'Navigator 2.0' and Beamer

Sandro Lovnički,
Friendly Fire

Table of Contents

“Navigator 2.0”

Why quotation marks? It’s no longer officially referred to as that, but **Router**

- https://flutter.dev/docs/development/ui/navigation/deep-linking

Version note: Navigator 2.0 is now called `Router`, which allows you to declaratively set the displayed routes based on the app’s current state. This API is opt-in.

The purpose of the new API is to replace the imperative navigation via push, pop, etc. and achieve *declarative navigation*.

“Navigator 2.0”

Why use it?

- Declarative navigation
- Deep links
- More Flutter-like
- Web URLs
- Interact with browser history

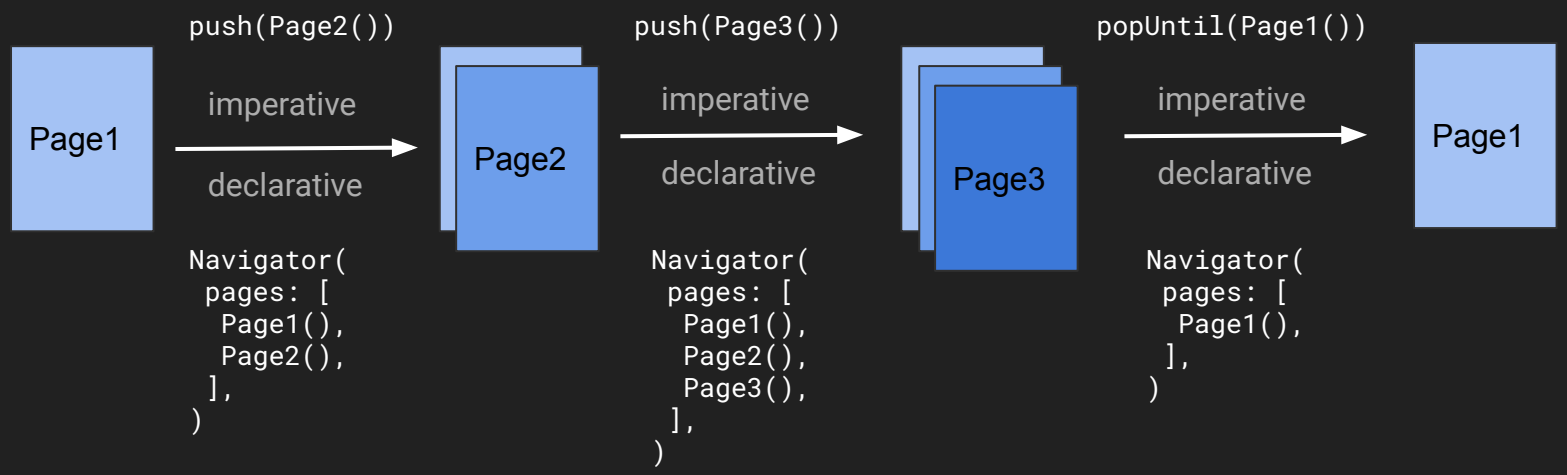
Pages API

- new property in `Navigator` widget: `pages`
  - https://api.flutter.dev/flutter/widgets/Navigator/pages.html
  - The list of pages that should be stacked on top of eachother

- new property in `Navigator` widget: `onPopPage`
  - https://api.flutter.dev/flutter/widgets/Navigator/onPopPage.html
  - What should happen when `Navigator.of(context).pop()` is invoked. Most commonly: remove the last page from the `pages` list.

In order to use the new Pages API, both need to be provided.

Imperative vs Declarative Navigation



Router

`Navigator`'s Pages API is enough to achieve declarative navigation, but we need a `Router` to use the full benefits, e.g. listening to the platform's incoming routes.

