Window Customization

The `BrowserWindow` module is the foundation of your Electron application, and it exposes many APIs that can change the look and behavior of your browser windows. In this tutorial, we will be going over the various use-cases for window customization on macOS, Windows, and Linux.

Create frameless windows

A frameless window is a window that has no chrome. Not to be confused with the Google Chrome browser, window *chrome* refers to the parts of the window (e.g. toolbars, controls) that are not a part of the web page.

To create a frameless window, you need to set `frame` to `false` in the `BrowserWindow` constructor.

```
const { BrowserWindow } = require('electron')
const win = new BrowserWindow({ frame: false })
```

Apply custom title bar styles *macOS Windows*

Title bar styles allow you to hide most of a BrowserWindow's chrome while keeping the system's native window controls intact and can be configured with the `titleBarStyle` option in the `BrowserWindow` constructor.

Applying the `hidden` title bar style results in a hidden title bar and a full-size content window.

```
const { BrowserWindow } = require('electron')
const win = new BrowserWindow({ titleBarStyle: 'hidden' })
```

Control the traffic lights *macOS*

On macOS, applying the `hidden` title bar style will still expose the standard window controls (“traffic lights”) in the top left.

Customize the look of your traffic lights *macOS*

The `customButtonsOnHover` title bar style will hide the traffic lights until you hover over them. This is useful if you want to create custom traffic lights in your HTML but still use the native UI to control the window.

```
const { BrowserWindow } = require('electron')
const win = new BrowserWindow({ titleBarStyle: 'customButtonsOnHover' })
```

Customize the traffic light position *macOS*

To modify the position of the traffic light window controls, there are two configuration options available.

Applying `hiddenInset` title bar style will shift the vertical inset of the traffic lights by a fixed amount.

```
const { BrowserWindow } = require('electron')
const win = new BrowserWindow({ titleBarStyle: 'hiddenInset' })
```

If you need more granular control over the positioning of the traffic lights, you can pass a set of coordinates to the `trafficLightPosition` option in the `BrowserWindow` constructor.

```
const { BrowserWindow } = require('electron')
const win = new BrowserWindow({
  titleBarStyle: 'hidden',
  trafficLightPosition: { x: 10, y: 10 }
})
```

Show and hide the traffic lights programmatically *macOS*

You can also show and hide the traffic lights programmatically from the main process. The `win.setWindowButtonVisibility` forces traffic lights to be show or hidden depending on the value of its boolean parameter.

```
const { BrowserWindow } = require('electron')
const win = new BrowserWindow()
// hides the traffic lights
win.setWindowButtonVisibility(false)
```

> *Note: Given the number of APIs available, there are many ways of achieving this. For instance, combining `frame: false` with `win.setWindowButtonVisibility(true)` will yield the same layout outcome as setting `titleBarStyle: 'hidden'`.*

Window Controls Overlay *macOS Windows*

The Window Controls Overlay API is a web standard that gives web apps the ability to customize their title bar region when installed on desktop. Electron exposes this API through the `BrowserWindow` constructor option `titleBarOverlay`.

This option only works whenever a custom `titlebarStyle` is applied on macOS or Windows. When `titleBarOverlay` is enabled, the window controls become exposed in their default position, and DOM elements cannot use the area underneath this region.

The `titleBarOverlay` option accepts two different value formats.

Specifying `true` on either platform will result in an overlay region with default system colors:

```
// on macOS or Windows
const { BrowserWindow } = require('electron')
const win = new BrowserWindow({
  titleBarStyle: 'hidden',
  titleBarOverlay: true
})
```

On Windows, you can also specify the color of the overlay and its symbols by setting `titleBarOverlay` to an object with the `color` and `symbolColor` properties. If an option is not specified, the color will default to its system color for the window control buttons:

```
// on Windows
const { BrowserWindow } = require('electron')
const win = new BrowserWindow({
  titleBarStyle: 'hidden',
  titleBarOverlay: {
    color: '#2f3241',
    symbolColor: '#74b1be'
  }
})
```

> *Note: Once your title bar overlay is enabled from the main process, you can access the overlay's color and dimension values from a renderer using a set of readonly [JavaScript APIs] and [CSS Environment Variables].*

Create transparent windows

By setting the `transparent` option to `true`, you can make a fully transparent window.

```
const { BrowserWindow } = require('electron')
const win = new BrowserWindow({ transparent: true })
```

Limitations

- You cannot click through the transparent area. See #1335 for details.
- Transparent windows are not resizable. Setting `resizable` to `true` may make a transparent window stop working on some platforms.
- The CSS `blur()` filter only applies to the window's web contents, 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).
- The window will not be transparent when DevTools is opened.
- On *Windows*:
  - Transparent windows will not work when DWM is disabled.
  - Transparent windows can not be maximized using the Windows system menu or by double clicking the title bar. The reasoning behind this can be seen on PR #28207.
- On *macOS*:
  - The native window shadow will not be shown on a transparent window.

Create click-through windows

To create a click-through window, i.e. making the window ignore all mouse events, you can call the win.setIgnoreMouseEvents(ignore) API:

```
const { BrowserWindow } = require('electron')
const win = new BrowserWindow()
win.setIgnoreMouseEvents(true)
```

Forward mouse events *macOS Windows*

Ignoring mouse messages makes the web contents oblivious to mouse movement, meaning that mouse movement events will not be emitted. On Windows and macOS, an optional parameter can be used to forward mouse move messages to the web page, allowing events such as `mouseleave` to be emitted:

```
const { BrowserWindow, ipcMain } = require('electron')
const path = require('path')
const win = new BrowserWindow({
  webPreferences: {
    preload: path.join(__dirname, 'preload.js')
  }
})

ipcMain.on('set-ignore-mouse-events', (event, ...args) => {
  const win = BrowserWindow.fromWebContents(event.sender)
  win.setIgnoreMouseEvents(...args)
})
```

```
window.addEventListener('DOMContentLoaded', () => {
  const el = document.getElementById('clickThroughElement')
  el.addEventListener('mouseenter', () => {
    ipcRenderer.send('set-ignore-mouse-events', true, { forward: true })
  })
  el.addEventListener('mouseleave', () => {
    ipcRenderer.send('set-ignore-mouse-events', false)
  })
})
```

This makes the web page click-through when over the `#clickThroughElement` element, and returns to normal outside it.

Set custom 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.

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

```
body {
  -webkit-app-region: drag;
}
```

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 only setting a custom titlebar as draggable, you also need to make all buttons in titlebar non-draggable.

Tip: disable text selection

When creating a draggable region, the dragging behavior may conflict with text selection. For example, when you drag the titlebar, you may accidentally select its text contents. To prevent this, you need to disable text selection within a draggable area like this:

```css
.titlebar {
  -webkit-user-select: none;
  -webkit-app-region: drag;
}
```

Tip: disable context menus

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.