Different flavours of Bloc

Enhancing different details

Enhancing different solutions

Content

- Several examples of Bloc
  - Basic Use examples
  - DIY
  - Some examples that can helpfull
    - Blueprint for project
    - Understand the concepts
    - To focus on what is common

Disclaimer: some examples are old

- The overall rationale is still valid
- Some do not use the new event handler…

```dart
/// The events which `CounterBloc` will react to.
abstract class CounterEvent {}

/// Notifies bloc to increment state.
class CounterIncrementPressed extends CounterEvent {}

/// A `CounterBloc` which handles converting `CounterEvent`s into `int`s.
class CounterBloc extends Bloc<CounterEvent, int> {
  /// The initial state of the `CounterBloc` is 0.
  CounterBloc() : super(0) {
    /// When a `CounterIncrementPressed` event is added,
    /// the current `state` of the bloc is accessed via the `state` property
    /// and a new state is emitted via `emit`.
    on<CounterIncrementPressed>((event, emit) => emit(state + 1));
  }
}
```

Disclaimer: some examples are old

- The overall rationale is still valid
- Some do not use the new event handler…
- In old versions must implement two methods
  - **initialState** (to inialize first state before any transaction).
  - **mapEventToState** (to map event in to correspond state).

```dart
enum CounterEvent { increment, decrement }

class CounterBloc extends Bloc<CounterEvent, int> {
  @override
  int get initialState => 0;

  @override
  Stream<int> mapEventToState(CounterEvent event) async* {
    switch (event) {
      case CounterEvent.decrement:
        yield state - 1;
        break;
      case CounterEvent.increment:
        yield state + 1;
        break;
    }
  }
}
```

Flutter BLoC and Provider: A Shopping Cart Example

Junji Zhi Follow

Jul 16, 2019 · 4 min read ★

After watching the Google I/O video again, my understanding of BLoC is that it's **a disciplined way to access and update data among widgets in the tree, like a global variable.**

Obviously, there is more than one way to implement BLoC. After thinking a bit more, I found that we didn't have to use the ReactiveX/rxDart or Streams. They come with their own jargons and building blocks by themselves, which leads to a certain learning curve.

In this article, I present one way to implement BLoC using the Provider package. You would find that ChangeNotifier + provider are enough to implement BLoC.

Example 1

https://medium.com/flutter-community/flutter-bloc-and-provider-a-shopping-cart-example-af75004e1666

Flutter BLoC and Provider: A Shopping Cart Example

Junji Zhi Follow
Jul 16, 2019 · 4 min read ★

After watching the Google I/O video again, my understanding of BLoC is that it's **a disciplined way to access and update data among widgets in the tree, like a global variable.**

Obviously, there is more than one way to implement BLoC. After thinking a bit more, I found that we didn't have to use the ReactiveX/rxDart or Streams. They come with their own jargons and building blocks by themselves, which leads to a certain learning curve.

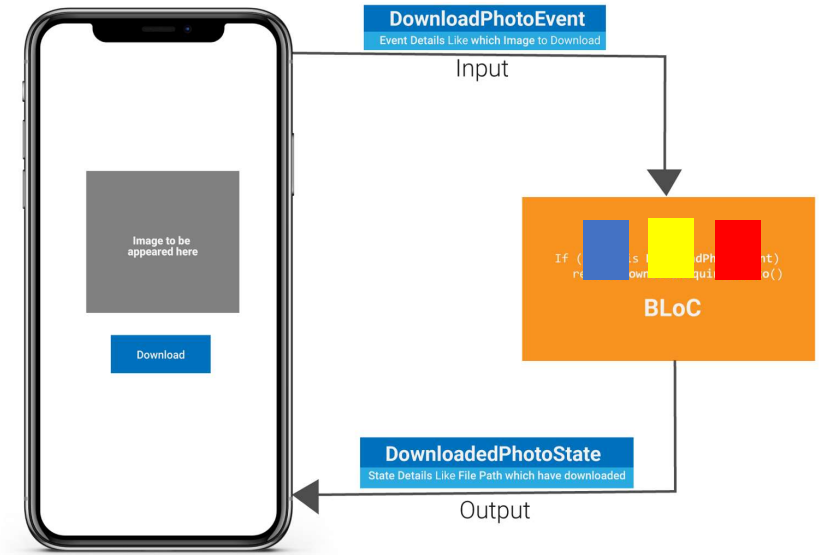
In this article, I present one way to implement BLoC using the Provider package. You would find that ChangeNotifier + provider are enough to implement BLoC.

https://medium.com/flutter-community/flutter-bloc-and-provider-a-shopping-cart-example-af75004e1666

The bloc: store the state

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

class CartBloc with ChangeNotifier {
  Map<int, int> _cart = {};

  Map<int, int> get cart => _cart;

  void addToCart(index) {
    if (_cart.containsKey(index)) {
      _cart[index] += 1;
    } else {
      _cart[index] = 1;
    }
    notifyListeners();
  }

  void clear(index) {
    if (_cart.containsKey(index)) {
      _cart.remove(index);
      notifyListeners();
    }
  }
}
```

The state

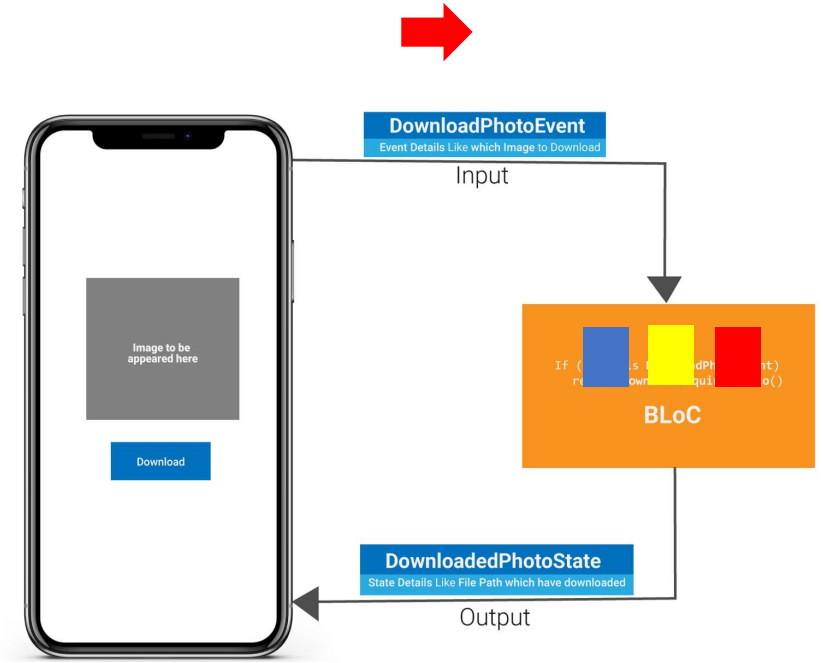


The bloc: handling events

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

class CartBloc with ChangeNotifier {
  Map<int, int> _cart = {};

  Map<int, int> get cart => _cart;

  void addToCart(index) {
    if (_cart.containsKey(index)) {
      _cart[index] += 1;
    } else {
      _cart[index] = 1;
    }
    notifyListeners();
  }

  void clear(index) {
    if (_cart.containsKey(index)) {
      _cart.remove(index);
      notifyListeners();
    }
  }
}
```

The events

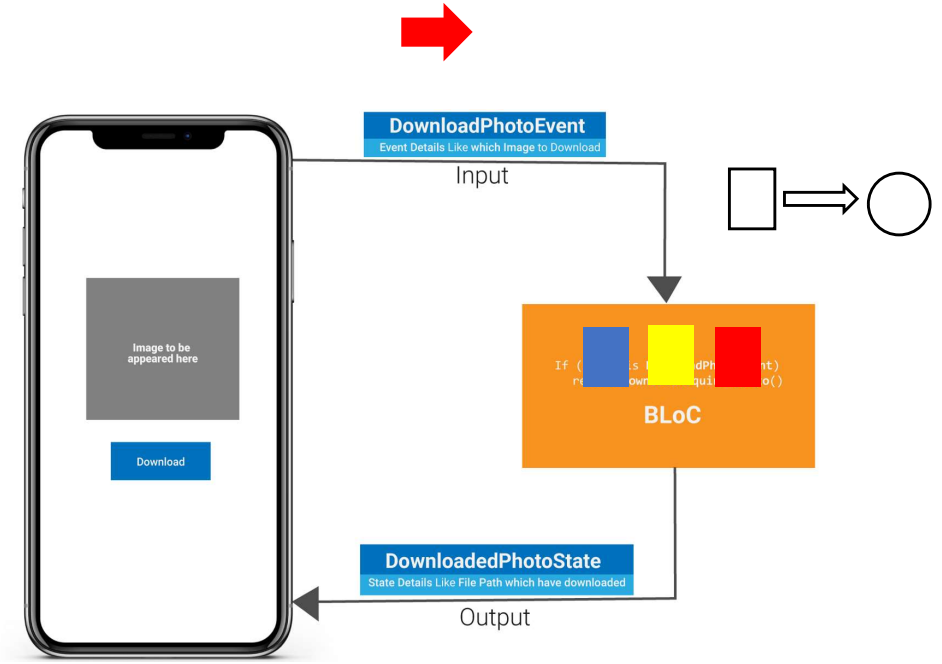
The events



The bloc: changing state

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

class CartBloc with ChangeNotifier {
  Map<int, int> _cart = {};

  Map<int, int> get cart => _cart;

  void addToCart(index) {
    if (_cart.containsKey(index)) {
      _cart[index] += 1;
    } else {
      _cart[index] = 1;
    }
    notifyListeners();
  }

  void clear(index) {
    if (_cart.containsKey(index)) {
      _cart.remove(index);
      notifyListeners();
    }
  }
}
```

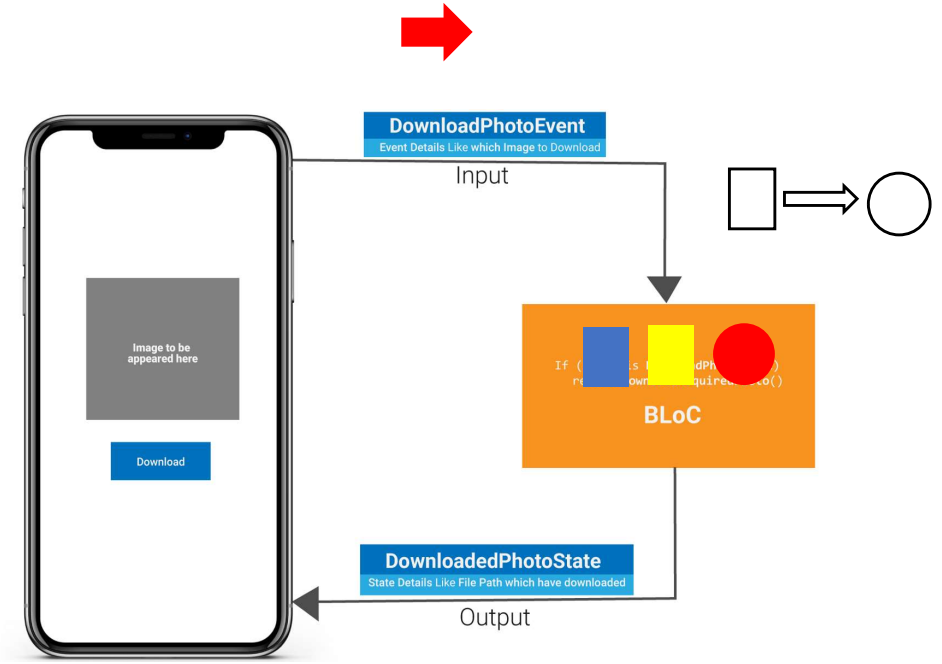
Changing state



The bloc: changing state

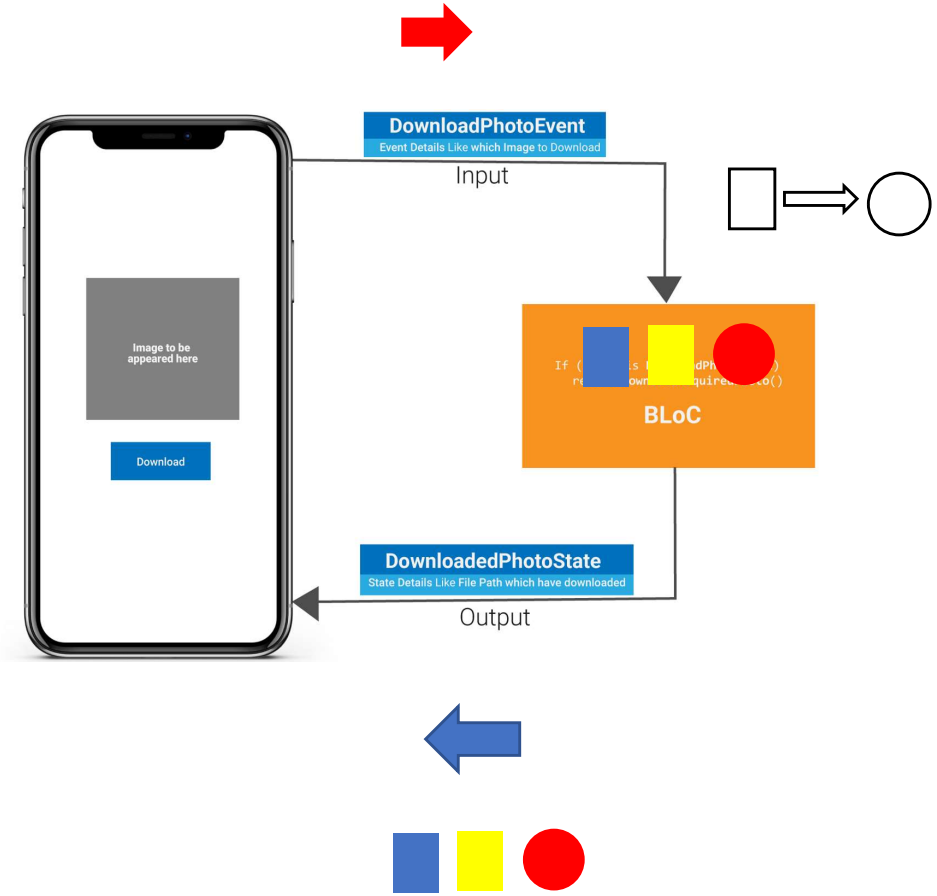
```dart
import 'package:flutter/material.dart';

