Modules: `module` API

The `Module` object

- {Object}

Provides general utility methods when interacting with instances of `Module`, the `module` variable often seen in CommonJS modules. Accessed via `import 'module'` or `require('module')`.

`module.builtinModules`

- {string[]}

A list of the names of all modules provided by Node.js. Can be used to verify if a module is maintained by a third party or not.

`module` in this context isn't the same object that's provided by the module wrapper. To access it, require the `Module` module:

```mjs
// module.mjs
// In an ECMAScript module
import { builtinModules as builtin } from 'module';
```

```cjs
// module.cjs
// In a CommonJS module
const builtin = require('module').builtinModules;
```

`module.createRequire(filename)`

- `filename` {string|URL} Filename to be used to construct the require function. Must be a file URL object, file URL string, or absolute path string.
- Returns: {require} Require function

```mjs
import { createRequire } from 'module';
const require = createRequire(import.meta.url);

// sibling-module.js is a CommonJS module.
const siblingModule = require('./sibling-module');
```

`module.syncBuiltinESMExports()`

The `module.syncBuiltinESMExports()` method updates all the live bindings for builtin ES Modules to match the properties of the CommonJS exports. It does not add or remove exported names from the ES Modules.

```cjs
const fs = require('fs');
const assert = require('assert');
const { syncBuiltinESMExports } = require('module');

fs.readFile = newAPI;
```

```
delete fs.readFileSync;

function newAPI() {
  // ...
}

fs.newAPI = newAPI;

syncBuiltinESMExports();

import('fs').then((esmFS) => {
  // It syncs the existing readFile property with the new value
  assert.strictEqual(esmFS.readFile, newAPI);
  // readFileSync has been deleted from the required fs
  assert.strictEqual('readFileSync' in fs, false);
  // syncBuiltinESMExports() does not remove readFileSync from esmFS
  assert.strictEqual('readFileSync' in esmFS, true);
  // syncBuiltinESMExports() does not add names
  assert.strictEqual(esmFS.newAPI, undefined);
});
```

Source map v3 support

> *Stability: 1 - Experimental*

Helpers for interacting with the source map cache. This cache is populated when source map parsing is enabled and source map include directives are found in a modules' footer.

To enable source map parsing, Node.js must be run with the flag `--enable-source-maps`, or with code coverage enabled by setting `NODE_V8_COVERAGE=dir`.

```
// module.mjs
// In an ECMAScript module
import { findSourceMap, SourceMap } from 'module';
```

```
// module.cjs
// In a CommonJS module
const { findSourceMap, SourceMap } = require('module');
```

`module.findSourceMap(path)`

- `path` {string}
- Returns: {module.SourceMap}

`path` is the resolved path for the file for which a corresponding source map should be fetched.

Class: `module.SourceMap`

`new SourceMap(payload)`

- `payload` {Object}

Creates a new `sourceMap` instance.

`payload` is an object with keys matching the Source map v3 format:

- `file` : {string}
- `version` : {number}
- `sources` : {string[]}
- `sourcesContent` : {string[]}
- `names` : {string[]}
- `mappings` : {string}
- `sourceRoot` : {string}

`sourceMap.payload`

- Returns: {Object}

Getter for the payload used to construct the `SourceMap` instance.

`sourceMap.findEntry(lineNumber, columnNumber)`

- `lineNumber` {number}
- `columnNumber` {number}
- Returns: {Object}

Given a line number and column number in the generated source file, returns an object representing the position in the original file. The object returned consists of the following keys:

- generatedLine: {number}
- generatedColumn: {number}
- originalSource: {string}
- originalLine: {number}
- originalColumn: {number}
- name: {string}