# Firesale

## Clone the Repository

**https://github.com/stevekinney/firesale-tutorial**

## We'll be working with four files for the duration of this tutorial:

- `lib/main.js`
- `lib/renderer.js`
- `lib/index.html`
- `lib/style.css`

## In `main.js`:

```
const electron = require('electron');
const electron = require('electron');
const app = electron.app;
const electron = require('electron');
const app = electron.app;

app.on('ready', function () {
  console.log('The application is ready.');
});
```

**There isn't much to look at yet, but if we run `electron .`, you should notice the following.**

1. **Our message is logged to the console.**
2. **An Electron icon pops up in the Dock.**

## Let's pull in the `BrowserWindow` module:

```
const BrowserWindow = electron.BrowserWindow;
```

## We'll need to define `mainWindow` as `null` in the top-level scope to avoid garbage collection.

1. **JavaScript has function scopes.**
2. **Our** `ready` **event listener is a function.**

**If we declared it in the lower scope, then it would eligible for garbage collection when we left that scope.**

## In `main.js`:

```
const electron = require('electron');
const app = electron.app;
const BrowserWindow = electron.BrowserWindow;

var mainWindow = null;

app.on('ready', function () {
  console.log('The application is ready.');

  mainWindow = new BrowserWindow();

  mainWindow.on('closed', function() {
    mainWindow = null;
  });
});
```

## In `main.js`:

```
app.on('ready', function () {
  console.log('The application is ready.');

  mainWindow = new BrowserWindow();

  mainWindow.loadURL('file://' + __dirname + '/index.html');

  mainWindow.on('closed', function() {
    mainWindow = null;
  });
});
```

## In index.html:

```html
<section class="controls">
  <button id="open-file">Open File</button>
  <button id="copy-html">Copy HTML</button>
  <button id="save-file">Save HTML</button>
</section>

<section class="content">
  <textarea class="raw-markdown"></textarea>
  <div class="rendered-html"></div>
</section>
```

## Requiring the Dialog Module

**In `main.js`:**

```
const dialog = electron.dialog;
```

## In `main.js`:

```
const openFile = function () {
  var files = dialog.showOpenDialog(mainWindow, {
    properties: ['openFile']
  });

  if (!files) { return; }

  console.log(files);
};
```

## We'll call this function immediately when the application is ready for now.

```
app.on('ready', function () {
  // More code…

  openFile();

  // More code…
});
```

## We only want the first file:

```
const openFile = function () {
  var files = dialog.showOpenDialog(mainWindow, {
    properties: ['openFile']
  });

  if (!files) { return; }

  var file = files[0];

  console.log(file);
};
```

## First we need the `fs` library from Node.

```
const fs = require('fs');
```

`fs.readFileSync` **returns a** `Buffer` **object. We know we're working with text. So, we'll turn that into a string using the** `toString()` **method.**

```
var file = files[0];
var content = fs.readFileSync(file).toString();
```

## Let's also filter by valid file extensions:

```
var files = dialog.showOpenDialog(mainWindow, {
  properties: ['openFile'],
  filters: [
    { name: 'Markdown Files', extensions: ['md', 'markdown', 'txt'] }
  ]
});
```

Sending Content to the Renderer Process

## Instead of logging to the console, let's send the content to the `mainWindow`. Replace the `console.log` in `openFile` with the following:

```
mainWindow.webContents.send('file-opened', file, content);
```

## In `index.html`, we'll load up the code for our renderer process.

```html
<script>
  require('./renderer');
</script>
```

## If we wanted the developer tools to open by default, we could add the following to our `main.js`:

```
app.on('ready', function () {
  // More code above…

  mainWindow.webContents.openDevTools();

  // More code below…
});
```

**The main process is sending the contents to the renderer process. But, the we need to listen for that message.**

**In** `renderer.js`**, we'll require Electron and the** `ipcRenderer` **module.**

```
const electron = require('electron');
const ipc = electron.ipcRenderer;
```

```
ipc.on('file-opened', function (event, file, content) {
  console.log(content);
});
```

**Now, let's do something with that content. To make things easier, we'll require jQuery in `renderer.js`:**

```
const $ = require('jquery');
```

## Let's also cache some selectors:

```
const $markdownView = $('.raw-markdown');
const $htmlView = $('.rendered-html');
const $openFileButton = $('#open-file');
const $saveFileButton = $('#save-file');
const $copyHtmlButton = $('#copy-html');
```

**When the renderer process gets a message on the `file-opened` channel from the main process, we'll display those contents in the `$markdownView` element.**

```
ipc.on('file-opened', function (event, file, content) {
  $markdownView.text(content);
});
```

## We'll use marked to render our Markdown as HTML.

**In** `renderer.js`**:**

```
const marked = require('marked');
```

## We're going to use this functionality in more than one place, so we'll make it it's own function:

```
function renderMarkdownToHtml(markdown) {
  var html = marked(markdown);
  $htmlView.html(html);
}
```

## When we hear that a file has opened, we'll update both parts of our application. In `renderer.js`:

```
ipc.on('file-opened', function (event, file, content) {
  $markdownView.text(content);
  renderMarkdownToHtml(content);
});
```

