Flutter State Management with Provider: Beginner’s Guide

Subtitle: Learn how to manage app-wide state in Flutter using the Provider package for scalable apps.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Using only `setState()` becomes messy as your app grows. **Provider** is a beginner-friendly state management solution that allows you to share data across multiple widgets without prop-drilling. This guide shows how to set it up and use it in your Flutter apps.

Step 1: Add Provider Dependency

In `pubspec.yaml`:

```
dependencies:
  provider: ^6.0.0
```

Run:

```
flutter pub get
```

Step 2: Create a Model Class

Define a simple model for the app state:

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

class Counter with ChangeNotifier {
  int _count = 0;

  int get count => _count;

  void increment() {
    _count++;
```

```
    notifyListeners();
  }

  void decrement() {
    _count--;
    notifyListeners();
  }
}
```

- `ChangeNotifier` allows widgets to listen for changes.
- `notifyListeners()` rebuilds listening widgets.

Step 3: Provide State to App

Wrap your app with **ChangeNotifierProvider**:

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

void main() {
  runApp(
    ChangeNotifierProvider(
      create: (context) => Counter(),
      child: MyApp(),
    ),
  );
}
```

Step 4: Consume State in Widgets

Use `Consumer` or `Provider.of` to access state:

Using Consumer:

```
Consumer<Counter>(
  builder: (context, counter, child) {
    return Text('Count: ${counter.count}');
  },
)
```

Using Provider.of:

```
int count = Provider.of<Counter>(context).count;
```

Step 5: Updating State

Call the model's methods inside button callbacks:

```
ElevatedButton(
  onPressed: () {
    Provider.of<Counter>(context, listen: false).increment();
  },
  child: Text("Increase"),
)
```

Exercise: Add both Increment and Decrement buttons using Provider.

Step 6: Mini Project

Build a **Counter App using Provider**:

- Display current count
- Increment and Decrement buttons
- Show count in multiple widgets (demonstrates shared state)
- Avoid prop-drilling between widgets

Step 7: Best Practices

- Use **ChangeNotifier** for small to medium apps
- Avoid placing large logic in widgets
- Keep models clean and focused
- Use multiple providers for complex apps
- Learn this before moving to Riverpod or Bloc

Key Takeaways

- Provider helps **share state across multiple widgets** efficiently
- `ChangeNotifier` + `notifyListeners()` triggers UI updates
- Avoid using `setState()` for app-wide state

- Proper state management makes your app **scalable and maintainable**

If you skip learning Provider, your Flutter apps will quickly become **hard to manage** as they grow. This is the bridge between beginner and intermediate Flutter skills.

Visit **haas.dev** for more Flutter guides, tutorials, and complete project resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
