Integrate the Connect library into a Flutter app

The Connect SDK is a library that captures visitors' interactions with mobile applications. You can integrate the library into your Flutter app and track the findings in the Acoustic Connect interface.

> Note
>
> New to Connect? Install our preconfigured sample app and explore the implementation faster.

Requirements

- **Acoustic Connect**. To use the Connect SDK, your company must have an active Connect subscription. Proper credentials are required for each app. For instructions, see Generate Connect credentials for integration.
- **Development environment**. To integrate the library into your app, you need a standard Flutter environment. Supported versions: 3.16 and later.
- **Mobile app compatibility**. The Connect library can function on end users' devices running Android 5.0 (API level 21) - Android 14 (API level 34). Supported iOS versions - 13.0 and later.

General guidelines

It is important that you assign **unique IDs** to all UI controls that you want to capture.

Initial setup

1. From the root project directory, run the following commands.

Shell

```
flutter pub add connect_flutter_plugin
flutter pub get
```

2. Open `lib/main.dart` and add an import statement there:

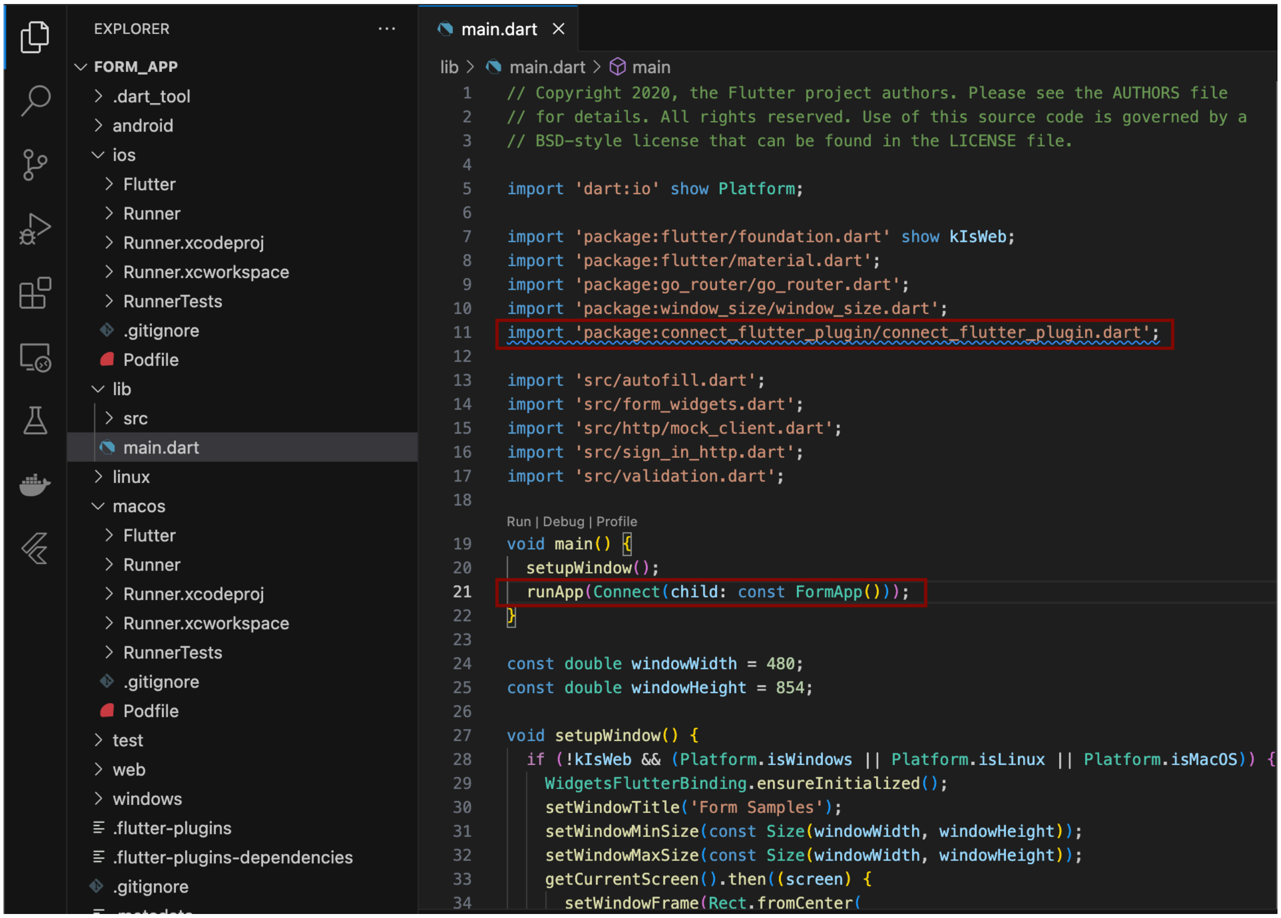
Dart

```
import 'package:connect_flutter_plugin/connect_flutter_plugin.dart';
```

