webContents

> *Render and control web pages.*

Process: Main

`webContents` is an EventEmitter. It is responsible for rendering and controlling a web page and is a property of the `BrowserWindow` object. An example of accessing the `webContents` object:

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

const win = new BrowserWindow({ width: 800, height: 1500 })
win.loadURL('http://github.com')

const 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` | null - The web contents that is focused in this application, otherwise returns `null`.

`webContents.fromId(id)`

- `id` Integer

Returns `WebContents` | undefined - A WebContents instance with the given ID, or `undefined` if there is no WebContents associated with the given ID.

`webContents.fromDevToolsTargetId(targetId)`

- `targetId` string - The Chrome DevTools Protocol TargetID associated with the WebContents instance.

Returns `WebContents` | undefined - A WebContents instance with the given TargetID, or `undefined` if there is no WebContents associated with the given TargetID.

When communicating with the Chrome DevTools Protocol, it can be useful to lookup a WebContents instance based on its assigned TargetID.

```
async function lookupTargetId (browserWindow) {
  const wc = browserWindow.webContents
  await wc.debugger.attach('1.3')
  const { targetInfo } = await wc.debugger.sendCommand('Target.getTargetInfo')
  const { targetId } = targetInfo
  const targetWebContents = await webContents.fromDevToolsTargetId(targetId)
}
```

Class: WebContents

> *Render and control the contents of a BrowserWindow instance.*

Process: [Main](#)
This class is not exported from the `'electron'` module. It is only available as a return value of other methods in the Electron API.

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
- `frameProcessId` Integer
- `frameRoutingId` Integer

This event is like `did-finish-load` but emitted when the load failed. The full list of error codes and their meaning is available [here](#).

Event: 'did-fail-provisional-load'

Returns:

- `event` Event
- `errorCode` Integer
- `errorDescription` string
- `validatedURL` string
- `isMainFrame` boolean
- `frameProcessId` Integer
- `frameRoutingId` Integer

This event is like `did-fail-load` but emitted when the load was cancelled (e.g. `window.stop()` was invoked).

Event: 'did-frame-finish-load'

Returns:

- `event` Event
- `isMainFrame` boolean
- `frameProcessId` Integer
- `frameRoutingId` Integer

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: 'dom-ready'

Returns:

- `event` Event

Emitted when the document in the top-level frame is loaded.

Event: 'page-title-updated'

Returns:

- `event` Event
- `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:

- `event` Event
- `favicons` string[] - Array of URLs.

Emitted when page receives favicon urls.

Event: 'new-window' *Deprecated*

Returns:

- `event` NewWindowWebContentsEvent
- `url` string
- `frameName` string
- `disposition` string - Can be `default` , `foreground-tab` , `background-tab` , `new-window` , `save-to-disk` and `other` .
- `options` BrowserWindowConstructorOptions - 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()` . Deprecated, and will now always be the empty array `[]` .
- `referrer` [Referrer] - The referrer that will be passed to the new window. May or may not result in the `Referer` header being sent, depending on the referrer policy.

- `postBody` PostBody (optional) - The post data that will be sent to the new window, along with the appropriate headers that will be set. If no post data is to be sent, the value will be `null` . Only defined when the window is being created by a form that set `target=_blank` .

Deprecated in favor of `webContents.setWindowOpenHandler` .

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, frameName, disposition,
options, additionalFeatures, referrer, postBody) => {
  event.preventDefault()
  const win = new BrowserWindow({
    webContents: options.webContents, // use existing webContents if provided
    show: false
  })
  win.once('ready-to-show', () => win.show())
  if (!options.webContents) {
    const loadOptions = {
      httpReferrer: referrer
    }
    if (postBody != null) {
      const { data, contentType, boundary } = postBody
      loadOptions.postData = postBody.data
      loadOptions.extraHeaders = `content-type: ${contentType};
boundary=${boundary}`
    }

