what you’re doing.
* WebViews: Do not add the nodeintegration attribute.
* WebViews: Do not use disablewebsecurity
* WebViews: Do not use allowpopups
* WebViews: Do not use insertCSS or executeJavaScript with remote CSS/JS.
* WebViews: Verify the options and params of all <webview> tags before they get attached using the will-attach-webview event:

app.on('web-contents-created', (event, contents) => {

contents.on('will-attach-webview', (event, webPreferences, params) => {

*// Strip away preload scripts if unused or verify their location is legitimate*

**delete** webPreferences.preload

**delete** webPreferences.preloadURL

*// Disable node integration*

webPreferences.nodeIntegration = false

*// Verify URL being loaded*

**if** (!params.src.startsWith('https://yourapp.com/')) {

event.preventDefault()

}

})

})

Again, this list merely minimizes the risk, it does not remove it. If your goal is to display a website, a browser will be a more secure option.

Supported Platforms

Following platforms are supported by Electron:

macOS

Only 64bit binaries are provided for macOS, and the minimum macOS version supported is macOS 10.9.

Windows

Windows 7 and later are supported, older operating systems are not supported (and do not work).

Both ia32 (x86) and x64 (amd64) binaries are provided for Windows. Please note, the ARM version of Windows is not supported for now.

Linux

The prebuilt ia32 (i686) and x64 (amd64) binaries of Electron are built on Ubuntu 12.04, the arm binary is built against ARM v7 with hard-float ABI and NEON for Debian Wheezy.

Whether the prebuilt binary can run on a distribution depends on whether the distribution includes the libraries that Electron is linked to on the building platform, so only Ubuntu 12.04 is guaranteed to work, but following platforms are also verified to be able to run the prebuilt binaries of Electron:

* Ubuntu 12.04 and later
* Fedora 21
* Debian 8

Testing on Headless CI Systems (Travis CI, Jenkins)

Being based on Chromium, Electron requires a display driver to function. If Chromium can’t find a display driver, Electron will simply fail to launch - and therefore not executing any of your tests, regardless of how you are running them. Testing Electron-based apps on Travis, Circle, Jenkins or similar Systems requires therefore a little bit of configuration. In essence, we need to use a virtual display driver.

Configuring the Virtual Display Server

First, install [Xvfb](https://en.wikipedia.org/wiki/Xvfb). It’s a virtual framebuffer, implementing the X11 display server protocol - it performs all graphical operations in memory without showing any screen output, which is exactly what we need.

Then, create a virtual xvfb screen and export an environment variable called DISPLAY that points to it. Chromium in Electron will automatically look for $DISPLAY, so no further configuration of your app is required. This step can be automated with Paul Betts’s [xvfb-maybe](https://github.com/paulcbetts/xvfb-maybe): Prepend your test commands with xvfb-maybe and the little tool will automatically configure xvfb, if required by the current system. On Windows or macOS, it will simply do nothing.

## On Windows or macOS, this just invokes electron-mocha

## On Linux, if we are in a headless environment, this will be equivalent

## to xvfb-run electron-mocha ./test/\*.js

xvfb-maybe electron-mocha ./test/\*.js

Travis CI

On Travis, your .travis.yml should look roughly like this:

addons:

apt:

packages:

- xvfb

install:

- export DISPLAY=':99.0'

- Xvfb :99 -screen 0 1024x768x24 > /dev/null 2>&1 &

Jenkins

For Jenkins, a [Xvfb plugin is available](https://wiki.jenkins-ci.org/display/JENKINS/Xvfb+Plugin).

Circle CI

Circle CI is awesome and has xvfb and $DISPLAY [already setup, so no further configuration is required](https://circleci.com/docs/environment#browsers).

AppVeyor

AppVeyor runs on Windows, supporting Selenium, Chromium, Electron and similar tools out of the box - no configuration is required.

