Generic Out of Band File Format Configuration for Node

Guy Bedford did all the leg work on these and has an open PR. He deserves all the credit. Send him lots of nice things : https://twitter.com/guybedford

The following lays out the first stage of an incremental plan to give Node a generic solution to the problem of file formats. This first stage only seeks to:

1. Setup well known presets for file format models, with the ability to add more complex models later
2. Integrate with ESM Loaders
3. Be able to be easily taught and reasoned about
4. Work in common cases but document edge case compromises
    a. The edge case compromises in this document all have alternative workflows should they be unacceptable.
5. Be declarative in nature to allow maximal compatibility with a variety of use cases

Further stages should be able to safely rely on these features as a more formal spec outside of this document is produced.

---

package.json#mode

Identified Problems

- ESM requires out of band information in order to reliably determine the format of a file
- Developers are seeking to use **.js** for the ESM format when possible

Considerations

- Method of consuming a file determining format means that files can exist in multiple modes at the same time, which has been considered problematic.
- For new applications, a common scenario is to write only ESM and not support CJS
- Categorical similarities to "format" in the ESM loader hooks would seek to use the same shortened forms when it makes sense.

Usage

Add a "mode" field to your package.json with one of two well known values for now. These will change the behavior of various file extensions within a package.

"mode"	**affect**
esm	.js files are treated as ESM
commonjs	.js files are treated as CJS

Note: when a file is treated as ESM, files cannot be **require()**'d since **require** does not work on ESM.

Reasoning

- .mjs can be used as an escape hatch for mixed format applications; designs regarding mixed formats can defer to the "commonjs" mode and .mjs.
- package.json is a well known boundary and does not impede JS spec like reject "use module" pragma. Developers can nest directories and package.json files to change the mode they are operating in. The mode is limited to package boundaries and setting the mode in one package boundary will not affect another. Package boundaries are defined as any directory containing a package.json file.
- mode is setup as a string with the intent to be scalable over time, it builds upon the idea of having presets as described in other designs, It takes an incremental approach of only introducing the well known presets first. It is capable of scaling to more complex designs as needed.

--mode

Identified Problems

- Not all situations have a package.json file
- **require()** treats any file without an extension as CJS

Considerations

- Categorical similarities to "format"/"mode" would seek to use the same shortened forms when it makes sense.

Usage

Add a --mode switch to your command line invocation.

--mode	**affect**
esm	Loads the CWD as a package boundary with the specified "mode" of "esm"
commonjs	Loads the CWD as a package boundary with the specified "mode" of "commonjs"

Note: when a file is treated as ESM, files cannot be **require()**'d since **require** does not work on ESM.

Reasoning

- Packages with a package.json can set the "mode" field
- Even if files do not have a package.json there may be multiple of them, this flag allows the default behavior to be set. Being unable to consume non-package CJS while this flag is set to "esm" seems a reasonable edge case. In such a situation a directory structure could be made, or using .mjs and .js without the flag works.aa

Punted

The following was punted to be a later addition **if** WASM/Webpackage/etc. prove to need it. The design laid out below should integrate directly against the designs above.

--format

Identified Problems

- **bin** files often do not have an extension
- When piping a file to Node over **STDIN**, there is no way to declare the format of the input

Considerations

- This remains separated from "mode" since it does not identify with any form of file extension.
- Matches the loader hook value for "format"
- package.json "bin" path values can contain extensions even if the name does not
- Categorical similarities to "mode" would seek to use the same shortened forms when it makes sense.

Usage

Add a --format switch to your command line invocation.

--format	**affect**
esm	The entry is treated as ESM
commonjs	The entry is treated as CJS

Note: when a file is treated as ESM, files cannot be **require()**'d since **require** does not work on ESM.

Reasoning

- While setup to scale, the main historical intent of Node is to consume .js files. The use of shorthands reflect this and allows an implementation to land without discussing the exact format of any future values intended for scaling the design space.
- Shorthands of "esm" and "common" can be reasoned about to expand to larger more complex values such as a MIME as scaling occurs. These values in particular do not contain any special characters and can be considered a reserved namespace of sorts.