class CartBloc with ChangeNotifier {
  Map<int, int> _cart = {};

  Map<int, int> get cart => _cart;

  void addToCart(index) {
    if (_cart.containsKey(index)) {
      _cart[index] += 1;
    } else {
      _cart[index] = 1;
    }
    notifyListeners();
  }

  void clear(index) {
    if (_cart.containsKey(index)) {
      _cart.remove(index);
      notifyListeners();
    }
  }
}
```

Changing state



The bloc: notify interested

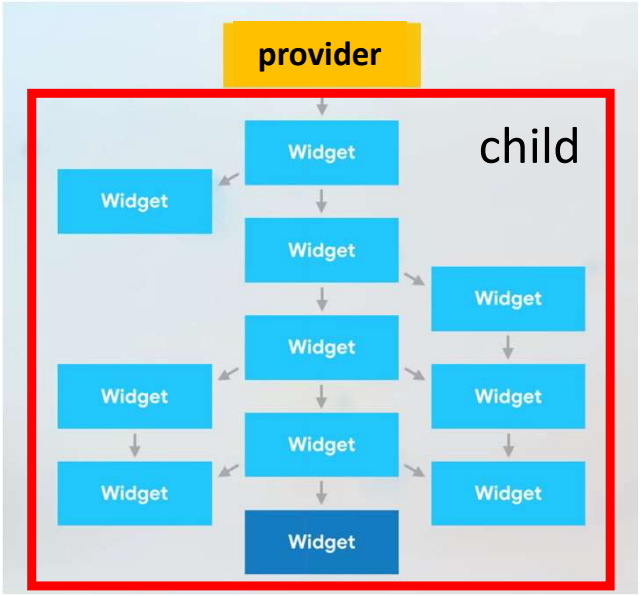
```dart
import 'package:flutter/material.dart';

class CartBloc with ChangeNotifier {
  Map<int, int> _cart = {};

  Map<int, int> get cart => _cart;

  void addToCart(index) {
    if (_cart.containsKey(index)) {
      _cart[index] += 1;
    } else {
      _cart[index] = 1;
    }
    notifyListeners();
  }

  void clear(index) {
    if (_cart.containsKey(index)) {
      _cart.remove(index);
      notifyListeners();
    }
  }
}
```



- Notify the interested – the registered listeners

The bloc: available in widget tree

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider<CartBloc>(
        builder: (context) => CartBloc(),
        child: MaterialApp(
          title: 'Flutter Shopping Cart Demo',
          theme: ThemeData(
            primarySwatch: Colors.blue,
          ),
          home: MyHomePage(title: 'Gift Shop'),
        ));
  }
}
```



the cart: listener

```
Widget build(BuildContext context) {
  var bloc = Provider.of<CartBloc>(context);
  int totalCount = 0;
  if (bloc.cart.length > 0) {
    totalCount = bloc.cart.values.reduce((a, b) => a + b);
  }
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
      actions: <Widget>[
        Container(
          child: GestureDetector(
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => CartPage(),
                ),
              );
            },
            child: _buildCartIcon(totalCount)  // icon with text overlay
          )
        ),
      )
    ],
```

Bloc can be seen as a provider

```
    body: GridView.count(
      crossAxisCount: 2,
      children: List.generate(6, (index) {
        return GestureDetector(
          onTap: () {
            bloc.addToCart(index);
          },
          child: Container(
            decoration: BoxDecoration(
              image: DecorationImage(
                image: AssetImage("assets/${index + 1}.jpg"),
                fit: BoxFit.fitWidth,
              ),
              borderRadius: BorderRadius.circular(12),
            ),
          ));
      }),
    ),
  );
}
```

the cart: using bloc state

```
Widget build(BuildContext context) {
  var bloc = Provider.of<CartBloc>(context);
  int totalCount = 0;
    if (bloc.cart.length > 0) {
      totalCount = bloc.cart.values.reduce((a, b) => a + b);
  }
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
      actions: <Widget>[
          Container(
              child: GestureDetector(
                onTap: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => CartPage(),
                    ),
                  );
                },
                child: _buildCartIcon(totalCount)  // icon with text overlay
              )
            ),
      )
    ],
```

bloc preserves common data
i.e. the state

```
    body: GridView.count(
      crossAxisCount: 2,
      children: List.generate(6, (index) {
        return GestureDetector(
            onTap: () {
              bloc.addToCart(index);
            },
            child: Container(
              decoration: BoxDecoration(
                image: DecorationImage(
                  image: AssetImage("assets/${index + 1}.jpg"),
                  fit: BoxFit.fitWidth,
                ),
                borderRadius: BorderRadius.circular(12),
              ),
            ));
      }),
    ),
  );
}
```

the cart: event source

```
Widget build(BuildContext context) {
  var bloc = Provider.of<CartBloc>(context);
  int totalCount = 0;
    if (bloc.cart.length > 0) {
      totalCount = bloc.cart.values.reduce((a, b) => a + b);
  }
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
      actions: <Widget>[
          Container(
              child: GestureDetector(
                onTap: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => CartPage(),
                    ),
                  );
                },
                child: _buildCartIcon(totalCount)  // icon with text overlay
              )
            ),
        )
      ],
```

Can be event source i.e. trigger operations that (may) change state

```
body: GridView.count(
  crossAxisCount: 2,
  children: List.generate(6, (index) {
    return GestureDetector(
        onTap: () {
          bloc.addToCart(index);
        },
        child: Container(
          decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage("assets/${index + 1}.jpg"),
              fit: BoxFit.fitWidth,
            ),
            borderRadius: BorderRadius.circular(12),
          ),
        ));
  }),
),
);
}
```

The cart page

```
class CartPage extends StatelessWidget {
  CartPage({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    var bloc = Provider.of<CartBloc>(context);
    var cart = bloc.cart;
    return Scaffold(
      appBar: AppBar(
        title: Text("Shopping Cart"),
      ),
      body: ListView.builder(
        itemCount: cart.length,
        itemBuilder: (context, index) {
          int giftIndex = cart.keys.toList()[index];
          int count = cart[giftIndex];
          return ListTile(
            leading: Container(
              height: 70,
              width: 70,
              decoration: BoxDecoration(
                image: DecorationImage(
                  image: AssetImage("assets/${giftIndex + 1}.jpg"),
                  fit: BoxFit.fitWidth,
                ),
                borderRadius: BorderRadius.circular(12),
              ),
            ),
            title: Text('Item Count: $count'),
            trailing: RaisedButton(
              child: Text('Clear'),
              color: Theme.of(context).buttonColor,
              elevation: 1.0,
              splashColor: Colors.blueGrey,
              onPressed: () {
                bloc.clear(giftIndex);
              },
            ),
```

