



dolphin

VS Code Extension User Manual

Dolphin Walkthroughs for Flutter

See the widget tree of any page, double-click a widget and turn it into a guided walkthrough step. Every dolphin option in one form, a live preview, a project map, a graph to design walkthroughs before the screens exist, and a PDF to share with your team.

1	Overview and how it works	2
2	Installation	3
3	Quick start	4
4	The widget tree	5
5	Wrapping a step	6
6	Every option in the form	8
7	Pages and sections (DolphinScope)	10
8	The app root and controller	11
9	Code or JSON config	13
10	The walkthrough map	14
11	The walkthrough graph and design first	15
12	Journeys	19
13	Exporting the map as PDF	20
14	Editor helpers	21
15	Settings and commands	23
16	Dark theme and About Dolphin	24
17	Troubleshooting	26
18	License and disclaimer	26

1 Overview and how it works

Dolphin Walkthroughs is a Visual Studio Code extension for the dolphin Flutter package. It shows the widgets of the page you are editing as a tree, and turns any widget into a walkthrough step with a double-click. You fill in a form; the extension writes the Dart code, and the JSON config when you want one.

What you get

- **Widget tree.** The widgets of the open Dart file, page by page, nested as in the code. Helper objects such as `EdgeInsets` and callbacks such as `onPressed` are left out. The tree follows your cursor.
- **Double-click to wrap.** Choose **DolphinTarget** for a step, **DolphinScope** for a page, or **Dolphin** on `MaterialApp` for the app root.
- **Every option in one form.** Content, rules, pointer position and the whole look: barrier, highlight, tooltip, text styles, pointer, counter and buttons, animation, behaviour and dark mode. Each field shows its default.
- **Live preview and code tab.** A sketch of the step in light and dark, and the exact code that will be written.
- **Code or JSON.** Keep content on the widget, or in a JSON config that the extension creates, registers and loads for you.
- **Walkthrough map and graph.** Every scope, step and journey in the project. Design new steps and journeys before the pages exist, then attach them to widgets later.
- **PDF export, checks and helpers.** A shareable map, problems while you type, completion, hover, snippets and code actions.

The three dolphin widgets

dolphin builds a walkthrough from three widgets. The extension writes all three for you.

```
Dolphin(controller)           once, above MaterialApp
└─ MaterialApp
  └─ DolphinScope(id: 'home')  one per page or section
    └─ Scaffold
      ├── DolphinTarget(id: 'search')  one per widget to highlight
      └─ DolphinTarget(id: 'profile')
```

WIDGET	ROLE	IN THE EXTENSION
<code>Dolphin</code>	Wraps the app once. Holds the <code>DolphinController</code> , which stores progress and draws the overlay above every page.	Double-click <code>MaterialApp</code> , choose Dolphin (app root) .
<code>DolphinScope</code>	Marks a page or section. It starts, pauses, resumes, skips and resets on its own.	Double-click the page widget, choose DolphinScope .
<code>DolphinTarget</code>	Wraps a widget to highlight. Needs only an <code>id</code> ; content can live on the widget or in JSON.	Double-click any widget, choose DolphinTarget .

Where things are

PLACE	WHAT IT HOLDS
Dolphin icon in the activity bar	The Widget Tree and the Walkthrough Map views.
Configure panel	The form that opens beside your code when you wrap or edit a widget.
Walkthrough Graph tab	The design surface: lanes, steps, journeys and the inspector.

Command Palette	Every action, under the Dolphin: prefix.
Editor right-click menu	A Dolphin submenu, and the light bulb menu on any widget.

Privacy. The extension works offline and collects nothing. It changes only the files you ask it to change. Only the About page reads public download and like counts, and only when you open it.

2 Installation

The extension runs on Windows, macOS and Linux, in VS Code and in editors built on it.

Requirements

ITEM	MINIMUM
VS Code	1.85
dolphin package	1.0.0
Dart extension	Recommended, for formatting after edits. The tree and forms work without it.

From the Marketplace

- 1 **Open Extensions.** Press `Ctrl+Shift+X` (`Cmd+Shift+X` on Mac).
- 2 **Search for** `Dolphin Walkthroughs` and click **Install**.
- 3 **Open your Flutter project.** The dolphin icon appears in the activity bar.

From a .vsix file

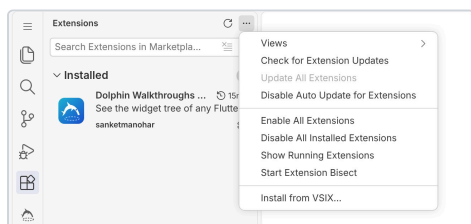


Figure 2.1 Install from VSIX in the Extensions view menu.

- 1 **Open Extensions** and click the ... menu at the top of the view.
- 2 **Choose Install from VSIX...** and pick `dolphin-walkthroughs-1.0.0.vsix`.
- 3 **Reload** when VS Code asks. The extension is listed under **Installed**.

From a terminal you can also run:

```
code --install-extension dolphin-walkthroughs-1.0.0.vsix
```

Add dolphin to your app

If the project does not use dolphin yet, run **Dolphin: Add dolphin to pubspec.yaml** from the Command Palette (`Ctrl+Shift+P`). It runs `flutter pub add dolphin` in the project folder. You can also run that command yourself.

Several projects open? The extension works per workspace folder and looks for the folder with a `pubspec.yaml`. Open the Flutter project as its own folder for the best results.

3 Quick start

From an empty project to a running walkthrough in seven steps. Each step is explained in detail in the sections that follow.

- 1 **Open the Dolphin view.** Click the dolphin icon in the activity bar. The **Widget Tree** shows the file you have open.
- 2 **Wrap the app.** Open the file with your `MaterialApp`, double-click `MaterialApp` in the tree and choose **Dolphin (app root)**. Keep the defaults and click **Wrap app**. The import, the `dolphinController` and the wrapper are added.
- 3 **Mark a page.** Open a page, double-click its `Scaffold` and choose **DolphinScope**. The id is suggested from the class name: `HomePage` becomes `home`.
- 4 **Add a step.** Double-click a button on that page and choose **DolphinTarget**. Write a title and a description.
- 5 **Check it.** Watch the preview, open the **Code** tab if you like, then click **Wrap widget**.
- 6 **Add more steps.** Repeat for every widget you want to explain. The map on the left fills in as you go.
- 7 **Run the app.** The walkthrough starts by itself when the page opens and remembers what each user has seen.

