Google

Android Internationalization

Unicode Conference #42

September 10, 2018

Mihai Niță

- A bit about me
  - I18N / L10N since 1997
  - Adobe , Netflix, Google

- Rules of engagement
  - Ask questions at any time
  - It is OK to interrupt

Intro

First look – what Android offers

Internationalization APIs (I18N)

- Formatters: number, currency, date, time
- Collation
- MessageFormat
- etc.

Localization framework (L10N)

- Mechanisms for locale aware resources loading, with matching, fallback, ...
- Used to localize the OS itself and all the applications (Google & 3rd party)

Standard widgets

I18N – APIs (not an exhaustive list)

JDK equivalent APIs (not an exhaustive list)

- Formatters, extending java.text.Format (for instance DateFormat, NumberFormat, MessageFormat)
- Others (for instance Collator, BreakIterator, Normalizer, Locale, Calendar)

Android additions

- android.text.BidiFormatter, android.text.format.DateFormat, android.text.format.DateUtils, android.text.format.Formatter

ICU4J (most of it) is public API since Android N (under android.icu)

I18N – Android implementation

Some of the i18n functionality is implemented “from scratch” (i.e. Locale)

The “tricky areas” (formatters, collators, code page conversion) use ICU4C

- ICU4C is the C/C++ version of ICU (International Components for Unicode)
- Using JNI (Java Native Interface) to convert back and forth
- ICU4C is not accessible, not even from the NDK (Native Development Kit)

Starting with Android Nougat, ICU4C is (mostly) replaced by ICU4J.

Localization

Resource folders (not only strings, but also layouts, drawables, styles, etc.)

Using the localized resources is simple:

```
Button button = (Button) findViewById(R.id.submitButton);
button.setText(R.string.submit);
```

or

```
<Button
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:id="@+id/submitButton"
  android:text="@string/submit" />
```

Let’s build something...

What to do — translating your app

- Move all strings to resources
- Never concatenate, always use placeholders (nothing shorter than a sentence)
- For placeholders use %1$d, or {0} or (best) {COUNT}, not %d
- Don’t reuse strings
- Include context for translators (like comments or screen-shots)
- Mark non-localizable strings (<string … translatable="false">…</string>)
- Use xliff tags:

```
<item quantity="other">Your card expires on <xliff:g id="shortExpirationDate"
example="Nov 12">%1$s</xliff:g>.</item>
```

What to do — good (i18n) behavior

- Use formatters, collators, etc.
- Use system pickers, avoid parsing
- Other things might also be localizable: images, styles, sounds, fonts, layouts
- Use the plurals / gender library (not the default Android way)
  https://android.googlesource.com/platform/external/messageformat/+/master/

What to do — plural and gender

The default Android way:

```
<plurals name="msgSongsAvailable">
  <item quantity="one">There is one song found.</item>
  <item quantity="other">There are %d songs found.</item>
</plurals>
…
int songCount = 42;
String songsFound = getResources().getQuantityString(
      R.plurals.msgSongsAvailable,
      songCount, songCount);
```

What to do — plural and gender

~~The default Android way:~~

```
~~<plurals name="msgSongsAvailable">~~
~~    <item quantity="one">There is one song found.</item>~~
~~    <item quantity="other">There are %d songs found.</item>~~
~~</plurals>~~
...
~~int songCount = 42;~~
~~String songsFound = getResources().getQuantityString(~~
~~        R.plurals.msgSongsAvailable,~~
~~        songCount, songCount);~~
```

DON'T USE THIS!!!

What to do — plural and gender

The recommended way (small library, was extracted from ICU):
https://android.googlesource.com/platform/external/messageformat/+/master/

```
<string name="msgSongsAvailable">
  {count, plural,
    =1     {There is one song found.}
    other {There are %d songs found.}
  }
</string>
…
int songCount = 42;
String msg = getResources().getString(R.plurals.msgSongsAvailable);
String songsFound = MessageFormat.formatNamedArgs(msg, "count", songCount);
```

Bidi – Native RTL support since Android 4.2 (API 17)

- Add android:supportsRtl="true" to the <application> element in the manifest file
- Change all of your app's "Left/Right" layout properties to new "Start/End" equivalents. For backward portability, add them, don't change them.
  For example, android:paddingLeft should become android:paddingStart
- APIs and flags
  - android:layoutDirection, android:textDirection: recommend using "locale" and setting them for (almost) all layouts (is a good default, but "it depends")
  - android:textAlignment: uses start / end concepts rather than left / right
  - getLayoutDirectionFromLocale(locale)
  - android.R.attr.autoMirrored: not a global change
    (see Material Design – Bidirectionality)
  - ldrtl resource qualifier