- Top-most router that interacts with platform: `MaterialApp.router()`
  - https://api.flutter.dev/flutter/material/MaterialApp/MaterialApp.router.html
  - Needs `routerDelegate` and `routeInformationParser`
  - Has all other properties the same as `MaterialApp()` constructor
- Inner (nested) routers: `Router` widget
  - https://api.flutter.dev/flutter/widgets/Router-class.html

Router

Responsible for handling the route information by creating a stack of pages that should be displayed on screen. It (re)builds a `Navigator` widget and fills its `pages` attribute (Pages API).

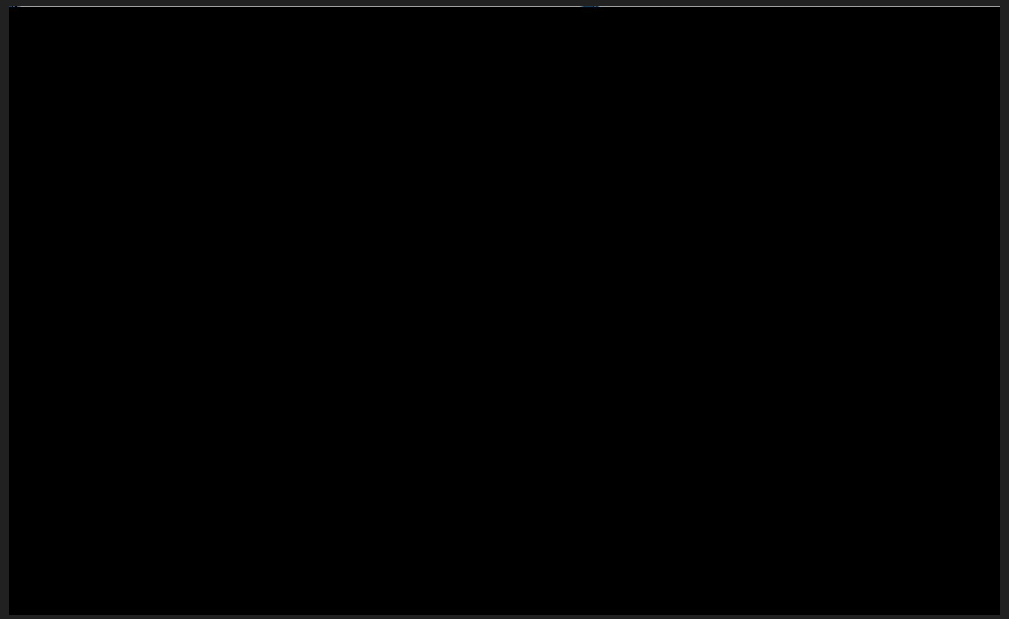
Consists of 4 building blocks, of which the most important is delegate.

- `routeInformationProvider` (defaults to `PlatformRouteInformationProvider`)
- `routeInformationParser` (parses `RouteInformation` into a type that delegate understands)
- `routerDelegate` (builds the `Navigator` widget)
- `backButtonDispatcher` (handles Android back button)

Example

https://medium.com/flutter/learning-flutters-new-navigation-and-routing-system-7c9068155ade

https://gist.github.com/johnpryan/5ce79aee5b5f83cfababa97c9cf0a204



Example
(App)

```
14  class BooksApp extends StatefulWidget {
15    @override
16    State<StatefulWidget> createState() => _BooksAppState();
17  }
18
19  class _BooksAppState extends State<BooksApp> {
20    BookRouterDelegate _routerDelegate = BookRouterDelegate();
21    BookRouteInformationParser _routeInformationParser =
22        BookRouteInformationParser();
23
24    @override
25    Widget build(BuildContext context) {
26      return MaterialApp.router(
27        title: 'Books App',
28        routerDelegate: _routerDelegate,
29        routeInformationParser: _routeInformationParser,
30      );
31    }
32  }
```

