Electron Documentation Style Guide

These are the guidelines for writing Electron documentation.

Headings

- Each page must have a single `#`-level title at the top.
- Chapters in the same page must have `##`-level headings.
- Sub-chapters need to increase the number of `#` in the heading according to their nesting depth.
- The page's title must follow APA title case.
- All chapters must follow APA sentence case.

Using `Quick Start` as example:

```
# Quick Start

...

## Main process

...

## Renderer process

...

## Run your app

...

### Run as a distribution

...

### Manually downloaded Electron binary

...
```

For API references, there are exceptions to this rule.

Markdown rules

This repository uses the `markdownlint` package to enforce consistent Markdown styling. For the exact rules, see the `.markdownlint.json` file in the root folder.

There are a few style guidelines that aren't covered by the linter rules:

- Use `sh` instead of `cmd` in code blocks (due to the syntax highlighter).

- Keep line lengths between 80 and 100 characters if possible for readability purposes.
- No nesting lists more than 2 levels (due to the markdown renderer).
- All `js` and `javascript` code blocks are linted with standard-markdown.
- For unordered lists, use asterisks instead of dashes.

Picking words

- Use "will" over "would" when describing outcomes.
- Prefer "in the _____ process" over "on".

API references

The following rules only apply to the documentation of APIs.

Title and description

Each module's API doc must use the actual object name returned by `require('electron')` as its title (such as `BrowserWindow`, `autoUpdater`, and `session`).

Directly under the page title, add a one-line description of the module as a markdown quote (beginning with `>`).

Using the `session` module as an example:

```
# session

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

Module methods and events

For modules that are not classes, their methods and events must be listed under the `## Methods` and `## Events` chapters.

Using `autoUpdater` as an example:

```
# autoUpdater

## Events

### Event: 'error'

## Methods

### `autoUpdater.setFeedURL(url[, requestHeaders])`
```

Classes

- API classes or classes that are part of modules must be listed under a `## Class: TheClassName` chapter.
- One page can have multiple classes.
- Constructors must be listed with `###`-level headings.
- Static Methods must be listed under a `### Static Methods` chapter.
- Instance Methods must be listed under an `### Instance Methods` chapter.
- All methods that have a return value must start their description with "Returns `[TYPE]` - [Return description]"
  - If the method returns an `Object`, its structure can be specified using a colon followed by a newline then an unordered list of properties in the same style as function parameters.
- Instance Events must be listed under an `### Instance Events` chapter.
- Instance Properties must be listed under an `### Instance Properties` chapter.
  - Instance Properties must start with "A [Property Type] ..."

Using the `Session` and `Cookies` classes as an example:

```
# session

## Methods

### session.fromPartition(partition)

## Static Properties

### session.defaultSession

## Class: Session

### Instance Events

#### Event: 'will-download'

### Instance Methods

#### `ses.getCacheSize()`

### Instance Properties

#### `ses.cookies`

## Class: Cookies

### Instance Methods
```

```
#### `cookies.get(filter, callback)`
```

Methods and their arguments

The methods chapter must be in the following form:

```
### `objectName.methodName(required[, optional]))`

* `required` string - A parameter description.
* `optional` Integer (optional) - Another parameter description.

...
```

Heading level The heading can be `###` or `####`-levels depending on whether the method belongs to a module or a class.

Function signature For modules, the `objectName` is the module's name. For classes, it must be the name of the instance of the class, and must not be the same as the module's name.

For example, the methods of the `Session` class under the `session` module must use `ses` as the `objectName`.

Optional arguments are notated by square brackets `[]` surrounding the optional argument as well as the comma required if this optional argument follows another argument:

```
required[, optional]
```

Argument descriptions More detailed information on each of the arguments is noted in an unordered list below the method. The type of argument is notated by either JavaScript primitives (e.g. `string`, `Promise`, or `Object`), a custom API structure like Electron's `Cookie`, or the wildcard `any`.

If the argument is of type `Array`, use `[]` shorthand with the type of value inside the array (for example,`any[]` or `string[]`).

If the argument is of type `Promise`, parametrize the type with what the promise resolves to (for example, `Promise<void>` or `Promise<string>`).

If an argument can be of multiple types, separate the types with `|`.

The description for `Function` type arguments should make it clear how it may be called and list the types of the parameters that will be passed to it.

Platform-specific functionality If an argument or a method is unique to certain platforms, those platforms are denoted using a space-delimited italicized list following the datatype. Values can be `macOS`, `Windows` or `Linux`.

```
* `animate` boolean (optional) _macOS_ _Windows_ - Animate the thing.
```

Events

The events chapter must be in following form:

```
### Event: 'wake-up'

Returns:

* `time` string

...
```

The heading can be `###` or `####`-levels depending on whether the event belongs to a module or a class.

The arguments of an event follow the same rules as methods.

Properties

The properties chapter must be in following form:

```
### session.defaultSession

...
```

The heading can be `###` or `####`-levels depending on whether the property belongs to a module or a class.

Documentation translations

See electron/i18n