BidiFormatter

- Use android.text.BidiFormatter to wrap placeholders in text
- Transparently inserts Unicode formatting characters to protect the placeholder
- If the string may have different direction from placeholder: numbers, usernames, dates, etc.
- The defaults are good for most use cases:

```
BidiFormatter bidiFormatter = android.text.BidiFormatter.getInstance();
String formattedString = MessageFormat.format(templateString, ...
    bidiFormatter.unicodeWrap(placeholderValue), ...);
```

- Support Library copy: android.support.v4.text.BidiFormatter
  (also supports CharSequence)

Get your app ready for Nougat and newer

Use the new android.os.LocaleList APIs to make you app “smarter”

If the default folder does not contain English, make a copy of it in the language folder (so if values contains German, make a copy of it in values-de)

Continue using tl locale id for Filipino, even if fil is now supported (since API 21)

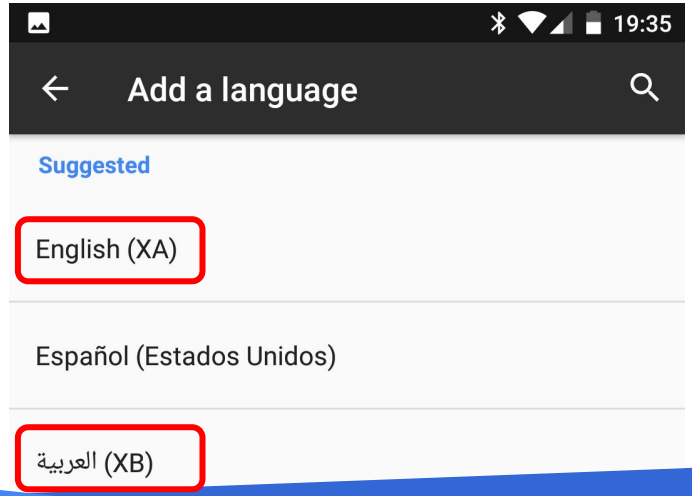
Don’t expect zh-TW to fallback to zh (zh-CN does, but zh-TW does not :-)

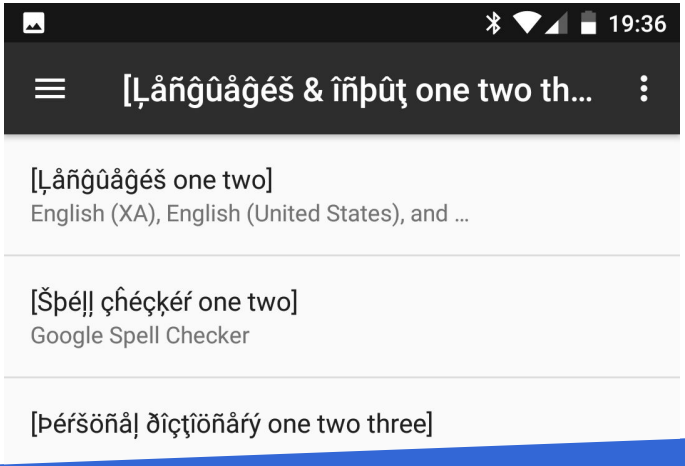
Add resConfigs in android.defaultConfig:

```
android { … defaultConfig {
    resConfigs "nb", "nn", "in", "iw", "tl", "zh", "zh-rTW", "b+es+419", "b+sr+Latn"
} … }
```

I18n Validation & Testing

- Use pseudo-translation (en-XA, ar-XB)
- Android Studio's code analysis
- Input using some "unusual" keyboards (Chinese or Japanese IMEs)





But wait, there’s more!

- ICU4J is public API (android.icu)
- Open Type font variation support
- Downloadable Fonts
- EmojiCompat support library
- Builder for Typeface
- XML Fonts

With each Android version

- The text engine is getting better (new scripts, hyphenation, typography)
- Unicode data updated (including new Emoji(s))
- ICU version updated
- More languages supported
- More fonts added, existing fonts might get updates

Found bugs? Have suggestions?

Let us know at:
https://source.android.com/setup/contribute/report-bugs

Localizing with Resources
https://developer.android.com/guide/topics/resources/localization.html

Localization Checklist
https://developer.android.com/distribute/tools/localization-checklist.html

String Resources
https://developer.android.com/guide/topics/resources/string-resource.html

Android multilingual support
https://developer.android.com/guide/topics/resources/multilingual-support.html

The plurals / gender library
https://android.googlesource.com/platform/external/messageformat/+/master/

Material Design – Bidirectionality
https://material.io/design/usability/bidirectionality.html

Material Design – Typography – Language support
https://material.io/design/typography/language-support.html

Q & A

THANK YOU