Text Translation Toolkit (T)

A Portable JavaScript Localization Infrastructure

Joel Sahleen, Globalization Architect

Background

Domo's mission is to transform the way business is managed by connecting business decision makers across an organization with the people and data they need to improve business results. Whether it's the CEO, the frontline worker or the data analyst, we make relevant data easily accessible, consumable and actionable to everyone, anywhere.

An Old Problem — Text Translation

- Platform with dozens of components, services and interfaces
- Multiple technologies, i18n frameworks and resource formats
- Short development, testing and release cycles (2 weeks)
- Distributed repositories with multiple branches and build systems
- No centralized localization infrastructure or automation
- Frequent ad-hoc localization builds
- Need to support complex message format syntax
- Need to exchange resources with third-party translation platforms
- Small globalization team with limited resources
- More important things to do than manage resource files all day

Our Solution

The Text Translation Toolkit (T)

A portable, drop-in, localization infrastructure, written in JavaScript, that takes (some of) the pain out of managing localizable application resources and getting them translated into other languages.

Features

- Component-based, extensible plug-in architecture
- Locale aware find, read, copy, move and remove files in batches
- Supports multiple resource file formats (and converts between them)
- Provides monolingual and bilingual XLIFF export and import
- Integrates with third-party translation APIs
- Supports pseudo-localization (example characters and expansion)
- Supports ICU message formatting validation using messageformat.js
- Provides metrics for resource file content changes across languages
- Easy enough for developers to manage their own localization processes
- Scriptable CLI commands that can be used in automation

A Typical Workflow

1. Install needed components as dev dependencies using npm
2. Create T.json configuration file with source, target and pseudo locales
3. Define file bundles using localizable read/write glob patterns
4. Optionally enable git branch tracking in exported file/bundle names
5. Read files and export to XLIFF (deliver manually)
6. Push files/XLIFFs to translation API and request translation
7. Pull translated files/XLIFFs back into source tree
8. Generate pseudo-localized files for any language
9. Validate message format syntax and correct any errors
10. Generate CSV metrics and import into Domo for visualization

CLI Command Examples

- T find all targets
- T read all source
- T copy all ../data2/{{language}}/{{title}}.json targets
- T read ../data/{{language}}/*.json
- T export all ../{{locale}}/{{bundle}}.{{branch}}.xliff
- T import all ../{{locale}}/{{bundle}}.{{branch}}.xliff
- T push -x all <vendor> source
- T request -x all <vendor> targets
- T pull -x all <vendor> targets
- T validate all mf
- T pseudo all source | T write ../data/{{language}}/{{title}}.json
- T read all | T convert properties | T write ./{{title}}_{{language}}.xml

DOMO

Thank you.