E2Eテストのための

Flutterアプリ実装ガイドライン

2023年7月31日 版

MagicPod

目次

1. はじめに

本ガイドラインは、Flutterを使って作られたiOS及びAndroidアプリをE2Eテスト可能にするためのアプリの実装ガイドラインです。

1-1. 本ガイドラインの適用範囲

本ガイドラインで紹介する方法は、Appium及びAppiumを内部で利用しているE2E自動テストツール全般に対し適用可能です。

1-2. 本ガイドラインの検証対象バージョン

本ガイドラインの内容を検証するにあたっては、以下のバージョンのライブラリ・ツールを利用しました。

- Flutter バージョン3.3.0
- Appium バージョン2.0.0-beta.71

Appiumを内部で利用しているE2E自動テストツールとしては、MagicPod (https://magicpod.com) を使った動作検証を行いました。

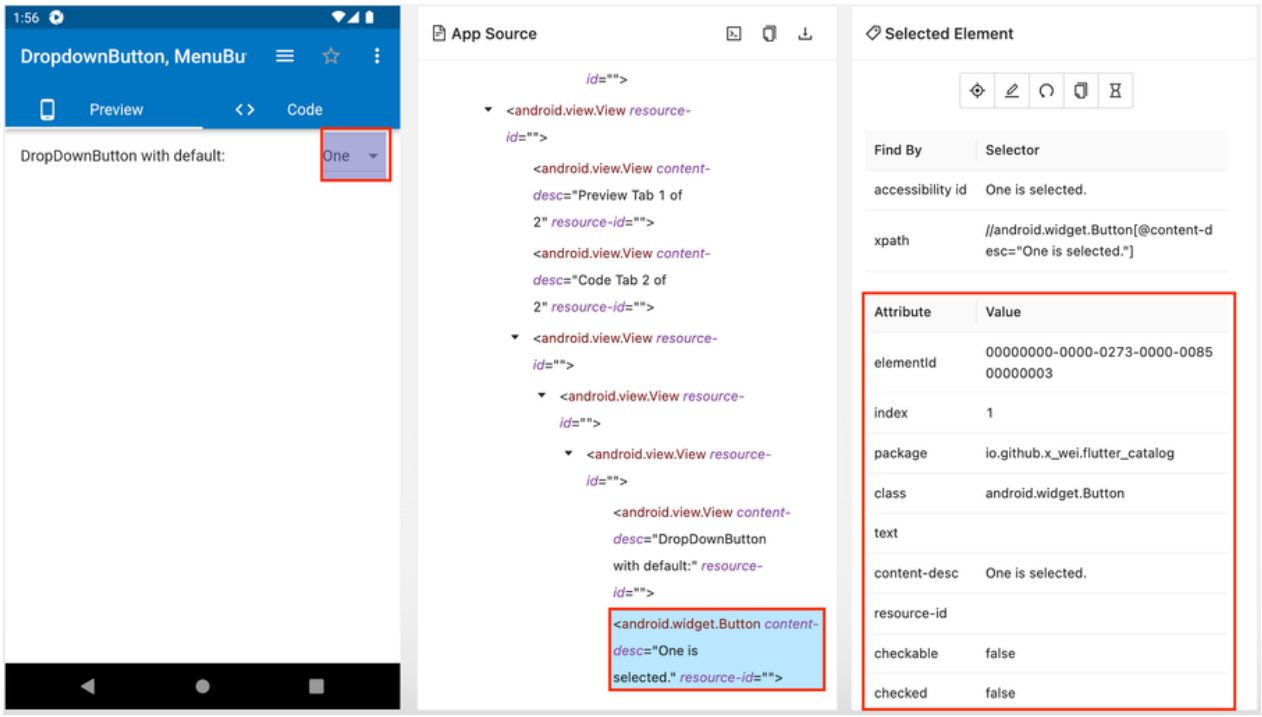
2. FlutterアプリのE2Eテスト自動化における問題

Appiumを使ってFlutterアプリのE2Eテストを自動化する際に頻繁に問題になるのが、Flutterアプリ上のUI要素を正しく認識できないことです。本章では、この問題について具体的に解説します。

