Localization 101:

A Beginner's Guide to Software Localization

Transifex™

If you're a developer who's been handed the responsibility of software localization and you're unfamiliar with the process, the road ahead may seem daunting. While localization can be complex, learning about localization in general and planning ahead can ensure the efficient and cost-effective localization of your product. In this guide, we'll share key information about software localization as well as outlining the steps for how to localize software from start to finish.

What is software localization?

Software Localization (also referred to as l10n) is the process of adapting or translating software to a specific locale's language, culture, and legal requirements. In many cases, localization will require modifications to the user-visible components of software such as the user interface, images, and documentation.

Effectively localized software will allow your users to connect with your software in a language and format that feels native to them.

For a software developer, the first step of localization is actually internationalization, or i18n. Internationalization is the process of designing a localizable user-interface and abstracting all localizable elements (user-visible strings; locale-specific data like date, time, and currency formats; and keyboard usage) out of your application source code and into external files that can be made available for translation.

What are the best practices for localization?

There are no hard and fast rules or specific global standards governing localization, but you'll find that adhering to 4 best practices will result in a hassle-free localization process:

1. **View localization as an extension of product development.**
   Rather than looking at software localization as an isolated task in the final stages of your product release, your entire development process should be conducted with an eye toward localization-related tasks at each step of the software development process, from release to maintenance. This mindset can ensure that you plan accordingly for localization and that your software can be translated into other languages without causing you extra work or time.

2. **Keep your source language simple.**
   Using industry-related or complicated words and phrases in your user interface and documentation may appeal to a select audience, but at the same time, it may fail to elicit positive responses in other languages or another culture. It also makes translation difficult or even impossible in some cases, so keep your source language as simple as possible.

3. **Focus on your content strings.**
   As a developer, much of your localization efforts will revolve around your "strings", or the words and phrases served within your application. Along with externalizing all user-visible strings, localization best practices include:

   • never hard code strings
   • avoid concatenating strings
   • always provide explanatory comments

   We'll discuss this last point further later in this guide.

4. **Deploy a localization platform.**
   To create a comprehensive localization process, you need to plan for the steps beyond software development, which can be simplified by using a robust localization platform. Localization platforms automate the steps of taking externalized strings, making them available to translators, managing the translation process including reviews and approvals, and, in some cases, automatically publishing translated string files back into the development repository for localized code deployment. Platforms that are designed specifically for developers often offer a higher level of customization and can include integrations with tools such as GitHub, Python, and Django. A handful of localization platforms are also integrated with translation providers so you don't have to spend time hiring and managing a team of translators.

How is localization done?

As we mentioned earlier, localization involves two key phases: Internationalization and Localization. We'll review the key elements of both phases to provide a roadmap for your process.

Phase 1: Internationalization

While localization is the actual process of adapting your software to another language, internationalization includes design and development practices that enable a simple and straightforward localization. During the internationalization phase, you'll need to:

Review Application Framework

Your ability to internationalize your software will be dependent on your application framework. An application framework that adequately supports internationalization should contain the following elements:

- Resources files
- Resource bundles
- Unicode support
- APIs and additional features to support multiple locales

An explanation of each of these elements is included below.

Plan for Text in Other Languages

Translated text may take up more space, or in some cases, less space, than your source language causing your neat and perfectly laid-out design to appear crowded or even indecipherable when translated. The design of your user interface must allow room for expansion and contraction of text strings in translated languages.

To ensure that your content is viewed how you intended, we recommend programming dynamic UI expansion into your software. For example, if you are developing an iOS app, you should use Auto Layout to ensure that your views realign suitably for every locale. For an Android app, you can build a dynamic UI with Fragments.

Make use of features supported by your application framework to program dynamic UI expansion.

Another point to consider: In languages such as Arabic, text is read from right-to-left (RTL) rather than left-to-right (LTR) so your design will also need to support RTL script.

Code Strings with Global Expansion in Mind

During the internationalization phase, strings must be extracted, localized, and reinserted back into the code. Coding strings under Unicode/UTF-8 will generally be the best option, unless you are working with Asian languages that require UTF-16.

Externalize Your Locale Specific Data

Strings that have been prepped for localization must be externalized, meaning you will have to save your strings to a translation file, also known as a resource file. We'll get into resource files in more depth below, but know that it's common to create multiple versions of your resources files for every target language.

Resource Files

Resource files for localization contain resources specific to a given language. Recent application frameworks typically define a key-delimiter-value syntax for resource files, which will determine how you create pairs of keys and values for all user-visible strings in your application. Then, based on the user's locale, appropriate values for these strings will be fetched and rendered in the user interface.

Resource files are created in various formats depending on the type of software you are localizing. The chart below explains some of the options in greater detail as well as their associated resource bundle format.

Resource Bundle

Resource bundle is a general term, but when used in regards to localization, it typically means a set of resource files sharing a common base name and having an additional language-specific component in its name to identify specific locales.

