Flutter 1.17 — no more Flavors, no more iOS Schemas. Command argument that changes everything

Denis Beketsky
May 8 · 6 min read ★

> *Just recently, Flutter 1.17 was released and it brings a lot of cool features. You can read more about it here*

Announcing Flutter 1.17

Includes Metal support for faster iOS performance, new Material components, new Network tracking tooling and more!

medium.com

It comes with lots of performance improvements, new Widgets, and more. But, besides all that, this version of Flutter also got one small, but very useful feature: compile-time variables. Yes, you heard me, starting from 1.17 you can build your Android and iOS application with predefined, compile-time variables that can be used in all layers of your applications — Dart, iOS, Android.

> ***Update 28.05.2020**: Added notes about ***.xcconfig** limitations in iOS paragraph.*

Let's take a look closer

If you are working with Flutter Web, you may know the Flutter Tool argument `--dart-define` . Primarily it was used to enable Skia for the Flutter Web like this `--dart-`

For example, if you run the flutter app with the following arguments

```
flutter run --dart-define=SOME_VAR=SOME_VALUE --dart-define=OTHER_VAR=OTHER_VALUE
```

then you can get these values in your Dart code like this

```dart
class EnvironmentConfig {
  static const SOME_VAR = String.fromEnvironment('SOME_VAR');
  static const OTHER_VAR = String.fromEnvironment('OTHER_VAR');
}
```

compile-time-var-example.dart hosted with ♡ by GitHub view raw

And use this class as your Environment specific config all over the project.

This means that now you can get rid of packages like **environment_config.**

> *If you don't know what this package does, you can read this article*

Configure your Flutter Environment in the Proper Way

Flutter multiple environment configurations, without copy/paste, flavors and exposing your environment credentials...

medium.com

Can I use these variables in native?

Short answer — yes, you can use these variables in native too but… it's a little bit trickier.

If you are curious how Flutter passes **Dart Defines** into the native layers, you can take a look at the **Flutter Tools package.** It treats Dart Defines in a slightly different way for each platform but, I will show you examples for iOS and Android builds.

To do so, I created a Flutter project that defines the Application Name and Application Suffix ID for Android and iOS based on **Dart Defines** variables.

value If this parameter not provider, awesomeApp...

github.com

Let's explore this application

This app works with just two compile-time variables which is `DEFINEEXAMPLE_APP_NAME` and `DEFINEEXAMPLE_APP_SUFFIX`. I would recommend adding a prefix to every compile-time variable just to avoid conflicts with variables that Flutter uses. In this example I'm using `DEFINEEXAMPLE_*`.

Dart code

The application is quite simple. If you open **main.dart** you will see that it defines **EnvironmentConfig class**, and **prints it's value**.

```dart
class EnvironmentConfig {
  static const APP_NAME = String.fromEnvironment(
    'DEFINEEXAMPLE_APP_NAME',
    defaultValue: 'awesomeApp'
  );
  static const APP_SUFFIX = String.fromEnvironment(
      'DEFINEEXAMPLE_APP_SUFFIX'
  );
}
```

main-config.dart hosted with ♡ by GitHub view raw

```dart
children: <Widget>[
  Text(
    'You defined ENV variables like',
    style: Theme.of(context).textTheme.headline5,
  ), // Text
  Text(
    'APP_NAME: ${EnvironmentConfig.APP_NAME}',
    style: Theme.of(context).textTheme.subtitle1,
  ), // Text
  Text(
    'APP_SUFFIX: ${EnvironmentConfig.APP_SUFFIX}',
    style: Theme.of(context).textTheme.subtitle1,
  ), // Text
  Padding(
    padding: const EdgeInsets.symmetric(vertical: 16.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
```

```
        style: Theme.of(context).textTheme.headline5,
      ), // Text
      FutureBuilder<PackageInfo>(
        future: PackageInfo.fromPlatform(),
        builder: (context, value) {
          if (!value.hasData) {
            return Container();
          }

          return Text(
            value.data.packageName,
            style: Theme.of(context).textTheme.headline6,
          ); // Text
        }
      ), // FutureBuilder
    ], // <Widget>[]
```