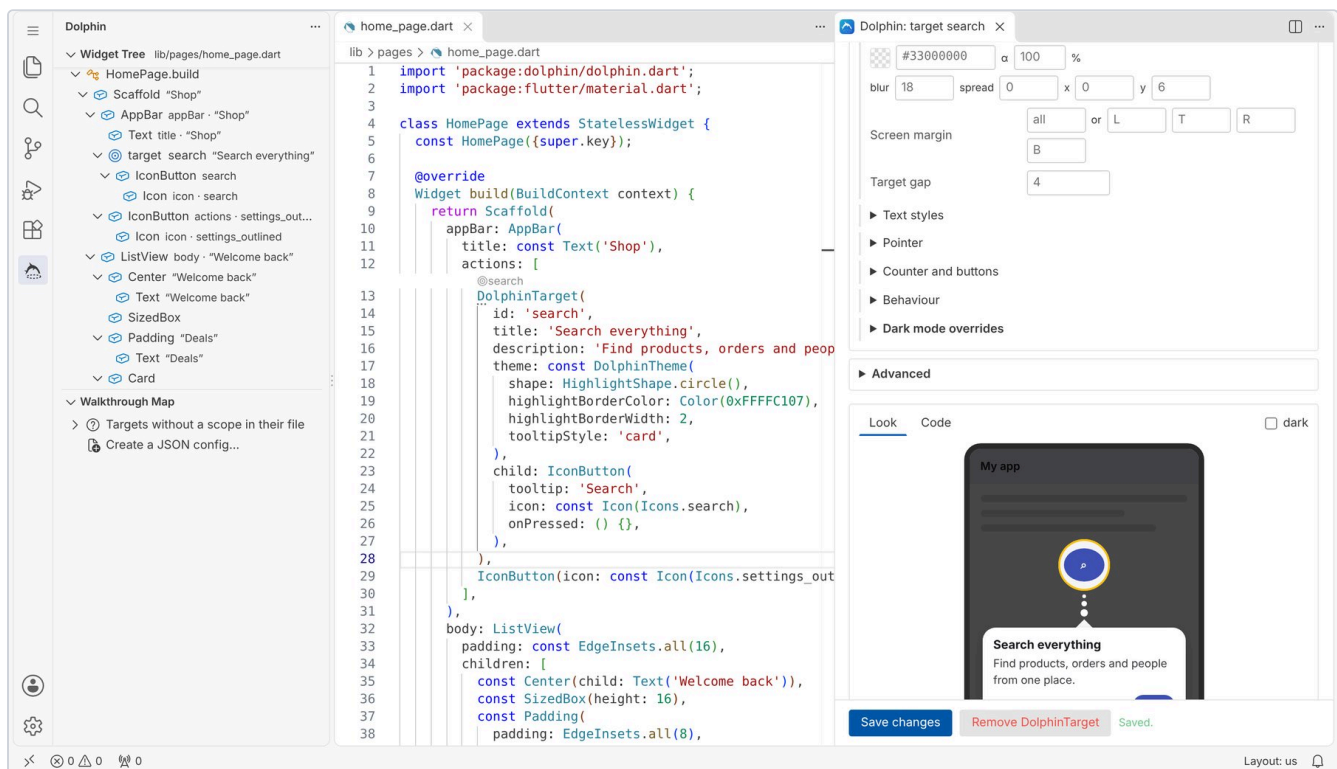


Figure 3.1 After wrapping: the tree shows the new target, the code is written in place, and the configure panel stays open for changes.

The code written for that step:

```
DolphinTarget(
  id: 'search',
  title: 'Search everything',
  description: 'Find products, orders and people from one place.',
  theme: const DolphinTheme(shape: HighlightShape.circle()),
  child: IconButton(
    tooltip: 'Search',
    icon: const Icon(Icons.search),
    onPressed: () {},
  ),
),
```

4 The widget tree

The Widget Tree view reads the Dart file in the active editor and lists its widgets the way they nest in the code, one group per `build` method.

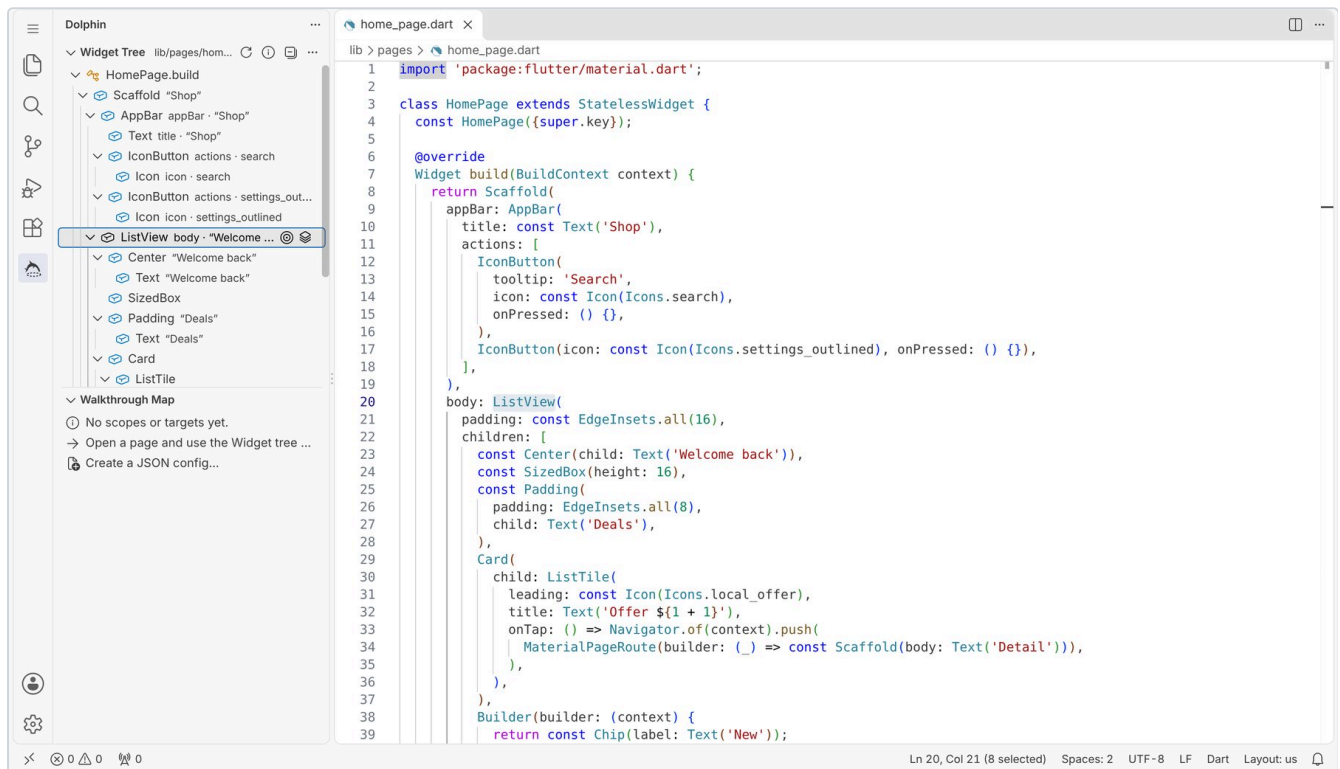


Figure 4.1 The Widget Tree beside `home_page.dart`. The selected row follows the cursor in the editor.