Example
(routeInformationParser)

```
34  class BookRouteInformationParser extends RouteInformationParser<BookRoutePath> {
35    @override
36    Future<BookRoutePath> parseRouteInformation(
37        RouteInformation routeInformation) async {
38      final uri = Uri.parse(routeInformation.location);
39
40      if (uri.pathSegments.length >= 2) {
41        var remaining = uri.pathSegments[1];
42        return BookRoutePath.details(int.tryParse(remaining));
43      } else {
44        return BookRoutePath.home();
45      }
46    }
47
48    @override
49    RouteInformation restoreRouteInformation(BookRoutePath path) {
50      if (path.isHomePage) {
51        return RouteInformation(location: '/');
52      }
53      if (path.isDetailsPage) {
54        return RouteInformation(location: '/book/${path.id}');
55      }
56      return null;
57    }
58  }
```

Example
(routerDelegate)

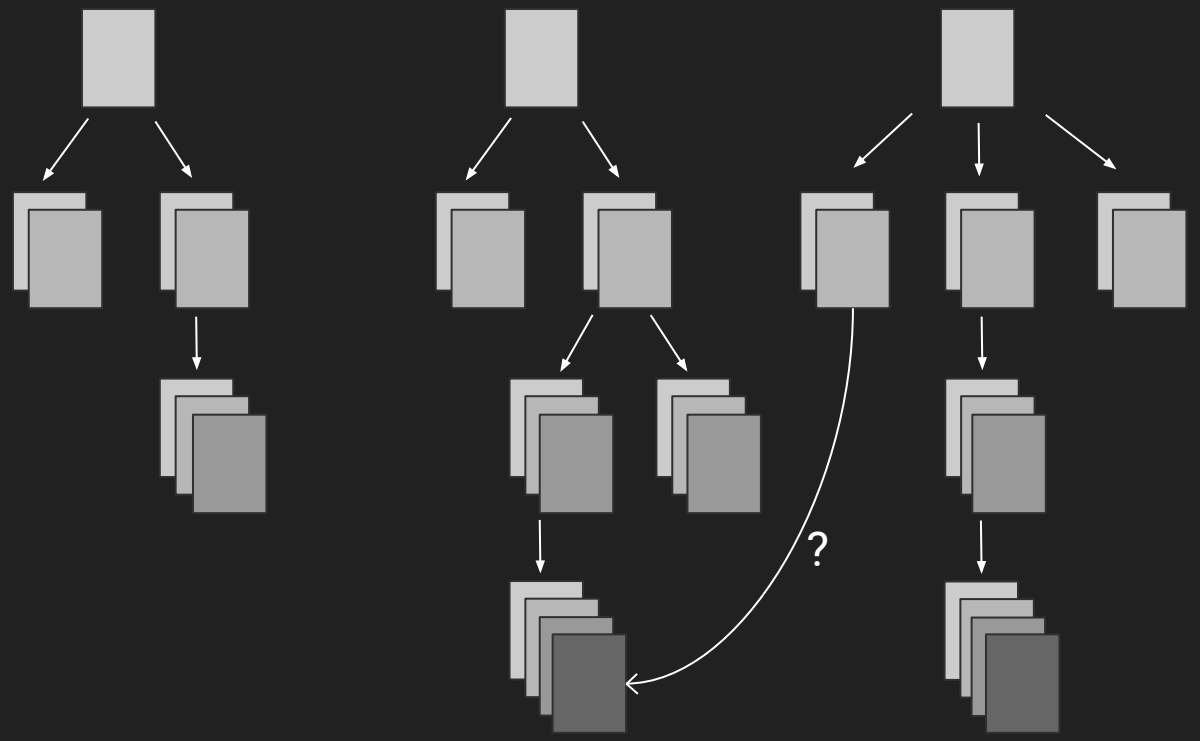
```
class BookRouterDelegate extends RouterDelegate<BookRoutePath>
    with ChangeNotifier, PopNavigatorRouterDelegateMixin<BookRoutePath> {
  final GlobalKey<NavigatorState> navigatorKey;

  Book _selectedBook;

  List<Book> books = [
    Book('Stranger in a Strange Land', 'Robert A. Heinlein'),
    Book('Foundation', 'Isaac Asimov'),
    Book('Fahrenheit 451', 'Ray Bradbury'),
  ];

  BookRouterDelegate() : navigatorKey = GlobalKey<NavigatorState>();

  BookRoutePath get currentConfiguration => _selectedBook == null
      ? BookRoutePath.home()
      : BookRoutePath.details(books.indexOf(_selectedBook));
```

```
@override
Widget build(BuildContext context) {
  return Navigator(
    key: navigatorKey,
    transitionDelegate: NoAnimationTransitionDelegate(),
    pages: [
      MaterialPage(
        key: ValueKey('BooksListPage'),
        child: BooksListScreen(
          books: books,
          onTapped: _handleBookTapped,
        ),
      ),
      if (_selectedBook != null) BookDetailsPage(book: _selectedBook)
    ],
    onPopPage: (route, result) {
      if (!route.didPop(result)) {
        return false;
      }

      // Update the list of pages by setting _selectedBook to null
      _selectedBook = null;
      notifyListeners();

      return true;
    },
  );
}

@override
Future<void> setNewRoutePath(BookRoutePath path) async {
  if (path.isDetailsPage) {
    _selectedBook = books[path.id];
  }
}

void _handleBookTapped(Book book) {
  _selectedBook = book;
  notifyListeners();
}
```