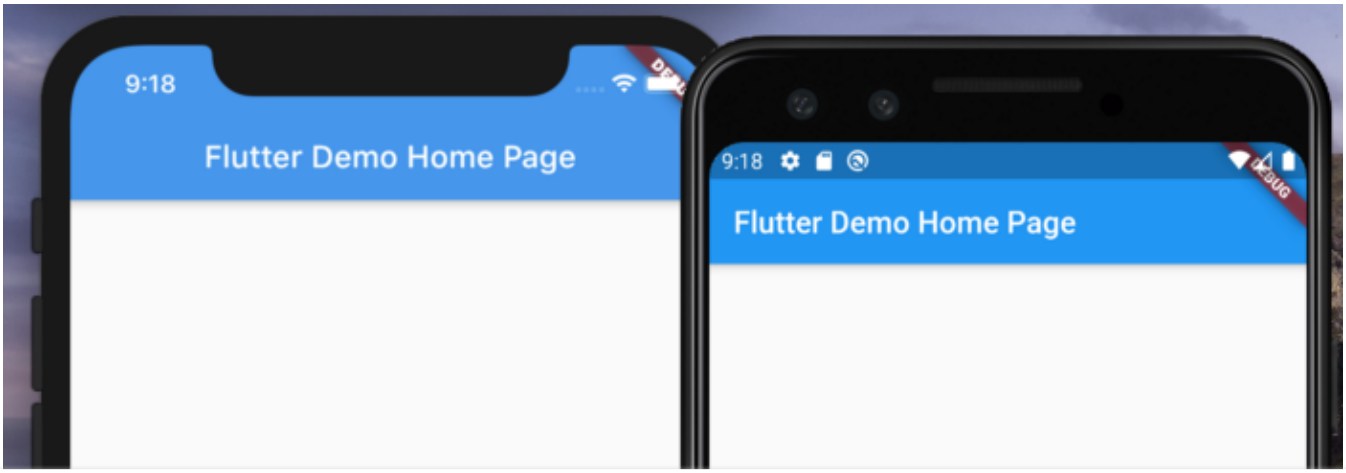
You can notice that I used `awesomeApp` as a default value for `DEFINEEXAMPLE_APP_NAME`. We're going to use this default value in all layers, including native. As for `APP_SUFFIX` — we will get an actual Application ID (`packageName`) in the FutureBuilder Widget from package_info plugin. And if `DEFINEEXAMPLE_APP_SUFFIX` is defined, you will see the Application ID on your screen.

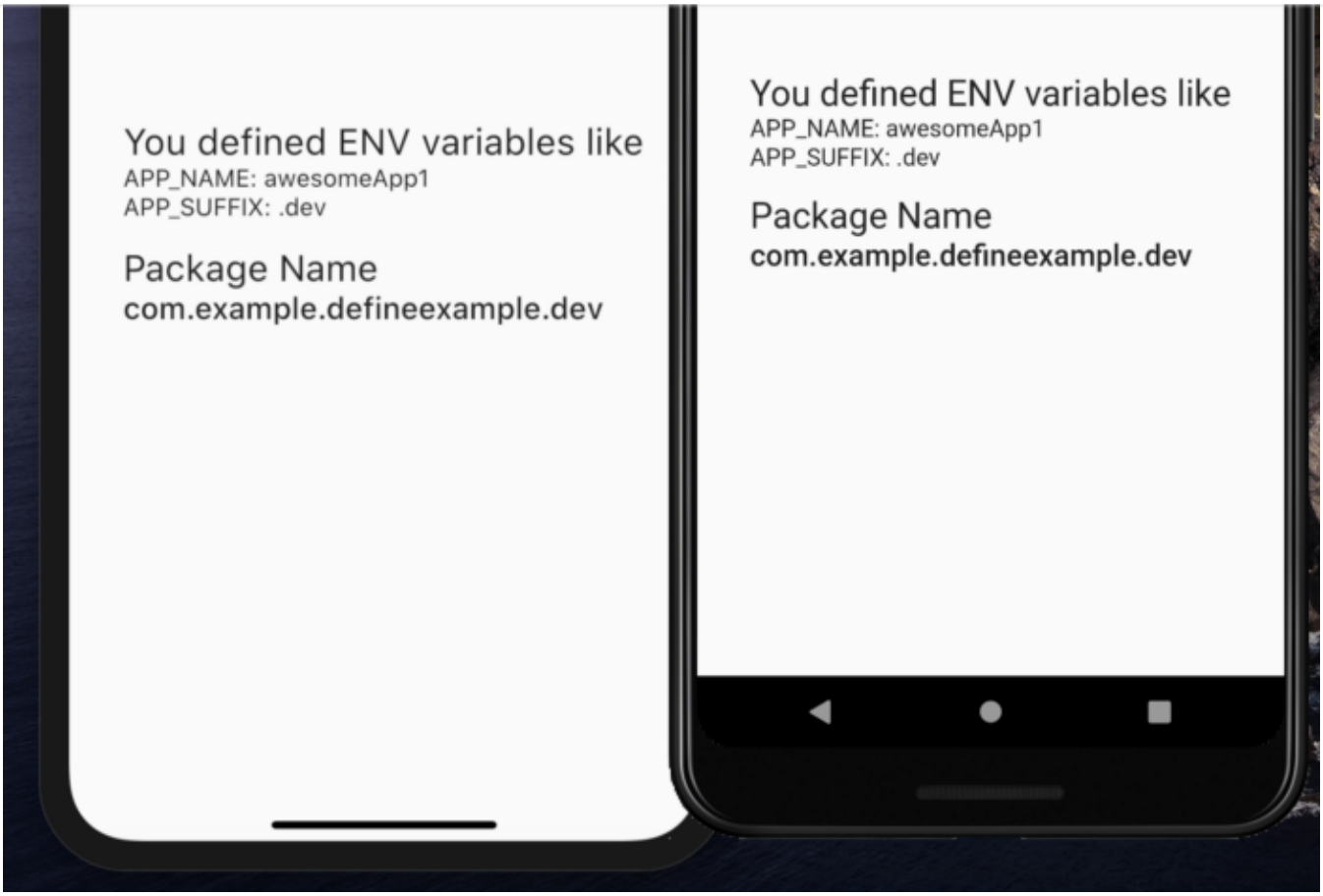
> *One important note: ensure that you are assigning environment variables to the `const` fields. Currently, Flutter has an issue if non-const variable was used.*

