ElectronJS doc translation in bangla

keshab patra

Contents

Chapter 1

Get started

Chapter 2

Processes in Electron

2.1 Process Model

2.2 ইন্টার-প্রসেস যোগাযোগ (Inter-Process Communication)

ইন্টার-প্রসেস যোগাযোগ (IPC) ইলেকট্রনের ফিচার সমৃদ্ধ ডেস্কটপ অ্যাপ্লিকেশন তৈরির একটি মূল অংশ। কারণ ইলেকট্রনের প্রসেস মডেলে মেইন এবং রেন্ডারার প্রসেসের বিভিন্ন দায়িত্ব থাকে, তাই অনেক সাধারণ কাজ সম্পন্ন করার জন্য IPC একমাত্র উপায়, যেমন UI থেকে একটি নেটিভ API কল করা বা নেটিভ মেনু থেকে আপনার ওয়েব কন্টেন্টে পরিবর্তন আনতে।

IPC চ্যানেল (IPC Channels)

ইলেকট্রনে, প্রসেসগুলি ডেভেলপার দ্বারা সংজ্ঞায়িত "চ্যানেল" এর মাধ্যমে ipcMain এবং ipcRenderer মডিউলগুলির সাহায্যে বার্তা প্রেরণ করে যোগাযোগ করে। এই চ্যানেলগুলি ইচ্ছেমতো নামকরণ করা যায় এবং দ্বিমুখী হয় (একই চ্যানেল নাম উভয় মডিউলে ব্যবহার করা যেতে পারে)।

এই প্রসঙ্গে, আমরা কয়েকটি মৌলিক IPC প্যাটার্ন নিয়ে আলোচনা করব এবং বাস্তব উদাহরণ দেখাব যা আপনার অ্যাপ কোডের জন্য রেফারেন্স হিসাবে ব্যবহার করা যেতে পারে।

2.2.1 কন্টেক্সট-আইসোলেটেড প্রসেস বোঝা (Understanding Context-Isolated Processes)

বাস্তবায়ন বিশদ বিবরণে যাওয়ার আগে, আপনাকে একটি প্রিলোড স্ক্রিপ্ট ব্যবহার করে একটি কন্টেক্সট-আইসোলেটেড রেন্ডারার প্রসেসে Node.js এবং Electron মডিউল আমদানির ধারণার সাথে পরিচিত হতে হবে।

ইলেকট্রনের প্রসেস মডেলের সম্পূর্ণ ওভারভিউয়ের জন্য, প্রসেস মডেল ডকুমেন্টেশন পড়তে পারেন। আপনার প্রিলোড স্ক্রিপ্ট থেকে contextBridge মডিউল ব্যবহার করে API উন্মোচনের একটি প্রাথমিক ধারণা পেতে context isolation টিউটোরিয়াল দেখুন।

প্যাটার্ন 1: রেন্ডারার থেকে মেইন (একমুখী)

রেন্ডারার প্রসেস থেকে মেইন প্রসেসে একমুখী IPC বার্তা পাঠানোর জন্য, আপনি `ipcRenderer.send` API ব্যবহার করতে পারেন যা মেইন প্রসেসে `ipcMain.on` API দ্বারা গ্রহণ করা হয়।

সাধারণত আপনি এই প্যাটার্নটি আপনার ওয়েব কন্টেন্ট থেকে একটি মেইন প্রসেস API কল করতে ব্যবহার করেন। আমরা এই প্যাটার্নটি প্রদর্শন করব একটি সাধারণ অ্যাপ তৈরি করে যা প্রোগ্রাম্যাটিকভাবে তার উইন্ডোর শিরোনাম পরিবর্তন করতে পারে।

1. `ipcMain.on` ব্যবহার করে ইভেন্ট শোনা (Listen for Events with ipcMain.on)

Listing 2.1: main.js (Main Process)

```
const { app, BrowserWindow, ipcMain } = require('electron')
const path = require('node:path')

function handleSetTitle (event, title) {
  const webContents = event.sender
  const win = BrowserWindow.fromWebContents(webContents)
  win.setTitle(title)
}

function createWindow () {
  const mainWindow = new BrowserWindow({
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })
  mainWindow.loadFile('index.html')
}

app.whenReady().then(() => {
  ipcMain.on('set-title', handleSetTitle)
  createWindow()
})
```

2. ipcRenderer.send উন্মোচন করুন প্রিলোড স্ক্রিপ্টের মাধ্যমে (Expose ipcRenderer.send via Preload)

Listing 2.2: preload.js (Preload Script)

```
const { contextBridge, ipcRenderer } = require('electron')

contextBridge.exposeInMainWorld('electronAPI', {
  setTitle: (title) => ipcRenderer.send('set-title', title)
})
```

3. রেন্ডারার প্রসেস UI তৈরি করুন (Build the Renderer Process UI)

Listing 2.3: index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src '
    self'">
<title>Hello World!</title>
</head>
<body>
Title: <input id="title"/>
<button id="btn" type="button">Set</button>
<script src="./renderer.js"></script>
</body>
</html>
```

Listing 2.4: renderer.js (Renderer Process)

```
const setButton = document.getElementById('btn')
const titleInput = document.getElementById('title')

setButton.addEventListener('click', () => {
```

```
  const title = titleInput.value
  window.electronAPI.setTitle(title)
})
```

প্যাটার্ন 2: রেন্ডারার থেকে মেইন (দ্বিমুখী)

দ্বিমুখী IPC এর সাধারণ ব্যবহার হল মেইন প্রসেস মডিউলকে আপনার রেন্ডারার প্রসেস কোড থেকে কল করা এবং একটি ফলাফলের জন্য অপেক্ষা করা। এটি `ipcRenderer.invoke` এবং `ipcMain.handle` ব্যবহার করে করা যেতে পারে।

পরবর্তী উদাহরণে, আমরা রেন্ডারার প্রসেস থেকে একটি নেটিভ ফাইল ডায়লগ খুলব এবং নির্বাচিত ফাইলের পথ ফেরত দেব।

1. `ipcMain.handle` ব্যবহার করে ইভেন্ট শোনা (Listen for Events with ipcMain.handle)

2. `ipcRenderer.invoke` উন্মোচন করুন প্রিলোড স্ক্রিপ্টের মাধ্যমে (Expose ipcRenderer.invoke via Preload)

3. রেন্ডারার প্রসেস UI তৈরি করুন (Build the Renderer Process UI)

এই ডেমোর জন্য, আপনাকে আপনার মেইন প্রসেস, রেন্ডারার প্রসেস, এবং প্রিলোড স্ক্রিপ্টে কোড যোগ করতে হবে।

2.3 Process Sandboxing

2.4 Message Ports

MessagePorts হল একটি ওয়েব ফিচার যা বিভিন্ন কনটেক্সটের মধ্যে বার্তা প্রেরণের অনুমতি দেয়। এটি **window.postMessage**-এর মতো, তবে ভিন্ন চ্যানেলে কাজ করে।

এই ডকুমেন্টের লক্ষ্য হল Electron কীভাবে Channel Messaging মডেল প্রসারিত করে তা ব্যাখ্যা করা এবং কীভাবে আপনি আপনার অ্যাপে MessagePorts ব্যবহার করতে পারেন তার কিছু উদাহরণ দেওয়া।

নিচে একটি সংক্ষিপ্ত উদাহরণ দেওয়া হলো যেখানে MessagePort কী এবং এটি কীভাবে কাজ করে তা দেখানো হয়েছে:

Listing 2.5: renderer.js (Renderer Process)

```
// MessagePorts are created in pairs. A connected pair of message ports is
// called a channel.
const channel = new MessageChannel()

// The only difference between port1 and port2 is in how you use them. Messages
// sent to port1 will be received by port2 and vice-versa.
const port1 = channel.port1
const port2 = channel.port2

// It's OK to send a message on the channel before the other end has registered
// a listener. Messages will be queued until a listener is registered.
port2.postMessage({ answer: 42 })

// Here we send the other end of the channel, port1, to the main process. It's
// also possible to send MessagePorts to other frames, or to Web Workers, etc.
ipcRenderer.postMessage('port', null, [port1])
```

Listing 2.6: main.js (Main Process)

```
// In the main process, we receive the port.
ipcMain.on('port', (event) => {
  // When we receive a MessagePort in the main process, it becomes a
  // MessagePortMain.
```

```
    const port = event.ports[0]

    // MessagePortMain uses the Node.js-style events API, rather than the
    // web-style events API. So .on('message', ...) instead of .onmessage = ...
    port.on('message', (event) => {
      // data is { answer: 42 }
      const data = event.data
    })