3. To wrap the Connect library over your app, add another statement to `lib/main.dart`. Replace `FormApp` with the name of your app.

Dart

```dart
runApp(Connect(child: const FormApp()));
```

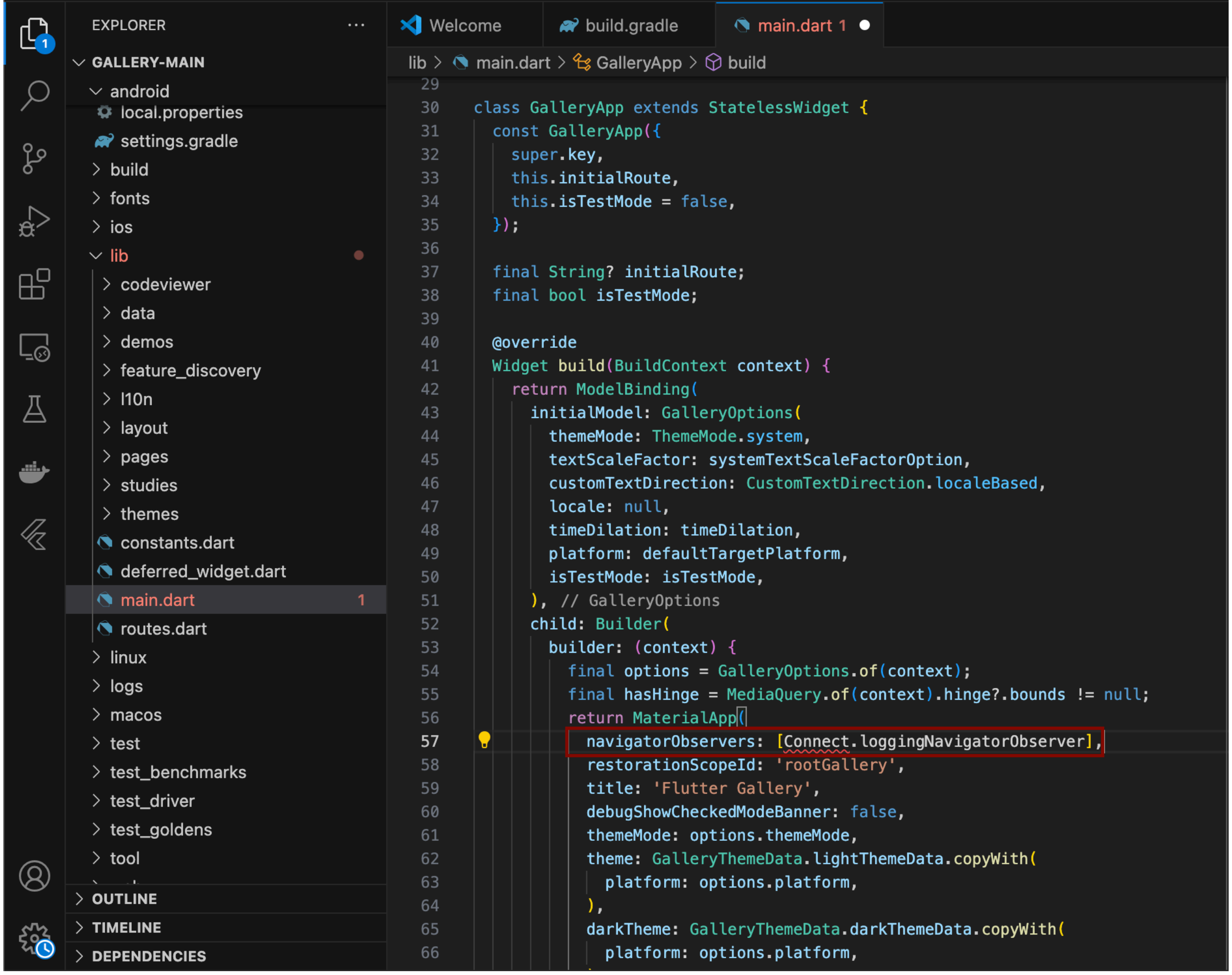


```dart
// Copyright 2020, the Flutter project authors. Please see the AUTHORS file
// for details. All rights reserved. Use of this source code is governed by a
// BSD-style license that can be found in the LICENSE file.

import 'dart:io' show Platform;

import 'package:flutter/foundation.dart' show kIsWeb;
import 'package:flutter/material.dart';
import 'package:go_router/go_router.dart';
import 'package:window_size/window_size.dart';
import 'package:connect_flutter_plugin/connect_flutter_plugin.dart';

import 'src/autofill.dart';
import 'src/form_widgets.dart';
import 'src/http/mock_client.dart';
import 'src/sign_in_http.dart';
import 'src/validation.dart';

void main() {
  setupWindow();
  runApp(Connect(child: const FormApp()));
}

const double windowWidth = 480;
const double windowHeight = 854;

void setupWindow() {
  if (!kIsWeb && (Platform.isWindows || Platform.isLinux || Platform.isMacOS)) {
    WidgetsFlutterBinding.ensureInitialized();
    setWindowTitle('Form Samples');
    setWindowMinSize(const Size(windowWidth, windowHeight));
    setWindowMaxSize(const Size(windowWidth, windowHeight));
    getCurrentScreen().then((screen) {
      setWindowFrame(Rect.fromCenter(
```