When you run this application with a command like

```
flutter run --dart-define=DEFINEEXAMPLE_APP_NAME=awesomeApp1 --dart-define=DEFINEEXAMPLE_APP_SUFFIX=.dev
```

You should see next:

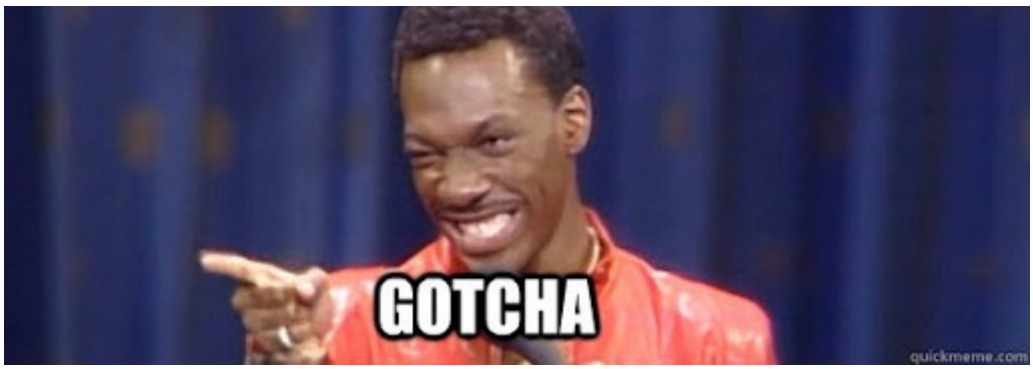




You can notice that it rendered `awesomeApp1`, which was passed from the arguments instead of the default `awesomeApp`. And `com.example.defineexample.dev`. Where `.dev` — defined Suffix.

Android configuration

Ok, so to pass compile-time variables into Android we will need to use Flavors



No worries! No Flavors, no tons of iOS Schemes, no multiple entries, and no copy/paste… well, except default values for different layers. 🤷

```
DEFINEEXAMPLE_APP_NAME=awesomeApp1,DEFINEEXAMPLE_APP_SUFFIX=.dev
```

So we will need to parse it before we can use each key-value pair. Full gradle code can be found here. Flutter passes Dart Defines in the project properties with `dart-defines` key:

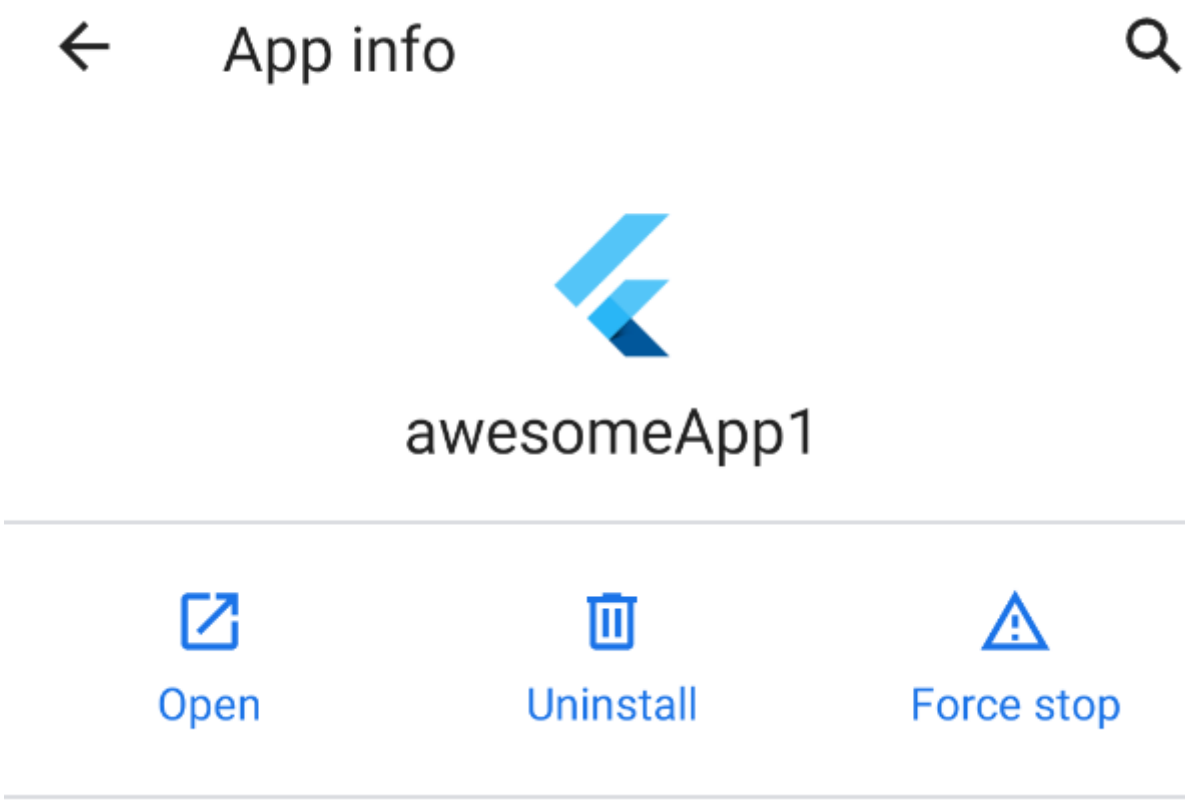
```
1  def dartEnvironmentVariables = [
2      DEFINEEXAMPLE_APP_NAME: 'awesomeApp',
3      DEFINEEXAMPLE_APP_SUFFIX: null
4  ];
5  if (project.hasProperty('dart-defines')) {
6      dartEnvironmentVariables = dartEnvironmentVariables + project.property('dart-defines')
7          .split(',')
8          .collectEntries { entry ->
9              def pair = entry.split('=')
10             [(pair.first()): pair.last()]
11         }
12 }
```