## Updating the HTML When the Markdown Changes

**Let's listen for the `keyup` event and reuse our `renderMarkdownToHtml` function.**

```
$markdownView.on('keyup', function () {
  var content = $(this).val();
  renderMarkdownToHtml(content);
});
```

## Wiring Up the Buttons

**In our application, we have three buttons in the top bar:**

1. **Open File**
2. **Copy HTML**
3. **Save HTML**

## As we mentioned earlier, we should *not* spawn a dialog box from the renderer process.

**Instead, we'll let the main process be in charge of that and have the renderer process just ask the main process.**

## We'll use the `remote` module in `renderer.js`:

```
const remote = electron.remote;
```

## Once we have the remote module, we can use it load up the main process.

**In** `renderer.js`**:**

```
const mainProcess = remote.require('./main');
```

## We'll export that `openFile` function we made a while back from `main.js`:

```
exports.openFile = openFile;
```

**Our `openFile` function is now available on the `mainProcess` object in `renderer.js`.**

```
$openFileButton.on('click', () => {
  mainProcess.openFile();
});
```

**When the "Open File" button is clicked, it will call the `openFile` function from the main process and display the file dialog.**

## Working with the Clipboard

**Now that we have the first button in place, we'll go ahead and get the second button working.**

## Let's require the `clipboard` module in `renderer.js`:

```
const clipboard = remote.clipboard;
```

**When the user clicks on the "Copy HTML" button, we'll go ahead and write the contents of the** `$htmlView` **element to the clipboard.**

```
$copyHtmlButton.on('click', () => {
  var html = $htmlView.html();
  clipboard.writeText(html);
});
```

**That's all that's required.**

Saving Files

## Let's add the following to `main.js`...

```
const saveFile = function (content) {
  var fileName = dialog.showSaveDialog(mainWindow, {
    title: 'Save HTML Output',
    defaultPath: app.getPath('documents'),
    filters: [
      { name: 'HTML Files', extensions: ['html'] }
    ]
  });

  if (!fileName) { return; }

  fs.writeFileSync(fileName, content);
};
```

## We'll also want to export this functionality in `main.js`:

```
exports.saveFile = saveFile;
```

## In `renderer.js`:

```
$saveFileButton.on('click', () => {
  var html = $htmlView.html();
  mainProcess.saveFile(html);
});
```

**We've successfully implemented a first pass at saving files to the filesystem with Electron.**

## Adding Menu Items

**When we overwrite the default menu, we have to build our own from scratch.**

## Let's go and pull in Electron's `Menu` module in `main.js`.

```
const Menu = electron.Menu;
```

Unfortunately, Electron's default menu is a "take it or leave it" affair.

Electron *does* however give us the ability to create a simple data structure and have it build the menu from a template.

```
var menu = Menu.buildFromTemplate(template);
```

**(We haven't defined a `template`, so this won't work yet.)**

## Once we have a template, we'll load it up like this in `main.js`:

```
app.on('ready', function () {
  var menu = Menu.buildFromTemplate(template);
  Menu.setApplicationMenu(menu);
});
```

## Get the Template Code

**http://bit.ly/fluent-menu**

## Electron allows us to define their "role," which will trigger the native OS behavior.

```
{
  label: 'Copy',
  accelerator: 'CmdOrCtrl+C',
  role: 'copy'
}
```

## We only want to add this menu if our Electron application is running in OS X.

```
if (process.platform == 'darwin') { … }
```

Fixing a bug

## URLs open inside of our application. Oh no!

## Introducing the Shell Module

**In** `renderer.js`**:**

```
const shell = electron.shell;
```

## Now, we'll listen for link clicks and ask them politely to open in a new window instead of stepping over our little application.

```
$(document).on('click', 'a[href^="http"]', function (event) {
  event.preventDefault();
  shell.openExternal(this.href);
});
```

## Appending to the Recent Documents Menu

**Operating systems keep a record of recent files. We want our application to hook into this functionality. Doing this is fairly, simple. In our `openFile` function, we'll add the following:**

```
app.addRecentDocument(file);
```

## Actually Getting It Working

```
app.on('open-file', function (event, file) {
  var content = fs.readFileSync(file).toString();
  mainWindow.webContents.send('file-opened', file, content);
});
```

## Accessing the Outside World

**Two more features:**

- **Show in File System**
- **Open in Default Editor**

## Let's add the following to the `.controls` element in `index.html`:

```
<button id="show-in-file-system" disabled="true">Show in File System</button>
<button id="open-in-default-editor" disabled="true">Open in Default Editor</button>
```

## We'll also store a reference to each of them in `renderer.js`.

```js
const $showInFileSystemButton = $('#show-in-file-system');
const $openInDefaultEditorButton = $('#open-in-default-editor');
```

**In** `renderer.js`**:**

```
var currentFile = null;
```

## We'll also modify our `file-opened` event listener to update `currentFile` and enable the buttons.

```
ipc.on('file-opened', function (event, file, content) {
  currentFile = file;

  $showInFileSystemButton.attr('disabled', false);
  $openInDefaultEditorButton.attr('disabled', false);

  $markdownView.text(content);
  renderMarkdownToHtml(content);
});
```

Congratulations!

## You Built an Application!