最初に、AppiumがどのようにUI要素を認識するのかを説明します。

FlutterアプリにSemanticsTreeというものがあるように、Appiumにもページソースというものがあります。ページソースとは画面上のUI要素を木構造で表現したデータ構造のことです。AppiumでE2Eテストを作成する際、このページソースから目的のUI要素を探し、そのUI要素情報を確認する必要があります。その確認手段の1つにAppium Inspector (https://inspector.appiumpro.com/) というものがあります。このAppium Inspectorを使うとアプリ画面の各UI要素が画面上のどこにあるか、どういった属性値を持つか等がわかります。

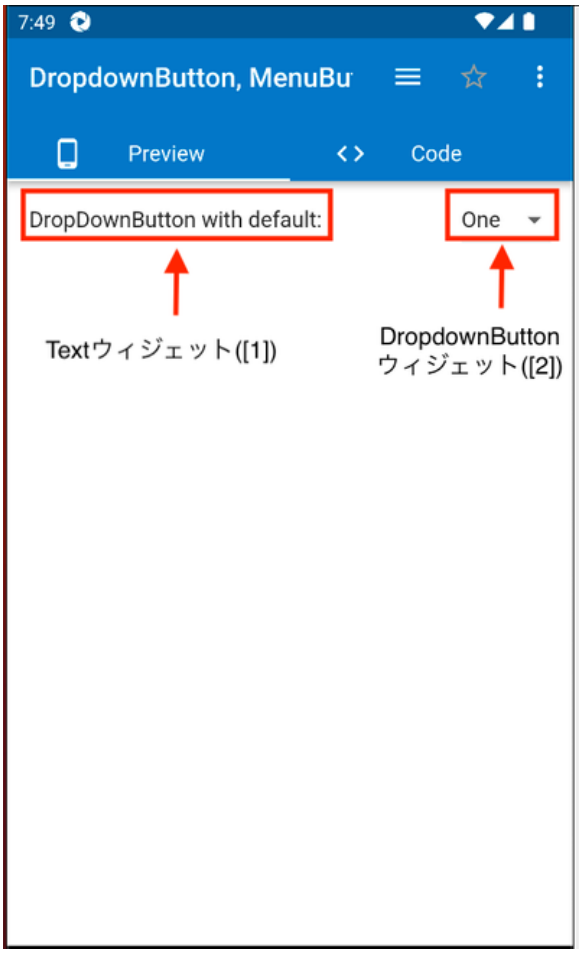
以下のAppium Inspectorの画面の左側では「One」というボタンが、アプリ画面上のどこにあるか、真ん中では「One」というボタンがページソース上のどこにあるか、右側では「One」というボタンがどういった属性値を持つかを示しています。



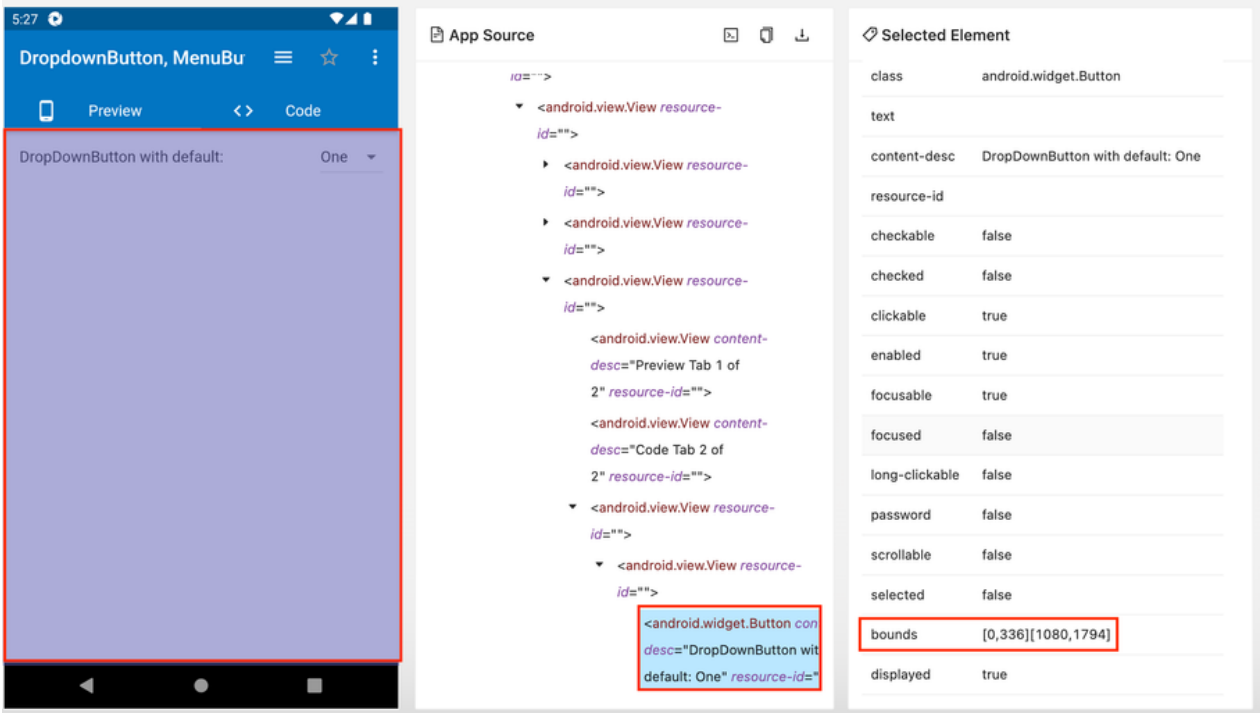
AppiumがFlutterアプリのUI要素をうまく認識できないケースには、大きく分けて次の2つのパターンがあります。