Reading a row

PART	MEANING
Widget name	The class, such as <code>IconButton</code> or <code>ListView</code> .
Slot	Where it sits in its parent: <code>appBar</code> , <code>body</code> , <code>actions</code> , <code>title</code> .
Label	A hint taken from its text, tooltip or icon, such as <code>"Shop"</code> or <code>search</code> .
scope home, target search	Widgets already wrapped by dolphin, with their ids.

Actions

- **Click** a row to put the cursor on that widget in the code.
- **Double-click** a row to open the wrap menu, or the configure form when it is already a dolphin widget.
- **Right-click** a row for **Wrap with DolphinTarget...**, **Wrap with DolphinScope...**, **Configure...**, **Remove wrapper (keep child)** and **Show in code**.
- The buttons at the top of the view: **Refresh**, **Open JSON config** and **About Dolphin**.

Tip. In the editor, put the cursor on any widget and press `Ctrl+Alt+T` (`Cmd+Alt+T` on Mac) to wrap it with `DolphinTarget` without going through the tree.

What is hidden. Values that are not widgets on screen, such as `EdgeInsets`, `TextStyle`, `Duration`, keys and callbacks, are left out, so the tree stays readable on large pages.

5 Wrapping a step

A step is a `DolphinTarget` around one widget. Double-click the widget and choose **DolphinTarget**.

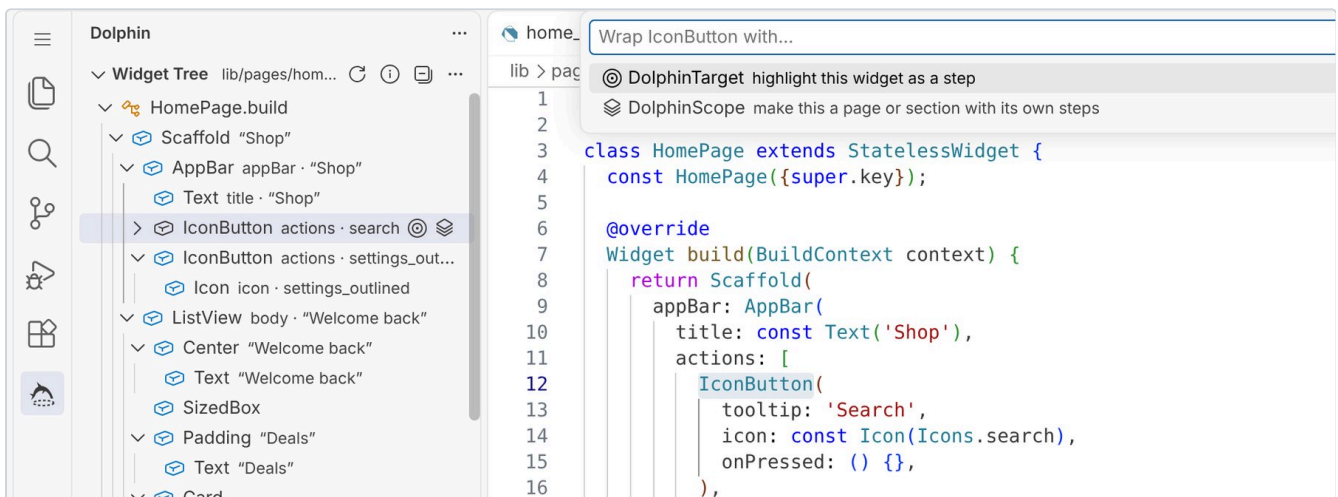


Figure 5.1 Double-click on `IconButton` opens the wrap menu at the top of the window.

Figure 5.2 The form for a new target. The id is suggested from the widget.

Figure 5.3 The Look group: highlight, border, glow and tooltip.

Filling in the form

- 1 **Choose where content goes.** In code writes it on the widget. In JSON config writes it to the config file and keeps only the id on the widget (section 9).
- 2 **Check the id.** It must be unique in its scope. Progress is saved by this id, so keep it stable once released.
- 3 **Write the title and description.** This is what the user reads.
- 4 **Open other groups as needed.** Groups are closed by default. A group shows how many of its fields are set, such as Basics 1 set.
- 5 **Click Wrap widget.** For an existing target the button reads **Save changes**, and **Remove DolphinTarget** takes the wrapper away.

Empty means default. Every field shows the dolphin default as grey placeholder text. Leave a field empty to use it; nothing is written for empty fields.

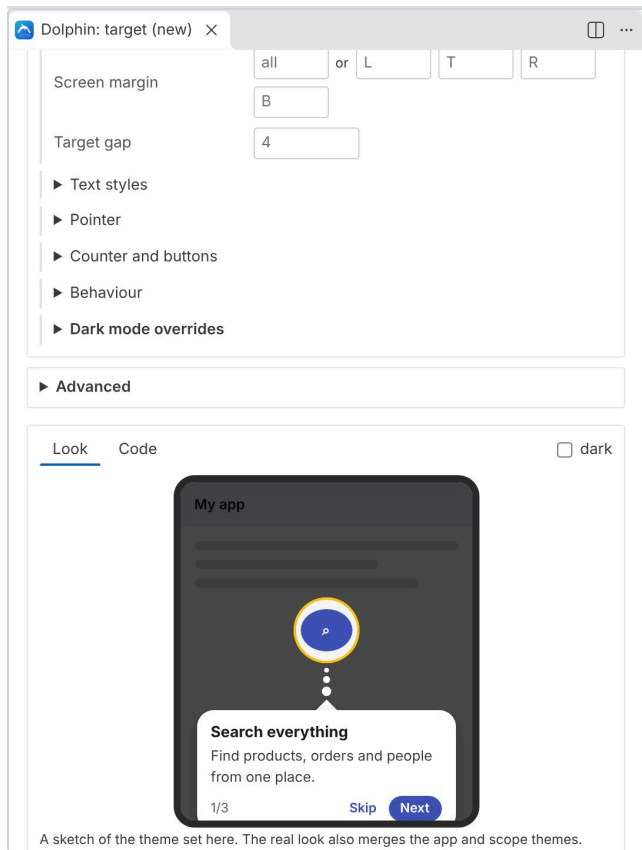


Figure 5.4 The live preview. Tick **dark** to see the dark theme.