Using Native Node Modules

The native Node modules are supported by Electron, but since Electron is very likely to use a different V8 version from the Node binary installed in your system, you have to manually specify the location of Electron’s headers when building native modules.

How to install native modules

Three ways to install native modules:

Using npm

By setting a few environment variables, you can use npm to install modules directly.

An example of installing all dependencies for Electron:

*# Electron's version.*

export npm\_config\_target=1.2.3

*# The architecture of Electron, can be ia32 or x64.*

export npm\_config\_arch=x64

export npm\_config\_target\_arch=x64

*# Download headers for Electron.*

export npm\_config\_disturl=https://atom.io/download/electron

*# Tell node-pre-gyp that we are building for Electron.*

export npm\_config\_runtime=electron

*# Tell node-pre-gyp to build module from source code.*

export npm\_config\_build\_from\_source=true

*# Install all dependencies, and store cache to ~/.electron-gyp.*

HOME=~/.electron-gyp npm install

Installing modules and rebuilding for Electron

You can also choose to install modules like other Node projects, and then rebuild the modules for Electron with the [electron-rebuild](https://github.com/paulcbetts/electron-rebuild) package. This module can get the version of Electron and handle the manual steps of downloading headers and building native modules for your app.

An example of installing electron-rebuild and then rebuild modules with it:

npm install --save-dev electron-rebuild

*# Every time you run "npm install", run this:*

./node\_modules/.bin/electron-rebuild

*# On Windows if you have trouble, try:*

.\node\_modules\.bin\electron-rebuild.cmd

Manually building for Electron

If you are a developer developing a native module and want to test it against Electron, you might want to rebuild the module for Electron manually. You can use node-gyp directly to build for Electron:

cd /path-to-module/

HOME=~/.electron-gyp node-gyp rebuild --target=1.2.3 --arch=x64 --dist-url=https://atom.io/download/electron

The HOME=~/.electron-gyp changes where to find development headers. The --target=1.2.3 is version of Electron. The --dist-url=... specifies where to download the headers. The --arch=x64 says the module is built for 64bit system.

Troubleshooting

If you installed a native module and found it was not working, you need to check following things:

* The architecture of module has to match Electron’s architecture (ia32 or x64).
* After you upgraded Electron, you usually need to rebuild the modules.
* When in doubt, run electron-rebuild first.

Modules that rely on prebuild

[prebuild](https://github.com/mafintosh/prebuild) provides a way to easily publish native Node modules with prebuilt binaries for multiple versions of Node and Electron.

If modules provide binaries for the usage in Electron, make sure to omit --build-from-source and the npm\_config\_build\_from\_source environment variable in order to take full advantage of the prebuilt binaries.

Modules that rely on node-pre-gyp

The [node-pre-gyp tool](https://github.com/mapbox/node-pre-gyp) provides a way to deploy native Node modules with prebuilt binaries, and many popular modules are using it.

Usually those modules work fine under Electron, but sometimes when Electron uses a newer version of V8 than Node, and there are ABI changes, bad things may happen. So in general it is recommended to always build native modules from source code.

If you are following the npm way of installing modules, then this is done by default, if not, you have to pass --build-from-source to npm, or set the npm\_config\_build\_from\_source environment variable.

Using Pepper Flash Plugin

Electron supports the Pepper Flash plugin. To use the Pepper Flash plugin in Electron, you should manually specify the location of the Pepper Flash plugin and then enable it in your application.

Prepare a Copy of Flash Plugin

On macOS and Linux, the details of the Pepper Flash plugin can be found by navigating to chrome://plugins in the Chrome browser. Its location and version are useful for Electron’s Pepper Flash support. You can also copy it to another location.

Add Electron Switch

You can directly add --ppapi-flash-path and --ppapi-flash-version to the Electron command line or by using the app.commandLine.appendSwitch method before the app ready event. Also, turn on plugins option of BrowserWindow.