2-1. 複数のウィジェットが1つのUI要素の塊として認識される

最初の問題は、「複数のウィジェットが、別々の要素ではなく塊の要素として認識される」という問題です。以下のFlutterアプリの画面を使って、具体的に説明しましょう。
この画面では、ListTileのtitleプロパティとして「DropDownButton with default:」というテキストを持つTextウィジェット([1])と、ListTileのtrailingプロパティとして「One」「Two」「Three」「Four」という選択肢を持つDropdownButtonウィジェット([2])を使っています。



さっそくAppium Inspectorを使い、それぞれのウィジェットがUI要素としてどのように認識されるかを確認します。結果は下図のとおりです。



結果を見ると、TextウィジェットとDropdownButtonウィジェットの領域が1つの塊になってしまい、さらにそのサイズも実際よりもずっと大きな領域として認識されてしまっています。これは、テスト自動化の際に次のような問題を引き起こします。

1. TextウィジェットとDropdownButtonウィジェットそれぞれの表示テキストの値を個別に取得できない。
2. UI要素の位置が実際のDropdownButtonの位置と大きく異なり、DropdownButtonをクリックコマンドでタップできない。Appiumのクリックコマンドは対象のUI要素の中心位置をクリックするため、UI要素の位置が実態と異なるとタップに失敗してしまう。(bounds属性の値からこのUI要素の左上の座標は(x, y) = (0, 336)、サイズは1080*1458であり、画面サイズ(1080*1794)と比較すると実態と大きく異なることがわかる)