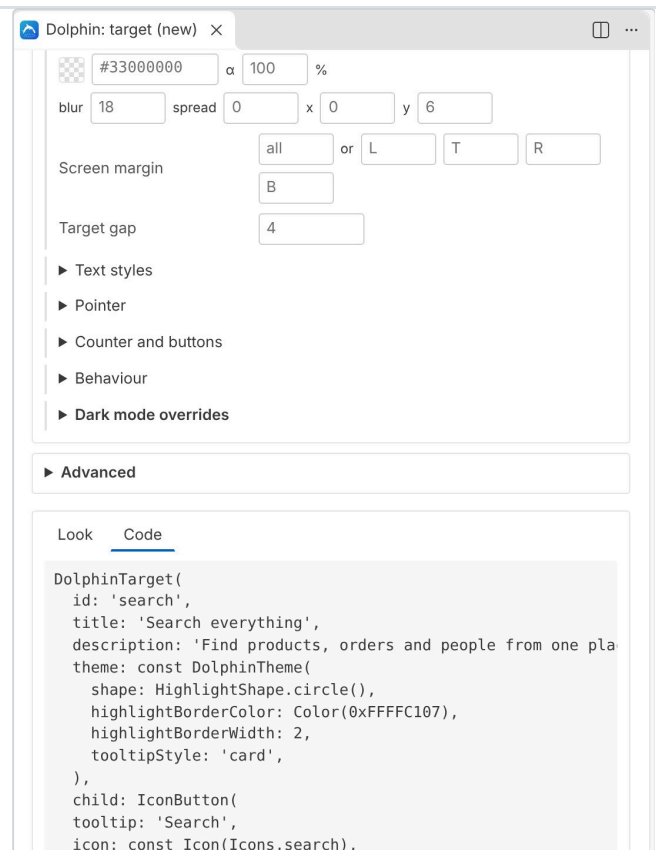


Figure 5.5 The Code tab shows exactly what will be written.

Editing an existing step

Double-click a `DolphinTarget` in the tree, click the **target** link above it in the code, or run **Dolphin: Configure...** with the cursor on it. The form opens with the current values. Anything the form does not know about, such as a variable or a function call, is kept exactly as you wrote it.

What the extension takes care of

- Adds `import 'package:dolphin/dolphin.dart';` where it is missing.
- Writes `const` where it can, and removes a `const` above the widget only when the new code cannot be constant.
- Formats the changed lines with the Dart formatter when the Dart extension is installed.
- Keeps one undo step: `Ctrl+Z` in the editor takes the whole change back.

6 Every option in the form

The form covers every parameter of `DolphinTarget`, `DolphinStep` and `DolphinTheme`. Fields marked *code* exist only in Dart and stay on the widget even in JSON mode.

Basics and content

FIELD	DEFAULT	WHAT IT DOES
Id	required	Unique inside its scope. Progress is stored by this id.
Scope <i>code</i>	nearest scope	Set it when the target sits outside its <code>DolphinScope</code> , for example in a shared app bar.

Title, Description	empty	The text of the tooltip.
Order	code order	Position in the scope when no sequence is set.
Enabled <i>code</i>	true	False removes the step from the walkthrough. The widget still shows.
Image asset, Image URL	empty	A picture shown in the tooltip.
Counter text	automatic	Replaces the automatic “1/3” for this step.
Semantics label	title	What screen readers announce.
Content version	empty	Raise it when the text changes. Users who saw an older version see the step again.
Extra data	empty	Free-form JSON for custom tooltip builders.
Tooltip builder <i>code</i>	empty	<code>(context, tooltip) => Widget</code> . Wins over every preset.

Rules and pointer position

FIELD	DEFAULT	WHAT IT DOES
Condition name	empty	A name registered with <code>registerCondition</code> . The step is skipped while it returns false.
Condition function <i>code</i>	empty	<code>() => bool</code> , written inline.
Auto advance after	off	Moves on by itself after this many milliseconds.
Wait for target	600 ms	How long to wait for a widget that appears late.
Scroll into view	true	Scrolls the widget into view before showing the step.
If the widget is missing	skip	<code>skip</code> , <code>skipAndMarkSeen</code> or <code>showCentered</code> .
Fallback rectangle	empty	Where the widget is assumed to be, in fractions of the screen, when it cannot be measured.
Pointer mode	auto	<code>manual</code> uses Pointer X or Pointer offset below.
Pointer X, Pointer offset	empty	Position of the pointer along the tooltip edge, 0 to 1, or in pixels.

Look

The same Look editor appears for a step, a scope and the app. A step overrides its scope, and a scope overrides the app.

GROUP	FIELDS
Barrier	Barrier colour (default <code>#B3000000</code>), opacity, blur.
Highlight	Shape (rectangle, rounded, circle, oval, stadium, diamond or beveled, optionally covering the whole widget), padding (6), offset, border colour, width, dash and gradient, glow colour and blur, pulse colour.

Tooltip	Style (card, bubble, minimal, banner, glass, compact or a registered builder), position (auto, above, below, left, right, center), background, text colour, primary colour, corner radius (14), padding, max width (340), shadow, screen margin (12), target gap (4).
Text styles	Title, description, counter and buttons: colour, size, weight and line height.
Pointer	Kind (none, arrow, dots, arrow and dots, line, custom), arrow width and height, dot sizes, spacing, shape and colours, line width and length, edge inset, custom painter.
Counter and buttons	Counter format (<code>{current}/{total}</code>), counter builder, show counter, Skip, Back and Next, and their labels.
Behaviour	Animation (transition, fade, move, curve, pulse), tap on target (advance, passThrough, passThroughAndAdvance, block), tap on barrier (none, next, dismiss, skipScope), back button (dismiss, previous, block, passThrough), respect safe area.
Dark mode overrides	Only what differs in dark mode. Everything else comes from the theme above.
Advanced	Key and controller, for special cases.

Colours. Each colour field has a swatch, a hex box and a separate opacity box in percent, so `#FFC107` at 60% is written as `Color(0x99FFC107)`.

7 Pages and sections

A `DolphinScope` marks a page or a section with its own steps. Progress, skip and reset work per scope.