4. In the same file, add the Connect Wrapper and Navigation Observer. It handles the app's navigation route changes and event logging.

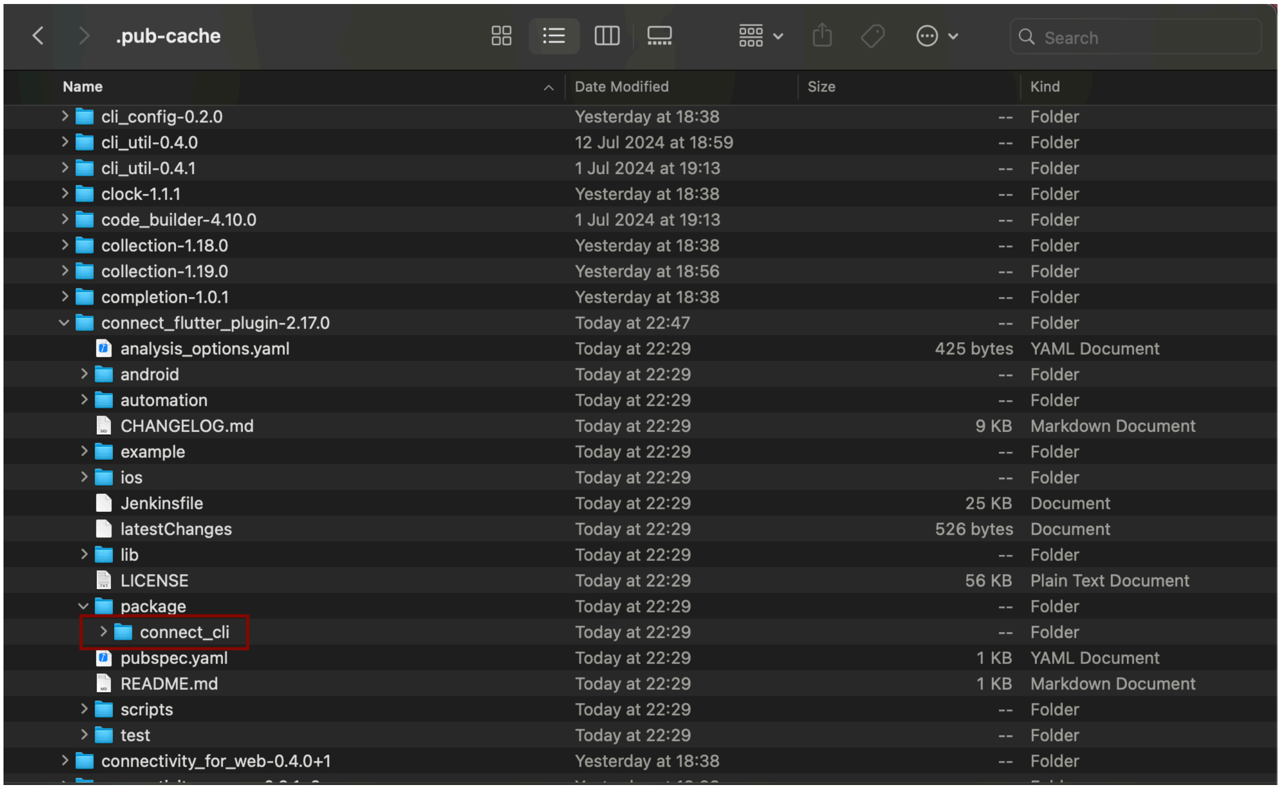
Dart

```dart
navigatorObservers: [Connect.loggingNavigatorObserver],
```



```dart

class GalleryApp extends StatelessWidget {
  const GalleryApp({
    super.key,
    this.initialRoute,
    this.isTestMode = false,
  });

  final String? initialRoute;
  final bool isTestMode;

  @override
  Widget build(BuildContext context) {
    return ModelBinding(
      initialModel: GalleryOptions(
        themeMode: ThemeMode.system,
        textScaleFactor: systemTextScaleFactorOption,
        customTextDirection: CustomTextDirection.localeBased,
        locale: null,
        timeDilation: timeDilation,
        platform: defaultTargetPlatform,
        isTestMode: isTestMode,
      ), // GalleryOptions
      child: Builder(
        builder: (context) {
          final options = GalleryOptions.of(context);
          final hasHinge = MediaQuery.of(context).hinge?.bounds != null;
          return MaterialApp(
            navigatorObservers: [Connect.loggingNavigatorObserver],
            restorationScopeId: 'rootGallery',
            title: 'Flutter Gallery',
            debugShowCheckedModeBanner: false,
            themeMode: options.themeMode,
            theme: GalleryThemeData.lightThemeData.copyWith(
              platform: options.platform,
            ),
            darkTheme: GalleryThemeData.darkThemeData.copyWith(
              platform: options.platform,
```

5. Copy the Connect Automation CLI directory from `$HOME/.pub-cache/hosted/pub.dev/connect_flutter_plugin-<version>/package/connect_cli` to the root of your user directory.