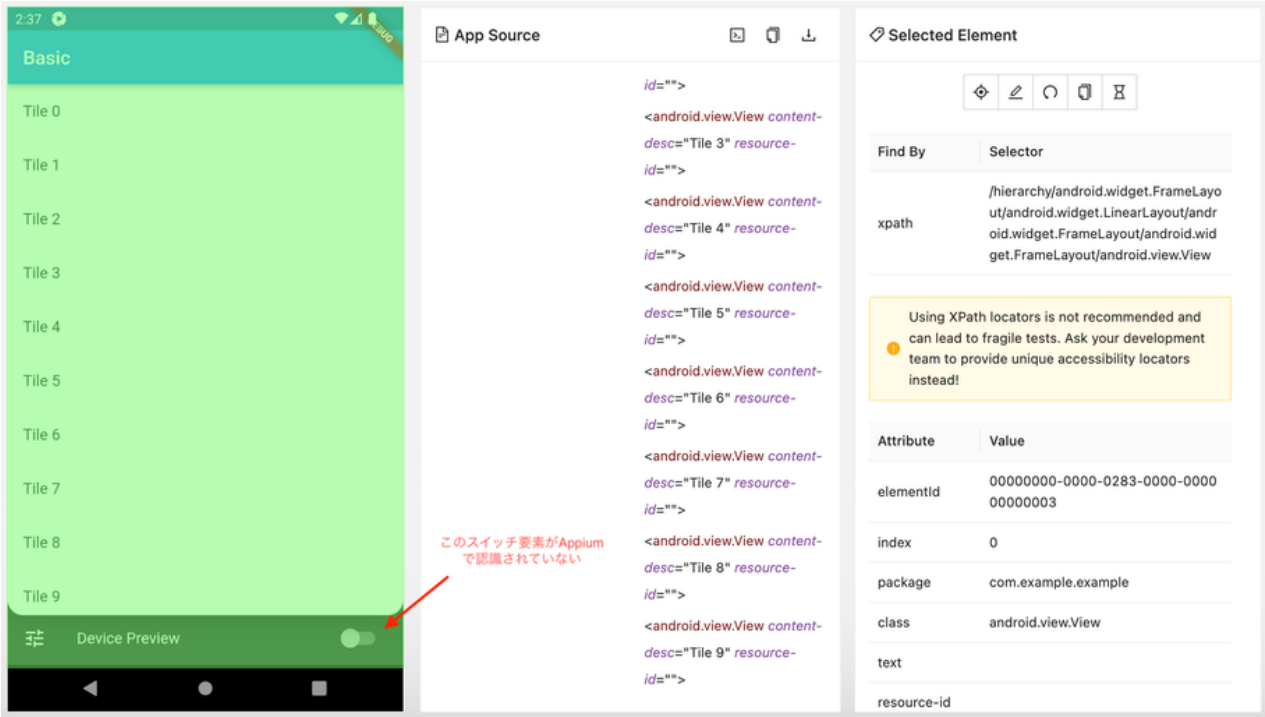
確認した範囲では、少なくとも以下の場合に同様の問題が発生しました。

- StackウィジェットのchildrenプロパティにContainerウィジェットを使う場合
- ColumnウィジェットchildrenプロパティにTextFieldウィジェットやTextウィジェット、HighlightViewウィジェット、Centerウィジェット等を使う場合

2-2. そもそもUI要素が認識されない

2つ目の問題は、「画面項目に対応するUI要素が全く認識されない」という問題です。今度は以下のFlutterアプリの画面を使って具体的に説明しましょう。画面右下のスイッチ([1])要素がどのように認識されるか、Appium Inspectorを使って調べてみます。

認識結果は以下のようになりました。



少しわかりにくいですが、画面右下のスイッチがUI要素として全く認識されていません。こうなると、クリックコマンドで要素をタップすることができなくなります。

3. 解決策

前章では、FlutterアプリのUI要素が正しく認識されない2つのパターンについて解説しました。この章では、これらの問題の解決法を紹介します。

3-1. Flutterを3.3.0以上にバージョンアップする

新しいバージョンのFlutterではUI要素認識に関する問題が大幅に改善されているため、まずはFlutterのバージョンを3.3.0以上にします。
https://github.com/flutter/flutter/issues/18060#issuecomment-1251740879
によると、iOS実機であればバージョン3.0以降、それ以外であればバージョン3.0未満でもこの改善は有効なようですが、本ガイドラインでは、ガイドラインの検証を通じて動作確認済のFlutter 3.3.0以降を推奨します。

3-2. アプリのUI要素の実装方法を見直す

続いて、前章で出てきた2つの問題に関する解決策を具体的に紹介します。解決には、うまく認識されないUI要素の実装方法を見直す必要があります。

3-2-1. ウィジェットをテストしたい単位でSemanticsNodeとして切り出す

まずは2-1.「複数のウィジェットが1つのUI要素の塊として認識される」という問題の解決策として、ウィジェットをテストしたい単位でSemanticsNodeとして切り出すことを説明します。TextウィジェットとDropdownButtonウィジェットの領域が1つのUI要素の塊として認識されてしまった、先ほどの画面を再掲します。