For example:

**const** {app, BrowserWindow} = require('electron')

**const** path = require('path')

*// Specify flash path, supposing it is placed in the same directory with main.js.*

**let** pluginName

**switch** (process.platform) {

**case** 'win32':

pluginName = 'pepflashplayer.dll'

**break**

**case** 'darwin':

pluginName = 'PepperFlashPlayer.plugin'

**break**

**case** 'linux':

pluginName = 'libpepflashplayer.so'

**break**

}

app.commandLine.appendSwitch('ppapi-flash-path', path.join(\_\_dirname, pluginName))

*// Optional: Specify flash version, for example, v17.0.0.169*

app.commandLine.appendSwitch('ppapi-flash-version', '17.0.0.169')

app.on('ready', () => {

**let** win = **new** BrowserWindow({

width: 800,

height: 600,

webPreferences: {

plugins: true

}

})

win.loadURL(`file:*//${\_\_dirname}/index.html`)*

*// Something else*

})

You can also try loading the system wide Pepper Flash plugin instead of shipping the plugins yourself, its path can be received by calling app.getPath('pepperFlashSystemPlugin').

Enable Flash Plugin in a <webview> Tag

Add plugins attribute to <webview> tag.

<webview src="http://www.adobe.com/software/flash/about/" plugins></webview>

Troubleshooting

You can check if Pepper Flash plugin was loaded by inspecting navigator.plugins in the console of devtools (although you can’t know if the plugin’s path is correct).

The architecture of Pepper Flash plugin has to match Electron’s one. On Windows, a common error is to use 32bit version of Flash plugin against 64bit version of Electron.

On Windows the path passed to --ppapi-flash-path has to use \ as path delimiter, using POSIX-style paths will not work.

For some operations, such as streaming media using RTMP, it is necessary to grant wider permissions to players’ .swf files. One way of accomplishing this, is to use [nw-flash-trust](https://github.com/szwacz/nw-flash-trust).

Using Selenium and WebDriver

From [ChromeDriver - WebDriver for Chrome](https://sites.google.com/a/chromium.org/chromedriver/):

WebDriver is an open source tool for automated testing of web apps across many browsers. It provides capabilities for navigating to web pages, user input, JavaScript execution, and more. ChromeDriver is a standalone server which implements WebDriver’s wire protocol for Chromium. It is being developed by members of the Chromium and WebDriver teams.

Setting up Spectron

[Spectron](https://electron.atom.io/spectron) is the officially supported ChromeDriver testing framework for Electron. It is built on top of [WebdriverIO](http://webdriver.io/)and has helpers to access Electron APIs in your tests and bundles ChromeDriver.

$ npm install --save-dev spectron

*// A simple test to verify a visible window is opened with a title*

**var** Application = require('spectron').Application

**var** assert = require('assert')

**var** app = **new** Application({

path: '/Applications/MyApp.app/Contents/MacOS/MyApp'

})

app.start().then(**function** () {

*// Check if the window is visible*

**return** app.browserWindow.isVisible()

}).then(**function** (isVisible) {

*// Verify the window is visible*

assert.equal(isVisible, true)

}).then(**function** () {

*// Get the window's title*

**return** app.client.getTitle()

}).then(**function** (title) {

*// Verify the window's title*

assert.equal(title, 'My App')

}).**catch**(**function** (error) {

*// Log any failures*

console.error('Test failed', error.message)

}).then(**function** () {

*// Stop the application*

**return** app.stop()

})

Setting up with WebDriverJs

[WebDriverJs](https://code.google.com/p/selenium/wiki/WebDriverJs) provides a Node package for testing with web driver, we will use it as an example.

1. Start ChromeDriver

First you need to download the chromedriver binary, and run it:

$ npm install electron-chromedriver

$ ./node\_modules/.bin/chromedriver

Starting ChromeDriver (v2.10.291558) on port 9515

Only local connections are allowed.

