Extracted from:

Programming Flutter

Native, Cross-Platform Apps the Easy Way

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Programming Flutter

Native, Cross-Platform Apps the Easy Way

Carmine Zaccagnino
edited by Michael Swaine

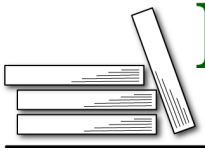
Programming Flutter

Native, Cross-Platform Apps the Easy Way

Carmine Zaccagnino

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Michael Swaine
Copy Editor: Jasmine Kwityn
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

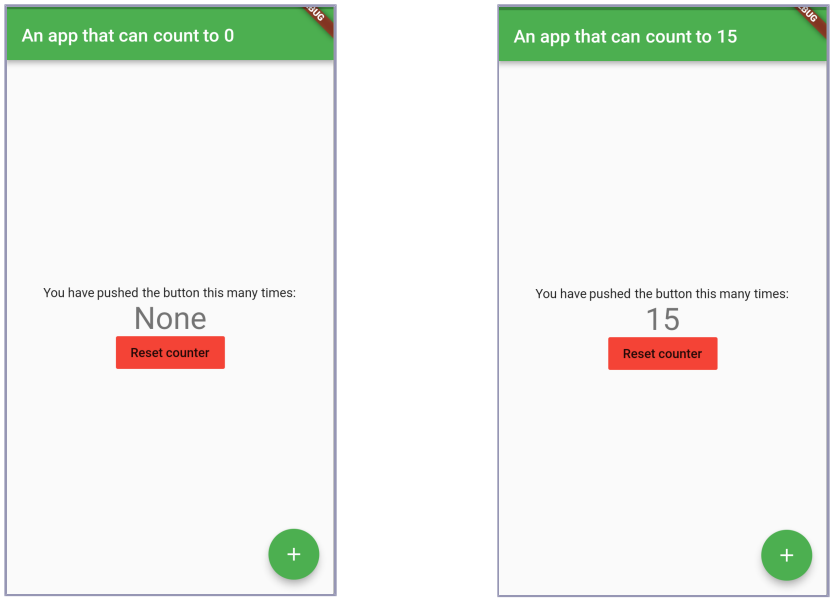
For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-695-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—February 2020

Give the App Your Personal Touch

Now let's modify what we've seen. We'll change the app color to be green, add a red reset button to reset the counter to 0, make the app display the string "None" instead of the counter when the counter is at 0 and change the appBar's title text. In doing this, we'll get a feel for how to make simple changes, and also an introduction to more complex Flutter syntax.

The result will look like this:



Change the App's Primary Color

We'll start editing our app with the simplest change: changing the color to green. This is as simple as changing the following few lines in MyApp's build method:

firstapp_starting/lib/main.dart

```
theme: ThemeData(
  primarySwatch: Colors.blue,
),
```

Go ahead and change Colors.blue to Colors.green so that those lines instead read as follows:

firstapp/lib/main.dart

```
theme: ThemeData(
  primarySwatch: Colors.green,
),
```

Make the Counter Display "None" Instead of 0

Now, let's change the counter text in the middle so that it displays "None" instead of 0.

To do this, we'll add a _displayedString variable to _MyHomePageState, below the counter's declaration.

Locate the following line:

```
int _counter = 0;
```

and, below it, add:

```
String _displayedString;
```

After we've done that, we need to set this variable to "None" if the counter is 0 and to a string representation of the counter if that isn't the case. We're going to need a conditional.

Use Conditionals in Flutter

To do this every time the counter is increased (which we do using a call to setState that triggers a re-render that calls the build method), this needs to be part of _MyHomePageState's build method. So, just above:

```
return new Scaffold(
```

add:

firstapp/lib/main.dart

```
if(_counter == 0) {
  _displayedString = "None";
} else {
  _displayedString = _counter.toString();
}
```

This is a simple if-else construct and, like in many other languages, it could also be expressed in a braceless way as:

```
if(_counter == 0) _displayedString = "None";
else              _displayedString = _counter.toString();
```

or, with a conditional expression, as:

```
_displayedString = _counter == 0 ? "None" : '$_counter';
```

You can find more information about conditional constructs and expressions in Conditional Constructs and Expressions, on page ?.

Set the Counter Text

Now we need to replace the counter with the string we've just generated.

Use String Literals in Dart

At the moment, the code we're using to generate the Text widget for the counter looks like this:

firstapp_starting/lib/main.dart

```
Text(
  '$_counter',
  style: Theme.of(context).textTheme.display1,
),
```

You might have noticed the syntax currently used to display the counter to the user:

```
'$_counter'
```

The single quotes (unlike in C, C++, and Java, single and double quotes are the same in Dart) enclose a string literal, and inside this string literal is a variable name (_counter) preceded by a dollar sign.

Use String Interpolation

In this case, this is an alternative to the syntax we used earlier to convert an integer to a string (_counter.toString()).

This is how string literals work in Dart: they are enclosed in single or double quotes. And we can include variables in string literals by preceding the variable name with a dollar sign.

