# Consolidate Core Atom Packages

## Status

Accepted

## Summary

Atom's official distribution is comprised of 92 core packages which provide its built-in functionality. These packages currently live in their own independent repositories in the Atom organization, all with their own separate issues, PRs, releases, and CI configurations. This RFC proposes that by consolidating most, if not all, of these core packages back into the `atom/atom` repo, we will see the following benefits:

- Less confusion for new contributors
- Simpler core package contribution experience
- Greatly reduced burden for maintainers

## Motivation

Let's cover each of the bullet points mentioned above:

### Less confusion for contributors

Imagine that a new contributor wants to add a small new feature to the `tree-view` package. The first place they are likely to look is the `atom/atom` repository. Scanning through the folders will lead to a dead end as nothing that looks like `tree-view` code can be found. They might take one of the following steps next:

- By reading README.md, maybe they will decide to click the link to the Atom Flight Manual and *maybe* find the Contributing to Official Atom Packages page there
- They could read the CONTRIBUTING.md file which has a section that explains where to find the repos for core packages and how to contribute, but we don't really have a clear pointer to that in our README.md
- If they don't happen to find that page, they might use Google to search for "atom tree view" and find the atom/tree-view repo and *maybe* read the CONTRIBUTING.md file which sends them to Atom's overall contribution documentation
- They might go to the Atom Forum or Slack community to ask how to contribute to a particular part of Atom and *hopefully* get a helpful response that points them in the right direction

Having all of the core Atom packages represented in a top-level `packages` folder, even if they all don't actually live in the repo, will go a long way to making the core package code more discoverable.

**Simpler core package contribution experience**

Separating core Atom features out into individual repositories and delivering them to Atom builds via `apm` is a great idea in theory because it validates the Atom package ecosystem and gives developers many examples of how to develop an Atom package. It also gives Atom developers real-world experience working with Atom's APIs so that we ensure community package authors have the same hackability that Atom developers enjoy.

On the other hand, having these packages live in separate repositories and released "independently" introduces a great deal of overhead when adding new features. Here is a comparison of the current package development workflow contrasted to what we could achieve with consolidated packages:

**Current Package Development Workflow** For example, to add a single feature to the `tree-view` package, one must:

1. Fork and clone the `tree-view` repository to their computer (making sure to pull the commit relevant to the version of Atom they are working with)
2. Run `apm install` and `apm link` inside of the repo folder
3. Make their desired changes to the code
4. Open a PR to the `tree-view` repo and wait for CI to pass and a maintainer to review it
5. Work with maintainers to get the PR approved and merged

After this is finished, an Atom maintainer must take the following steps:

1. Clone the `tree-view` repo
2. Run `apm publish` to publish a new release of the package
3. Edit `package.json` in the Atom repo to reflect the new version of `tree-view`
4. Commit and push the changes to the relevant branch where the change belongs (`master` or `1.nn-releases`)

**Simplified Package Development** If we were to move `tree-view` (or any other core Atom package) back into `atom/atom`, the development workflow would look more like this:

1. Fork and clone `atom/atom` and switch to a release branch if necessary
2. Build Atom and launch it in dev mode
3. Make desired changes to the code in `packages/tree-view`
4. Open a PR on `atom/atom` and wait for CI to pass and a maintainer to review it
5. Work with maintainers to get the PR approved and merged

At this point, the change is merged into Atom and ready for inclusion in the next release.

**Greatly reduced burden for maintainers**

Since packages all have their own repositories, this means that we have to watch 91 different repos for issues and pull requests. This also means that we have to redirect issues filed on `atom/atom` to the appropriate repository when a user doesn't know where it belongs. Even more importantly, there's not an easy way to prioritize and track issues across the Atom organization without using GitHub Projects.

Also, as mentioned above, there's the added duty of doing the package "version dance" when we merge any new PRs to a package repository: publish the package update, update `package.json` in Atom. It's very easy to forget to do this and not have community contributions included in the next Atom release!

The more core packages live in `atom/atom`, the less work Atom maintainers have to do overall.

## Explanation

Many of Atom's core packages now live in the core `atom/atom` repository. To the Atom user, this change will be imperceptible as these packages still show up in the list of Core Packages in the Settings View. Users can still optionally disable these packages.