    win.loadURL(url, loadOptions) // existing webContents will be navigated
automatically
  }
  event.newGuest = win
})
```

Event: 'did-create-window'

Returns:

- `window` BrowserWindow
- `details` Object
  - `url` string - URL for the created window.
  - `frameName` string - Name given to the created window in the `window.open()` call.
  - `options` BrowserWindowConstructorOptions - The options used to create the BrowserWindow. They are merged in increasing precedence: parsed options from the `features` string from

`window.open()`, security-related webPreferences inherited from the parent, and options given by [`webContents.setWindowOpenHandler`]. Unrecognized options are not filtered out.
  - `referrer` [Referrer] - The referrer that will be passed to the new window. May or may not result in the `Referer` header being sent, depending on the referrer policy.
  - `postBody` [PostBody] (optional) - The post data that will be sent to the new window, along with the appropriate headers that will be set. If no post data is to be sent, the value will be `null`. Only defined when the window is being created by a form that set `target=_blank`.
  - `disposition` string - Can be `default`, `foreground-tab`, `background-tab`, `new-window`, `save-to-disk` and `other`.

Emitted *after* successful creation of a window via `window.open` in the renderer. Not emitted if the creation of the window is canceled from [`webContents.setWindowOpenHandler`].

See [`window.open()`] for more details and how to use this in conjunction with `webContents.setWindowOpenHandler`.

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-start-navigation'

Returns:

- `event` Event
- `url` string
- `isInPlace` boolean
- `isMainFrame` boolean
- `frameProcessId` Integer
- `frameRoutingId` Integer

Emitted when any frame (including main) starts navigating. `isInPlace` will be `true` for in-page navigations.

Event: 'will-redirect'

Returns:

- `event` Event
- `url` string
- `isInPlace` boolean

* `isMainFrame` boolean
* `frameProcessId` Integer
* `frameRoutingId` Integer

Emitted when a server side redirect occurs during navigation. For example a 302 redirect.

This event will be emitted after `did-start-navigation` and always before the `did-redirect-navigation` event for the same navigation.

Calling `event.preventDefault()` will prevent the navigation (not just the redirect).

Event: 'did-redirect-navigation'

Returns:

* `event` Event
* `url` string
* `isInPlace` boolean
* `isMainFrame` boolean
* `frameProcessId` Integer
* `frameRoutingId` Integer

Emitted after a server side redirect occurs during navigation. For example a 302 redirect.

This event cannot be prevented, if you want to prevent redirects you should checkout out the `will-redirect` event above.

Event: 'did-navigate'

Returns:

* `event` Event
* `url` string
* `httpResponseCode` Integer - -1 for non HTTP navigations
* `httpStatusText` string - empty for non HTTP navigations

Emitted when a main frame 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-frame-navigate'

Returns:

* `event` Event
* `url` string
* `httpResponseCode` Integer - -1 for non HTTP navigations
* `httpStatusText` string - empty for non HTTP navigations,
* `isMainFrame` boolean
* `frameProcessId` Integer
* `frameRoutingId` Integer

Emitted when any frame 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
- `frameProcessId` Integer
- `frameRoutingId` Integer

Emitted when an in-page navigation happened in any frame.

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.showMessageBoxSync(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()
  }
})
```

Note: This will be emitted for `BrowserViews` but will *not* be respected - this is because we have chosen not to tie the `BrowserView` lifecycle to its owning BrowserWindow should one exist per the specification.

Event: 'crashed' *Deprecated*

Returns:

- `event` Event

- `killed` boolean

Emitted when the renderer process crashes or is killed.

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

Event: 'render-process-gone'

Returns:

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

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

Event: 'unresponsive'

Emitted when the web page becomes unresponsive.

Event: 'responsive'

Emitted when the unresponsive web page becomes responsive again.

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.
  - `code` string - Equivalent to KeyboardEvent.code.
  - `isAutoRepeat` boolean - Equivalent to KeyboardEvent.repeat.
  - `isComposing` boolean - Equivalent to KeyboardEvent.isComposing.
  - `shift` boolean - Equivalent to KeyboardEvent.shiftKey.
  - `control` boolean - Equivalent to KeyboardEvent.controlKey.
  - `alt` boolean - Equivalent to KeyboardEvent.altKey.
  - `meta` boolean - Equivalent to KeyboardEvent.metaKey.
  - `location` number - Equivalent to KeyboardEvent.location.
  - `modifiers` string[] - See InputEvent.modifiers.

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` :

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

const 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: '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: 'zoom-changed'

Returns:

- `event` Event
- `zoomDirection` string - Can be `in` or `out` .

Emitted when the user is requesting to change the zoom level using the mouse wheel.

Event: 'blur'

Emitted when the `WebContents` loses focus.

Event: 'focus'

Emitted when the `WebContents` gains focus.

Note that on macOS, having focus means the `WebContents` is the first responder of window, so switching focus between windows would not trigger the `focus` and `blur` events of `WebContents` , as the first responder of each window is not changed.

The `focus` and `blur` events of `WebContents` should only be used to detect focus change between different `WebContents` and `BrowserView` in the same window.

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
- `callback` Function
  - `isTrusted` boolean - Indicates whether the certificate can be considered trusted.
- `isMainFrame` boolean

Emitted when failed to verify the `certificate` for `url` .

The usage is the same with the `certificate-error` event of `app` .

Event: 'select-client-certificate'

Returns:

- `event` Event
- `url` URL
- `certificateList` Certificate[]
- `callback` Function
  - `certificate` 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` .

Event: 'login'

Returns:

- `event` Event
- `authenticationResponseDetails` Object
  - `url` URL
- `authInfo` Object

    - `isProxy` boolean
    - `scheme` string
    - `host` string
    - `port` Integer
    - `realm` string
- `callback` Function
    - `username` string (optional)
    - `password` string (optional)

Emitted when `webContents` wants to do basic auth.