Need cart details, get bloc

Present cart information

This is not gift any more,
Warn bloc

Bloc State Management with Easy Approach

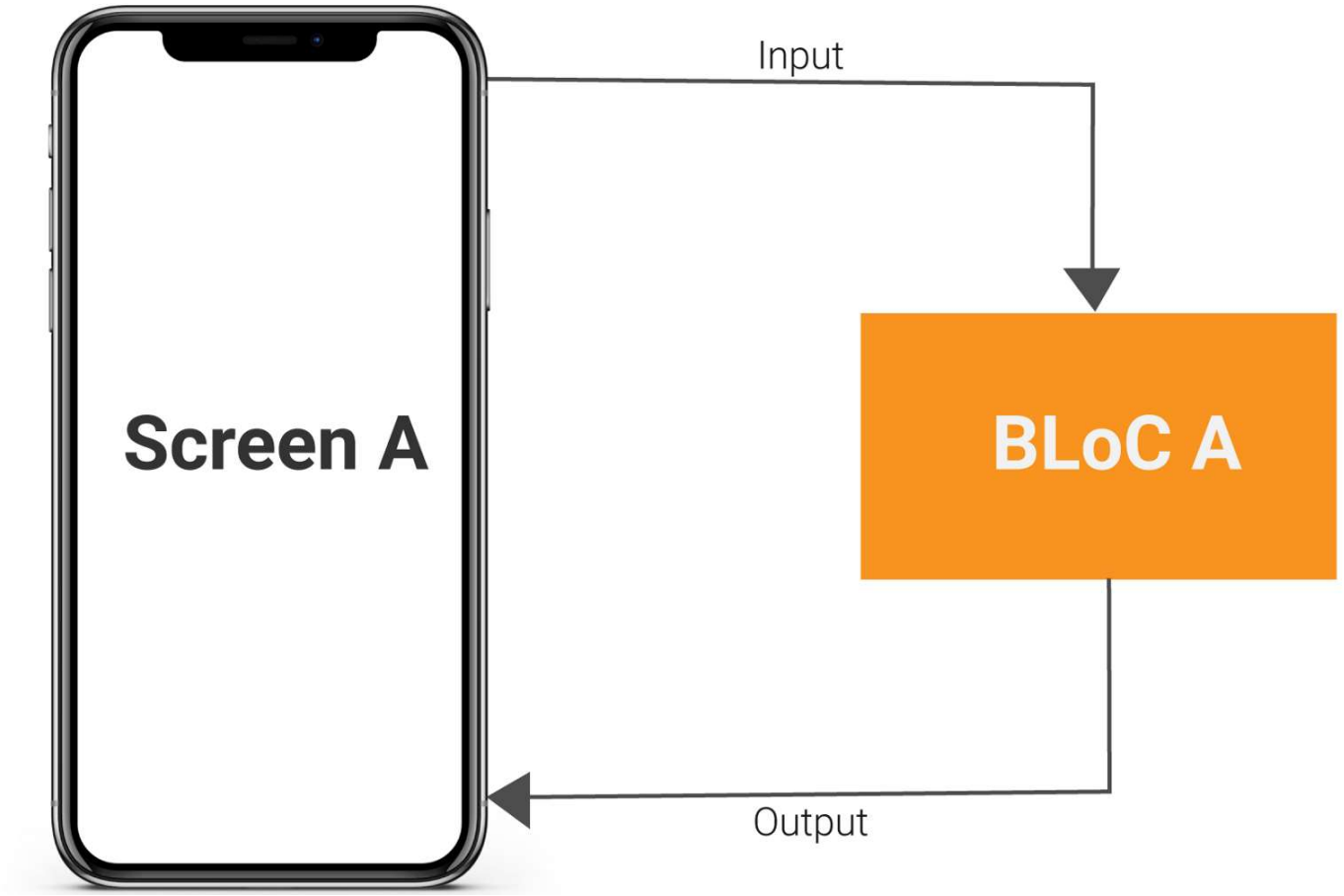
Maaz Aftab Follow
Feb 16 · 5 min read

- Simple example
- Using bloc
- Define events and states

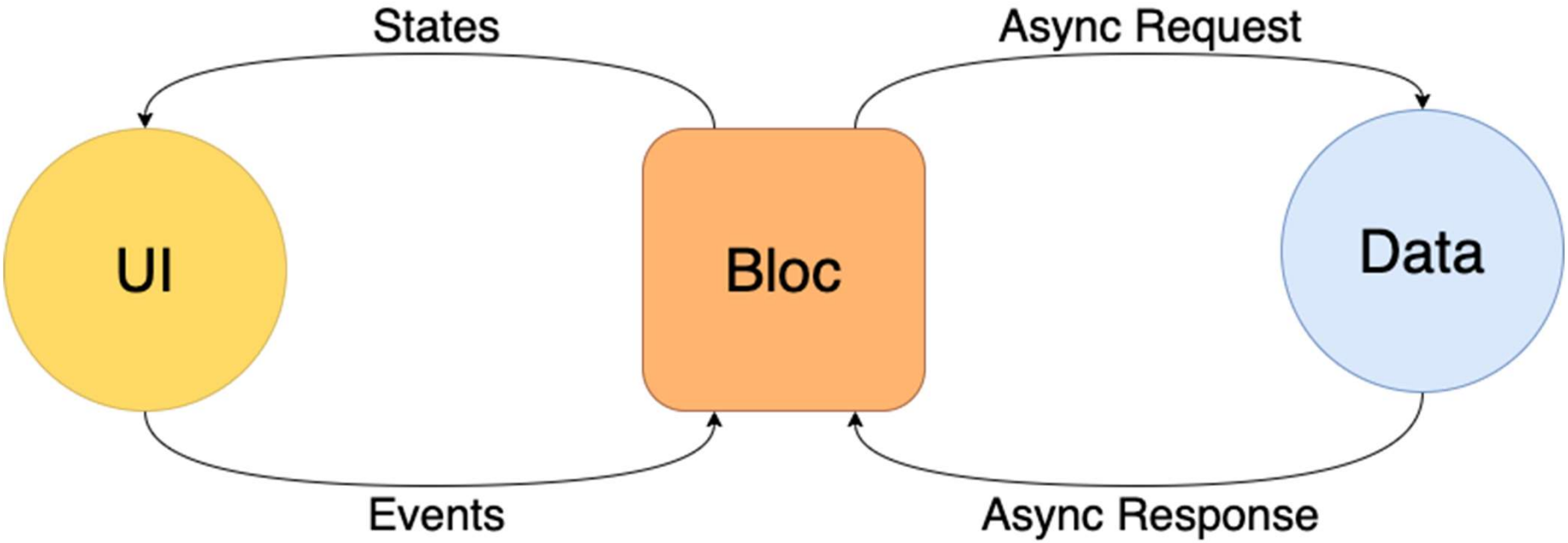
Example 2

Bloc State Management with Easy Approach

Maaz Aftab Follow
Feb 16 · 5 min read



https://medium.com/flutterpub/bloc-state-management-with-easy-approach-b53fb6d15829


States
Async Request
UI
Bloc
Data
Events
Async Response

Glossary

Events are the input to a Bloc. They are commonly UI events such as button presses. `Events` are `added` to the Bloc and then converted to `States`.

States are the output of a Bloc. Presentation components can listen to the stream of states and redraw portions of themselves based on the given state (see `BlocBuilder` for more details).

Transitions occur when an `Event` is `added` after `mapEventToState` has been called but before the `Bloc`'s state has been updated. A `Transition` consists of the currentState, the event which was added, and the nextState.

BlocSupervisor oversees `Bloc`s and delegates to `BlocDelegate`.

BlocDelegate handles events from all `Bloc`s which are delegated by the `BlocSupervisor`. Can be used to intercept all `Bloc` events, transitions, and errors. **It is a great way to handle logging/analytics as well as error handling universally**.

```dart
enum CounterEvent { increment, decrement }
```

```dart
class CounterBloc extends Bloc<CounterEvent, int> {
  @override
  int get initialState => 0;

  @override
  Stream<int> mapEventToState(CounterEvent event) async* {
    switch (event) {
      case CounterEvent.decrement:
        yield state - 1;
        break;
      case CounterEvent.increment:
        yield state + 1;
        break;
    }
  }
}
```

```dart
void main() {
  final counterBloc = CounterBloc();

  counterBloc.add(CounterEvent.increment);
  counterBloc.add(CounterEvent.increment);
  counterBloc.add(CounterEvent.increment);

  counterBloc.add(CounterEvent.decrement);
  counterBloc.add(CounterEvent.decrement);
  counterBloc.add(CounterEvent.decrement);
}
```

Events are the input to a Bloc. They are commonly UI events such as button presses. `Events` are `added` to the Bloc and then converted to `States`.

States are the output of a Bloc. Presentation components can listen to the stream of states and redraw portions of themselves based on the given state (see `BlocBuilder` for more details).

```dart
void main() {
  final counterBloc = CounterBloc();

  counterBloc.add(CounterEvent.increment);
  counterBloc.add(CounterEvent.increment);
  counterBloc.add(CounterEvent.increment);

  counterBloc.add(CounterEvent.decrement);
  counterBloc.add(CounterEvent.decrement);
  counterBloc.add(CounterEvent.decrement);
}
```

```dart
// { currentState: 0, event: CounterEvent.increment, nextState: 1 }
// { currentState: 1, event: CounterEvent.increment, nextState: 2 }
// { currentState: 2, event: CounterEvent.increment, nextState: 3 }

// { currentState: 3, event: CounterEvent.decrement, nextState: 2 }
// { currentState: 2, event: CounterEvent.decrement, nextState: 1 }
// { currentState: 1, event: CounterEvent.decrement, nextState: 0 }
```

BlocDelegate Interface

onEvent is a method that can be overridden to handle whenever an `Event` is added to **any** `Bloc` . **It is a great pla**

onTransition is a meth
in **any** `Bloc` . **It is a gr**

onError is a method th
any `Bloc` . **It is a grea**

```dart
class SimpleBlocDelegate extends BlocDelegate {
  @override
  void onEvent(Bloc bloc, Object event) {
    super.onEvent(bloc, event);
    print(event);
  }

  @override
  void onTransition(Bloc bloc, Transition transition) {
    super.onTransition(bloc, transition);
    print(transition);
  }

  @override
  void onError(Bloc bloc, Object error, StackTrace stacktrace) {
    super.onError(bloc, error, stacktrace);
    print('$error, $stacktrace');
  }
}
```

Transitions occur when an `Event` is `added` after `mapEventToState` has been called but before the `Bloc` 's state has been updated. A `Transition` consists of the currentState, the event which was added, and the nextState.

```
void main() {
  BlocSupervisor.delegate = SimpleBlocDelegate();

  final counterBloc = CounterBloc();

  counterBloc.add(CounterEvent.increment); // { currentState: 0, event:
  counterBloc.add(CounterEvent.increment); // { currentState: 1, event:
  counterBloc.add(CounterEvent.increment); // { currentState: 2, event:

  counterBloc.add(CounterEvent.decrement); // { currentState: 3, event:
  counterBloc.add(CounterEvent.decrement); // { currentState: 2, event:
  counterBloc.add(CounterEvent.decrement); // { currentState: 1, event:
}
```

BlocSupervisor oversees `Bloc` s and delegates to `BlocDelegate` .

BlocDelegate handles events from all `Bloc` s which are delegated by the `BlocSupervisor` . Can be used to intercept all `Bloc` events, transitions, and errors. **It is a great way to handle logging/analytics as well as error handling universally**.

```dart
class SimpleBlocDelegate extends BlocDelegate {
  @override
  void onEvent(Bloc bloc, Object event) {
    super.onEvent(bloc, event);
    print('bloc: ${bloc.runtimeType}, event: $event');
  }

  @override
  void onTransition(Bloc bloc, Transition transition) {
    super.onTransition(bloc, transition);
    print('bloc: ${bloc.runtimeType}, transition: $transitior
  }

  @override
  void onError(Bloc bloc, Object error, StackTrace stacktrace
    super.onError(bloc, error, stacktrace);
    print('bloc: ${bloc.runtimeType}, error: $error');
  }
}
```

```dart
void main() {
  BlocSupervisor.delegate = SimpleBlocDelegate();

  final counterBloc = CounterBloc();

  counterBloc.add(CounterEvent.increment);
  counterBloc.add(CounterEvent.increment);
  counterBloc.add(CounterEvent.increment);

  counterBloc.add(CounterEvent.decrement);
  counterBloc.add(CounterEvent.decrement);
  counterBloc.add(CounterEvent.decrement);

  counterBloc.add(null); // Triggers Exception

  // The exception triggers `SimpleBlocDelegate.onE
  // but does not impact bloc functionality.
  counterBloc.add(CounterEvent.increment);
  counterBloc.add(CounterEvent.decrement);
}
```

https://github.com/felangel/bloc/blob/master/packages/bloc/example/main.dart

Need to define events that matter

```dart
//Base Class for Bloc Event extends Equatable to make it comparable
abstract class CounterBlocEvent extends Equatable{

}

class DecreaseCounterEvent extends CounterBlocEvent{

  //overide this method when class extends equatable

  @override
  // TODO: implement props
  List<Object> get props => [];
}


class IncreaseCounterEvent extends CounterBlocEvent{

//overide this method when class extends equatable

  @override
  // TODO: implement props
  List<Object> get props => [];
}
```

https://medium.com/flutterpub/bloc-state-management-with-easy-approach-b53fb6d15829

And the relevant state

```dart
abstract class CounterBlocState extends Equatable{

}

class LatestCounterState extends CounterBlocState{
  final int newCounterValue;

  LatestCounterState({this.newCounterValue});

  //overide this method as base class extends equatable and pass property inside props list
  @override
  // TODO: implement props
  List<Object> get props => [newCounterValue];
}
```

https://medium.com/flutterpub/bloc-state-management-with-easy-approach-b53fb6d15829

To extend bloc class

- must implement two methods
  - **initialState** (to inialize first state before any transaction).
  - **mapEventToState** (to map event in to correspond state).

```
CounterBlocState get initialState => LatestCounterState(newCounterValue: 0);
```

https://medium.com/flutterpub/bloc-state-management-with-easy-approach-b53fb6d15829

Event to state

```dart
@override
Stream<CounterBlocState> mapEventToState(CounterBlocEvent event) async*{

  // TODO: implement mapEventToState
  if(event is IncreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue + 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }else if(event is DecreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue - 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }
```

https://medium.com/flutterpub/bloc-state-management-with-easy-approach-b53fb6d15829

Event to state

An event to state machine

```
@override
Stream<CounterBlocState> mapEventToState(CounterBlocEvent event) async*{

  // TODO: implement mapEventToState
  if(event is IncreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue + 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }else if(event is DecreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue - 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }
```

https://medium.com/flutterpub/bloc-state-management-with-easy-approach-b53fb6d15829

Flutter Essential: Bloc, Networking (REST API call), Unit Testing, Code Coverage.

Everything you need to know in one tutorial.

Kustiawanto Halim Follow

Dec 28, 2019 · 10 min read

- Dealling with external REST

Example 3

Flutter Essential: Bloc, Networking (REST API call), Unit Testing, Code Coverage.

Everything you need to know in one tutorial.

Kustiawanto Halim Follow

Dec 28, 2019 · 10 min read

WORTH A LOOK

3

We will only use one endpoint, https://quote-garden.herokuapp.com/quotes/random to get a random q… it in our app. The response structure of the endpoint is simple J… follows:

1. Project Setup
2. REST API
3. Data Model
4. Data Provider
5. Repository
6. Business Logic (Bloc)
7. Presentation
8. Unit Testing
9. Code Coverage
10. What's next?