    // MessagePortMain queues messages until the .start() method has been called.
    port.start()
  })
```

Channel Messaging API ডকুমেন্টেশন MessagePorts কীভাবে কাজ করে তা শিখতে একটি দুর্দান্ত উপায়।

2.4.1 মেইন প্রসেসে MessagePorts

Renderer-এ, **MessagePort** ক্লাসটি ওয়েবের মতোই আচরণ করে। তবে মেইন প্রসেস কোনো ওয়েব পেজ নয়—এটিতে Blink ইন্টিগ্রেশন নেই, তাই এখানে MessagePort বা MessageChannel ক্লাস উপস্থিত নেই।

মেইন প্রসেসে MessagePorts পরিচালনা এবং ব্যবহার করার জন্য, Electron দুটি নতুন ক্লাস যুক্ত করেছে:

- MessagePortMain
- MessageChannelMain

এই ক্লাসগুলো Renderer-এর MessagePort এবং MessageChannel-এর মতোই কাজ করে।

MessagePort অবজেক্ট Renderer বা Main Process যে কোনো একটিতে তৈরি করা যেতে পারে এবং **ipcRenderer.postMessage** এবং **WebContents.postMessage** মেথড ব্যবহার করে একে একে পাঠানো সম্ভব।

তবে মনে রাখবেন:

- সাধারণ IPC মেথড (যেমন **send** এবং **invoke**) ব্যবহার করে MessagePort পাঠানো যায় না।
- শুধুমাত্র `postMessage` মেথড ব্যবহার করেই MessagePort ট্রান্সফার করা সম্ভব।

মেইন প্রসেসের মাধ্যমে MessagePort পাস করে, আপনি দুটি পৃষ্ঠার মধ্যে সংযোগ স্থাপন করতে পারেন, যা অন্যথায় একে অপরের সাথে যোগাযোগ করতে পারত না (যেমন same-origin restrictions এর কারণে)।

2.4.2 এক্সটেনশন: `close` ইভেন্ট

Electron **MessagePort**-কে আরও কার্যকর করার জন্য **একটি নতুন ফিচার** যোগ করেছে, যা ওয়েবে নেই। এটি হল **`close` ইভেন্ট**, যা তখন **emit** হয় যখন **চ্যানেলের অপর প্রান্ত বন্ধ হয়ে যায়**।

এছাড়াও, **MessagePorts স্বয়ংক্রিয়ভাবে বন্ধ হতে পারে** যদি সেগুলো **garbage collection**-এর মাধ্যমে অপসারিত হয়।

আপনি **`close` ইভেন্ট** শুনতে পারেন:

1. `port.onclose` ব্যবহার করে
2. `port.addEventListener('close', ...)` মেথড ব্যবহার করে

Main Process-এ `close` ইভেন্ট শোনার জন্য `port.on('close', ...)` মেথড ব্যবহার করা হয়।

2.4.3 ব্যবহারের উদাহরণ

দুটি Renderer-এর মধ্যে MessageChannel সেটআপ করা

এই উদাহরণে, Main Process একটি MessageChannel তৈরি করবে এবং প্রতিটি port আলাদা Renderer Process-এ পাঠাবে।

এতে Main Process-কে মধ্যস্থতাকারী হিসেবে ব্যবহার না করেই দুটি Renderer নিজেদের মধ্যে বার্তা পাঠাতে পারবে।

Listing 2.7: main.js (Main Process)

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

app.whenReady().then(async () => {
  // create the windows.
  const mainWindow = new BrowserWindow({
    show: false,
    webPreferences: {
      contextIsolation: false,
      preload: 'preloadMain.js'
    }
  })

  const secondaryWindow = new BrowserWindow({
    show: false,
    webPreferences: {
      contextIsolation: false,
      preload: 'preloadSecondary.js'
    }
  })

  // set up the channel.
  const { port1, port2 } = new MessageChannelMain()

  // once the webContents are ready, send a port to each webContents with postMessage
     .
  mainWindow.once('ready-to-show', () => {
    mainWindow.webContents.postMessage('port', null, [port1])
  })

  secondaryWindow.once('ready-to-show', () => {
    secondaryWindow.webContents.postMessage('port', null, [port2])
  })
})
```

এরপর, আপনার **preload script-এ IPC** এর মাধ্যমে **port** গ্রহণ করুন এবং **listener** সেটআপ করুন।

Preload Scripts (preloadMain.js এবং preloadSecondary.js)

```
//preloadMain.js and preloadSecondary.js (Preload scripts)
const { ipcRenderer } = require('electron')

ipcRenderer.on('port', e => {
  // port received, make it globally available.
  window.electronMessagePort = e.ports[0]

  window.electronMessagePort.onmessage = messageEvent => {
    // handle message
  }
})
```

এই উদাহরণে, **messagePort** সরাসরি **window অবজেক্টের সাথে যুক্ত (bound)** করা হয়েছে। তবে, **contextIsolation** ব্যবহার করে এবং প্রত্যাশিত প্রতিটি বার্তার জন্য **contextBridge**-এর নির্দিষ্ট কল সেটআপ করা ভালো। তবে, এই উদাহরণকে সহজ রাখার জন্য আমরা তা করিনি।

আপনি **context isolation**-এর একটি উদাহরণ এই পৃষ্ঠার পরবর্তী অংশে "**একটি context-isolated পৃষ্ঠার প্রধান প্রসেস এবং প্রধান ওয়ার্ল্ডের মধ্যে সরাসরি যোগাযোগ**" শিরোনামে খুঁজে পেতে পারেন।

এর মানে হল `window.electronMessagePort` গ্লোবালভাবে উপলব্ধ এবং আপনি **আপনার অ্যাপের যেকোনো স্থান থেকে** `postMessage` কল করে **অন্য Renderer-এ বার্তা পাঠাতে পারেন**।

Listing 2.8: Renderer Process (renderer.js)

```
// elsewhere in your code to send a message to the other renderers message handler
```

```
window.electronMessagePort.postMessage('ping')
```

2.4.4 ওয়ার্কার প্রসেস

এই উদাহরণে, আপনার অ্যাপে একটি ওয়ার্কার প্রসেস রয়েছে, যা একটি লুকানো (hidden) উইন্ডো হিসাবে কাজ করে।

আপনি চান যে অ্যাপের মূল পৃষ্ঠা সরাসরি ওয়ার্কার প্রসেসের সাথে যোগাযোগ করতে পারে, যাতে প্রধান প্রসেসের মাধ্যমে বার্তা পাঠানোর ফলে সৃষ্ট অতিরিক্ত পারফরম্যান্স ওভারহেড এড়ানো যায়।

Listing 2.9: Main Process (main.js)

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

app.whenReady().then(async () => {
  // The worker process is a hidden BrowserWindow, so that it will have access
  // to a full Blink context (including e.g. <canvas>, audio, fetch(), etc.)
  const worker = new BrowserWindow({
    show: false,
    webPreferences: { nodeIntegration: true }
  })
  await worker.loadFile('worker.html')

  // The main window will send work to the worker process and receive results
  // over a MessagePort.
  const mainWindow = new BrowserWindow({
    webPreferences: { nodeIntegration: true }
  })
  mainWindow.loadFile('app.html')

  // We can't use ipcMain.handle() here, because the reply needs to transfer a
  // MessagePort.
  // Listen for message sent from the top-level frame
  mainWindow.webContents.mainFrame.ipc.on('request-worker-channel', (event) => {
    // Create a new channel ...
    const { port1, port2 } = new MessageChannelMain()
    // ... send one end to the worker ...
    worker.webContents.postMessage('new-client', null, [port1])
    // ... and the other end to the main window.
    event.senderFrame.postMessage('provide-worker-channel', null, [port2])
    // Now the main window and the worker can communicate with each other
    // without going through the main process!
  })
})
```

Listing 2.10: Worker Process (worker.html)

```
// worker.html
<script>
const { ipcRenderer } = require('electron')

const doWork = (input) => {
  // Something cpu-intensive.
  return input * 2
}

// We might get multiple clients, for instance if there are multiple windows,
// or if the main window reloads.
ipcRenderer.on('new-client', (event) => {
  const [ port ] = event.ports
  port.onmessage = (event) => {
    // The event data can be any serializable object (and the event could even
    // carry other MessagePorts with it!)
    const result = doWork(event.data)
    port.postMessage(result)
```

```
  }
})
</script>
```

Listing 2.11: App.html

```
// app.html
<script>
const { ipcRenderer } = require('electron')

// We request that the main process sends us a channel we can use to
// communicate with the worker.
ipcRenderer.send('request-worker-channel')

ipcRenderer.once('provide-worker-channel', (event) => {
  // Once we receive the reply, we can take the port...
  const [ port ] = event.ports
  // ... register a handler to receive results ...
  port.onmessage = (event) => {
    console.log('received result:', event.data)
  }
  // ... and start sending it work!
  port.postMessage(21)
})
</script>
```

2.4.5 প্রত্যুত্তর স্ট্রিম (Reply Streams)

Electron-এর বিল্ট-ইন **IPC মেথডগুলো সাধারণত দুটি মোড সমর্থন করে**:

1. **Fire-and-forget** (যেমন `send`) – যেখানে বার্তা পাঠানো হয় কিন্তু প্রতিক্রিয়া আশা করা হয় না।
2. **Request-response** (যেমন `invoke`) – যেখানে একটি অনুরোধ পাঠানো হলে একটি নির্দিষ্ট প্রতিক্রিয়া ফেরত আসে।

তবে, MessageChannels ব্যবহার করে আপনি "**response stream**" বাস্তবায়ন করতে পারেন, যেখানে **একটি মাত্র অনুরোধের জন্য ধারাবাহিকভাবে ডাটা স্ট্রিম আকারে পাঠানো হয়**।

Listing 2.12: Renderer Process (renderer.js)