The usage is the same with [the `login` event of `app`]().

Event: 'found-in-page'

Returns:

- `event` Event
- `result` Object
    - `requestId` Integer
    - `activeMatchOrdinal` Integer - Position of the active match.
    - `matches` Integer - Number of Matches.
    - `selectionArea` Rectangle - 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'

Returns:

- `event` Event
- `color` (string | null) - Theme color is in format of '#rrggbb'. It is `null` when no theme color is set.

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 (optional) - the size of the `image` .
- `hotspot` 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` or `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.
  - `frame` WebFrameMain - Frame from which the context menu was invoked.
  - `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 text of the selection that the context menu was invoked on.
  - `altText` string - Alt text of the selection that the context menu was invoked on.
  - `suggestedFilename` string - Suggested filename to be used when saving file through 'Save Link As' option of context menu.
  - `selectionRect` Rectangle - Rect representing the coordinates in the document space of the selection.

   - `selectionStartOffset` number - Start position of the selection text.
   - `referrerPolicy` [Referrer] - The referrer policy of the frame on which the menu is invoked.
   - `misspelledWord` string - The misspelled word under the cursor, if any.
   - `dictionarySuggestions` string[] - An array of suggested words to show the user to replace the `misspelledWord` . Only available if there is a misspelled word and spellchecker is enabled.
   - `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` .
   - `spellcheckEnabled` boolean - If the context is editable, whether or not spellchecking is enabled.
   - `menuSourceType` string - Input source that invoked the context menu. Can be `none` , `mouse` , `keyboard` , `touch` , `touchMenu` , `longPress` , `longTap` , `touchHandle` , `stylus` , `adjustSelection` , or `adjustSelectionReset` .
   - `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.
     - `canPrint` boolean - Whether the media element can be printed.
     - `canSave` boolean - Whether or not the media element can be downloaded.
     - `canShowPictureInPicture` boolean - Whether the media element can show picture-in-picture.
     - `isShowingPictureInPicture` boolean - Whether the media element is currently showing picture-in-picture.
     - `canRotate` boolean - Whether the media element can be rotated.
     - `canLoop` boolean - Whether the media element can be looped.
   - `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.
     - `canEditRichly` boolean - Whether the renderer believes it can edit text richly.

Emitted when there is a new context menu that needs to be handled.

Event: 'select-bluetooth-device'

Returns:

- `event` Event
- `devices` BluetoothDevice[]

- `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.

If no event listener is added for this event, all bluetooth requests will be cancelled.

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

let win = null
app.commandLine.appendSwitch('enable-experimental-web-platform-features')

app.whenReady().then(() => {
  win = new BrowserWindow({ width: 800, height: 600 })
  win.webContents.on('select-bluetooth-device', (event, deviceList, callback) => {
    event.preventDefault()
    const result = deviceList.find((device) => {
      return device.deviceName === 'test'
    })
    if (!result) {
      callback('')
    } else {
      callback(result.deviceId)
    }
  })
})
```

Event: 'paint'

Returns:

- `event` Event
- `dirtyRect` Rectangle
- `image` NativeImage - 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')

const 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` WebPreferences - 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` Record<string, string> - 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.

Event: 'did-attach-webview'

Returns:

- `event` Event
- `webContents` WebContents - The guest web contents that is used by the `<webview>` .

Emitted when a `<webview>` has been attached to this web contents.

Event: 'console-message'

Returns:

- `event` Event
- `level` Integer - The log level, from 0 to 3. In order it matches `verbose` , `info` , `warning` and `error` .
- `message` string - The actual console message
- `line` Integer - The line number of the source that triggered this console message
- `sourceId` string

Emitted when the associated window logs a console message.

Event: 'preload-error'

Returns:

- `event` Event
- `preloadPath` string
- `error` Error

Emitted when the preload script `preloadPath` throws an unhandled exception `error` .

Event: 'ipc-message'

Returns:

- `event` Event
- `channel` string
- `...args` any[]

Emitted when the renderer process sends an asynchronous message via `ipcRenderer.send()` .

Event: 'ipc-message-sync'

Returns:

- `event` Event
- `channel` string
- `...args` any[]

Emitted when the renderer process sends a synchronous message via `ipcRenderer.sendSync()`.

Event: 'preferred-size-changed'

Returns:

- `event` Event
- `preferredSize` Size - The minimum size needed to contain the layout of the document—without requiring scrolling.

Emitted when the `WebContents` preferred size has changed.

This event will only be emitted when `enablePreferredSizeMode` is set to `true` in `webPreferences`.

Event: 'frame-created'

Returns:

- `event` Event
- `details` Object
  - `frame` WebFrameMain

Emitted when the mainFrame, an `<iframe>`, or a nested `<iframe>` is loaded within the page.

Instance Methods

`contents.loadURL(url[, options])`

- `url` string
- `options` Object (optional)
  - `httpReferrer` (string | Referrer) (optional) - An HTTP Referrer url.
  - `userAgent` string (optional) - A user agent originating the request.
  - `extraHeaders` string (optional) - Extra headers separated by "\n".
  - `postData` (UploadRawData | UploadFile)[] (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.

Returns `Promise<void>` - the promise will resolve when the page has finished loading (see `did-finish-load`), and rejects if the page fails to load (see `did-fail-load`). A noop rejection handler is already attached, which avoids unhandled rejection errors.

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.loadFile(filePath[, options])`

- `filePath` string
- `options` Object (optional)
  - `query` Record<string, string> (optional) - Passed to `url.format()`.
  - `search` string (optional) - Passed to `url.format()`.
  - `hash` string (optional) - Passed to `url.format()`.

Returns `Promise<void>` - the promise will resolve when the page has finished loading (see `did-finish-load`), and rejects if the page fails to load (see `did-fail-load`).

Loads the given file in the window, `filePath` should be a path to an HTML file relative to the root of your application. For instance an app structure like this:

```
| root
| - package.json
| - src
|   - main.js
|   - index.html
```

Would require code like this

```
win.loadFile('src/index.html')
```

`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')
const win = new BrowserWindow({ width: 800, height: 600 })
win.loadURL('http://github.com').then(() => {
  const 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.forcefullyCrashRenderer()`

Forcefully terminates the renderer process that is currently hosting this `webContents`. This will cause the `render-process-gone` event to be emitted with the `reason=killed || reason=crashed`. Please note that some webContents share renderer processes and therefore calling this method may also crash the host process for other webContents as well.

Calling `reload()` immediately after calling this method will force the reload to occur in a new process. This should be used when this process is unstable or unusable, for instance in order to recover from the `unresponsive` event.

```
contents.on('unresponsive', async () => {
  const { response } = await dialog.showMessageBox({
    message: 'App X has become unresponsive',
    title: 'Do you want to try forcefully reloading the app?',
    buttons: ['OK', 'Cancel'],
    cancelId: 1
  })
  if (response === 0) {
    contents.forcefullyCrashRenderer()
    contents.reload()
  }
})
```

`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[, options])`

- `css` string
- `options` Object (optional)

  - `cssOrigin` string (optional) - Can be either 'user' or 'author'. Sets the [cascade origin](https://www.w3.org/TR/css3-cascade/#cascade-origin) of the inserted stylesheet. Default is 'author'.

Returns `Promise<string>` - A promise that resolves with a key for the inserted CSS that can later be used to remove the CSS via `contents.removeInsertedCSS(key)`.

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

```
contents.on('did-finish-load', () => {
  contents.insertCSS('html, body { background-color: #f00; }')
})
```

`contents.removeInsertedCSS(key)`

- `key` string

Returns `Promise<void>` - Resolves if the removal was successful.

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

```
contents.on('did-finish-load', async () => {
  const key = await contents.insertCSS('html, body { background-color: #f00; }')
  contents.removeInsertedCSS(key)
})
```

`contents.executeJavaScript(code[, userGesture])`

- `code` string
- `userGesture` boolean (optional) - Default is `false`.

Returns `Promise<any>` - 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.

Code execution will be suspended until web page stop loading.