Implicit:
Async calls
Accessing a REST service
Parsing json

```
Future<Quote> fetchQuote() async {
  final url = '$_baseUrl/quotes/random';
  final response = await this.httpClient.get(url);

  if (response.statusCode != 200) {
    throw new Exception('error getting quotes');
  }

  final json = jsonDecode(response.body);
  return Quote.fromJson(json);
}
```

```
class QuoteRepository {
  final QuoteApiClient quoteApiClient;

  QuoteRepository({@required this.quoteApiClient})
      : assert(quoteApiClient != null);

  Future<Quote> fetchQuote() async {
    return await quoteApiClient.fetchQuote();
  }
}
```

https://medium.com/flutter-community/flutter-essential-what-you-need-to-know-567ad25dcd8f

Flutter Essential: Bloc, Networking (REST API call), Unit Testing, Code Coverage.

Everything you need to know in one tutorial.

Kustiawanto Halim Follow

Dec 28, 2019 · 10 min read

WORTH A LOOK

3

We will only use one endpoint, https://quote-garden.herokuapp.com/quotes/random to get a random q… it in our app. The response structure of the endpoint is simple J… follows:

Implicit:
Async calls
Accessing a REST service
Parsing json

1. Project Setup
2. REST API
3. Data Model
4. Data Provider
5. Repository
6. Business Logic (Bloc)
7. Presentation
8. Unit Testing

```
dependencies:
  flutter:
    sdk: flutter
  flutter_bloc: ^3.0.0
  http: ^0.12.0
  meta: ^1.1.6
  equatable: ^1.0.0
```

```
Future<Quote> fetchQuote() async {
  final url = '$_baseUrl/quotes/random';
  final response = await this.httpClient.get(url);

  if (response.statusCode != 200) {
    throw new Exception('error getting quotes');
  }

  final json = jsonDecode(response.body);
  return Quote.fromJson(json);
```

```
class QuoteRepository {
  final QuoteApiClient quoteApiClient;

  QuoteRepository({@required this.quoteApiClient})
      : assert(quoteApiClient != null);

  Future<Quote> fetchQuote() async {
    return await quoteApiClient.fetchQuote();
  }
}
```

```
abstract class QuoteEvent extends Equatable {
  const QuoteEvent();
}

class FetchQuote extends QuoteEvent {
  const FetchQuote();

  @override
  List<Object> get props => [];
}
```

https://medium.com/flutter-community/flutter-essential-what-you-need-to-know-567ad25dcd8f

Flutter Essential: Bloc, Networking (REST API call), Unit Testing, Code Coverage.

Everything you need to know in one tutorial.

Kustiawanto Halim Follow

Dec 28, 2019 · 10 min read

The events

```dart
abstract class QuoteEvent extends Equatable {
  const QuoteEvent();
}

class FetchQuote extends QuoteEvent {
  const FetchQuote();

  @override
  List<Object> get props => [];
}
```

The state

```dart
abstract class QuoteState extends Equatable {
  const QuoteState();

  @override
  List<Object> get props => [];
}

class QuoteEmpty extends QuoteState {}

class QuoteLoading extends QuoteState {}

class QuoteLoaded extends QuoteState {
  final Quote quote;

  const QuoteLoaded({@required this.quote}) : assert(quote != null);

  @override
  List<Object> get props => [quote];
}

class QuoteError extends QuoteState {}
```

```dart
class QuoteBloc extends Bloc<QuoteEvent, QuoteState> {
  final QuoteRepository repository;

  QuoteBloc({@required this.repository}) : assert(repository != null);

  @override
  QuoteState get initialState => QuoteEmpty();

  @override
  Stream<QuoteState> mapEventToState(QuoteEvent event) async* {
    if (event is FetchQuote) {
      yield QuoteLoading();
      try {
        final Quote quote = await repository.fetchQuote();
        yield QuoteLoaded(quote: quote);
      } catch (_) {
        yield QuoteError();
      }
    }
```

https://medium.com/flutter-community/flutter-essential-what-you-need-to-know-567ad25dcd8f

Flutter Essential: Bloc, Networking (REST API call), Unit Testing, Code Coverage.

Everything you need to know in one tutorial.

Kustiawanto Halim Follow

Dec 28, 2019 · 10 min read

The events

```dart
abstract class QuoteEvent extends Equatable {
  const QuoteEvent();
}

class FetchQuote extends QuoteEvent {
  const FetchQuote();

  @override
  List<Object> get props => [];
}
```

The state

```dart
abstract class QuoteState extends Equatable {
  const QuoteState();

  @override
  List<Object> get props => [];
}

class QuoteEmpty extends QuoteState {}

class QuoteLoading extends QuoteState {}

class QuoteLoaded extends QuoteState {
  final Quote quote;

  const QuoteLoaded({@required this.quote}) : assert(quote != null);

  @override
  List<Object> get props => [quote];
}

class QuoteError extends QuoteState {}
```

```dart
class QuoteBloc extends Bloc<QuoteEvent, QuoteState> {
  final QuoteRepository repository;

  QuoteBloc({@required this.repository}) : assert(repository != null);

  @override
  QuoteState get initialState => QuoteEmpty();

  @override
  Stream<QuoteState> mapEventToState(QuoteEvent event) async* {
    if (event is FetchQuote) {
      yield QuoteLoading();
      try {
        final Quote quote = await repository.fetchQuote();
        yield QuoteLoaded(quote: quote);
      } catch (_) {
        yield QuoteError();
      }
    }
  }
```

https://medium.com/flutter-community/flutter-essential-what-you-need-to-know-567ad25dcd8f

Flutter Essential: Bloc, Networking (REST API call), Unit Testing, Code Coverage.

Everything you need to know in one tutorial.

Kustiawanto Halim Follow

Dec 28, 2019 · 10 min read

UI depends on state

```
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return BlocBuilder<QuoteBloc, QuoteState>(
      builder: (context, state) {
        if (state is QuoteEmpty) {
          BlocProvider.of<QuoteBloc>(context).add(FetchQuote());
        }
        if (state is QuoteError) {
          return Center(
            child: Text('failed to fetch quote'),
          );
        }
        if (state is QuoteLoaded) {
          return ListTile(
            leading: Text(
              '${state.quote.id}',
              style: TextStyle(fontSize: 10.0),
            ),
            title: Text(state.quote.quoteText),
            isThreeLine: true,
            subtitle: Text(state.quote.quoteAuthor),
            dense: true,
          );
        }
        return Center(
          child: CircularProgressIndicator(),
        );
      },
    );
  }
}
```

https://medium.com/flutter-community/flutter-essential-what-you-need-to-know-567ad25dcd8f

Flutter Essential: Bloc, Networking (REST API call), Unit Testing, Code Coverage.

Everything you need to know in one tutorial.

Kustiawanto Halim Follow

Dec 28, 2019 · 10 min read

Depending on state

Different widgets are produced

```
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return BlocBuilder<QuoteBloc, QuoteState>(
      builder: (context, state) {
        if (state is QuoteEmpty) {
          BlocProvider.of<QuoteBloc>(context).add(FetchQuote());
        }
        if (state is QuoteError) {
          return Center(
            child: Text('failed to fetch quote'),
          );
        }
        if (state is QuoteLoaded) {
          return ListTile(
            leading: Text(
              '${state.quote.id}',
              style: TextStyle(fontSize: 10.0),
            ),
            title: Text(state.quote.quoteText),
            isThreeLine: true,
            subtitle: Text(state.quote.quoteAuthor),
            dense: true,
          );
        }
        return Center(
          child: CircularProgressIndicator(),
        );
      },
    );
  }
}
```

https://medium.com/flutter-community/flutter-essential-what-you-need-to-know-567ad25dcd8f

FlutterDevs

OCTOBER 12, 2020 . UNCATEGORIZED

Managing the state of a Widget using bloc | Flutter

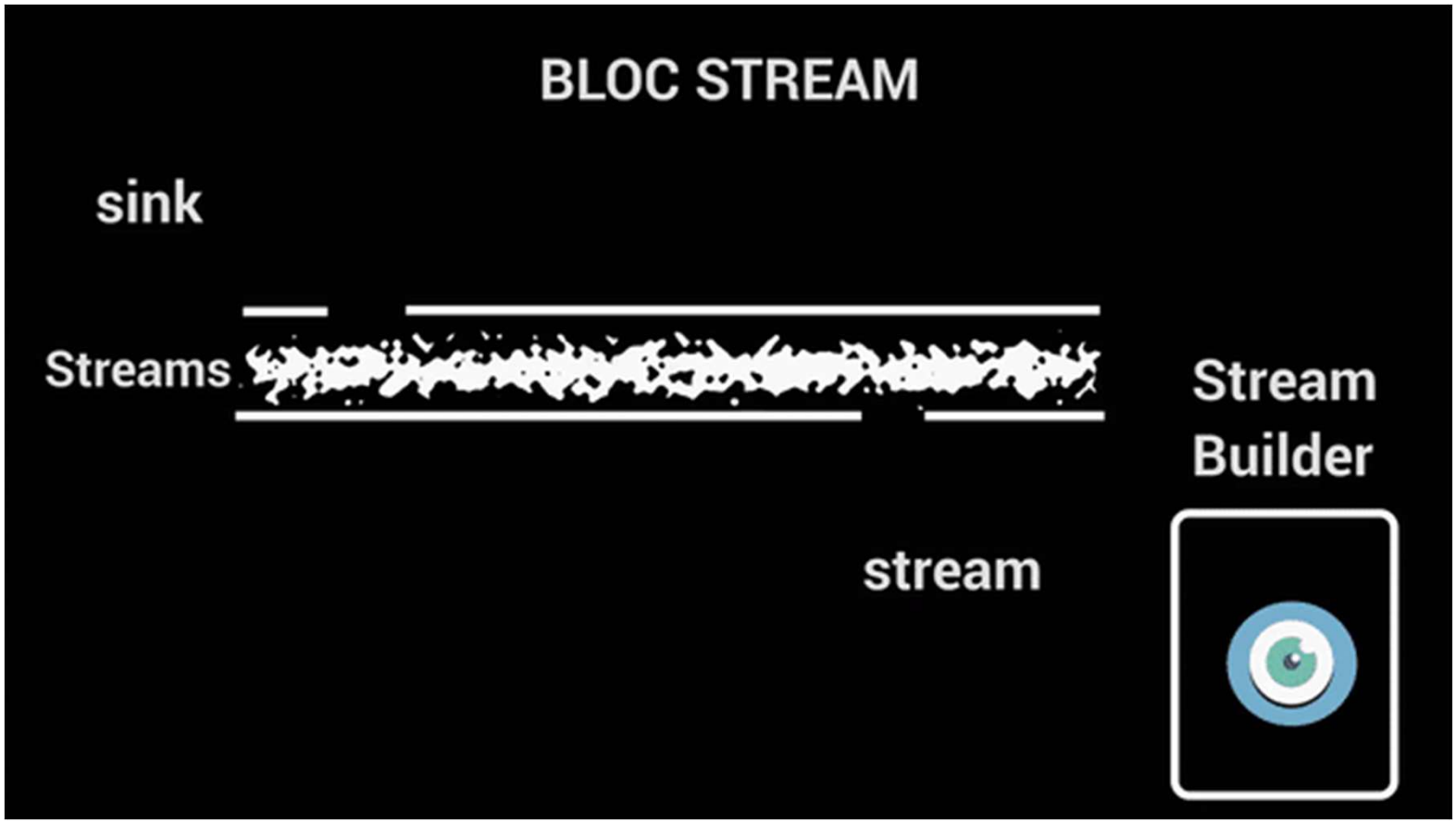
It's very hectic to manage state in flutter it when we have a complex UI and complex widget tree.

Before we jump into the state handling using bloc you need to understand about state management.

Example 4

https://flutterdevs.com/blog/managing-the-state-of-a-widget-using-bloc-flutter/

Sink and Stream.



