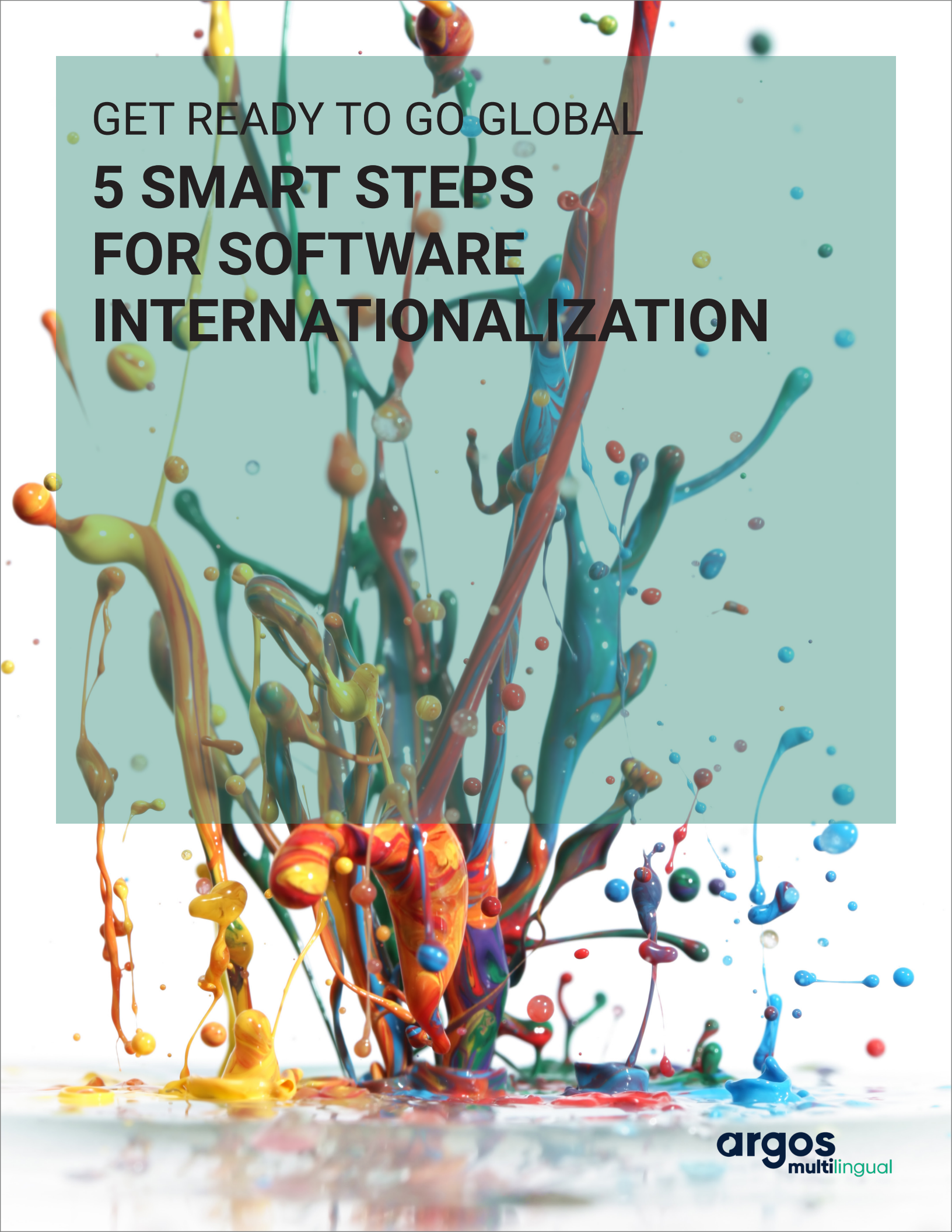
GET READY TO GO GLOBAL

5 SMART STEPS FOR SOFTWARE INTERNATIONALIZATION



argos multilingual

Being Global-Ready

Thinking about launching your software in multiple language markets? Leading CTOs and development managers set their teams up for success with these proven best practices for software internationalization.

In today's global marketplace, a successful business strategy includes effectively selling your products and services across multiple language markets. Additional language markets mean more revenue, faster.

To meet global business needs, software internationalization is the smart way to go. Internationalization (sometimes abbreviated "i18n") is the process of getting your software ready for localization into specific languages. The more thoroughly you prepare your software code for localization, the more smoothly and quickly you can launch your software products in target markets around the world.

Even if your international sales plans are in the future, or your exit strategy is acquisition, developing software that is ready for localization means you're building an asset with the greatest possible business value. Additionally, if you anticipate competitors in your market space, localization-ready code can create an important competitive barrier.