```
contents.executeJavaScript('fetch("https://jsonplaceholder.typicode.com/users/1").then
 => resp.json())', true)
  .then((result) => {
    console.log(result) // Will be the JSON object from the fetch call
  })
```

`contents.executeJavaScriptInIsolatedWorld(worldId, scripts[, userGesture])`

- `worldId` Integer - The ID of the world to run the javascript in, `0` is the default world, `999` is the world used by Electron's `contextIsolation` feature. You can provide any integer here.
- `scripts` [WebSource[]](structures/web-source.md)

- `userGesture` boolean (optional) - Default is `false`.

Returns `Promise<any>` - A promise that resolves with the result of the executed code or is rejected if the result of the code is a rejected promise.

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

`contents.setIgnoreMenuShortcuts(ignore)`

- `ignore` boolean

Ignore application menu shortcuts while this web contents is focused.

`contents.setWindowOpenHandler(handler)`

- `handler` Function<{action: 'deny'} | {action: 'allow', overrideBrowserWindowOptions?: BrowserWindowConstructorOptions}>
  - `details` Object
    - `url` string - The *resolved* version of the URL passed to `window.open()`. e.g. opening a window with `window.open('foo')` will yield something like `https://the-origin/the/current/path/foo`.
    - `frameName` string - Name of the window provided in `window.open()`
    - `features` string - Comma separated list of window features provided to `window.open()`.
    - `disposition` string - Can be `default`, `foreground-tab`, `background-tab`, `new-window`, `save-to-disk` or `other`.
    - `referrer` [Referrer](structures/referrer.md) - The referrer that will be passed to the new window. May or may not result in the `Referer` header being sent, depending on the referrer policy.
    - `postBody` [PostBody](structures/post-body.md) (optional) - The post data that will be sent to the new window, along with the appropriate headers that will be set. If no post data is to be sent, the value will be `null`. Only defined when the window is being created by a form that set `target=_blank`.

  Returns `{action: 'deny'} | {action: 'allow', overrideBrowserWindowOptions?: BrowserWindowConstructorOptions}` - `deny` cancels the creation of the new window. `allow` will allow the new window to be created. Specifying `overrideBrowserWindowOptions` allows customization of the created window. Returning an unrecognized value such as a null, undefined, or an object without a recognized 'action' value will result in a console error and have the same effect as returning `{action: 'deny'}`.

Called before creating a window a new window is requested by the renderer, e.g. by `window.open()`, a link with `target="_blank"`, shift+clicking on a link, or submitting a form with `<form target="_blank">`. See [`window.open()`](window-open.md) for more details and how to use this in conjunction with `did-create-window`.

`contents.setAudioMuted(muted)`

- `muted` boolean

Mute the audio on the current web page.

`contents.isAudioMuted()`

Returns `boolean` - Whether this page has been muted.

`contents.isCurrentlyAudible()`

Returns `boolean` - Whether audio is currently playing.

`contents.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.

`contents.getZoomFactor()`

Returns `number` - the current zoom factor.

`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. The formula for this is `scale := 1.2 ^ level`.

> ***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.*

`contents.getZoomLevel()`

Returns `number` - the current zoom level.

`contents.setVisualZoomLevelLimits(minimumLevel, maximumLevel)`

- `minimumLevel` number
- `maximumLevel` number

Returns `Promise<void>`

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

> ***NOTE***: *Visual zoom is disabled by default in Electron. To re-enable it, call:*
>
> ```
> contents.setVisualZoomLevelLimits(1, 3)
> ```

`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

Returns `Promise<void>`

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 to begin a new text finding session with this request. Should be `true` for initial requests, and `false` for follow-up requests. Defaults to `false` .
  - `matchCase` boolean (optional) - Whether search should be case-sensitive, defaults to `false` .

Returns `Integer` - The request id used for the request.

Starts a request to find all matches for the `text` in the web page. The result of the request can be obtained by subscribing to [`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])`

- `rect` [Rectangle] (optional) - The area of the page to be captured.

Returns `Promise<NativeImage>` - Resolves with a [NativeImage]

Captures a snapshot of the page within `rect`. Omitting `rect` will capture the whole visible page.

`contents.isBeingCaptured()`

Returns `boolean` - Whether this page is being captured. It returns true when the capturer count is large then 0.

`contents.incrementCapturerCount([size, stayHidden, stayAwake])`

- `size` [Size] (optional) - The preferred size for the capturer.
- `stayHidden` boolean (optional) - Keep the page hidden instead of visible.
- `stayAwake` boolean (optional) - Keep the system awake instead of allowing it to sleep.

Increase the capturer count by one. The page is considered visible when its browser window is hidden and the capturer count is non-zero. If you would like the page to stay hidden, you should ensure that `stayHidden` is set to true.

This also affects the Page Visibility API.

`contents.decrementCapturerCount([stayHidden, stayAwake])`

- `stayHidden` boolean (optional) - Keep the page in hidden state instead of visible.
- `stayAwake` boolean (optional) - Keep the system awake instead of allowing it to sleep.

Decrease the capturer count by one. The page will be set to hidden or occluded state when its browser window is hidden or occluded and the capturer count reaches zero. If you want to decrease the hidden capturer count instead you should set `stayHidden` to true.

`contents.getPrinters()` ***Deprecated***

Get the system printer list.

Returns `PrinterInfo[]`

Deprecated: Should use the new `contents.getPrintersAsync` API.

`contents.getPrintersAsync()`

Get the system printer list.

Returns `Promise<PrinterInfo[]>` - Resolves with a `PrinterInfo[]`

`contents.print([options], [callback])`

- `options` Object (optional)
  - `silent` boolean (optional) - Don't ask user for print settings. Default is `false`.
  - `printBackground` boolean (optional) - Prints the background color and image of the web page. Default is `false`.
  - `deviceName` string (optional) - Set the printer device name to use. Must be the system-defined name and not the 'friendly' name, e.g 'Brother_QL_820NWB' and not 'Brother QL-820NWB'.
  - `color` boolean (optional) - Set whether the printed web page will be in color or grayscale. Default is `true`.
  - `margins` Object (optional)
    - `marginType` string (optional) - Can be `default`, `none`, `printableArea`, or `custom`. If `custom` is chosen, you will also need to specify `top`, `bottom`, `left`, and `right`.
    - `top` number (optional) - The top margin of the printed web page, in pixels.
    - `bottom` number (optional) - The bottom margin of the printed web page, in pixels.
    - `left` number (optional) - The left margin of the printed web page, in pixels.
    - `right` number (optional) - The right margin of the printed web page, in pixels.
  - `landscape` boolean (optional) - Whether the web page should be printed in landscape mode. Default is `false`.
  - `scaleFactor` number (optional) - The scale factor of the web page.
  - `pagesPerSheet` number (optional) - The number of pages to print per page sheet.
  - `collate` boolean (optional) - Whether the web page should be collated.
  - `copies` number (optional) - The number of copies of the web page to print.
  - `pageRanges` Object[] (optional) - The page range to print. On macOS, only one range is honored.
    - `from` number - Index of the first page to print (0-based).
    - `to` number - Index of the last page to print (inclusive) (0-based).
  - `duplexMode` string (optional) - Set the duplex mode of the printed web page. Can be `simplex`, `shortEdge`, or `longEdge`.
  - `dpi` Record<string, number> (optional)
    - `horizontal` number (optional) - The horizontal dpi.
    - `vertical` number (optional) - The vertical dpi.
  - `header` string (optional) - string to be printed as page header.
  - `footer` string (optional) - string to be printed as page footer.

    - `pageSize` string | Size (optional) - Specify page size of the printed document. Can be `A3`, `A4`, `A5`, `Legal`, `Letter`, `Tabloid` or an Object containing `height`.
- `callback` Function (optional)
    - `success` boolean - Indicates success of the print call.
    - `failureReason` string - Error description called back if the print fails.

When a custom `pageSize` is passed, Chromium attempts to validate platform specific minimum values for `width_microns` and `height_microns`. Width and height must both be minimum 353 microns but may be higher on some operating systems.

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.

Use `page-break-before: always;` CSS style to force to print to a new page.

Example usage:

```
const options = {
  silent: true,
  deviceName: 'My-Printer',
  pageRanges: [{
    from: 0,
    to: 1
  }]
}
win.webContents.print(options, (success, errorType) => {
  if (!success) console.log(errorType)
})
```

`contents.printToPDF(options)`

- `options` Object
    - `headerFooter` Record<string, string> (optional) - the header and footer for the PDF.
        - `title` string - The title for the PDF header.
        - `url` string - the url for the PDF footer.
    - `landscape` boolean (optional) - `true` for landscape, `false` for portrait.
    - `marginsType` Integer (optional) - Specifies the type of margins to use. Uses 0 for default margin, 1 for no margin, and 2 for minimum margin.
    - `scaleFactor` number (optional) - The scale factor of the web page. Can range from 0 to 100.
    - `pageRanges` Record<string, number> (optional) - The page range to print.
        - `from` number - Index of the first page to print (0-based).
        - `to` number - Index of the last page to print (inclusive) (0-based).
    - `pageSize` string | Size (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.

Returns `Promise<Buffer>` - Resolves with the generated PDF data.

Prints window's web page as PDF with Chromium's preview printing custom settings.

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,
  pageSize: 'A4',
  scaleFactor: 100
}
```

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')
const path = require('path')
const os = require('os')

const win = new BrowserWindow({ width: 800, height: 600 })
win.loadURL('http://github.com')

win.webContents.on('did-finish-load', () => {
  // Use default printing options
  const pdfPath = path.join(os.homedir(), 'Desktop', 'temp.pdf')
  win.webContents.printToPDF({}).then(data => {
    fs.writeFile(pdfPath, data, (error) => {
      if (error) throw error
      console.log(`Wrote PDF successfully to ${pdfPath}`)
    })
  }).catch(error => {
    console.log(`Failed to write PDF to ${pdfPath}: `, error)
  })
})
```

`contents.addWorkSpace(path)`

- `path` string

Adds the specified path to DevTools workspace. Must be used after DevTools creation:

```
const { BrowserWindow } = require('electron')
const 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.setDevToolsWebContents(devToolsWebContents)`

- `devToolsWebContents` WebContents

Uses the `devToolsWebContents` as the target `WebContents` to show devtools.

The `devToolsWebContents` must not have done any navigation, and it should not be used for other purposes after the call.

By default Electron manages the devtools by creating an internal `WebContents` with native view, which developers have very limited control of. With the `setDevToolsWebContents` method, developers can use any `WebContents` to show the devtools in it, including `BrowserWindow` , `BrowserView` and `<webview>` tag.

Note that closing the devtools does not destroy the `devToolsWebContents` , it is caller's responsibility to destroy `devToolsWebContents` .

