Translation Bundles

aka message bundles, language bundles, localization bundles, language modules, ...

What is it ?

Translation bundles are Magnolia Modules consisting of, and only of, translation files for several other Magnolia modules, in a given language. Each language has its own translation bundle.

Why ?

Historically, translation files for every language where bundled with the module that used them. This led to several issues: translations were incomplete, inconsistent, and outdated. We have now extracted them into a module-per-language structure, with the following goals:

- **decouple the lifecycle of translations from that of Magnolia and its modules.** This means we can release modules faster, and delay the translations. This also means, and perhaps more importantly, we can incrementally improve the translations without depending on the module being released.
- **provide an easier workflow** for both translators and developers.* In the previous situation, it is difficult for translators to locate the files to be translated. This often means developers have to get involved to provide the files and re-integrate them after translations. It is now simpler to provide full access to a given translation bundle Git repository to a translator.

How does it work ?

Each translation bundle contains translations for all modules of **a given language**.

Each translation bundle is maintained and **released** individually, so that translations can happen asynchronously to Magnolia and its modules' lifecycle, as well as other languages'.

Each translation bundle **is a Magnolia module**, and has a module descriptor. This allows translators to indicate which modules, and optionally which versions of the modules, are translated.

English

Currently, we are treating English as our "master" language. This means english translation files will remain bundled with their respective modules.

However, this might change in the future. We could decide to also extract english files, so as to provide them with the same flexibility. For this to happen, we'll need to figure out some form of tooling or file format to define the "master" keys. (This could probably be generated, in part thanks to the I18nKeyGenerator mechanism.

Translation processes

Adding keys

When adding translation items in a module, translations will ideally need to be "triggered". We still don't have a defined workflow for this, so the process (or lack thereof) is the same. Idea: similar to "needs release notes", we could have a checkbox in Jira for "needs translations" for those issues that add language items. However, since translation must happen in several languages asynchronously, this is probably not the end of it.

Removing keys ?

Ideally, we won't be removing keys too often. Unused keys will just be... unused. But sometimes, we'll want to clean up. When that has to happen, we bump dependencies' versions (e.g "this translation bundle requires foo-bar 2.0 at minimum", which where said key was removed/unused). New versions of this translation bundle won't "compatible" with older versions of some modules – so older modules will be "stuck" without improved translation. I think that's a trade-off we can live with, rather than maintaining various branches of translation bundles.

Key fallbacks

Some languages will need to override certain specific keys, while some won't (e.g depending on the context of an app or dialog, some wording might change in some languages, but not in others). Tooling will need to help with this. (TODO)

Technical implementation

All translation bundles are on our Git repository at https://git.magnolia-cms.com/lang/lang-<locale>.git

There is a "meta" project at https://git.magnolia-cms.com/lang/lang-reactor.git, which is essentially a Maven reactor and a collection of Git submodules. This project is **never released**, and is only meant to continuous integration purposes.

Jenkins setup

There is a single Jenkins job. It uses a trigger on the Git repo of `lang-reactor`, but the magic is in Git hooks. We've set up hooks on every lang-* repo, which trigger the `lang-reactor` Jenkins job whenever one of the language bundles is modified.

The Jenkins job, in turn, is configured to update all submodules at every build (so there is no need to maintain the tip of the submodules in the `lang-reactor.git` repo.)

It is currently setup to do `git submodule foreach git pull origin master` as a pre-build task, although the Git plugin should be able to do it; the hooks are also necessary because Jenkins currently doesn't poll the submodules. (which is perhaps the same issue)

Jira

There is a single Jira project, and one component per language.

To add a language

- Create the appropriate Git repository:

```
NEW_LOCALE=<new locale>
git clone https://git.magnolia-cms.com/lang/lang-$NEW_LOCALE.git
echo "" | ssh git@git.magnolia-cms.com setdesc lang/lang-$NEW_LOCALE # This sets a description file for the
repo, not strictly necessary, but good practice for self-created repo. leaving it empty is fine
```

- Copy the .gitignore of an existing translation bundle
- Copy and adapt the pom.xml of an existing translation bundle (<artifactId>, <name>, <scm>)
- Copy and adapt the module descriptor xml of an existing translation bundle (<name>, <dependencies>)
- For now, we don't need *Maven* dependencies to the modules we're translating.
- Add translation files under `src/main/resources/mgnl-i18n/`
- Finally, add the submodule in the `lang-reactor`:

```
git clone https://git.magnolia-cms.com/lang/lang-reactor.git
git submodule add ssh://git@git.magnolia-cms.com/lang/lang-$NEW_LOCALE # we use ssh urls for Jenkins'
sake
vi pom.xml # Add it to the <modules> section
git add -p; git commit; git push
```

Random Tips

Use aspell to check spelling

```
sudo port install aspell
sudo port install aspell-dict-$NEW_LOCALE
# or brew install aspell

for f in $(find . -name *$NEW_LOCALE.properties); do
  echo "~~~~~~~~~ $f ~~~~~~~~"
  cat $f | sed -E 's;.=(.);\1;' | grep -vE '^(#|$)' | aspell -a -l $NEW_LOCALE --dont-suggest
done
```

TODOs

- Do we need an MVH (to setup/check /server/i18n/system/languages) ?
    - we could perhaps start shipping default config with just english in there, and each module would add its own at install.
    - If so, where do we put it ? Just depend on core... or have it in core ? (kinds of put a hard dependency on core version tho)
    - Or another module that has optional deps on the lang modules (so it is installed AFTER all langs, but that wouldn't kick in if someone ADDS a lang, unless it does so at startup time rather than update-time)
- process(es) for transla-tions/-tors
- tools to visualize completeness of translations
    - since we have the key generation mechanism, we probably want tooling to verify translations based on that (e.g for example certain languages will need to override certain specific keys, while some won't – tooling should allow to see translation completion regardless)
    - when adding/remove keys, workflows should be triggered (human or machine) to get translations done for those keys.