```
// renderer.js (Renderer Process)
const makeStreamingRequest = (element, callback) => {
  // MessageChannels are lightweight--it's cheap to create a new one for each
  // request.
  const { port1, port2 } = new MessageChannel()

  // We send one end of the port to the main process ...
  ipcRenderer.postMessage(
  'give-me-a-stream',
  { element, count: 10 },
  [port2]
  )

  // ... and we hang on to the other end. The main process will send messages
  // to its end of the port, and close it when it's finished.
  port1.onmessage = (event) => {
    callback(event.data)
  }
  port1.onclose = () => {
    console.log('stream ended')
  }
}
```

```
makeStreamingRequest(42, (data) => {
  console.log('got response data:', data)
})
// We will see "got response data: 42" 10 times.
```

Listing 2.13: Main Process (main.js)

```
// main.js (Main Process)
ipcMain.on('give-me-a-stream', (event, msg) => {
  // The renderer has sent us a MessagePort that it wants us to send our
  // response over.
  const [replyPort] = event.ports

  // Here we send the messages synchronously, but we could just as easily store
  // the port somewhere and send messages asynchronously.
  for (let i = 0; i < msg.count; i++) {
    replyPort.postMessage(msg.element)
  }

  // We close the port when we're done to indicate to the other end that we
  // won't be sending any more messages. This isn't strictly necessary--if we
  // didn't explicitly close the port, it would eventually be garbage
  // collected, which would also trigger the 'close' event in the renderer.
  replyPort.close()
})
```

2.4.6 একটি Context-isolated পৃষ্ঠার প্রধান প্রসেস এবং প্রধান ওয়ার্ল্ডের মধ্যে সরাসরি যোগাযোগ

যখন **context isolation** সক্রিয় থাকে, তখন প্রধান প্রসেস থেকে Renderer-এ পাঠানো IPC বার্তাগুলো **isolated world**-এ পৌঁছায়, প্রধান world-এ নয়।

তবে, কিছু ক্ষেত্রে আপনি বার্তাগুলো সরাসরি প্রধান world-এ পাঠাতে চাইতে পারেন, যাতে **isolated world**-এর মধ্য দিয়ে যেতে না হয়।

Listing 2.14: main.js (Main Process)

```
const { BrowserWindow, app, MessageChannelMain } = require('electron')
const path = require('node:path')

app.whenReady().then(async () => {
  // Create a BrowserWindow with contextIsolation enabled.
  const bw = new BrowserWindow({
    webPreferences: {
      contextIsolation: true,
      preload: path.join(__dirname, 'preload.js')
    }
  })
  bw.loadURL('index.html')

  // We'll be sending one end of this channel to the main world of the
  // context-isolated page.
  const { port1, port2 } = new MessageChannelMain()

  // It's OK to send a message on the channel before the other end has
  // registered a listener. Messages will be queued until a listener is
  // registered.
  port2.postMessage({ test: 21 })

  // We can also receive messages from the main world of the renderer.
  port2.on('message', (event) => {
    console.log('from renderer main world:', event.data)
```

```
  })
  port2.start()

  // The preload script will receive this IPC message and transfer the port
  // over to the main world.
  bw.webContents.postMessage('main-world-port', null, [port1])
})
```

Listing 2.15: preload.js (Preload Script)

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

// We need to wait until the main world is ready to receive the message before
// sending the port. We create this promise in the preload so it's guaranteed
// to register the onload listener before the load event is fired.
const windowLoaded = new Promise(resolve => {
  window.onload = resolve
})

ipcRenderer.on('main-world-port', async (event) => {
  await windowLoaded
  // We use regular window.postMessage to transfer the port from the isolated
  // world to the main world.
  window.postMessage('main-world-port', '*', event.ports)
})
```

Listing 2.16: index.html

```
// index.html
<script>
window.onmessage = (event) => {
  // event.source === window means the message is coming from the preload
  // script, as opposed to from an <iframe> or other source.
  if (event.source === window && event.data === 'main-world-port') {
    const [ port ] = event.ports
    // Once we have the port, we can communicate directly with the main
    // process.
    port.onmessage = (event) => {
      console.log('from main process:', event.data)
      port.postMessage(event.data.test * 2)
    }
  }
}
</script>
```

Chapter 3

API

3.1 BroswerWindow

Here's the provided '.md' content converted to LaTeX format:

প্রক্রিয়া: মেইন (Main)

এই মডিউলটি ব্যবহার করা যাবে না যতক্ষণ না অ্যাপ মডিউলের `ready` ইভেন্ট প্রকাশিত হয়।

```
// In the main process
const { BrowserWindow } = require('electron')

const win = new BrowserWindow({ width: 800, height: 600 })

// Load a remote URL
win.loadURL('https://github.com')

// Or load a local HTML file
win.loadFile('index.html')
```

3.1.1 উইন্ডো কাস্টমাইজেশন

`BrowserWindow` ক্লাসটি অ্যাপের উইন্ডোগুলোর চেহারা এবং আচরণ পরিবর্তনের জন্য বিভিন্ন উপায় উন্মুক্ত করে। আরও বিস্তারিত জানতে Window Customization টিউটোরিয়াল দেখুন।

3.1.2 উইন্ডো দেখানোর উপযুক্ত উপায়

একটি উইন্ডোতে সরাসরি page লোড করার সময়, ব্যবহারকারীরা pageটি ধাপে ধাপে লোড হতে দেখতে পারেন, যা একটি নেটিভ অ্যাপের জন্য ভালো অভিজ্ঞতা নয়। একটি ঝলক ছাড়া উইন্ডো দেখানোর জন্য দুটি সমাধান রয়েছে।

ready-to-show ইভেন্ট ব্যবহার করা

যদি উইন্ডোটি এখনও দেখানো না হয়ে থাকে, তাহলে `ready-to-show` ইভেন্টটি প্রকাশিত হবে যখন রেন্ডারার প্রক্রিয়া প্রথমবারের মতো পৃষ্ঠাটি রেন্ডার করবে। এই ইভেন্টের পরে উইন্ডো দেখালে কোনো দৃশ্যমান ঝলক থাকবে না:

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

win.once('ready-to-show', () => {
  win.show()
})
```

এই ইভেন্টটি সাধারণত `did-finish-load` ইভেন্টের পরে প্রকাশিত হয়, তবে অনেক রিমোট রিসোর্সসহ পৃষ্ঠাগুলোর ক্ষেত্রে এটি `did-finish-load` ইভেন্টের আগেও প্রকাশিত হতে পারে।

দ্রষ্টব্য: এই ইভেন্টটি ব্যবহার করলে, যদিও `show: false` সেট করা থাকে, তবুও রেন্ডারারকে দৃশ্যমান হিসাবে গণ্য করা হবে এবং এটি রেন্ডার করবে। যদি `paintWhenInitiallyHidden: false` সেট করা থাকে, তবে এই ইভেন্টটি কখনও প্রকাশিত হবে না।

backgroundColor প্রোপার্টি সেট করা

যদি অ্যাপটি জটিল হয়, তাহলে `ready-to-show` ইভেন্ট অনেক দেরিতে প্রকাশিত হতে পারে, যা অ্যাপটিকে ধীর মনে করায়। এই ক্ষেত্রে, উইন্ডোটি তৎক্ষণাৎ দেখানোর পরামর্শ দেওয়া হয় এবং অ্যাপের ব্যাকগ্রাউন্ডের সাথে মিল রেখে একটি `backgroundColor` সেট করা যেতে পারে:

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

const win = new BrowserWindow({ backgroundColor: '#2e2c29' })
win.loadURL('https://github.com')
```

যারা `ready-to-show` ইভেন্ট ব্যবহার করে, তাদের জন্যও `backgroundColor` সেট করাই সুপারিশ করা হয়, যাতে অ্যাপটি আরও নেটিভ মনে হয়।

কিছু বৈধ `backgroundColor` মানের উদাহরণ:

```
const win = new BrowserWindow()
win.setBackgroundColor('hsl(230, 100%, 50%)')
win.setBackgroundColor('rgb(255, 145, 145)')
win.setBackgroundColor('#ff00a3')
win.setBackgroundColor('blueviolet')
```

আরও বিস্তারিত জানতে দেখুন win.setBackgroundColor এর বৈধ অপশন।

3.1.3 প্যারেন্ট এবং চাইল্ড উইন্ডো

`parent` অপশন ব্যবহার করে চাইল্ড উইন্ডো তৈরি করা যায়:

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

const top = new BrowserWindow()
const child = new BrowserWindow({ parent: top })
child.show()
top.show()
```

চাইল্ড উইন্ডো সর্বদা `top` উইন্ডোর উপরে প্রদর্শিত হবে।

3.1.4 মডাল উইন্ডো (Modal Window)

একটি মডাল উইন্ডো হল এমন একটি চাইল্ড উইন্ডো যা প্যারেন্ট উইন্ডোকে নিষ্ক্রিয় করে। একটি মডাল উইন্ডো তৈরি করতে `parent` এবং `modal` উভয় অপশন সেট করতে হবে:

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

const top = new BrowserWindow()
const child = new BrowserWindow({ parent: top, modal: true, show: false })
child.loadURL('https://github.com')
child.once('ready-to-show', () => {
 child.show()
})
```

3.1.5 পৃষ্ঠা দৃশ্যমানতা (Page Visibility)