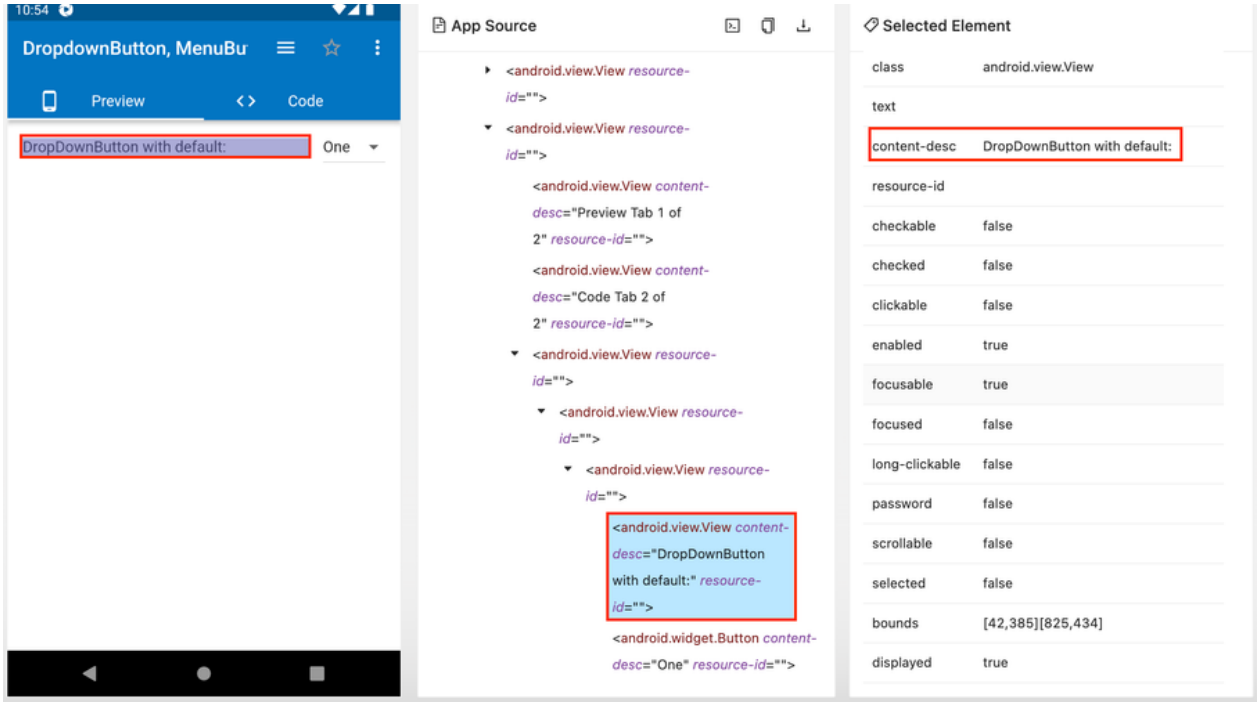
この画面はListTileウィジェットやTextウィジェット、DropdownButtonウィジェットを組み合わせて実装されています。

```dart
@override
Widget build(BuildContext context) {
  return Column(
    children: <Widget>[
      ListTile(
        title: const Text('DropDownButton with default:'),
        trailing: DropdownButton<String>(
          // Must be one of items.value.
          value: _btn1SelectedVal,
          onChanged: (String? newValue) {
            if (newValue != null) {
              setState(() => _btn1SelectedVal = newValue);
            }
          },
          items: this._dropDownMenuItems,
        ), // DropdownButton
      ), // ListTile
    ], // <Widget>[]
  ); // Column
}
```

