session

> Manage browser sessions, cookies, cache, proxy settings, etc.

Process: **Main**

The `session` module can be used to create new `Session` objects.

You can also access the `session` of existing pages by using the `session` property of **`WebContents`**, or from the `session` module.

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

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

const ses = win.webContents.session
console.log(ses.getUserAgent())
```

Methods

The `session` module has the following methods:

`session.fromPartition(partition[, options])`

- `partition` string
- `options` Object (optional)
  - `cache` boolean - Whether to enable cache. Default is `true` unless the **`--disable-http-cache` switch** is used.

Returns `Session` - A session instance from `partition` string. When there is an existing `Session` with the same `partition`, it will be returned; otherwise a new `Session` instance will be created with `options`.

If `partition` starts with `persist:`, the page will use a persistent session available to all pages in the app with the same `partition`. if there is no `persist:` prefix, the page will use an in-memory session. If the `partition` is empty then default session of the app will be returned.

To create a `Session` with `options`, you have to ensure the `Session` with the `partition` has never been used before. There is no way to change the `options` of an existing `Session` object.

`session.fromPath(path[, options])`

- `path` string
- `options` Object (optional)
  - `cache` boolean - Whether to enable cache. Default is `true` unless the **`--disable-http-cache` switch** is used.

Returns `Session` - A session instance from the absolute path as specified by the `path` string. When there is an existing `Session` with the same absolute path, it will be returned; otherwise a new `Session` instance will be created with `options`. The call will throw an error if the path is not an absolute path. Additionally, an error will be thrown if an empty string is provided.

To create a `Session` with `options`, you have to ensure the `Session` with the `path` has never been used before. There is no way to change the `options` of an existing `Session` object.

Properties

The `session` module has the following properties:

`session.defaultSession`

A `Session` object, the default session object of the app.

Class: Session

> Get and set properties of a session.

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.

You can create a `Session` object in the `session` module:

```
const { session } = require('electron')
const ses = session.fromPartition('persist:name')
console.log(ses.getUserAgent())
```

Instance Events

The following events are available on instances of `Session`:

Event: 'will-download'

Returns:

- `event` Event
- `item` **DownloadItem**
- `webContents` **WebContents**

Emitted when Electron is about to download `item` in `webContents`.

Calling `event.preventDefault()` will cancel the download and `item` will not be available from next tick of the process.

```
const { session } = require('electron')
session.defaultSession.on('will-download', (event, item, webContents) => {
  event.preventDefault()
  require('got')(item.getURL()).then((response) => {
    require('node:fs').writeFileSync('/somewhere', response.body)
  })
})
```

Event: 'extension-loaded'

Returns:

- `event` Event
- `extension` **Extension**

Emitted after an extension is loaded. This occurs whenever an extension is added to the "enabled" set of extensions. This includes:

- Extensions being loaded from `Session.loadExtension`.
- Extensions being reloaded:
  - from a crash.
  - if the extension requested it (`chrome.runtime.reload()`).

Event: 'extension-unloaded'

Returns:

- `event` Event
- `extension` **Extension**

Emitted after an extension is unloaded. This occurs when `Session.removeExtension` is called.

Event: 'extension-ready'

Returns:

- `event` Event
- `extension` **Extension**

Emitted after an extension is loaded and all necessary browser state is initialized to support the start of the extension's background page.

Event: 'file-system-access-restricted'

Returns:

- `event` Event
- `details` Object

    - `origin` string - The origin that initiated access to the blocked path.
    - `isDirectory` boolean - Whether or not the path is a directory.
    - `path` string - The blocked path attempting to be accessed.
- `callback` Function
    - `action` string - The action to take as a result of the restricted path access attempt.
        - `allow` - This will allow `path` to be accessed despite restricted status.
        - `deny` - This will block the access request and trigger an **`AbortError`**.
        - `tryAgain` - This will open a new file picker and allow the user to choose another path.

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

async function createWindow () {
  const mainWindow = new BrowserWindow()

  await mainWindow.loadURL('https://buzzfeed.com')

  session.defaultSession.on('file-system-access-restricted', async (e, details,
callback) => {
    const { origin, path } = details
    const { response } = await dialog.showMessageBox({
      message: `Are you sure you want ${origin} to open restricted path ${path}?
`,
      title: 'File System Access Restricted',
      buttons: ['Choose a different folder', 'Allow', 'Cancel'],
      cancelId: 2
    })

    if (response === 0) {
      callback('tryAgain')
    } else if (response === 1) {
      callback('allow')
    } else {
      callback('deny')
    }
  })

  mainWindow.webContents.executeJavaScript(`
    window.showDirectoryPicker({
      id: 'electron-demo',
      mode: 'readwrite',
      startIn: 'downloads',
```

```
    }).catch(e => {
      console.log(e)
    })`, true
  )
}

app.whenReady().then(() => {
  createWindow()

  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) createWindow()
  })
})

app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') app.quit()
})
```

Event: 'preconnect'

Returns:

- `event` Event
- `preconnectUrl` string - The URL being requested for preconnection by the renderer.
- `allowCredentials` boolean - True if the renderer is requesting that the connection include credentials (see the **spec** for more details.)

Emitted when a render process requests preconnection to a URL, generally due to a **resource hint**.

Event: 'spellcheck-dictionary-initialized'

Returns:

- `event` Event
- `languageCode` string - The language code of the dictionary file

Emitted when a hunspell dictionary file has been successfully initialized. This occurs after the file has been downloaded.

Event: 'spellcheck-dictionary-download-begin'

Returns:

- `event` Event
- `languageCode` string - The language code of the dictionary file

Emitted when a hunspell dictionary file starts downloading

Event: 'spellcheck-dictionary-download-success'

Returns:

- `event` Event
- `languageCode` string - The language code of the dictionary file

Emitted when a hunspell dictionary file has been successfully downloaded

Event: 'spellcheck-dictionary-download-failure'

Returns:

- `event` Event
- `languageCode` string - The language code of the dictionary file

Emitted when a hunspell dictionary file download fails. For details on the failure you should collect a netlog and inspect the download request.

Event: 'select-hid-device'

Returns:

- `event` Event
- `details` Object
  - `deviceList` **HIDDevice[]**
  - `frame` **WebFrameMain** | null - The frame initiating this event. May be `null` if accessed after the frame has either navigated or been destroyed.
- `callback` Function
  - `deviceId` string | null (optional)

Emitted when a HID device needs to be selected when a call to `navigator.hid.requestDevice` is made. `callback` should be called with `deviceId` to be selected; passing no arguments to `callback` will cancel the request. Additionally, permissioning on `navigator.hid` can be further managed by using **`ses.setPermissionCheckHandler(handler)`** and **`ses.setDevicePermissionHandler(handler)`**.

