Create in JavaScript your polyglot file

HTML / ZIP / PNG

Introduction

- Web developer for 20+ years
- Author of:
  - zip.js: library for reading and writing zip files
  - SingleFile: extension and command-line tool for saving a web page as a single HTML file
    - use of Data URIs to reference and store binary content
    - output file size larger than the size of all resources added together
- Can SingleFile and zip.js be combined?
- Is it possible to go further ?

Web project

1 - Creating a test project (HTML, CSS, JS) in `/project`

Inclusion of external resources:

- `<link href=style.css>`
- `<img src=image.png>`
- `<script src=script.js></script>`
- `@import url(properties.css)`
- `url(background.png)`

UTF-8 encoded content:

- `<figcaption>Communauté des développeurs ...</figcaption>`
- `alert("Hello la communauté RennesJS !  ")`
- `content: "  "`

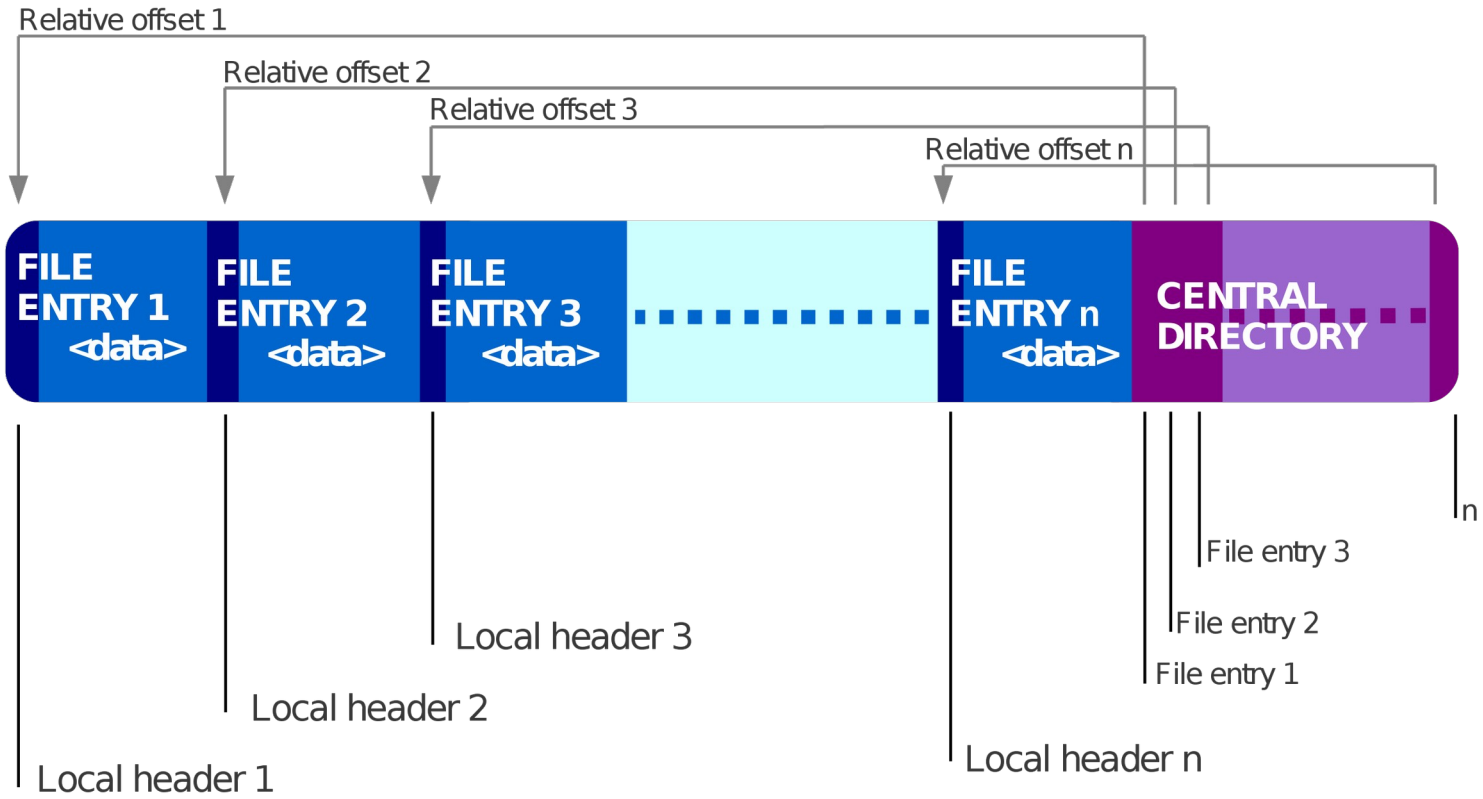
ZIP Format

- Created in 1989 by Phil Katz at PKWARE (publisher of PKZIP)
- Support of:
  - DEFLATE compression since version 2.0 (1993)
  - AES encryption since version 5.2 (2003)
  - Version 2.0+ on most operating systems
- Examples of formats based on the ZIP format:
  - LibreOffice/MS Office documents `.ODT`, `.DOCX`, ...
  - Java Archives `.JAR`
  - Android Packages `.APK` and iOS Archives `.IPA`
  - Web Extensions `.CRX` and `.XPI`

ZIP Format

- File entries followed by the central directory
- Suitable for streaming (read/write) but with some limitations
- Some metadata are stored twice in local headers and the central directory: file name, last modification date, data size, etc.
- The central directory is required and authoritative
- The central directory contains the positions (relative offsets) of each entry in the ZIP file
- The data in local headers can be used to repair a ZIP

Structure of a ZIP file



Source: https://en.wikipedia.org/wiki/ZIP_(file_format)

JavaScript & ZIP Format

- Native support in browsers, Node.js, Deno, Bun, etc.:
  - ZIP: no
  - DEFLATE: yes (via gzip/zlib)
