Main Process Modules Menu

Menu

Class: Menu

> Create native application menus and context menus.

Process: **Main**

WARNING

Electron's built-in classes cannot be subclassed in user code. For more information, see **the FAQ**.

`new Menu()`

Creates a new menu.

Static Methods

The `Menu` class has the following static methods:

`Menu.setApplicationMenu(menu)`

- `menu` Menu | null

Sets `menu` as the application menu on macOS. On Windows and Linux, the `menu` will be set as each window's top menu.

Also on Windows and Linux, you can use a `&` in the top-level item name to indicate which letter should get a generated accelerator. For example, using `&File` for the file menu would result in a generated `Alt-F` accelerator that opens the associated menu. The indicated character in the button label then gets an underline, and the `&` character is not displayed on the button label.

In order to escape the `&` character in an item name, add a proceeding `&`. For example, `&&File` would result in `&File` displayed on the button label.

Passing `null` will suppress the default menu. On Windows and Linux, this has the additional effect of removing the menu bar from the window.

> **NOTE**
>
> The default menu will be created automatically if the app does not set one. It contains standard items such as `File`, `Edit`, `View`, `Window` and `Help`.

`Menu.getApplicationMenu()`

Returns `Menu | null` - The application menu, if set, or `null`, if not set.

> **NOTE**
>
> The returned `Menu` instance doesn't support dynamic addition or removal of menu items. **Instance properties** can still be dynamically modified.

`Menu.sendActionToFirstResponder(action)` *macOS*

- `action` string

Sends the `action` to the first responder of application. This is used for emulating default macOS menu behaviors. Usually you would use the `role` property of a `MenuItem`.

See the **macOS Cocoa Event Handling Guide** for more information on macOS' native actions.

`Menu.buildFromTemplate(template)`

- `template` (MenuItemConstructorOptions | MenuItem)[]

Returns `Menu`

Generally, the `template` is an array of `options` for constructing a **MenuItem**. The usage can be referenced above.

You can also attach other fields to the element of the `template` and they will become properties of the constructed menu items.

Instance Methods

The `menu` object has the following instance methods:

`menu.popup([options])`

- `options` Object (optional)
  - `window` **BaseWindow** (optional) - Default is the focused window.
  - `frame` **WebFrameMain** (optional) - Provide the relevant frame if you want certain OS-level features such as Writing Tools on macOS to function correctly. Typically, this should be `params.frame` from the **`context-menu` event** on a WebContents, or the **`focusedFrame` property** of a WebContents.
  - `x` number (optional) - Default is the current mouse cursor position. Must be declared if `y` is declared.
  - `y` number (optional) - Default is the current mouse cursor position. Must be declared if `x` is declared.
  - `positioningItem` number (optional) *macOS* - The index of the menu item to be positioned under the mouse cursor at the specified coordinates. Default is -1.
  - `sourceType` string (optional) *Windows* *Linux* - This should map to the `menuSourceType` provided by the `context-menu` event. It is not recommended to set this value manually, only provide values you receive from other APIs or leave it `undefined`. Can be `none`, `mouse`, `keyboard`, `touch`, `touchMenu`, `longPress`, `longTap`, `touchHandle`, `stylus`, `adjustSelection`, or `adjustSelectionReset`.
  - `callback` Function (optional) - Called when menu is closed.

Pops up this menu as a context menu in the `BaseWindow`.

`menu.closePopup([window])`

- `window` **BaseWindow** (optional) - Default is the focused window.

Closes the context menu in the `window`.

`menu.append(menuItem)`

- `menuItem` **MenuItem**

Appends the `menuItem` to the menu.

`menu.getMenuItemById(id)`

- `id` string

Returns `MenuItem | null` the item with the specified `id`

`menu.insert(pos, menuItem)`

- `pos` Integer
- `menuItem` **MenuItem**

Inserts the `menuItem` to the `pos` position of the menu.

Instance Events

Objects created with `new Menu` or returned by `Menu.buildFromTemplate` emit the following events:

> **NOTE**
>
> Some events are only available on specific operating systems and are labeled as such.

Event: 'menu-will-show'

Returns:

- `event` Event

Emitted when `menu.popup()` is called.

Event: 'menu-will-close'

Returns:

- `event` Event

Emitted when a popup is closed either manually or with `menu.closePopup()`.

Instance Properties

`menu` objects also have the following properties:

`menu.items`

A `MenuItem[]` array containing the menu's items.

Each `Menu` consists of multiple `MenuItem`s and each `MenuItem` can have a submenu.

Examples

An example of creating the application menu with the simple template API:

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

const isMac = process.platform === 'darwin'