Name	Date Modified	Size	Kind
cli_config-0.2.0	Yesterday at 18:38	--	Folder
cli_util-0.4.0	12 Jul 2024 at 18:59	--	Folder
cli_util-0.4.1	1 Jul 2024 at 19:13	--	Folder
clock-1.1.1	Yesterday at 18:38	--	Folder
code_builder-4.10.0	1 Jul 2024 at 19:13	--	Folder
collection-1.18.0	Yesterday at 18:38	--	Folder
collection-1.19.0	Yesterday at 18:56	--	Folder
completion-1.0.1	Yesterday at 18:38	--	Folder
connect_flutter_plugin-2.17.0	Today at 22:47	--	Folder
analysis_options.yaml	Today at 22:29	425 bytes	YAML Document
android	Today at 22:29	--	Folder
automation	Today at 22:29	--	Folder
CHANGELOG.md	Today at 22:29	9 KB	Markdown Document
example	Today at 22:29	--	Folder
ios	Today at 22:29	--	Folder
Jenkinsfile	Today at 22:29	25 KB	Document
latestChanges	Today at 22:29	526 bytes	Document
lib	Today at 22:29	--	Folder
LICENSE	Today at 22:29	56 KB	Plain Text Document
package	Today at 22:29	--	Folder
connect_cli	Today at 22:29	--	Folder
pubspec.yaml	Today at 22:29	1 KB	YAML Document
README.md	Today at 22:29	1 KB	Markdown Document
scripts	Today at 22:29	--	Folder
test	Today at 22:29	--	Folder
connectivity_for_web-0.4.0+1	Yesterday at 18:38	--	Folder

6. Navigate to the `connect_cli` directory that you have just created and fetch dependencies for your project.

Shell

```
cd $HOME/connect_cli
flutter pub get
```

7. From the root project directory, run a Dart file.

Shell

```
cd <YOUR_PROJECT_PATH>
dart run $HOME/connect_cli/bin/connect_cli.dart
```

