Updatable Bundled Packages

Status

Proposed

Summary

This feature will enable an opt-in subset of bundled Atom packages to be updated with `apm` outside of the Atom release cycle. This will enable users to receive new functionality and bug fixes for some bundled packages as regularly as needed without waiting for them to be included in a new Atom release. This is especially important for packages like GitHub and Teletype which provide essential Atom functionality and could be improved independently of Atom.

Motivation

Atom currently uses a monthly release cycle with staged Stable and Beta releases so that major issues get caught early in Beta before reaching the Stable release. Because Atom releases updates monthly, this means that a new feature merged into `master` right after a new Atom release could take one month to reach the next Beta and then another month to reach Stable.

Since a large part of Atom's built-in functionality is provided by bundled packages, it makes sense to allow some of those packages to be updated independently of Atom's monthly release cycle so that users can receive new features and fixes whenever they become available.

Bundled packages are treated differently than community packages that you can install using `apm`:

- You are not prompted to update them when new versions are released on `apm`
- `apm` will warn you at the command line when you try to install or update a bundled package
- If a user intentionally installs a bundled package from `apm` the dalek package will show a warning in the "deprecations" view asking the user to remove the offending package

Despite all this, if the user *does* manually install an update to a bundled package using `apm`, it will be loaded into the editor and updated dutifully as new releases occur. The only new functionality needed is to enable `apm` to check bundled packages for updates when those packages haven't yet been installed in the user's `~/.atom/packages` folder.

The primary use case for this improvement is enabling the GitHub package to ship improvements more frequently than Atom's release cycle since many of its improvements can be done without changes to Atom itself. If this approach is proven to work well for the GitHub package, we might also consider using it to ship Teletype as a bundled Atom package.

Explanation

Any bundled Atom package can opt in to new updates released via `apm` by adding `"coreUpdatable": true` to its `package.json` file. This causes `apm` to consider it as part of the list of packages it checks for updates. If a community (non-bundled) package sets this field to `true` or `false` it will be ignored as it's only relevant to bundled packages.

Atom shows update notifications for Updatable bundled packages whenever they are available so long as those updates support the engine version of the current Atom build. Bundled package updates can also be found and installed in the Settings view's *Updates* tab.

The `dalek` package is aware of the new "Updatable" metadata and excludes updated bundled packages from its deprecation warnings.

User Experience Examples

1. The user downloads and installs Atom 1.28.0 which includes GitHub package version 0.15.0. Two weeks later, GitHub package 0.16.0 is released with a few new features. The user is prompted to update to the new version and gets the new features even though Atom 1.29.0 hasn't been released yet.
2. The user downloads and installs Atom 1.28.0, including GitHub package 0.15.0, which was released two weeks prior. Since that release the GitHub package has been updated to version 0.15.1 on `apm`. When the user starts Atom for the first time they are prompted to update the GitHub package.
3. In the future, a user has an old install of Atom 1.28.0 and waits a long time between installing Atom updates. The GitHub package releases version 0.25.0 but the user is not prompted to install it because the GitHub package has set `engines` in `package.json` to restrict to Atom 1.32.0 and above.

Rules for Updatable Bundled Packages

Any package that opts into this behavior must adhere to these rules:

1. **Each release must ensure that its `engines` field in `package.json` reflects the necessary Atom version for the Atom, Electron, and Node.js APIs used in the package**. This field defines the range of Atom versions in which the package is expected to work. The field should always be set to the lowest possible Atom version that the package supports.
2. **Any new update to a bundled package *must* support current Stable *and* Beta releases**. This enables the user to upgrade the package

and continue to use it in side-by-side Stable and Beta installs on their machine. If a package wants to use API features of a newer version of Atom while still supporting older Atom versions, it must do so in a way that is aware of the user's version and adjust itself accordingly.

3. **Atom's `package.json` *must* stay up to date with the latest supported version of the package** in the `master` and Beta release branches. This ensures that the user always gets the latest version of the package in a new release and also benefits from its inclusion in Atom's snapshot.

