GENERATING RESOURCES

1. Why we should generate resources?

UIColor problem:

Currently in IB colors are represented by RGB values and in code we are using UIColor extension that provides some additional color used in project. Main problem is that we don't have one source of colors. If we will generate colors from Assets catalog we will be able to use color by name in IB and code, and if we change RGB value, without name changes, it will change color automatically in whole project.

Additional :

Also we can name **colors by specification**. For example instead of "red" use "alert_color". In future we will be able to **change** RGB/HEX of that **color without renaming** of color in whole project. But this will cause additional work from designer in order to name **colors by specification**.

UIImage problem:

When we need to create image in code we need to call **UIColor(named: "some name")**. In this case we **can write incorrect name** or in future **rename old image with new name, but forget to change name in code**. As result some images can be absent on screen.

Generation of images to struct will prevent from this situation, it will show us error, that we are using incorrect name of resource, but problem will be present in xib/storyboard files.

UIFont problem:

Situation with font is different, but if we are using swiftgen, why don't use all features of it.

2. SwiftGen integration

a. Install swiftgen via homebrew
b. Download latest version from link
c. Unzip it in the root directory of your project

d. In project Build Phases create new script and add following to script body:

"${PROJECT_DIR}/swiftgen/bin/swiftgen"

Notes:

After unzip folder name will be **SwiftGen**, you can **rename** it to **swiftgen or change script body according to the name of folder.**

3. Create swiftgen.yml and add following:

```
input_dir: {PROJECT_NAME}/Supporting Files/
output_dir: {PROJECT_NAME}/Generated/
strings:
  inputs: Base.lproj/Localizable.strings
  outputs:
    - templateName: structured-swift4
      output: Localization.swift
      params:
        enumName: L
fonts:
  inputs: Fonts
  outputs:
    - templateName: swift4
      output: Fonts.swift
      params:
        enumName: F
xcassets:
  - inputs: Images/Assets.xcassets
    outputs:
      - templatePath: swift4_images
        output: Images.swift
        params:
          enumName: I
  - inputs: Colors/Colors.xcassets
    outputs:
      - templateName: swift4_colors
        output: Colors.swift
        params:
          enumName: CLR
```

Notes:

swift4_images and **swift4_colors** are custom templates that can be downloaded from link, after that they should be copied to */usr/local/share/SwiftGen/templates/xcassets* folder

4. Rules

a. Create separate asset catalog for colors and place it at path *{PROJECT_NAME}/Supporting Files/Colors*
b. Create folder for fonts and place it at path *{PROJECT_NAME}/Supporting Files/Fonts*
c. Use default assets catalog for images and place it at path *{PROJECT_NAME}/Supporting Files/Images*

5. Result

Fonts:

For IB usage of fonts is not changes, but in code we can write following ***F.NunitoSans.black.font(size: 24)*** instead of ***UIFont.nunitoSansBlack(withSize: 24)***

Images:

For IB usage of images assets is not changes, but in code we can write following ***I.unselectedPageIndicator.image*** instead of ***UIImage(named: "unselected_page_indicator")***

Colors:

Check *Image 1* to see how to add new color to xcassets and *Image 2* to see how to use it in IB and *Image 3* to see how it displayed when applied, in code we can use following ***Clr.customBlack.color*** instead of ***UIColor.dark***

Notes:

Zeplin desktop app can export all colors from styleguide to xcassets catalog.
Steps: *Open app > Select project > Open Styleguide > Export to xcassets and select xcassets catalog for colors*

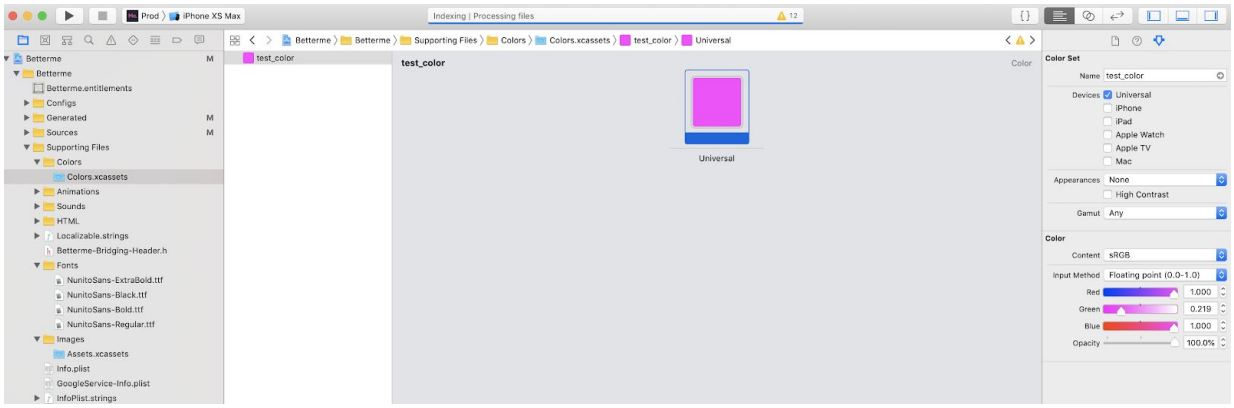

Image 1 - Example of Colors.xsassets

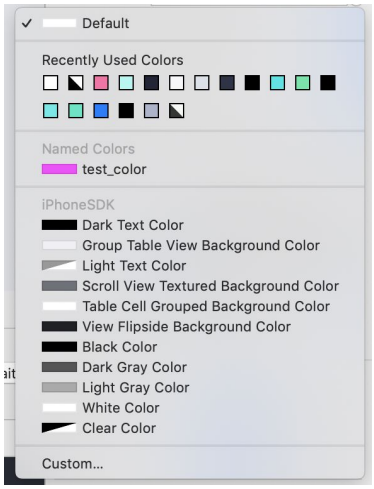

Image 2 - Example of color usage in IB

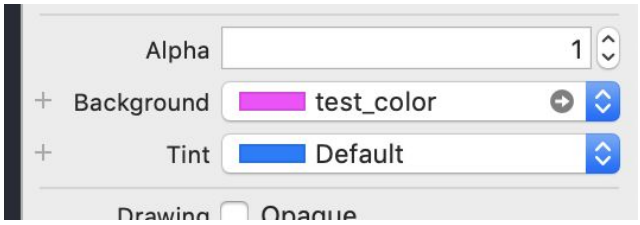

Image 3 - Example of applied color