8. Install the pods in the iOS version of the library.

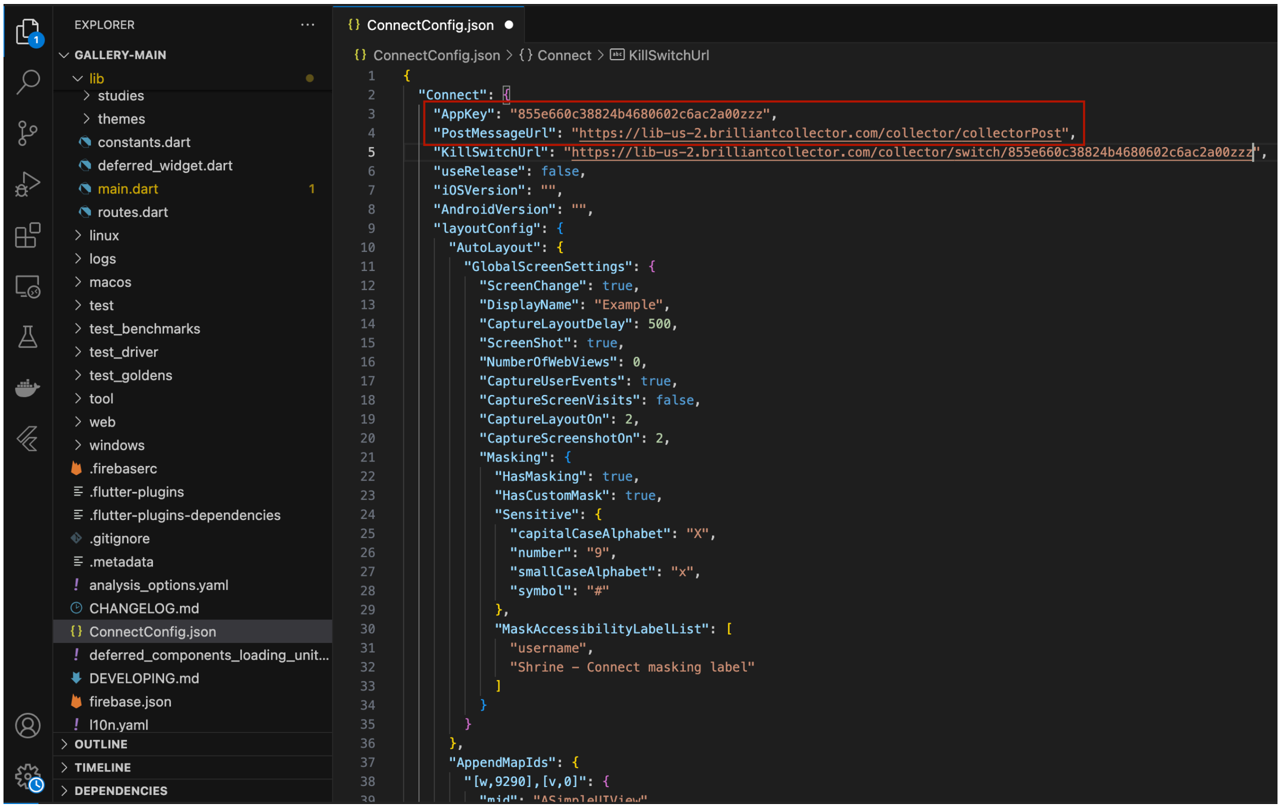
Shell

```
cd ios && pod install
```

Required configuration

Steps

1. In the root project directory, open **ConnectConfig.json** and update the `AppKey` and `PostMessageUrl` parameters in it.
2. If necessary, update other parameters (the list is available below).



```
{
  "Connect": {
    "AppKey": "855e660c38824b4680602c6ac2a00zzz",
    "PostMessageUrl": "https://lib-us-2.brilliantcollector.com/collector/collectorPost",
    "KillSwitchUrl": "https://lib-us-2.brilliantcollector.com/collector/switch/855e660c38824b4680602c6ac2a00zzz",
    "useRelease": false,
    "iOSVersion": "",
    "AndroidVersion": "",
    "layoutConfig": {
      "AutoLayout": {
        "GlobalScreenSettings": {
          "ScreenChange": true,
          "DisplayName": "Example",
          "CaptureLayoutDelay": 500,
          "ScreenShot": true,
          "NumberOfWebViews": 0,
          "CaptureUserEvents": true,
          "CaptureScreenVisits": false,
          "CaptureLayoutOn": 2,
          "CaptureScreenshotOn": 2,
          "Masking": {
            "HasMasking": true,
            "HasCustomMask": true,
            "Sensitive": {
              "capitalCaseAlphabet": "X",
              "number": "9",
              "smallCaseAlphabet": "x",
              "symbol": "#"
            },
            "MaskAccessibilityLabelList": [
              "username",
              "Shrine - Connect masking label"
            ]
          }
        }
      },
      "AppendMapIds": {
        "[w,9290],[v,0]": {
          "mid": "ASimpleUIView"
```

3. Navigate to the root project directory and run the Connect Command Line Tool.

Shell

```
cd <YOUR_PROJECT_PATH>
dart run $HOME/connect_cli/bin/connect_cli.dart
```

> 🚧 Important
>
> Make sure you run this command each time after updating **ConnectConfig.json**.

4. Build and run your apps.

Parameters

Here is a full list of configuration parameters you may need during the initial installation.