An example of showing devtools in a `<webview>` tag:

```html
<html>
<head>
  <style type="text/css">
    * { margin: 0; }
    #browser { height: 70%; }
    #devtools { height: 30%; }
  </style>
</head>
<body>
  <webview id="browser" src="https://github.com"></webview>
  <webview id="devtools" src="about:blank"></webview>
  <script>
    const { ipcRenderer } = require('electron')
    const emittedOnce = (element, eventName) => new Promise(resolve => {
      element.addEventListener(eventName, event => resolve(event), { once: true })
    })
    const browserView = document.getElementById('browser')
    const devtoolsView = document.getElementById('devtools')
    const browserReady = emittedOnce(browserView, 'dom-ready')
    const devtoolsReady = emittedOnce(devtoolsView, 'dom-ready')
    Promise.all([browserReady, devtoolsReady]).then(() => {
      const targetId = browserView.getWebContentsId()
      const devtoolsId = devtoolsView.getWebContentsId()
      ipcRenderer.send('open-devtools', targetId, devtoolsId)
    })
  </script>
</body>
</html>
```

```
// Main process
const { ipcMain, webContents } = require('electron')
ipcMain.on('open-devtools', (event, targetContentsId, devtoolsContentsId) => {
  const target = webContents.fromId(targetContentsId)
  const devtools = webContents.fromId(devtoolsContentsId)
  target.setDevToolsWebContents(devtools)
  target.openDevTools()
})
```

An example of showing devtools in a `BrowserWindow` :

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

let win = null
let devtools = null

app.whenReady().then(() => {
  win = new BrowserWindow()
  devtools = new BrowserWindow()
  win.loadURL('https://github.com')
  win.webContents.setDevToolsWebContents(devtools.webContents)
  win.webContents.openDevTools({ mode: 'detach' })
})
```

`contents.openDevTools([options])`

- `options` Object (optional)
  - `mode` string - Opens the devtools with specified dock state, can be `left` , `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.
  - `activate` boolean (optional) - Whether to bring the opened devtools window to the foreground. The default is `true` .

Opens the devtools.

When `contents` is a `<webview>` tag, the `mode` would be `detach` by default, explicitly passing an empty `mode` can force using last used dock state.

`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.inspectSharedWorker()`

Opens the developer tools for the shared worker context.

`contents.inspectSharedWorkerById(workerId)`

- `workerId` string

Inspects the shared worker based on its ID.

`contents.getAllSharedWorkers()`

Returns `SharedWorkerInfo[]` - Information about all Shared Workers.

`contents.inspectServiceWorker()`

Opens the developer tools for the service worker context.

`contents.send(channel, ...args)`

- `channel` string
- `...args` any[]

Send an asynchronous message to the renderer process via `channel` , along with arguments. Arguments will be serialized with the Structured Clone Algorithm, just like `postMessage` , so prototype chains will not be included. Sending Functions, Promises, Symbols, WeakMaps, or WeakSets will throw an exception.

> ***NOTE**: Sending non-standard JavaScript types such as DOM objects or special Electron objects will throw an exception.*

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.whenReady().then(() => {
  win = new BrowserWindow({ width: 800, height: 600 })
  win.loadURL(`file://${__dirname}/index.html`)
  win.webContents.on('did-finish-load', () => {
    win.webContents.send('ping', 'whoooooooh!')
  })
})
```

```html
<!-- index.html -->
<html>
<body>
  <script>
    require('electron').ipcRenderer.on('ping', (event, message) => {
      console.log(message) // Prints 'whoooooooh!'
    })
  </script>
</body>
</html>
```

`contents.sendToFrame(frameId, channel, ...args)`

- `frameId` Integer | [number, number] - the ID of the frame to send to, or a pair of `[processId, frameId]` if the frame is in a different process to the main frame.
- `channel` string
- `...args` any[]

Send an asynchronous message to a specific frame in a renderer process via `channel`, along with arguments. Arguments will be serialized with the Structured Clone Algorithm, just like `postMessage`, so prototype chains will not be included. Sending Functions, Promises, Symbols, WeakMaps, or WeakSets will throw an exception.

> ***NOTE:*** *Sending non-standard JavaScript types such as DOM objects or special Electron objects will throw an exception.*

The renderer process can handle the message by listening to `channel` with the `ipcRenderer` module.

If you want to get the `frameId` of a given renderer context you should use the `webFrame.routingId` value. E.g.

```js
// In a renderer process
console.log('My frameId is:', require('electron').webFrame.routingId)
```

You can also read `frameId` from all incoming IPC messages in the main process.

```js
// In the main process
ipcMain.on('ping', (event) => {
  console.info('Message came from frameId:', event.frameId)
})
```

`contents.postMessage(channel, message, [transfer])`

- `channel` string
- `message` any
- `transfer` MessagePortMain[] (optional)

Send a message to the renderer process, optionally transferring ownership of zero or more [`MessagePortMain`][] objects.

The transferred `MessagePortMain` objects will be available in the renderer process by accessing the `ports` property of the emitted event. When they arrive in the renderer, they will be native DOM `MessagePort` objects.