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

let win = null

app.whenReady().then(() => {
  win = new BrowserWindow()

  win.webContents.session.setPermissionCheckHandler((webContents, permission,
requestingOrigin, details) => {
    if (permission === 'hid') {
      // Add logic here to determine if permission should be given to allow HID
selection
      return true
    }
    return false
  })

  // Optionally, retrieve previously persisted devices from a persistent store
  const grantedDevices = fetchGrantedDevices()

  win.webContents.session.setDevicePermissionHandler((details) => {
    if (new URL(details.origin).hostname === 'some-host' && details.deviceType
=== 'hid') {
      if (details.device.vendorId === 123 && details.device.productId === 345) {
        // Always allow this type of device (this allows skipping the call to
`navigator.hid.requestDevice` first)
        return true
      }

      // Search through the list of devices that have previously been granted
permission
      return grantedDevices.some((grantedDevice) => {
        return grantedDevice.vendorId === details.device.vendorId &&
               grantedDevice.productId === details.device.productId &&
               grantedDevice.serialNumber && grantedDevice.serialNumber ===
details.device.serialNumber
```

```
      })
    }
    return false
  })

  win.webContents.session.on('select-hid-device', (event, details, callback) => {
    event.preventDefault()
    const selectedDevice = details.deviceList.find((device) => {
      return device.vendorId === 9025 && device.productId === 67
    })
    callback(selectedDevice?.deviceId)
  })
})
```

Event: 'hid-device-added'

Returns:

- `event` Event
- `details` Object
  - `device` **HIDDevice**ⓘ
  - `frame` **WebFrameMain** | null - The frame initiating this event. May be `null` if accessed after the frame has either navigated or been destroyed.

Emitted after `navigator.hid.requestDevice` has been called and `select-hid-device` has fired if a new device becomes available before the callback from `select-hid-device` is called. This event is intended for use when using a UI to ask users to pick a device so that the UI can be updated with the newly added device.

Event: 'hid-device-removed'

Returns:

- `event` Event
- `details` Object
  - `device` **HIDDevice**ⓘ
  - `frame` **WebFrameMain** | null - The frame initiating this event. May be `null` if accessed after the frame has either navigated or been destroyed.

Emitted after `navigator.hid.requestDevice` has been called and `select-hid-device` has fired if a device has been removed before the callback from `select-hid-device` is called. This event is intended for use when using a UI to ask users to pick a device so that the UI can be updated to remove the specified device.

Event: 'hid-device-revoked'

Returns:

- `event` Event
- `details` Object
  - `device` **HIDDevice**
  - `origin` string (optional) - The origin that the device has been revoked from.

Emitted after `HIDDevice.forget()` has been called. This event can be used to help maintain persistent storage of permissions when `setDevicePermissionHandler` is used.

Event: 'select-serial-port'

Returns:

- `event` Event
- `portList` **SerialPort[]**
- `webContents` **WebContents**
- `callback` Function
  - `portId` string

Emitted when a serial port needs to be selected when a call to `navigator.serial.requestPort` is made. `callback` should be called with `portId` to be selected, passing an empty string to `callback` will cancel the request. Additionally, permissioning on `navigator.serial` can be managed by using **ses.setPermissionCheckHandler(handler)** with the `serial` permission.

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

let win = null

app.whenReady().then(() => {
```

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

  win.webContents.session.setPermissionCheckHandler((webContents, permission,
requestingOrigin, details) => {
    if (permission === 'serial') {
      // Add logic here to determine if permission should be given to allow
serial selection
      return true
    }
    return false
  })

  // Optionally, retrieve previously persisted devices from a persistent store
  const grantedDevices = fetchGrantedDevices()

  win.webContents.session.setDevicePermissionHandler((details) => {
    if (new URL(details.origin).hostname === 'some-host' && details.deviceType
=== 'serial') {
      if (details.device.vendorId === 123 && details.device.productId === 345) {
        // Always allow this type of device (this allows skipping the call to
`navigator.serial.requestPort` first)
        return true
      }

      // Search through the list of devices that have previously been granted
permission
      return grantedDevices.some((grantedDevice) => {
        return grantedDevice.vendorId === details.device.vendorId &&
               grantedDevice.productId === details.device.productId &&
               grantedDevice.serialNumber && grantedDevice.serialNumber ===
details.device.serialNumber
      })
    }
    return false
  })

  win.webContents.session.on('select-serial-port', (event, portList, webContents,
callback) => {
    event.preventDefault()
    const selectedPort = portList.find((device) => {
      return device.vendorId === '9025' && device.productId === '67'
```

```
    })
    if (!selectedPort) {
      callback('')
    } else {
      callback(selectedPort.portId)
    }
  })
})
```

Event: 'serial-port-added'

Returns:

- `event` Event
- `port` **SerialPort**
- `webContents` **WebContents**

Emitted after `navigator.serial.requestPort` has been called and `select-serial-port` has fired if a new serial port becomes available before the callback from `select-serial-port` is called. This event is intended for use when using a UI to ask users to pick a port so that the UI can be updated with the newly added port.

Event: 'serial-port-removed'

Returns:

- `event` Event
- `port` **SerialPort**
- `webContents` **WebContents**

Emitted after `navigator.serial.requestPort` has been called and `select-serial-port` has fired if a serial port has been removed before the callback from `select-serial-port` is called. This event is intended for use when using a UI to ask users to pick a port so that the UI can be updated to remove the specified port.

Event: 'serial-port-revoked'

Returns:

- `event` Event
- `details` Object
  - `port` **SerialPort**
  - `frame` **WebFrameMain** | null - The frame initiating this event. May be `null` if accessed after the frame has either navigated or been destroyed.
  - `origin` string - The origin that the device has been revoked from.

Emitted after `SerialPort.forget()` has been called. This event can be used to help maintain persistent storage of permissions when `setDevicePermissionHandler` is used.

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

app.whenReady().then(() => {
  const win = new BrowserWindow({
    width: 800,
    height: 600
  })

  win.webContents.session.on('serial-port-revoked', (event, details) => {
    console.log(`Access revoked for serial device from origin ${details.origin}`)
  })
})
```

```
// Renderer Process

const portConnect = async () => {
  // Request a port.
  const port = await navigator.serial.requestPort()

  // Wait for the serial port to open.
  await port.open({ baudRate: 9600 })

  // ...later, revoke access to the serial port.
  await port.forget()
}
```

Event: 'select-usb-device'

Returns:

- `event` Event
- `details` Object
  - `deviceList` **USBDevice[]**
  - `frame` **WebFrameMain** | null - The frame initiating this event. May be `null` if accessed after the frame has either navigated or been destroyed.
- `callback` Function
  - `deviceId` string (optional)

Emitted when a USB device needs to be selected when a call to `navigator.usb.requestDevice` is made. `callback` should be called with `deviceId` to be selected; passing no arguments to `callback` will cancel the request. Additionally, permissioning on `navigator.usb` can be further managed by using **`ses.setPermissionCheckHandler(handler)`** and **`ses.setDevicePermissionHandler(handler)`**.

