ANDROID STRING RESOURCES

MJ , Android Engineer at STRV

STRV

ANDROID STRING RESOURCES

Rich String Resources for Compose

01

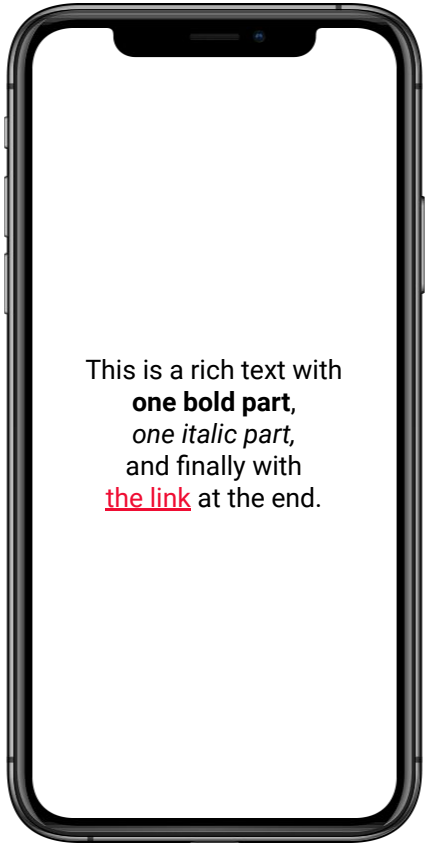
Cross Platform Shared Strings

02

STRING RESOURCES COMPOSE

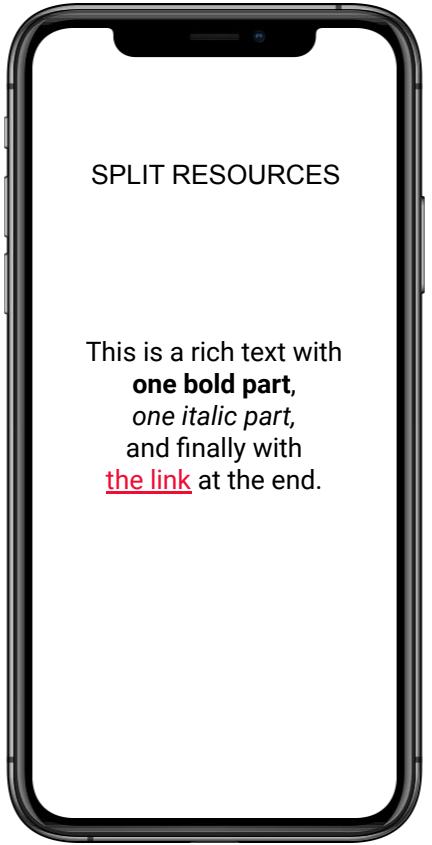
01


RICH TEXT


This is a rich text with
one bold part,
one italic part,
and finally with
the link at the end.

Benefits of Rich String Resources

- Developer is not forced update the UI code with every tweaking of the copy
- Instantly clear, what the text emphasis is on
- We can move this definition to be remotely adjusted (Teaser to part 2)




```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="sample_text_plain_1">This is a rich text with </string>
    <string name="sample_text_plain_2">one bold part, </string>
    <string name="sample_text_plain_3">one italic part, </string>
    <string name="sample_text_plain_4">and finally with </string>
    <string name="sample_text_plain_5">the link</string>
    <string name="sample_text_plain_6"> at the end.</string>

</resources>
```


```
@Composable
private fun SplitResourceText() {
    val context = LocalContext.current

    val richText = buildAnnotatedString { this: AnnotatedString.Builder

        append(stringResource(id = R.string.sample_text_plain_1))

        withStyle(style = SpanStyle(fontWeight = FontWeight.Bold)) { this: AnnotatedString.Builder
            append(stringResource(id = R.string.sample_text_plain_2))
        }

        withStyle(style = SpanStyle(fontStyle = FontStyle.Italic)) { this: AnnotatedString.Builder
            append(stringResource(id = R.string.sample_text_plain_3))
        }

        append(stringResource(id = R.string.sample_text_plain_4))

        pushStringAnnotation(WEB_LINK_TAG, WEB_LINK)
        withStyle(
            style = SpanStyle(
                color = Color.Red,
                textDecoration = TextDecoration.Underline,
            )
        ) { this: AnnotatedString.Builder
            append(stringResource(id = R.string.sample_text_plain_5))
        }
        pop()

        append(stringResource(id = R.string.sample_text_plain_6))

    }
```


