GUIDE

Structuring Your Content & Projects for Continuous Localization

Table of Contents

Introduction

Lessons from 10+ Years of Taking Companies Global

A Guide for Fast, Tech-Focused Companies

We put together this guide for fast-moving companies like you — to empower your team to understand and leverage the power of continuous localization to overcome all those challenges that come with the outdated form of localization (a lot of manual files and email chains). In this guide, we break down the ins and outs of effectively structuring your content and projects for continuous localization, so you can finally localize at the speed of your content and product creation.

"...you can finally localize at the speed of your content and product creation..."

Learning Challenges to Overcome Them

In our journey building this platform, we've learned firsthand from global customers the pain points they face when applying localization (L10n) processes to their modern software development. From properly keeping track of content changes to making sure each team member can effectively focus on their own section of the workflow (instead of having developers grapple with issues like dealing with files sent manually via email because translators had trouble using a repository)...the list of challenges gets quite long.

Transifex: 10+ Years of Taking Companies Global

We founded Transifex with the mission to break down language barriers. How we build products has evolved greatly over the last decade, with faster release cycles transforming the pace of localization that companies are seeking. Today, every company is a fast-moving company — that's what software enables. And we want our content to be localized as quickly as its created.

We felt we needed to build a platform that would not only support frequent changes of content, but also create fluid workflows that removed all the bureaucracy and red tape that comes with trying to collaborate across the many dispersed roles of a localization team. The result: our ever-evolving Transifex platform and resources like this guide.

Common Software Localization Mistakes

Before we dive into the nitty gritty of continuous localization set up, let's start with a 'what not to do' section. Whether you are just embarking on your localization journey or trying to localize continuously, familiarize yourself with the list below to set a baseline of pitfalls to avoid.

"...An increasing amount of back and forth will be required to get things done...."

1. Not taking i18n seriously.

Failing to apply internationalization (i18n) best practices from the beginning can be a huge setback that results in a lot of technical debt due to the addition of costly quality control cycles that will delay – or even prevent – you from getting your content properly localized altogether.

Companies that have a process that includes a fair amount of copy & paste into spreadsheets or cloud storage folders (which will get replicated into numerous languages) will learn that things can quickly become very unmanageable. An increasing amount of back and forth will be required to get things done.

To be fair, the status quo of internationalization practices that developers need to follow for software development today is part of the limitations we see on day-to-day localization processes. Programming language frameworks are looking at things only through a technical lens, and often not giving the necessary attention to interoperability across platforms or to how people will handle content from the linguistic point of view.

"People can't be effective if the process and tools in place don't allow them to be...."

2. Lack of tools to support efficiency and collaboration.

Localization is an inherently cross-departmental process that requires timely collaboration, especially when there are tight deadlines to meet. When the content to be localized is floating around without a consolidated place to manage it, efficiency will go down drastically as the organization grows.

People can't be effective if the process and tools in place don't allow them to be. Decreasing complexity around how to find the correct files, and helping people to interface with content more easily will allow you to streamline the process. Issues will always come up (such as a string not being properly pluralized by a junior developer) but making sure different people can easily collaborate to resolve them, and even decrease the number of issues, is at the core of what will make localization ultimately successful.

3. Lack of control over content

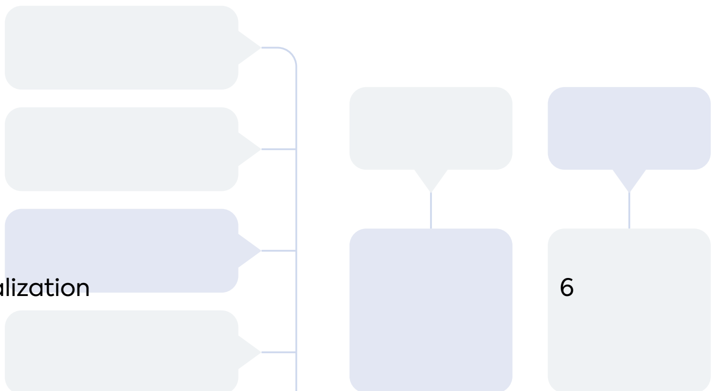
We often see the mindset of the "one-off project" being applied to localization, but having control over your organization's content is also key. It gives you the opportunity to manage the state of your content on your own terms and keep track of your translations regardless of where they originate from. As your team gets better at releasing content and features faster, you need to find ways to track the state/status of each content, which might be coming from different locations (web, marketing, iOS, Android, databases).