Resource file formats

Desktop Applications		iOS Apps		Android Apps	
Properties files		Localizable.Strings files		Strings.xml files	
Properties files are used in typical software deployments. It is common to create multiple .properties files, one for each target language.		For iOS, the files with localizable strings are referred to as the Strings file and you must make use of these with the NSLocalizedString macro. Your application should have specific locale directories and you must localize the Localizable.string files in each language folder.		For an Android app, all your localizable strings will go to the strings.xml file and into appropriate locale directories.	
messages_en.properties	messages_de.properties	en.lproj/Localizable.strings	de.lproj/Localizable.strings	res/values/strings.xml	res/values-de/strings.xml
label.name = Your name label.greeting = Welcome	label.name = Ihren namen label.greeting = willkommen	/* Label item Your name */ "name" = "Your name"; /* Label item Greeting*/ "greeting"="Welcome";	/* Label item Your name */ "name" = "Ihren namen"; /* Label item Greeting*/ "greeting"=" willkommen";	<?xml version="1.0" encoding="utf-8"?> <resources> <string name="name"> Your name </string> <string name="greeting"> Welcome </string> </resources>	<?xml version="1.0" encoding="utf-8"?> <resources> <string name="name"> Ihren namen </string> <string name="greeting"> willkommen </string> </resources>
Resource Bundle		Application Bundle		Resource Bundle	

Focus On Your Strings

The final thing you'll need to do before the actual translation process revolves around your strings. Referring back to our best practices, you'll need to:

- **Avoid hard-coded strings.** All user-visible strings must be externalized appropriately. Avoiding hard coded strings will make your life easier, and when unsure, perform pseudo localization to root out hard coded strings. Pseudo-localization is often performed in a separate localization testing branch, allowing you to replace your strings using a regular expression. Then, when you run your software, any hard-coded string will be clearly visible.
- **Avoid concatenation of strings.** Two or more strings are sometimes concatenated by developers for the purpose of saving space. However, word order varies significantly in each language and string concatenation will most likely result in translation errors in the localization process.
- **Provide self-explanatory comments.** Providing explanatory comments for your strings to define context wherever possible will go a long way in assuring better translation accuracy with less back and forth communication efforts. This means time savings for you and fewer headaches.

Internationalization should not be treated as a separate step in your software development process, but rather a fundamental thought in every stage of your design and development process.

Phase 2: Localization

In the past, people often localized content using spreadsheets, requiring developers to copy and paste strings and source content into the spreadsheets before sending it off to a single or multiple translators. Translators would then have to access the spreadsheet, put in their translations, and some sort of quality assurance measure would be implemented to ensure translations were correct before pushing the new strings back into the software. While this is just one translation option, the remainder of our article will focus on using a localization platform, a more commonly adopted way of localizing content that helps developers save time, while ensuring overall quality of translations.

Select a localization platform that provides a good editor, supports multiple source formats, allows your translation cycles to integrate well with your build cycles, and offers additional localization features designed for developers.

Extract Resource Files

To start the localization process, you'll need to extract your resource files for translation. Localization platforms like Transifex support a variety of source file formants and you'll be able to directly upload your resource files. In some cases, you'll have to export your resource files into standard XLIFF (XML Localization Interchange File Format) files or other localization file formats to make them suitable for translation into multiple languages.

If you are using Transifex, all you need to do is directly upload the resource files of your project in one of the supported resource file formats. The platform will automatically extract all your source strings and make them available for translation. If you are using GitHub, you can also integrate Transifex with your GitHub repository using Txgh. This will enable you to automatically send new or untranslated strings directly into Transifex, and retrieve translated strings automatically when the translation process is complete.

Translate

When it comes to translating your content, it's crucial to take the time to select the right translator for the job – ideally a native language speaker who has experience with translations. If you decide to use a localization platform, you'll likely be provided with an integrated translation provider or you can invite translators of your choice from an outside translation agency. Whether through the platform or personal invite, your translators will have access to your source strings, can view them in the platform's editor, and translate them appropriately and within the context of your content.

Review

All translations must be reviewed for accuracy, language quality, terminology, and any other requirements of your software. The translators, based on feedback from you, a translation administrator, or project manager, must make any necessary modifications. If you're using Transifex, the platform has built-in features for review as well as options to keep your communication with your translators efficient and quick.deploy your localized application with the new translations.

Copy Translated Files into Code Structure

After your translations are completed, you'll need to copy the translated files into your code structure. Quality translation platforms will provide an option to pull the translated files that are ready for use into your application, as mentioned above. The next step is to import these translated files into your application and deploy your localized application with the new translations.

When is the localization process complete?

Once you have made sure that your product is bug-free and sufficiently documented, announce a "string freeze" on your product. This means you can no longer change the code in a way that affects the source strings (except for specific improvements). The string freeze will allow translators to work on a stable set of strings and ensure adequate time is available to translate and review. Before the final release, you can obtain the translation for all your target languages, compile them into your product and release it.

Have other questions about localization? Visit our website at www.transifex.com!