A bloc & a provider

```dart
class NavigationDrawerBloc {
  final navigationController = StreamController();
  Stream get getNavigation => navigationController.stream;

  void updateNavigation(String navigation) {

    navigationController.sink.add("Home");

  }

  void dispose() {
    navigationController
        .close();
  }
}
```

```dart
class NavigationProvider {
  String currentNavigation = "Home";

  void updateNavigation(String navigation) {
    currentNavigation = navigation;
  }
}
```

Implement Provider class into Bloc

```
class NavigationDrawerBloc {
  final navigationController = StreamController();
  NavigationProvider navigationProvider = new NavigationProvider();

  Stream get getNavigation => navigationController.stream;

  void updateNavigation(String navigation) {
    navigationProvider.updateNavigation(navigation);
    navigationController.sink.add(navigationProvider
        .currentNavigation);
  }

  void dispose() {
    navigationController
        .close();
  }
}
```

```
class NavigationProvider {
  String currentNavigation = "Home";

  void updateNavigation(String navigation) {
    currentNavigation = navigation;
  }
}
```

Implement Provider class into Bloc

```
class NavigationDrawerBloc {
  final navigationController = StreamController();
  NavigationProvider navigationProvider = new NavigationProvider();

  Stream get getNavigation => navigationController.stream;

  void updateNavigation(String navigation) {
    navigationProvider.updateNavigation(na
    navigationController.sink.add(navigati
        .currentNavigation);
  }

  void dispose() {
    navigationController
        .close();
  }
}
```

```
class NavigationDrawerBloc {
  final navigationController = StreamController();
  Stream get getNavigation => navigationController.stream;

  void updateNavigation(String navigation) {

    navigationController.sink.add("Home");

  }

  void dispose() {
    navigationController
        .close();
  }
}
```

```
Drawer(
child: Column(
  children: <Widget>[
    UserAccountsDrawerHeader(
        accountName: Text("Admin"),
        currentAccountPicture: CircleAvatar(child: Icon(I
        accountEmail: Text("12345@gmail.com")),
    ListTile(
      title: Text("Home"),
      onTap: () {
        Navigator.of(context).pop();

        bloc.updateNavigation("Home");

      },
    ),
    ListTile(
      title: Text("Page One"),
      onTap: () {

        Navigator.of(context).pop();
        bloc.updateNavigation("PageOne");

      },
    ),
    ListTile(
      title: Text("Page Two"),
      onTap: () {

        Navigator.of(context).pop();

        bloc.updateNavigation("PageTwo");
      },
    ),
  ],
),
),
```

The scaffold (parent)


```
return Scaffold(
      appBar: AppBar(),
      drawer: Drawer(),
      body: StreamBuilder(
        stream: bloc.getNavigation,
        initialData: bloc.navigationProvider.currentNavigation,
        builder: (context, snapshot) {
          if (bloc.navigationProvider.currentNavigation == "Home") {
            return Home();
          }
          if (bloc.navigationProvider.currentNavigation == "PageOne") {
            return PageOne();
          }
          if (bloc.navigationProvider.currentNavigation == "PageTwo") {
            return PageTwo();
          }
        }, // access the data in our Stream here
      ),
    );
```

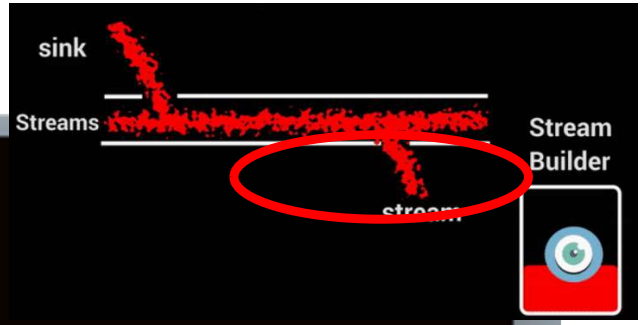
Body “looking to Stream”

```
return Scaffold(
      appBar: AppBar(),
      drawer: Drawer(),
      body: StreamBuilder(
        stream: bloc.getNavigation,
        initialData: bloc.navigationProvider.currentNavigation,
        builder: (context, snapshot) {
          if (bloc.navigationProvider.currentNavigation == "Home") {
            return Home();
          }
          if (bloc.navigationProvider.currentNavigation == "PageOne") {
            return PageOne();
          }
          if (bloc.navigationProvider.currentNavigation == "PageTwo") {
            return PageTwo();
          }
        }, // access the data in our Stream here
      ),
    );
```

The stream

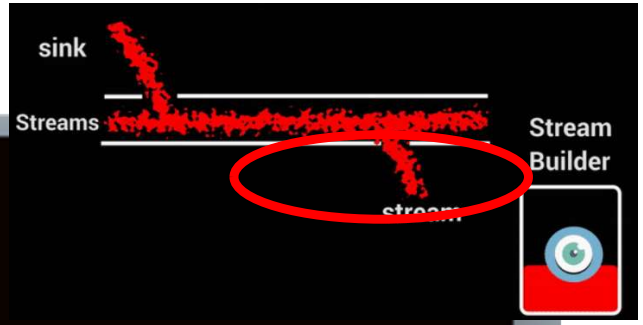

```
return Scaffold(
      appBar: AppBar(),
      drawer: Drawer(),
      body: StreamBuilder(
        stream: bloc.getNavigation,
        initialData: bloc.navigationProvider.currentNavigation,
        builder: (context, snapshot) {
          if (bloc.navigationProvider.currentNavigation == "Home") {
            return Home();
          }
          if (bloc.navigationProvider.currentNavigation == "PageOne") {
            return PageOne();
          }
          if (bloc.navigationProvider.currentNavigation == "PageTwo") {
            return PageTwo();
          }
        }, // access the data in our Stream here
      ),
    );
```

receiving



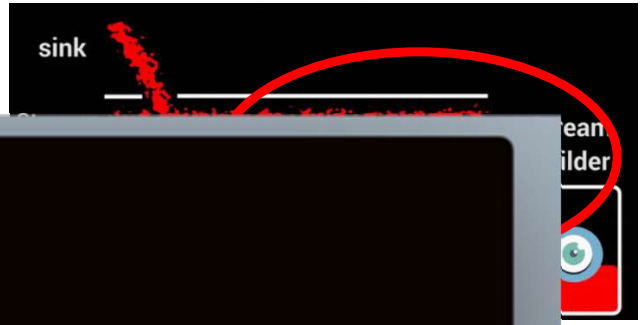

```
return Scaffold(
      appBar: AppBar(),
      drawer: Drawer(),
      body: StreamBuilder(
        stream: bloc.getNavigation,
        initialData: bloc.navigationProvider.currentNavigation,
        builder: (context, snapshot) {
          if (bloc.navigationProvider.currentNavigation == "Home") {
            return Home();
          }
          if (bloc.navigationProvider.currentNavigation == "PageOne") {
            return PageOne();
          }
          if (bloc.navigationProvider.currentNavigation == "PageTwo") {
            return PageTwo();
          }
        }, // access the data in our Stream here
      ),
    );
```

receiving




```
return Scaffold(
      appBar: AppBar(),
      drawer: Drawer(),
      body: StreamBuilder(
        stream: bloc.getNavigation,
        initialData: bloc.navigationProvider.currentNavigation,
        builder: (context, snapshot) {
          if (bloc.navigationProvider.currentNavigation == "Home") {
            return Home();
          }
          if (bloc.navigationProvider.currentNavigation == "PageOne") {
            return PageOne();
          }
          if (bloc.navigationProvider.currentNavigation == "PageTwo") {
            return PageTwo();
          }
        }, // access the data in our Stream here
      ),
    );
```

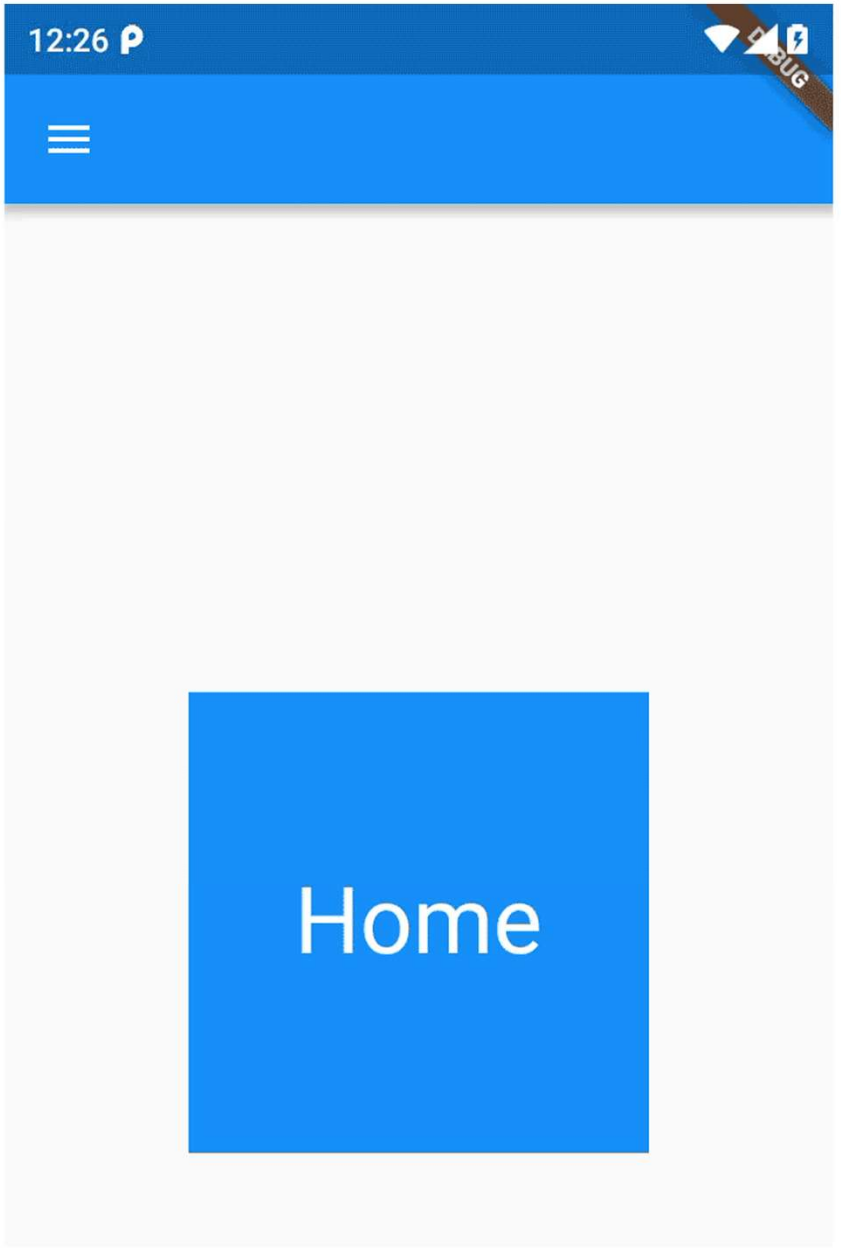
acting




```
return Scaffold(
      appBar: AppBar(),
      drawer: Drawer(),
      body: StreamBuilder(
        stream: bloc.getNavigation,
        initialData: bloc.navigationProvider.currentNavigation,
        builder: (context, snapshot) {
          if (bloc.navigationProvider.currentNavigation == "Home") {
            return Home();
          }
          if (bloc.navigationProvider.currentNavigation == "PageOne") {
            return PageOne();
          }
          if (bloc.navigationProvider.currentNavigation == "PageTwo") {
            return PageTwo();
          }
        }, // access the data in our Stream here
```

At each “change” it returns a new widget to Scaffold body
Depending on the bloc “currentNavigation”

```dart
return Scaffold(
    appBar: AppBar(),
    drawer: Drawer(),
    body: StreamBuilder(
      stream: bloc.getNavigation,
      initialData: bloc.navigationProvider.currentNavigation,
      builder: (context, snapshot) {
        if (bloc.navigationProvider.currentNavigation == "Home") {
          return Home();
        }
        if (bloc.navigationProvider.currentNavigation == "PageOne") {
          return PageOne();
        }
        if (bloc.navigationProvider.currentNavigation == "PageTwo") {
          return PageTwo();
        }
      }, // access the data in our Stream here
    ),
  );
```



