Playing with the filesystem API

There have been multiple attempts at getting a Filesystem API on the web. The one I'm referring to in this post is a Chromium-only feature available without a flag since Chrome 86, not the old style API. This is also different from the API available to Chrome extensions.

I used a standard HTML5 Boilerplate document. We add a single CSS class inside a `style` element to hide elements.

```
.hidden {
  display: none;
}
```

The HTML for the demo is simple. We create a text area and two buttons and assign IDs to each; this will make it easier to manipulate the elements with Javascript.

```
<textarea id="content"></textarea>
<button id="open">Open File</button>
<button id="save">Save File</button>
```

I've broken the Javascript into four sections to make it easier to reason what each section does.

The first section defines constants that other parts of the script will use.

The first one (`supportsNativeFS`) is a feature query to detect support for the API.

The next block captures references to the text area and each button using getElementById.

```
const supportsNativeFS = 'chooseFileSystemEntries' in window ||
```

```
  'showOpenFilePicker' in window

const openBtn = document.getElementById('open');
const saveBtn = document.getElementById('save');
const content = document.getElementById('content');
```

Next, we check if the browser supports the Native Filesystem API (meaning, Chrome 86 and later).

If the feature is supported then we hide the not-supported message by adding the `hidden` class to the element with the `not-supported` ID, and we log the information to console.

If the feature is **not** supported then we hide the save button and log the information to the console.

For production code I'd put the event listeners inside the if statement or use the feature detection insde the listeners to see if the feature is supported.

```
if (supportsNativeFS) {
  document.getElementById('not-supported').classList.add('hidden');
  console.log('Browser supports Native Filesystem API');
} else {
  document.getElementById('butSave').classList.toggle('hidden', true);
  console.log('Native Filesystem API is not supported');
}
```

The event listener for the `openBtn` button does the following:

1. Grabs a reference to the file handle of the file the user selected
2. Gets the file
3. Puts the content of the file inside the contents variable
4. Inserts the value of contents (the text of the file) into the content textarea's inner HTML

```
let fileHandle;
let contents;
```

```
openBtn.addEventListener('click', async () => {
  [fileHandle] = await window.showOpenFilePicker(); // 1
  const file = await fileHandle.getFile(); // 2
  contents = await file.text(); // 3
  content.innerHTML = contents; // 4
});
```

The save button does more work. When the user clicks the save button, the browser will present a dialogue to enter a file name to save the content under.

> Once we enter the name of the file and click save, the browser will:

1. Create a `FileSystemWritableFileStream` to write to
2. Write the contents of the file to the stream.
3. Close the file and write the contents to disk.

```
saveBtn.addEventListener('click', async () => {
  const options = {
    types: [{
      description: 'Markdown Files',
      accept: 'Markdown Files'markdown': ['.md'],
      },
    }, ],
  };
  const handle = await window.showSaveFilePicker(options);

  const writable = await fileHandle.createWritable();': ['// 1
  await writable.write(contents); // 2
  await writable.close(); // 3
});
```

The File System Access API: simplifying access to local files presents examples of how to use the API to manage access to files. It goes more in depth about what you can do witht the API.

> Reading and writing files and directories with the browser-nativefs library provides an abstraction library for the Native Filesystem API available in different versions of Chrome.