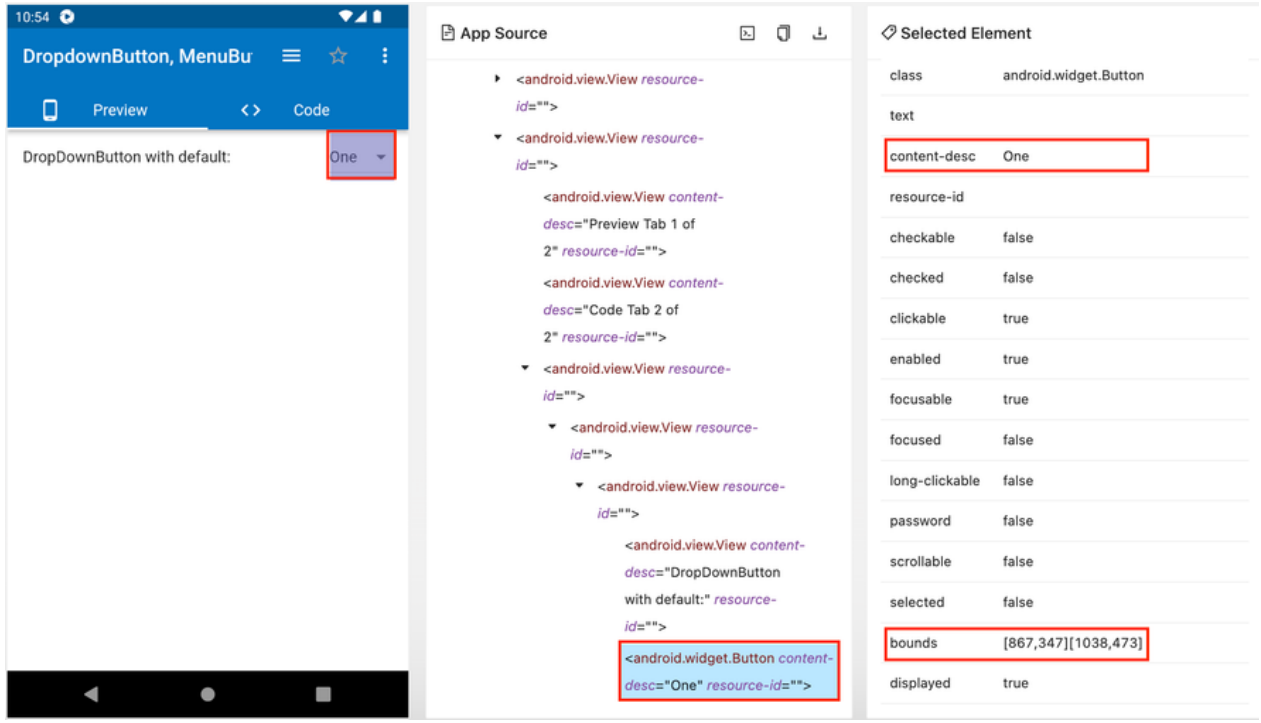
SemanticsNodeとしてTextウィジェットとDropdownButtonウィジェットを切り出します。そのために、containerプロパティをtrueにしたSemanticsウィジェットでそれぞれのウィジェットをラップします。

```dart
@override
Widget build(BuildContext context) {
  return Column(
    children: <Widget>[
      ListTile(
        title: Semantics(
          container: true,
          child: const Text('DropDownButton with default:')
        ), // Semantics
        trailing: Semantics(
          container: true,
          child: DropdownButton<String>(
            // Must be one of items.value.
            value: _btn1SelectedVal,
            onChanged: (String? newValue) {
              if (newValue != null) {
                setState(() => _btn1SelectedVal = newValue);
              }
            },
            items: this._dropDownMenuItems,
          ), // DropdownButton
        ), // Semantics
      ), // ListTile
    ], // <Widget>[]
  ); // Column
}
```

すると、Textウィジェットが独立した１つのUI要素として認識されます。



DropdownButtonウィジェットも同様に独立した１つのUI要素として認識されました。



それぞれのウィジェットが独立したUI要素として認識されたため、それぞれのUI要素の位置も実態に沿ったものになりました。そのため、DropdownButtonウィジェットもAppiumのクリックコマンドで操作できるようになりました。これで最低限のE2Eテスタビリティを確保することができました。また、アクセシビリティの観点からも、この改善によりAndroidの「TalkBack」やiOSの「VoiceOver」においてもより正確なボタンの位置がわかるようになりました。

今回はTextウィジェットとDropdownButtonウィジェットを個別のUI要素として切り出しましたが、ListTileウィジェットを個別のUI要素として切り出しても問題ありません。その場合は、GestureDetector (https://api.flutter.dev/flutter/widgets/GestureDetector-class.html) 等を使い、ListTileウィジェットでタップイベントを検知するように変更する方がよいでしょう。

3-2-2. StackウィジェットのZオーダーを適切に設定する

続いて、2-2.「そもそもUI要素が認識されない」という問題の解決策として、StackウィジェットのZオーダーを適切に設定する方法を説明します。画面右下のスイッチ要素が認識されなかった、先ほどの画面を再掲します。