app-default.gradle hosted with ♡ by GitHub view raw

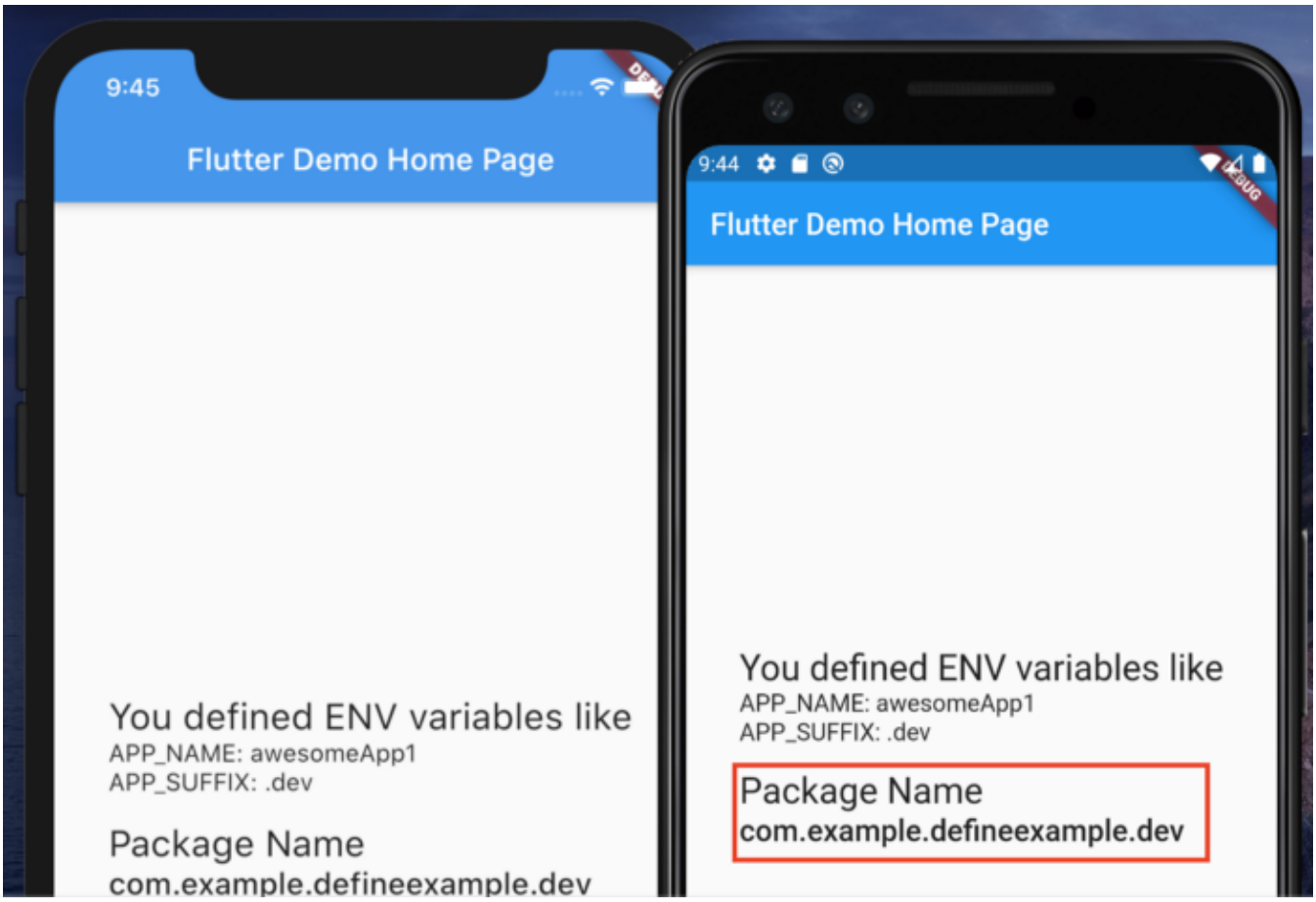
First, it defines defaults (lines 1–4), then, converts a string from `dart-defines` into Map and merges the result with defaults. So now we can define `resValue` and `applicationIdSuffix`

```
defaultConfig {
    // TODO: Specify your own unique Application ID (https://developer.android.com/studio/b
    applicationId "com.example.defineexample"
    applicationIdSuffix dartEnvironmentVariables.DEFINEEXAMPLE_APP_SUFFIX
    minSdkVersion 16
    targetSdkVersion 28
    versionCode flutterVersionCode.toInteger()
    versionName flutterVersionName
    testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    resValue "string", "app_name", dartEnvironmentVariables.DEFINEEXAMPLE_APP_NAME
}
```

After we defined String resource it can be used for example in your android/app/src/main/AndroidManifest.xml to specify Application Name. But basically you can use this set of values anywhere in your project.



Also as Package Name



iOS configuration

During iOS build, Flutter Tool creates a couple of files, including `Generated.xcconfig` and `flutter_export_environment.sh`. They can be found in **ios/Flutter**. Those files are in gitignore by default, so GitHub project won't contain them. But after the build, you will find **Dart Defines** in those files, defined like this:

```
DART_DEFINES=DEFINEEXAMPLE_APP_NAME=awesomeApp1,DEFINEEXAMPLE_APP_SU
FFIX=.dev
```

Since this format can't be used for plist, to define Application Name and Application ID Suffix, we need to define new variables, based on **Dart Defines**. So let's start.