4. Not providing enough context to produce quality translations.

When talking about your brand it's always important to make sure your localized content has the right tone and style. Context should be managed together with your content and most importantly, it should be passed on in a very streamlined way. The key is to set things up for success in the first place so that you can catch fewer issues earlier in the process, rather than dealing with them later (when it's too late).

"The key is to set things up for success in the first place so that you can catch fewer issues earlier in the process...."

Now that we've walked through the common mistakes, you know what to avoid and are ready to start your journey to setting up the best continuous localization workflows for your projects and teams.



Part 1

Internationalization (i18n)

The Importance of Internationalization (i18n)

Before you localize, you must make sure to adapt your software to enable full multilingual capabilities. This is called internationalization and its the process of adapting your software so that it can properly and scalably support formats based on languages and locales. It is important to internationalize early in the process so that you create ongoing processes that will scale and integrate into all future sprints and releases of translated content.

In short, internationalization lays the foundation of having well-structured content and workflows from design to development — across locales and languages.

Remember

Don't treat i18n as a separate step in your development process, but rather as a foundational function that should be integrated into every step of your localization design and development workflows.

Finding & Fixing i18n Bugs

As you internationalize, bugs will inevitably arise. When you're working with thousands to millions of lines of code, these issues will undoubtedly be difficult to find. Here's where continuous localization comes in to help simplify the process. Continuous localization enables you to check the new source code as it is written so that both i18n and L10n are seamlessly part of your agile development cycles — saving you time and money.

The Software Localization Checklist

The key phases and steps to keep in mind as you lay your i18n foundation and build continuous L10n on top of it.

PHASE ONE

Review your application framework before you externalize any strings to make sure your software can support your internationalization and localization efforts.

- ☐ Review your software requirements and architecture.
- ☐ Determine whether or not your current database requires changes.
- ☐ Determine if you will need to consider third-party libraries for your content.

PHASE TWO

Plan ahead for text in other languages. Translated text can take more or less space (and even read in the opposite direction!) depending on the target language.

- ☐ Determine how locale(s) will be selected and whether you will set a fallback.
- ☐ Determine how you will support your target locales within your application's programming languages.
- ☐ Evaluate which locale-specific changes will need to be made within your database and add any necessary locale frameworks.

PHASE THREE

Externalize your strings with the following in mind:

- ☐ Code your strings with global expansion in mind. We recommend Unicode/UTF-8 as the best coding option (unless you are working with Asian languages that require UTF-16). Replace locale-limited functions and code.
- ☐ Avoid hard-coded and concatenated strings, to make sure that you can easily automate your localization processes and also make sure you don't switch up any word order that could change the meaning of translated content in the target language.
- ☐ Provide self-explanatory comments for your strings, which (believe us) will go a long way for translation accuracy and also overall workflow (read: less back-and-forth communication across teams, which means more time saved and fewer headaches!)

Part 2

Structuring Your Content & Projects

Organizing Your Projects & Resources

Projects can be organized in any way you like. Think about the projects as directories and resources as files. Here are a few things to keep in mind:

- Group related content into a single project. For example, one project for your iOS app, and another for your website. Each project has certain settings and configuration options (e.g. translation team, translation memory, workflow settings).
- Create separate projects for content that needs to be translated to separate sets of languages. Each of your projects will be translated to one or more target languages, so creating separate projects will help with organization. If you really want to have content in the same project which needs to be translated into different languages, you will need to tell your translators not to translate those files into some of the project's languages.
- Assign each project to a specific team of translators and reviewers. If you have two resources that you need translated by 2 different teams, put them in separate projects.

Building a Global Content Repository

After you've gotten everything organized, it's time to start thinking about how to build a global content repository. In today's software development workflow, there are various ways to manage your content. With modern localization technology like Transifex Native, you can leverage the power that is building a modern i18n library which, instead of files, uses a centralized, cloud-based global content repository. In practice, this means that source content and translations are automatically synced to a global content repository that's accessible at any time.

Here are just some of the benefits of building and using a global content repository:

'Fileless' resources

You and your teammates can interact with the content without the limitations of physical files.

Ease of editing

With functionalities like "Search Strings" in Transifex, you can now search and edit content and projects as a whole regardless of how you've structured things.

Quality & consistency

Having one main repository means that you can implement things like Translation Memory to ensure translation consistency across content and projects.

Structuring Your Files

If you are using files, then one file is associated with one resource. In certain cases, you might have thousands of phrases in your database which can be grouped in different ways into resources. This is really up to you! You can have one big resource or multiple smaller ones.

Here are the advantages of each:

- One big resource will make it easier for translators to go over the phrases.
- Multiple smaller resources will allow you to further group your phrases in some logical way and to have multiple same phrases which need to be translated in different ways, etc.

