Opening windows from the renderer

There are several ways to control how windows are created from trusted or untrusted content within a renderer. Windows can be created from the renderer in two ways:

- clicking on links or submitting forms adorned with `target=_blank`
- JavaScript calling `window.open()`

For same-origin content, the new window is created within the same process, enabling the parent to access the child window directly. This can be very useful for app sub-windows that act as preference panels, or similar, as the parent can render to the sub-window directly, as if it were a `div` in the parent. This is the same behavior as in the browser.

Electron pairs this native Chrome `Window` with a BrowserWindow under the hood. You can take advantage of all the customization available when creating a BrowserWindow in the main process by using `webContents.setWindowOpenHandler()` for renderer-created windows.

BrowserWindow constructor options are set by, in increasing precedence order: parsed options from the `features` string from `window.open()`, security-related webPreferences inherited from the parent, and options given by `webContents.setWindowOpenHandler`. Note that `webContents.setWindowOpenHandler` has final say and full privilege because it is invoked in the main process.

`window.open(url[, frameName][, features])`

- `url` string
- `frameName` string (optional)
- `features` string (optional)

Returns `Window` | null

`features` is a comma-separated key-value list, following the standard format of the browser. Electron will parse `BrowserWindowConstructorOptions` out of this list where possible, for convenience. For full control and better ergonomics, consider using `webContents.setWindowOpenHandler` to customize the BrowserWindow creation.

A subset of `WebPreferences` can be set directly, unnested, from the features string: `zoomFactor`, `nodeIntegration`, `preload`, `javascript`, `contextIsolation`, and `webviewTag`.

For example:

```
window.open('https://github.com', '_blank', 'top=500,left=200,frame=false,nodeIntegration=no
```

Notes:

- Node integration will always be disabled in the opened `window` if it is disabled on the parent window.

- Context isolation will always be enabled in the opened `window` if it is enabled on the parent window.
- JavaScript will always be disabled in the opened `window` if it is disabled on the parent window.
- Non-standard features (that are not handled by Chromium or Electron) given in `features` will be passed to any registered `webContents`'s `did-create-window` event handler in the `options` argument.
- `frameName` follows the specification of `windowName` located in the native documentation.
- When opening `about:blank`, the child window's `WebPreferences` will be copied from the parent window, and there is no way to override it because Chromium skips browser side navigation in this case.

To customize or cancel the creation of the window, you can optionally set an override handler with `webContents.setWindowOpenHandler()` from the main process. Returning `{ action: 'deny' }` cancels the window. Returning `{ action: 'allow', overrideBrowserWindowOptions: { ... } }` will allow opening the window and setting the `BrowserWindowConstructorOptions` to be used when creating the window. Note that this is more powerful than passing options through the feature string, as the renderer has more limited privileges in deciding security preferences than the main process.

In addition to passing in `action` and `overrideBrowserWindowOptions`, `outlivesOpener` can be passed like: `{ action: 'allow', outlivesOpener: true, overrideBrowserWindowOptions: { ... } }`. If set to `true`, the newly created window will not close when the opener window closes. The default value is `false`.

Native `Window` example

```
// main.js
const mainWindow = new BrowserWindow()

// In this example, only windows with the `about:blank` url will be created.
// All other urls will be blocked.
mainWindow.webContents.setWindowOpenHandler(({ url }) => {
  if (url === 'about:blank') {
    return {
      action: 'allow',
      overrideBrowserWindowOptions: {
        frame: false,
        fullscreenable: false,
        backgroundColor: 'black',
        webPreferences: {
          preload: 'my-child-window-preload-script.js'
        }
      }
```

```
    }
  }
  return { action: 'deny' }
})

// renderer process (mainWindow)
const childWindow = window.open('', 'modal')
childWindow.document.write('<h1>Hello</h1>')
```