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

let win = null

app.whenReady().then(() => {
  win = new BrowserWindow()

  win.webContents.session.setPermissionCheckHandler((webContents, permission,
requestingOrigin, details) => {
    if (permission === 'usb') {
      // Add logic here to determine if permission should be given to allow USB
selection
      return true
    }
    return false
  })

  // Optionally, retrieve previously persisted devices from a persistent store
(fetchGrantedDevices needs to be implemented by developer to fetch persisted
permissions)
  const grantedDevices = fetchGrantedDevices()

  win.webContents.session.setDevicePermissionHandler((details) => {
```

```
    if (new URL(details.origin).hostname === 'some-host' && details.deviceType
=== 'usb') {
      if (details.device.vendorId === 123 && details.device.productId === 345) {
        // Always allow this type of device (this allows skipping the call to
`navigator.usb.requestDevice` first)
        return true
      }

      // Search through the list of devices that have previously been granted
permission
      return grantedDevices.some((grantedDevice) => {
        return grantedDevice.vendorId === details.device.vendorId &&
               grantedDevice.productId === details.device.productId &&
               grantedDevice.serialNumber && grantedDevice.serialNumber ===
details.device.serialNumber
      })
    }
    return false
  })

  win.webContents.session.on('select-usb-device', (event, details, callback) => {
    event.preventDefault()
    const selectedDevice = details.deviceList.find((device) => {
      return device.vendorId === 9025 && device.productId === 67
    })
    if (selectedDevice) {
      // Optionally, add this to the persisted devices (updateGrantedDevices
needs to be implemented by developer to persist permissions)
      grantedDevices.push(selectedDevice)
      updateGrantedDevices(grantedDevices)
    }
    callback(selectedDevice?.deviceId)
  })
})
```

Event: 'usb-device-added'

Returns:

- `event` Event
- `device` **USBDevice**
- `webContents` **WebContents**

Emitted after `navigator.usb.requestDevice` has been called and `select-usb-device` has fired if a new device becomes available before the callback from `select-usb-device` is called. This event is intended for use when using a UI to ask users to pick a device so that the UI can be updated with the newly added device.

Event: 'usb-device-removed'

Returns:

- `event` Event
- `device` **USBDevice**
- `webContents` **WebContents**

Emitted after `navigator.usb.requestDevice` has been called and `select-usb-device` has fired if a device has been removed before the callback from `select-usb-device` is called. This event is intended for use when using a UI to ask users to pick a device so that the UI can be updated to remove the specified device.

Event: 'usb-device-revoked'

Returns:

- `event` Event
- `details` Object
  - `device` **USBDevice**
  - `origin` string (optional) - The origin that the device has been revoked from.

Emitted after `USBDevice.forget()` has been called. This event can be used to help maintain persistent storage of permissions when `setDevicePermissionHandler` is used.

Instance Methods

The following methods are available on instances of `Session`:

`ses.getCacheSize()`

Returns `Promise<Integer>` - the session's current cache size, in bytes.

`ses.clearCache()`

Returns `Promise<void>` - resolves when the cache clear operation is complete.

Clears the session's HTTP cache.

`ses.clearStorageData([options])`

- `options` Object (optional)
  - `origin` string (optional) - Should follow `window.location.origin`'s representation `scheme://host:port`.
  - `storages` string[] (optional) - The types of storages to clear, can be `cookies`, `filesystem`, `indexdb`, `localstorage`, `shadercache`, `websql`, `serviceworkers`, `cachestorage`. If not specified, clear all storage types.
  - `quotas` string[] (optional) - The types of quotas to clear, can be `temporary`. If not specified, clear all quotas.

Returns `Promise<void>` - resolves when the storage data has been cleared.

`ses.flushStorageData()`

Writes any unwritten DOMStorage data to disk.

`ses.setProxy(config)`

- `config` **ProxyConfig**

Returns `Promise<void>` - Resolves when the proxy setting process is complete.

Sets the proxy settings.

You may need `ses.closeAllConnections` to close currently in flight connections to prevent pooled sockets using previous proxy from being reused by future requests.

`ses.resolveHost(host, [options])`

- `host` string - Hostname to resolve.
- `options` Object (optional)

- `queryType` string (optional) - Requested DNS query type. If unspecified, resolver will pick A or AAAA (or both) based on IPv4/IPv6 settings:
  - `A` - Fetch only A records
  - `AAAA` - Fetch only AAAA records.
- `source` string (optional) - The source to use for resolved addresses. Default allows the resolver to pick an appropriate source. Only affects use of big external sources (e.g. calling the system for resolution or using DNS). Even if a source is specified, results can still come from cache, resolving "localhost" or IP literals, etc. One of the following values:
  - `any` (default) - Resolver will pick an appropriate source. Results could come from DNS, MulticastDNS, HOSTS file, etc
  - `system` - Results will only be retrieved from the system or OS, e.g. via the `getaddrinfo()` system call
  - `dns` - Results will only come from DNS queries
  - `mdns` - Results will only come from Multicast DNS queries
  - `localOnly` - No external sources will be used. Results will only come from fast local sources that are available no matter the source setting, e.g. cache, hosts file, IP literal resolution, etc.
- `cacheUsage` string (optional) - Indicates what DNS cache entries, if any, can be used to provide a response. One of the following values:
  - `allowed` (default) - Results may come from the host cache if non-stale
  - `staleAllowed` - Results may come from the host cache even if stale (by expiration or network changes)
  - `disallowed` - Results will not come from the host cache.
- `secureDnsPolicy` string (optional) - Controls the resolver's Secure DNS behavior for this request. One of the following values:
  - `allow` (default)
  - `disable`

Returns **Promise<ResolvedHost>** - Resolves with the resolved IP addresses for the `host`.

`ses.resolveProxy(url)`

- `url` URL

Returns `Promise<string>` - Resolves with the proxy information for `url`.

`ses.forceReloadProxyConfig()`

Returns `Promise<void>` - Resolves when the all internal states of proxy service is reset and the latest proxy configuration is reapplied if it's already available. The pac script will be fetched from `pacScript` again if the proxy mode is `pac_script`.

`ses.setDownloadPath(path)`

- `path` string - The download location.

Sets download saving directory. By default, the download directory will be the `Downloads` under the respective app folder.

`ses.enableNetworkEmulation(options)`

- `options` Object
  - `offline` boolean (optional) - Whether to emulate network outage. Defaults to false.
  - `latency` Double (optional) - RTT in ms. Defaults to 0 which will disable latency throttling.
  - `downloadThroughput` Double (optional) - Download rate in Bps. Defaults to 0 which will disable download throttling.
  - `uploadThroughput` Double (optional) - Upload rate in Bps. Defaults to 0 which will disable upload throttling.

Emulates network with the given configuration for the `session`.

