# Open source localization

*Jeff Beatty & Staś Małolepszy*

Open source. When a coworker begins giving you suggestions about the latest open source application you can download to your laptop, what do you typically think of? Maybe some commonly associated buzzwords such as *free*, *hacker* or *source code*. How about an image of a programmer diligently typing away at a computer, lines upon lines of code flying across the screen for hours and hours? If this is what you think of when you hear the term *open source*, you're not alone and you're not far off.

The open source movement is about open development and design of software applications. Open source software is not developed behind closed doors by a small team of developers, but in the open, and by everyone willing to enhance the application through their own time commitment and expertise. This philosophy automatically lends itself to the recently popular crowdsourcing methodology – organizing a volunteer community to perform a given task for your organization. For open source, that task is generally software engineering. For us at Mozilla, it means a lot more.

At Mozilla, we're all about the web. We love the web. Our mission is centered on putting the power of the web into people's hands, working on behalf of you, the user, and promoting openness, innovation and opportunity on the web. We believe the web belongs to the people who make it and use it, no matter where they are or what language they speak. This being the case, we place localization very high on our list of priorities. We have an innovative vision for what open localization should look like. We're excited to share this vision with everyone else who wants to see a more open and collaborative form of localization. Through developing open localization technology, we're hoping to accomplish three main goals: cause a linguistic power shift from source code engineers to localizers, improve localizers' ability to freely express themselves while localizing an application, and improve localizability within web and mobile technology. We've been developing some new localization technologies that we think have the potential to accomplish these goals.

## L20n

We affectionately call one of our most innovative, open localization technologies L20n. The name is a play on the acronym l10n, short for localization, and the notion of next generation technology. L20n is a localization framework (which is partly comprised of a scripting declarative programming language) meant to transfer the ability to localize software using the fullness of any language from the developer to the localizer. L20n empowers localizers to be more independent of source language developers and have more control and flexibility in localizing software according to their native language's demands.

One of the primary challenges these localizers face is that they have to rely on an application's programmed logic to appropriately process strings in their native language. They have almost no control over how an application will handle their language's plural, singular and gender syntax, not to mention the challenges presented when an application concatenates strings (combination of variables containing strings) with only the source language in mind.

Currently the responsibility for resolving these challenges falls on the shoulders of developers. While coding, they must incorporate internationalization and localizable coding practices into their applications that account for these challenges. Unfortunately, this presents a larger-than-life expectation for coders: not every coder can understand the syntactical intricacies of every single language. In addition, developer toolkits don't always contain prelocalized materials to help them internationalize. Should localizers encounter a localization bug, they cannot fix it themselves without understanding how to

*Jeff Beatty is program manager of localization at Mozilla. This successful program and its contributors actively ship Firefox in over 90 locales in a rapid development release cycle.*

*Staś Małolepszy 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.*

code, and even then they risk creating bugs in the application itself. How many of you have found yourselves in that same situation? This is because an application's programming logic and its localization logic are organically connected. Enter L20n.

L20n seeks to separate these two logic sets, allowing localizers to have more control over how their language appears and operates within the application's programming logic. Localizers will be able to establish their own contextual plural and gender-specific forms within specific strings (or even chain multiple plural forms within a single string using object variables and attributes) without causing disruptions in the application. Establishing context also changes the way strings are concatenated within the application. Taking this approach, localizers are able to contextualize and process plurals, genders, declensions, conjugations, registers and time of day variations with more expressive liberty.

Here's a simple example of what we're describing. Let's look at how L20n intelligently processes gender by looking at the strings "Firefox has been updated," and "Aurora has been updated" in English (Aurora is the name of the test version of Firefox) and comparing them to a gender-dominant language such as Polish. We'll use brandName as a variable to store Firefox/Aurora and update as a variable to contain the rest of the string.