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="sample_text_plain">This is a rich text with one bold part,
        one italic part, and finally with the link at the end.</string>
    <string name="sample_text_plain_bold">one bold part</string>
    <string name="sample_text_plain_italic">one italic part</string>
    <string name="sample_text_plain_link">the link</string>

</resources>
```

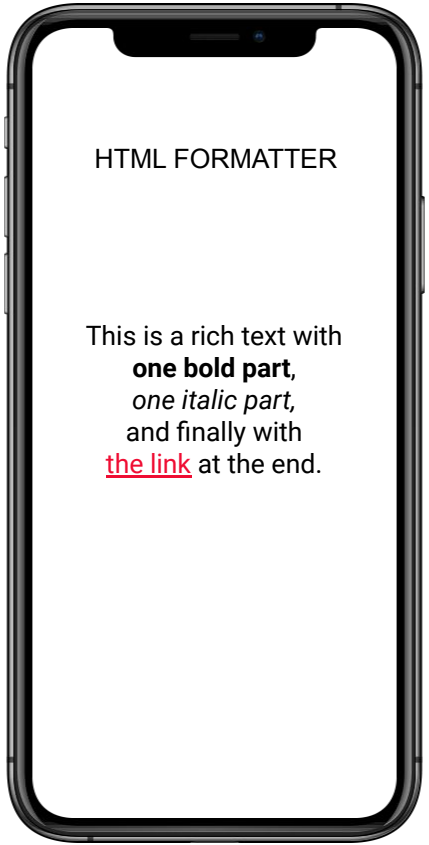

```
@Composable
private fun DuplicatedResourceText() {
    val context = LocalContext.current

    val plainText = stringResource(id = R.string.sample_text_plain)
    val richText = buildAnnotatedString { this: AnnotatedString.Builder
        append(plainText)

        val boldPart = stringResource(id = R.string.sample_text_plain_bold)
        addStyle(
            style = SpanStyle(fontWeight = FontWeight.Bold),
            start = plainText.indexOf(boldPart),
            end = plainText.indexOf(boldPart) + boldPart.length
        )

        val italicPart = stringResource(id = R.string.sample_text_plain_italic)
        addStyle(
            style = SpanStyle(fontStyle = FontStyle.Italic),
            start = plainText.indexOf(italicPart),
            end = plainText.indexOf(italicPart) + italicPart.length
        )

        val linkPart = stringResource(id = R.string.sample_text_plain_link)
        addStringAnnotation(
            tag = WEB_LINK_TAG, WEB_LINK,
            start = plainText.indexOf(linkPart),
            end = plainText.indexOf(linkPart) + linkPart.length
        )
        addStyle(
            style = SpanStyle(
                color = Color.Red,
                textDecoration = TextDecoration.Underline,
            ),
            start = plainText.indexOf(linkPart),
            end = plainText.indexOf(linkPart) + linkPart.length
        )
    }
```




```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="sample_text_plain">
        <![CDATA[
        This is a rich text with one <b>bold part</b>,
        one <i>italic part</i>,
        and finally with the <a href="sample_link">link at the end.</a>
        ]]>
        </string>

</resources>
```

What if we feel that using HTML in XML resources this way is not the good one.


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="sample_text_custom">This is a rich text with **one bold part**,
        and **another bold part**.</string>

</resources>
```


```
fun String.withBoldParts(
    boldDelimiter: String = "**",
): AnnotatedString {
    val parts = this.split(boldDelimiter)
    return buildAnnotatedString { this: AnnotatedString.Builder
        parts.forEachIndexed { index, part ->
            if (index % 2 != 0) {
                withStyle(style = SpanStyle(fontWeight = FontWeight.Bold)) {
                    append(part)
                }
            } else {
                append(part)
            }
        }
    }
}

@Composable
private fun CustomResourceText() {

    val richText = stringResource(id = R.string.sample_text_custom).withBoldParts()

    Text(text = richText)
}
```

What if we want to use some standardized language in string definition.