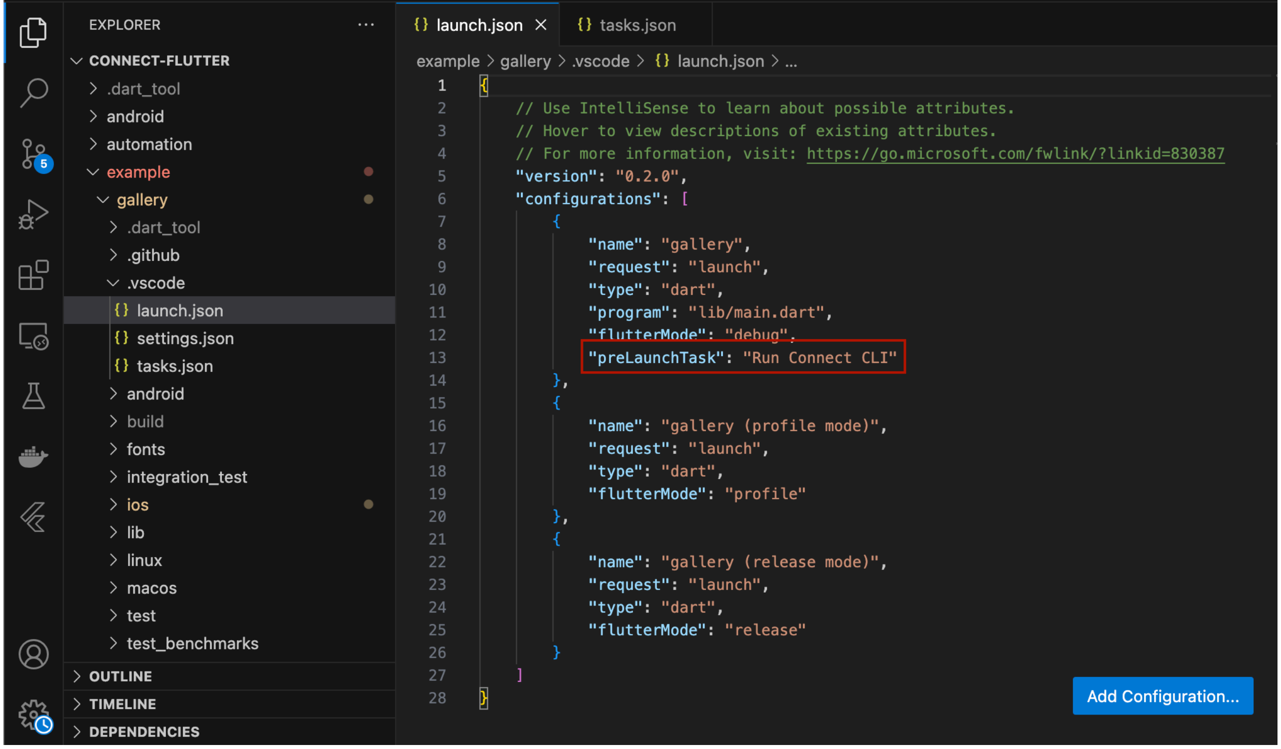
Parameter	Values	Required?	Description
`AppKey`	String	Required	The application key generated for your Flutter app
`PostMessageUrl`	String	Required	The collector URL associated with your Connect subscription
`KillSwitchUrl`	String	Optional	For production, you will need to replace the default application key at the end of the URL with your application key. You can do it now, but this is not required.
`useRelease`	Boolean. The default value is `true` .	Optional	Set the value to `false` to switch to the latest beta version of the library. It contains debug information.
`iOSVersion`	String	Optional	Under the hood, the Connect Flutter SDK is a wrapper for two Connect SDKs: iOS and Android. Using this parameter, you can downgrade the iOS SDK to an older version.
`AndroidVersion`	String	Optional	Using this parameter, you can downgrade the Android SDK to an older version.

Final project preparation

Launch task creation (optional)