Example Structure:

```
Project: Documentation
    Resource: FAQ
    Resource: About
    Resource: Introduction
Project: Android app
    Resource: UI labels specific to customer X
    Resource: Translations of city names showing in the app
    Resource: UI phrases
Project: User-generated content
    Resource: Customized menu label
    Resource: User Comments
    Resource: Product Reviews
```

NOTE:

In this example, you could also argue that the resource type "UI labels specific to customer X" might be better grouped on a separate project, so that they're all together. It's really up to you. Here's some other cases and structures that we'd recommend:

If you have just 1-2 customers, leave it under the project.

- If you have 100 customers, put it under a separate project.
- If the number of labels per customer is just a couple, instead of having 100 resources of 2phrases, you might want to have just one resource with 200 phrases and use tags to distinguish each customer.

Common Localization File Formats

No matter how distinct they may seem, all localization frameworks have one thing in common: they need a way to store localized text. For a localization effort that's entirely self-contained, it doesn't matter how your translations are stored. However, if you plan to integrate your localization effort with outside tools or translators, the format your translations are stored in can quickly become a barrier to progress.

Here are some the more common localization file formats and established best practices. Whether you're starting a new project or looking to integrate an existing localization effort, knowing which file formats to target can save you time and effort.

Gettext (.po)

Gettext is a popular internationalization framework used in a wide variety of programming languages and operating systems. In addition to supporting a diverse number of languages, language rules, and locale-specific settings, it's supported by a large number of tools such as Poedit, gted, and Virtaal.

A key benefit of gettext is standardization. GNU gettext is one of the more popular open-source implementations of gettext and has been ported to PHP, Python, Perl, and more. WordPress, Ubuntu, and LibreOffice use gettext to provide translations.

XML Localization Interchange File Format (.xliff)

XLIFF is an industry standard format based on XML. XLIFF was designed specifically for the localization industry as a bridge between platforms and tools, such as your application and a localization service. XLIFF also serves to

"No matter how distinct they may seem, all localization frameworks have one thing in common..."

standardize the way information is transferred throughout the localization process, ensuring interoperability across different workflows. Even applications such as Microsoft SharePoint rely on XLIFF to transfer localization information to and from translators.

Like gettext, XLIFF defines a standard format for storing localized text. Your team can use tools such as the Translate Toolkit to generate and convert XLIFF files. Not only does this help speed up the localization process, but it helps ensure your XLIFF files will be complete and parsable by services such as Transifex.

Extensible Markup Language (.xml)

Unlike most of the other formats on this list, XML is a language used to encode data within a document. Whereas most formats provide a rigid structure for defining your localization data, the structure of an XML document is defined within the document itself. This means one XML document can have a drastically different structure than another document even if both files use a completely valid syntax.

Because of this, XML forms the base of many formats such as Windows resource files (.resx, .resw), Android string resource files (.xml), and the XML Localization Interchange File Format (.xliff). Although these formats use the same language, they implement it in slightly different ways. For example, the following XML defines the string "Hello world!" with an ID of "hello_world" in Android:

```
<string name="hello_world">Hello world!</string>
```

The following XML shows the same concept implemented in a Windows desktop application:

```
<data name="hello_world" xml:space="preserve">
     <value>Hello world</value>
</data>
```

Although any platform capable of interpreting XML can parse these files, the platform has to know how the XML document is structured. This is why some localization platforms support some XML-based format, but not others.

XML Localization Interchange File Format (.xliff)

As with XML, JSON is a general-purpose format for transmitting data between applications. JSON supports many of the same benefits as XML, such as a flexible and dynamic structure that can be read by any JSON interpreter (of which there are many). JSON is also arguably more human-readable than XML, making it easier for developers and translators to work with.

“we recommend only storing non-empty string values by surrounding them in double quotes.”

A common problem that many localization teams experience with JSON is invalid data types. Values stored in a JSON file can consist of multiple data types including strings, numbers, and empty (or null) values. For localization purposes, we recommend only storing non-empty string values by surrounding them in double quotes.

Lastly, complex, nested JSON objects can cause problems for certain parsers. A value stored in JSON can be a string, a number, an empty value, a collection of strings or numbers, or even another JSON document. Some localization frameworks may not support complex structures without first requiring additional parsing rules to be defined. JSON is famously used by **MediaWiki** to store over 23,000 translations.

Java Properties (.properties)

Properties files are commonly used in Java applications to store application configuration settings. They're commonly used for localization due to their simplicity and readability. When used for localization, a property file consists primarily of two strings: an identifier followed by the localized text. Different formats (such as **Mozilla localization files**) may provide different features, but they all follow the same basic structure.