```
const win = new BrowserWindow()

// To emulate a GPRS connection with 50kbps throughput and 500 ms latency.
win.webContents.session.enableNetworkEmulation({
  latency: 500,
  downloadThroughput: 6400,
  uploadThroughput: 6400
})

// To emulate a network outage.
win.webContents.session.enableNetworkEmulation({ offline: true })
```

`ses.preconnect(options)`

- `options` Object
  - `url` string - URL for preconnect. Only the origin is relevant for opening the socket.
  - `numSockets` number (optional) - number of sockets to preconnect. Must be between 1 and 6. Defaults to 1.

Preconnects the given number of sockets to an origin.

`ses.closeAllConnections()`

Returns `Promise<void>` - Resolves when all connections are closed.

> **NOTE**
>
> It will terminate / fail all requests currently in flight.

`ses.fetch(input[, init])`

- `input` string | **GlobalRequest**
- `init` **RequestInit** & { bypassCustomProtocolHandlers?: boolean } (optional)

Returns `Promise<GlobalResponse>` - see **Response**.

Sends a request, similarly to how `fetch()` works in the renderer, using Chrome's network stack. This differs from Node's `fetch()`, which uses Node.js's HTTP stack.

Example:

```
async function example () {
  const response = await net.fetch('https://my.app')
  if (response.ok) {
    const body = await response.json()
    // ... use the result.
  }
}
```

See also **`net.fetch()`**, a convenience method which issues requests from the **default session**.

See the MDN documentation for **`fetch()`** for more details.

Limitations:

- `net.fetch()` does not support the `data:` or `blob:` schemes.
- The value of the `integrity` option is ignored.
- The `.type` and `.url` values of the returned `Response` object are incorrect.

By default, requests made with `net.fetch` can be made to **custom protocols** as well as `file:`, and will trigger **webRequest** handlers if present. When the non-standard `bypassCustomProtocolHandlers` option is set in RequestInit, custom protocol handlers will not be called for this request. This allows forwarding an intercepted request to the built-in handler. **webRequest** handlers will still be triggered when bypassing custom protocols.

```
protocol.handle('https', (req) => {
  if (req.url === 'https://my-app.com') {
    return new Response('<body>my app</body>')
  } else {
    return net.fetch(req, { bypassCustomProtocolHandlers: true })
  }
})
```

`ses.disableNetworkEmulation()`

Disables any network emulation already active for the `session`. Resets to the original network configuration.

`ses.setCertificateVerifyProc(proc)`

- `proc` Function | null
  - `request` Object
    - `hostname` string
    - `certificate` **Certificate**
    - `validatedCertificate` **Certificate**
    - `isIssuedByKnownRoot` boolean - `true` if Chromium recognises the root CA as a standard root. If it isn't then it's probably the case that this certificate was generated by a MITM proxy whose root has been installed locally (for example, by a corporate proxy). This should not be trusted if the `verificationResult` is not `OK`.

        - `verificationResult` string - `OK` if the certificate is trusted, otherwise an error like `CERT_REVOKED`.
        - `errorCode` Integer - Error code.
    - `callback` Function
        - `verificationResult` Integer - Value can be one of certificate error codes from **here**. Apart from the certificate error codes, the following special codes can be used.
            - `0` - Indicates success and disables Certificate Transparency verification.
            - `-2` - Indicates failure.
            - `-3` - Uses the verification result from chromium.

Sets the certificate verify proc for `session`, the `proc` will be called with `proc(request, callback)` whenever a server certificate verification is requested. Calling `callback(0)` accepts the certificate, calling `callback(-2)` rejects it.

Calling `setCertificateVerifyProc(null)` will revert back to default certificate verify proc.

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

win.webContents.session.setCertificateVerifyProc((request, callback) => {
  const { hostname } = request
  if (hostname === 'github.com') {
    callback(0)
  } else {
    callback(-2)
  }
})
```

> **NOTE:** The result of this procedure is cached by the network service.

`ses.setPermissionRequestHandler(handler)`

- `handler` Function | null
    - `webContents` **WebContents** - WebContents requesting the permission. Please note that if the request comes from a subframe you should use `requestingUrl` to check the request origin.
    - `permission` string - The type of requested permission.

- `clipboard-read` - Request access to read from the clipboard.
- `clipboard-sanitized-write` - Request access to write to the clipboard.
- `display-capture` - Request access to capture the screen via the **Screen Capture API**.
- `fullscreen` - Request control of the app's fullscreen state via the **Fullscreen API**.
- `geolocation` - Request access to the user's location via the **Geolocation API**
- `idle-detection` - Request access to the user's idle state via the **IdleDetector API**.
- `media` - Request access to media devices such as camera, microphone and speakers.
- `mediaKeySystem` - Request access to DRM protected content.
- `midi` - Request MIDI access in the **Web MIDI API**.
- `midiSysex` - Request the use of system exclusive messages in the **Web MIDI API**.
- `notifications` - Request notification creation and the ability to display them in the user's system tray using the **Notifications API**
- `pointerLock` - Request to directly interpret mouse movements as an input method via the **Pointer Lock API**. These requests always appear to originate from the main frame.
- `keyboardLock` - Request capture of keypresses for any or all of the keys on the physical keyboard via the **Keyboard Lock API**. These requests always appear to originate from the main frame.
- `openExternal` - Request to open links in external applications.
- `speaker-selection` - Request to enumerate and select audio output devices via the **speaker-selection permissions policy**.
- `storage-access` - Allows content loaded in a third-party context to request access to third-party cookies using the **Storage Access API**.
- `top-level-storage-access` - Allow top-level sites to request third-party cookie access on behalf of embedded content originating from another site in the same related website set using the **Storage Access API**.
- `window-management` - Request access to enumerate screens using the `getScreenDetails` API.
- `unknown` - An unrecognized permission request.
- `fileSystem` - Request access to read, write, and file management capabilities using the **File System API**.

- `callback` Function
  - `permissionGranted` boolean - Allow or deny the permission.

- `details` **PermissionRequest** ⓘ | **FilesystemPermissionRequest** ⓘ | **MediaAccessPermissionRequest** ⓘ | **OpenExternalPermissionRequest** ⓘ - Additional information about the permission being requested.

Sets the handler which can be used to respond to permission requests for the `session`. Calling `callback(true)` will allow the permission and `callback(false)` will reject it. To clear the handler, call `setPermissionRequestHandler(null)`. Please note that you must also implement `setPermissionCheckHandler` to get complete permission handling. Most web APIs do a permission check and then make a permission request if the check is denied.

```
const { session } = require('electron')
session.fromPartition('some-partition').setPermissionRequestHandler((webContents,
permission, callback) => {
  if (webContents.getURL() === 'some-host' && permission === 'notifications') {
    return callback(false) // denied.
  }

  callback(true)
})
```

`ses.setPermissionCheckHandler(handler)`

- `handler` Function<boolean> | null
  - `webContents` (**WebContents** | null) - WebContents checking the permission. Please note that if the request comes from a subframe you should use `requestingUrl` to check the request origin. All cross origin sub frames making permission checks will pass a `null` webContents to this handler, while certain other permission checks such as `notifications` checks will always pass `null`. You should use `embeddingOrigin` and `requestingOrigin` to determine what origin the owning frame and the requesting frame are on respectively.
  - `permission` string - Type of permission check.
    - `clipboard-read` - Request access to read from the clipboard.
    - `clipboard-sanitized-write` - Request access to write to the clipboard.
    - `geolocation` - Access the user's geolocation data via the **Geolocation API**
    - `fullscreen` - Control of the app's fullscreen state via the **Fullscreen API**.
    - `hid` - Access the HID protocol to manipulate HID devices via the **WebHID API**.
    - `idle-detection` - Access the user's idle state via the **IdleDetector API**.