The localization resource file for English might look like the following:

```
<brandName "Firefox">
<update "{{ brandName }} has been updated.">
```

When the developer asks for the value of the update message in English, L20n will substitute "Firefox" for `{{ brandName }}` and display the string "Firefox has been updated." Simple enough, right? Similarly, if the value of `{{ brandName }}` is Aurora, the same logic will be applied and the final message will be "Aurora has been updated."

```
<brandName "Aurora">
<update "{{ brandName }} has been updated.">
```

There's nothing special about the example above because English nouns don't have genders (in principle). Let's add Polish gender rules to the mix and see how L20n copes with the challenge.

L20n allows us to extend the data model of each message with additional

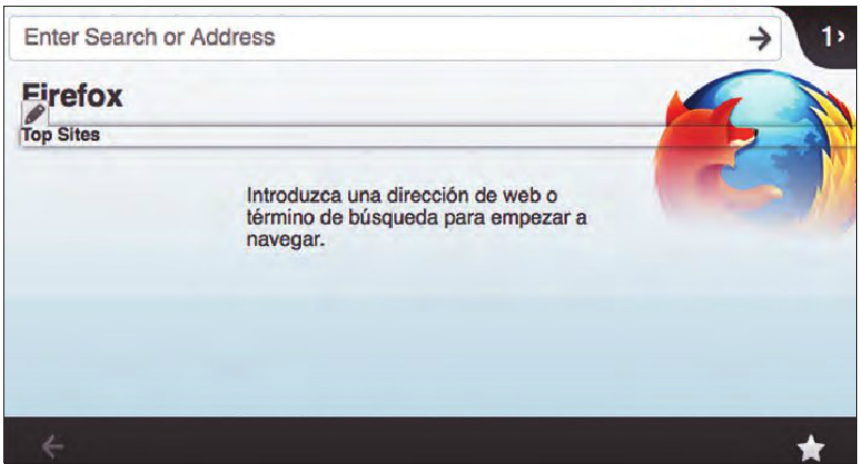


Figure 1: Pontoon being used to localize the browser app of the upcoming Firefox mobile OS.

pieces of information called attributes. Here, we're adding the gender attribute to the Polish version of the localization resource. The name is arbitrary and chosen by the translator. This additional metadata will only be present in Polish and doesn't affect the L20n code for other languages or the source code of the software.

For the official version of Firefox, we'd use the following resource:

```
<brandName "Firefox"
 gender: 'male'>
```

And for the development version used for testing, the same resource would read:

```
<brandName "Aurora"
 gender: 'female'>
```

We've now identified the gender for each brand name. Now let's use it to produce grammatically correct versions of the update message in Polish. The following string would be present in the Polish versions of both Firefox and Aurora.

```
<update[brandName..gender] {
  male:   '{{  brandName  }} został zaktualizowany.'
  female:  '{{  brandName  }} została zaktualizowana.'
}>
```

Notice how update has changed. Its value is now a hash (a dictionary-like


Figure 2: Localization workbench being used to localize Firefox OS web app seen in Figure 1.