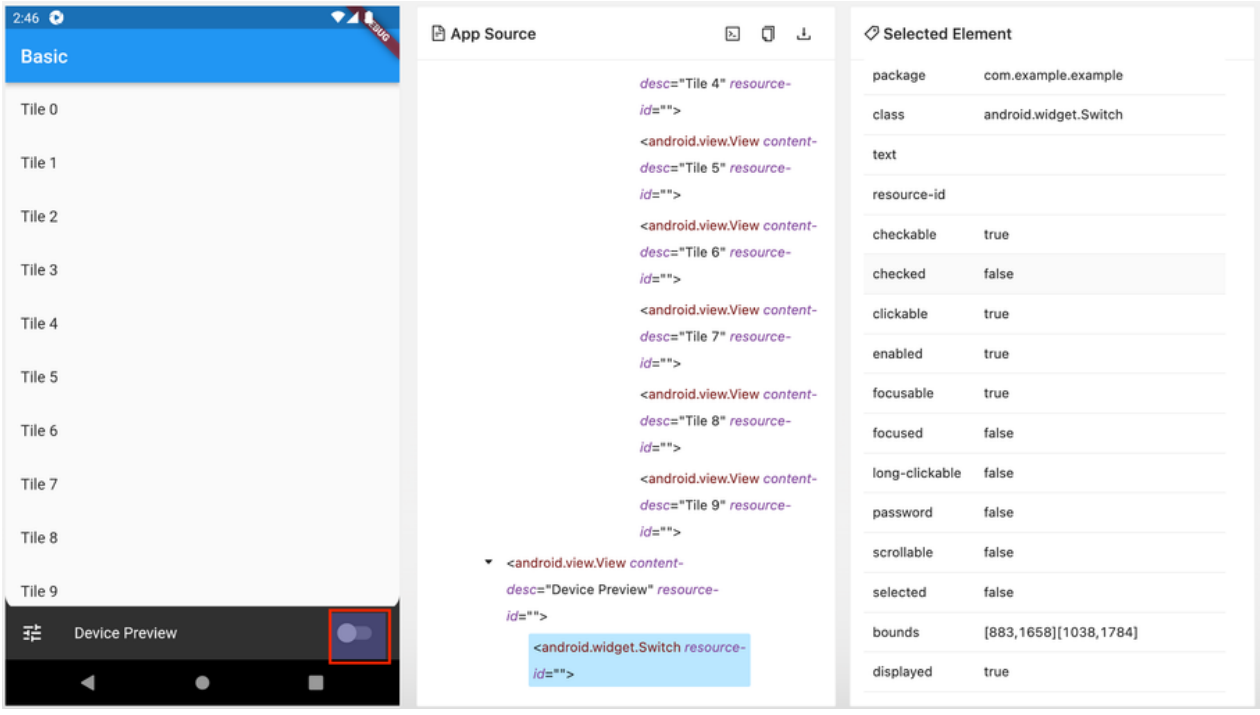
この画面はStackウィジェットを使って実装されています。複雑に見えますが、要はStackウィジェットを使い、Positionedウィジェットと AnimatedPositionedウィジェットを順番に積んでいるだけです。

```
554    return Stack(
555      children: <Widget>[
556        Positioned(
557          key: const Key('Small'),
558          bottom: 0,
559          right: 0,
560          left: 0,
561          child: DevicePreviewSmallLayout(
562            slivers: widget.tools,
563            maxMenuHeight: constraints.maxHeight * 0.5,
564            scaffoldKey: scaffoldKey,
565            onMenuVisibleChanged: (isVisible) => setState(() {
566              _isToolPanelPopOverOpen = isVisible;
567            }),
568          ), // DevicePreviewSmallLayout
569        ), // Positioned
570        AnimatedPositioned(
571          key: const Key('preview'),
572          duration: const Duration(milliseconds: 200),
573          left: 0,
574          right: rightPanelOffset,
575          top: 0,
576          bottom: bottomPanelOffset,
577          child:  ClipRRect(
578            borderRadius: borderRadius,
579            child: Builder(
580              builder: _buildPreview
581            ), // Builder
582          ), // ClipRRect
583        ), // AnimatedPositioned
584      ], // <Widget>[]
585    ); // Stack
```