Problems

1. How to handle a large application with 20-50 screens?
2. How to provide some data to just certain page stacks? (e.g. all the pages that are shop related should have access to “cart provider”, but all the settings pages should not)
3. Too specific implementation details in parser and delegate
4. Named routes
5. Simple, beginner-friendly API

Problems



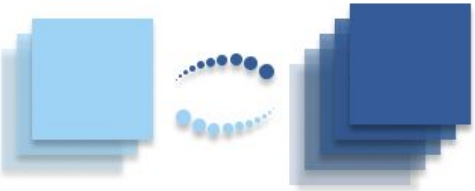
beamer 0.14.0

Published Jun 8, 2021 • beamer.dev Null safety • Latest: 0.14.0 / Prerelease: 1.0.0-pre.4.0

FLUTTER | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS

262

Readme | Changelog | Example | Installing | Versions | Scores | Admin



pub v0.14.0 | codecov 96% | style pedantic

commits 39/month | stars 186 | forks 46

closed issues 232 | closed pull requests 49

contributors 14 | chat 23 online

Buy me a coffee

Handle your application routing, synchronize it with browser URL and more. Beamer uses the power of Router and implements all the underlying logic for you.

262 LIKES | 130 PUB POINTS | 92% POPULARITY

Publisher

beamer.dev

Metadata

A routing package that lets you navigate through guarded page stacks and URLs using the Router and Navigator's Pages API, aka "Navigator 2.0".

Repository (GitHub)

