Fluent: Firefox’s new localization system

Jeff Beatty

Jeff Beatty is the head of localization at Mozilla, the makers of the popular open source web browser Firefox. He holds an MS in multilingual computing and localization from the University of Limerick.

Staś Malolepszy

Staś Malolepszy works with hundreds of volunteers around the globe who continue to deliver top quality localization of Firefox in nearly 100 locales to over 400 million users worldwide.
One of the constant challenges in developing global software is reducing technical debt and legacy code. Ruthless prioritization takes place when it becomes clear that an organization needs to replace its legacy code with something more efficient and modern. Very often, legacy code that affects internationalization (i18n) and localization (l10n) is one of the last areas of the codebase to be prioritized in this effort. This is the situation we found ourselves in at Mozilla. Firefox and its rendering engine, Gecko, had become bloated and filled with legacy code that needed refurbishing. Thanks to the Firefox Quantum release in 2017, this is no longer the case. However, part of that legacy codebase was i18n/l10n. In fact, prior to 2018, this part of the codebase hadn’t been altered or updated in nearly 20 years. As a result, Firefox had a number of significant i18n/l10n problems:

- Yellow Screen of Death (YSOD): users were confronted with a YSOD XML parsing error when a translated string was malformed, effectively rendering their browser useless.
- English fallback: if a string was untranslated, it would appear in English, whether the user understood English or not.
- Single-locale builds: users struggled to find Firefox in the right language due to there being over 100 different builds of Firefox to choose from for download and install.
- Source strings had global impact: monolingual developers were expected to craft source language strings in syntax that, while natural-sounding in English, affected all target language translations and produced unnatural-sounding translations.
- No pseudolocalization: as a practice, pseudolocalization was nonexistent. I18n problems were discovered manually, often post-release.
- Multiple string formats in one product: requiring developers and localizers to know how to form correct strings in both .dtd and .properties files for one single product, introduced high onboarding costs and a high risk for errors (which would produce YSOD).
- Long wait time for localization updates: users had to wait for the next version of Firefox (between 6-18 weeks) before localization errors would be corrected.

While some of these challenges are unique to Mozilla, many of them plague every company out there creating global software. With almost 100 supported languages, Firefox faces many unique and common industry localization challenges. Using traditional localization solutions, these are difficult to overcome. We’ve found that software localization has been dominated by some outdated paradigms, which introduce significant problems.

1. Translations map one-to-one to the source language.
2. Users receive localization updates in the form of new executable builds.
3. User language preferences are binary, with English as the default fallback locale.

With a broad, long-term vision, we began working on Fluent, a modern localization system that not only addresses Firefox’s legacy i18n/l10n code, but also aims to overturn these paradigms for everyone who develops global software.

**Problem paradigm: Translations map one-to-one to the source language**

The grammar of the source language, which at Mozilla is English, imposes limits on the expressiveness of the translation. Consider the following message that appears in Firefox when the user tries to close a window with more than one tab:

```
tabs-close-warning-multiple =

You are about to close {$count} tabs.
Are you sure you want to continue?
```

In English, a single variant of the message is enough for all possible values of \$count.

In English, a variant of the message is enough for all possible values of \$count.

Many translators, however, will quickly point out that the word *tab* will take different forms depending on the exact value of the \$count variable.

In traditional localization solutions, the onus of adapting this message to other languages is on developers. They need to account for the fact that other languages distinguish between more than one plural form, even if English doesn’t. As the number of languages supported in the application grows, this problem scales up quickly — and not well.

- In some languages, nouns have genders that require different forms of adjectives and past participles. In
French, connecté, connectée, connectés and connectées all mean connected.

- Style guides may require that different terms be used depending on the platform the software runs on. In English Firefox, we use Settings on Windows and Preferences on other systems, to match the wording of the user’s operating system. In Japanese, the difference is more stark: some computer-related terms are spelled with a different writing system depending on the user’s operating system.

- The context and the target audience of the application may require adjustments to the copy. In English, software used in accounting may format numbers differently than a social media website. But in other languages, such a distinction may not be necessary.

There are many grammatical and stylistic variations that don’t map one-to-one between languages. Supporting all of them using traditional localization solutions isn’t straightforward. Some language features require trade-offs in order to support them, or aren’t possible at all.

Fluent turns this localization paradigm on its head. Rather than require developers to predict all possible permutations of complexity in all supported languages, Fluent keeps the source language as simple as it can be. We call this idea asymmetric localization, and it makes it possible to cater to the grammar and style of other languages, independently of the source language.

Consider the Czech translation of the “tab close” message discussed above. The word panel (tab) must take one of two plural forms: panely for counts of 2, 3 and 4, and panely for all other numbers.

```
tabs-close-warning-multiple = {$count -> [few] Chystáte se zavřít {$count} panely.

Opravdu chcete pokračovat?
* [other] Chystáte se zavřít {$count} panelů.

Opravdu chcete pokračovat?
```

Fluent empowers translators to create grammatically correct translations and leverage the expressive power of their language. With Fluent, the Czech translation can now benefit from correct plural forms for all possible values of the $count variable.

