Localization Checklist

Android and Google Play give you a worldwide audience for your app, with an addressable user base that's growing very rapidly in countries such as Japan, Korea, India, Brazil, Russia, and elsewhere.

To maximize your app's distribution potential and earn high ratings from users around the world, we strongly encourage you to localize your app.

Localization involves a variety of tasks throughout your app's development cycle, and advance planning is essential. Some of the tasks include translating your UI strings and localizing dates and times, layouts, text direction, and finally your Google Play store listing.

This document helps you identify key aspects of localization to prepare for and the tasks you'll need to perform, to get your app ready for a successful worldwide launch on Google Play.

CHECKLIST

SEE ALSO

Google Play Badge Builder
Device Art Generator
Translations in Google Play
ADT Translation Manager Plugin

1. Identify target languages and locales

A basic but important step in preparing for localization is identifying the countries where you will distribute your app and the languages spoken there. Google Play lets you distribute your app broadly to hundreds of countries, reaching users who speak a variety of languages.

For international users, you can manage your app on three main dimensions: country, locale, and language. Of those, language is the key consideration for localization, although locale is also significant because of differences in formats for dates, times, currencies, and similar information. Users control both the language and locale used on their Android devices and in turn those affect the display of your app, once installed.

Typically, you would decide which countries to target first, based on overall market size and opportunity, app category, competitive landscape, local pricing and financial factors, and so on. Then, based on your country targeting, you would determine the languages you need to support in your app.

You will need to decide when to localize into some or all of the languages in your targeted countries. In some countries it might make most sense to deliver an app in a major regional or international language only, rather than in all locally spoken languages. Similarly, based on overall market size, you might decide to deliver your app in only a small number of key languages and offer English or another language for other countries. You can add more languages in the future as your app's userbase grows.

Localizing your app is particularly important in countries where there is a large market opportunity and English or another international language is not widely used. Once you have identified your target languages, you can focus your development, translation, testing, and marketing efforts to these markets.

Related resources:

- Supported locations for distributing applications on Google Play. .

2. Design for localization

After you've determined your target languages for localization, assess what you'll need to do to support them in your app and plan the work early. Consider the vocabulary expansion, script requirements, character spacing and wrapping constraints, left-to-right and right-to-left support, and other potential factors in each language.

Design a single set of flexible layouts

As you create your layouts, make sure that any UI elements that hold text are designed generously. It's good to allow more space than necessary for your language (up to 30% more is normal) to accommodate other languages.

Also, elements should be able to expand horizontally or vertically to accommodate variations in the width and height of UI strings or input text. Your text strings should not overlap borders or the screen edge in any of your target languages.

If you design your UI carefully, you can typically use a single set of layouts for all of the languages you support. See Building a Flexible UI (/training/basics/fragments/fragment-ui.html) for more information.

Use alternative layouts where needed

In cases where your UI can't accommodate text in one of your target languages, you can create an alternative layout (/guide/topics/resources/providing-resources. html#AlternativeResources) for that language only. Android makes it easy to declare sets of layouts and other resources to load for specific languages, locales, screen sizes, and so on, simply by tagging them with the appropriate resource qualifiers.

Although you can use alternative layouts to work around isolated issues, they can also make your app harder to maintain over time. In general, using a single, more flexible layout is preferred.

Support RTL layouts and text

If you are distributing to countries where right-to-left (RTL) scripts are used, should consider implementing support for RTL layouts and text display and editing, to the extent possible.

Android 4.1 introduced limited support for bidirectional text, allowing apps to display and edit text in both left-to-right (LTR) and right-to-left (RTL) scripts. Android 4.2 added full native support for RTL layouts (http://android-developers.blogspot.fr/2013/03/native-rtl-support-in- android-42.html), including layout mirroring, so that you can deliver the same great app experience to all of your users.

At a minimum, for Android 4.2 users, it's simple to add basic RTL layout mirroring, which goes a long way toward meeting the needs of RTL users.

Use system-provided formats for dates, times, numbers, and currencies

Where your app specifies dates, times, numbers, currencies, and other entities that can vary by locale, make sure to use the system-provided formats, rather than app-specific formats. Keep in mind that not every locale uses the same thousands separator, decimal separator, or percent sign.

Android provides a variety of utilities for formatting and converting patterns across locales, such as `DateUtils` `(/reference/android/text/format/DateUtils.html)` and `DateFormat` `(/reference/java/text/DateFormat.html)` for dates; `String.format()` `(/reference/java/lang/String.html#format(java.lang.String, java.lang.Object...))` or `DecimalFormat` `(/reference/java/text/DecimalFormat.html)` for numbers and currency; `PhoneNumberUtils` `(/reference/android/telephony/PhoneNumberUtils.html)` for phone numbers; and others.