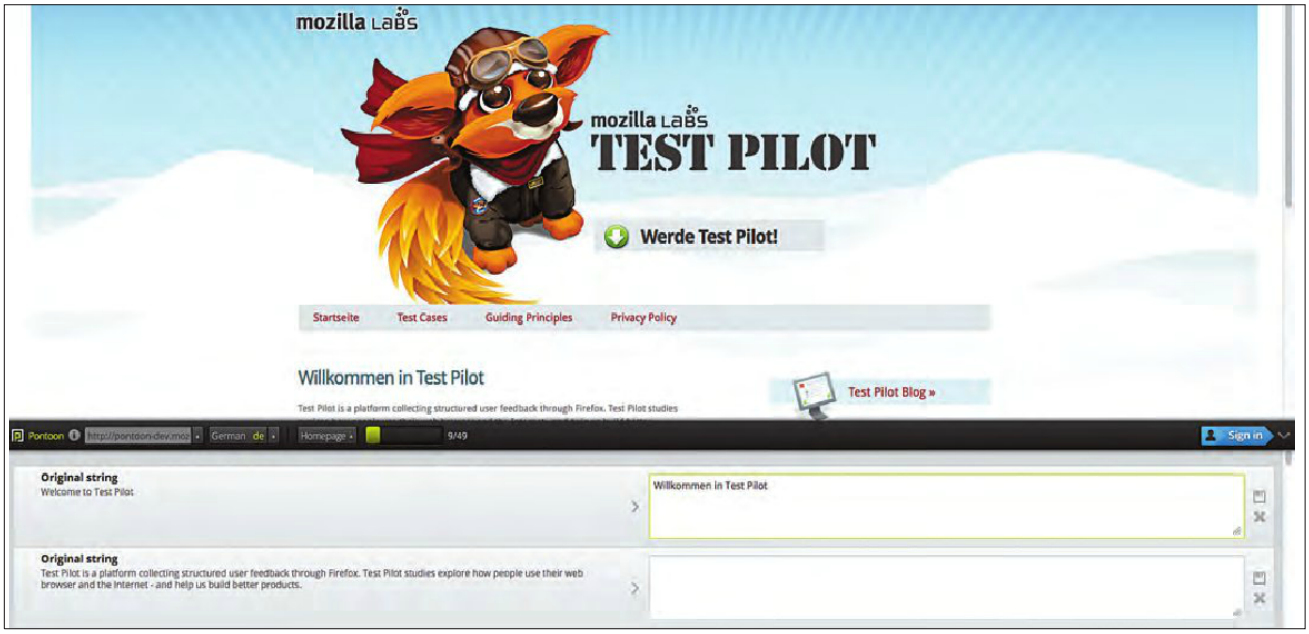


Figure 3: Pontoon demo of website localization. This image displays the localization workbench, including most of its features.

variable object which can contain multiple values called keys) with two keys: male and female, each with a string value assigned to it. Essentially, it is two strings encapsulated under one identifier: update. Furthermore, the brackets following the update identifier introduce an index. It tells L20n to look at the value of the expression between square brackets (the gender attribute of the brand name object, as specified by the double-dot selector) and choose the corresponding key from the hash.

By adding gender to the brand name, the translator has determined that if the brand name is masculine, then the male version of the string is used: *Firefox został zaktualizowany*. If the brand name is feminine, then the female version of the string is used: *Aurora została zaktualizowana*. Pretty neat, right? By localizing this way, the localizer can translate for both forms and leave it up to the localization logic of the application to determine which gender form is appropriate for the corresponding product name. The same general logic can also be applied to cases involving plurals, verb conjugations, declensions and so on.

An important thing to remember here: all of this is happening in the L20n localization file (remember how we're separating localization logic from application logic? This is where that happens). The source, English in our case, is left unchanged. Similarly, the developers don't even have to know about the fact that some languages support genders. All they ask of L20n is for the value of the update string, and L20n makes sure it's grammatically correct and fits the context.

Where other localization platforms and toolkits show lists of supported features, L20n is a framework that allows localizers and localization engineers to implement, or codify, the logic and the grammar of any language. Rather than providing ready-made solutions, L20n hopes – in a truly open source spirit – to leverage the diversity of its users and let them extend it with features specific to their languages. And thanks to this community-driven effort, we've already seen an implementation of plural rules in over 80 languages, as well as implementations of declensions, genders and operating system-specific selectors for access keys. L20n is grammar-feature-agnostic and it doesn't favor or discriminate creative grammars. It lends itself to linguistic variety by providing more flexibility of linguistic expression that previously was unavailable to localizers, all without making the source code developer think too much.

## Pontoon

How many of you are familiar with the instructions shampoo makers used to put on bottles of their shampoo? You won't find them printed this way anymore, but the shampoo instructions for use always used to consist of the same three steps: lather, rinse, repeat. What does shampoo have to do with localization technology?