In Czech, $count values of 2, 3 and 4 require a special plural form of the noun.

At the same time, no changes are required to the source code or the source copy. In fact, the logic added by the Czech translator to the Czech translation doesn’t affect any other language. The same message in French is a simple sentence, similar to the English one:

```
tabs-close-warning-multiple = Vous êtes sur le point de fermer {$count} onglets.
```

Voulez-vous vraiment continuer ?

The concept of asymmetric localization is the key innovation of Fluent, built upon 20 years of Mozilla’s history of successfully shipping localized software. Many key ideas in Fluent have also been inspired by XLIFF and ICU’s MessageFormat. Asymmetric localization doesn’t stop at plurals, however. Fluent translations can vary depending on the gender, the grammatical case, the operating system and many more variables. All of this happens in isolation; the fact that one language benefits from more advanced logic doesn’t require any other localization to apply it. Each localization is in control of how complex the translation becomes.

**Problem paradigm: Users receive localization updates in the form of new executable builds**

According to the traditional software localization process, a localized product is produced as a result of building static language resources into an executable file, which is then distributed to users. Any update to these language resources requires a new executable file and for the distribution chain to carry that to users. Because of this, most software companies elect to postpone localization updates from the moment they’re available to a time in which they can be bundled with other improvements to the software. While this is a cost- and effort-efficient means of producing software updates, it also treats localization, and users of localized products, as second-class citizens by prolonging the user’s exposure to broken or unintelligible localization.

With Fluent, this process can be decoupled, allowing for localization updates to ship independent of a broader release schedule. Rather than language resources being part of the software package alone, they’re delivered to users via secure API calls when they start up the software. Even better, these API calls make it possible to deliver localization updates without intervention from the user — no need to manually initiate an update or even restart the software. For web apps, the process is even more efficient: users see updates immediately, without even needing to refresh the page.
This asynchronous localization delivery method enables innovations in localization quality assurance and quality control. With this method, we can significantly reduce the overall time a user is exposed to bad localization. In the case of Firefox, this means reducing error correction turnaround time from weeks to minutes. For Mozilla, whose primary localization model is community-based, reducing this time frame incentivizes participation, as the contributions can be made visible in production almost instantly. This method also enables robust, live, in-context localization for software packages, allowing you to capture localization issues in real time during the translation process.

**Problem paradigm: User language preferences are binary, with English as the default fallback locale**

Language choice in software is regularly presented to users as an either/or binary option (such as: either you want to use Firefox in English, or you want to use Firefox in Spanish). This becomes problematic when a user interface (UI) element is left untranslated in the target locale; the only fallback option this paradigm allows for is the source locale, which is English in most cases.

To address this problem for global users, Fluent implements a robust locale fallback chain mechanism that more accurately reflects the linguistic state of global users. While English may have a large foothold globally, nearly 85% of the world is unable to understand English content, according to numbers from Ethnologue. Through this fallback chain mechanism, Fluent can serve content to users in alternative target languages rather than automatically defaulting to English. Fluent does this through its runtime API, when a translation for a UI element isn’t available in the user’s primary locale, Fluent looks through the list of the user’s preferred locales for one that has the translation available and serves it to the user. For users who opt in to this feature by defining their list of preferred locales, this creates a mixed-content localization that conveys messages to the user in their primary language, as well as alternative target languages they understand.

For Mozilla, this locale fallback chain is critical, as we often localize Firefox and other projects into long tail locales through volunteers and we are subject to the availability of those volunteers to localize new content when it’s available. With Fluent, we have the flexibility of patiently waiting for volunteers to localize new content for those locales, while still serving users content in a relevant language.

Fluent has also made it easier to ship localized software in long tail locales by allowing us to prioritize content within a single project or product. By doing so, we ensure that primary content within the product is localized into that locale, while allowing alternative and relevant locales to cover the secondary and tertiary content. Investment is thus reduced to produce these long tail localizations, while still enabling us to serve a localized minimum viable product of sorts to users on a locale-by-locale basis.

**What’s next for Fluent**

We’re thrilled about the opportunities that Fluent presents for changing the localization landscape and improving localized user experiences for all Firefox users. Fluent recently launched the 1.0 release of its specification, with experimental bindings available for HTML, Python, Javascript, React and Rust. We’re actively in the process of porting all of Firefox’s UI to the Fluent localization system. Thanks to the help of a dedicated group of students from Michigan State University, we’ve been able to port approximately 26% of Firefox to Fluent. To track our progress with porting Firefox to Fluent, please visit https://arewefluentyet.org.

Being an open source project, this release is the culmination of collaboration from many people around the globe. Now that Fluent 1.0 has been launched, we’re looking forward to collaborating with CAT and TMS developers to increase adoption for Fluent in the industry. We also plan to present Fluent’s syntax as a viable next generation candidate for Unicode’s MessageFormat in ICU. Finally, we see potential in performing targeted user studies to evaluate user perceptions on the new paradigms Fluent introduces to the localized user experience.

To learn more about Fluent or helping us expand Fluent’s interoperability, visit https://projectfluent.org. [M]