If you hard-code your formats based on assumptions about the user's locale, your app could encounter problems when the user changes to another locale. The easiest and most reliable approach is to always use system-provided formats and utilities.

Include a full set of default resources

Make sure that your app can run properly regardless of language or locale by providing a complete set of default resources. The app's default resources are those that are *not marked* with any language or locale qualifiers, for example those stored in `res/drawable/` and `res/values/`. If your app attempts to load a resource that isn't available in the current language or in the default set, the app will crash.

Whatever the default language you are using in your app, make sure that you store the associated layouts, drawables, and strings in default resource directories, without language or locale qualifiers.

Related resources:

- Native RTL Support in Android 4.2 — Blog post that explains how to support RTL in your UI.
- Quantity Strings (Plurals) — Developer guide describing how to work with string plurals according to rules of grammar in a given locale.
- `Locale` — Reference information about how to use locale data determine exactly what CLDR data or version of the Unicode spec a particular Android platform version uses.

3. Manage strings for localization

It's important to manage your app's UI strings properly, so that you deliver a great experience for users and make localization straightforward.

Move all strings into strings.xml

As you build your app, remember that it's a best practice to keep all of your UI strings in a single file that's easy to update and localize. Declare *all* of your strings as resources in a default `strings.xml` file. Do not hard-code any strings into your compiled code—hard-coded strings are much more difficult to extract, translate, and load properly.

If you keep all of your default strings in a `strings.xml` file, you can quickly extract them for translation, and once the translated strings are integrated back into your app with appropriate qualifiers, your app can load them without any changes to your compiled code.

If you generate images with text, put those strings in `strings.xml` as well, and regenerate the images after translation.

Follow Android guidelines for UI strings

As you design and develop your UI, make sure that you pay close attention to *how* you talk to your user. In general, use a succinct and compressed style (/design/style/writing.html) that is friendly but brief, and use a consistent style throughout your UI.

Make sure that you read and follow the Android Design recommendations for writing style and word choice (/design/style/writing.html). Doing so will make your app appear more polished to the user and will help users understand your UI more quickly.

Also, always use Android standard terminology wherever possible—such as for UI elements such as "Action Bar," "Options Menu," "System Bar," "Notifications," and so on. Using Android terms correctly and consistently makes translation easier and results in a better end-product for users.

Provide sufficient context for declared strings

As you declare strings in your `strings.xml` file, make sure to describe the context in which the string is used. Add comments before each string that may need clarification. This information will be invaluable to translators and will help you manage your strings more effectively over time.

For example, background information to provide might include:

- What is this string for? When/where is it presented to the user?
- Where is this in the layout? For example, if it's a button, translations are less flexible than if it were a text box.

Here's an example:

```
<!-- The action for submitting a form. This text is on a button that can fit 30 chars -->
<string name="login_submit_button">Sign in</string>
```

Mark message parts that should not be translated

Often strings contain contain text that should not be translated to other languages. Common examples might be a piece of code, a placeholder for a value, a special symbol, or a name. As you prepare you strings for translation, look for and mark text that should remain as-is, without translation, so that translators do not change it.

To mark text that should not be translated, use an `<xliff:g>` placeholder tag. Here's an example tag that ensures the text "%1$s" will not be changed during translation (otherwise it could break the message):

```
<string name="countdown">
    <xliff:g id="time" example="5 days>%1$s</xliff:g>until holiday
</string>
```

When you declare a placeholder tag, always add an `id` attribute that explains what the placeholder is for. If your app will later replace the placeholder value, be sure to provide an example attribute to clarify the expected usage.

Here are some more examples of placeholder tag usage:

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <!-- Example placeholder for a special unicode symbol -->
    <string name="star_rating">Check out our 5
        <xliff:g id="star">\u2605</xliff:g>
    </string>
    <!-- Example placeholder for a for a URL -->
    <string name="app_homeurl">
        Visit us at <xliff:g id="application_homepage">http://my/app/home.html</xliff:g>
    </string>
    <!-- Example placeholder for a name -->
    <string name="prod_name">
        Learn more at <xliff:g id="prod_gamegroup">Game Group</xliff:g>
    </string>
    <!-- Example placeholder for a literal -->
    <string name="promo_message">
        Please use the "<xliff:g id="promotion_code">ABCDEFG</xliff:g>" to get a discount.
    </string>
    ...
</resources>
```

Related resources:

- String Resources — Developer guide explaining how to use string resources in your UI.
- Writing Style — Android Design guidelines for voice and style in your UI.
- XML Localisation Interchange File Format (XLIFF) — Background information on XLIFF.

4. Translate UI strings and other resources

Translating your app's UI strings and resources to your target languages is the key phase of localization, and it's the one that requires the most care and planning.

In general, it's recommended to work with a professional translator to ensure that the work goes smoothly, stays on schedule, and results in a high-quality product that will enhance the value of your app. If you are considering machine translations as an alternative, keep in mind that automated translations are less reliable than high-quality professional translations and may not produce as good an experience for your users.

Prepare for translation

Getting high-quality translation output depends in part on your input. To get ready for translation, make sure that your `strings.xml` file is well organized, well commented, and accurate.

Here are some ways to prepare your strings for translation:

- Make sure your strings are formatted correctly and consistently.
- Follow the strings recommendations listed in Manage strings for localization, above.
- Clean up the `strings.xml` file and remove unused strings.
- Place comments in the file to identify the owner, origin, and the version of the file, as well as any special instructions for translators.
- Identify existing translations, if any, and include those in an outgoing zip file or other package that you will send to translators.
- Identify drawables or other resources that require translation and include them in the outgoing package for translators.

Additionally, consider translating your app's store listing details — app title and description, release notes, and so on — as well as other international marketing materials.

- Create a terminology list that explains the meaning and usage of key terms used in your product, your market, or the underlying technology. Add the list to the outgoing package.

Send your strings for translation

Early in the development cycle, contact professional translation vendors for your target languages to get an idea of cost, lead time required, turnaround time, and so on. Then select a vendor and secure their services, making sure to include multiple iterations in the cost as a safeguard. Google Play can help you do this — see Purchase professional translations (#gp-trans), below.

As soon as your app's UI strings and design are stable, work with your development team to extract all of the strings and other resources from the app and package them together for the translator. If appropriate, you can version the outgoing package for later identification.

When the outgoing package is ready, send it to the translator or share it with them over a cloud platform such as Google Drive. Keep a record of what you sent and when you sent it, to cross-reference against returning translations and billing invoices from the translator.

When your translations are complete, take a preliminary look at the translations. Check that all files were translated, check for potential encoding issues, and make sure that declaration formats are intact.

If everything looks good, carefully move the localized directories and files back into your app's resources. Make sure to tag the directories with the appropriate language and locale qualifiers so that they'll later be loaded properly.

After the translations are merged back into your app, start testing the localized app (#testing).

Purchase professional translations through the Developer Console

Google Play can help you quickly find and purchase translations of your app. In the Developer Console, you can browse a list of third-party vendors who are pre-qualified by Google to offer high-quality translation at competitive prices. You can upload the strings you want translated, select the languages you want to translate into, and select your translation vendor based on time and price.

Once you've purchased translations, you'll receive an email from your vendor. Your translations are a direct business agreement between you and your vendor; you'll need to work directly with the vendor to manage the translation process and deliverables and resolve any support issues.

Join the translation pilot

Google Play is offering translation services as part of a pilot program. If you're interested, sign up on the APK page in your Developer Console.

If you join, also try the ADT Translation Manager Plugin (/sdk/installing/installing-adt.html#tmgr), which makes it easy to upload your strings to the Developer Console and download translations right into your project.

5. Test your localized app

Once you've received your translated strings and resources and moved them back into your app, you need to test the app to make sure that it's ready for distribution to your international users.

Manual testing can help you discover localization issues in your layouts and strings that can affect user satisfaction and, ultimately, your app's user rating.

Set up a test environment

To test your localized app, you'll need to set up an environment consisting of multiple devices (or virtual devices) and screen sizes, based on the markets and form factors you are targeting. Note that the range of devices in specific regions might be different. If possible, match your test devices to the actual devices likely to be available to users.

Look for common localization issues

On each test device, set the language or locale in Settings. Install and launch the app and then navigate through all of the UI flows, dialogs, and user interactions. Enter text in inputs. Some things to look for include:

- Clipped text, or text that overlaps the edge of UI elements or the screen
- Poor line wrapping
- Incorrect word breaks or punctuation
- Incorrect alphabetical sorting
- Incorrect layout direction or text direction
- Untranslated text — if your default strings are displayed instead of translated strings, then you may have overlooked those strings for translation or marked the resources directory with an incorrect language qualifier.

For cases where your strings have expanded in translation and no longer fit your layouts, it's recommended to simplify your default text, simplify your translated text, or adjust your default layouts. If none of those resolves the issue, you can create a custom layout for the language.