Well, we like to look at the standard localization process for strings in both websites and applications as following a "shampoo process" but with one additional step: externalize strings, localize strings, integrate strings for testing, repeat if needed. The shampoo process is a standard process for localizing, but it is not without its challenges. By externalizing strings, localizers are expected to translate strings without visual context. This can result in mistranslations and issues with user interface (UI) real estate, and requires the use of multiple tools to get the job done, such as a browser, a text editor and a computer-aided translation tool, all being used simultaneously. The process itself can be very time consuming with all the switching from tool to tool. It also doesn't establish a framework for real-time collaboration with a team. We think that we've created an open source localization tool that can resolve all of these challenges.

Pontoon is an open what-you-see-is-what-you-get (WYSIWYG) localization tool for localizing web pages built with open standards, like HTML and JavaScript. With just a snippet of code, a website administrator can add Pontoon to a website to begin allowing volunteer translation suggestions. Pontoon can also be added to any mobile web app built in HTML5.

Many of Pontoon's features are what you would come to expect from your standard community oriented, computer-aided translation tool: translation memory (TM) support, machine translation support, translation suggestion approval workflows and a small project dashboard. If you were to play with Pontoon, you would notice that the TM support is a bit different than what you're used to. In its current state, Pontoon is plugged into our online TMsystem called Transvision, which leverages all source and target content from previously localized Mozilla projects. Part of ensuring that Pontoon is an open localization technology is also incorporating support for open industry standards such as TMX and TBX files.

**You can contribute to L20n and Pontoon by visiting the following links:**

- Pontoon demo: http://pontoon-dev.mozillalabs.com
- Pontoon design and roadmap: https://wiki.mozilla.org/L10n:Pontoon
- Pontoon code repository: https://github.com/mathjazz/pontoon
- L20n code repository: https://github.com/L20n
- L20n design and roadmap: https://wiki.mozilla.org/L20n
- L20n mailing list for getting involved in the tools discussion: tools-l10n@lists.mozilla.org

This type of support is on Pontoon's development roadmap. What you also may find unique about Pontoon is the ability to leverage content from other locales. This feature can result in a much leaner localization process with faster time-to-market, should you be looking to expand into new locales similar to

locales you already support. For example, expanding localization support into Mexican Spanish can be much simpler and faster if you leverage your Argentine Spanish community's work directly. Pontoon allows you to do that within the application itself. And last on the feature list is Mozilla Persona integration for browser-based user authentication.

With easy-to-use features like these, Pontoon is able to provide open source solutions to the shampoo approach to localization. Since Pontoon is a WYSIWYG and web-based tool, localizers no longer have to translate strings without context. This allows them to see immediately what UI real estate issues their contributions may have on the page and resolve them in real time. Translation and integration of translated strings from resource files for testing all happen in real time, moving the shampoo process behind the scenes and away from the localizer's workflow. Having a robust feature set in a web-based tool also reduces the need to hop from one application to another to get the job done. Your localizers can leverage TM content and resolve common web localization issues all within one application, saving time and frustration. This is especially crucial in a community-driven localization effort. Making localization easy is half the battle to keeping your contributors happy and active.

## Open localization is your localization

We'll admit, you may have seen many of these features before in other WYSIWYG localization tools. You may be actively using one of these solutions as part of your community-driven localization strategy. The biggest difference between what you've seen before and what L20n and Pontoon offer is the chance to be in control of what is developed in these tools. The best part about open source projects is that if you want something in your tool changed, as a stakeholder in the project, you have a say. In fact, you have more than a say; you can help determine the destiny of the project by contributing to it. Want to see Pontoon support other localization standards like XLIFF or PO? You can make that happen by adding code to the project. If you're not technically inclined, you can even make it happen by being a super user and interacting directly with Pontoon's developers. By being vocal in Pontoon's design and development, you can make Pontoon an even greater application! The same is true with L20n. Adding open source to your localization strategy gives you more collaborative power over how your strategy is realized, consequently improving your ability to localize robustly and making the process better for your localizers, community, and ultimately, your users. Our strategy is based on putting users first, no matter where they are. We make these tools to provide open source solutions to your localization problems. Join us in our attempts to revolutionize open localization. Open localization, after all, is your localization. **M**