protocol

Register a custom protocol and intercept existing protocol requests.

Process: Main

An example of implementing a protocol that has the same effect as the `file://` protocol:

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

app.whenReady().then(() => {
  protocol.registerFileProtocol('atom', (request, callback) => {
    const url = request.url.substr(7)
    callback({ path: path.normalize(`${__dirname}/${url}`) })
  })
})
```

Note: All methods unless specified can only be used after the `ready` event of the `app` module gets emitted.

Using `protocol` with a custom `partition` or `session`

A protocol is registered to a specific Electron `session` object. If you don't specify a session, then your `protocol` will be applied to the default session that Electron uses. However, if you define a `partition` or `session` on your `browserWindow`'s `webPreferences`, then that window will use a different session and your custom protocol will not work if you just use `electron.protocol.XXX`.

To have your custom protocol work in combination with a custom session, you need to register it to that session explicitly.

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

app.whenReady().then(() => {
  const partition = 'persist:example'
  const ses = session.fromPartition(partition)

  ses.protocol.registerFileProtocol('atom', (request, callback) => {
    const url = request.url.substr(7)
    callback({ path: path.normalize(`${__dirname}/${url}`) })
  })

  mainWindow = new BrowserWindow({ webPreferences: { partition } })
})
```

Methods

The `protocol` module has the following methods:

`protocol.registerSchemesAsPrivileged(customSchemes)`

- `customSchemes` CustomScheme[]

Note: This method can only be used before the `ready` event of the `app` module gets emitted and can be called only once.

Registers the `scheme` as standard, secure, bypasses content security policy for resources, allows registering ServiceWorker, supports fetch API, and streaming video/audio. Specify a privilege with the value of `true` to enable the capability.

An example of registering a privileged scheme, that bypasses Content Security Policy:

```
const { protocol } = require('electron')
protocol.registerSchemesAsPrivileged([
  { scheme: 'foo', privileges: { bypassCSP: true } }
])
```

A standard scheme adheres to what RFC 3986 calls generic URI syntax. For example `http` and `https` are standard schemes, while `file` is not.

Registering a scheme as standard allows relative and absolute resources to be resolved correctly when served. Otherwise the scheme will behave like the `file` protocol, but without the ability to resolve relative URLs.

For example when you load following page with custom protocol without registering it as standard scheme, the image will not be loaded because non-standard schemes can not recognize relative URLs:

```
<body>
  <img src='test.png'>
</body>
```

Registering a scheme as standard will allow access to files through the FileSystem API. Otherwise the renderer will throw a security error for the scheme.

By default web storage apis (localStorage, sessionStorage, webSQL, indexedDB, cookies) are disabled for non standard schemes. So in general if you want to register a custom protocol to replace the `http` protocol, you have to register it as a standard scheme.

Protocols that use streams (http and stream protocols) should set `stream: true`. The `<video>` and `<audio>` HTML elements expect protocols to buffer their responses by default. The `stream` flag configures those elements to correctly expect streaming responses.

`protocol.registerFileProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` (string | ProtocolResponse)

Returns `boolean` - Whether the protocol was successfully registered

Registers a protocol of `scheme` that will send a file as the response. The `handler` will be called with `request` and `callback` where `request` is an incoming request for the `scheme`.

To handle the `request`, the `callback` should be called with either the file's path or an object that has a `path` property, e.g. `callback(filePath)` or `callback({ path: filePath })`. The `filePath` must be an absolute path.

By default the `scheme` is treated like `http:`, which is parsed differently from protocols that follow the "generic URI syntax" like `file:`.

`protocol.registerBufferProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` (Buffer | ProtocolResponse)

Returns `boolean` - Whether the protocol was successfully registered

Registers a protocol of `scheme` that will send a `Buffer` as a response.

The usage is the same with `registerFileProtocol`, except that the `callback` should be called with either a `Buffer` object or an object that has the `data` property.

Example:

```
protocol.registerBufferProtocol('atom', (request, callback) => {
  callback({ mimeType: 'text/html', data: Buffer.from('<h5>Response</h5>') })
})
```

`protocol.registerStringProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` (string | ProtocolResponse)

Returns `boolean` - Whether the protocol was successfully registered

Registers a protocol of `scheme` that will send a `string` as a response.

The usage is the same with `registerFileProtocol`, except that the `callback` should be called with either a `string` or an object that has the `data` property.

`protocol.registerHttpProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` ProtocolResponse

Returns `boolean` - Whether the protocol was successfully registered

Registers a protocol of `scheme` that will send an HTTP request as a response.

The usage is the same with `registerFileProtocol`, except that the `callback` should be called with an object that has the `url` property.

`protocol.registerStreamProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` (ReadableStream | ProtocolResponse)

Returns `boolean` - Whether the protocol was successfully registered

Registers a protocol of `scheme` that will send a stream as a response.

The usage is the same with `registerFileProtocol`, except that the `callback` should be called with either a `ReadableStream` object or an object that has the `data` property.

Example:

```
const { protocol } = require('electron')
const { PassThrough } = require('stream')

function createStream (text) {
  const rv = new PassThrough() // PassThrough is also a Readable stream
  rv.push(text)
  rv.push(null)
  return rv
}

protocol.registerStreamProtocol('atom', (request, callback) => {
```

```
  callback({
    statusCode: 200,
    headers: {
      'content-type': 'text/html'
    },
    data: createStream('<h5>Response</h5>')
  })
})
```

It is possible to pass any object that implements the readable stream API (emits `data`/`end`/`error` events). For example, here's how a file could be returned:

```
protocol.registerStreamProtocol('atom', (request, callback) => {
  callback(fs.createReadStream('index.html'))
})
```

`protocol.unregisterProtocol(scheme)`

- `scheme` string

Returns `boolean` - Whether the protocol was successfully unregistered

Unregisters the custom protocol of `scheme`.

`protocol.isProtocolRegistered(scheme)`

- `scheme` string

Returns `boolean` - Whether `scheme` is already registered.

`protocol.interceptFileProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` (string | ProtocolResponse)

Returns `boolean` - Whether the protocol was successfully intercepted

Intercepts `scheme` protocol and uses `handler` as the protocol's new handler which sends a file as a response.

`protocol.interceptStringProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` (string | ProtocolResponse)

Returns `boolean` - Whether the protocol was successfully intercepted

Intercepts `scheme` protocol and uses `handler` as the protocol's new handler which sends a `string` as a response.

`protocol.interceptBufferProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` (Buffer | ProtocolResponse)

Returns `boolean` - Whether the protocol was successfully intercepted

Intercepts `scheme` protocol and uses `handler` as the protocol's new handler which sends a `Buffer` as a response.

`protocol.interceptHttpProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` ProtocolResponse

Returns `boolean` - Whether the protocol was successfully intercepted

Intercepts `scheme` protocol and uses `handler` as the protocol's new handler which sends a new HTTP request as a response.

`protocol.interceptStreamProtocol(scheme, handler)`

- `scheme` string
- `handler` Function
  - `request` ProtocolRequest
  - `callback` Function
    - `response` (ReadableStream | ProtocolResponse)

Returns `boolean` - Whether the protocol was successfully intercepted

Same as `protocol.registerStreamProtocol`, except that it replaces an existing protocol handler.

`protocol.uninterceptProtocol(scheme)`

- `scheme` string

Returns `boolean` - Whether the protocol was successfully unintercepted

Remove the interceptor installed for `scheme` and restore its original handler.

`protocol.isProtocolIntercepted(scheme)`

- `scheme` string

Returns `boolean` - Whether `scheme` is already intercepted.