Remember the port number 9515, which will be used later

2. Install WebDriverJS

$ npm install selenium-webdriver

3. Connect to ChromeDriver

The usage of selenium-webdriver with Electron is basically the same with upstream, except that you have to manually specify how to connect chrome driver and where to find Electron’s binary:

**const** webdriver = require('selenium-webdriver')

**const** driver = **new** webdriver.Builder()

*// The "9515" is the port opened by chrome driver.*

.usingServer('http://localhost:9515')

.withCapabilities({

chromeOptions: {

*// Here is the path to your Electron binary.*

binary: '/Path-to-Your-App.app/Contents/MacOS/Electron'

}

})

.forBrowser('electron')

.build()

driver.get('http://www.google.com')

driver.findElement(webdriver.By.name('q')).sendKeys('webdriver')

driver.findElement(webdriver.By.name('btnG')).click()

driver.wait(() => {

**return** driver.getTitle().then((title) => {

**return** title === 'webdriver - Google Search'

})

}, 1000)

driver.quit()

Setting up with WebdriverIO

[WebdriverIO](http://webdriver.io/) provides a Node package for testing with web driver.

1. Start ChromeDriver

First you need to download the chromedriver binary, and run it:

$ npm install electron-chromedriver

$ ./node\_modules/.bin/chromedriver --url-base=wd/hub --port=9515

Starting ChromeDriver (v2.10.291558) on port 9515

Only local connections are allowed.

Remember the port number 9515, which will be used later

2. Install WebdriverIO

$ npm install webdriverio

3. Connect to chrome driver

**const** webdriverio = require('webdriverio')

**const** options = {

host: 'localhost', *// Use localhost as chrome driver server*

port: 9515, *// "9515" is the port opened by chrome driver.*

desiredCapabilities: {

browserName: 'chrome',

chromeOptions: {

binary: '/Path-to-Your-App/electron', *// Path to your Electron binary.*

args: [*/\* cli arguments \*/*] *// Optional, perhaps 'app=' + /path/to/your/app/*

}

}

}

**let** client = webdriverio.remote(options)

client

.init()

.url('http://google.com')

.setValue('#q', 'webdriverio')

.click('#btnG')

.getTitle().then((title) => {

console.log('Title was: ' + title)

})

.end()

Workflow

To test your application without rebuilding Electron, simply [place](https://github.com/electron/electron/blob/master/docs/tutorial/application-distribution.md) your app source into Electron’s resource directory.

Alternatively, pass an argument to run with your electron binary that points to your app’s folder. This eliminates the need to copy-paste your app into Electron’s resource directory.

Using Widevine CDM Plugin

In Electron you can use the Widevine CDM plugin shipped with Chrome browser.

Getting the plugin

Electron doesn’t ship with the Widevine CDM plugin for license reasons, to get it, you need to install the official Chrome browser first, which should match the architecture and Chrome version of the Electron build you use.

**Note:** The major version of Chrome browser has to be the same with the Chrome version used by Electron, otherwise the plugin will not work even though navigator.plugins would show it has been loaded.

Windows & macOS

Open chrome://components/ in Chrome browser, find WidevineCdm and make sure it is up to date, then you can find all the plugin binaries from the APP\_DATA/Google/Chrome/WidevineCDM/VERSION/\_platform\_specific/PLATFORM\_ARCH/ directory.

APP\_DATA is system’s location for storing app data, on Windows it is %LOCALAPPDATA%, on macOS it is ~/Library/Application Support. VERSION is Widevine CDM plugin’s version string, like 1.4.8.866. PLATFORM is mac or win. ARCH is x86 or x64.

On Windows the required binaries are widevinecdm.dll and widevinecdmadapter.dll, on macOS they are libwidevinecdm.dylib and widevinecdmadapter.plugin. You can copy them to anywhere you like, but they have to be put together.

Linux

On Linux the plugin binaries are shipped together with Chrome browser, you can find them under /opt/google/chrome, the filenames are libwidevinecdm.so and libwidevinecdmadapter.so.