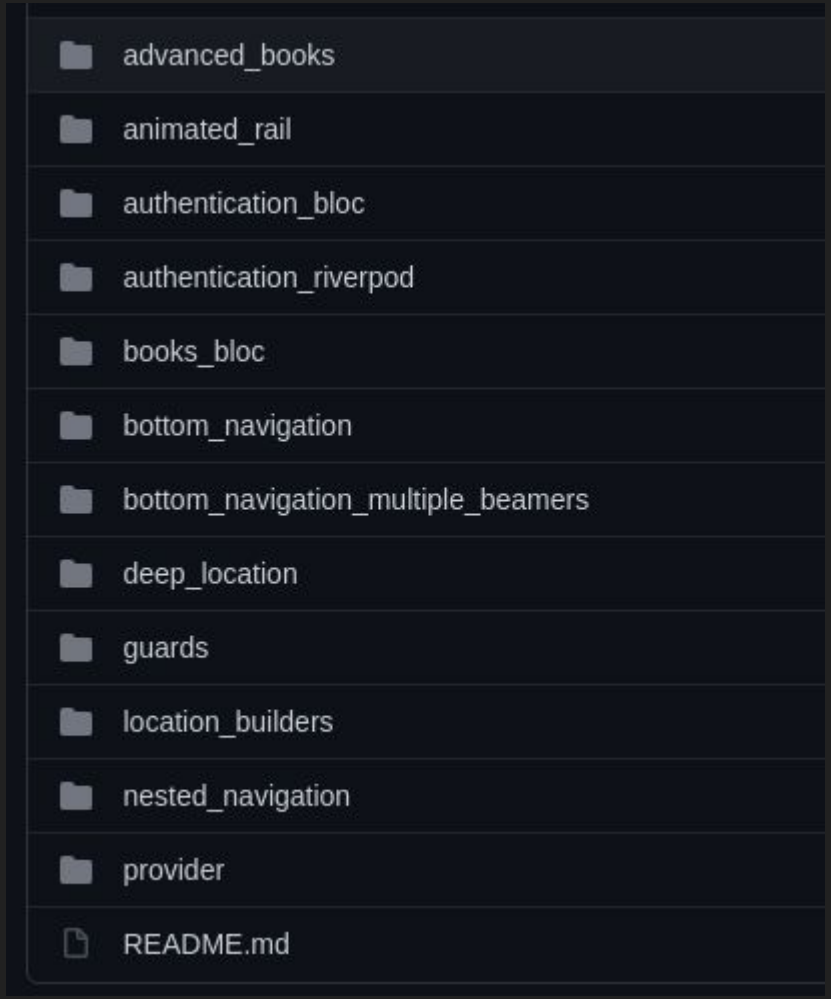
View/report issues

Documentation

API reference

Beamer

- https://pub.dev/packages/beamer
  - 262 likes
  - 92% popularity
- https://github.com/slovnicki/beamer
  - 186 stars
  - 46 forks
  - 14 contributors
- https://discord.gg/8hDJ7tP5Mz
  - 131 members



Beamer

History, motivation and goals

- Friendly Fire mobile/web/desktop app

- Full navigation control on all platforms
- Generic implementation for parser and delegate
- Separate the responsibility of building contextually different page stacks
- Robust, but simple
- Both declarative and imperative API (with declarative under the hood for both)
- Close to the original “Navigator 2.0” concept and flow (not reinventing the wheel)

Beamer

Key concepts

- `BeamerParser` (built in)
  - parses route information into `BeamState`
- `BeamerDelegate` (built in)
  - generic router delegate
  - decides `BeamLocation` via `locationBuilder` using `BeamState`
  - builds `Navigator` with pages provided by `BeamLocation`
- `BeamLocation` (developer extends it)
  - defines supported URIs
  - provides the rules for building a page stack

