Frequently Asked Questions

State Not Updating

❔ **Question**: I'm yielding a state in my bloc but the UI is not updating. What am I doing wrong?

💡 **Answer**: If you're using Equatable make sure to pass all properties to the props getter.

✅ GOOD

Copy to clipboard

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

class StateA extends MyState {
    final String property;

    const StateA(this.property);

    @override
    List<Object> get props => [property]; // pass all properties to props
}
```

❌ BAD

dart

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

class StateA extends MyState {
    final String property;

    const StateA(this.property);

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

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

class StateA extends MyState {
    final String property;

    const StateA(this.property);

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

In addition, make sure you are yielding a new instance of the state in your bloc.

✅ GOOD

```dart
@override
Stream<MyState> mapEventToState(MyEvent event) async* {
    // always create a new instance of the state you are going to yield
    yield state.copyWith(property: event.property);
}
```

```dart
@override
Stream<MyState> mapEventToState(MyEvent event) async* {
    final data = _getData(event.info);
    // always create a new instance of the state you are going to yield
    yield MyState(data: data);
}
```

❌ BAD

```dart
@override
Stream<MyState> mapEventToState(MyEvent event) async* {
    // never modify/mutate state
    state.property = event.property;
    // never yield the same instance of state
    yield state;
}
```

> `Equatable` properties should always be copied rather than modified. If an `Equatable` class contains a `List` or `Map` as properties, be sure to use `List.from` or `Map.from` respectively to ensure that equality is evaluated based on the values of the properties rather than the reference.

When to use Equatable

❔ **Question**: When should I use Equatable?

💡 **Answer**:

```dart
@override
Stream<MyState> mapEventToState(MyEvent event) async* {
    yield StateA('hi');
    yield StateA('hi');
}
```

In the above scenario if `StateA` extends `Equatable` only one state change will occur (the second yield will be ignored). In general, you should use `Equatable` if you want to optimize your code to reduce the number of rebuilds. You should not use `Equatable` if you want the same state back-to-back to trigger multiple transitions.

In addition, using `Equatable` makes it much easier to test blocs since we can expect specific instances of bloc states rather than using `Matchers` or `Predicates`.

```dart
blocTest(
    '...',
    build: () => MyBloc(),
    act: (bloc) => bloc.add(MyEvent()),
    expect: [
        MyStateA(),
        MyStateB(),
    ],
)
```

Without `Equatable` the above test would fail and would need to be rewritten like:

```dart
blocTest(
    '...',
    build: () => MyBloc(),
    act: (bloc) => bloc.add(MyEvent()),
    expect: [
        isA<MyStateA>(),
        isA<MyStateB>(),
    ],
)
```

Handling Errors

❔ **Question**: How can I handle an error while still showing previous data?

💡 **Answer**:

This highly depends on how the state of the bloc has been modeled. In cases where data should still be retained even in the presence of an error, consider using a single state class.

```dart
enum Status { initial, loading, success, failure }

class MyState {
  const MyState({
    this.data = Data.empty,
    this.error = '',
    this.status = Status.initial,
  });

  final Data data;
  final String error;
  final Status status;

  MyState copyWith({Data data, String error, Status status}) {
    return MyState(
      data: data ?? this.data,
      error: error ?? this.error,
      status: status ?? this.status,
    );
  }
}
```

This will allow widgets to have access to the `data` and `error` properties simultaneously and the bloc can use `state.copyWith` to retain old data even when an error has occurred.

```dart
if (event is DataRequested) {
  try {
    final data = await _repository.getData();
    yield state.copyWith(status: Status.success, data: data);
  } on Exception {
    yield state.copyWith(status: Status.failure, error: 'Something went w
  }
}
```

Bloc vs. Redux

❔ **Question**: What's the difference between Bloc and Redux?

💡 **Answer**:

BLoC is a design pattern that is defined by the following rules:

1. Input and Output of the BLoC are simple Streams and Sinks.
2. Dependencies must be injectable and Platform agnostic.
3. No platform branching is allowed.
4. Implementation can be whatever you want as long as you follow the above rules.

The UI guidelines are:

1. Each "complex enough" component has a corresponding BLoC.
2. Components should send inputs "as is".
3. Components should show outputs as close as possible to "as is".
4. All branching should be based on simple BLoC boolean outputs.

The Bloc Library implements the BLoC Design Pattern and aims to abstract RxDart in order to simplify the developer experience.

The three principles of Redux are:

1. Single source of truth
2. State is read-only
3. Changes are made with pure functions

The bloc library violates the first principle; with bloc state is distributed across multiple blocs. Furthermore, there is no concept of middleware in bloc and bloc is designed to make async state changes very easy, allowing you to emit multiple states for a single event.

Bloc vs. Provider

❔ **Question**: What's the difference between Bloc and Provider?

💡 **Answer**: `provider` is designed for dependency injection (it wraps `InheritedWidget`). You still need to figure out how to manage your state (via `ChangeNotifier`, `Bloc`, `Mobx`, etc...). The Bloc Library uses `provider` internally to make it easy to provide and access blocs throughout the widget tree.

Navigation with Bloc

❔ **Question**: How do I do navigation with Bloc?

💡 **Answer**: Check out Flutter Navigation

BlocProvider.of() Fails to Find Bloc

❔ **Question**: When using `BlocProvider.of(context)` it cannot find the bloc. How can I fix this?

💡 **Answer**: You cannot access a bloc from the same context in which it was provided so you must ensure `BlocProvider.of()` is called within a child `BuildContext` .

✅ GOOD

```dart
@override
Widget build(BuildContext context) {
  BlocProvider(
    create: (_) => BlocA(),
    child: MyChild();
  );
}

class MyChild extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: () {
        final blocA = BlocProvider.of<BlocA>(context);
        ...
      },
    )
    ...
  }
}
```

```dart
@override
Widget build(BuildContext context) {
  BlocProvider(
    create: (_) => BlocA(),
    child: Builder(
      builder: (context) => ElevatedButton(
        onPressed: () {
          final blocA = BlocProvider.of<BlocA>(context);
          ...
        },
      ),
    ),
  );
}
```

❌ BAD

```dart
@override
Widget build(BuildContext context) {
  BlocProvider(
    create: (_) => BlocA(),
    child: ElevatedButton(
      onPressed: () {
        final blocA = BlocProvider.of<BlocA>(context);
        ...
      }
    )
  );
}
```

Project Structure

❔ **Question**: How should I structure my project?

💡 **Answer**: While there is really no right/wrong answer to this question, some recommended references are

- Flutter Architecture Samples - Brian Egan

- Flutter Shopping Card Example
- Flutter TDD Course - ResoCoder

The most important thing is having a **consistent** and **intentional** project structure.

‹ PREVIOUS
Naming Conventions

NEXT ›
Migration