Important Note About *xcconfig

iOS ***.xcconfig** has some limitations when it comes to variable values. You can read about it here or google for a probable solution. But basically, it treats the sequence `//` as a comment delimiter. This means that you won't be able to specify `API_URL` for example like this `https://example.com` since everything after `//` will be ignored. In the same time, if you run your build with the following argument:

```
--dart-define=https://example.com
```

Flutter Tools will create `flutter_export_environment.sh` file with the following row:

Also as `Generated.xcconfig` with the following:

```
...
DART_DEFINES=APP_URL=https://example.com
```

So in your build **Pre-Action** (you will read about it in the next paragraph), you can try to figure out a workaround how to get actual value of `DART_DEFINES` to provide proper parsing of `API_URL` if you want to.

. . .

1. Create Defineexample-defaults.xcconfig file with defaults:

```
DEFINEEXAMPLE_APP_NAME=awesomeApp
DEFINEEXAMPLE_APP_SUFFIX=
```

2. Import it in Debug and Release configurations

And also let's import config file that we will create in just a few moments:

```
#include "Generated.xcconfig"
#include "Defineexample-defaults.xcconfig"
#include "Defineexample.xcconfig"
```

Debug.xcconfig hosted with ♡ by GitHub view raw

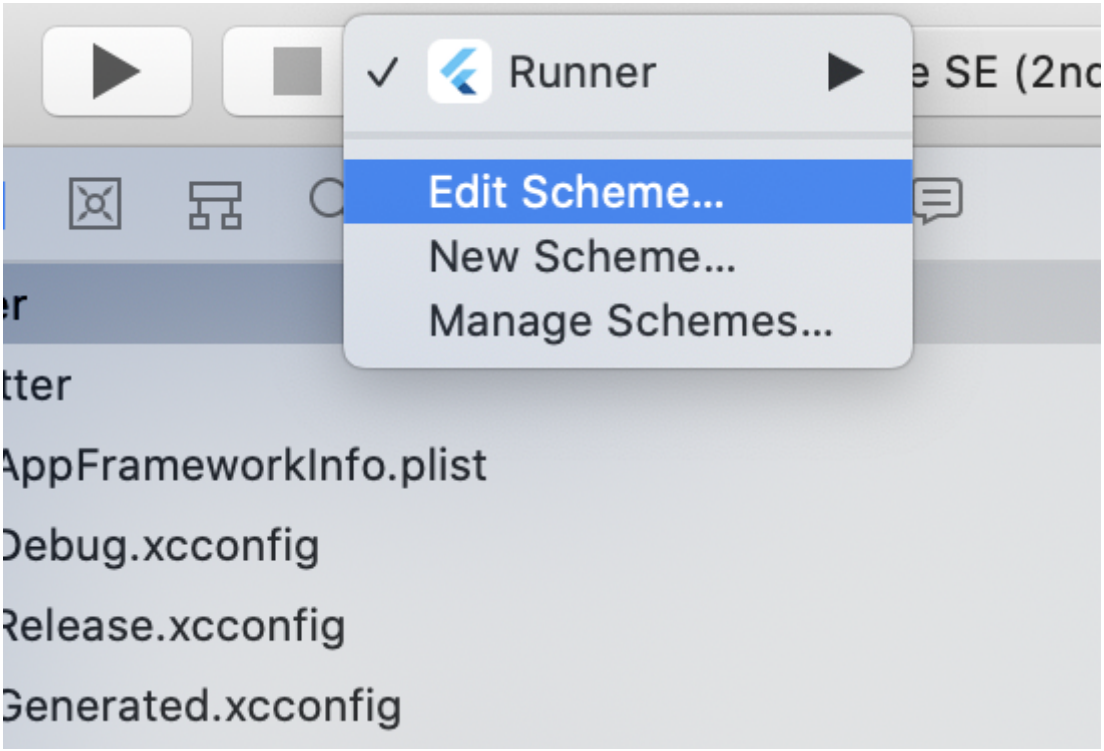
Debug.xcconfig and Release.xcconfig (they're same basically)

Consider putting `Defineexample.xcconfig` after `Defineexample-defaults.xcconfig`, so values from Flutter Run command could override default values.