For rule #3, it will be important to have automation to ensure that current Beta release and `master` are kept up to date with the latest compatible version of any updatable bundled package as it will be difficult for maintainers to do that manually. This could be accomplished by a nightly CI run which is focused explicitly on bumping package dependencies in this manner.

Drawbacks

Possible API incompatibility

The primary drawback of this approach is that Updatable bundled packages might exhibit problems on older Atom versions due to missing or changed APIs in Atom, Electron, or Node.js. The solution for these packages is to keep their `engines` field updated appropriately, but there's still a chance that some updates will slip through without the necessary engine version changes. If this does occur and users are affected by it, the solution is to publish a new update which rolls back the package to the functionality of its previous release and then publish another new update with the new functionality restored and the proper `engines` version in place.

Increased Atom startup time

Another major drawback is that the snapshotted code for the bundled package will no longer be used since a newer version has been installed. This updated version of the package cannot be easily added back into Atom's snapshot so it could cause a noticeable drag on Atom's startup time. Some quick measurements with Timecop show a 10x increase in GitHub package load time for bundled (snapshot) vs updated (non-snapshot) package code:

GitHub Package Code	Load Time
Bundled	52 ms
Updated (first load)	5026 ms
Updated (subsequent loads)	591 ms

There was no measurable effect on shell or window startup time, only package load time. It seems that the transpilation phase of the first load of the package incurs a 100x increase in load time. Pre-transpilation of the package code (either when shipped or when installed using `apm`) will be useful in mitigating this cost. Further investigation into snapshotting package code will be needed to understand if the load time increase can be mitigated.

There is a possibility that the GitHub package could load parts of its codebase on demand to mitigate the increased startup time when not loaded as part of Atom's snapshot. This approach is discussed in more detail at atom/github#1522.

Incompatibility across Atom release channels

One other possible drawback is that an updated version of a bundled package might not be compatible across two different Atom channels. For example, if the user installs a new update to a bundled package that only supports the current Atom Beta release or higher, the user will no longer have access to that package if they open Atom Stable. However, this drawback is no different than what the user would face today installing a community package under the same circumstances, so this could be considered a general problem in the Atom package ecosystem.

Finally, one risk of this approach is that the Atom team forgets to update a bundled package to its latest appropriate version on `apm` just before a new release. If this happens, the user will install a new Atom update and then be prompted to update a package that should have been snapshotted and shipped in-box. To avoid this problem we could add some build automation that checks for the latest version of a bundled package to see if the current Atom build would be supported by it.

Rationale and alternatives

This is the best approach for updating bundled packages because it allows those packages to take control of their own release cycle so long as they manage their Atom engine version correctly. It also does so in a way that allows us to decide which packages can be updated independently, reducing the likelihood of problems for users.

The primary alternative to this approach is to speed up the Atom release cycle so that bundled Atom package updates will reach users more frequently. This approach will be investigated independently of this RFC as it may still be valuable even with Updatable bundled packages.

Unresolved questions

- What unresolved questions do you expect to resolve through the RFC process before this gets merged?

Is it enough to just depend on the `engines` field of `package.json` to protect users from installing a package update that doesn't work with their version of

Atom?

- What unresolved questions do you expect to resolve through the implementation of this feature before it is released in a new version of Atom?

Is there any optimization we can use to reduce the performance hit of loading updated bundled packages?

- What related issues do you consider out of scope for this RFC that could be addressed in the future independently of the solution that comes out of this RFC?

One issue that's out of scope for this RFC is how we ship new features and fixes to the core components of Atom (not its bundled packages) more frequently. There are two options we can investigate to accomplish this:

- **Ship Atom updates more frequently, possibly every two weeks**
- **Introduce a channel for nightly builds which surface the latest changes every day**

Both of these possibilities will be covered in future RFCs as they could be implemented independently of the feature described in this RFC.