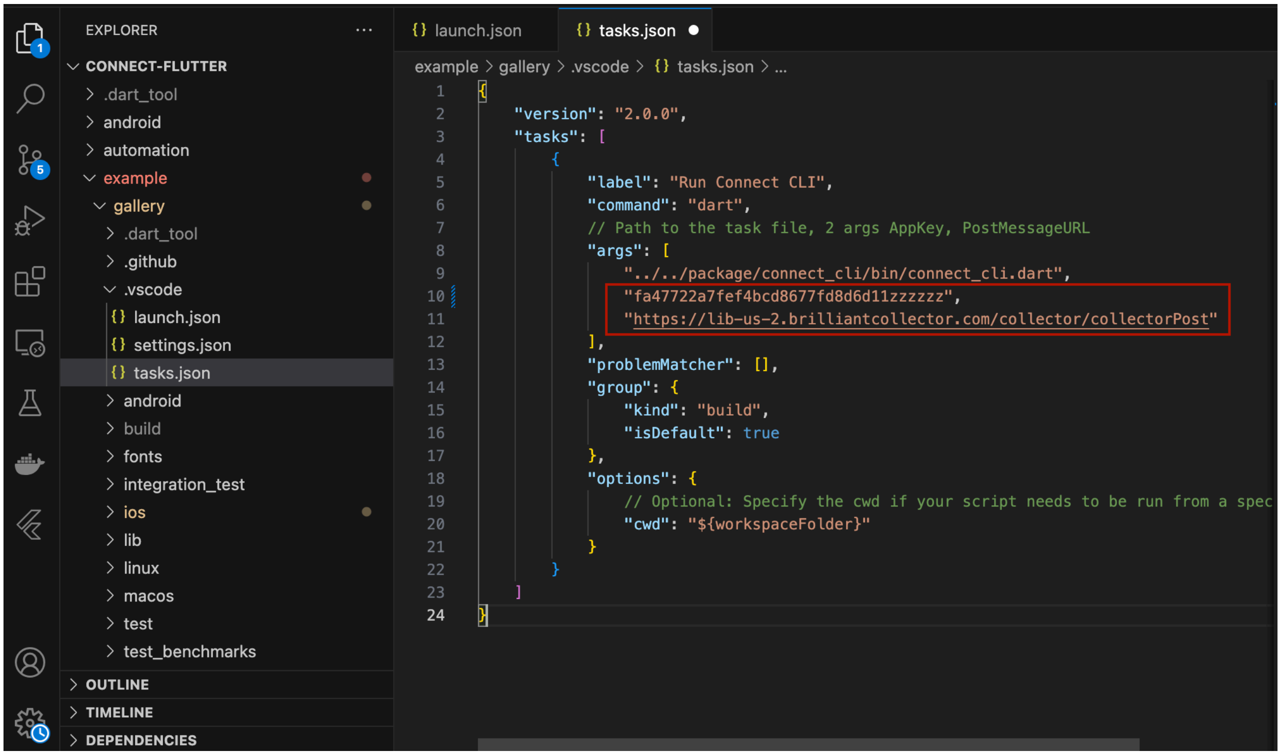
Our sample app contains a hidden directory (**.vscode**). The files in it integrate the above steps into a pre-build flow. You can re-use this directory for your app.

1. Create the same directory in the root of your app.
2. Add the prelaunch task labeled **Run Connect CLI**.



```json
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "gallery",
            "request": "launch",
            "type": "dart",
            "program": "lib/main.dart",
            "flutterMode": "debug",
            "preLaunchTask": "Run Connect CLI"
        },
        {
            "name": "gallery (profile mode)",
            "request": "launch",
            "type": "dart",
            "flutterMode": "profile"
        },
        {
            "name": "gallery (release mode)",
            "request": "launch",
            "type": "dart",
            "flutterMode": "release"
        }
    ]
}
```

3. Copy **tasks.json** from our sample app.
4. In your **tasks.json**, update the `args` section with your application key and endpoint URL.



```json
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "Run Connect CLI",
            "command": "dart",
            // Path to the task file, 2 args AppKey, PostMessageURL
            "args": [
                "../../package/connect_cli/bin/connect_cli.dart",
                "fa47722a7fef4bcd8677fd8d6d11zzzzzz",
                "https://lib-us-2.brilliantcollector.com/collector/collectorPost"
            ],
            "problemMatcher": [],
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "options": {
                // Optional: Specify the cwd if your script needs to be run from a spec
                "cwd": "${workspaceFolder}"
            }
        }
    ]
}
```

Material navigation and navigation widgets

Does your app use Material navigation or any of the following widgets: TabBar, TabBarView or BottomNavigationBar? Logging screen change events for them requires some additional setup.

For example, if a BottomNavigationBar is used without a proper Route-based navigation setup (e.g., by showing and hiding widgets within the same screen), LoggingNavigatorObserver won't be able to log navigation events.

To log navigation events for these types of widgets, add the following code snippets when the user interacts with these widgets (e.g., `onTap` or `onChanged` callback functions).

Dart

```dart
PluginTealeaf.logScreenLayout('logicalPageName');
```

Troubleshooting

If there's a build issue or the Connect Flutter plugin is not using the latest SDK, run the following commands.

Shell

```shell
cd android && ./gradlew build --refresh-dependencies
cd ios && pod update
```