If we need to use string interpolation with an expression or dot notation (for example, to use member variables of objects or display results of expressions), we need to enclose the variable name in braces. We'll need to do this shortly for the app title.

You can find more information about string literals and string interpolation in Characters and Strings, on page ?.

Back to our example. Since we now want to display a string, we can replace:

```
'$_counter'
```

to be just:

```
displayedString
```

since we don't need to use string literals.

Change the App's Title

To change the appBar's title to "An app that can count to [0, 1, 2, 3...]" we need to use string interpolation.

We will use a fixed string defined in the app's constructor and a variable part, which will be created inside the build method.

The fixed part is the title we give to MyHomePage's constructor (and which then becomes widget.title), which will be "An app that can count to":

firstapp_starting/lib/main.dart

```
  home: MyHomePage(title: 'Flutter Demo Home Page'),
);
```

will have to become:

firstapp/lib/main.dart

```
  home: MyHomePage(title: 'An app that can count to'),
);
```

Now we need to edit _MyHomePageState's build method again, since that's what gets called every time the app's state changes, and that's where the variable part of the string will have to be generated.

To do what I just described, locate the following lines:

firstapp_starting/lib/main.dart

```
appBar: AppBar(
  title: Text(widget.title),
),
```

and change:

```
widget.title
```

to:

```
'${widget.title} $_counter'
```

so that those lines now instead read as follows:

firstapp/lib/main.dart

```
appBar: AppBar(
  title: Text('${widget.title} $_counter'),
),
```

As you can see, we've done the opposite of what we did to the counter, and we had to use braces for widget.title because dot notation was needed to access a member variable.

Add a Reset Button

The last thing we need to do to get to the app in the screenshots shown earlier is to add the reset button.

Before we add the button itself, let's define a simple method that changes the state and resets the counter to 0.

Use setState() to Reset the Counter

Inside _MyHomePageState's definition, below _incrementCounter()'s definition, add the following code:

firstapp/lib/main.dart

```
void _resetCounter() {
  setState(() {
    _counter = 0;
  });
}
```

This is very similar to _incrementCounter() but, instead of incrementing the counter, we set it to 0.

Make the Reset Button

Now that we've defined a method that resets the counter and triggers a re-render, we need to create a button to call that function. So before we do that, we need to know how we do that with Flutter.

Buttons in Flutter: The FlatButton and the IconButton

If we want to quickly create a button in Flutter we can choose between a FlatButton or an IconButton.

The FlatButton is ideal when you want a button that displays text: it has a child attribute that is usually a Text widget, but you have the option to set it to anything.

The reason why the FlatButton is used mostly for text buttons is that using an IconButton exists specifically for the creation of icon-based buttons, by allowing the developer to specify an Icon object as its icon attribute, with the button only consisting of the icon and a small amount of padding.

One of the attributes that show the difference in focus is that the FlatButton's color attribute controls the color of the button itself (the background to its content), whereas the IconButton's color attribute control's the icon's color.

A RaisedButton also exists, but it is just a slightly different looking FlatButton.

Implement the Reset Button

For this example we'll make a FlatButton.

To add the button, locate the following lines in _MyHomePageState's build method:

firstapp_starting/lib/main.dart

```
children: <Widget>[
  Text(
    'You have pushed the button this many times:',
  ),
  Text(
    '$_counter',
    style: Theme.of(context).textTheme.display1,
  ),
],
```

and add, inside the square brackets, the following to the list of widgets:

firstapp/lib/main.dart

```
FlatButton(
  onPressed: _resetCounter,
  color: Colors.red,
  child: Text(
    "Reset counter",
    style: Theme.of(context).textTheme.button,
  ),
),
```

First of all, the action we want to perform is to fire the _resetCounter function, so we set that as the onPressed attribute.

The FlatButton's color attribute should not be confused with the CSS attribute that goes by the same name: as I described earlier, it changes the button's background color, not the text color, which we'll leave black.

The child attribute is just a text string with the built-in button text theme.

The final code is:

firstapp_oldfab/lib/main.dart

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Flutter Demo',
     theme: ThemeData(
       primarySwatch: Colors.green,
     ),
```

```
      home: MyHomePage(title: 'An app that can count to'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;
  String _displayedString;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }
  void _resetCounter() {
    setState(() {
      _counter = 0;
    });
  }
  @override
  Widget build(BuildContext context) {
    if(_counter == 0) {
      _displayedString = "None";
    } else {
      _displayedString = _counter.toString();
    }
    return Scaffold(
      appBar: AppBar(
        title: Text('${widget.title} $_counter'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              _displayedString,
              style: Theme.of(context).textTheme.display1,
            ),
            FlatButton(
              onPressed: _resetCounter,
              color: Colors.red,
```

```
            child: Text(
              "Reset counter",
              style: Theme.of(context).textTheme.button,
            ),
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: Icon(Icons.add),
    ),
  );
 }
}
```

Now, build and run or reload the app and the result will be the same as the screenshots we saw earlier.