- Rich ecosystem of third-party libraries:
  - zip.js
  - fflate
  - client-zip
  - yazl/yauzl
  - JSZip
  - ...

Example with zip.js

Writing a ZIP file

```
import { ZipWriter, Uint8ArrayWriter } from "@zip-js/zip-js"

const zipDataWriter = new Uint8ArrayWriter()
const zipWriter = new ZipWriter(zipDataWriter)

for await (const { name } of readDirectory(inputFolder)) {
    const readableStream = await readFileStream(name)
    await zipWriter.add(name, readableStream)
}

await zipWriter.close()
const zipData = zipDataWriter.getData() // Uint8Array
console.log("zip file data:", zipData)
```

Example with zip.js

Reading a ZIP file

```
import { ZipReader, BlobReader, BlobWriter } from "@zip-js/zip-js"

const zipReader = new ZipReader(new BlobReader(blob))
const entries = await zipReader.getEntries()

for (const entry of entries) {
    const blob = await entry.getData(new BlobWriter())
    console.log("file:", entry.filename, "blob:", blob)
}

await zipReader.close()
```

Integration in the web project

2 - Writing and reading the zip file in JavaScript

Addition of `/scripts`:

- `create-zip.js` and `read-zip.js`: test files
- `lib/utils-fs.js`:
  - `readDirectory(folder)`: returns the filenames in a directory as a string array
  - `readBinaryFile(filename)`: returns a file as a `Uint8Array`
  - `createFile(filename)`: creates a file and returns the write stream
- `lib/utils.js`:
  - `encodeText(text)`: encodes `UTF-8` text as a `Uint8Array`
- `lib/utils-zip.js`:
  - `getZipData(inputFolder)`: returns the `Uint8Array` content of a ZIP storing files read from a folder

ZIP format (cont.)

- Extensible format:
  - Add 64KB of data after the ZIP file (comment)
  - Offset greater than zero for the first entry in the zip file
- Self-extracting HTML page structure:
  - HTML content up to an opening tag `<!--`
  - ZIP file content
  - closing tag `-->` and end of HTML content
- HTML content includes:
  - Script of zip.js library (`zip.min.js`)
  - Script to extract and display the content of the zip file