- `media` - Access to media devices such as camera, microphone and speakers.
- `mediaKeySystem` - Access to DRM protected content.
- `midi` - Enable MIDI access in the **Web MIDI API**.
- `midiSysex` - Use system exclusive messages in the **Web MIDI API**.
- `notifications` - Configure and display desktop notifications to the user with the **Notifications API**.
- `openExternal` - Open links in external applications.
- `pointerLock` - Directly interpret mouse movements as an input method via the **Pointer Lock API**. These requests always appear to originate from the main frame.
- `serial` - Read from and write to serial devices with the **Web Serial API**.
- `storage-access` - Allows content loaded in a third-party context to request access to third-party cookies using the **Storage Access API**.
- `top-level-storage-access` - Allow top-level sites to request third-party cookie access on behalf of embedded content originating from another site in the same related website set using the **Storage Access API**.
- `usb` - Expose non-standard Universal Serial Bus (USB) compatible devices services to the web with the **WebUSB API**.
- `deprecated-sync-clipboard-read` *Deprecated* - Request access to run `document.execCommand("paste")`

- `requestingOrigin` string - The origin URL of the permission check
- `details` Object - Some properties are only available on certain permission types.
  - `embeddingOrigin` string (optional) - The origin of the frame embedding the frame that made the permission check. Only set for cross-origin sub frames making permission checks.
  - `securityOrigin` string (optional) - The security origin of the `media` check.
  - `mediaType` string (optional) - The type of media access being requested, can be `video`, `audio` or `unknown`
  - `requestingUrl` string (optional) - The last URL the requesting frame loaded. This is not provided for cross-origin sub frames making permission checks.
  - `isMainFrame` boolean - Whether the frame making the request is the main frame

Sets the handler which can be used to respond to permission checks for the `session`. Returning `true` will allow the permission and `false` will reject it. Please note that you must also implement

`setPermissionRequestHandler` to get complete permission handling. Most web APIs do a permission check and then make a permission request if the check is denied. To clear the handler, call `setPermissionCheckHandler(null)`.

```
const { session } = require('electron')
const url = require('url')
session.fromPartition('some-partition').setPermissionCheckHandler((webContents,
permission, requestingOrigin) => {
  if (new URL(requestingOrigin).hostname === 'some-host' && permission ===
'notifications') {
    return true // granted
  }

  return false // denied
})
```

`ses.setDisplayMediaRequestHandler(handler[, opts])`

- `handler` Function | null
  - `request` Object
    - `frame` **WebFrameMain** | null - Frame that is requesting access to media. May be `null` if accessed after the frame has either navigated or been destroyed.
    - `securityOrigin` String - Origin of the page making the request.
    - `videoRequested` Boolean - true if the web content requested a video stream.
    - `audioRequested` Boolean - true if the web content requested an audio stream.
    - `userGesture` Boolean - Whether a user gesture was active when this request was triggered.
  - `callback` Function
    - `streams` Object
      - `video` Object | **WebFrameMain** (optional)
        - `id` String - The id of the stream being granted. This will usually come from a **DesktopCapturerSource** object.
        - `name` String - The name of the stream being granted. This will usually come from a **DesktopCapturerSource** object.
      - `audio` String | **WebFrameMain** (optional) - If a string is specified, can be `loopback` or `loopbackWithMute`. Specifying a loopback device will capture system

audio, and is currently only supported on Windows. If a WebFrameMain is specified, will capture audio from that frame.

    - `enableLocalEcho` Boolean (optional) - If `audio` is a **WebFrameMain** and this is set to `true`, then local playback of audio will not be muted (e.g. using `MediaRecorder` to record `WebFrameMain` with this flag set to `true` will allow audio to pass through to the speakers while recording). Default is `false`.
- `opts` Object (optional) *macOS* *Experimental*
  - `useSystemPicker` Boolean - true if the available native system picker should be used. Default is `false`. *macOS* *Experimental*

This handler will be called when web content requests access to display media via the `navigator.mediaDevices.getDisplayMedia` API. Use the **desktopCapturer** API to choose which stream(s) to grant access to.

`useSystemPicker` allows an application to use the system picker instead of providing a specific video source from `getSources`. This option is experimental, and currently available for MacOS 15+ only. If the system picker is available and `useSystemPicker` is set to `true`, the handler will not be invoked.

```
const { session, desktopCapturer } = require('electron')

session.defaultSession.setDisplayMediaRequestHandler((request, callback) => {
  desktopCapturer.getSources({ types: ['screen'] }).then((sources) => {
    // Grant access to the first screen found.
    callback({ video: sources[0] })
  })
  // Use the system picker if available.
  // Note: this is currently experimental. If the system picker
  // is available, it will be used and the media request handler
  // will not be invoked.
}, { useSystemPicker: true })
```

Passing a **WebFrameMain** object as a video or audio stream will capture the video or audio stream from that frame.

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

```
session.defaultSession.setDisplayMediaRequestHandler((request, callback) => {
  // Allow the tab to capture itself.
  callback({ video: request.frame })
})
```

Passing `null` instead of a function resets the handler to its default state.

`ses.setDevicePermissionHandler(handler)`

- `handler` Function<boolean> | null
  - `details` Object
    - `deviceType` string - The type of device that permission is being requested on, can be `hid`, `serial`, or `usb`.
    - `origin` string - The origin URL of the device permission check.
    - `device` **HIDDevice** ⓘ | **SerialPort** ⓘ | **USBDevice** ⓘ - the device that permission is being requested for.

Sets the handler which can be used to respond to device permission checks for the `session`. Returning `true` will allow the device to be permitted and `false` will reject it. To clear the handler, call `setDevicePermissionHandler(null)`. This handler can be used to provide default permissioning to devices without first calling for permission to devices (eg via `navigator.hid.requestDevice`). If this handler is not defined, the default device permissions as granted through device selection (eg via `navigator.hid.requestDevice`) will be used. Additionally, the default behavior of Electron is to store granted device permission in memory. If longer term storage is needed, a developer can store granted device permissions (eg when handling the `select-hid-device` event) and then read from that storage with `setDevicePermissionHandler`.

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

let win = null

app.whenReady().then(() => {
  win = new BrowserWindow()

  win.webContents.session.setPermissionCheckHandler((webContents, permission,
requestingOrigin, details) => {
    if (permission === 'hid') {
      // Add logic here to determine if permission should be given to allow HID
```

```
selection
      return true
    } else if (permission === 'serial') {
      // Add logic here to determine if permission should be given to allow
serial port selection
    } else if (permission === 'usb') {
      // Add logic here to determine if permission should be given to allow USB
device selection
    }
    return false
  })

  // Optionally, retrieve previously persisted devices from a persistent store
  const grantedDevices = fetchGrantedDevices()

  win.webContents.session.setDevicePermissionHandler((details) => {
    if (new URL(details.origin).hostname === 'some-host' && details.deviceType
=== 'hid') {
      if (details.device.vendorId === 123 && details.device.productId === 345) {
        // Always allow this type of device (this allows skipping the call to
`navigator.hid.requestDevice` first)
        return true
      }