Deep Patel · Jul 15, 2020 · 8 min read

Flutter: A Better Navigation Using BLoC

How to make your Flutter app navigation stateful so your user doesn't have to open a new page for every navigation item.

```
import 'package:bloc/bloc.dart';
import 'package:nice_nav/bloc/blocs.dart';

class NavDrawerBloc extends Bloc<NavDrawerEvent, NavDrawerState> {

// You can also have an optional constructor here that takes
  // a repository that you can use later to make network requests

// this is the initial state the user will see when
  // the bloc is first created
  @override
  NavDrawerState get initialState => NavDrawerState(NavItem.page_one);

  @override
  Stream<NavDrawerState> mapEventToState(NavDrawerEvent event) async*
{
    // this is where the events are handled, if you want to call a
method
    // you can yield* instead of the yield, but make sure your
    // method signature returns Stream<NavDrawerState> and is async*
    if (event is NavigateTo) {
      // only route to a new location if the new location is different
      if (event.destination != state.selectedItem) {
        yield NavDrawerState(event.destination);
      }
    }
  }
}
```

Counter example using generics

Using flutter_bloc

Example 5

Another bloc: the events

- Has state
  - counter
- A function
  - update state
  - Input: Event
  - output: the counter
- PS: all solutions fall in similar syntax…

```
enum CounterEvent { increment, decrement }

class CounterBloc extends Bloc<CounterEvent, int> {
  @override
  int get initialState => 0;

  @override
  Stream<int> mapEventToState(CounterEvent event) async*
    switch (event) {
      case CounterEvent.decrement:
        yield state - 1;
        break;
      case CounterEvent.increment:
        yield state + 1;
        break;
    }
  }
}
```

Another bloc: as generic

- Has state
  - counter
- A function
  - update state
  - Input: Event
  - output: the counter
- PS: all solutions fall in similar syntax…

```
enum CounterEvent { increment, decrement }

class CounterBloc extends Bloc<CounterEvent, int> {
  @override
  int get initialState => 0;

  @override
  Stream<int> mapEventToState(CounterEvent event) async*
    switch (event) {
      case CounterEvent.decrement:
        yield state - 1;
        break;
      case CounterEvent.increment:
        yield state + 1;
        break;
    }
  }
}
```

Another bloc: the events

- Has state
  - counter
- A function
  - update state
  - Input: Event
  - output: the counter
- PS: all solutions fall in similar syntax…

```dart
enum CounterEvent { increment, decrement }

class CounterBloc extends Bloc<CounterEvent, int> {
  @override
  int get initialState => 0;

  @override
  Stream<int> mapEventToState(CounterEvent event) async*
    switch (event) {
      case CounterEvent.decrement:
        yield state - 1;
        break;
      case CounterEvent.increment:
        yield state + 1;
        break;
    }
  }
}
```

Another bloc: event to state

- Has state
  - counter
- A function
  - update state
  - Input: Event
  - output: the counter

```
enum CounterEvent { increment, decrement }

class CounterBloc extends Bloc<CounterEvent, int> {
  @override
  int get initialState => 0;

  @override
  Stream<int> mapEventToState(CounterEvent event) async*
    switch (event) {
      case CounterEvent.decrement:
        yield state - 1;
        break;
      case CounterEvent.increment:
        yield state + 1;
        break;
    }
  }
}
```

Another bloc: the stream, async, yield

- Has state
  - counter
- A function
  - update state
  - Input: Event
  - output: the counter

```
enum CounterEvent { increment, decrement }

class CounterBloc extends Bloc<CounterEvent, int> {
  @override
  int get initialState => 0;

  @override
  Stream<int> mapEventToState(CounterEvent event) async* {
    switch (event) {
      case CounterEvent.decre
        yield state - 1;
        break;
      case CounterEvent.increment:
        yield state + 1;
        break;
    }
  }
}
```

Place the result in the stream

Results may take a while WAIT!!!

Place this in stream

Another bloc: the stream, async, yield

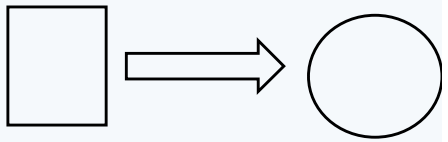
- Has state
  - counter
- A function
  - update state
  - Input: Event
  - output: the counter

???

```dart
enum CounterEvent { increment, decrement }

class CounterBloc extends Bloc<CounterEvent, int> {
  @override
  int get initialState => 0;

  @override
  Stream<int> mapEventToState(CounterEvent event) async*
    switch (event) {
      case CounterEvent.decrement:
        yield state - 1;
        break;
      case CounterEvent.increment:
        yield state + 1;
        break;
    }
  }
}
```

??

??



Counter example with state & event classes

Using https://pub.dev/packages/bloc

Example 6

Other example

```
@override
Stream<CounterBlocState> mapEventToState(CounterBlocEvent event) async*{

  // TODO: implement mapEventToState
  if(event is IncreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue + 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }else if(event is DecreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue - 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }
```

https://medium.com/flutterpub/bloc-state-management-with-easy-approach-b53fb6d15829

Other example

Using class to route Event to state

```
@override
Stream<CounterBlocState> mapEventToState(CounterBlocEvent event) async*{

  // TODO: implement mapEventToState
  if(event is IncreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue + 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }else if(event is DecreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue - 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }
```

https://medium.com/flutterpub/bloc-state-management-with-easy-approach-b53fb6d15829

Other example

Handling async

```
@override
Stream<CounterBlocState> mapEventToState(CounterBlocEvent event) async*{

  // TODO: implement mapEventToState
  if(event is IncreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue + 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }else if(event is DecreaseCounterEvent){

    //Fetching Current Counter Value From Current State
    int currentCounterValue = (state as LatestCounterState).newCounterValue;

    //Applying business Logic
    int newCounterValue = currentCounterValue - 1;

    //Adding new state to the Stream, yield is used to add state to the stream
    yield LatestCounterState(newCounterValue: newCounterValue);

  }
```

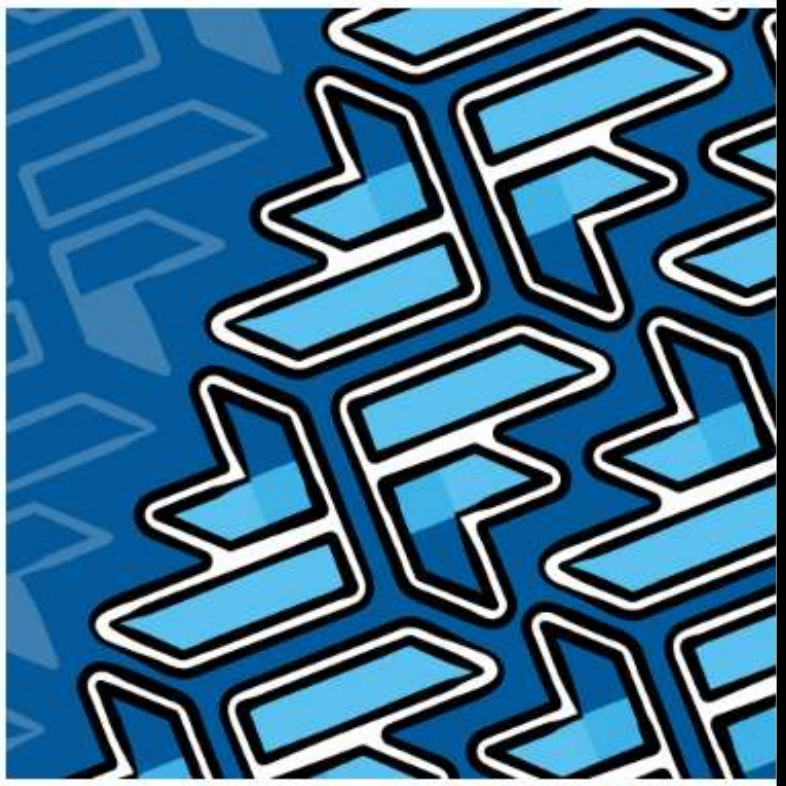
https://medium.com/flutterpub/bloc-state-management-with-easy-approach-b53fb6d15829

Getting Started with the BLoC Pattern

See how to use the popular BLoC pattern to architect your Flutter app and manage the flow of data through your widgets using Dart streams.

By Brian Kayfitz

Aug 30 2019 · Article (30 mins) · Intermediate



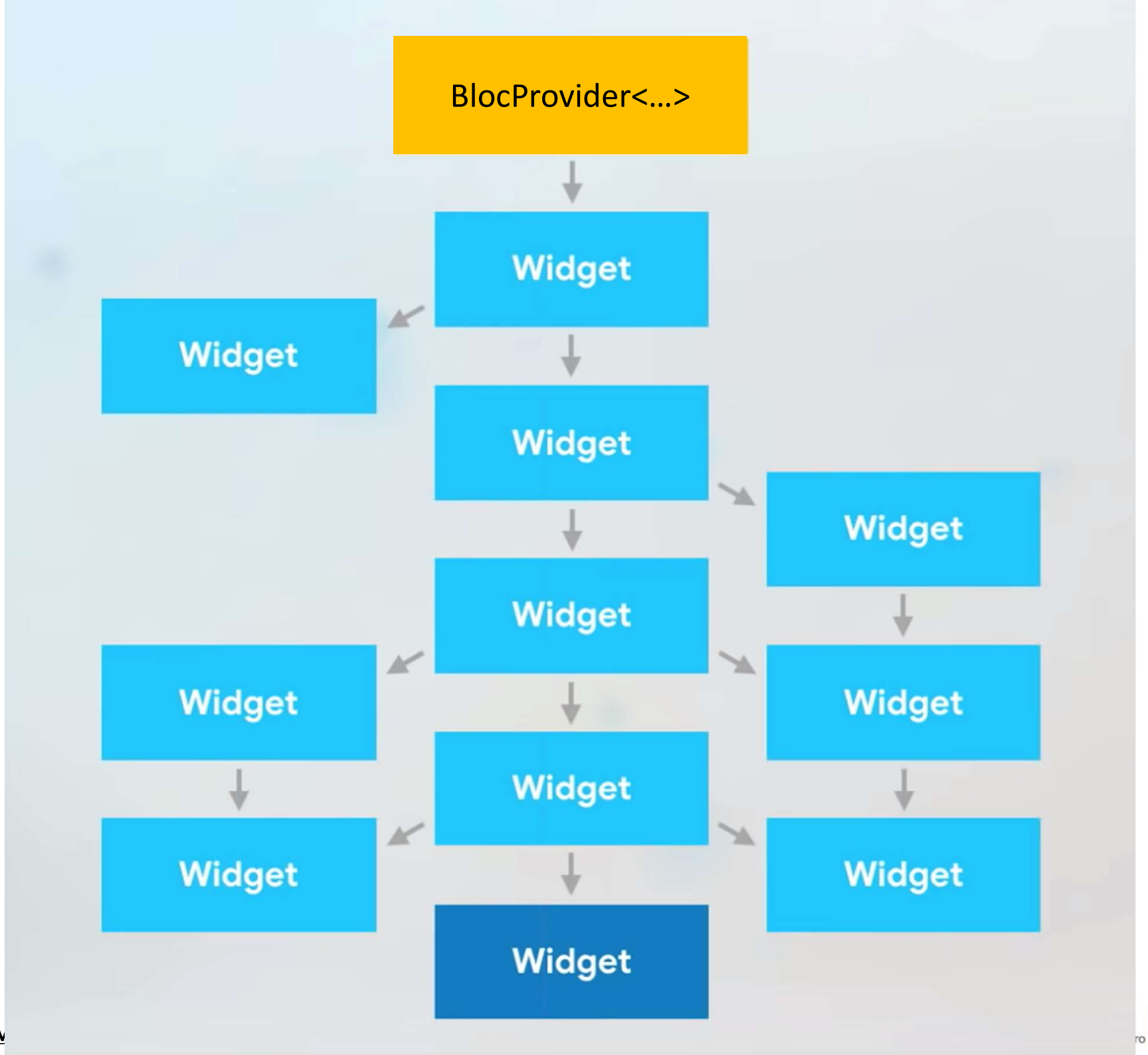
Example 7

https://www.raywenderlich.com/4074597-getting-started-with-the-bloc-pattern

```dart
class LocationBloc implements Bloc {
  Location _location;
  Location get selectedLocation => _location;

  // 1
  final _locationController = StreamController<Location>();

  // 2
  Stream<Location> get locationStream => _locationController.stream;

  // 3
  void selectLocation(Location location) {
    _location = location;
    _locationController.sink.add(location);
  }

  // 4
  @override
  void dispose() {
    _locationController.clos
  }
}
```