Beam Location

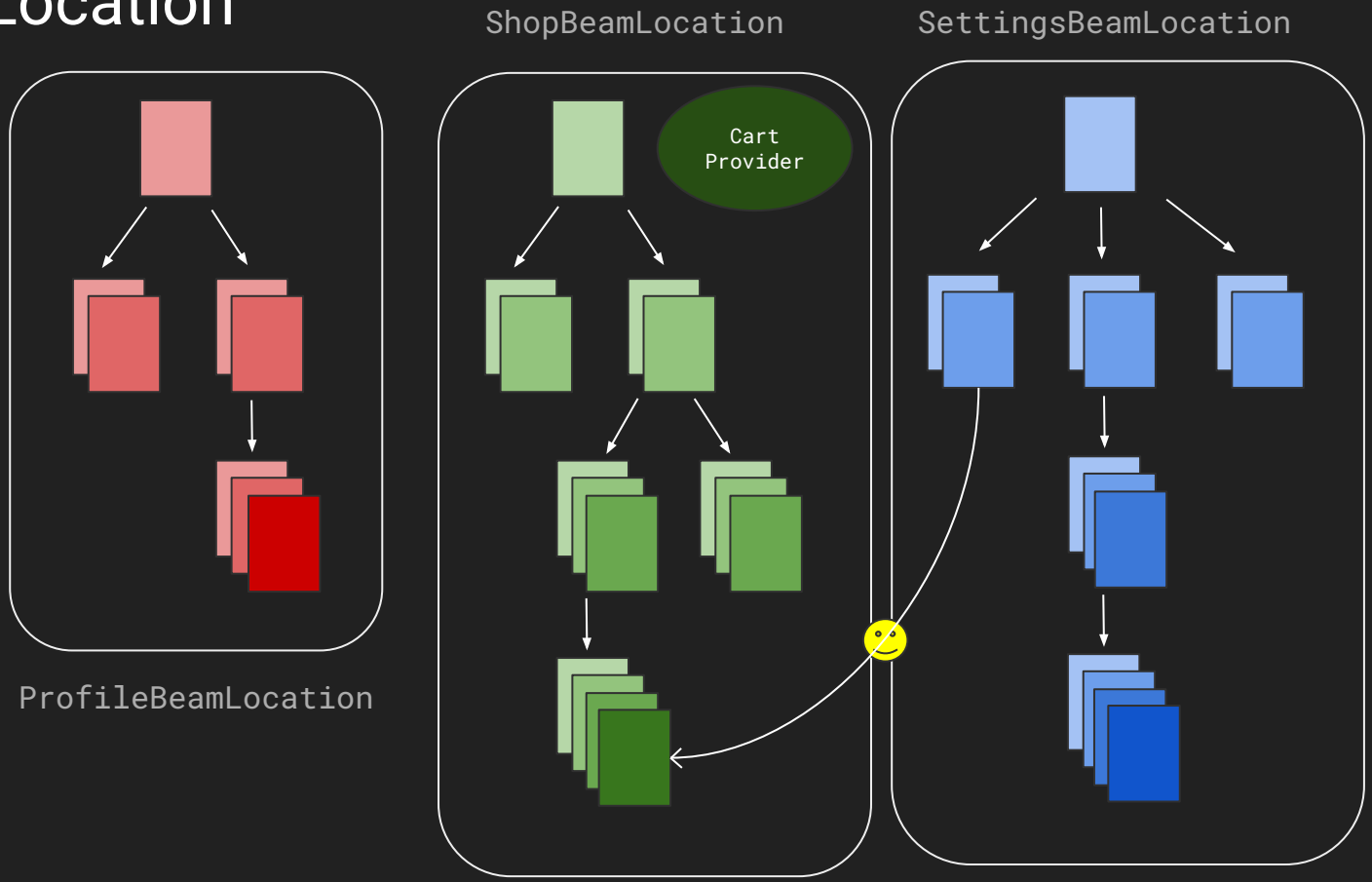
The most important construct with 3 roles:

- know which URIs it can handle: `pathBlueprints`
- know how to build a stack of pages: `buildPages`
- keep a `state` that provides a link between the first 2

The purpose of having multiple `BeamLocation`s is to architecturally separate unrelated "places" in an application.

For example, `BooksLocation` can handle all the pages related to books and ArticlesLocation everything related to articles. In the light of this scoping, `BeamLocation` also has a `builder` for wrapping an entire stack of its pages with some Provider so the similar data can be shared between similar pages.