Page Visibility API নিম্নলিখিতভাবে কাজ করে:

- **সকল প্ল্যাটফর্মে**, দৃশ্যমানতার অবস্থা নির্ধারণ করা হয় উইন্ডোটি লুকানো বা মিনিমাইজ করা হয়েছে কিনা।
- **macOS-এ**, দৃশ্যমানতার অবস্থা **occlusion state** ট্র্যাক করে। যদি উইন্ডোটি অন্য একটি উইন্ডো দ্বারা সম্পূর্ণভাবে আচ্ছাদিত হয়, তাহলে এটি **hidden** হিসেবে গণ্য হবে। অন্য প্ল্যাটফর্মগুলিতে, উইন্ডো কেবল **hidden** হবে যখন এটি মিনিমাইজ করা হবে বা `win.hide()` দ্বারা লুকানো হবে।
- যদি একটি `BrowserWindow` **show: false** সহ তৈরি করা হয়, তাহলে উইন্ডোটি লুকানো থাকলেও প্রাথমিকভাবে এটি **visible** হিসেবে গণ্য হবে।
- যদি `backgroundThrottling` নিষ্ক্রিয় করা হয়, তাহলে উইন্ডো **মিনিমাইজ, occluded বা hidden** হলেও দৃশ্যমান থাকবে।
- **বিদ্যুৎ খরচ কমানোর জন্য**, যখন দৃশ্যমানতার অবস্থা **hidden** হয়, তখন ব্যয়বহুল অপারেশনগুলি বিরতি দেওয়ার পরামর্শ দেওয়া হয়।

3.1.6 প্ল্যাটফর্ম-সংক্রান্ত নোটিশ

- **macOS-এ**, মডাল উইন্ডোগুলি **parent উইন্ডোর সাথে সংযুক্ত শীট** (sheets) হিসাবে প্রদর্শিত হবে।
- **macOS-এ**, যখন প্যারেন্ট উইন্ডো সরানো হয়, তখন চাইল্ড উইন্ডো তার আপেক্ষিক অবস্থান বজায় রাখবে। তবে **Windows এবং Linux-এ**, চাইল্ড উইন্ডো প্যারেন্ট উইন্ডোর সাথে সরবে না।
- **Linux-এ**, মডাল উইন্ডোর ধরণ **dialog** এ পরিবর্তিত হবে।
- **Linux-এ**, অনেক **ডেস্কটপ এনভায়রনমেন্ট** মডাল উইন্ডো লুকানোর সমর্থন করে না।

3.1.7 ক্লাস: BrowserWindow extends BaseWindow

ব্রাউজার উইন্ডো তৈরি ও নিয়ন্ত্রণ করুন।

প্রক্রিয়া: মূল (Main)
`BrowserWindow` একটি `EventEmitter`।
এটি `options` দ্বারা নির্ধারিত নেটিভ বৈশিষ্ট্য সহ একটি নতুন `BrowserWindow` তৈরি করে।

নতুন BrowserWindow তৈরি করা

```
new BrowserWindow([options])
```

- **options** → `BrowserWindowConstructorOptions` *(ঐচ্ছিক)*
  - **webPreferences** *(ঐচ্ছিক)* → ওয়েব পৃষ্ঠার বৈশিষ্ট্যের সেটিংস।
  - **devTools** *(ঐচ্ছিক, ডিফল্ট: true)* → `DevTools` সক্রিয় করবেন কিনা। যদি `false` সেট করা হয়, তাহলে `BrowserWindow.webContents.openDevTools()` ব্যবহার করা যাবে না।
  - **nodeIntegration** *(ঐচ্ছিক, ডিফল্ট: false)* → Node.js ইন্টিগ্রেশন সক্রিয় কিনা।
  - **nodeIntegrationInWorker** *(ঐচ্ছিক, ডিফল্ট: false)* → ওয়েব ওয়ার্কারদের জন্য Node.js ইন্টিগ্রেশন চালু করবেন কিনা। এটি সম্পর্কে বিস্তারিত জানতে দেখুন Multithreading।
  - **nodeIntegrationInSubFrames** *(ঐচ্ছিক, পরীক্ষামূলক)* → সাব-ফ্রেমের (যেমন, iframes এবং চাইল্ড উইন্ডো) জন্য Node.js ইন্টিগ্রেশন চালু করবেন কিনা।
  - **preload** *(ঐচ্ছিক)* → একটি স্ক্রিপ্ট লোড করা যা পৃষ্ঠার অন্যান্য স্ক্রিপ্ট চলার আগে চালু হবে। এটি Node.js API-তে সর্বদা অ্যাক্সেস পাবে।

- **sandbox** *(ঐচ্ছিক)* → রেন্ডারার প্রসেসকে Chromium OS-স্তরের স্যান্ডবক্সের সাথে সামঞ্জস্যপূর্ণ করবে এবং Node.js ইঞ্জিন নিষ্ক্রিয় করবে।
- **session** *(ঐচ্ছিক)* → একটি নির্দিষ্ট সেশন সেট করুন। `session` এবং `partition` একসাথে প্রদান করা হলে `session` প্রাধান্য পাবে।
- **partition** *(ঐচ্ছিক)* → session-এর পার্টিশন স্ট্রিং সেট করুন। `persist:` থাকলে এটি স্থায়ী সেশন হবে, না থাকলে ইন-মেমোরি সেশন হবে।
- **zoomFactor** *(ঐচ্ছিক, ডিফল্ট: 1.0)* → পৃষ্ঠার জুম ফ্যাক্টর সেট করুন (যেমন, `3.0` মানে 300
- **javascript** *(ঐচ্ছিক, ডিফল্ট: true)* → JavaScript চালু/বন্ধ করুন।
- **webSecurity** *(ঐচ্ছিক, ডিফল্ট: true)* → same-origin policy নিষ্ক্রিয় করুন কিনা।
- **allowRunningInsecureContent** *(ঐচ্ছিক, ডিফল্ট: false)* → HTTPS পৃষ্ঠায় HTTP কন্টেন্ট চালানোর অনুমতি দিন কিনা।
- **images** *(ঐচ্ছিক, ডিফল্ট: true)* → ছবি লোড করা চালু/বন্ধ করুন।
- **imageAnimationPolicy** *(ঐচ্ছিক, ডিফল্ট: animate)* → GIF এর মতো চিত্র অ্যানিমেশন চালানোর নিয়ম সেট করুন (`animate`, `animateOnce`, বা `noAnimation`)।
- **textAreasAreResizable** *(ঐচ্ছিক, ডিফল্ট: true)* → `<textarea>` উপাদানগুলো পুনরায় আকার পরিবর্তনযোগ্য কিনা।
- **webgl** *(ঐচ্ছিক, ডিফল্ট: true)* → WebGL সক্রিয় করুন।
- **plugins** *(ঐচ্ছিক, ডিফল্ট: false)* → প্লাগইন চালু/বন্ধ করুন।
- **experimentalFeatures** *(ঐচ্ছিক, ডিফল্ট: false)* → Chromium-এর পরীক্ষামূলক ফিচার চালু করুন।
- **scrollBounce** *(ঐচ্ছিক, শুধুমাত্র macOS, ডিফল্ট: false)* → স্ক্রল বাউন্স (রাবার ব্যান্ডিং) চালু করুন।
- **enableBlinkFeatures** *(ঐচ্ছিক)* → একটি ফিচারের তালিকা প্রদান করুন যা চালু করতে চান।
- **disableBlinkFeatures** *(ঐচ্ছিক)* → একটি ফিচারের তালিকা প্রদান করুন যা নিষ্ক্রিয় করতে চান।

ফন্ট ও এনকোডিং সেটিংস

- **defaultFontFamily** *(ঐচ্ছিক)* → ডিফল্ট ফন্ট সেট করুন।
  - **standard** *(ডিফল্ট: Times New Roman)*
  - **serif** *(ডিফল্ট: Times New Roman)*
  - **sansSerif** *(ডিফল্ট: Arial)*
  - **monospace** *(ডিফল্ট: Courier New)*
  - **cursive** *(ডিফল্ট: Script)*
  - **fantasy** *(ডিফল্ট: Impact)*
  - **math** *(ডিফল্ট: Latin Modern Math)*
- **defaultFontSize** *(ঐচ্ছিক, ডিফল্ট: 16)*
- **defaultMonospaceFontSize** *(ঐচ্ছিক, ডিফল্ট: 13)*
- **minimumFontSize** *(ঐচ্ছিক, ডিফল্ট: 0)*
- **defaultEncoding** *(ঐচ্ছিক, ডিফল্ট: ISO-8859-1)*

অন্যান্য সেটিংস

- **backgroundThrottling** *(ঐচ্ছিক, ডিফল্ট: true)* → পৃষ্ঠাটি ব্যাকগ্রাউন্ডে গেলে অ্যানিমেশন ও টাইমার সীমিত করবে কিনা।
- **offscreen** *(ঐচ্ছিক, ডিফল্ট: false)* → অফস্ক্রিন রেন্ডারিং সক্রিয় করুন।