Test for default resources

After you've tested your app in all of your supported languages and locales, make sure to test it again in an *unsupported language* and locale. This will help you make sure that your app includes a full set of default strings and resources, so that your app is usable to all users, regardless of their preferred language.

Review with native-language speakers

During or after testing, it's recommended that you let native speakers review your localized app. One way to do that is through beta testing with regional users — Google Play can help you do this. See Plan a beta release (#beta) for more information.

Prepare for international launch

Getting your app translated is a key part of localization, but to help your product attract users and gain visibility, you should prepare for launch in your target countries and create a broader launch and marketing plan for international users.

Localize your Google Play listing

If you want your app to be successful in international markets, it's essential to localize your Google Play store listing. You can manage your localized listing in the Developer Console.

Well before launch, decide on your app title, description, promotional text, marketing names and programs, and other text and images. Send your listing text and images for translation early, so that you have them ready when beta testing begins. When your translated text is available, you can add it through the Developer Console.

Localize your Google Play listing

Highlight what's great about your app to all of your users! Localize your listing in the Developer Console:

- App title and description
- App screenshots on phones and tablets
- Promotional graphics and videos.

Also, since you've made the effort to create a great localized app, let users know about it! Take screenshots of your UI in each language, for phones and 7- and 10- inch tablets. You can upload screenshots to the Developer Console for each language you support. These will be of great value to users browsing your app listing in other languages.

It's also essential to create localized versions of your promotional graphics and videos. For example, your app's feature graphic might include text that should be translated, for maximum effectiveness, or you might want to take a different visual approach in one country than you do in another. You can create different versions of your promotional graphics for each language and upload them to the Developer Console. If you offer a promotional video, you can create localized versions of it and then add a link to the correct localized video for each language you support.

Plan a beta release in key countries

Before launching your app, it's always valuable to get real-world feedback from users — even more so when you are launching an app in a new language, country, or region. In those cases, it's highly recommended that you distribute a pre-release version of your app to users across your key markets and provide an easy means for them to provide feedback and report bugs.

Easy beta testing

Google Play now lets you set up groups of alpha and beta testers, anywhere around the world. Check out this powerful feature next time you sign in to the Developer Console.

Google Play can help you set up a beta program for your app. After you sign in to the Developer Console and upload your APK, you can set up groups of users for alpha testing and beta testing the app. You can start with a small group of alpha testers, then move to a larger group of beta testers. Once users are added, they access your app's store listing and install the app. User feedback from alpha and beta testers goes directly to you and is not posted as public reviews.

The feedback you receive will help you adjust your UI, translations, and store listing to ensure a great experience for users.

Plan for international marketing

For highest visibility across countries, consider an international marketing or advertising campaign. The scope of the campaign might vary based on the budget you can support, but in general it's cost-effective and productive to do regional or country-specific marketing at launch and after.

Create localized Google Play badges

If you are preparing international marketing, make sure to include a localized Google Play badge (/distribute/googleplay/promote/badges.html) to tell users you're on Google Play. You can use the badge generator to quickly build localized badges that you can use on web sites or marketing materials. High-resolution assets are also available.

Create Localized Device Art

If you feature product shots of your app running on Android devices, make sure that those shots look great and reflect the latest in Android devices. To help you create high-quality marketing materials, use the drag-and-drop Device Art Generator (/distribute/promote/device-art.html) to quickly frame your screen shot on a Nexus device.

Check your Optimization Tips

As you prepare for launch, make sure to sign into the Developer Console and check your app's Optimization Tips. The Optimization Tips let you know when you are missing parts of your localized store listing and provide other helpful reminders for a successful localized launch.

Support International Users after Launch

After you launch your app internationally, you should be prepared to support users in a variety of languages and time zones. The extent of your international user support depends on your budget, but at a minimum you should watch your ratings, reviews, and download stats carefully after launch.

Here are some suggestions:

- Use the app stats in the Developer Console to compare your downloads, installs, and uninstalls, and ratings across languages and countries—If your downloads or ratings are not keeping up in specific languages or countries, consider options for improving your product or changing your marketing approach.
- Check reviews regularly—Google Play translates all user reviews for you, so you can stay in touch with how international users feel about your app, what features they like and what issues are affecting them. By watching reviews, you can spot technical issues that may affect many users in a particular country, then fix and update your app.
- Respond to reviews if possible—It's good to engage with international users in their language or a common language if possible. If not, you can try using translation tools, although results may not be predictable. If your app gets very popular in a language, consider getting support help from native-language speakers.
- Make sure there's a link to any support resources on your web site. Consider setting up language-specific user groups, Google+ communities, or other support forums.

By following these practices for localizing your app, promoting and marketing to international users, and providing ongoing support, you can attract many new users to your app and maintain their loyalty.

Make sure to read the Launch Checklist (/distribute/googleplay/publish/preparing.html) to learn more about how to plan, build, and launch your app on Google Play.