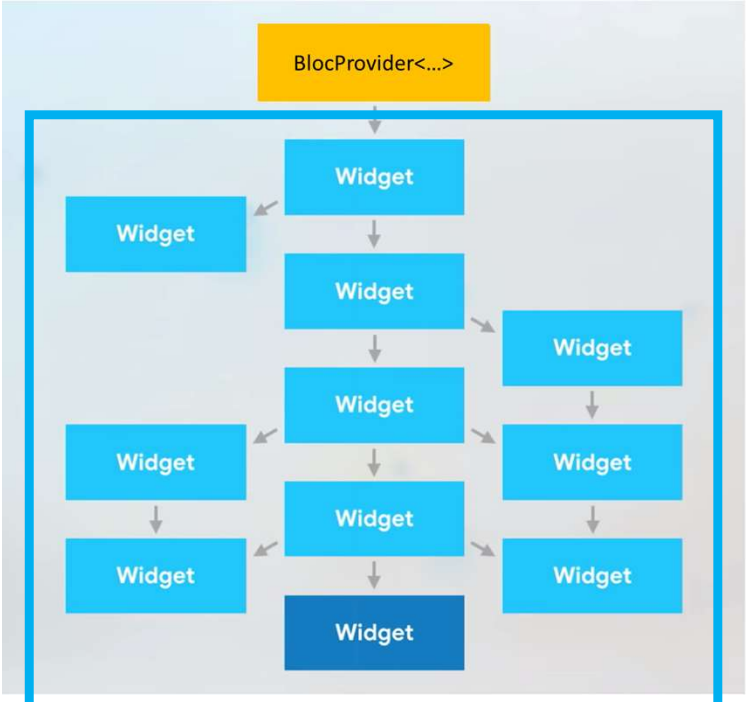
1. Here a private `StreamController` is declared that will manage the stream and sink for this BLoC. `StreamController`s use generics to tell the type system what kind of object will be emitted from the stream.
2. This line exposes a public getter to the `StreamController`'s stream.
3. This function represents the input for the BLoC. A `Location` model object will be provided as parameter that is cached in the object's private `_location` property and then added to sink for the stream.
4. Finally, in clean up method, the `StreamController` is closed when this object is deallocated. If you do not do this, the IDE will complain that the `StreamController` is leaking.

```dart
// 1
class BlocProvider<T extends Bloc> extends StatefulWidget {
  final Widget child;
  final T bloc;

  const BlocProvider({Key key, @required this.bloc, @required this.child})
      : super(key: key);

  // 2
  static T of<T extends Bloc>(BuildContext context) {
    final type = _providerType<BlocProvider<T>>();
    final BlocProvider<T> provider =
context.ancestorWidgetOfExactType(type);
    return provider.bloc;
  }

  // 3
  static Type _providerType<T>() => T;

  @override
  State createState() => _BlocProviderState();
}

class _BlocProviderState extends State<BlocProvider> {
  // 4
  @override
  Widget build(BuildContext context) => widget.child;

  // 5
  @override
  void dispose() {
    widget.bloc.dispose();
    super.dispose();
  }
}
```


BlocProvider<...>
Widget
Widget
Widget
Widget
Widget
Widget
Widget
Widget
Widget
Widget
Widget

```dart
return BlocProvider<LocationBloc>(
  bloc: LocationBloc(),
  child: MaterialApp(
    title: 'Restaurant Finder',
    theme: ThemeData(
      primarySwatch: Colors.red,
    ),
    home: MainScreen(),
  ),
);
```

```dart
class LocationBloc implements Bloc {
  Location _location;
  Location get selectedLocation => _location;

  // 1
  final _locationController = StreamController<Location>();

  // 2
  Stream<Location> get locationStream => _locationController.stream;

  // 3
  void selectLocation(Location location) {
    _location = location;
    _locationController.sink.add(location);
  }

  // 4
  @override
  void dispose() {
    _locationController.close();
  }
}
```



Flutter: BLoC with Streams

Tino Kallinich Follow

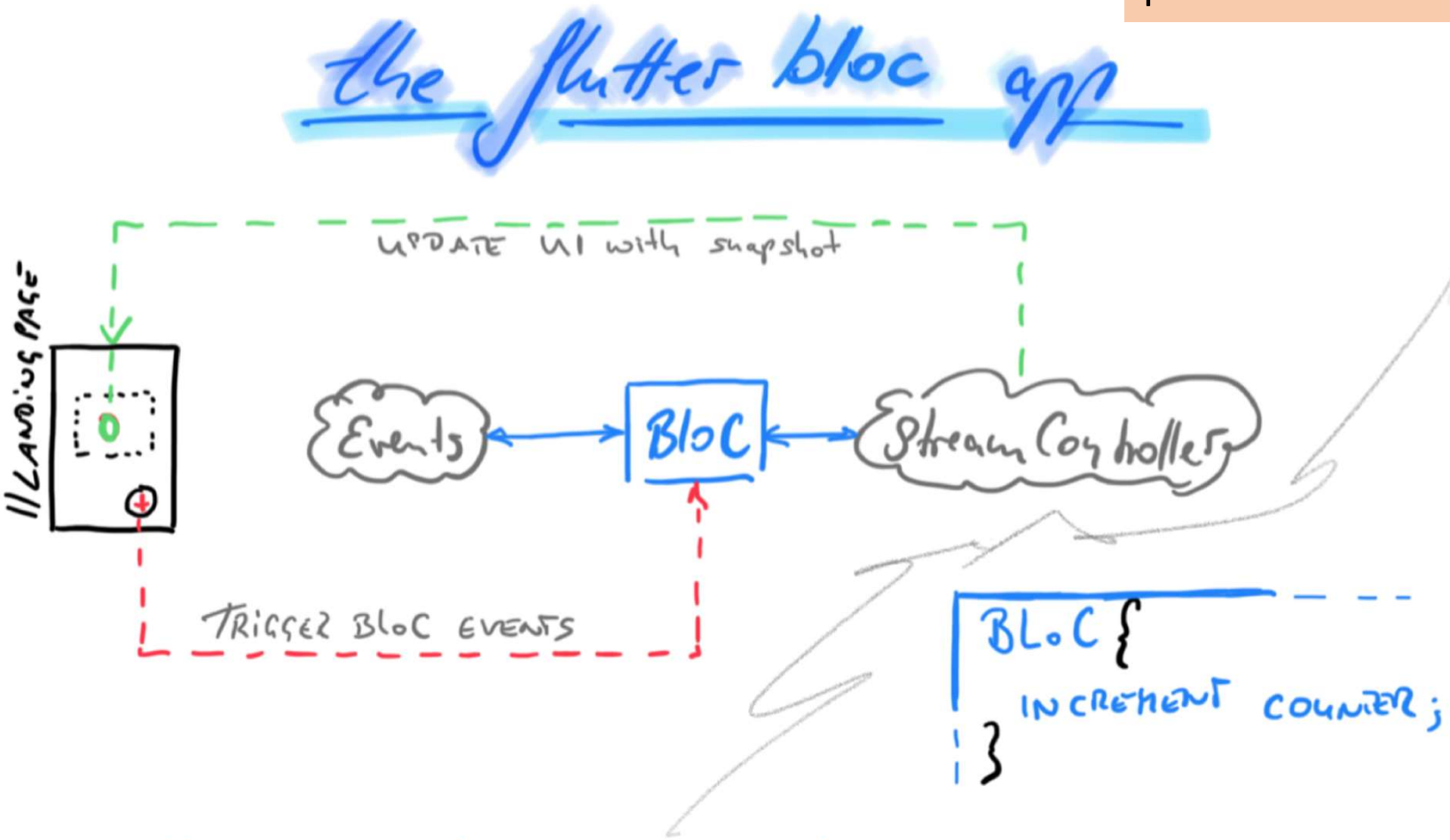
Feb 26, 2019 · 4 min read

- DIY with counter example
- Not using a framework
- Interesting to see how it could be done

Flutter: BLoC with Streams

Tino Kallinich Follow
Feb 26, 2019 · 4 min read

Good example of using Streams and bloc…
Can be a good starting point to a custom bloc

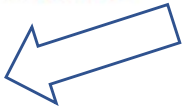


https://medium.com/flutter-community/flutter-bloc-with-streams-6ed8d0a63bb8

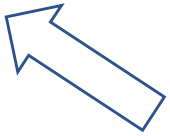
https://github.com/SeemannMx/flutter_app_bloc

Example 8

```dart
class CounterBLoC{

  int _counter = 0;

  // init and get StreamController
  final _counterStreamController = StreamController<int>();
  StreamSink<int> get counter_sink => _counterStreamController.sink;

  // expose data from stream
  Stream<int> get stream_counter => _counterStreamController.stream;

  final _counterEventController = StreamController<CounterEvent>();
  // expose sink for input events
  Sink <CounterEvent> get counter_event_sink => _counterEventController.sink;

  CounterBLoC() {  _counterEventController.stream.listen(_count);  }

  _count(CounterEvent event) => counter_sink.add(++_counter);

  dispose(){
    _counterStreamController.close();
    _counterEventController.close();
  }
}

abstract class CounterEvent{}

class IncrementEvent extends CounterEvent{}
```



Register to “listen” to stream and process incoming events with _count



In this example the state is the counter and no class uses…
A the function is obvious

Do not forget: close streams

```
class CounterBLoC{

  int _counter = 0;

  // init and get StreamController
  final _counterStreamController = StreamController<int>();
  StreamSink<int> get counter_sink => _counterStreamController.sink;

  // expose data from stream
  Stream<int> get stream_counter => _counterStreamController.stream;

  final _counterEventController = StreamController<CounterEvent>();
  // expose sink for input events
  Sink <CounterEvent> get counter_event_sink => _counterEventController.sink;

  CounterBLoC() {  _counterEventController.stream.listen(_count);  }


  _count(CounterEvent event) => counter_sink.add(++_counter);

  dispose(){
    _counterStreamController.close();
    _counterEventController.close();
  }
}
```

```
abstract class CounterEvent{}

class IncrementEvent extends CounterEvent{}
```

Using in UI

```
abstract class CounterEvent{}

class IncrementEvent extends CounterEvent{}
```

Create the bloc

```
//LandingPage.dart

final bloc = CounterBLoC();
...
onPressed: () => _bloc.counter_event_sink.add(IncrementEvent()),
...
```

Pressing trigger event
→ Event to stream

```
// Scaffold() widget

body: StreamBuilder(
    stream: _bloc.stream_counter,
    initialData: 0,
    builder: (context, snapshot) {
      return Center(child: Text(    snapshot.data.toString()    ));
    }
);
```

Builds the widget based on stream input

Using in UI

```
abstract class CounterEvent{}

class IncrementEvent extends CounterEvent{}
```

Create the bloc

```
//LandingPage.dart

final bloc = CounterBLoC();
...
onPressed: () => _bloc.counter_event_sink.add(IncrementEvent()),
...
```

Pressing trigger event
→ Event to stream

The UI depends on bloc

```
// Scaffold() widget

body: StreamBuilder(
    stream: _bloc.stream_counter,
    initialData: 0,
    builder: (context, snapshot) {
      return Center(child: Text(    snapshot.data.toString()    ));
    }
);
```

Builds based on context and on snapsho
i.e. the state send using the stream

Flutter App: fetching data from the API using the BLoC pattern architecture

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

- Not the most clear example
- Realistic retrieval and managment of contents from API

Example 9

Flutter App: fetching data from the API using the BLoC pattern architecture

Note: the rationale is agnostic to solution used to implement bloc

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

A good approach to encapsulate

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.12.0+2
  rxdart: ^0.22.0
  intl: ^0.15.8
```

```
class Repository {
  ApiProvider appApiProvider = ApiProvider();

  Future<WeatherResponse> fetchLondonWeather() =>
appApiProvider.fetchLondonWeather();
}
```

A web client