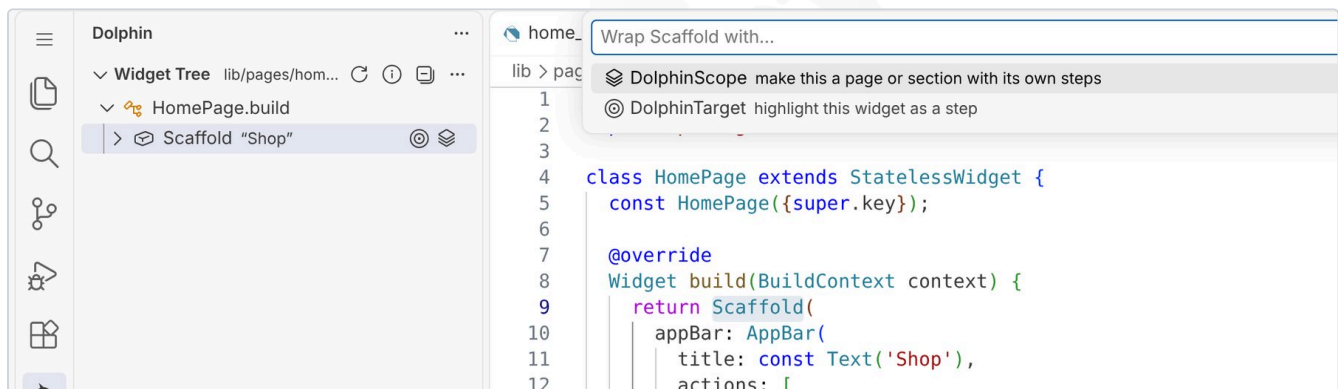


Figure 7.1 Double-click the page widget, usually `Scaffold`, and choose **DolphinScope**.

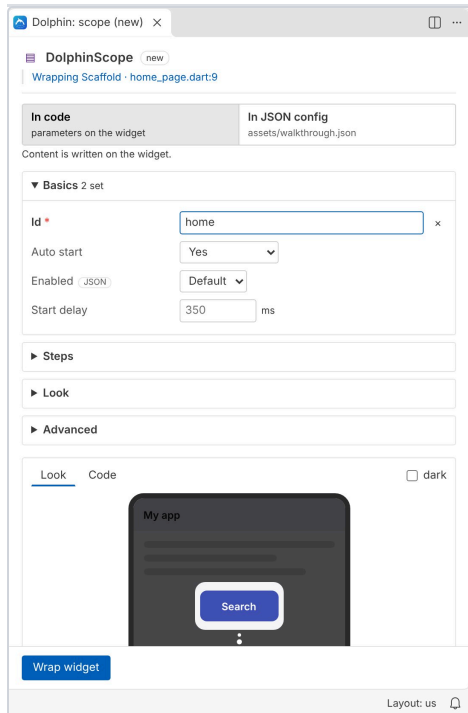


Figure 7.2 The scope form. The id comes from the class name.

FIELD	WHAT IT DOES
Id	One id per page or section.
Auto start	Start when the page becomes visible with unseen steps. Default yes.
Enabled <i>JSON</i>	Only in the JSON config. In code, use <code>controller.setScopeEnabled</code> .
Start delay	Lets page animations settle first. Default 350 ms.
Step order	The explicit order of step ids. Steps not listed follow in their natural order.
Never show	Step ids to leave out of this scope.
Inline steps <i>code</i>	A list of <code>DolphinStep</code> written on the scope.
Look	The theme for every step in the scope.

Above each scope in the code, a link such as **scope home · 2 steps** opens this form.

Targets outside their scope

A target normally belongs to the nearest `DolphinScope` above it. When a widget is built elsewhere, for example in a shared app bar or a bottom sheet, set the **Scope** field of the target. The extension warns when a target has no scope above it in its file, and that is fine when the page puts one around it.

8 The app root and controller

One `Dolphin` widget above `MaterialApp`, `CupertinoApp` or `WidgetsApp` holds the controller for the whole app.

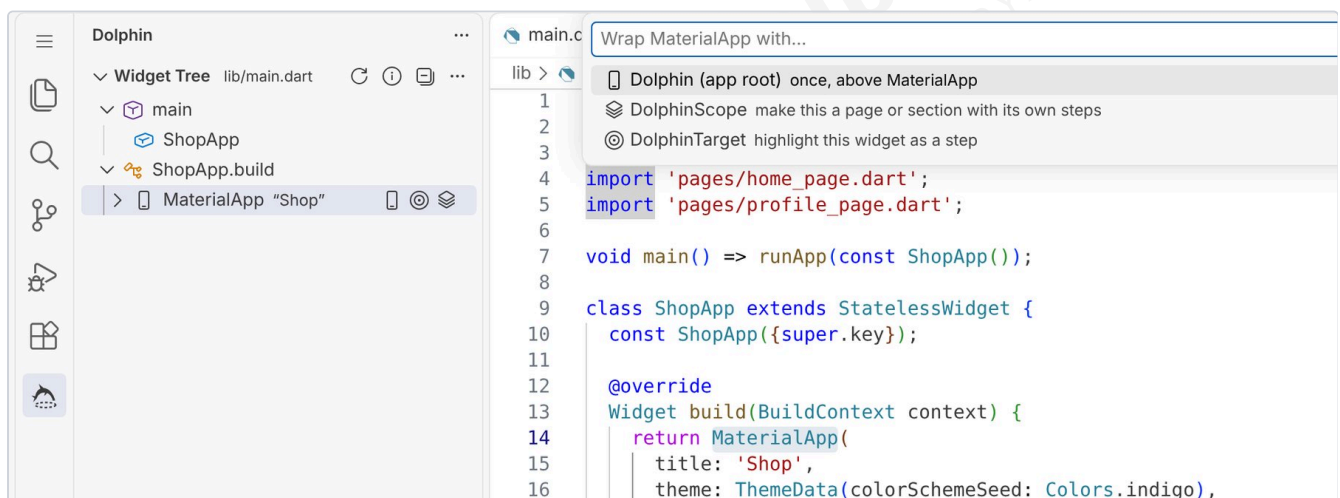


Figure 8.1 On `MaterialApp` the menu also offers **Dolphin (app root)**.