      // Search through the list of devices that have previously been granted
permission
      return grantedDevices.some((grantedDevice) => {
        return grantedDevice.vendorId === details.device.vendorId &&
               grantedDevice.productId === details.device.productId &&
               grantedDevice.serialNumber && grantedDevice.serialNumber ===
details.device.serialNumber
      })
    } else if (details.deviceType === 'serial') {
      if (details.device.vendorId === 123 && details.device.productId === 345) {
        // Always allow this type of device (this allows skipping the call to
`navigator.hid.requestDevice` first)
        return true
      }
    }
    return false
  })

  win.webContents.session.on('select-hid-device', (event, details, callback) => {
    event.preventDefault()
```

```
    const selectedDevice = details.deviceList.find((device) => {
      return device.vendorId === 9025 && device.productId === 67
    })
    callback(selectedDevice?.deviceId)
  })
})
```

`ses.setUSBProtectedClassesHandler(handler)`

- `handler` Function<string[]> | null
  - `details` Object
    - `protectedClasses` string[] - The current list of protected USB classes. Possible class values include:
      - `audio`
      - `audio-video`
      - `hid`
      - `mass-storage`
      - `smart-card`
      - `video`
      - `wireless`

Sets the handler which can be used to override which **USB classes are protected**. The return value for the handler is a string array of USB classes which should be considered protected (eg not available in the renderer). Valid values for the array are:

- `audio`
- `audio-video`
- `hid`
- `mass-storage`
- `smart-card`
- `video`
- `wireless`

Returning an empty string array from the handler will allow all USB classes; returning the passed in array will maintain the default list of protected USB classes (this is also the default behavior if a

handler is not defined). To clear the handler, call `setUSBProtectedClassesHandler(null)`.

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

let win = null

app.whenReady().then(() => {
  win = new BrowserWindow()

  win.webContents.session.setUSBProtectedClassesHandler((details) => {
    // Allow all classes:
    // return []
    // Keep the current set of protected classes:
    // return details.protectedClasses
    // Selectively remove classes:
    return details.protectedClasses.filter((usbClass) => {
      // Exclude classes except for audio classes
      return usbClass.indexOf('audio') === -1
    })
  })
})
```

`ses.setBluetoothPairingHandler(handler)` *Windows* *Linux*

- `handler` Function | null
  - `details` Object
    - `deviceId` string
    - `pairingKind` string - The type of pairing prompt being requested. One of the following values:
      - `confirm` This prompt is requesting confirmation that the Bluetooth device should be paired.
      - `confirmPin` This prompt is requesting confirmation that the provided PIN matches the pin displayed on the device.
      - `providePin` This prompt is requesting that a pin be provided for the device.
    - `frame` **WebFrameMain** | null - The frame initiating this handler. May be `null` if accessed after the frame has either navigated or been destroyed.
    - `pin` string (optional) - The pin value to verify if `pairingKind` is `confirmPin`.
  - `callback` Function

- `response` Object
  - `confirmed` boolean - `false` should be passed in if the dialog is canceled. If the `pairingKind` is `confirm` or `confirmPin`, this value should indicate if the pairing is confirmed. If the `pairingKind` is `providePin` the value should be `true` when a value is provided.
  - `pin` string | null (optional) - When the `pairingKind` is `providePin` this value should be the required pin for the Bluetooth device.

Sets a handler to respond to Bluetooth pairing requests. This handler allows developers to handle devices that require additional validation before pairing. When a handler is not defined, any pairing on Linux or Windows that requires additional validation will be automatically cancelled. macOS does not require a handler because macOS handles the pairing automatically. To clear the handler, call `setBluetoothPairingHandler(null)`.

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

function createWindow () {
  let bluetoothPinCallback = null

  const mainWindow = new BrowserWindow({
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })

  mainWindow.webContents.session.setBluetoothPairingHandler((details, callback)
=> {
    bluetoothPinCallback = callback
    // Send a IPC message to the renderer to prompt the user to confirm the
pairing.
    // Note that this will require logic in the renderer to handle this message
and
    // display a prompt to the user.
    mainWindow.webContents.send('bluetooth-pairing-request', details)
  })

  // Listen for an IPC message from the renderer to get the response for the
Bluetooth pairing.
  mainWindow.webContents.ipc.on('bluetooth-pairing-response', (event, response)
```

```
=> {
    bluetoothPinCallback(response)
  })
}

app.whenReady().then(() => {
  createWindow()
})
```

`ses.clearHostResolverCache()`

Returns `Promise<void>` - Resolves when the operation is complete.

Clears the host resolver cache.

`ses.allowNTLMCredentialsForDomains(domains)`

- `domains` string - A comma-separated list of servers for which integrated authentication is enabled.

Dynamically sets whether to always send credentials for HTTP NTLM or Negotiate authentication.

```
const { session } = require('electron')
// consider any url ending with `example.com`, `foobar.com`, `baz`
// for integrated authentication.
session.defaultSession.allowNTLMCredentialsForDomains('*example.com, *foobar.com, *baz')

