Maintaining ICU in Node.js

Background

International Components for Unicode (ICU4C) is used both by V8 and also by Node.js directly to provide internationalization functionality. To quote from icu-project.org:

> *ICU is a mature, widely used set of C/C++ and Java libraries providing Unicode and Globalization support for software applications. ICU is widely portable and gives applications the same results on all platforms and between C/C++ and Java software.*

Data dependencies

ICU consumes and includes:

- Extracted locale data from CLDR
- Extracted Unicode data.
- Time zone (tz) data

The current versions of these items can be viewed for node with `node -p process.versions`:

```
$ node -p process.versions

{
  …
  cldr: '35.1',
  icu: '64.2',
  tz: '2019a',
  unicode: '12.1'
}
```

Time zone data

Time zone data files are updated independently of ICU CLDR data. ICU and its main data files do not need to be upgraded in order to apply time zone data file fixes.

The IANA tzdata project releases new versions and announces them on the `tz-announce` mailing list.

The Unicode project takes new releases and publishes updated time zone data files in the icu/icu-data repository.

All modern versions of Node.js use the version 44 ABI of the time zone data files.

Example: updating the ICU `.dat` file

- Decompress `deps/icu/source/data/in/icudt##l.dat.bz2`, where `##` is the ICU major version number.
- Clone the icu/icu-data repository and copy the latest `tzdata` release `le` files into the `source/data/in` directory.
- Follow the upstream ICU instructions to patch the ICU `.dat` file:

  > *`for i in zoneinfo64.res windowsZones.res timezoneTypes.res metaZones.res; do icupkg -a $i icudt*l.dat`*

- Optionally, verify that there is only one of the above files listed when using `icupkg -l`.
- Optionally, extract each file using `icupkg -x` and verify the `shasum` matches the desired value.
- Compress the `.dat` file with the same filename as in the first step.
- Build, test, verifying `process.versions.tz` matches the desired version.
- Create a new minor version release.

Release schedule

ICU typically has >1 release a year, particularly coinciding with a major release of Unicode. The current release schedule is available on the ICU website on the left sidebar.

V8 depends on ICU

V8 will aggressively upgrade to a new ICU version, due to requirements for features/bugfixes needed for Ecma402 support. The minimum required version of ICU is specified within the V8 source tree. If the ICU version is too old, V8 will not compile.

```
// deps/v8/src/objects/intl-objects.h
#define V8_MINIMUM_ICU_VERSION 65
```

V8 in Node.js depends on the ICU version supplied by Node.js.

The file `tools/icu/icu_versions.json` contains the current minimum version of ICU that Node.js is known to work with. This should be *at least* the same version as V8, so that users will find out earlier that their ICU is too old. A test case validates this when Node.js is built.

How to upgrade ICU

- Make sure your Node.js workspace is clean (`git status` should be sufficient).
- Configure Node.js with the specific ICU version you want to upgrade to, for example:

```
./configure \
    --with-intl=full-icu \
    --with-icu-source=https://github.com/unicode-org/icu/releases/download/release-
67-1/icu4c-67_1-src.tgz
make
```

> *Note in theory, the equivalent `vcbuild.bat` commands should work also, but the commands below are makefile-centric.*

- If there are ICU version-specific changes needed, you may need to make changes in `tools/icu/icu-generic.gyp` or add patch files to `tools/icu/patches`.
    - Specifically, look for the lists in `sources!` in the `tools/icu/icu-generic.gyp` for files to exclude.
- Verify the Node.js build works:

```
make test-ci
```

Also running

```
new Intl.DateTimeFormat('es', { month: 'long' }).format(new Date(9E8));
```

…Should return `enero` not `January`.

- Now, run the shrink tool to update `deps/icu-small` from `deps/icu`

> *:warning: Do not modify any source code in `deps/icu-small` ! See section below about floating patches to ICU.*

```
python tools/icu/shrink-icu-src.py
```

- Now, do a clean rebuild of Node.js to test:

```
make -k distclean
./configure
make
```

- Test this newly default-generated Node.js

```
process.versions.icu;
new Intl.DateTimeFormat('es', { month: 'long' }).format(new Date(9E8));
```

(This should print your updated ICU version number, and also `enero` again.)

You are ready to check in (`git add`) the updated `deps/icu-small`.

> *:warning: Do not modify any source code in `deps/icu-small` ! See section below about floating patches to ICU.*

- Now, rebuild the Node.js license.

```
# clean up - remove deps/icu
make clean
tools/license-builder.sh
```

- Update the URL and hash for the full ICU file in `tools/icu/current_ver.dep`. It should match the ICU URL used in the first step. When this is done, the following should build with small ICU.

```
# clean up
rm -rf out deps/icu deps/icu4c*
./configure --with-intl=small-icu --download=all
make
make test-ci
```

- Commit the change to the `deps/icu-small`, `tools/icu/current_ver.dep` and `LICENSE` files.

Floating patches to ICU

Floating patches are applied at `configure` time. The "patch" files are used instead of the original source files. The patch files are complete `.cpp` files replacing the original contents.

Patches are tied to a specific ICU version. They won't apply to a future ICU version. We assume that you filed a bug against ICU and upstreamed the fix, so the patch won't be needed in a later ICU version.

Example

For example, to patch `source/tools/toolutil/pkg_genc.cpp` for ICU version 63:

```
# go to your Node.js source directory
cd <node>

# create the floating patch directory
mkdir -p tools/icu/patches/63

# create the subdirectory for the file(s) to patch:
mkdir -p tools/icu/patches/63/source/tools/toolutil/

# copy the file to patch
cp deps/icu-small/source/tools/toolutil/pkg_genc.cpp \
tools/icu/patches/63/source/tools/toolutil/pkg_genc.cpp

# Make any changes to this file:
(edit tools/icu/patches/63/source/tools/toolutil/pkg_genc.cpp )

# test
make clean && ./configure && make
```

You should see a message such as:

```
INFO: Using floating patch "tools/icu/patches/63/source/tools/toolutil/pkg_genc.cpp"
from "tools/icu"
```

Clean up

Any patches older than the minimum version given in `tools/icu/icu_versions.json` ought to be deleted, because they will never be used.

Why not just modify the ICU source directly?

Especially given the V8 dependencies above, there may be times when a floating patch to ICU is required. Though it seems expedient to simply change a file in `deps/icu-small`, this is not the right approach for the following reasons:

1. **Repeatability.** Given the complexity of merging in a new ICU version, following the steps above in the prior section of this document ought to be repeatable without concern for overriding a patch.

2. **Verifiability.** Given the number of files modified in an ICU PR, a floating patch could easily be missed or dropped altogether next time something is landed.

3. **Compatibility.** There are a number of ways that ICU can be loaded into Node.js (see the top level README.md). Only modifying `icu-small` would cause the patch not to be landed in case the user specifies the ICU source code another way.