webFrame

> Customize the rendering of the current web page.

Process: Renderer

`webFrame` export of the Electron module is an instance of the `WebFrame` class representing the current frame. Sub-frames can be retrieved by certain properties and methods (e.g. `webFrame.firstChild`).

An example of zooming current page to 200%.

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

webFrame.setZoomFactor(2)
```

Methods

The `WebFrame` class has the following instance methods:

`webFrame.setZoomFactor(factor)`

- `factor` Double - Zoom factor; default is 1.0.

Changes the zoom factor to the specified factor. Zoom factor is zoom percent divided by 100, so 300% = 3.0.

The factor must be greater than 0.0.

`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.

> **NOTE**: The zoom policy at the Chromium level is same-origin, meaning that the zoom level for a specific domain propagates across all instances of windows with the same domain. Differentiating the window URLs will make zoom work per-window.

`webFrame.getZoomLevel()`

Returns `number` - The current zoom level.

`webFrame.setVisualZoomLevelLimits(minimumLevel, maximumLevel)`

- `minimumLevel` number
- `maximumLevel` number

Sets the maximum and minimum pinch-to-zoom level.

> **NOTE**: Visual zoom is disabled by default in Electron. To re-enable it, call:
>
> ```
> webFrame.setVisualZoomLevelLimits(1, 3)
> ```
>
> **NOTE**: Visual zoom only applies to pinch-to-zoom behavior. Cmd+/-/0 zoom shortcuts are controlled by the 'zoomIn', 'zoomOut', and 'resetZoom' MenuItem roles in the application Menu. To disable shortcuts, manually define the Menu and omit zoom roles from the definition.

`webFrame.setSpellCheckProvider(language, provider)`

- `language` string
- `provider` Object
  - `spellCheck` Function
    - `words` string[]
    - `callback` Function
      - `misspeltWords` string[]

Sets a provider for spell checking in input fields and text areas.

If you want to use this method you must disable the builtin spellchecker when you construct the window.

```
const mainWindow = new BrowserWindow({
  webPreferences: {
    spellcheck: false
  }
})
```

The `provider` must be an object that has a `spellCheck` method that accepts an array of individual words for spellchecking. The `spellCheck` function runs asynchronously and calls the `callback` function with an array of misspelt words when complete.

An example of using node-spellchecker as provider:

```
const { webFrame } = require('electron')
const spellChecker = require('spellchecker')
webFrame.setSpellCheckProvider('en-US', {
  spellCheck (words, callback) {
    setTimeout(() => {
      const spellchecker = require('spellchecker')
```