A unique point to properties files is that they require ISO-8859-1 (or Latin-1) encoding as opposed to the UTF-8 encoding common to most other formats. While this won't make a significant impact on your use of properties files, it is something for developers to be aware of.

Comma-Separated Values (.csv)

The CSV format is perhaps the simplest format for storing localization information. It consists of groups of two strings separated by a comma, with each group placed on a new line. CSV files are extremely straightforward, easy to parse, and easy to work with, although their flexibility is limited when compared with other formats. **Magento** uses CSV files to manage localized

Rule of Thumb: Use Standard Resources

When structuring your files, make sure you are using standard resource file formats for your localizable text. This will ensure that the localized software performs exactly as it does in your source language and therefore reduce demands made to your developers during the localization process. The result? Much easier and quicker automation of the translation process, reducing time (that would have been dedicated to engineering and QA hours) and therefore lowering costs.

We recommend using standard file formats like: java, .net reex, traditional windows resources, and xml. If you're working in a custom developer environment, use a consistent file structure like XLIFF.

Other Considerations

While the file format you use is more likely to be determined by your localization framework, it's good to know what's out there. **Transifex supports** over **25 localization file formats** in addition to those listed above. You may also be able to convert an existing format to another using tools such as **Translate Toolkit**. For more information, **contact us with your localization questions**.

Part 3

Workflows & Integrations

Finding the Best Approach for Your Team: Try, Test, & Try Again

Before you dive into this section, remember that each company and localization team has its own needs and workflows. So, it's important for you to be aware of the different approaches, try them out, and see what best fits your needs. To test things out, follow these 3 steps:

- Create different files for each version of the app/website.
- Update the existing file with any newly inserted strings.
- Create a different project/workflow for different content types you want to localize (documentation, website, app etc).

Other Integrations Making Developers' Lives Easier

After you've integrated with your TMS, and if that TMS happens to be Transifex, here are some fun integrations and features that help ease workflows:

- Slack
- Amazon Translate
- GitHub
- KantanMT
- Bitbucket
- LexiQA
- Dark Mode

Integrating with your TMS

Before you start localizing, it's important to have an automation in place and spend time on properly setting things up before starting your localization process. This way, you won't need to worry about localization after that, reducing time and costs. Here's what you need to consider:

- How the generated content will be pushed to your TMS. An API is required to set up the connection between the two systems.
- How the translations will be pulled from the TMS once these are done. Except for the API is also vital to have a way to get notified that the translations are done. Checking manually when the translations are completed is not an efficient way and introduces more manual work and delays which ideally should be avoided.
- How engineers and localization managers will communicate with the translators and address their questions. If engineers, for example, do not have access to the TMS (this is a common case), it is important to integrate the tool the engineers use and the TMS so that: a) any questions submitted by translators will be shared with engineers immediately, and b) any answers provided by engineers will be shared with translators immediately.

"Above all, don't let your company or team size stop you. Even small companies invest time in automating this process..."

Above all, don't let your company or team size stop you. Even small companies invest time (if they have available engineering resources) in automating this process and integrating with their TMS.

Wrap Up

The Future of Localization

Continuous & Cloud-Based

To date, the localization world follows some basic internationalization and localization principles that have not changed in decades. For example, the idea that content to be translated should live in a separate directory in your code, within a special file format, which is not consistent across different programming language frameworks, makes reusability of translations across different platforms a real challenge.

Without rethinking the underlying architecture applied to software localization, we are coming to the edge of what we can do to keep adding substantial value to the process. In this time dominated by cloud computing architecture, there is a big potential for internationalization and localization to become first-class citizens of the software development stack, instead of an afterthought.

To usher in the next era of continuous localization, we must take a step back to reevaluate how software localization can evolve to improve these processes that were long overdue for big changes. For example, we see cloud-based systems — like that which is utilized by continuous localization processes — changing the game for developers by enabling them to automatically sync repository changes, control team roles and user access to content, and specify content to be translated in an online letter.

We've come a long way since the early days of localization and will continue to work to build solutions to alleviate the localization pain points for developers and the localization teams in which they play essential roles.

"we must take a step back to reevaluate how software localization can evolve to improve these processes"

About Transifex Native

Here at Transifex, we have create a solution that is transforming the future of localization: Transifex Native. To learn more about how Transifex is transforming software localization, putting global content management of the center of i18n and L10n, read our **Transifex Native Series.**

Go Global with Continuous Localization

To learn more about continuous localization and sign up for your free trial, visit **www.transifex.com/native**.