Beam Location



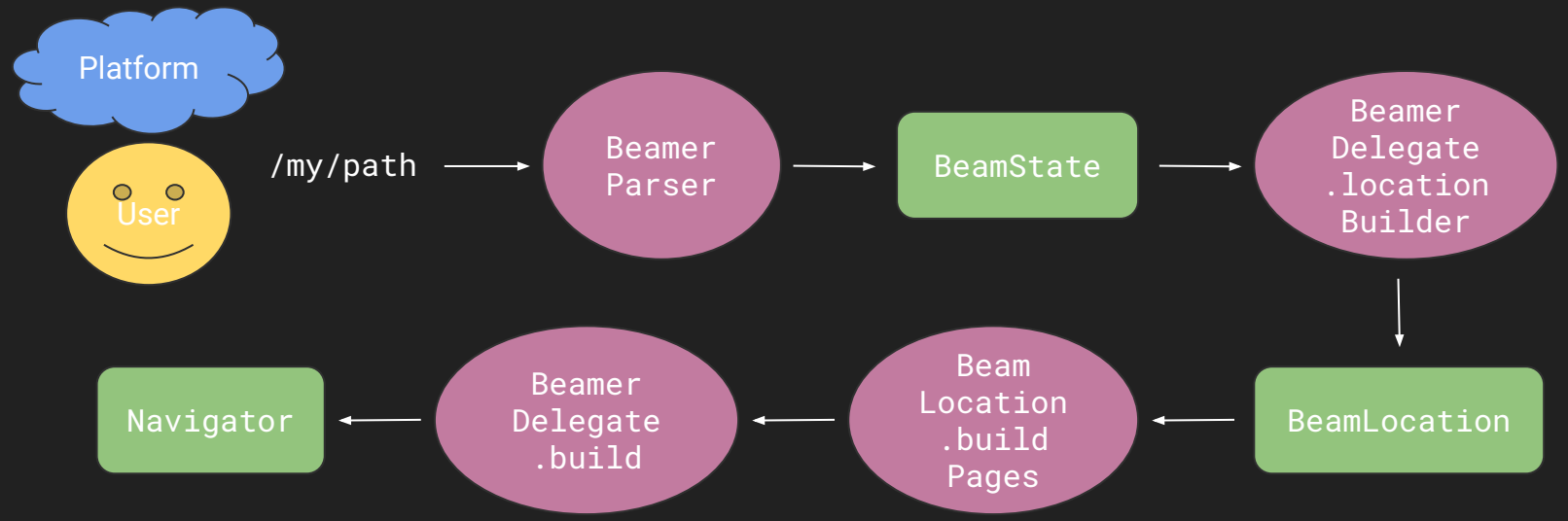
Beam Location
(example)

```
class BooksLocation extends BeamLocation {
  BooksLocation(BeamState state) : super(state);

  @override
  List<String> get pathBlueprints => ['/books/:bookId'];

  @override
  List<BeamPage> buildPages(BuildContext context, BeamState state) => [
        BeamPage(
          key: ValueKey('home'),
          child: HomeScreen(),
        ), // BeamPage
        if (state.uri.pathSegments.contains('books'))
          BeamPage(
            key: ValueKey('books'),
            child: BooksScreen(),
          ), // BeamPage
        if (state.pathParameters.containsKey('bookId'))
          BeamPage(
            key: ValueKey('book-${state.pathParameters['bookId']}'),
            child: BookDetailsScreen(
              books.firstWhere(
                  (book) => book['id'] == state.pathParameters['bookId']!),
            ), // BookDetailsScreen
          ), // BeamPage
      ];
}
```

Beam State

- A data object that represents the `state` of `Beamer` and `BeamLocation`
- Keeps various URI attributes such as `pathBlueprintSegments`, `pathParameters`, `queryParameters` and arbitrary key-value `data`
- Can be created `fromUri` and transformed `toUri`
- Used while building pages and reporting the current route to platform

Flow


Platform
User
/my/path
Beamer Parser
BeamState
Beamer Delegate .location Builder
BeamLocation
Beam Location .build Pages
Beamer Delegate .build
Navigator

Problems 1, 2 and 3 solved.

