BrowserView

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.

Class: BrowserView

> *Create and control views.*

Process: Main

Example

```
// In the main process.
const { app, BrowserView, BrowserWindow } = require('electron')

app.whenReady().then(() => {
  const win = new BrowserWindow({ width: 800, height: 600 })

  const view = new BrowserView()
  win.setBrowserView(view)
  view.setBounds({ x: 0, y: 0, width: 300, height: 300 })
  view.webContents.loadURL('https://electronjs.org')
})
```

`new BrowserView([options])` *Experimental*

- `options` Object (optional)
  - `webPreferences` Object (optional) - See BrowserWindow.

Instance Properties

Objects created with `new BrowserView` have the following properties:

`view.webContents` *Experimental*

A `WebContents` object owned by this view.

Instance Methods

Objects created with `new BrowserView` have the following instance methods:

`view.setAutoResize(options)` *Experimental*

- `options` Object
  - `width` boolean (optional) - If `true`, the view's width will grow and shrink together with the window. `false` by default.
  - `height` boolean (optional) - If `true`, the view's height will grow and shrink together with the window. `false` by default.
  - `horizontal` boolean (optional) - If `true`, the view's x position and width will grow and shrink proportionally with the window. `false` by default.

  - `vertical` boolean (optional) - If `true`, the view's y position and height will grow and shrink proportionally with the window. `false` by default.

`view.setBounds(bounds)` ***Experimental***

- `bounds` [Rectangle]

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

`view.getBounds()` ***Experimental***

Returns `Rectangle`

The `bounds` of this BrowserView instance as `Object`.

`view.setBackgroundColor(color)` ***Experimental***

- `color` string - Color in Hex, RGB, ARGB, HSL, HSLA or named CSS color format. The alpha channel is optional for the hex type.

Examples of valid `color` values:

- Hex
  - #fff (RGB)
  - #ffff (ARGB)
  - #ffffff (RRGGBB)
  - #ffffffff (AARRGGBB)
- RGB
  - rgb(([\d]+),\s*([\d]+),\s*([\d]+))
    - e.g. rgb(255, 255, 255)
- RGBA
  - rgba(([\d]+),\s*([\d]+),\s*([\d]+),\s*([\d.]+))
    - e.g. rgba(255, 255, 255, 1.0)
- HSL
  - hsl((-?[\d.]+),\s*([\d.]+)%,\s*([\d.]+)%)
    - e.g. hsl(200, 20%, 50%)
- HSLA
  - hsla((-?[\d.]+),\s*([\d.]+)%,\s*([\d.]+)%,\s*([\d.]+))
    - e.g. hsla(200, 20%, 50%, 0.5)
- Color name
  - Options are listed in SkParseColor.cpp
  - Similar to CSS Color Module Level 3 keywords, but case-sensitive.
    - e.g. `blueviolet` or `red`

Note: Hex format with alpha takes `AARRGGBB` or `ARGB`, *not* `RRGGBBA` or `RGA`.