Internationalization support

Node.js has many features that make it easier to write internationalized programs. Some of them are:

- Locale-sensitive or Unicode-aware functions in the ECMAScript Language Specification:
  - `String.prototype.normalize()`
  - `String.prototype.toLowerCase()`
  - `String.prototype.toUpperCase()`
- All functionality described in the ECMAScript Internationalization API Specification (aka ECMA-402):
  - `Intl` object
  - Locale-sensitive methods like `String.prototype.localeCompare()` and `Date.prototype.toLocaleString()`
- The WHATWG URL parser's internationalized domain names (IDNs) support
- `require('buffer').transcode()`
- More accurate REPL line editing
- `require('util').TextDecoder`
- `RegExp` Unicode Property Escapes

Node.js and the underlying V8 engine use International Components for Unicode (ICU) to implement these features in native C/C++ code. The full ICU data set is provided by Node.js by default. However, due to the size of the ICU data file, several options are provided for customizing the ICU data set either when building or running Node.js.

Options for building Node.js

To control how ICU is used in Node.js, four `configure` options are available during compilation. Additional details on how to compile Node.js are documented in BUILDING.md.

- `--with-intl=none` / `--without-intl`
- `--with-intl=system-icu`
- `--with-intl=small-icu`
- `--with-intl=full-icu` (default)

An overview of available Node.js and JavaScript features for each `configure` option:

Feature	`none`	`system-icu`	`small-icu`	`full-icu`
`String.prototype.normalize()`	none (function is no-op)	full	full	full
`String.prototype.to*Case()`	full	full	full	full
`Intl`	none (object does not exist)	partial/full (depends on OS)	partial (English-only)	full
`String.prototype.localeCompare()`	partial (not locale-aware)	full	full	full
`String.prototype.toLocale*Case()`	partial (not locale-aware)	full	full	full

`Number.prototype.toLocaleString()`	partial (not locale-aware)	partial/full (depends on OS)	partial (English-only)	full
`Date.prototype.toLocale*String()`	partial (not locale-aware)	partial/full (depends on OS)	partial (English-only)	full
Legacy URL Parser	partial (no IDN support)	full	full	full
WHATWG URL Parser	partial (no IDN support)	full	full	full
`require('buffer').transcode()`	none (function does not exist)	full	full	full
REPL	partial (inaccurate line editing)	full	full	full
`require('util').TextDecoder`	partial (basic encodings support)	partial/full (depends on OS)	partial (Unicode-only)	full
`RegExp` Unicode Property Escapes	none (invalid `RegExp` error)	full	full	full

The "(not locale-aware)" designation denotes that the function carries out its operation just like the non-`Locale` version of the function, if one exists. For example, under `none` mode, `Date.prototype.toLocaleString()` 's operation is identical to that of `Date.prototype.toString()` .

Disable all internationalization features (`none`)

If this option is chosen, ICU is disabled and most internationalization features mentioned above will be **unavailable** in the resulting `node` binary.

Build with a pre-installed ICU (`system-icu`)

Node.js can link against an ICU build already installed on the system. In fact, most Linux distributions already come with ICU installed, and this option would make it possible to reuse the same set of data used by other components in the OS.

Functionalities that only require the ICU library itself, such as `String.prototype.normalize()` and the WHATWG URL parser, are fully supported under `system-icu` . Features that require ICU locale data in addition, such as `Intl.DateTimeFormat` *may* be fully or partially supported, depending on the completeness of the ICU data installed on the system.

Embed a limited set of ICU data (`small-icu`)

This option makes the resulting binary link against the ICU library statically, and includes a subset of ICU data (typically only the English locale) within the `node` executable.

Functionalities that only require the ICU library itself, such as `String.prototype.normalize()` and the WHATWG URL parser, are fully supported under `small-icu` . Features that require ICU locale data in addition, such

as `Intl.DateTimeFormat`, generally only work with the English locale:

```
const january = new Date(9e8);
const english = new Intl.DateTimeFormat('en', { month: 'long' });
const spanish = new Intl.DateTimeFormat('es', { month: 'long' });

console.log(english.format(january));
// Prints "January"
console.log(spanish.format(january));
// Prints "M01" on small-icu
// Should print "enero"
```

This mode provides a balance between features and binary size.

Providing ICU data at runtime

If the `small-icu` option is used, one can still provide additional locale data at runtime so that the JS methods would work for all ICU locales. Assuming the data file is stored at `/some/directory`, it can be made available to ICU through either:

- The `NODE_ICU_DATA` environment variable:

  ```
  env NODE_ICU_DATA=/some/directory node
  ```

- The `--icu-data-dir` CLI parameter:

  ```
  node --icu-data-dir=/some/directory
  ```

(If both are specified, the `--icu-data-dir` CLI parameter takes precedence.)

ICU is able to automatically find and load a variety of data formats, but the data must be appropriate for the ICU version, and the file correctly named. The most common name for the data file is `icudt6X[bl].dat`, where `6X` denotes the intended ICU version, and `b` or `l` indicates the system's endianness. Check "ICU Data" article in the ICU User Guide for other supported formats and more details on ICU data in general.

The full-icu npm module can greatly simplify ICU data installation by detecting the ICU version of the running `node` executable and downloading the appropriate data file. After installing the module through `npm i full-icu`, the data file will be available at `./node_modules/full-icu`. This path can be then passed either to `NODE_ICU_DATA` or `--icu-data-dir` as shown above to enable full `Intl` support.

Embed the entire ICU (`full-icu`)

This option makes the resulting binary link against ICU statically and include a full set of ICU data. A binary created this way has no further external dependencies and supports all locales, but might be rather large. This is the default behavior if no `--with-intl` flag is passed. The official binaries are also built in this mode.

Detecting internationalization support

To verify that ICU is enabled at all (`system-icu`, `small-icu`, or `full-icu`), simply checking the existence of `Intl` should suffice:

```
const hasICU = typeof Intl === 'object';
```

Alternatively, checking for `process.versions.icu`, a property defined only when ICU is enabled, works too:

```
const hasICU = typeof process.versions.icu === 'string';
```

To check for support for a non-English locale (i.e. `full-icu` or `system-icu`), `Intl.DateTimeFormat` can be a good distinguishing factor:

```
const hasFullICU = (() => {
  try {
    const january = new Date(9e8);
    const spanish = new Intl.DateTimeFormat('es', { month: 'long' });
    return spanish.format(january) === 'enero';
  } catch (err) {
    return false;
  }
})();
```

For more verbose tests for `Intl` support, the following resources may be found to be helpful:

- btest402: Generally used to check whether Node.js with `Intl` support is built correctly.
- Test262: ECMAScript's official conformance test suite includes a section dedicated to ECMA-402.