Using the plugin

After getting the plugin files, you should pass the widevinecdmadapter’s path to Electron with --widevine-cdm-path command line switch, and the plugin’s version with --widevine-cdm-version switch.

**Note:** Though only the widevinecdmadapter binary is passed to Electron, the widevinecdm binary has to be put aside it.

The command line switches have to be passed before the ready event of app module gets emitted, and the page that uses this plugin must have plugin enabled.

Example code:

**const** {app, BrowserWindow} = require('electron')

*// You have to pass the filename of `widevinecdmadapter` here, it is*

*// \* `widevinecdmadapter.plugin` on macOS,*

*// \* `libwidevinecdmadapter.so` on Linux,*

*// \* `widevinecdmadapter.dll` on Windows.*

app.commandLine.appendSwitch('widevine-cdm-path', '/path/to/widevinecdmadapter.plugin')

*// The version of plugin can be got from `chrome://plugins` page in Chrome.*

app.commandLine.appendSwitch('widevine-cdm-version', '1.4.8.866')

**let** win = null

app.on('ready', () => {

win = **new** BrowserWindow({

webPreferences: {

*// The `plugins` have to be enabled.*

plugins: true

}

})

win.show()

})

Verifying the plugin

To verify whether the plugin works, you can use following ways:

* Open devtools and check whether navigator.plugins includes the Widevine CDM plugin.
* Open https://shaka-player-demo.appspot.com/ and load a manifest that uses Widevine.
* Open http://www.dash-player.com/demo/drm-test-area/, check whether the page says bitdash uses Widevine in your browser, then play the video.

Windows Store Guide

With Windows 10, the good old win32 executable got a new sibling: The Universal Windows Platform. The new .appx format does not only enable a number of new powerful APIs like Cortana or Push Notifications, but through the Windows Store, also simplifies installation and updating.

Microsoft [developed a tool that compiles Electron apps as .appx packages](https://github.com/catalystcode/electron-windows-store), enabling developers to use some of the goodies found in the new application model. This guide explains how to use it - and what the capabilities and limitations of an Electron AppX package are.

Background and Requirements

Windows 10 “Anniversary Update” is able to run win32 .exe binaries by launching them together with a virtualized filesystem and registry. Both are created during compilation by running app and installer inside a Windows Container, allowing Windows to identify exactly which modifications to the operating system are done during installation. Pairing the executable with a virtual filesystem and a virtual registry allows Windows to enable one-click installation and uninstallation.

In addition, the exe is launched inside the appx model - meaning that it can use many of the APIs available to the Universal Windows Platform. To gain even more capabilities, an Electron app can pair up with an invisible UWP background task launched together with the exe - sort of launched as a sidekick to run tasks in the background, receive push notifications, or to communicate with other UWP applications.

To compile any existing Electron app, ensure that you have the following requirements:

* Windows 10 with Anniversary Update (released August 2nd, 2016)
* The Windows 10 SDK, [downloadable here](https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk)
* At least Node 4 (to check, run node -v)

Then, go and install the electron-windows-store CLI:

npm install -g electron-windows-store

Step 1: Package Your Electron Application

Package the application using [electron-packager](https://github.com/electron-userland/electron-packager) (or a similar tool). Make sure to remove node\_modules that you don’t need in your final application, since any module you don’t actually need will just increase your application’s size.

The output should look roughly like this:

├── Ghost.exe

├── LICENSE

├── content\_resources\_200\_percent.pak

├── content\_shell.pak

├── d3dcompiler\_47.dll

├── ffmpeg.dll

├── icudtl.dat

├── libEGL.dll

├── libGLESv2.dll

├── locales

│   ├── am.pak

│   ├── ar.pak

│   ├── [...]