What if we want to use some more complex formatting cases.

MARKDOWN COMPOSER

Markdown composer for Compose UI

- Introduced by Erik Hellman
- Easy to integrate and adjust
- Using commonmark for it’s rendering

Commonmark language

- Strongly defined markdown language
- Highly compatible specification of markdown


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="sample_text_markdown">
        This is a rich text with **one bold part**,
        *one italic part*,
        and finally with [the link](https://jenicek.dev)
        at the end.</string>
</resources>
```


```
@Composable
fun MarkdownResourceCard() {
    ResourceCard(titleRes = "MARKDOWN RESOURCE") {
        MarkdownResourceTextWithDefaultTheme()
    }
}
```

Michal *

```
@Composable
private fun MarkdownResourceTextWithDefaultTheme() {
    val context = LocalContext.current
    MDDocument(
        stringResId = "This is a rich text with **one bold part**, *one italic...",
    )
    { link ->
        context.navigateToWeb(link)
    }
}
```


```
data class MarkdownTheme(
    val colors: Colors,
    val textStyle: TextStyle,
    val linkTextStyle: TextStyle,
    val codeTextStyle: TextStyle,
    val strongEmphasisTextStyle: TextStyle,
    val emphasisTextStyle: TextStyle,
    val fencedCodeBlockTextStyle: TextStyle,
    val orderedListTextStyle: TextStyle,
    val bulletListTextStyle: TextStyle,
    val paragraphTextStyle: TextStyle,
    val blockQuoteTextStyle: TextStyle,
)
```


```
val DefaultMarkdownTheme: MarkdownTheme
    @Composable
    get() = MarkdownTheme(
        colors = MaterialTheme.colors,
        textStyle = TextStyle(),
        linkTextStyle = TextStyle(
            color = MaterialTheme.colors.primary,
            textDecoration = TextDecoration.Underline
        ),
        codeTextStyle = TextStyle(
            fontFamily = FontFamily.Monospace
        ),
        strongEmphasisTextStyle = TextStyle(
            fontWeight = FontWeight.Bold
        ),
        emphasisTextStyle = TextStyle(
            fontStyle = FontStyle.Italic
        ),
        fencedCodeBlockTextStyle = TextStyle(fontFamily = FontFamily.Monospace),
        orderedListTextStyle = MaterialTheme.typography.body1,
        bulletListTextStyle = MaterialTheme.typography.body1,
        paragraphTextStyle = MaterialTheme.typography.body1,
        blockQuoteTextStyle = MaterialTheme.typography.body1
    )
```


```
@Composable
private fun MarkdownResourceTextWithCustomTheme() {
    val context = LocalContext.current
    MDDocument(
        stringResId = "This is a rich text with **one bold part**, *one italic...",
        markdownTheme = KotoxMarkdownTheme,
    )
    { link ->
        context.navigateToWeb(link)
    }
}
```

Michal *

