Creating a New Electron Browser Module

Welcome to the Electron API guide! If you are unfamiliar with creating a new Electron API module within the `browser` directory, this guide serves as a checklist for some of the necessary steps that you will need to implement.

This is not a comprehensive end-all guide to creating an Electron Browser API, rather an outline documenting some of the more unintuitive steps.

Add your files to Electron's project configuration

Electron uses GN as a meta build system to generate files for its compiler, Ninja. This means that in order to tell Electron to compile your code, we have to add your API's code and header file names into `filenames.gni`.

You will need to append your API file names alphabetically into the appropriate files like so:

“‘cpp title=‘filenames.gni’ lib_sources = [“path/to/api/api_name.cc”, “path/to/api/api_name.h”,]

lib_sources_mac = [“path/to/api/api_name_mac.h”, “path/to/api/api_name_mac.mm”,]

lib_sources_win = [“path/to/api/api_name_win.cc”, “path/to/api/api_name_win.h”,]

lib_sources_linux = [“path/to/api/api_name_linux.cc”, “path/to/api/api_name_linux.h”,]

~~~
Note that the Windows, macOS and Linux array additions are optional and should only be addec

## Create API documentation

Type definitions are generated by Electron using [`@electron/docs-parser`](https://github.cc

## Set up `ObjectTemplateBuilder` and `Wrappable`

Electron constructs its modules using [`object_template_builder`](https://www.electronjs.org

[`wrappable`](https://chromium.googlesource.com/chromium/src/+/refs/heads/main/gin/wrappable

Here is a basic example of code that you may need to add, in order to incorporate `object_te

In your `api_name.h` file:

```cpp title='api_name.h'

#ifndef ELECTRON_SHELL_BROWSER_API_ELECTRON_API_{API_NAME}_H_
#define ELECTRON_SHELL_BROWSER_API_ELECTRON_API_{API_NAME}_H_

#include "gin/handle.h"
~~~

```
#include "gin/wrappable.h"

namespace electron {

namespace api {

class ApiName : public gin::Wrappable<ApiName>  {
 public:
  static gin::Handle<ApiName> Create(v8::Isolate* isolate);

  // gin::Wrappable
  static gin::WrapperInfo kWrapperInfo;
  gin::ObjectTemplateBuilder GetObjectTemplateBuilder(
      v8::Isolate* isolate) override;
  const char* GetTypeName() override;
} // namespace api
} // namespace electron
```

In your `api_name.cc` file:

"'cpp title='api_name.cc' #include "shell/browser/api/electron_api_safe_storage.h"

#include "shell/browser/browser.h" #include "shell/common/gin_converters/base_converter.h" #include "shell/common/gin_converters/callback_converter.h" #include "shell/common/gin_helper/dictionary.h" #include "shell/common/gin_helper/object_template_builder.h" #include "shell/common/node_includes.h" #include "shell/common/platform_util.h"

namespace electron {

namespace api {

gin::WrapperInfo ApiName::kWrapperInfo = {gin::kEmbedderNativeGin};

gin::ObjectTemplateBuilder ApiName::GetObjectTemplateBuilder(v8::Isolate* isolate) { return gin::ObjectTemplateBuilder(isolate) .SetMethod("methodName", &ApiName::methodName); }

const char* ApiName::GetTypeName() { return "ApiName"; }

// static gin::Handle ApiName::Create(v8::Isolate* isolate) { return gin::CreateHandle(isolate, new ApiName()); }

} // namespace api

} // namespace electron

namespace {

void Initialize(v8::Local exports, v8::Local unused, v8::Local context, void* priv) { v8::Isolate* isolate = context->GetIsolate(); gin_helper::Dictionary dict(isolate, exports); dict.Set("apiName", electron::api::ApiName::Create(isolate)); }

} // namespace

~~~~
## Link your Electron API with Node

In the [`typings/internal-ambient.d.ts`](https://github.com/electron/electron/blob/main/typi

```ts title='typings/internal-ambient.d.ts'
interface Process {
    _linkedBinding(name: 'electron_browser_{api_name}', Electron.ApiName);
}
~~~~

At the very bottom of your `api_name.cc` file:

`cpp title='api_name.cc' NODE_LINKED_MODULE_CONTEXT_AWARE(electron_browser_{api_name},Initia`

In your `shell/common/node_bindings.cc` file, add your node binding name to Electron's built-in modules.

```
cpp title='shell/common/node_bindings.cc' #define ELECTRON_BUILTIN_MODULES(V)
\   V(electron_browser_{api_name})
```

> Note: More technical details on how Node links with Electron can be found on our blog.

Expose your API to TypeScript

Export your API as a module

We will need to create a new TypeScript file in the path that follows:

`"lib/browser/api/{electron_browser_{api_name}}.ts"`

An example of the contents of this file can be found here.

Expose your module to TypeScript

Add your module to the module list found at `"lib/browser/api/module-list.ts"` like so:

```
typescript title='lib/browser/api/module-list.ts' export const
browserModuleList: ElectronInternal.ModuleEntry[] = [   { name:
'apiName', loader: () => require('./api-name') }, ];
```