// consider all urls for integrated authentication.
session.defaultSession.allowNTLMCredentialsForDomains('*')
```

`ses.setUserAgent(userAgent[, acceptLanguages])`

- `userAgent` string
- `acceptLanguages` string (optional)

Overrides the `userAgent` and `acceptLanguages` for this session.

The `acceptLanguages` must a comma separated ordered list of language codes, for example `"en-US,fr,de,ko,zh-CN,ja"`.

This doesn't affect existing `WebContents`, and each `WebContents` can use `webContents.setUserAgent` to override the session-wide user agent.

`ses.isPersistent()`

Returns `boolean` - Whether or not this session is a persistent one. The default `webContents` session of a `BrowserWindow` is persistent. When creating a session from a partition, session prefixed with `persist:` will be persistent, while others will be temporary.

`ses.getUserAgent()`

Returns `string` - The user agent for this session.

`ses.setSSLConfig(config)`

- `config` Object
  - `minVersion` string (optional) - Can be `tls1`, `tls1.1`, `tls1.2` or `tls1.3`. The minimum SSL version to allow when connecting to remote servers. Defaults to `tls1`.
  - `maxVersion` string (optional) - Can be `tls1.2` or `tls1.3`. The maximum SSL version to allow when connecting to remote servers. Defaults to `tls1.3`.
  - `disabledCipherSuites` Integer[] (optional) - List of cipher suites which should be explicitly prevented from being used in addition to those disabled by the net built-in policy. Supported literal forms: 0xAABB, where AA is `cipher_suite[0]` and BB is `cipher_suite[1]`, as defined in RFC 2246, Section 7.4.1.2. Unrecognized but parsable cipher suites in this form will not return an error. Ex: To disable TLS_RSA_WITH_RC4_128_MD5, specify 0x0004, while to disable TLS_ECDH_ECDSA_WITH_RC4_128_SHA, specify 0xC002. Note that TLSv1.3 ciphers cannot be disabled using this mechanism.

Sets the SSL configuration for the session. All subsequent network requests will use the new configuration. Existing network connections (such as WebSocket connections) will not be terminated, but old sockets in the pool will not be reused for new connections.

`ses.getBlobData(identifier)`

- `identifier` string - Valid UUID.

Returns `Promise<Buffer>` - resolves with blob data.

`ses.downloadURL(url[, options])`

- `url` string
- `options` Object (optional)
  - `headers` Record<string, string> (optional) - HTTP request headers.

Initiates a download of the resource at `url`. The API will generate a **DownloadItem** that can be accessed with the **will-download** event.

> **NOTE**
>
> This does not perform any security checks that relate to a page's origin, unlike `webContents.downloadURL`.

`ses.createInterruptedDownload(options)`

- `options` Object
  - `path` string - Absolute path of the download.
  - `urlChain` string[] - Complete URL chain for the download.
  - `mimeType` string (optional)
  - `offset` Integer - Start range for the download.
  - `length` Integer - Total length of the download.
  - `lastModified` string (optional) - Last-Modified header value.
  - `eTag` string (optional) - ETag header value.
  - `startTime` Double (optional) - Time when download was started in number of seconds since UNIX epoch.

Allows resuming `cancelled` or `interrupted` downloads from previous `Session`. The API will generate a **DownloadItem** that can be accessed with the **will-download** event. The **DownloadItem** will not have any `WebContents` associated with it and the initial state will be `interrupted`. The download will start only when the `resume` API is called on the **DownloadItem**.

`ses.clearAuthCache()`

Returns `Promise<void>` - resolves when the session's HTTP authentication cache has been cleared.

`ses.setPreloads(preloads)` *Deprecated*

- `preloads` string[] - An array of absolute path to preload scripts

Adds scripts that will be executed on ALL web contents that are associated with this session just before normal `preload` scripts run.

Deprecated: Use the new `ses.registerPreloadScript` API.

`ses.getPreloads()` *Deprecated*

Returns `string[]` an array of paths to preload scripts that have been registered.

Deprecated: Use the new `ses.getPreloadScripts` API. This will only return preload script paths for `frame` context types.

`ses.registerPreloadScript(script)`

- `script` **PreloadScriptRegistration** ⓘ - Preload script

Registers preload script that will be executed in its associated context type in this session. For `frame` contexts, this will run prior to any preload defined in the web preferences of a WebContents.

Returns `string` - The ID of the registered preload script.

`ses.unregisterPreloadScript(id)`

- `id` string - Preload script ID

Unregisters script.

`ses.getPreloadScripts()`

Returns **PreloadScript[]** ⓘ: An array of paths to preload scripts that have been registered.

`ses.setCodeCachePath(path)`

- `path` String - Absolute path to store the v8 generated JS code cache from the renderer.

Sets the directory to store the generated JS **code cache** for this session. The directory is not required to be created by the user before this call, the runtime will create if it does not exist otherwise will use the existing directory. If directory cannot be created, then code cache will not be used and all operations related to code cache will fail silently inside the runtime. By default, the directory will be `Code Cache` under the respective user data folder.

Note that by default code cache is only enabled for http(s) URLs, to enable code cache for custom protocols, `codeCache: true` and `standard: true` must be specified when registering the protocol.

`ses.clearCodeCaches(options)`

- `options` Object
  - `urls` String[] (optional) - An array of url corresponding to the resource whose generated code cache needs to be removed. If the list is empty then all entries in the cache directory will be removed.

Returns `Promise<void>` - resolves when the code cache clear operation is complete.

`ses.getSharedDictionaryUsageInfo()`

Returns `Promise<SharedDictionaryUsageInfo[]>` - an array of shared dictionary information entries in Chromium's networking service's storage.

Shared dictionaries are used to power advanced compression of data sent over the wire, specifically with Brotli and ZStandard. You don't need to call any of the shared dictionary APIs in Electron to make use of this advanced web feature, but if you do, they allow deeper control and inspection of the shared dictionaries used during decompression.

To get detailed information about a specific shared dictionary entry, call `getSharedDictionaryInfo(options)`.

`ses.getSharedDictionaryInfo(options)`

- `options` Object
  - `frameOrigin` string - The origin of the frame where the request originates. It's specific to the individual frame making the request and is defined by its scheme, host, and port. In

practice, will look like a URL.
    - `topFrameSite` string - The site of the top-level browsing context (the main frame or tab that contains the request). It's less granular than `frameOrigin` and focuses on the broader "site" scope. In practice, will look like a URL.

Returns `Promise<SharedDictionaryInfo[]>` - an array of shared dictionary information entries in Chromium's networking service's storage.

To get information about all present shared dictionaries, call `getSharedDictionaryUsageInfo()`.

`ses.clearSharedDictionaryCache()`

Returns `Promise<void>` - resolves when the dictionary cache has been cleared, both in memory and on disk.

`ses.clearSharedDictionaryCacheForIsolationKey(options)`

- `options` Object
    - `frameOrigin` string - The origin of the frame where the request originates. It's specific to the individual frame making the request and is defined by its scheme, host, and port. In practice, will look like a URL.
    - `topFrameSite` string - The site of the top-level browsing context (the main frame or tab that contains the request). It's less granular than `frameOrigin` and focuses on the broader "site" scope. In practice, will look like a URL.

Returns `Promise<void>` - resolves when the dictionary cache has been cleared for the specified isolation key, both in memory and on disk.

`ses.setSpellCheckerEnabled(enable)`

- `enable` boolean

Sets whether to enable the builtin spell checker.

`ses.isSpellCheckerEnabled()`

Returns `boolean` - Whether the builtin spell checker is enabled.

`ses.setSpellCheckerLanguages(languages)`

- `languages` string[] - An array of language codes to enable the spellchecker for.

The built in spellchecker does not automatically detect what language a user is typing in. In order for the spell checker to correctly check their words you must call this API with an array of language codes. You can get the list of supported language codes with the `ses.availableSpellCheckerLanguages` property.

> **NOTE**
>
> On macOS, the OS spellchecker is used and will detect your language automatically. This API is a no-op on macOS.

`ses.getSpellCheckerLanguages()`

Returns `string[]` - An array of language codes the spellchecker is enabled for. If this list is empty the spellchecker will fallback to using `en-US`. By default on launch if this setting is an empty list Electron will try to populate this setting with the current OS locale. This setting is persisted across restarts.

> **NOTE**
>
> On macOS, the OS spellchecker is used and has its own list of languages. On macOS, this API will return whichever languages have been configured by the OS.

`ses.setSpellCheckerDictionaryDownloadURL(url)`

- `url` string - A base URL for Electron to download hunspell dictionaries from.

By default Electron will download hunspell dictionaries from the Chromium CDN. If you want to override this behavior you can use this API to point the dictionary downloader at your own hosted version of the hunspell dictionaries. We publish a `hunspell_dictionaries.zip` file with each release which contains the files you need to host here.

The file server must be **case insensitive**. If you cannot do this, you must upload each file twice: once with the case it has in the ZIP file and once with the filename as all lowercase.

If the files present in `hunspell_dictionaries.zip` are available at `https://example.com/dictionaries/language-code.bdic` then you should call this api with `ses.setSpellCheckerDictionaryDownloadURL('https://example.com/dictionaries/')`. Please note the trailing slash. The URL to the dictionaries is formed as `${url}${filename}`.

> **NOTE**
>
> On macOS, the OS spellchecker is used and therefore we do not download any dictionary files. This API is a no-op on macOS.

`ses.listWordsInSpellCheckerDictionary()`

Returns `Promise<string[]>` - An array of all words in app's custom dictionary. Resolves when the full dictionary is loaded from disk.

`ses.addWordToSpellCheckerDictionary(word)`

- `word` string - The word you want to add to the dictionary

Returns `boolean` - Whether the word was successfully written to the custom dictionary. This API will not work on non-persistent (in-memory) sessions.

> **NOTE**
>
> On macOS and Windows, this word will be written to the OS custom dictionary as well.

`ses.removeWordFromSpellCheckerDictionary(word)`

- `word` string - The word you want to remove from the dictionary

Returns `boolean` - Whether the word was successfully removed from the custom dictionary. This API will not work on non-persistent (in-memory) sessions.

> **NOTE**
>
> On macOS and Windows, this word will be removed from the OS custom dictionary as well.

`ses.loadExtension(path[, options])` *Deprecated*

- `path` string - Path to a directory containing an unpacked Chrome extension
- `options` Object (optional)
  - `allowFileAccess` boolean - Whether to allow the extension to read local files over `file://` protocol and inject content scripts into `file://` pages. This is required e.g. for loading devtools extensions on `file://` URLs. Defaults to false.

Returns `Promise<Extension>` - resolves when the extension is loaded.

This method will raise an exception if the extension could not be loaded. If there are warnings when installing the extension (e.g. if the extension requests an API that Electron does not support) then they will be logged to the console.

Note that Electron does not support the full range of Chrome extensions APIs. See **Supported Extensions APIs** for more details on what is supported.

Note that in previous versions of Electron, extensions that were loaded would be remembered for future runs of the application. This is no longer the case: `loadExtension` must be called on every boot of your app if you want the extension to be loaded.

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

app.whenReady().then(async () => {
  await session.defaultSession.loadExtension(
    path.join(__dirname, 'react-devtools'),
    // allowFileAccess is required to load the devtools extension on file://
URLs.
    { allowFileAccess: true }
  )
  // Note that in order to use the React DevTools extension, you'll need to
  // download and unzip a copy of the extension.
})
```