```
@Composable
private fun MarkdownResourceTextWithCustomThemeExtra() {
    val context = LocalContext.current
    MDDocument(
        stringResId = "This is a rich text with **one bold part**, *one italic...",
        markdownTheme = KotoxMarkdownTheme.copy(
            linkTextStyle = KotoxMarkdownTheme.linkTextStyle.copy(
                color = Color.Red
            )
        ),
    )
    { link ->
        context.navigateToWeb(link)
    }
}
```

https://github.com/kotoMJ/kotox-android/tree/main/

kotox-mobile-strings

SHARED STRINGS 02

LOCAL STRINGS

The definition

- In share text document
  - Jira
  - Google doc
  - …
- In the design system
  - Figma
  - Zeplin
  - Sketch

Downsides

- Out of sync cross platform
- Complicated cross platform copy management

SHARED STRINGS

Localisation platform

- Phrase
- **PoEditor**
- Crowding
- Lokalise
- …

Benefits

- **Define source of truth** - the localisation platform
- Copy manageable by variety of roles:
  - Mobile Engineer
  - QA Engineer
  - Copywriter
  - CEO :-)
- Strings might be imported to Android codebase in the batch with minimal engineering effort.

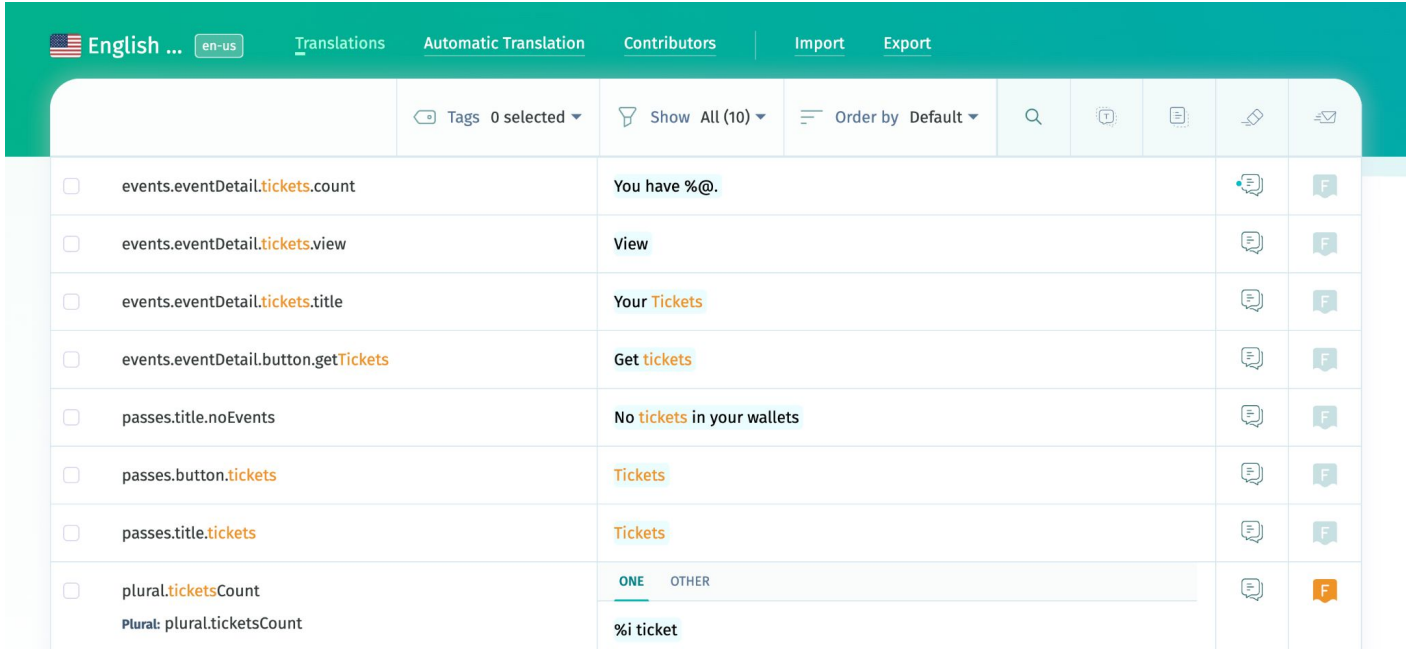
CROSS PLATFORM COMMON FACTORS

https://poeditor.com/kb/how-to-keep-ios-and-android-strings-in-the-same-localization-project

- **Define terms/keys naming**
  - Naming conventions
  - Separators
- **Define placeholders**
  - Parameter & plural placeholders
  - https://poeditor.com/kb/placeholder-validation
- Eventually **define tagging system**
  - separate group of terms/keys .

INIT THE LOCALISATION PLATFORM

- Create the project
- Populate terms/keys & translations



CONNECTING PLATFORMS

- **Open API**
- Figma Integration
- Script
- Plugin

Figma
Design

Figma is an online UI and UX design app, with design, prototyping, and code-generation tools.

OpenAPI
Dev

A specification to describe, produce, consume, and visualize RESTful web services.

Poesie is the **POE**ditor **S**tring **I**nternationalization **E**xtractor.

PoEditor Android Gradle Plug-in

JitPack 3.4.0

Simple plug-in that eases importing PoEditor localized strings to your Android project.

MIGRATE APPS WITH LOCAL STRINGS

- It’s never too late to migrate to localisation platform
  - It’s a bit painful to migrate
  - It’s a joy to work with string resources after the migration.
- Migration steps:
  - Choose the **pivot platform** app
  - Define **Cross Platform Factors**
  - **Import strings** from pivot app **to localisation platform**
  - **Refactor keys in non-pivot app**
  - Prepare **import mechanism** according to Cross Platform Factors.

ADVANCED SOLUTION

Use advanced solution when platforms can't share the project across client platforms

- PoEditor - share translations cross project
  - https://poeditor.com/help/how_to_use_translation_memory
- Apply **custom transformation** using custom solution
  - Avoid complexity with multiple projects in the localisation platform
  - Stay flexible for any other customizations or localisation platform switch

IMPORT LOGIC

- Download resource file from API

  https://poeditor.com/docs/api#openapi
- Post-Process the resource file
  - Avoid processing iOS resources

    (by `_ios` suffix)
  - Fix resource key

    separated by **dots** -> **underscore**
  - Remove Android specific key part

    key suffix `_android` suffix
  - Transform iOs placeholders to Android

    https://poeditor.com/kb/placeholder-validation
  - Determine target module/file

    based on resource key

PoEditor Android Gradle Plug-in

JitPack 3.4.0

Simple plug-in project.

- poeditor
  - api
    - OkHttpClientExtensions.kt
    - PoEditorApi
    - PoEditorApiController.kt
    - PoEditorApiModel.kt
  - xml
    - AndroidXmlWriter.kt
    - Constants.kt
    - DocumentExtensions.kt
    - StringExtensions.kt
    - XmlCdataPostProcessor.kt
    - XmlPluralStringPostProcessor.kt
    - XmlPostProcessor
    - XmlSingleStringPostProcessor.kt
    - XmlTextContentPostProcessorExtensions.kt
  - PoEditorImportController
  - ResourceModuleMapping.kt

CUSTOM ANDROID PLUGIN