├── natives\_blob.bin

├── node.dll

├── resources

│   ├── app

│   └── atom.asar

├── snapshot\_blob.bin

├── squirrel.exe

└── ui\_resources\_200\_percent.pak

Step 2: Running electron-windows-store

From an elevated PowerShell (run it “as Administrator”), run electron-windows-store with the required parameters, passing both the input and output directories, the app’s name and version, and confirmation that node\_modules should be flattened.

electron-windows-store `

--input-directory C:\myelectronapp `

--output-directory C:\output\myelectronapp `

--flatten true `

--package-version 1.0.0.0 `

--package-name myelectronapp

Once executed, the tool goes to work: It accepts your Electron app as an input, flattening the node\_modules. Then, it archives your application as app.zip. Using an installer and a Windows Container, the tool creates an “expanded” AppX package - including the Windows Application Manifest (AppXManifest.xml) as well as the virtual file system and the virtual registry inside your output folder.

Once the expanded AppX files are created, the tool uses the Windows App Packager (MakeAppx.exe) to create a single-file AppX package from those files on disk. Finally, the tool can be used to create a trusted certificate on your computer to sign the new AppX package. With the signed AppX package, the CLI can also automatically install the package on your machine.

Step 3: Using the AppX Package

In order to run your package, your users will need Windows 10 with the so-called “Anniversary Update” - details on how to update Windows can be found [here](https://blogs.windows.com/windowsexperience/2016/08/02/how-to-get-the-windows-10-anniversary-update).

In opposition to traditional UWP apps, packaged apps currently need to undergo a manual verification process, for which you can apply [here](https://developer.microsoft.com/en-us/windows/projects/campaigns/desktop-bridge). In the meantime, all users will be able to just install your package by double-clicking it, so a submission to the store might not be necessary if you’re simply looking for an easier installation method. In managed environments (usually enterprises), the Add-AppxPackage [PowerShell Cmdlet can be used to install it in an automated fashion](https://technet.microsoft.com/en-us/library/hh856048.aspx).

Another important limitation is that the compiled AppX package still contains a win32 executable - and will therefore not run on Xbox, HoloLens, or Phones.

Optional: Add UWP Features using a BackgroundTask

You can pair your Electron app up with an invisible UWP background task that gets to make full use of Windows 10 features - like push notifications, Cortana integration, or live tiles.

To check out how an Electron app that uses a background task to send toast notifications and live tiles, [check out the Microsoft-provided sample](https://github.com/felixrieseberg/electron-uwp-background).

Optional: Convert using Container Virtualization

To generate the AppX package, the electron-windows-store CLI uses a template that should work for most Electron apps. However, if you are using a custom installer, or should you experience any trouble with the generated package, you can attempt to create a package using compilation with a Windows Container - in that mode, the CLI will install and run your application in blank Windows Container to determine what modifications your application is exactly doing to the operating system.

Before running the CLI for the first time, you will have to setup the “Windows Desktop App Converter”. This will take a few minutes, but don’t worry - you only have to do this once. Download and Desktop App Converter from [here](https://www.microsoft.com/en-us/download/details.aspx?id=51691). You will receive two files: DesktopAppConverter.zip and BaseImage-14316.wim.

1. Unzip DesktopAppConverter.zip. From an elevated PowerShell (opened with “run as Administrator”, ensure that your systems execution policy allows us to run everything we intend to run by calling Set-ExecutionPolicy bypass.
2. Then, run the installation of the Desktop App Converter, passing in the location of the Windows base Image (downloaded as BaseImage-14316.wim), by calling .\DesktopAppConverter.ps1 -Setup -BaseImage .\BaseImage-14316.wim.
3. If running the above command prompts you for a reboot, please restart your machine and run the above command again after a successful restart.

Once installation succeeded, you can move on to compiling your Electron app.

[Electron](https://electron.atom.io/) [Docs](https://electron.atom.io/docs/) [Blog](https://electron.atom.io/blog/) [Community](https://electron.atom.io/community/) [Apps](https://electron.atom.io/apps/) [Releases](https://electron.atom.io/releases/) [Discuss](https://discuss.atom.io/c/electron) [Repository](https://github.com/electron/electron)[with  by](https://github.com/)

ss