```
      const misspelled = words.filter(x => spellchecker.isMisspelled(x))
      callback(misspelled)
    }, 0)
  }
})
```

`webFrame.insertCSS(css[, options])`

- `css` string
- `options` Object (optional)
  - `cssOrigin` string (optional) - Can be either 'user' or 'author'. Sets the cascade origin of the inserted stylesheet. Default is 'author'.

Returns `string` - A key for the inserted CSS that can later be used to remove the CSS via `webFrame.removeInsertedCSS(key)`.

Injects CSS into the current web page and returns a unique key for the inserted stylesheet.

`webFrame.removeInsertedCSS(key)`

- `key` string

Removes the inserted CSS from the current web page. The stylesheet is identified by its key, which is returned from `webFrame.insertCSS(css)`.

`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. Unless the frame is suspended (e.g. showing a modal alert), execution will be synchronous and the callback will be invoked before the method returns. For compatibility with an older version of this method, the error parameter is second.
  - `result` Any
  - `error` Error

Returns `Promise<any>` - A promise that resolves with the result of the executed code or is rejected if execution throws or results in 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.executeJavaScriptInIsolatedWorld(worldId, scripts[, userGesture, callback])`

- `worldId` Integer - The ID of the world to run the javascript in, `0` is the default main world (where content runs), `999` is the world used by Electron's `contextIsolation` feature. Accepts values in the range 1..536870911.
- `scripts` WebSource[]
- `userGesture` boolean (optional) - Default is `false`.
- `callback` Function (optional) - Called after script has been executed. Unless the frame is suspended (e.g. showing a modal alert), execution will be synchronous and the callback will be invoked before the method returns. For compatibility with an older version of this method, the error parameter is second.
  - `result` Any
  - `error` Error

Returns `Promise<any>` - A promise that resolves with the result of the executed code or is rejected if execution could not start.

Works like `executeJavaScript` but evaluates `scripts` in an isolated context.

Note that when the execution of script fails, the returned promise will not reject and the `result` would be `undefined`. This is because Chromium does not dispatch errors of isolated worlds to foreign worlds.

`webFrame.setIsolatedWorldInfo(worldId, info)`

- `worldId` Integer - The ID of the world to run the javascript in, `0` is the default world, `999` is the world used by Electrons `contextIsolation` feature. Chrome extensions reserve the range of IDs in `[1 << 20, 1 << 29)`. You can provide any integer here.
- `info` Object
  - `securityOrigin` string (optional) - Security origin for the isolated world.
  - `csp` string (optional) - Content Security Policy for the isolated world.
  - `name` string (optional) - Name for isolated world. Useful in devtools.

Set the security origin, content security policy and name of the isolated world. Note: If the `csp` is specified, then the `securityOrigin` also has to be specified.

`webFrame.getResourceUsage()`

Returns `Object`:

- `images` MemoryUsageDetails

- `scripts` MemoryUsageDetails
- `cssStyleSheets` MemoryUsageDetails
- `xslStyleSheets` MemoryUsageDetails
- `fonts` MemoryUsageDetails
- `other` MemoryUsageDetails

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).

`webFrame.getFrameForSelector(selector)`

- `selector` string - CSS selector for a frame element.

Returns `WebFrame` - The frame element in `webFrame's` document selected by `selector`, `null` would be returned if `selector` does not select a frame or if the frame is not in the current renderer process.

`webFrame.findFrameByName(name)`

- `name` string

Returns `WebFrame` - A child of `webFrame` with the supplied `name`, `null` would be returned if there's no such frame or if the frame is not in the current renderer process.

`webFrame.findFrameByRoutingId(routingId)`

- `routingId` Integer - An `Integer` representing the unique frame id in the current renderer process. Routing IDs can be retrieved from `WebFrame` instances (`webFrame.routingId`) and are also passed by frame specific `WebContents` navigation events (e.g. `did-frame-navigate`)

Returns `WebFrame` - that has the supplied `routingId`, `null` if not found.

`webFrame.isWordMisspelled(word)`

- `word` string - The word to be spellchecked.

Returns `boolean` - True if the word is misspelled according to the built in spellchecker, false otherwise. If no dictionary is loaded, always return false.

`webFrame.getWordSuggestions(word)`

- `word` string - The misspelled word.

Returns `string[]` - A list of suggested words for a given word. If the word is spelled correctly, the result will be empty.

Properties

`webFrame.top` *Readonly*

A `WebFrame | null` representing top frame in frame hierarchy to which `webFrame` belongs, the property would be `null` if top frame is not in the current renderer process.

`webFrame.opener` *Readonly*

A `WebFrame | null` representing the frame which opened `webFrame`, the property would be `null` if there's no opener or opener is not in the current renderer process.

`webFrame.parent` *Readonly*

A `WebFrame | null` representing parent frame of `webFrame`, the property would be `null` if `webFrame` is top or parent is not in the current renderer process.

`webFrame.firstChild` ***Readonly***

A `WebFrame | null` representing the first child frame of `webFrame`, the property would be `null` if `webFrame` has no children or if first child is not in the current renderer process.

`webFrame.nextSibling` ***Readonly***

A `WebFrame | null` representing next sibling frame, the property would be `null` if `webFrame` is the last frame in its parent or if the next sibling is not in the current renderer process.

`webFrame.routingId` ***Readonly***

An `Integer` representing the unique frame id in the current renderer process. Distinct WebFrame instances that refer to the same underlying frame will have the same `routingId`.