```kotlin
fun Project.poEditorPlugin(projectDir: String) {

    tasks.register( name: "importPoEditorStrings") { this
        val poEditorApiToken =
            System.getenv( name: "POEDITOR_API_TOKEN")
                ?: project.property( propertyName: "POEDITOR_API_TOKEN")
        val poEditorProjectId = System.getenv( name: "POEDITOR_PROJECT_I
            ?: project.property( propertyName: "POEDITOR_PROJECT_ID") as

        doLast { this: Task
            PoEditorImportController(
                poEditorApiUrl = "https://api.poeditor.com/v2/",
                poEditorApiToken = poEditorApiToken,
                poEditorProjectId = poEditorProjectId
            ).executeImport(
                projectDir,
            )
        }
    }

}
```

PoEditor Android Gradle Plug-in

JitPack 3.4.0

Simple plug-in

project.

- poeditor
  - api
    - OkHttpClientExtensions.kt
    - PoEditorApi
    - PoEditorApiController.kt
    - PoEditorApiModel.kt
  - xml
    - AndroidXmlWriter.kt
    - Constants.kt
    - DocumentExtensions.kt
    - StringExtensions.kt
    - XmlCdataPostProcessor.kt
    - XmlPluralStringPostProcessor.kt
    - XmlPostProcessor
    - XmlSingleStringPostProcessor.kt
    - XmlTextContentPostProcessorExtensions.kt
  - PoEditorImportController
  - ResourceModuleMapping.kt

CUSTOM ANDROID PLUGIN

https://github.com/kotoMJ/kotox-android/tree/main/build-logic/convention/src/main/kotlin/cz/kotox/android/poeditor

```kotlin
fun Project.poEditorPlugin(projectDir: String) {

    tasks.register( name: "importPoEditorStrings") { this: Task
        val poEditorApiToken =
            System.getenv( name: "POEDITOR_API_TOKEN")
                ?: project.property( propertyName: "POEDITOR_API_TOKEN") as String
        val poEditorProjectId = System.getenv( name: "POEDITOR_PROJECT_ID")
            ?: project.property( propertyName: "POEDITOR_PROJECT_ID") as String

        doLast { this: Task
            PoEditorImportController(
                poEditorApiUrl = "https://api.poeditor.com/v2/",
                poEditorApiToken = poEditorApiToken,
                poEditorProjectId = poEditorProjectId
            ).executeImport(
                projectDir,
            )
        }
    }

}
```

THANK YOU!

MJ / michal.jenicek@strv.com

STRV

QUESTIONS

STRV