# Title: Translations - ember-i18n

At FreshSales, we support quite a few languages to ensure we provide a wide range of support for different regions.

Built with Rails on the backend and Ember on the frontend, we tend to have multiple sets of language files which will turn out to be tedious to maintain as days progress. In the Rails world, translations are maintained in the folder path as *config/locales/<lang>.yml* which is a rails convention.

For the frontend, we use [ember-i18n](https://github.com/jamesarosen/ember-i18n) for translating text to render in the handlebars and javascript. Ember-i18n requires you to maintain these locales inside *app/locales* path. It becomes even trickier to maintain a set of Rails-based translation files and a separate set for Ember. To ensure we have a single source of truth w.r.t translations, we wrote a middleman add-on which transpires our Rails-based .yml config to .js based frontend locales. ([Refer ember-yaml-to-json](https://www.npmjs.com/package/ember-i18n-yaml-to-json)).

Now that js files are compiled, it is ready to use them for frontend. So configuring *yamlToJson* engine to dispatch the js files to *app/locales* will solve the problem. On the flip side, anything inside *app/locales* is merged into your ember app *app.js* which tends to grow as days progress and as translations increases in number. At a point, our frontend grew to 4.xMB uncompressed and un-gzipped. This will make our apps slow when more translations are brought in.

That’s when we moved the *app/locales* based config to *assets/locales* and ensured fingerprinting is handled by the [broccoli-asset-rev](https://www.npmjs.com/package/broccoli-asset-rev) package. Now that your translations are fingerprinted based on md5 hash of the file contents, your post-deployment js fingerprint hash has no issues of app loading slowness. But that’s not the end, there is another problem associated with this. Since the fingerprinted translations are not inside your project app.js you need to manually fetch the file based on the locale. This can be done in 2 ways, either by appending the translations to index.html as a dynamic path where the locale is set from server-side (if your Rails app serves your app’s index.html) or as a lazy-loaded JS injection. We preferred the second approach as we didn’t want to couple our backend and frontend closely. But since the path is going to be dynamic, broccoli-asset-rev does not contain the appropriate hash for the asset it just fingerprinted. Yet another headache to the already persisting problem.

One drawback with the broccoli-asset-rev is that there is no post-processed hook after a file is finger-printed. Had there been an option, we can simply fetch the hash and use it in our *config/environment.js* and use that for lazy loading.

To overcome this problem, we had another plan. Since all our translations were moved to *assets/locales* as a parent, we asked broccoli-asset-rev to ignore the fingerprinting for those files inside this path and wrote our own [*customHash*](https://github.com/freshdesk/freshsales/blob/0b2a8c4a7b88df26a40f2227d03b907b4a5f4089/frontend/config/production.js#L26) method. This meant we wrote our own fingerprint for *assets/locales/<lang>/translations.js* and for everything else, we use the default fingerprinting done by broccoli-asset-rev. Since this *customHash* is under our control, we use it in our *beforeModel()* to fetch the translations dynamically. You can simply write a function inside the compiled translations.js file to return the I18nised hash. That way you don’t tend to bloat the window namespace as well as ensure translations work lazily.

Having multiple sources for translations can be time-consuming for your dev team as well as when you increase the support for many more languages. Hopefully this technique lets you keep your asset pipeline in check without giving up on the actual advantages post-deployment.