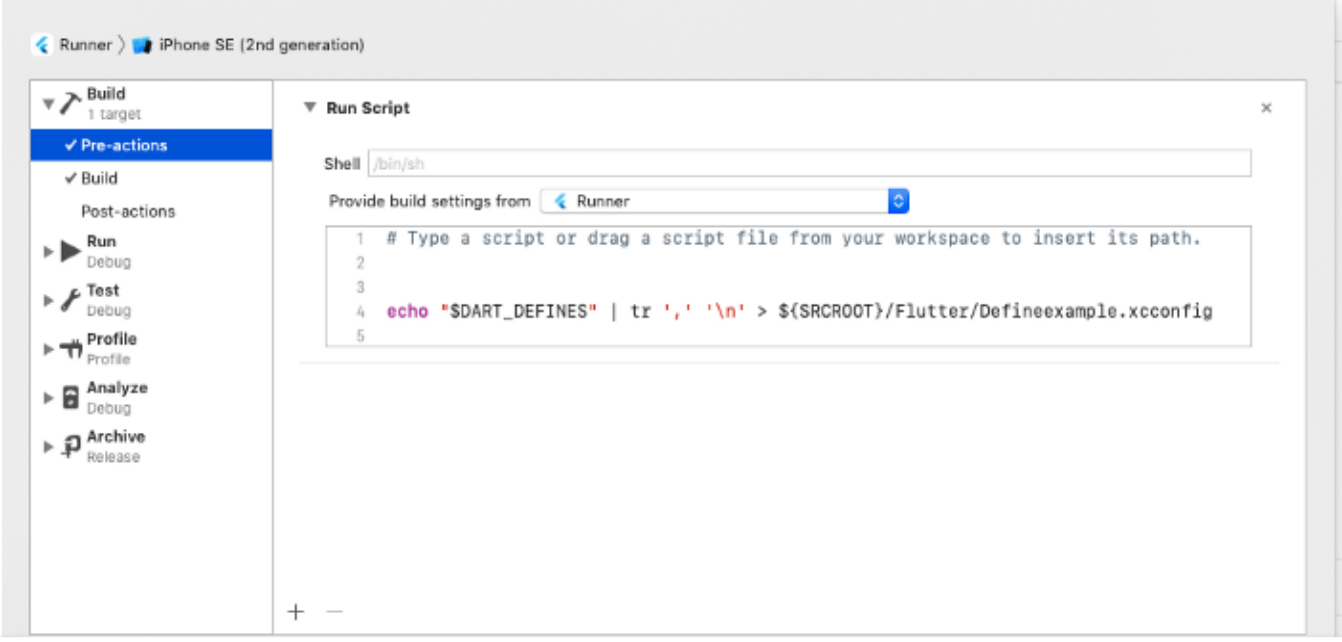
3. Edit your plist file and define Bundle name from `DEFINEEXAMPLE_APP_NAME` and Suffix from `DEFINEEXAMPLE_APP_SUFFIX`

Now it's time to create **Defineexample.xcconfig.** To create it we need to add Pre-Build action to the iOS build.

4. Edit Schema:



5. Add Pre-Action:



with the following command

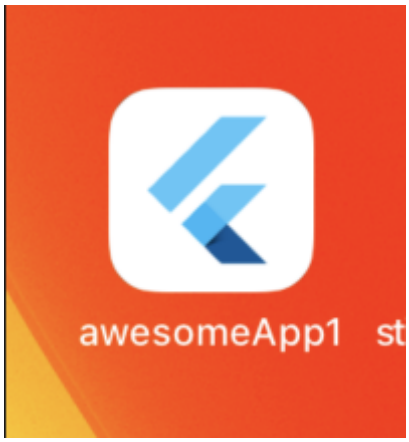
```
echo "$DART_DEFINES" | tr ',' '\n' >
${SRCROOT}/Flutter/Defineexample.xcconfig
```

During the build, this command will create the following **Defineexample.xcconfig** file:

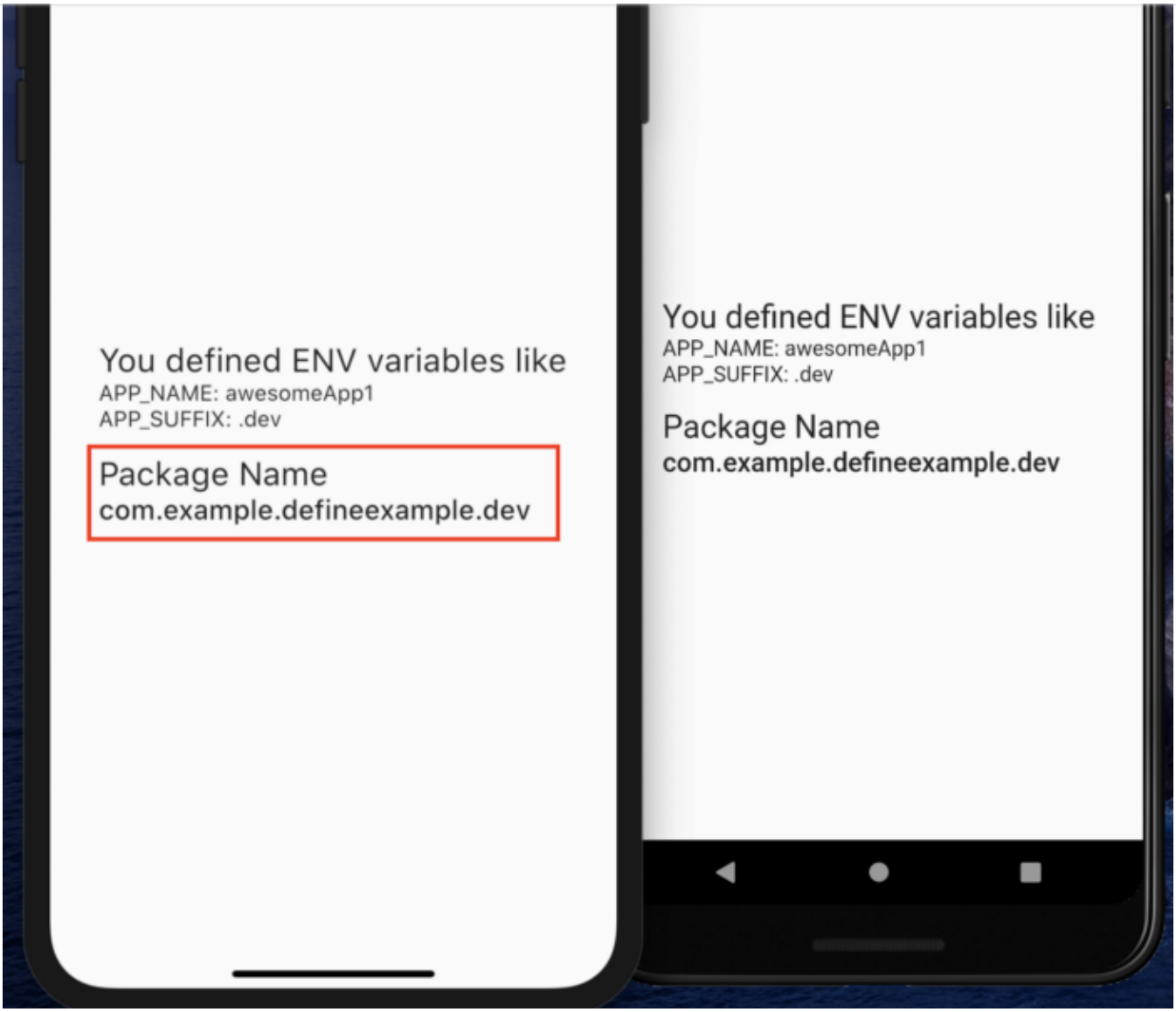
```
DEFINEEXAMPLE_APP_NAME=awesomeApp1
DEFINEEXAMPLE_APP_SUFFIX=.dev
```

> *You can choose different* ***.xcconfig** *file names if you want to. Just ensure that they won't be modified by Flutter Tool scripts.*
>
> *Also, keep in mind limitations of* ***.xcconfig** *if you need to respect* `//` *during the parse.*

When the application will be installed on your device, you can notice that it has a name, that we defined during the build:



Also as Package Name:



Same as with Android, now you can use this set of values anywhere in your project.

That's pretty much it) Example project can be found here:

TatsuUkraine/flutter_define_example

A new Flutter application. Run and define your DEFINEEXAMPLE_APP_NAME value If this parameter not provider, awesomeApp...

github.com

Thank you for your time and happy coding!