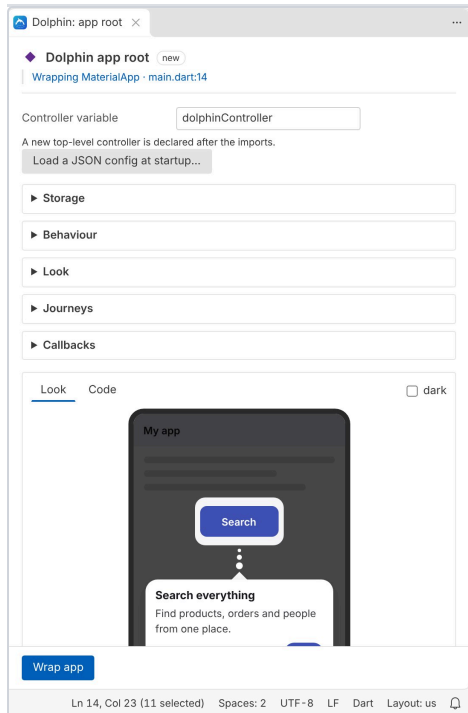
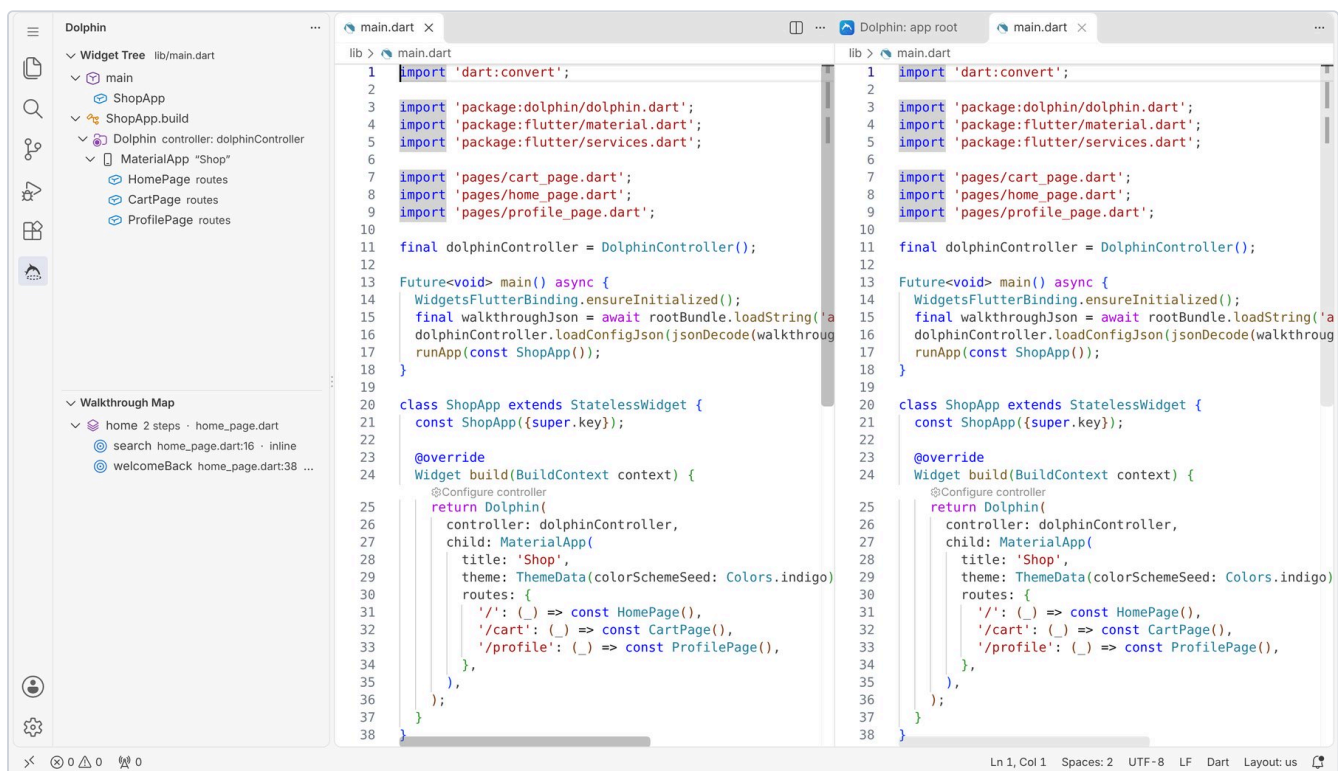


Figure 8.2 The controller form.

GROUP	FIELDS
Storage	Store (memory, SharedPreferences or SQLite) and config.
Behaviour	Auto start, enabled, start delay, wait for target, scroll into view, missing widget, which data wins when it is in both places, and how the counter counts.
Look	Theme mode (app, system, light, dark), app theme and app dark theme.
Journeys	Navigator key, journey navigator, and what Skip does inside a journey.
Callbacks	<code>onEvent</code> for analytics, <code>onProgressChanged</code> to sync progress with your server, <code>onError</code> , debug logging.

The controller is declared as a top-level variable, `dolphinController` by default. Change its name in **Controller variable**.

Figure 8.3 `main.dart` before and after: the import, the controller, the JSON loading in `main()` and the `Dolphin` wrapper.

No app root? Without a `Dolphin` widget the walkthrough never starts. The extension shows a warning on every scope until one is added.

9 Code or JSON config

Every step can keep its content on the widget or in a JSON config. The JSON lets you change text and style without touching the code, or from your server.

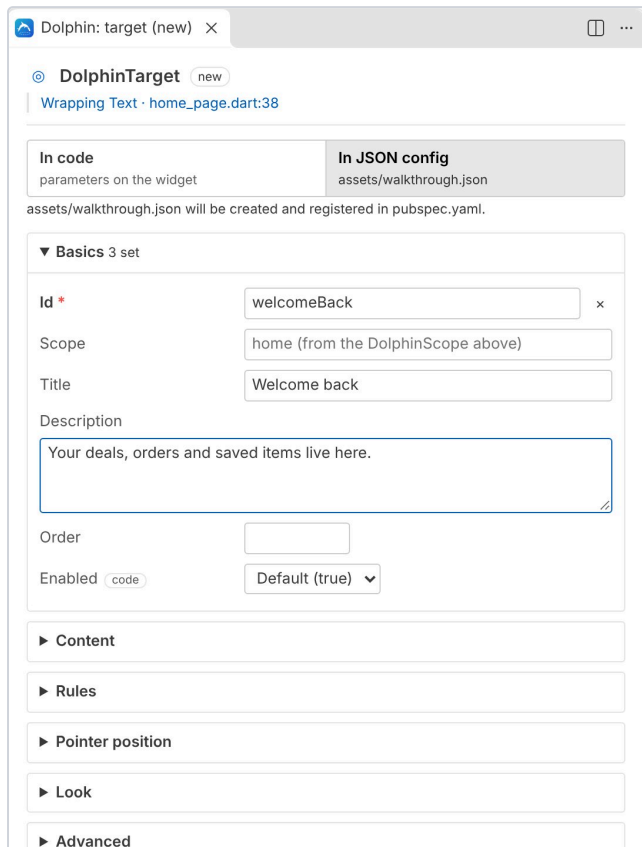


Figure 9.1 In JSON config selected. The form shows where the file will be.