- **contextIsolation** *(ঐচ্ছিক, ডিফল্ট: true)* → Electron API এবং preload স্ক্রিপ্ট আলাদা জাভাস্ক্রিপ্ট কনটেক্সটে চালানো হবে কিনা।
- **webviewTag** *(ঐচ্ছিক, ডিফল্ট: false)* → `<webview>` ট্যাগ ব্যবহার করতে পারবেন কিনা।
- **additionalArguments** *(ঐচ্ছিক)* → একটি অ্যারে, যা `process.argv`-এর সাথে যুক্ত হবে।
- **safeDialogs** *(ঐচ্ছিক, ডিফল্ট: false)* → নিরাপদ ডায়ালগ সক্রিয় করুন।
- **disableDialogs** *(ঐচ্ছিক, ডিফল্ট: false)* → ডায়ালগ সম্পূর্ণরূপে নিষ্ক্রিয় করুন।
- **navigateOnDragDrop** *(ঐচ্ছিক, ডিফল্ট: false)* → ড্র্যাগ এবং ড্রপ করা ফাইল বা লিংক নেভিগেট করবে কিনা।
- **autoplayPolicy** *(ঐচ্ছিক, ডিফল্ট: no-user-gesture-required)* → অটোপ্লে নীতি নির্ধারণ করুন।
- **disableHtmlFullscreenWindowResize** *(ঐচ্ছিক, ডিফল্ট: false)* → HTML ফুলস্ক্রিনে প্রবেশ করলে উইন্ডো পুনরায় আকার পরিবর্তন নিষিদ্ধ করবে কিনা।
- **accessibleTitle** *(ঐচ্ছিক)* → স্ক্রিন রিডার-এর জন্য একটি বিকল্প টাইটেল প্রদান করুন।
- **spellcheck** *(ঐচ্ছিক, ডিফল্ট: true)* → বিল্ট-ইন বানান পরীক্ষক সক্রিয় করুন।
- **enableWebSQL** *(ঐচ্ছিক, ডিফল্ট: true)* → WebSQL API চালু করবেন কিনা।
- **transparent** *(ঐচ্ছিক, ডিফল্ট: true)* → অতিথি (guest) পৃষ্ঠার জন্য ব্যাকগ্রাউন্ড স্বচ্ছ (transparent) করবেন কিনা। > **নোট:** অতিথি পৃষ্ঠার **লেখা ও ব্যাকগ্রাউন্ডের রঙ মূল (root) এলিমেন্টের রঙ স্কিম থেকে নেওয়া হবে**। যখন স্বচ্ছতা (transparency) চালু থাকবে, তখন **লেখার রঙ পরিবর্তন হবে**, কিন্তু **ব্যাকগ্রাউন্ড স্বচ্ছ থাকবে**।
- **enableDeprecatedPaste** *(ঐচ্ছিক, **অপ্রচলিত** (Deprecated), ডিফল্ট: false)* → "paste" execCommand চালু করবেন কিনা।
- **paintWhenInitiallyHidden** *(ঐচ্ছিক, ডিফল্ট: true)* → যখন উইন্ডো `show: false` দিয়ে তৈরি হয়, তখন রেন্ডারার সক্রিয় থাকবে কিনা। > **নোট:** যদি **document.visibilityState** প্রথম লোডেই সঠিকভাবে কাজ করতে হয়, তাহলে **এটি 'false' সেট করা উচিত**। **'false' সেট করলে 'ready-to-show' ইভেন্ট চালু হবে না**।

3.1.8 ইভেন্ট লিস্ট (Event List)

`new BrowserWindow` দিয়ে তৈরি অবজেক্টগুলো নিম্নলিখিত ইভেন্টগুলি উৎপন্ন করে —

নোট: কিছু ইভেন্ট নির্দিষ্ট অপারেটিং সিস্টেমের জন্য প্রযোজ্য, যা উল্লেখ করা আছে।

`page-title-updated`

রিটার্ন করে:

- `event` → ইভেন্ট অবজেক্ট
- `title` → নতুন শিরোনাম (string)
- `explicitSet` → শিরোনাম ব্যবহারকারী দ্বারা সেট করা হয়েছে কিনা (boolean)

যখন ডকুমেন্টের শিরোনাম পরিবর্তিত হয়, তখন এটি ট্রিগার হয়। যদি `event.preventDefault()` কল করা হয়, তাহলে নেটিভ উইন্ডোর শিরোনাম পরিবর্তন হবে না। `explicitSet` false হবে যদি শিরোনাম ফাইল URL থেকে স্বয়ংক্রিয়ভাবে তৈরি হয়।

close

রিটার্ন করে:

– `event` → ইভেন্ট অবজেক্ট

যখন উইন্ডো বন্ধ হতে যাচ্ছে, তখন এটি ট্রিগার হয়। এটি `beforeunload` ও `unload` ইভেন্টের আগেই চালু হয়।`event.preventDefault()` কল করলে উইন্ডো বন্ধ হওয়া বাতিল করা যাবে।

সাধারণত উইন্ডো বন্ধ হওয়া প্রতিরোধ করতে `beforeunload` ব্যবহার করুন। এটি উইন্ডো পুনরায় লোড হলেও ট্রিগার হবে। Electron-এ `undefined` ছাড়া অন্য কোনো মান ফিরিয়ে দিলে উইন্ডো বন্ধ হবে না।

```
window.onbeforeunload = (e) => {
 console.log('I do not want to be closed')

 // Unlike usual browsers that a message box will be prompted to users, returning
 // a non-void value will silently cancel the close.
 // It is recommended to use the dialog API to let the user confirm closing the
 // application.
 e.returnValue = false
}
```

> **নোট:** `window.onbeforeunload = handler` এবং `window.addEventListener('beforeunload', handler)` এর মধ্যে একটি সূক্ষ্ম পার্থক্য আছে। সুসংগত আচরণের জন্য `event.returnValue` স্পষ্টভাবে সেট করা উচিত।

‘

closed

যখন উইন্ডো সম্পূর্ণরূপে বন্ধ হয়, তখন এটি ট্রিগার হয়। এই ইভেন্ট পাওয়ার পর **উইন্ডোর রেফারেন্স মুছে ফেলা উচিত** এবং সেটি আর ব্যবহার করা উচিত নয়।

session-end ***(শুধুমাত্র Windows)***

যখন উইন্ডোর সেশন শেষ হতে যাচ্ছে, যেমন:

– **ফোর্স শাটডাউন**,
– **কম্পিউটার রিস্টার্ট**,
– **সেশন লগ-অফ**,

তখন এটি ট্রিগার হয়।

unresponsive

যখন ওয়েবপেজ অপ্রতিক্রিয়াশীল (Unresponsive) হয়ে যায়, তখন এটি ট্রিগার হয়।

‘

responsive

যখন ওয়েবপেজ আবার প্রতিক্রিয়াশীল (Responsive) হয়ে ওঠে, তখন এটি ট্রিগার হয়।

blur

যখন উইন্ডো ফোকাস হারায়, তখন এটি ট্রিগার হয়।

focus

যখন উইন্ডো ফোকাস পায়, তখন এটি ট্রিগার হয়।

show

যখন উইন্ডো দেখানো হয়, তখন এটি ট্রিগার হয়।

hide

যখন উইন্ডো লুকানো হয়, তখন এটি ট্রিগার হয়।

app-command(শুধুমাত্র Windows ও Linux)

রিটার্ন করে:

- `event` → **ইভেন্ট অবজেক্ট**
- `command` → **স্ট্রিং, কোন অ্যাপ-কমান্ড ইনভোক করা হয়েছে**

যখন কোনো অ্যাপ-কমান্ড (App Command) চালানো হয়, তখন এটি ট্রিগার হয়।

- সাধারণত মিডিয়া কী-বোর্ড শর্টকাট বা ব্রাউজার কমান্ডের জন্য ব্যবহৃত হয়।
- উইন্ডোজে "Back" বাটন যুক্ত মাউস থাকলে সেটিও এই ইভেন্ট ট্রিগার করতে পারে।
- কমান্ডের ফরম্যাট:
  - "APPCOMMAND_" অংশ সরানো হয়।
  - সব ছোট হাতের অক্ষরে রূপান্তর করা হয়।
  - আন্ডারস্কোর (_) পরিবর্তে হাইফেন (-) বসানো হয়।
  - উদাহরণ:
    - `APPCOMMAND_BROWSER_BACKWARD` → `'browser-backward'`
    - `APPCOMMAND_MEDIA_PLAY_PAUSE` → `'media-play-pause'`

```
const { BrowserWindow } = require('electron')
const win = new BrowserWindow()
win.on('app-command', (e, cmd) => {
  // Navigate the window back when the user hits their mouse back button
  if (cmd === 'browser-backward' && win.webContents.canGoBack()) {
    win.webContents.goBack()
  }
})
```

Linux-এ সমর্থিত অ্যাপ-কমান্ড (App Commands)

Linux-এ **নিম্নলিখিত অ্যাপ-কমান্ড সমর্থিত:**

- `'browser-backward'` → ব্রাউজারে পিছনে যাওয়ার কমান্ড
- `'browser-forward'` → ব্রাউজারে সামনে যাওয়ার কমান্ড

`swipe` ***(শুধুমাত্র macOS)***

রিটার্ন করে:

- `event` → ইভেন্ট অবজেক্ট
- `direction` → **স্ট্রিং (swipe-এর দিক: `'up'`, `'right'`, `'down'`, `'left'`)**