```
class ApiProvider {
  Client client = Client();
  final _baseUrl =
      "https://samples.openweathermap.org/data/2.5/weather?
q=London,uk&appid=b6907d289e10d714a6e88b30761fae22";

  Future<WeatherResponse> fetchLondonWeather() async {
    final response = await client.get("$_baseUrl");
    print(response.body.toString());

    if (response.statusCode == 200) {
      return WeatherResponse.fromJson(json.decode(response.body));
    } else {
      throw Exception('Failed to load weather');
    }
  }
}
```

https://medium.com/zipper-studios/flutter-fetch-data-from-api-and-architect-your-app-using-bloc-pattern-b826f80d6996

Flutter App: fetching data from the API using the BLoC pattern architecture

Note: the rationale is agnostic to solution used to implement bloc

9

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.12.0+2
  rxdart: ^0.22.0
  intl: ^0.15.8
```

```
class Repository {
  ApiProvider appApiProvider = ApiProvider();

  Future<WeatherResponse> fetchLondonWeather() =>
appApiProvider.fetchLondonWeather();
}
```

```
class ApiProvider {
  Client client = Client();
  final _baseUrl =
      "https://samp                                    ?
q=London,uk&appid=b6907d289e10d714a6e88b30761fae22";

  Future<WeatherResponse> fetchLondonWeather() async {
    final response = await client.get("$_baseUrl");
    print(response.body.toString());

    if (response.statusCode == 200) {
      return WeatherResponse.fromJson(json.decode(response.body));
    } else {
      throw Exception('Failed to load weather');
    }
  }
}
```

WEB can take sometime to answer

https://medium.com/zipper-studios/flutter-fetch-data-from-api-and-architect-your-app-using-bloc-pattern-b826f80d6996

Flutter App: fetching data from the API using the BLoC pattern architecture

Note: the rationale is agnostic to solution used to implement bloc

9

Loredana Petrea Follow

Nov 12, 2019 · 8 min read

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.12.0+2
  rxdart: ^0.22.0
  intl: ^0.15.8
```

```
class Repository {
  ApiProvider appApiProvider = ApiProvider();

  Future<WeatherResponse> fetchLondonWeather() =>
appApiProvider.fetchLondonWeather();
}
```

```
class ApiProvider {
  Client client = Client();
  final _baseUrl =
      "https://samp                    org/data/2.5/weather?
q=London,uk&appid=b6907d289e10d714a6e88b30761fae22";

  Future<WeatherResponse> fetchLondonWeather() async {
    final response = await client.get("$_baseUrl");
    print(response.body.toString());

    if (response.statusCode == 200) {
      return WeatherResponse.fromJson(json.decode(response.body));
    } else {
      throw Exception('Failed to load weather');
    }
  }
}
```

So we must wait

https://medium.com/zipper-studios/flutter-fetch-data-from-api-and-architect-your-app-using-bloc-pattern-b826f80d6996

Flutter App: fetching data from the API using the BLoC pattern architecture

Note: the rationale is agnostic to solution used to implement bloc

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.12.0+2
  rxdart: ^0.22.0
  intl: ^0.15.8
```

```
class Repository {
  ApiProvider appApiProvider = ApiProvider();

  Future<WeatherResponse> fetchLondonWeather() =>
appApiProvider.fetchLondonWeather();
}
```

An any client must know this and wait

```
class ApiProv
  Client clie
  final _baseUrl =
      "https://samples.openweathermap.org/data/2.5/weather?
q=London,uk&appid=b6907d289e10d714a6e88b30761fae22";

  Future<WeatherResponse> fetchLondonWeather() async {
    final response = await client.get("$_baseUrl");
    print(response.body.toString());

    if (response.statusCode == 200) {
      return WeatherResponse.fromJson(json.decode(response.body));
    } else {
      throw Exception('Failed to load weather');
    }
  }
}
```

https://medium.com/zipper-studios/flutter-fetch-data-from-api-and-architect-your-app-using-bloc-pattern-b826f80d6996

Flutter App: fetching data from the API using the BLoC pattern architecture

Note: the rationale is agnostic to solution used to implement bloc

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.12.0+2
  rxdart: ^0.22.0
  intl: ^0.15.8
```

```
class Repository {
  ApiProvider appApiProvider = ApiProvider();

  Future<WeatherResponse> fetchLondonWeather() =>
appApiProvider.fetchLondonWeather();
}
```

The result will arrive in the future

```
class ApiProv
  Client clie
  final _baseUrl =
      "https://samples.openweathermap.org/data/2.5/weather?
q=London,uk&appid=b6907d289e10d714a6e88b30761fae22";

  Future<WeatherResponse> fetchLondonWeather() async {
    final response = await client.get("$_baseUrl");
    print(response.body.toString());

    if (response.statusCode == 200) {
      return WeatherResponse.fromJson(json.decode(response.body));
    } else {
      throw Exception('Failed to load weather');
    }
  }
}
```

https://medium.com/zipper-studios/flutter-fetch-data-from-api-and-architect-your-app-using-bloc-pattern-b826f80d6996

Flutter App: fetching data from the API using the BLoC pattern architecture

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

```
class WeatherBloc {
  Repository _repository = Repository();

  final _weatherFetcher = PublishSubject<WeatherResponse>();

  Observable<WeatherResponse> get weather => _weatherFetcher.stream;

  fetchLondonWeather() async {
    WeatherResponse weatherResponse = await
_repository.fetchLondonWeather();
    _weatherFetcher.sink.add(weatherResponse);
  }

  dispose() {
    _weatherFetcher.close();
  }
}

final weatherBloc = WeatherBloc();
```

Not ideal, but can create a global accessible bloc

Flutter App: fetching data from the API using the BLoC pattern architecture

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

```
class WeatherBloc {
  Repository _repository =

  final _weatherFetcher = P

  Observable<WeatherRespons

  fetchLondonWeather() asyn
    WeatherResponse weather
_repository.fetchLondonWeat
    _weatherFetcher.sink.ad
  }

  dispose() {
    _weatherFetcher.close()
  }
}

final weatherBloc = Weather
```

```
class WeatherScreen extends StatefulWidget {
  @override
  WeatherScreenState createState() => WeatherScreenState();
}

class WeatherScreenState extends State<Wea

  @override
  Widget build(BuildContext context) {
    weatherBloc.fetchLondonWeather();
    return StreamBuilder(
        stream: weatherBloc.weather,
        builder: (context, AsyncSnapshot<WeatherResponse> snapshot) {
          if (snapshot.hasData) {
            return _buildWeatherScreen(snapshot.data);
          } else if (snapshot.hasError) {
            return Text(snapshot.error.toString());
          }
          return Center(child: CircularProgressIndicator());
        });
  }
  Container _buildWeatherScreen(WeatherResponse data) {
    return Container(
      padding: const EdgeInsets.all(17.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          _buildTitle(data.name),
          _buildCoord(data.coord),
          _buildMain(data.main),
          _buildWindInfo(data.wind),
          _buildSys(data.sys),
        ],
      ),
    );
```

using the bloc, i.e. trigger event

Flutter App: fetching data from the API using the BLoC pattern architecture

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

```
class WeatherBloc {
  Repository _repository =

  final _weatherFetcher = P

  Observable<WeatherRespons

  fetchLondonWeather() asyn
    WeatherResponse weather
_repository.fetchLondonWeat
    _weatherFetcher.sink.ad
  }

  dispose() {
    _weatherFetcher.close()
  }
}

final weatherBloc = Weather
```

```
class WeatherScreen extends StatefulWidget {
  @override
  WeatherScreenState createState() => WeatherScreenState();
}

class WeatherScreenState extends State<Wea

  @override
  Widget build(BuildContext context) {
    weatherBloc.fetchLondonWeather();
    return StreamBuilder(
      stream: weatherBloc.weather,
      builder: (context, AsyncSnapshot<WeatherResponse> snapshot) {
        if (snapshot.hasData) {
          return _buildWeatherScreen(snapshot.data);
        } else if (snapshot.hasError) {
          return Text(snapshot.error.toString());
        }
        return Center(child: CircularProgressIndicator());
      });
  }
  Container _buildWeatherScreen(WeatherResponse data) {
    return Container(
      padding: const EdgeInsets.all(17.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          _buildTitle(data.name),
          _buildCoord(data.coord),
          _buildMain(data.main),
          _buildWindInfo(data.wind),
          _buildSys(data.sys),
        ],
      ),
    );
```

build the UI based on stream

Flutter App: fetching data from the API using the BLoC pattern architecture

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

```
class WeatherBloc {
  Repository _repository =

  final _weatherFetcher = P

  Observable<WeatherRespons

  fetchLondonWeather() asyn
    WeatherResponse weather
_repository.fetchLondonWeat
    _weatherFetcher.sink.ad
  }

  dispose() {
    _weatherFetcher.close()
  }
}

final weatherBloc = Weather
```

```
class WeatherScreen extends StatefulWidget {
  @override
  WeatherScreenState createState() => WeatherScreenState();
}

class WeatherScreenState extends State<Wea

  @override
  Widget build(BuildContext context) {
    weatherBloc.fetchLondonWeather();
    return StreamBuilder(
        stream: weatherBloc.weather,
        builder: (context, AsyncSnapshot<WeatherResponse> snapshot) {
          if (snapshot.hasData) {
            return _buildWeatherScreen(snapshot.data);
          } else if (snapshot.hasError) {
            return Text(snapshot.error.toString());
          }
          return Center(child: CircularProgressIndicator());
        });
  }
  Container _buildWeatherScreen(WeatherResponse data) {
    return Container(
      padding: const EdgeInsets.all(17.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          _buildTitle(data.name),
          _buildCoord(data.coord),
          _buildMain(data.main),
          _buildWindInfo(data.wind),
          _buildSys(data.sys),
        ],
      ),
    );
```

The (available) response

Flutter App: fetching data from the API using the BLoC pattern architecture

Loredana Petrea Follow
Nov 12, 2019 · 8 min read

```
class WeatherBloc {
  Repository _repository =

  final _weatherFetcher = P

  Observable<WeatherRespons

  fetchLondonWeather() asyn
    WeatherResponse weather
_repository.fetchLondonWeat
    _weatherFetcher.sink.ad
  }

  dispose() {
    _weatherFetcher.close()
  }
}

final w
```

```
class WeatherScreen extends StatefulWidget {
  @override
  WeatherScreenState createState() => WeatherScreenState();
}

class WeatherScreenState extends State<WeatherScreen> {

  @override
  Widget build(BuildContext context) {
    weatherBloc.fetchLondonWeather();
    return StreamBuilder(
        stream: weatherBloc.weather,
        builder: (context, AsyncSnapshot<WeatherResponse> snapshot) {
          if (snapshot.hasData) {
            return _buildWeatherScreen(snapshot.data);
          } else if (snapshot.hasError) {
            return Text(snapshot.error.toString());
          }
          return Center(child: CircularProgressIndicator());
        });
  }
  Container _buildWeatherScreen(WeatherResponse data) {
    return Container(
                                    nsets.all(17.0),
                                     M
                                    [
                                    .na
                                    .co
                                    main),
                                    ata.wind),
            _buildSys(data.sys),
          ],
        ),
    );
```

The state is provided by http request status:

- Ended (hasdata),
- error
- or still downloading

Note that the UI differs as the builder build custom widgets

Weather App with “flutter_bloc”

Felix Angelov Follow

Feb 11, 2019 · 9 min read

Hey everyone, today we’re going to build a minimal (but functional) weather app in Flutter which demonstrates how to manage multiple blocs to implement dynamic theming, pull-to-refresh, and much more.

https://medium.com/flutter-community/weather-app-with-flutter-bloc-e24a7253340d

Getting Started With Flutter BLoC

Kacper Kogut
Nov 12, 2019 | 18 min read Flutter Dart BLoC

I have to admit it. My first experience with Flutter was not great. It was very unstable when I started working with it, and what put me off was the lack of architecture patterns. It was hard for me to easily structure my app, and I had to create custom logic for good communication between components. So I abandoned my Flutter projects and waited for what the time will bring.

Recently, I had to create a multiplatform application. So it was either Flutter or React Native. Since my skills in web development ended on writing "Hello

https://www.netguru.com/codestories/flutter-bloc

things have changed, and a lot of new architecture patterns came into play. I

The END