const template = [
  // { role: 'appMenu' }
  ...(isMac
    ? [{
        label: app.name,
        submenu: [
          { role: 'about' },
          { type: 'separator' },
          { role: 'services' },
          { type: 'separator' },
          { role: 'hide' },
          { role: 'hideOthers' },
          { role: 'unhide' },
          { type: 'separator' },
          { role: 'quit' }
        ]
      }]
    : []),
  // { role: 'fileMenu' }
  {
      label: 'File',
      submenu: [
        isMac ? { role: 'close' } : { role: 'quit' }
      ]
    },
    // { role: 'editMenu' }
    {
      label: 'Edit',
      submenu: [
        { role: 'undo' },
        { role: 'redo' },
        { type: 'separator' },
        { role: 'cut' },
        { role: 'copy' },
        { role: 'paste' },
        ...(isMac
          ? [
              { role: 'pasteAndMatchStyle' },
              { role: 'delete' },
              { role: 'selectAll' },
              { type: 'separator' },
              {
                label: 'Speech',
                submenu: [
                  { role: 'startSpeaking' },
                  { role: 'stopSpeaking' }
                ]
              }
            ]
          : [
              { role: 'delete' },
              { type: 'separator' },
              { role: 'selectAll' }
            ])
      ]
    },
    // { role: 'viewMenu' }
    {
      label: 'View',
      submenu: [
        { role: 'reload' },
        { role: 'forceReload' },
        { role: 'toggleDevTools' },
        { type: 'separator' },
        { role: 'resetZoom' },
        { role: 'zoomIn' },
        { role: 'zoomOut' },
        { type: 'separator' },
        { role: 'togglefullscreen' }
      ]
    },
    // { role: 'windowMenu' }
    {
      label: 'Window',
      submenu: [
        { role: 'minimize' },
        { role: 'zoom' },
        ...(isMac
          ? [
              { type: 'separator' },
              { role: 'front' },
              { type: 'separator' },
              { role: 'window' }
            ]
          : [
              { role: 'close' }
            ])
      ]
    },
    {
      role: 'help',
      submenu: [
        {
          label: 'Learn More',
          click: async () => {
            const { shell } = require('electron')
            await shell.openExternal('https://electronjs.org')
          }
        }
      ]
    }
  ]

  const menu = Menu.buildFromTemplate(template)
  Menu.setApplicationMenu(menu)
```

Render process

To create menus initiated by the renderer process, send the required information to the main process using IPC and have the main process display the menu on behalf of the renderer.

Below is an example of showing a menu when the user right clicks the page:

```
// renderer
window.addEventListener('contextmenu', (e) => {
  e.preventDefault()
  ipcRenderer.send('show-context-menu')
})

ipcRenderer.on('context-menu-command', (e, command) => {
  // ...
})

// main
ipcMain.on('show-context-menu', (event) => {
  const template = [
    {
      label: 'Menu Item 1',
      click: () => { event.sender.send('context-menu-command', 'menu-item-1') }
    },
    { type: 'separator' },
    { label: 'Menu Item 2', type: 'checkbox', checked: true }
  ]
  const menu = Menu.buildFromTemplate(template)
  menu.popup({ window: BrowserWindow.fromWebContents(event.sender) })
})
```

Notes on macOS Application Menu

macOS has a completely different style of application menu from Windows and Linux. Here are some notes on making your app's menu more native-like.

Standard Menus

On macOS there are many system-defined standard menus, like the `Services` and `Windows` menus. To make your menu a standard menu, you should set your menu's `role` to one of the following and Electron will recognize them and make them become standard menus:

- `window`
- `help`
- `services`

Standard Menu Item Actions

macOS has provided standard actions for some menu items, like `About xxx`, `Hide xxx`, and `Hide Others`. To set the action of a menu item to a standard action, you should set the `role` attribute of the menu item.

Main Menu's Name

On macOS the label of the application menu's first item is always your app's name, no matter what label you set. To change it, modify your app bundle's `Info.plist` file. See **About Information Property List Files** for more information.

Menu Sublabels

Menu sublabels, or **subtitles**, can be added to menu items using the `sublabel` option. Below is an example based on the renderer example above:

```
// main
ipcMain.on('show-context-menu', (event) => {
  const template = [
    {
      label: 'Menu Item 1',
      sublabel: 'Subtitle 1',
      click: () => { event.sender.send('context-menu-command', 'menu-item-1') }
    },
    { type: 'separator' },
    { label: 'Menu Item 2', sublabel: 'Subtitle 2', type: 'checkbox', checked:
true }
  ]
  const menu = Menu.buildFromTemplate(template)
  menu.popup({ window: BrowserWindow.fromWebContents(event.sender) })
})
```

Setting Menu for Specific Browser Window (*Linux* *Windows*)

The `setMenu` **method** of browser windows can set the menu of certain browser windows.

Menu Item Position

You can make use of `before`, `after`, `beforeGroupContaining`, `afterGroupContaining` and `id` to control how the item will be placed when building a menu with `Menu.buildFromTemplate`.

- `before` - Inserts this item before the item with the specified id. If the referenced item doesn't exist the item will be inserted at the end of the menu. Also implies that the menu item in question should be placed in the same “group” as the item.
- `after` - Inserts this item after the item with the specified id. If the referenced item doesn't exist the item will be inserted at the end of the menu. Also implies that the menu item in question should be placed in the same “group” as the item.
- `beforeGroupContaining` - Provides a means for a single context menu to declare the placement of their containing group before the containing group of the item with the specified id.
- `afterGroupContaining` - Provides a means for a single context menu to declare the placement of their containing group after the containing group of the item with the specified id.

By default, items will be inserted in the order they exist in the template unless one of the specified positioning keywords is used.

Examples

Template:

```
[
  { id: '1', label: 'one' },
  { id: '2', label: 'two' },
  { id: '3', label: 'three' },
  { id: '4', label: 'four' }
]
```

Menu:

```
- 1
- 2
- 3
- 4
```

Template:

```
[
  { id: '1', label: 'one' },
  { type: 'separator' },
  { id: '3', label: 'three', beforeGroupContaining: ['1'] },
  { id: '4', label: 'four', afterGroupContaining: ['2'] },
  { type: 'separator' },
  { id: '2', label: 'two' }
]
```

Menu:

```
- 3
- 4
- ---
- 1
- ---
- 2
```

Template:

```
[
  { id: '1', label: 'one', after: ['3'] },
  { id: '2', label: 'two', before: ['1'] },
  { id: '3', label: 'three' }
]
```

Menu:

```
- ---
- 3
- 2
- 1
```

Edit this page