webFrameMain

Control web pages and iframes.

Process: Main

The `webFrameMain` module can be used to lookup frames across existing `WebContents` instances. Navigation events are the common use case.

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

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

win.webContents.on(
  'did-frame-navigate',
  (event, url, isMainFrame, frameProcessId, frameRoutingId) => {
    const frame = webFrameMain.fromId(frameProcessId, frameRoutingId)
    if (frame) {
      const code = 'document.body.innerHTML = document.body.innerHTML.replaceAll("heck", "h
      frame.executeJavaScript(code)
    }
  }
)
```

You can also access frames of existing pages by using the `mainFrame` property of `WebContents`.

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

async function main () {
  const win = new BrowserWindow({ width: 800, height: 600 })
  await win.loadURL('https://reddit.com')

  const youtubeEmbeds = win.webContents.mainFrame.frames.filter((frame) => {
    try {
      const url = new URL(frame.url)
      return url.host === 'www.youtube.com'
    } catch {
      return false
    }
  })

  console.log(youtubeEmbeds)
}

main()
```

Methods

These methods can be accessed from the `webFrameMain` module:

`webFrameMain.fromId(processId, routingId)`

- `processId` Integer - An `Integer` representing the internal ID of the process which owns the frame.
- `routingId` Integer - An `Integer` representing the unique frame ID in the current renderer process. Routing IDs can be retrieved from `WebFrameMain` instances (`frame.routingId`) and are also passed by frame specific `WebContents` navigation events (e.g. `did-frame-navigate`).

Returns `WebFrameMain | undefined` - A frame with the given process and routing IDs, or `undefined` if there is no WebFrameMain associated with the given IDs.

Class: WebFrameMain

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: 'dom-ready' Emitted when the document is loaded.

Instance Methods

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

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

Returns `Promise<unknown>` - 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.

`frame.reload()` Returns `boolean` - Whether the reload was initiated successfully. Only results in `false` when the frame has no history.

`frame.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.

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

`frame.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.mainFrame.postMessage('port', { message: 'hello' }, [port1])

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

Instance Properties

`frame.url` *Readonly* A `string` representing the current URL of the frame.

`frame.top` *Readonly* A `WebFrameMain | null` representing top frame in the frame hierarchy to which `frame` belongs.

`frame.parent` *Readonly* A `WebFrameMain | null` representing parent frame of `frame`, the property would be `null` if `frame` is the top frame in the frame hierarchy.

`frame.frames` *Readonly* A `WebFrameMain[]` collection containing the direct descendents of `frame`.

`frame.framesInSubtree` ***Readonly*** A `WebFrameMain[]` collection containing every frame in the subtree of `frame`, including itself. This can be useful when traversing through all frames.

`frame.frameTreeNodeId` ***Readonly*** An `Integer` representing the id of the frame's internal FrameTreeNode instance. This id is browser-global and uniquely identifies a frame that hosts content. The identifier is fixed at the creation of the frame and stays constant for the lifetime of the frame. When the frame is removed, the id is not used again.

`frame.name` ***Readonly*** A `string` representing the frame name.

`frame.osProcessId` ***Readonly*** An `Integer` representing the operating system `pid` of the process which owns this frame.

`frame.processId` ***Readonly*** An `Integer` representing the Chromium internal `pid` of the process which owns this frame. This is not the same as the OS process ID; to read that use `frame.osProcessId`.

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

`frame.visibilityState` ***Readonly*** A `string` representing the visibility state of the frame.

See also how the Page Visibility API is affected by other Electron APIs.