যখন তিন আঙুল দিয়ে সুইপ করা হয়, তখন এটি ট্রিগার হয়।

- পুরনো macOS-স্টাইল ট্র্যাকপ্যাড সুইপের জন্য ডিজাইন করা হয়েছে।
- এটি কাজ করার জন্য `System Preferences > Trackpad > More Gestures` থেকে `'Swipe between pages'` সেটিং 'Swipe with two or three fingers' করতে হবে।

`rotate-gesture` **(শুধুমাত্র `macOS`)**

রিটার্ন করে:

- `event` → ইভেন্ট অবজেক্ট
- `rotation` → ফ্লোট (রোটেশনের এঙ্গেল, ডিগ্রিতে)

যখন ট্র্যাকপ্যাডে ঘোরানোর (rotation) ইশারা করা হয়, তখন এটি ট্রিগার হয়।

- রোটেশন চলতে থাকলে বারবার ইভেন্ট ফায়ার হয়।
- শেষ ইভেন্টটি সবসময় `0` হবে।
- কাউন্টার-ক্লকওয়াইজ (+), ক্লকওয়াইজ (-) এঙ্গেল ভ্যালু।

`sheet-begin` ***(শুধুমাত্র macOS)***

যখন উইন্ডোতে একটি নতুন "sheet" খোলে, তখন এটি ট্রিগার হয়।

`sheet-end` ***(শুধুমাত্র macOS)***

যখন "sheet" বন্ধ হয়ে যায়, তখন এটি ট্রিগার হয়।

`new-window-for-tab` ***(শুধুমাত্র macOS)***

যখন ইউজার নেটিভ "নতুন ট্যাব" বাটনে ক্লিক করে, তখন এটি ট্রিগার হয়।

`system-context-menu` ***(শুধুমাত্র Windows)***

রিটার্ন করে:

- `event` → ইভেন্ট অবজেক্ট
- `point` → স্ক্রিন কো-অর্ডিনেট যেখানে কনটেক্সট মেনু ট্রিগার হয়েছে

যখন সিস্টেম কনটেক্সট মেনু ট্রিগার হয়, তখন এটি ইভেন্ট হয়।

- সাধারণত উইন্ডোর শিরোনাম বার বা `-webkit-app-region: drag` এর উপর রাইট-ক্লিক করলে ট্রিগার হয়।
- `event.preventDefault()` কল করলে কনটেক্সট মেনু দেখানো বন্ধ হবে।

ইভেন্ট	প্ল্যাটফর্ম	ডেসক্রিপশন
page-title-updated	All	document এর শিরোনাম পরিবর্তনে এটি ট্রিগার হয়
close	All	যখন window বন্ধ হতে যাচ্ছে, তখন এটি ট্রিগার হয়
closed	All	যখন window সম্পূর্নরূপে বন্ধ হয়, তখন এটি ট্রিগার হয়
session-end	Win	যখন window session শেষ হতে যাচ্ছে, তখন এটি ট্রিগার হয়
unresponsive	All	যখন webpage unresponsive হয়ে যায়, তখন এটি ট্রিগার হয়
responsive	All	যখন ওয়েবপেজ আবার প্রতিক্রিয়াশীল (Responsive) হয়ে ওঠে, তখন এটি ট্রিগার হয়
blur	All	যখন উইন্ডো ফোকাস হারায়, তখন এটি ট্রিগার হয়
focus	All	যখন উইন্ডো ফোকাস পায়, তখন এটি ট্রিগার হয়
show	All	যখন উইন্ডো দেখানো হয়, তখন এটি ট্রিগার হয়
hide	All	যখন উইন্ডো লুকানো হয়, তখন এটি ট্রিগার হয়
app-command	Win, Linux	যখন কোনো অ্যাপ-কমান্ড (App Command) চালানো হয়, তখন এটি ট্রিগার হয়
browser-backward	Linux	ব্রাউজারে পিছনে যাওয়ার কমান্ড
browser-forward	Linux	ব্রাউজারে সামনে যাওয়ার কমান্ড
swipe	Mac	যখন তিন আঙুল দিয়ে সুইপ করা হয়, তখন এটি ট্রিগার হয়
rotate-gesture	Mac	যখন ট্র্যাকপ্যাডে ঘোরানোর (rotation) ইশারা করা হয়, তখন এটি ট্রিগার হয়
sheet-begin	Mac	যখন উইন্ডোতে একটি নতুন "sheet" খোলে, তখন এটি ট্রিগার হয়
sheet-end	Mac	যখন "sheet" বন্ধ হয়ে যায়, তখন এটি ট্রিগার হয়
new-window-for-tab	Mac	যখন ইউজার নেটিভ "নতুন ট্যাব" বাটনে ক্লিক করে, তখন এটি ট্রিগার হয়
system-context-menu	Win	যখন সিস্টেম কনটেক্সট মেনু ট্রিগার হয়, তখন এটি ইভেন্ট হয়

3.1.9 Static Methods (BrowserWindow এর স্ট্যাটিক মেথডস)

`BrowserWindow.getAllWindows()`

সকল ওপেন করা ব্রাউজার উইন্ডোর অ্যারে রিটার্ন করে।

```
let windows = BrowserWindow.getAllWindows();
console.log(windows);
```

`BrowserWindow.getFocusedWindow()`

বর্তমানে ফোকাস থাকা উইন্ডো রিটার্ন করে, যদি না থাকে তাহলে `null`।

```
let focusedWin = BrowserWindow.getFocusedWindow();
console.log(focusedWin);
```

BrowserWindow.fromWebContents(webContents)

প্যারামিটার:

- `webContents` → একটি `WebContents` অবজেক্ট

যে উইন্ডো এই `webContents` এর মালিক, সেটি রিটার্ন করে (বা `null`)।

```
let win = BrowserWindow.fromWebContents(someWebContents);
console.log(win);
```

BrowserWindow.fromId(id)

প্যারামিটার:

- `id` → ইন্টিজার, উইন্ডোর আইডি

নির্দিষ্ট `id`-এর উইন্ডো রিটার্ন করে (বা `null`)।

```
let win = BrowserWindow.fromId(1);
console.log(win);
```

BrowserWindow.fromBrowserView(browserView)

(Deprecated) `BrowserView` ডিপ্রিকেটেড, নতুন `WebContentsView` ব্যবহার করা উচিত।

```
let win = BrowserWindow.fromBrowserView(someBrowserView);
console.log(win);
```

3.1.10 BrowserWindow এর ইনস্ট্যান্স প্রপার্টিস (Instance Properties)

```
const { BrowserWindow } = require('electron')
// In this example `win` is our instance
const win = new BrowserWindow({ width: 800, height: 600 })
win.loadURL('https://github.com')
```

1. `win.webContents` (ReadOnly)

এই উইন্ডোর মালিকানাধীন `WebContents` অবজেক্ট। ওয়েবপেজ সংক্রান্ত সব ইভেন্ট এবং অপারেশন এর মাধ্যমে পরিচালিত হয়। বিস্তারিত জানতে `webContents` ডকুমেন্টেশন দেখুন।

- `WebContents` এর method ও events সম্পর্কে বিস্তারিত জানতে [documentation]() দেখুন।

2. `win.id` (ReadOnly)

প্রত্যেক উইন্ডোর জন্য একটি ইউনিক আইডি থাকে।

3. `win.tabbingIdentifier` (macOS, ReadOnly)

কোনো `tabbingIdentifier` পাস করা থাকলে সেটি রিটার্ন করে, না থাকলে `undefined`।

4. `win.autoHideMenuBar`

`true` করলে মেনু বার অটো-হাইড হবে এবং `Alt` চাপলে দৃশ্যমান হবে।

- যদি মেনু বার আগে থেকেই দৃশ্যমান থাকে, `true` সেট করলেও তা সাথে সাথে লুকাবে না।

5. `win.simpleFullScreen`

উইন্ডো কি simple (pre-Lion) ফুলস্ক্রিন মোডে আছে কিনা, তা নির্ধারণ করে।

6. `win.fullScreen`

উইন্ডো কি ফুলস্ক্রিন মোডে আছে কিনা, তা নির্ধারণ করে।

7. `win.focusable` (Windows & macOS)

উইন্ডো কি ফোকাস করা যাবে কিনা তা নির্ধারণ করে।

8. `win.visibleOnAllWorkspaces` (macOS & Linux)

উইন্ডো কি সব ওয়ার্কস্পেসে দৃশ্যমান থাকবে কিনা তা নির্ধারণ করে।

Note: Windows-এ সবসময় `false` রিটার্ন করবে।

9. `win.shadow`

উইন্ডোতে ছায়া (shadow) থাকবে কিনা তা নির্ধারণ করে।

10. `win.menuBarVisible` (Windows & Linux)

উইন্ডোর মেনু বার দৃশ্যমান থাকবে কিনা তা নির্ধারণ করে।

- `Alt` চাপলে লুকানো মেনু বার আবার দেখা যাবে।

11. `win.kiosk`

উইন্ডো কি কিয়স্ক মোডে আছে কিনা তা নির্ধারণ করে।

12. `win.documentEdited` (macOS)

উইন্ডোর ডকুমেন্ট কি সম্পাদিত হয়েছে কিনা তা নির্ধারণ করে।

- `true` করলে টাইটেল বারে থাকা আইকন ধূসর হয়ে যাবে।

13. `win.representedFilename` (macOS)