For maintainers and contributors, there will be less juggling of repositories and no more publishing of updates to these packages with `apm`:

Contributors now clone and build `atom/atom` to work on improvements to core packages. They will no longer have to use `apm link` in dev mode to test changes they make to packages in the repo's `packages` folder. Core packages that aren't consolidated still have folders under `packages` with README.md files that point to the home repository for that package.

When a contributor sends a PR to `atom/atom` that only affects files in a folder under `packages`, only the specs for the relevant package folders will be executed using Atom's CI scripts. This means that a full Atom build will not be required when no Atom Core code is changed in a PR. Package specs are also now run against all 3 OSes on Atom `master` and release builds.

Atom maintainers no longer have to publish new versions to consolidated core packages and then edit `package.json` to bump the package version in a particular Atom release branch (Stable, Beta, or `master`). When a PR against a consolidated core package in `atom/atom` is merged, no version number change is required and the changes will immediately be a part of the next release from that branch.

## Drawbacks

One possible drawback of this approach is that there might be some initial confusion where core Atom packages live, especially if some are consolidated into `atom/atom` and others still live in their own repositories. We will manage this confusion by doing the following:

- Include a `README.md` file in the `packages` folder which lists core packages that are not consolidated in the Atom repo. This will enable users to find the home repositories of those packages.
- Archive the repositories for consolidated core packages, but only after migrating existing issues, merging or closing existing PRs, and updating the README.md to point to the new home of the package code.

Also, contributors will now have to fork, clone, and build `atom/atom` to contribute to core packages where they would previously just need to clone the package repository. This might put added burden on them such as installing necessary build dependencies on their machine that they wouldn't otherwise need. It is very likely we could simplify this process for them, though.

One final drawback is that it will now be harder to have single-package maintainers. We currently have 7 core packages where there is a maintainer who isn't a part of the core Atom maintainers team. These maintainers generally are able to merge community PRs and make commits to those packages with their own judgement. If we get rid of individual package repositories, do we now make those maintainers full Atom maintainers?

## Rationale and alternatives

The Motivation section explains most of the rationale, so this section will focus on the process of consolidating packages back into `atom/atom`. The set of packages we've chosen to consolidate were evaluated based on a few factors:

- Number of open issues and PRs (exclude any with $> 10$ open PRs)
- Time since last update (longer duration since last update is prioritized)
- Number of package-only maintainers on the repo (exclude any with package maintainers for now)

Using this criteria, all 91 packages have been evaluated and categorized to determine whether they are good candidates for consolidation:

**Initial Consolidation Candidates**

| Package | Open Issues | Open PRs | Outside Maintainers | Last Updated |
|---|---|---|---|---|
| **about** | 2 | 0 | 0 | 7/11/18 |
| **archive-view** | 10 | 0 | 0 | 6/3/18 |

| Package | Open Issues | Open PRs | Outside Maintainers | Last Updated |
|---|---|---|---|---|
| **atom-dark-syntax** | 5 | 0 | 0 | 12/6/17 |
| **atom-dark-ui** | 1 | 2 | 0 | 2/13/18 |
| **atom-light-syntax** | 1 | 0 | 0 | 10/17/16 |
| **atom-light-ui** | 1 | 0 | 0 | 2/13/18 |
| **autoflow** | 17 | 4 | 0 | 4/17/18 |
| **autosave** | 13 | 0 | 0 | 9/16/17 |
| **background-tips** | 3 | 2 | 0 | 2/17/18 |
| **base16-tomorrow-dark-theme** | 5 | 0 | 0 | 1/10/17 |
| **base16-tomorrow-light-theme** | 1 | 0 | 0 | 1/10/17 |
| **bookmarks** | 19 | 4 | 0 | 12/10/17 |
| **bracket-matcher** | 74 | 8 | 0 | 3/20/18 |
| **command-palette** | 18 | 6 | 0 | 2/27/18 |
| **dalek** | 2 | 0 | 0 | 2/28/18 |
| **deprecation-cop** | 5 | 0 | 0 | 9/7/17 |
| **dev-live-reload** | 4 | 0 | 0 | 11/14/17 |
| **encoding-selector** | 11 | 2 | 0 | 4/19/18 |
| **exception-reporting** | 5 | 0 | 0 | 2/6/18 |
| **git-diff** | 38 | 1 | 0 | 1/18/18 |
| **go-to-line** | 5 | 2 | 0 | 1/25/18 |
| **grammar-selector** | 3 | 1 | 0 | 4/12/18 |