The good news is that some advance planning and development discipline can help you maximize ROI on your core product, and accelerate your successful international launch. Begin now and follow these 5 smart steps for software internationalization. Note that following these steps in order is the ideal scenario, but sometimes unrealistic or not feasible if you've already launched your software. Regardless, even tackling a few steps, in order or not, will help make your software localization process smoother and more cost effective.

1. Plan for localization

Start communicating your plan for internationalization as soon as possible to streamline your time-to-market. Involve your development team and other stakeholders in the discussion right from the start.

- **Write being "global-ready" into your core product strategy and convey it to your team.** Tell your developers from the outset that they're developing a product with international sales potential. It's essential that you integrate internationalization guidelines for development into your workflow now. Writing your code to support language and locale flexibility when the slate is clean will save you significant time and money later. Or, if you're refactoring legacy code, use that opportunity for internationalization of the software to ease your transition to going global.

- **Start working with your translation and localization agency now, even before your source content is ready for translation.** The sooner you can get expert input, the better and less costly your project will be. The right agency for your project will truly take a partnership approach, helping you plan for localization according to your business goals. This includes taking into account budget timeline, available language assets and other money-saving opportunities before project kick-off. Ask your agency for ideas to make the process more efficient. Brainstorm with your development team as well. For example, which is best for your development team's workflow: a daily bug report from your agency, or having bugs entered directly into your database? The right agency will find ways to efficiently integrate their contribution into the workflow your team is using.

- **Get all stakeholders on board as early as possible.** This includes marketing, sales, finance, investors, documentation, training, customer support, and QA/QC teams. Once your revenue and infrastructure teams know you've got a solid plan for delivering a global-ready product, the more support they can provide on the business side. International success is a win for everyone. Talk with them about quantifiably prioritizing international markets, analyzing customer potential, defining the competitive landscape, and other new market issues. This planning will help ensure your development team has the budget and resources needed to deliver localization-ready code.

Now it's time to roll up your sleeves and have your development team implement the following guidelines for internationalization.

2. Take the text out of the code

The first major step in preparing software for localization is to separate the text from the code.

- **Externalize any translatable text into resource files, message catalogs or any other suitable string file.** Any text object or value needing translation is better not being included in the code files. Why? Because your code is safest when only developers – and not localizers – can edit it. Additionally, the localization team's work goes faster and is more accurate if they have a complete set of strings and don't have to worry about messing up the code.

  Next, install any localized files in a locale-specific file structure, or name them with a locale-specific suffix. This is an essential internationalization practice that allows you to maintain a single source code version for all languages, where only the resource files will use different languages. This practice cuts down the localization costs dramatically, as it allows programmers to focus on what they are best at: creating code.

• **Exclude invariant textual objects from localization.** For example, it is generally advisable not to translate the following:

– User names, group names, and passwords
– System or host names
– Names of terminals (/dev/tty*), printers and special devices
– Shell variables and environment variable names
– Message queues, semaphores and shared memory labels
– UNIX commands and command line options (e.g., ls -l is still ls -l in all locales)
– Commands such as /usr/bin/dos2unix and /usr/ccs/bin/gprof
– Commands that are XPG4-compliant (in /usr/xpg4/bin/vi) and have equivalent non-XPG4 commands; non-XPG4 commands that are not fully internationalized. For example, /usr/bin/vi does not process non-EUC codesets, but /usr/xpg4/bin/vi is fully internationalized and can process characters in any locale
– Some GUI textual components, such as keyboard mnemonics and keyboard accelerators

Instead, externalize only the translatable strings. Clearly identify objects that should not be translated as “not for translation,” to ensure that translators or linguists do not accidentally make changes to the code.

• **Always use unique IDs.** Do not rely on ‘IF’ conditions, booleans or on a certain sort order to evaluate string values in your code. Assign a fixed and unique ID to each and every translatable string.

3. Set coding standards

You'll save a lot of time and effort if you establish coding methods early on to support multiple languages and character sets.

- **Use Unicode functions and methods to support virtually all languages or virtually any script.** Applications that store and retrieve text data must allow for input, output, and display of all the characters from any given language. Using Unicode encoding, particularly UTF-8, solves the problem of unsupported character sets and the potential corruption of special characters and accents, including single and double byte languages, as well as right-to-left or bi-directional ones. When you don't use this approach, the likelihood of corrupted characters increases. This, in turn, creates problems deeper than displaying in the wrong way since corrupted characters can disrupt code function.
- **Choose only Unicode-supported third party software.** Your third-party software products and components need to comply with standard internationalization practices. Not using internationalization-ready software can slow or block your global-readiness. If globalization is in your expansion plans, make sure your vendors are ready to support you.

4. Prepare strings for localization

The core of internationalization is to use best practices in handling text strings. The cleaner your strings, the easier and more error-free your localization process will be later.