Example
(App)

```
class MyApp extends StatelessWidget {
  final routerDelegate = BeamerDelegate(
    locationBuilder: (state) => BooksLocation(state),
  ); // BeamerDelegate

  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      routerDelegate: routerDelegate,
      routeInformationParser: BeamerParser(),
    ); // MaterialApp.router
  }
}
```

Example
(BeamLocation)

```
class BooksLocation extends BeamLocation {
  BooksLocation(BeamState state) : super(state);

  @override
  List<String> get pathBlueprints => ['/books/:bookId'];

  @override
  List<BeamPage> buildPages(BuildContext context, BeamState state) => [
        BeamPage(
          key: ValueKey('home'),
          child: HomeScreen(),
        ), // BeamPage
        if (state.uri.pathSegments.contains('books'))
          BeamPage(
            key: ValueKey('books'),
            child: BooksScreen(),
          ), // BeamPage
        if (state.pathParameters.containsKey('bookId'))
          BeamPage(
            key: ValueKey('book-${state.pathParameters['bookId']}'),
            child: BookDetailsScreen(
              books.firstWhere(
                  (book) => book['id'] == state.pathParameters['bookId']!),
            ), // BookDetailsScreen
          ), // BeamPage
      ];
}
```

Example
(beaming)

- Declarative

```
Beamer.of(context).currentBeamLocation.update(
    (state) => state.copyWith(
      pathBlueprintSegments: ['books', ':bookId'],
      pathParameters: {'bookId': '3'},
    ),
  );
```

- Imperative

```
Beamer.of(context).beamToNamed('/books/3');
```

Location Builders

- SimpleLocationBuilder
  - No need for custom BeamLocations

```
final routerDelegate = BeamerDelegate(
  locationBuilder: SimpleLocationBuilder(
    routes: {
      '/': (context, state) => HomeScreen(),
      '/books': (context, state) => BooksScreen(),
      '/books/:bookId': (context, state) {
        final bookId = int.parse(state.pathParameters['bookId']!);
        return BookDetailsScreen(bookId: bookId);
      },
    },
  ), // SimpleLocationBuilder
); // BeamerDelegate
```

- BeamerLocationBuilder
  - Beamer determines the right BeamLocation

```
final routerDelegate = BeamerDelegate(
  locationBuilder: BeamerLocationBuilder(
    beamLocations: [
      BooksLocation(),
    ],
  ), // BeamerLocationBuilder
); // BeamerDelegate
```

Problems 4 and 5 solved.

Flutter UXR

https://github.com/flutter/uxr/wiki/Navigator-2.0-API-Usability-Research

Navigator 2.0 API Usability Research

Tao Dong edited this page on May 1 · 18 revisions

TL; DR

This project aims to establish a usability standard and a method for designing high-level navigation APIs for Flutter. If you have any feedback after reading this page, please post your comments to this issue. The latest status of this project can be found in issue #31. This project is also referred to as Routing API Research.

Motivation

The Navigator 2.0 API in Flutter provides many desirable enhancements on the original Navigator API, but it's also considered to be complex and hard to use by Flutter users. To simplify Navigator 2.0, the Flutter user community has started experimenting with alternate APIs. Flutter's DevRel team has also explored potential simplifications via an experimental package called page_router. We would like to make sure these explorations are fruitful and converging on an API that makes common navigation patterns straightforward to implement. To achieve this outcome, we are following the User-Centered Design process to create a high-level navigation API for Flutter.

Goals

- Design or endorse an easy-to-use package for implementing common navigation patterns, especially for use cases on the web.
- Establish a model API design process for Flutter's future development

Non-goals

In this project, we are not pursuing the following goals, but they remain a possibility in the future:

- Making the page_router package production-ready
- Changing the existing Navigator 2.0 API

Flutter UXR

- 3 finalists have been chosen for further study, one of which is beamer
  - https://github.com/flutter/uxr/issues/10#issuecomment-820669779

- Continuous study updates
  - https://github.com/flutter/uxr/issues/31

- Announcement of the winner is planned for July

Questions...

Thank you!