This API does not support loading packed (.crx) extensions.

> **NOTE**

> This API cannot be called before the `ready` event of the `app` module is emitted.

> **NOTE**
>
> Loading extensions into in-memory (non-persistent) sessions is not supported and will throw an error.

Deprecated: Use the new `ses.extensions.loadExtension` API.

`ses.removeExtension(extensionId)` *Deprecated*

- `extensionId` string - ID of extension to remove

Unloads an extension.

> **NOTE**
>
> This API cannot be called before the `ready` event of the `app` module is emitted.

Deprecated: Use the new `ses.extensions.removeExtension` API.

`ses.getExtension(extensionId)` *Deprecated*

- `extensionId` string - ID of extension to query

Returns `Extension | null` - The loaded extension with the given ID.

> **NOTE**
>
> This API cannot be called before the `ready` event of the `app` module is emitted.

Deprecated: Use the new `ses.extensions.getExtension` API.

`ses.getAllExtensions()` *Deprecated*

Returns `Extension[]` - A list of all loaded extensions.

> **NOTE**
>
> This API cannot be called before the `ready` event of the `app` module is emitted.

Deprecated: Use the new `ses.extensions.getAllExtensions` API.

`ses.getStoragePath()`

Returns `string | null` - The absolute file system path where data for this session is persisted on disk. For in memory sessions this returns `null`.

`ses.clearData([options])`

- `options` Object (optional)
  - `dataTypes` String[] (optional) - The types of data to clear. By default, this will clear all types of data. This can potentially include data types not explicitly listed here. (See Chromium's `BrowsingDataRemover` for the full list.)
    - `backgroundFetch` - Background Fetch
    - `cache` - Cache (includes `cachestorage` and `shadercache`)
    - `cookies` - Cookies
    - `downloads` - Downloads
    - `fileSystems` - File Systems
    - `indexedDB` - IndexedDB
    - `localStorage` - Local Storage
    - `serviceWorkers` - Service Workers
    - `webSQL` - WebSQL
  - `origins` String[] (optional) - Clear data for only these origins. Cannot be used with `excludeOrigins`.
  - `excludeOrigins` String[] (optional) - Clear data for all origins except these ones. Cannot be used with `origins`.
  - `avoidClosingConnections` boolean (optional) - Skips deleting cookies that would close current network connections. (Default: `false`)
  - `originMatchingMode` String (optional) - The behavior for matching data to origins.
    - `third-parties-included` (default) - Storage is matched on origin in first-party contexts and top-level-site in third-party contexts.

- `origin-in-all-contexts` - Storage is matched on origin only in all contexts.

Returns `Promise<void>` - resolves when all data has been cleared.

Clears various different types of data.

This method clears more types of data and is more thorough than the `clearStorageData` method.

> **NOTE**
>
> Cookies are stored at a broader scope than origins. When removing cookies and filtering by `origins` (or `excludeOrigins`), the cookies will be removed at the **registrable domain** level. For example, clearing cookies for the origin `https://really.specific.origin.example.com/` will end up clearing all cookies for `example.com`. Clearing cookies for the origin `https://my.website.example.co.uk/` will end up clearing all cookies for `example.co.uk`.

> **NOTE**
>
> Clearing cache data will also clear the shared dictionary cache. This means that any dictionaries used for compression may be reloaded after clearing the cache. If you wish to clear the shared dictionary cache but leave other cached data intact, you may want to use the `clearSharedDictionaryCache` method.

For more information, refer to Chromium's `BrowsingDataRemover` **interface**.

Instance Properties

The following properties are available on instances of `Session`:

`ses.availableSpellCheckerLanguages` *Readonly*

A `string[]` array which consists of all the known available spell checker languages. Providing a language code to the `setSpellCheckerLanguages` API that isn't in this array will result in an error.

`ses.spellCheckerEnabled`

A `boolean` indicating whether builtin spell checker is enabled.

`ses.storagePath` *Readonly*

A `string | null` indicating the absolute file system path where data for this session is persisted on disk. For in memory sessions this returns `null`.

`ses.cookies` *Readonly*

A **`Cookies`** object for this session.

`ses.extensions` *Readonly*

A **`Extensions`** object for this session.

`ses.serviceWorkers` *Readonly*

A **`ServiceWorkers`** object for this session.

`ses.webRequest` *Readonly*

A **`WebRequest`** object for this session.

`ses.protocol` *Readonly*

A **`Protocol`** object for this session.

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

app.whenReady().then(() => {
  const protocol = session.fromPartition('some-partition').protocol
  if (!protocol.registerFileProtocol('atom', (request, callback) => {
    const url = request.url.substr(7)
    callback({ path: path.normalize(path.join(__dirname, url)) })
  })) {
    console.error('Failed to register protocol')
  }
})
```

`ses.netLog` *Readonly*

A **NetLog** object for this session.

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

app.whenReady().then(async () => {
  const netLog = session.fromPartition('some-partition').netLog
  netLog.startLogging('/path/to/net-log')
  // After some network events
  const path = await netLog.stopLogging()
  console.log('Net-logs written to', path)
})
```

Edit this page