| Package | Open Issues | Open PRs | Outside Maintainers | Last Updated |
|---|---|---|---|---|
| **image-view** | 4 | 4 | 0 | 7/9/18 |
| **incompatible-packages** | 1 | 0 | 0 | 4/25/17 |
| **keybinding-resolver** | 11 | 3 | 0 | 7/6/18 |
| **language-clojure** | 13 | 3 | 0 | 1/26/18 |
| **language-coffee-script** | 9 | 2 | 0 | 11/1/17 |
| **language-csharp** | 1 | 1 | 0 | 4/27/18 |
| **language-css** | 6 | 7 | 0 | 6/11/18 |
| **language-gfm** | 52 | 9 | 0 | 6/15/18 |
| **language-git** | 4 | 2 | 0 | 4/18/17 |
| **language-html** | 11 | 4 | 0 | 7/5/18 |
| **language-hyperlink** | 2 | 3 | 0 | 10/25/17 |
| **language-json** | 1 | 0 | 0 | 5/11/18 |
| **language-less** | 5 | 1 | 0 | 6/11/18 |
| **language-make** | 7 | 3 | 0 | 11/26/16 |
| **language-mustache** | 0 | 0 | 0 | 2/5/18 |
| **language-objective-c** | 2 | 0 | 0 | 12/1/15 |
| **language-php** | 25 | 7 | 0 | 6/11/18 |
| **language-property-list** | 1 | 0 | 0 | 3/11/17 |
| **language-python** | 33 | 4 | 0 | 6/18/18 |
| **language-ruby** | 38 | 10 | 0 | 10/25/17 |

| Package | Open Issues | Open PRs | Outside Maintainers | Last Updated |
|---|---|---|---|---|
| **language-ruby-on-rails** | 9 | 6 | 0 | 12/7/17 |
| **language-sass** | 12 | 5 | 0 | 5/2/18 |
| **language-shellscript** | 12 | 3 | 0 | 6/18/18 |
| **language-source** | 0 | 0 | 0 | 1/6/15 |
| **language-sql** | 6 | 4 | 0 | 1/26/18 |
| **language-text** | 1 | 0 | 0 | 3/9/18 |
| **language-todo** | 10 | 6 | 0 | 1/26/18 |
| **language-toml** | 1 | 0 | 0 | 1/6/18 |
| **language-typescript** | 6 | 0 | 0 | 6/18/18 |
| **language-xml** | 2 | 1 | 0 | 6/12/17 |
| **language-yaml** | 8 | 2 | 0 | 3/9/18 |
| **line-ending-selector** | 10 | 0 | 0 | 5/18/18 |
| **link** | 0 | 1 | 0 | 11/14/17 |
| **metrics** | 1 | 2 | 0 | 7/5/18 |
| **notifications** | 29 | 8 | 0 | 3/22/18 |
| **one-dark-syntax** | 4 | 0 | 0 | 5/27/18 |
| **one-dark-ui** | 13 | 1 | 0 | 5/1/18 |
| **one-light-syntax** | 2 | 1 | 0 | 5/27/18 |
| **one-light-ui** | 2 | 0 | 0 | 5/1/18 |
| **open-on-github** | 8 | 3 | 0 | 11/21/17 |

| Package | Open Issues | Open PRs | Outside Maintainers | Last Updated |
|---|---|---|---|---|
| **package-generator** | 10 | 2 | 0 | 11/16/17 |
| **status-bar** | 25 | 3 | 0 | 11/6/17 |
| **styleguide** | 12 | 2 | 0 | 4/12/18 |
| **tabs** | 66 | 7 | 0 | 5/13/18 |
| **timecop** | 5 | 0 | 0 | 11/4/17 |
| **update-package-dependencies** | 0 | 0 | 0 | 12/10/17 |
| **welcome** | 0 | 0 | 0 | 11/21/17 |
| **whitespace** | 31 | 6 | 0 | 5/30/18 |
| **wrap-guide** | 3 | 4 | 0 | 11/27/17 |

