Pre-transpiled Atom packages

Status

Proposed

Summary

This feature will enable package authors to use conventional npm tooling and package.json conventions to take advantage of JavaScript transpilers like Babel or TypeScript.

Motivation

Transpiling packages on *publish* rather than *load* will have great benefits for package authors:

- Standard `npm` tooling like `prepare` scripts will work for apm packages exactly as they work for npm packages. This will remove the need for custom transpiler pipeline modules like atom-babel6-transpiler or atom-typescript-transpiler with their own, independent documentation, configuration and setup.
- Packages can move transpiler-related dependencies to `devDependencies` and trim installation bloat substantially. (as a data point, the TypeScript compiler is 30MB.)
- First-time package load will no longer take a hit from transpiling all of the source into the cache.

Explanation

Package publishing

During the `apm publish` call, apm will invoke `npm pack` to run all standard npm lifecycle hooks and prepare a `.tar.gz` file. apm then uploads the `.tar.gz` file to atom.io, which uploads it to an S3 bucket.

The `npm version` call will still be skipped if the `--tag` is provided, so manual publishing with `apm publish --tag` will still work as it does today.

Package installation

When a user installs a package from atom.io, atom.io first checks to see if it has a precompiled tarball in its S3 bucket. If one is found, the artifact's public URL is returned as the `dist` field in the API response. Otherwise, the existing logic is used to return the GitHub tag tarball URL that's returned now.

Drawbacks

Doing this makes installing a package in production more different than loading it during development. This increases the number of variables that can cause issues between local development and the production of an `apm publish` artifact, like tweaking your `.npmignore` file properly.

Rationale and alternatives

Alternative: publish packages to Actual Npm.org. We could identify Atom packages in the npm registry by the `engine` field we already use, which should keep regular npm from installing it by mistake. The downsides here are:

- It becomes harder to search for *just* Atom packages; we'd have to hack npm search a bit.
- "Starring" would likely break.

- The transition path for existing users of apm and atom.io is not as smooth.
- Easier to typo `apm` and `npm` commands and have an undesirable outcome.

Unresolved questions

Do we want to deprecate transpilation-on-demand for local development, as well? It may add a bit of friction for package development, but transpilers like TypeScript tend to offer a `--watch` option to transpile live, and it would let us eliminate a lot of complexity in the way Atom loads JavaScript.