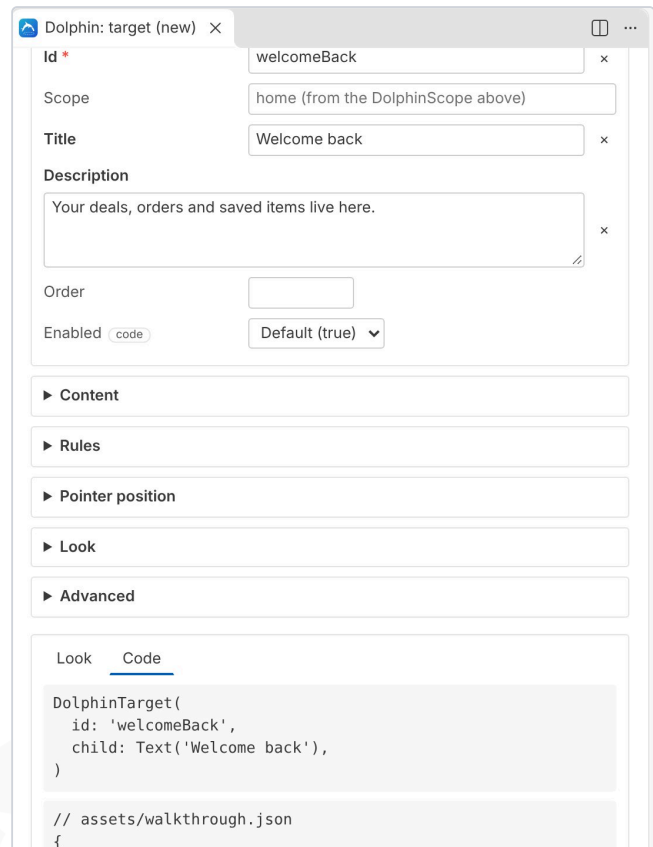


Figure 9.2 The Code tab shows both the Dart and the JSON that will be written.

Creating the config

The first time you store a step in JSON, the extension asks to create `assets/walkthrough.json`. It registers the file under `flutter: assets:` in `pubspec.yaml` and then offers to load it at startup.

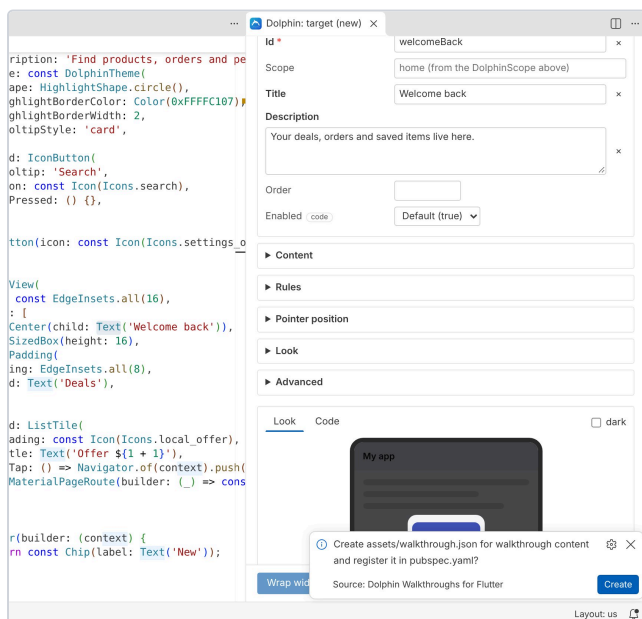


Figure 9.3 The offer to create the config.

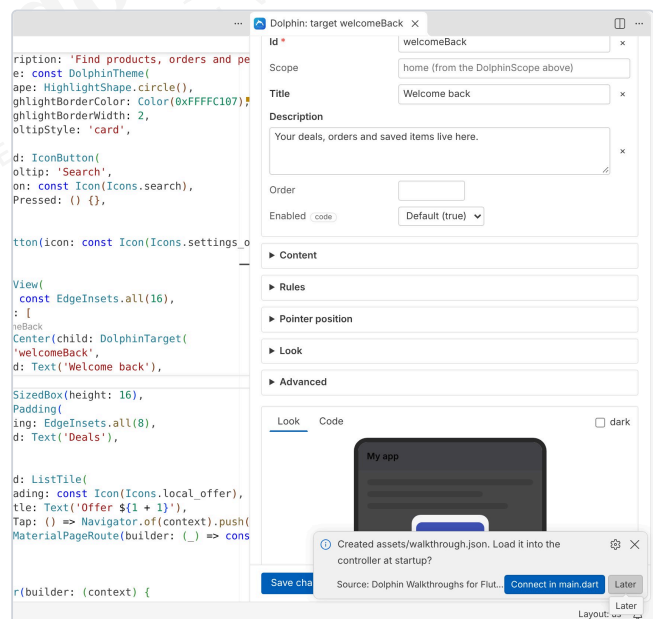


Figure 9.4 Connect in main.dart loads it when the app starts.

The same step in both places:

```
// Dart
DolphinTarget(
  id: 'welcomeBack',
  child: Text('Welcome back'),
)

// assets/walkthrough.json
{ "scopes": { "home": { "steps": {
  "welcomeBack": {
    "title": "Welcome back",
    "description": "Your deals live here."
  } } } } }
```

How the config stays in sync

- **Found automatically.** Any JSON file in the project with a `scopes` key, preferring files under `assets/`. Set `dolphin.configFile` to choose one.
- **Updated in place.** Saving a step changes only its own entry. Your comments, order and layout are kept.
- **Existing projects.** If dolphin and a config are already in the project, the extension reads them as they are. Steps already in the JSON open in JSON mode with their values.
- **Renames.** Changing an id in the form renames the widget, the JSON entry, the step order and every journey that uses it.
- **Checked.** Files named `walkthrough*.json`, `*dolphin*.json`, `onboarding*.json` or `showcase*.json` get completion and validation for every key dolphin reads.

Where progress is saved. The JSON holds content and settings only. What each user has seen is saved on their device by dolphin, and can be synced with `onProgressChanged`.

10 The walkthrough map

The Walkthrough Map view lists every scope in the project, its steps in the order users see them, and every journey. It reads the code and the JSON config together.