**Packages to be Consolidated Later** The following packages will not be consolidated until the stated reasons can be resolved or we decide on a consolidation strategy for them:

| Package | Open Issues | Open PRs | Outside Maintainers | Last Updated | Reason |
|---|---|---|---|---|---|
| **find-and-replace** | 219 | 17 | 0 | 6/4/18 | Too many open PRs |
| **fuzzy-finder** | 89 | 22 | 0 | 5/17/18 | Too many open PRs |
| **github** | | | | | Independent project |
| **language-c** | 53 | 15 | 0 | 7/10/18 | Too many open PRs |
| **language-go** | 12 | 2 | **1** | 6/18/18 | Package maintainer, possibly inactive? |

| Package | Open Issues | Open PRs | Outside Maintainers | Last Updated | Reason |
|---|---|---|---|---|---|
| **language-java** | 8 | 2 | **1** | 6/11/18 | Package maintainer |
| **language-javascript** | 66 | 12 | 0 | 7/6/18 | Too many open PRs |
| **language-perl** | 17 | 1 | **1** | 10/30/17 | Package maintainer, possibly inactive? |
| **markdown-preview** | 139 | 12 | 0 | 1/8/18 | Too many open PRs |
| **settings-view** | 137 | 18 | 0 | 5/17/18 | Too many open PRs |
| **snippets** | 57 | 4 | **1** | 4/17/18 | Package maintainer |
| **solarized-dark-syntax** | 8 | 3 | **1** | 5/27/18 | Package maintainer |
| **solarized-light-syntax** | 2 | 3 | **1** | 5/27/18 | Package maintainer |
| **spell-check** | 68 | 14 | **1** | 5/25/18 | Too many open PRs, package maintainer |
| **symbols-view** | 86 | 13 | 0 | 12/10/17 | Too many open PRs |

| Package | Open Issues | Open PRs | Outside Maintainers | Last Updated | Reason |
|---|---|---|---|---|---|
| **tree-view** | 210 | 36 | 0 | 3/21/18 | Too many open PRs |

**Packages to Never Consolidate** These packages will not be consolidated because they would inhibit contributions from our friends in the Nuclide team at Facebook:

- **autocomplete-atom-api**
- **autocomplete-css**
- **autocomplete-html**
- **autocomplete-plus**
- **autocomplete-snippets**

### Consolidation Process

To consolidate a single core package repository back into `atom/atom`, the following steps will be taken:

1. All open pull requests on the package's repository must either be closed or merged before consolidation can proceed
2. The package repository's code in `master` will be copied over to Atom's `packages` folder in a subfolder bearing that package's name.
3. Atom's `package.json` file will be updated to change the package's `packageDependencies` entry to reference its local path with the following syntax: `"tree-view": "file:./packages/tree-view"`
4. A test build will be created locally to manually verify that the package loads and works correctly at first glance
5. The package specs for the newly-consolidated package will be run against the local Atom build
6. A PR will be sent to `atom/atom` to verify that CI passes with the introduction of the consolidated package
7. Once CI is clean and the PR is approved, the PR will be merged
8. The package's original repository will have all of its existing issues moved over to `atom/atom` using a bulk issue mover tool, assigning a label to those issues relative to the package name, like `packages/tree-view`
9. The package's original repository will have its README.md updated to point contributors to the code's new home in `atom/atom`
10. The package's original repository will now be archived on GitHub

### Alternative Approaches

One alternative approach would be to break this core Atom functionality out of packages and put it directly in the Atom codebase without treating them as packages. This would simplify the development process even further but with the following drawbacks:

- The Atom team would have less regular exposure to Atom package development
- Users would no longer be able to disable core packages to replace their behavior with other packages (different tree views, etc)

## Unresolved questions

- Is there a good reason to not move the `language-*` packages into `atom/atom`?

  One concern here is that there exist projects which depend directly on these repositories for the TextMate syntax grammars they contain. Moving the code into `atom/atom` would require that we notify the consumers of the grammars so that they can redirect their requests to the `atom/atom` repo.

- Should we use `git subtree` to migrate the entire commit history of these packages over or just depend on the history from a package's original repository?

  For now, we won't use `git subtree` due to the possibility that bringing over thousands of commits could cause unknown problems in the Atom repo. We may try this for newly consolidated packages in the future if we decide that not having the package commit history is a sufficient impediment to problem investigations.

- What are the criteria we might use to eventually decide to move larger packages like `tree-view`, `settings-view`, and `find-and-replace` back into `atom/atom`?

- Will we be losing any useful data about these packages if we don't have standalone repositories anymore?

- Should we use this as an opportunity to remove any unnecessary packages from the core Atom distribution?