উইন্ডোর সাথে সম্পর্কিত ফাইলের পথ (pathname) রিটার্ন করে।

ফাইলের আইকন টাইটেল বারে দেখাবে।

14. `win.title`

উইন্ডোর টাইটেল সেট বা রিটার্ন করে।

15. `win.minimizable` (macOS & Windows)

উইন্ডো কি মিনিমাইজ করা যাবে কিনা তা নির্ধারণ করে।

Linux-এ সেটার কাজ করবে না (`no-op`), তবে `getter` ঠিকমতো কাজ করবে।

16. `win.maximizable` (macOS & Windows)

উইন্ডো কি ম্যাক্সিমাইজ করা যাবে কিনা তা নির্ধারণ করে।

Linux-এ সেটার কাজ করবে না (`no-op`)।

17. `win.fullScreenable`

উইন্ডোর ম্যাক্সিমাইজ/জুম বাটন কি ফুলস্ক্রিন মোডে যাবে কিনা তা নির্ধারণ করে।

18. `win.resizable`

উইন্ডো কি ম্যানুয়ালি রিসাইজ করা যাবে কিনা তা নির্ধারণ করে।

19. `win.closable` (macOS & Windows)

উইন্ডো কি ইউজার ম্যানুয়ালি ক্লোজ করতে পারবে কিনা তা নির্ধারণ করে।

Linux-এ সেটার কাজ করবে না (`no-op`)।

20. `win.movable` (macOS & Windows)

উইন্ডো কি ম্যানুয়ালি সরানো যাবে কিনা তা নির্ধারণ করে।

Linux-এ সেটার কাজ করবে না (`no-op`)।

21. `win.excludedFromShownWindowsMenu` (macOS)

উইন্ডো কি অ্যাপের Windows মেনুতে দেখানো হবে কিনা তা নির্ধারণ করে।

ডিফল্ট ভ্যালু: `false`।

সংক্ষেপে BrowserWindow এর ইনস্ট্যান্স প্রপার্টিস

প্রপার্টি	প্ল্যাটফর্ম	ডেসক্রিপশন
`win.id`	All	উইন্ডোর ইউনিক আইডি
`win.title`	All	উইন্ডোর টাইটেল
`win.fullScreen`	All	ফুলস্ক্রিন মোড টগল করা যায়
`win.minimizable`	`macOS, Windows`	মিনিমাইজ করা যাবে কিনা
`win.maximizable`	`macOS, Windows`	ম্যাক্সিমাইজ করা যাবে কিনা
`win.resizable`	All	রিসাইজ করা যাবে কিনা
`win.closable`	`macOS, Windows`	ক্লোজ করা যাবে কিনা
`win.movable`	`macOS, Windows`	মুভ করা যাবে কিনা
`win.kiosk`	All	কিয়স্ক মোড সক্রিয় কিনা
`win.documentEdited`	`macOS`	ডকুমেন্ট সম্পাদিত কিনা
`win.excludedFromShownWindowsMenu`	`macOS`	উইন্ডো Windows মেনুতে থাকবে কিনা

```
const win = new BrowserWindow({ height: 600, width: 600 })

const template = [
{
  role: 'windowmenu'
}
]

win.excludedFromShownWindowsMenu = true

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

3.1.11 BrowserWindow এর ইনস্ট্যান্স মেথডস (Instance Methods)

1. `win.destroy()`

উইন্ডোকে জোরপূর্বক বন্ধ করে। `unload` এবং `beforeunload` ইভেন্টস ট্রিগার হয় না। `close` ইভেন্টও ট্রিগার হয় না, তবে `closed` ইভেন্ট নিশ্চিতভাবে ট্রিগার হয়।

2. `win.close()`

ব্যবহারকারী ম্যানুয়ালি ক্লোজ বাটনে ক্লিক করলে যা হয়, সেটাই করে। ওয়েবপেজ চাইলে `close` ইভেন্ট হ্যান্ডল করে বন্ধ হওয়া প্রতিরোধ করতে পারে।

3. `win.focus()`

উইন্ডোটিকে ফোকাস করে।

4. `win.blur()`

উইন্ডোর উপর থেকে ফোকাস সরিয়ে নেয়।

5. `win.isFocused()`

উইন্ডো ফোকাসড কিনা তা চেক করে। `true` বা `false` রিটার্ন করে।

6. `win.isDestroyed()`

উইন্ডো ধ্বংস (destroyed) হয়েছে কিনা তা চেক করে। `true` বা `false` রিটার্ন করে।

7. `win.show()`

উইন্ডোকে দৃশ্যমান করে এবং ফোকাস দেয়।

8. `win.showInactive()`

উইন্ডোকে দৃশ্যমান করে, কিন্তু ফোকাস দেয় না।

9. `win.hide()`

উইন্ডোকে লুকিয়ে ফেলে।

10. win.isVisible()

উইন্ডো দৃশ্যমান কিনা তা চেক করে। true বা false রিটার্ন করে।

11. win.isModal()

উইন্ডো কি মডাল মোডে আছে কিনা তা চেক করে।

12. win.maximize()

উইন্ডোটিকে ম্যাক্সিমাইজ করে। যদি উইন্ডো লুকানো থাকে, তবে এটি দেখানো হবে তবে ফোকাস দেওয়া হবে না।

13. win.unmaximize()

উইন্ডোকে আনম্যাক্সিমাইজ করে।

14. win.isMaximized()

উইন্ডো ম্যাক্সিমাইজ করা হয়েছে কিনা তা চেক করে।

15. win.minimize()

উইন্ডোটিকে মিনিমাইজ করে।

16. win.restore()

উইন্ডোটিকে মিনিমাইজ অবস্থা থেকে পুনরুদ্ধার করে।

17. win.isMinimized()

উইন্ডো মিনিমাইজ হয়েছে কিনা তা চেক করে।

18. win.setFullScreen(flag)

উইন্ডোকে ফুলস্ক্রিন মোডে নিয়ে যায় বা সেখান থেকে সরায়। macOS-এ এটি অ্যাসিনক্রোনাসভাবে ঘটে।

19. win.isFullScreen()

উইন্ডো কি ফুলস্ক্রিন মোডে আছে কিনা তা চেক করে।

20. win.setSimpleFullScreen(flag) (macOS)

উইন্ডোকে Simple Fullscreen মোডে নেয় বা সরায়। macOS 10.7 (Lion) এর আগের নেটিভ ফুলস্ক্রিন আচরণ ইমুলেট করে।

```
win.setSimpleFullScreen(true);  //
    Simple fullscreen চালু
win.setSimpleFullScreen(false); //
    Simple fullscreen বন্ধ
```

21. win.isSimpleFullScreen() (macOS)

উইন্ডো Simple Fullscreen মোডে আছে কিনা চেক করে।

```
console.log(win.isSimpleFullScreen()
    ); // true বা false
```

22. win.isNormal()

উইন্ডো কি সাধারণ অবস্থায় আছে কিনা চেক করে (না ম্যাক্সিমাইজড, না মিনিমাইজড, না ফুলস্ক্রিন)।

```
console.log(win.isNormal()); // true
    বা false
```

23. win.setAspectRatio(aspectRatio[, extraSize])

উইন্ডোর নির্দিষ্ট Aspect Ratio ধরে রাখে। extraSize প্যারামিটার ব্যবহার করে অ্যাসপেক্ট রেশিও গণনায় বাদ রাখা যায়।

```
// 16:9 অ্যাসপেক্টরেশিওসেটকরাহলো ,
    যেখানে 40px চওড়াও 50px
    লম্বাঅতিরিক্তজায়গাবাদরাখাহবে
win.setAspectRatio(16 / 9, { width:
    40, height: 50 });

// অ্যাসপেক্টরেশিওরিসেটকরতে 0 পাসকরুন
win.setAspectRatio(0);
```

24. win.setBackgroundColor(backgroundColor)

উইন্ডোর ব্যাকগ্রাউন্ড কালার সেট করে। সমর্থিত ফরম্যাট: Hex, RGB, RGBA, HSL, HSLA, CSS রঙের নাম।

```
win.setBackgroundColor('#ff0000');
    // লালরঙ
win.setBackgroundColor('rgba(0, 255,
    0, 0.5)'); // ৫০% স্বচ্ছসবুজ
win.setBackgroundColor('blueviolet')
    ; // নীলবেগুনি- রঙ
```

25. win.previewFile(path[, displayName]) (macOS)

Quick Look ব্যবহার করে একটি ফাইল প্রিভিউ করে। displayName প্যারামিটার শুধুমাত্র প্রদর্শনের জন্য।

```
win.previewFile('/Users/user/
    Documents/sample.pdf', 'My PDF
    File');
```

26. win.closeFilePreview() (macOS)

Quick Look প্যানেল বন্ধ করে।

```
win.closeFilePreview();
```

27. win.setBounds(bounds[, animate])

উইন্ডোর অবস্থান ও আকার নির্ধারণ করে।

```
win.setBounds({ x: 100, y: 50, width
    : 800, height: 600 }); //
    নতুনঅবস্থানওআকারসেটকরা
win.setBounds({ width: 1024 }); //
    শুধুমাত্রপ্রস্থপরিবর্তনকরা
```

28. win.getBounds()

উইন্ডোর বর্তমান bounds অবজেক্ট রিটার্ন করে।