Self-extracting HTML page template

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8>
    <title>Please wait...</title>
    <script>${ZIP_JS_SCRIPT}</script>
  </head>
  <body>
    <p>Please wait...</p>
    <!-- ZIP data -->
    <script type=module>${MAIN_SCRIPT}</script>
  </body>
</html>
```

Self-extracting HTML/ZIP file (1/2)

3 - Reading the ZIP file from the HTML page

Addition of `/assets`:

- `zip.min.js`: ZIP file reading
- `page-extraction.js`: extraction and display of the page

Additions to `/scripts`:

- `lib/html-template.js`: HTML template for self-extracting page
- `index.js`: generation of self-extracting HTML/ZIP file

Self-extracting HTML/ZIP file (2/2)

Changes in `/scripts`:

- Addition in `lib/utils.js`:
  - `minifyScript(script)`: returns the minified script
- Addition in `lib/utils-fs.js`:
  - `readTextFile(filename)`: returns the contents of a file as text

Self-extracting HTML/ZIP file

4 - Extracting and displaying the page `/index.html` from the ZIP file

Addition in `/assets/page-extraction.js`:

- `getExtension(filename)`: returns the extension of a file name (e.g. `".txt"`)

Changes in `/assets/page-extraction.js`:

- `extractResources`: add support for text files
- `displayPage`: HTML page display implementation

Self-extracting HTML/ZIP file

5 - Displaying the full HTML page

Addition in `/assets/page-extraction.js`:

- `resolveDependencies(resources)`: replaces external resource paths with resources extracted from the ZIP

6 - Handling scripts in the page

Change in `/assets/page-extraction.js`:

- `displayPage`: replace scripts in the DOM with their clones

Reading the page from the filesystem

Reading the ZIP file from the DOM (1/2)

7 - Displaying ZIP data in hexadecimal format and read in text form via `Node#textContent`

Temporary removals in `/assets/page-extraction.js`:

- `extractResources`
- `resolveDependencies`
- `getZipData`

Temporary change in `/assets/page-extraction.js`:

- Replacement of `displayPage(resources)` with `displayData()`

Addition of temporary styles to display text in monospace fonts in `/scripts/lib/html-template.js`

Reading the ZIP file from the DOM (2/2)

8 - Comparing the impact of `UTF-8` and `windows-1252` encoding

Change to the `charset` attribute of the `<meta>` tag

Reading the ZIP file from the DOM

9 - Testing data corruption caused by encodings and determine the best encoding

Data corruption:

- Character encoding on 1 byte (e.g. `windows-1252`) vs several bytes (e.g. `UTF-8`)
- Replacement by a line feed `\n` of all:
  - carriage return `\r`
  - carriage return followed immediately by a line feed `\r\n`
- Characters with a code greater than `127`, i.e. beyond the 7-bit ASCII table

Reading the ZIP file from the DOM (1/2)

10 - Bypassing the call to `await fetch("")`

Changes in `/scripts/lib/html-template.js`:

- Replacement of `utf-8` encoding with `windows-1252`
- Addition of ■ `--><script type=text/json>` text
- Addition of ■ ZIP extra data (encoded in JSON)
- Replacement of ■ `-->` with `</script>` in the end of the HTML content

Self-extracting HTML page template

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=windows-1252>
    <title>Please wait...</title>
    <script>${ZIP_JS_SCRIPT}</script>
  </head>
  <body>
    <p>Please wait...</p>
    <!-- ZIP data -->
    <script type=text/json> ZIP extra data </script>
    <script type=module>${MAIN_SCRIPT}</script>
  </body>
</html>
```

Reading the ZIP file from the DOM (2/2)

Addition in `/scripts/lib/utils-zip.js`:

- `getExtraData(zipData)`: generation of ZIP extra data

Change in `/scripts/index.js`:

- `writeFile`: addition of ZIP extra data

Change in `/assets/page-extraction.js`:

- `getZipData`: read content via the DOM comment (ZIP data) and the JSON script (ZIP extra data)

Reading the ZIP file from the DOM

11 - Fixing MIME type issues in external resources

Changes in `/assets/page-extraction.js`:

- `resolveDependencies`: support for MIME resource types (text or binary)
- `extractResources`: change of the code testing the filename extension of a resource

The page remains encoded in `windows-1252`

PNG format

- Created in 1996
- Standard, royalty-free image format
- Lossless compression (`DEFLATE`)
- File composed of a sequence of chunks
- Chunk structure:
  - Chunk data size (4 bytes)
  - Chunk type (4 bytes) : `IHDR`, `IDAT`, `IEND`, `tEXt` ...
  - Chunk data (variable size)
  - Cyclic redundant code (CRC32) computed from all block data

PNG format

- Minimal structure of a PNG file:

  PNG signature (8 bytes) - mandatory

  ```
  89 50 4E 47  0D 0A 1A 0A
  ```

  `IHDR` chunk for the header (13 bytes) - mandatory

  ```
  00 00 00 0D  49 48 44 52  ...
  ```

  `IDAT` chunk(s) for the image data

  `IEND` chunk for the trailer (12 bytes)

  ```
  00 00 00 00  49 45 4E 44  AE 42 60 82
  ```

- Optional chunk(s):
  - `tEXt` chunks for storing random text or binary data

    ```
    xx xx xx xx  74 45 58 74  ...
    ```

PNG format

Data type		Data in hexadecimal	Mandatory	Size (bytes)
PNG signature		89 50 4E 47 0D 0A 1A 0A	✓	8
Header	IHDR	00 00 00 0D **49 48 44 52** ...	✓	13
...				
Data	IDAT	xx xx xx xx **49 44 41 54** ...		12 + n
...				
Trailer	IEND	00 00 00 00 **49 45 4E 44** AE 42 60 82	✓	12

Polyglot PNG format (1/3)

12 - Encapsulating the HTML/ZIP file in a PNG file

Polyglot PNG file structure:

PNG signature

`IHDR` - Header chunk

`tEXt` - Text chunk

HTML content up to the opening tag `<!--`

`IDAT` - Data chunk(s)

`tEXt` - Text chunk

closing tag `-->` and end of the HTML content

`IEND` - Trailer chunk

Polyglot PNG format (2/3)

Data type		Data in hexadecimal	Mandatory	Size (bytes)
PNG signature		89 50 4E 47 0D 0A 1A 0A	✓	8
Header	IHDR	00 00 00 0D **49 48 44 52** ...	✓	13
Text	tEXt	xx xx xx xx **74 45 58 74** ...		12 + n
...				
Data	IDAT	xx xx xx xx **49 44 41 54** ...		12 + n
...				
Text	tEXt	xx xx xx xx **74 45 58 74** ...		12 + n
Trailer	IEND	00 00 00 00 **49 45 4E 44** AE 42 60 82	✓	12

Template of the polyglot HTML page

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=windows-1252>
    <title>Please wait...</title>
    <script>${ZIP_JS_SCRIPT}</script>
  </head>
  <body>
    <p>Please wait...</p>
    <!-- PNG data -->
    <!-- ZIP data -->
    <script type=text/json> ZIP extra data </script>
    <script type=module>${MAIN_SCRIPT}</script>
  </body>
</html>
```

Polyglot PNG format (3/3)

Change in `/assets/html-template.js`:

- Addition of the HTML comment to store image data (`IDAT` chunks)

Addition of `utils-png.js` in `/scripts/lib`:

- `createTextChunk(keyword, payloadData)`: returns a new `tEXt` chunk
- `getHeaderData(imageData)`: returns the PNG signature + `IHDR` chunk
- `getDataChunks(imageData)`: returns `IDAT` chunk(s)
- `getTrailerChunk(imageData)`: returns `IEND` chunk
- `getChunkDataOffset()`: returns data offset in a `tEXt` chunk

Addition in `/scripts/lib/utils.js`:

- `mergeData(...data)`: merges multiple instances of `Uint8Array`

Change in `/scripts/index.js`:

- `writeFile`: integrate PNG format

Polyglot PNG format

Improvements in `/assets/page-extraction.js`:

13 - Removing text nodes induced by PNG format

Addition of `cleanupPNGData()`

14 - Fixing HTML page rendering mode and improve support of scripts

Use `document.open()`, `document.write()` and `document.close()` in `displayPage()`

15 - Reusing the (polyglot) image in the HTML page

Deletion of the image in `DEPENDENCIES`

Move of image `image.png` in `/assets`

Replacement of `image.png` with `#` in `/project/index.html`

Conclusion

- Limitations of the final implementation:
  - Manual dependency resolution
  - Exceeding the 64KB data limit after ZIP
  - Presence of `-->` in ZIP or PNG binary data
  - Using `String#replaceAll()` to replace paths in text files instead of relying on parsing
  - No `<meta>` tag containing the `CSP`
  - No support for frames...
- Alternative formats: MHTML, Web Bundle, WARC/WACZ, MAFF ...
- Is it dangerous?
  - GIFAR