- **Allow for text expansion in messages, especially for GUI items.** Most of your target languages will require more space, sometimes up to 30-100 percent more.

  Here are some Microsoft translations into German that help illustrate this:

  Link – Verknüpfung
  Login – Anmeldung
  BAM (Business Activity Monitoring) – Geschäftsaktivitätsüberwachung

  Try to place labels above controls, not beside them, so that the labels have room to expand.

  TIP: keep the string length well below your field limit to account for the extra characters that may be needed.

Text expansion will affect layout. Plan for text expansion at the following rates:

0-10 characters = +101-200%

11-20 characters = +81-100%

21-30 characters = +61-80%

31-50 characters = +41-60%

50-70 characters = +31-40%

More than 70 characters = +30-%

- **Avoid text in icons and bitmaps.** As described above, translated text may be too long to fit. This can lead to text truncation in displayed images.
- **Avoid variables if possible.** During translation, variables raise queries as to the gender and single/plural form of the substituted words, making it very difficult to translate accurately on the first attempt any sentences that incorporate those variables.
- **If variables must be used, offer a list of replacements.** Consider all possibilities and allow for gender and plural variations in the translation of the sentences that contain variables. From an internationalization point of view, a higher number of strings is preferable to taking the chances of using the wrong or less suitable form.
- **Avoid concatenation.** Use complete sentences instead, even if that means repeating segments. Concatenation can work for a single language, when word order is predictable. However, word orders vary widely from one language to another. This means string concatenation in the source could render nonsense phrase structures in the localized compiled build that would require tracking, rework and a substantial additional cost.
- **Replace hard returns in the middle of sentences with break tags or soft returns.** Hard carriage returns a signal that the sentence has ended to the translation memory tools used by your translation and localization agency. Inserting hard returns into the middle of a sentence leads to incomplete sentences in the translation database and corrupts the sentence structure in the translated files. Use break tags (such as
) or a soft return.
- **Consider additional breaks for your target languages.** Sentence structure changes in different languages, as well as the length of sentence parts. Therefore, additional breaks and shorter concise sentences may make your code easier to localize.
- **To set up your project for efficient localization, provide context and don't alphabetically sort strings in string tables and resource bundles.** Offer as much context as you can with the externalized strings. This helps your translation and localization agency better adapt the translation to that context. No context means it takes longer to verify and correct your translations. Provide context easily by giving your agency access to the actual product, screenshots, documentation, help files, commented strings, and other product elements. Avoid alphabetical sorting because ordering in your source language will not automatically work in your target languages.

5. Test, fix, iterate

Before you even localize, you can do a test-run of the localization process. This will help you fix any bugs so that the actual localization can go more smoothly.

- **Make the most out of pseudo-translation.** Pseudo-translation is the process of replacing or adding special characters to your software string to find bugs, and then compiling it as well as running it through your normal QA processes. This allows you to predict and correct internationalization issues in the source even before starting localization. To detect character encoding issues and hard-coded text remaining in source files, testers use pseudo-translation to easily track errors. It is also used to identify text inflation and avoid truncation during testing. Work with your QA team and your translation and localization agency to incorporate pseudo-translation into your process.

Pseudo-translation in action!

Here's an example of a few strings from a C resource file, with their respective pseudo-translations in Japanese:

IDS_TITLE_OPEN_SKIN "Select Device"
IDS_TITLE_OPEN_SKIN "日本SイlイctDイvウcイ本日"
IDS_MY_OPEN "&Open"
IDS_MY_OPEN "日本&Opイn日"

In these strings, Japanese characters replace the vowels in all English words. After compilation, testers are easily able to detect corrupt characters (junk characters replacing the Japanese characters), or strings that remain fully in English (source strings still embedded in the code).

- **Check symbols for cultural connotations before you standardize.** Before launching your new release to your target markets, verify that all images and symbols are culturally appropriate for each locale. Do they lead to respective connotations to make the relevant impact? Do they convey the right message about your brand? Do they damage your brand in the target market? Your translation and localization agency or an independent reviewer can help you with a cultural audit.

Follow these guidelines to expedite software localization and speed your product launch. You'll reduce your testing, rework and other quality assurance costs. What's more, the culturally tailored, tighter product you'll launch will be more successful in your target markets.

About Argos

Argos Multilingual provides global language solutions. With over 30 years of experience, we serve clients in the high-tech, life sciences, human resources, and financial industries.

We make it easy for businesses to grow globally and connect with expert talent anywhere in the world. With production centers in Europe, the Americas, and Asia, we follow a strategy of building robust programs for continuous translation and localization.

You can expect a long-term and transparent partnership, backed by innovative solutions around technology, AI & data, creative content, and quality assurance.

For more information and contact details visit our website at www.argosmultilingual.com