```
console.log(win.getBounds());
// আউটপুট: { x: 100, y: 50, width:
    800, height: 600 }
```

29. win.getBackgroundColor()

উইন্ডোর ব্যাকগ্রাউন্ড কালার রিটার্ন করে (Hex ফরম্যাটে)।

```
console.log(win.getBackgroundColor()
    ); // #ff0000
```

---

30. win.setContentBounds(bounds[, animate])

উইন্ডোর ক্লায়েন্ট এরিয়া (ওয়েবপেজ) এর আকার ও অবস্থান নির্ধারণ করে।

```
win.setContentBounds({ x: 100, y:
    50, width: 800, height: 600 });
```

31. win.getContentBounds()

উইন্ডোর ক্লায়েন্ট এরিয়া (ওয়েবপেজ) এর বর্তমান আকার ও অবস্থান রিটার্ন করে।

32. win.getNormalBounds()

উইন্ডোর সাধারণ অবস্থার (maximized, minimized বা fullscreen ছাড়া) আকার ও অবস্থান রিটার্ন করে।

33. win.setEnabled(enable)

উইন্ডো সক্রিয় (enable) বা নিষ্ক্রিয় (disable) করে।

```
win.setEnabled(false); //
    উইন্ডোনিষ্ক্রিয়করাহলো
win.setEnabled(true);  //
    উইন্ডোসক্রিয়করাহলো
```

34. win.isEnabled()

উইন্ডো সক্রিয় কিনা চেক করে।

35. win.setSize(width, height[, animate])

উইন্ডোর আকার নির্ধারণ করে।

```
win.setSize(1024, 768);
```

36. win.getSize()

উইন্ডোর বর্তমান প্রস্থ ও উচ্চতা রিটার্ন করে।

37. win.setContentSize(width, height[, animate])

উইন্ডোর কন্টেন্ট এরিয়ার আকার নির্ধারণ করে।

BrowserWindow এর ইনস্ট্যান্স প্রপার্টিস

– `win.setContentSize(800, 600);`

38. win.getContentSize()

উইন্ডোর কন্টেন্ট এরিয়ার প্রস্থ ও উচ্চতা রিটার্ন করে।

39. win.setMinimumSize(width, height)

উইন্ডোর সর্বনিম্ন আকার নির্ধারণ করে।

```
win.setMinimumSize(400, 300);
```

40. win.getMinimumSize()

উইন্ডোর সর্বনিম্ন প্রস্থ ও উচ্চতা রিটার্ন করে।

41. win.setMaximumSize(width, height)

উইন্ডোর সর্বোচ্চ আকার নির্ধারণ করে।

```
win.setMaximumSize(1600, 1200);
```

42. win.getMaximumSize()

উইন্ডোর সর্বোচ্চ প্রস্থ ও উচ্চতা রিটার্ন করে।

43. win.setResizable(resizable)

উইন্ডো রিসাইজ করা যাবে কিনা সেট করে।

```
win.setResizable(false);
```

44. win.isResizable()

উইন্ডো রিসাইজ করা যাবে কিনা চেক করে।

45. win.setMovable(movable) (macOS, Windows)

ব্যবহারকারী উইন্ডো সরাতে পারবে কিনা নির্ধারণ করে।

```
win.setMovable(false);
```

46. win.isMovable() (macOS, Windows)

উইন্ডো সরানো যাবে কিনা চেক করে।

47. win.setMinimizable(minimizable) (macOS, Windows)

উইন্ডো মিনিমাইজ করা যাবে কিনা নির্ধারণ করে।

```
win.setMinimizable(false);
```

48. win.isMinimizable() (macOS, Windows)

উইন্ডো মিনিমাইজ করা যাবে কিনা চেক করে।

49. win.setMaximizable(maximizable) (macOS, Windows)

উইন্ডো ম্যাক্সিমাইজ করা যাবে কিনা নির্ধারণ করে।

```
win.setMaximizable(false);
```

50. win.isMaximizable() (macOS, Windows)

উইন্ডো ম্যাক্সিমাইজ করা যাবে কিনা চেক করে।

51. win.setFullScreenable(fullscreenable)

উইন্ডো ফুলস্ক্রিন করা যাবে কিনা নির্ধারণ করে।

```
win.setFullScreenable(false);
```

52. win.isFullScreenable()

উইন্ডো ফুলস্ক্রিন করা যাবে কিনা চেক করে।

53. win.setClosable(closable) (macOS, Windows)

উইন্ডো ব্যবহারকারী বন্ধ করতে পারবে কিনা নির্ধারণ করে।

```
win.setClosable(false);
```

54. win.isClosable() (macOS, Windows)

উইন্ডো বন্ধ করা যাবে কিনা চেক করে।

55. win.setHiddenInMissionControl(hidden) (macOS)

Mission Control-এ উইন্ডো লুকানো হবে কিনা নির্ধারণ করে।

```
win.setHiddenInMissionControl(true);
```

56. win.isHiddenInMissionControl() (macOS)

Mission Control-এ উইন্ডো লুকানো হবে কিনা চেক করে।

58. win.setAlwaysOnTop(flag[, level][, relativeLevel])

উইন্ডো সবসময় অন্য উইন্ডোগুলোর ওপরে থাকবে কিনা সেট করে।

```
win.setAlwaysOnTop(true);
```

59. win.isAlwaysOnTop()

উইন্ডো সবসময় টপে থাকবে কিনা চেক করে।

60. win.moveAbove(mediaSourceId)

উইন্ডোকে নির্দিষ্ট আরেকটি উইন্ডোর উপরে সরায়।

```
win.moveAbove("window:1869:0");
```

সংক্ষেপে BrowserWindow এর ইনস্ট্যান্স মেথডস

মেথড	কাজ
`win.destroy()`	জোরপূর্বক উইন্ডো বন্ধ করে
`win.close()`	উইন্ডো বন্ধ করার চেষ্টা করে
`win.focus()`	উইন্ডোতে ফোকাস দেয়
`win.blur()`	উইন্ডো থেকে ফোকাস সরিয়ে নেয়
`win.isFocused()`	উইন্ডো ফোকাসড কিনা চেক করে
`win.isDestroyed()`	উইন্ডো ধ্বংস হয়েছে কিনা চেক করে
`win.show()`	উইন্ডো দেখায় ও ফোকাস দেয়
`win.showInactive()`	উইন্ডো দেখায়, কিন্তু ফোকাস দেয় না
`win.hide()`	উইন্ডো লুকিয়ে ফেলে
`win.isVisible()`	উইন্ডো দৃশ্যমান কিনা চেক করে
`win.isModal()`	উইন্ডো মডাল কিনা চেক করে
`win.maximize()`	উইন্ডো ম্যাক্সিমাইজ করে
`win.unmaximize()`	উইন্ডো আনম্যাক্সিমাইজ করে
`win.isMaximized()`	উইন্ডো ম্যাক্সিমাইজ কিনা চেক করে
`win.minimize()`	উইন্ডো মিনিমাইজ করে
`win.restore()`	মিনিমাইজ অবস্থা থেকে পুনরুদ্ধার করে
`win.isMinimized()`	উইন্ডো মিনিমাইজ কিনা চেক করে
`win.setFullScreen(true/false)`	উইন্ডো ফুলস্ক্রিন মোডে নেয় বা সরায়
`win.isFullScreen()`	উইন্ডো ফুলস্ক্রিন কিনা চেক করে
`win.setSimpleFullScreen(flag)`	Simple Fullscreen চালু বা বন্ধ করে (macOS)
`win.isSimpleFullScreen()`	উইন্ডো Simple Fullscreen মোডে কিনা চেক করে
`win.isNormal()`	উইন্ডো সাধারণ অবস্থায় আছে কিনা চেক করে
`win.setAspectRatio(aspectRatio[, extraSize])`	নির্দিষ্ট অ্যাসপেক্ট রেশিও ধরে রাখে
`win.setBackgroundColor (backgroundColor)`	উইন্ডোর ব্যাকগ্রাউন্ড রঙ সেট করে
`win.previewFile(path[, displayName])`	Quick Look ব্যবহার করে ফাইল প্রিভিউ করে (macOS)
`win.closeFilePreview()`	Quick Look প্রিভিউ বন্ধ করে (macOS)
`win.setBounds(bounds[, animate])`	উইন্ডোর অবস্থান ও আকার পরিবর্তন করে
`win.getBounds()`	উইন্ডোর বর্তমান `bounds` রিটার্ন করে
`win.getBackgroundColor()`	উইন্ডোর ব্যাকগ্রাউন্ড রঙ রিটার্ন করে
`win.setContentBounds(bounds[, animate])`	কন্টেন্ট এরিয়ার আকার নির্ধারণ
`win.getContentBounds()`	কন্টেন্ট এরিয়ার বর্তমান আকার ও অবস্থান
`win.setSize(width, height[, animate])`	উইন্ডোর আকার নির্ধারণ
`win.getSize()`	উইন্ডোর বর্তমান আকার
`win.setResizable(resizable)`	উইন্ডো রিসাইজ করা যাবে কিনা নির্ধারণ