Updating the version of TypeScript

This document provides an overview of how to update the version of TypeScript that is built into the Deno CLI.

Pre-flight checklist

- [ ] git and a local clone of this repository
- [ ] An npm package manager installed and available

Background information

The TypeScript engine (`/cli/tsc/00_typescript.js`) provides type checking and transformation services for the Deno CLI.

During the build (`/cli/build.rs`) the TypeScript engine, and integration code (`/cli/tsc/99_main_compiler.js`) are loaded into a V8 isolate, which in turn loads several of the baseline type libraries (`/cli/dts/*.d.ts`) as well as type libraries provided by extensions (`/ext/*`). This isolate is then snapshotted and included in the build of the CLI. This becomes the *COMPILER* snapshot.

There are a few type libraries that are not loaded into the isolate, as they are not "default" libraries at runtime, but we want to make them available to be loaded dynamically for a user. They are built into the binary as text resources during the build process.

Because Deno often supports web standards in advance of TypeScript supporting them, we find ourselves in the undesirable situation of having to "patch" the published type libraries included with TypeScript. Also, to allow TypeScript to access any type libraries that aren't included with the distribution, we have to patch the list in the TypeScript, so they are available to users.

While in theory a lot of this process could be automated, things can and do change in both Deno and TypeScript, and often you don't know that they have changed until you try to go through the process. Because updates to TypeScript are relatively infrequent, and the process requires a decent amount of contextual awareness, it is the author's opinion that it is best to spend the 10-15 minutes updating by hand to ensure that everything works as expected.

Updating

1. If not present, create an empty npm project outside of the source tree of Deno. (e.g. `ts` , which will be used in this example).
2. Install the version of TypeScript you want to update to inside the empty project. (e.g. `npm i typescript@latest`).
    - TypeScript usually updates the `beta` and `rc` release tags as well, but you will want to check that, to make sure you are getting the version you expect.
3. Copy `ts/node_modules/typescript/lib/typescript.js` to `deno/cli/tsc/00_typescript.js` .
4. Copy `ts/node_modules/typescript/lib/*.d.ts` to `deno/cli/dts/` .
5. Delete `deno/cli/dts/protocol.d.ts` , `deno/cli/dts/tsserverlibrary.d.ts` , `deno/cli/dts/typescriptServices.d.ts` as they are not included in the CLI in any fashion.

6. Analyze the diff to `deno/cli/dts`:

    - A lot of the diff will be line termination differences which git will normalize when committed.
    - Notice any `lib.es*.d.ts` files that have been added. Yearly, TypeScript will usually add the new ES target in the first release post the new year. Also, as standards reach Stage 3, they usually get added to `lib.esnext.*.d.ts` files.
    - Remove any "dead" lib files. Currently TypeScript includes `lib.esnext.promise.d.ts`, `lib.esnext.string.d.ts`, `lib.esnext.weakref.d.ts` which contain definitions that were ratified and are now included in the other libs and aren't even referenced by `lib.esnext.d.ts` or within the TypeScript compiler. 🤦
    - Revert/merge any changes to lib files that provide forward support. This is the "hardest" thing, because you need to assess if the updated version of TypeScript now includes the type definitions that we forward support. Currently there are three:
        - `lib.es2021.intl.d.ts` contains additional `Intl` APIs that were ratified and included, but for some reason never added to the TypeScript libs. PR https://github.com/microsoft/TypeScript/pull/47254 has been sitting there for 15 months without being merged for some reason. 🤦 You will likely need to revert the deletion of this code from `lib.es2021.intl.d.ts`.
        - `lib.esnext.array.d.ts` contains additional array APIs. These likely will be moved to ES2022 at some point, but currently only the `Array.prototype.at` has been added. You will likely need to revert the deletion of the lib from `lib.esnext.d.ts`.
        - We add `lib.dom.asynciterables.d.ts` because for some reason TypeScript has not built these into the libraries. (See: https://github.com/microsoft/TypeScript/issues/29867)

7. Based on the changes to the lib files, you will need to edit the map of lib names to files in the TypeScript compiler (`deno/cli/tsc/00_typescript.js`). The map is currently named `libEntries` and will look a bit like this:

    ```
    var libEntries = [
      // JavaScript only
      ["es5", "lib.es5.d.ts"],
      ["es6", "lib.es2015.d.ts"],
      ["es2015", "lib.es2015.d.ts"],
      ["es7", "lib.es2016.d.ts"],
      // ...
    ];
    ```

    You will want to make sure this matches what is on disk. TypeScript often maps `esnext.*` values to the ratified version of them to ensure they are less "breaking" so you will want to make sure, like for `esnext.array` that it points at `lib.esnext.array.d.ts`. You will also want to revert the deletion of `dom.asynciterables`.

8. For any new lib files that were added, but not included in the snapshot (e.g. `lib.es####.full.d.ts`) add them to `STATIC_ASSETS` in `deno/cli/tsc.rs`.

9. For any new lib files that will be loaded as part of the snapshot, (e.g. `lib.es####.d.ts`) add them to `libs` in `deno/cli/build.rs`.
10. For any APIs that we might have had forward support for but now are part of the libs, edit `deno/cli/tests/unit/esnext_test.ts` to remove the tests, so we are only testing our modifications.
11. Do a build and test for the changes. Specifically `esnext_test.ts` should pass and is an indicator that something isn't right.
12. Commit your changes and submit the PR.
13. Pat yourself on the back for a job well done.