Stackウィジェットではchildlen配列の後ろにあるウィジェットほど、CSS的には高いZオーダーが与えられます。つまり、より画面前方に表示されるようになります。この順番を正しく並べないと、たとえ画面上は問題なく表示されていたとしても、UI要素としてAppiumで認識できないことがあります。そのため、PositionedウィジェットとAnimatedPositionedウィジェットの位置を入れ替えます。

```
554     return Stack(
555       children: <Widget>[
556         AnimatedPositioned(
557           key: const Key('preview'),
558           duration: const Duration(milliseconds: 200),
559           left: 0,
560           right: rightPanelOffset,
561           top: 0,
562           bottom: bottomPanelOffset,
563           child:  ClipRRect(
564             borderRadius: borderRadius,
565             child: Builder(
566               builder: _buildPreview
567             ), // Builder
568           ), // ClipRRect
569         ), // AnimatedPositioned
570         Positioned(
571           key: const Key('Small'),
572           bottom: 0,
573           right: 0,
574           left: 0,
575           child: DevicePreviewSmallLayout(
576             slivers: widget.tools,
577             maxMenuHeight: constraints.maxHeight * 0.5,
578             scaffoldKey: scaffoldKey,
579             onMenuVisibleChanged: (isVisible) => setState(() {
580               _isToolPanelPopOverOpen = isVisible;
581             }),
582           ), // DevicePreviewSmallLayout
583         ), // Positioned
584       ], // <Widget>[]
585     ); // Stack
```

こうすることで「Device Preview」というラベル横のスイッチを認識することができました。



このように、ウィジェット間のZオーダーを適切に設定し直すことで、認識できなかった要素が認識可能になります。

4. 実装チュートリアル

FlutterアプリをE2Eテスト可能にするための具体的なステップは次の通りです。

4-1. Flutterのバージョンアップ

最初に、利用しているFlutterのバージョンを3.3.0以上にします。

4-2. うまく認識されないUI要素のリストアップ

次に、Appium Inspector、uiautomatorviewer、MagicPodのいずれかを使ってテスト対象のFlutterアプリを立ち上げ、E2Eテストで使用する画面を1つずつ開いて、テストで使うUI要素が正しく認識されているか一つずつ確認し、認識がうまくいかない要素をリストアップします。おすすめはAppium Inspectorです。iOSでもAndroidでも要素の認識のされ方は基本的に同じなので、リストアップ作業はどちらか一方でだけ行えば大丈夫です。

具体的な確認手順は 2章「FlutterアプリのE2Eテスト自動化における問題」を参考にしてください。

- Appium Inspectorを使う場合、セットアップ手順と利用方法は https://support.magic-pod.com/hc/ja/articles/4408926683033#sec3_2 を参考にしてください。
- uiautomatorviewerを使う場合、利用方法は https://support.magic-pod.com/hc/ja/articles/4408926683033#sec3_1 を参考にしてください。この場合Androidアプリを使う必要があります。
- MagicPodを使う場合、MagicPod側の問題で、UI要素が別の他の大きなUI要素によって隠されてうまく認識できない場合があります。https://support.magic-pod.com/hc/ja/articles/4409255879961 の手順に従って上側にある大きな要素を隠していくと要素が見つかる場合は、Flutterアプリに問題はありません。

この作業は、テスト自動化経験があれば非エンジニアでも実施できます。

4-3. アプリのコードの修正

リストアップ作業が終わったら、問題がある各要素に対するアプリのコードを読んで、3-2.「アプリのUI要素の実装方法を見直す」を参考に、要素がうまく認識できるよう、各要素のアプリのコードを修正してください。

5. アクセシビリティ情報への影響

「ウィジェットをテストしたい単位でSemanticsNodeとして切り出す」においてSemanticsNodeとして切り出した要素は、スクリーンリーダーの読み上げ対象となります。ほとんどの場合E2E自動テストで操作・取得したいUI要素はスクリーンリーダーで読み上げて欲しい要素と同じであり、通常は問題は起きないと考えられますが、UI要素の実装を見直す際はスクリーンリーダー読み上げなどのアクセシビリティへの影響にも注意してください。

E2Eテストのための
Flutterアプリ実装ガイドライン
2023年7月31日 版

株式会社MagicPod
〒103-0015
東京都中央区日本橋箱崎町1-2
The Shore 日本橋茅場町4 階

問い合わせ先
https://magicpod.com/contact/