For example:

```
// Main process
const { port1, port2 } = new MessageChannelMain()
webContents.postMessage('port', { message: 'hello' }, [port1])

// Renderer process
ipcRenderer.on('port', (e, msg) => {
  const [port] = e.ports
  // ...
})
```

`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 - Set the emulated screen size (screenPosition == mobile).
  - `viewPosition` 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 - Set the emulated view size (empty means no override)
  - `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(inputEvent)`

- `inputEvent` MouseInputEvent | MouseWheelInputEvent | KeyboardInputEvent

Sends an input `event` to the page. **Note:** The `BrowserWindow` containing the contents needs to be focused for `sendInputEvent()` to work.

`contents.beginFrameSubscription([onlyDirty ,]callback)`

- `onlyDirty` boolean (optional) - Defaults to `false`.
- `callback` Function
  - `image` NativeImage
  - `dirtyRect` Rectangle

Begin subscribing for presentation events and captured frames, the `callback` will be called with `callback(image, dirtyRect)` when there is a presentation event.

The `image` is an instance of NativeImage that stores the captured frame.

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`, `image` 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 - The path to the file being dragged.
  - `files` string[] (optional) - The paths to the files being dragged. (`files` will override `file` field)
  - `icon` NativeImage | string - 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)`

- `fullPath` string - The absolute 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.

Returns `Promise<void>` - resolves if the page is saved.

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

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

win.webContents.on('did-finish-load', async () => {
  win.webContents.savePage('/tmp/test.html', 'HTMLComplete').then(() => {
    console.log('Page was saved successfully.')
  }).catch(err => {
    console.log(err)
  })
})
```

`contents.showDefinitionForSelection()` *macOS*

Shows pop-up dictionary that searches the selected word on the page.

`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 240 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 for more details.

`contents.getMediaSourceId(requestWebContents)`

- `requestWebContents` WebContents - Web contents that the id will be registered to.

Returns `string` - The identifier of a WebContents stream. This identifier can be used with `navigator.mediaDevices.getUserMedia` using a `chromeMediaSource` of `tab`. The identifier is restricted to the web contents that it is registered to and is only valid for 10 seconds.

`contents.getOSProcessId()`

Returns `Integer` - The operating system `pid` of the associated renderer process.

`contents.getProcessId()`

Returns `Integer` - The Chromium internal `pid` of the associated renderer. Can be compared to the `frameProcessId` passed by frame specific navigation events (e.g. `did-frame-navigate`)

`contents.takeHeapSnapshot(filePath)`

- `filePath` string - Path to the output file.

Returns `Promise<void>` - Indicates whether the snapshot has been created successfully.

Takes a V8 heap snapshot and saves it to `filePath` .

`contents.getBackgroundThrottling()`

Returns `boolean` - whether or not this WebContents will throttle animations and timers when the page becomes backgrounded. This also affects the Page Visibility API.

`contents.setBackgroundThrottling(allowed)`

- `allowed` boolean

Controls whether or not this WebContents will throttle animations and timers when the page becomes backgrounded. This also affects the Page Visibility API.

`contents.getType()`

Returns `string` - the type of the webContent. Can be `backgroundPage` , `window` , `browserView` , `remote` , `webview` or `offscreen` .

`contents.setImageAnimationPolicy(policy)`

- `policy` string - Can be `animate` , `animateOnce` or `noAnimation` .

Sets the image animation policy for this webContents. The policy only affects *new* images, existing images that are currently being animated are unaffected. This is a known limitation in Chromium, you can force image animation to be recalculated with `img.src = img.src` which will result in no network traffic but will update the animation policy.

This corresponds to the animationPolicy accessibility feature in Chromium.

Instance Properties

`contents.audioMuted`

A `boolean` property that determines whether this page is muted.

`contents.userAgent`

A `string` property that determines the user agent for this web page.

`contents.zoomLevel`

A `number` property that determines the zoom level for this web contents.

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. The formula for this is `scale := 1.2 ^ level`.

`contents.zoomFactor`

A `number` property that determines the zoom factor for this web contents.

The zoom factor is the zoom percent divided by 100, so 300% = 3.0.

`contents.frameRate`

An `Integer` property that sets the frame rate of the web contents to the specified number. Only values between 1 and 240 are accepted.

Only applicable if *offscreen rendering* is enabled.

`contents.id` *Readonly*

A `Integer` representing the unique ID of this WebContents. Each ID is unique among all `WebContents` instances of the entire Electron application.

`contents.session` *Readonly*

A `Session` used by this webContents.

`contents.hostWebContents` *Readonly*

A `WebContents` instance that might own this `WebContents`.

`contents.devToolsWebContents` *Readonly*

A `WebContents | null` property that represents the of DevTools `WebContents` associated with a given `WebContents`.

Note: Users should never store this object because it may become `null` when the DevTools has been closed.

`contents.debugger` *Readonly*

A `Debugger` instance for this webContents.

`contents.backgroundThrottling`

A `boolean` property that determines whether or not this WebContents will throttle animations and timers when the page becomes backgrounded. This also affects the Page Visibility API.

`contents.mainFrame` *Readonly*

A `WebFrameMain` property that represents the top frame of the page's frame hierarchy.