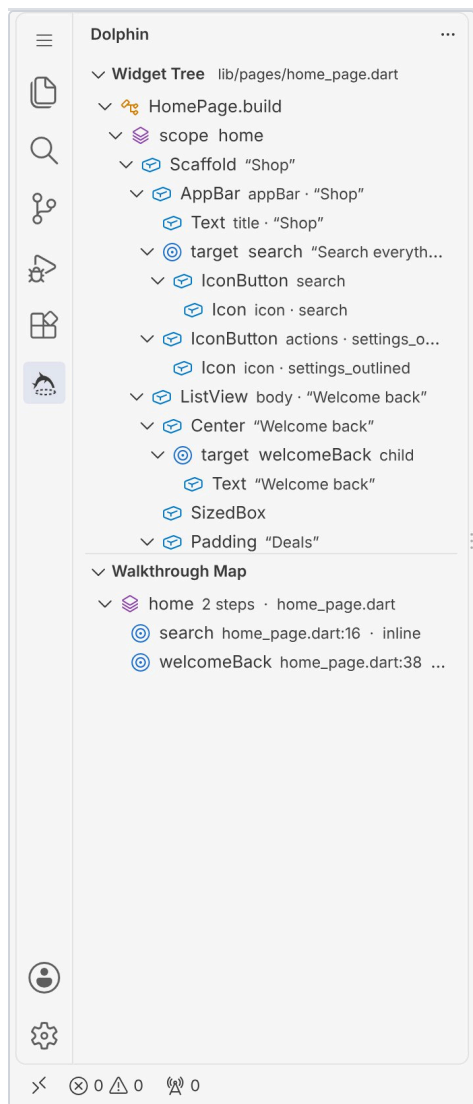


Figure 10.1 The Walkthrough Map under the Widget Tree.

ROW	MEANING
Scope	Page id, number of steps and the file.
Step	Id, file and line, and where its content lives: inline or JSON .
Planned	In the JSON config only, not attached to a widget yet.
Warning	Something to check: a duplicate id, no content, an unknown style or condition.
Journeys	Each journey with its pages in order.

- Click a step to open its widget in the code.
- Right-click for **Configure...** and **Open in config**.
- The title bar buttons open the **graph**, **export a PDF**, **add a journey** and **refresh**.

Steps that are not under any scope are listed under **Targets without a scope**, so nothing gets lost.

11 The walkthrough graph

The graph shows each scope as a lane and its steps as cards, in the order users see them. Journeys are drawn as arrows from page to page. It is also where you design walkthroughs before the screens exist.

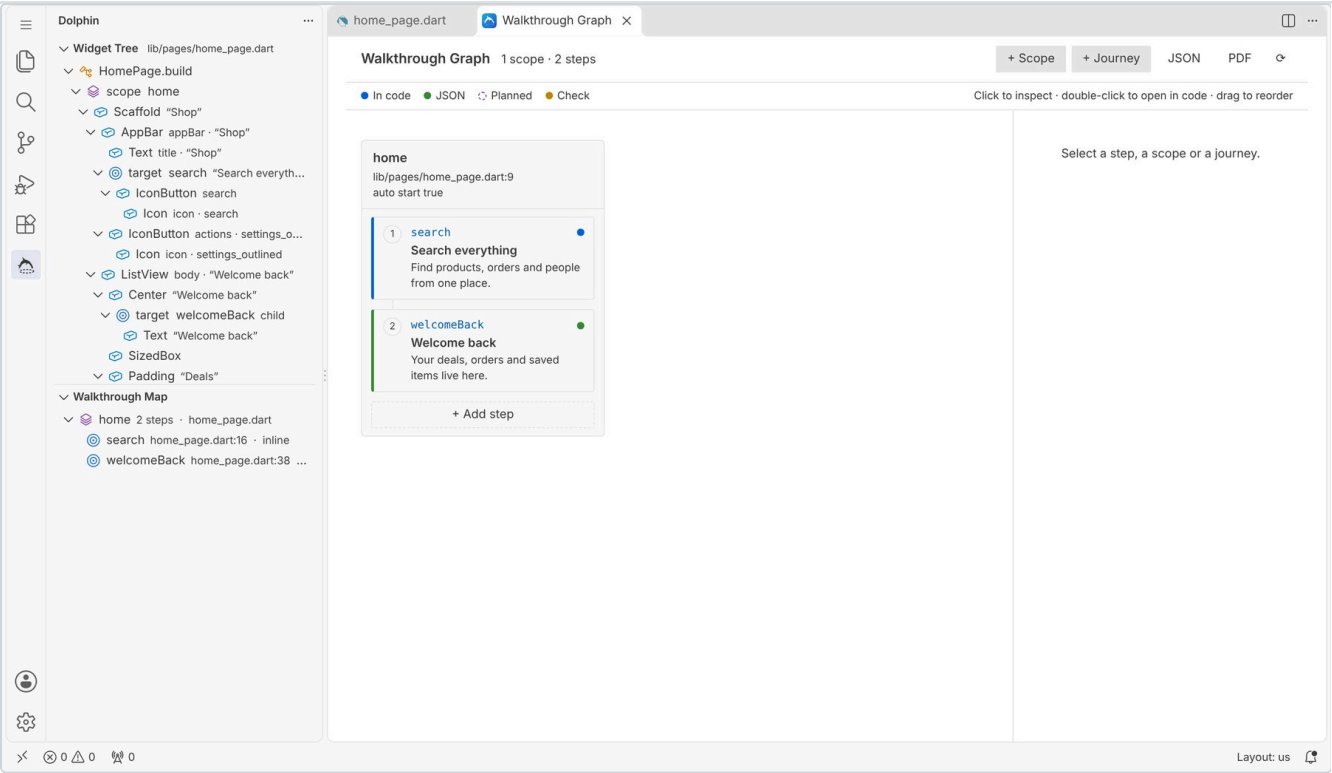


Figure 11.1 Open it with the graph icon on the Walkthrough Map, or **Dolphin: Open walkthrough graph**.

Working with cards

ACTION	RESULT
Click a card	Shows it in the inspector on the right, where you can edit its title and description.
Double-click a card	Opens the wrapped widget in the code. For a planned step it opens its JSON entry.
Drag a card	Changes the order. The new order is saved as <code>sequence</code> in the JSON config.
Double-click a lane title	Opens the page's <code>DoLphinScope</code> .
JSON, PDF buttons	Open the config, or export the map as a PDF (section 13).

■ content in code ■ content in JSON □ planned ■ something to check

Design first

Plan a walkthrough with your designer before the pages are built. Everything is written to the JSON config and shows as **planned**.

- 1 + **Scope** adds a page, for example `cart`.
- 2 + **Add step** in a lane opens the **New step** form: id, title and description.
- 3 + **Journey** links pages in order (section 12).

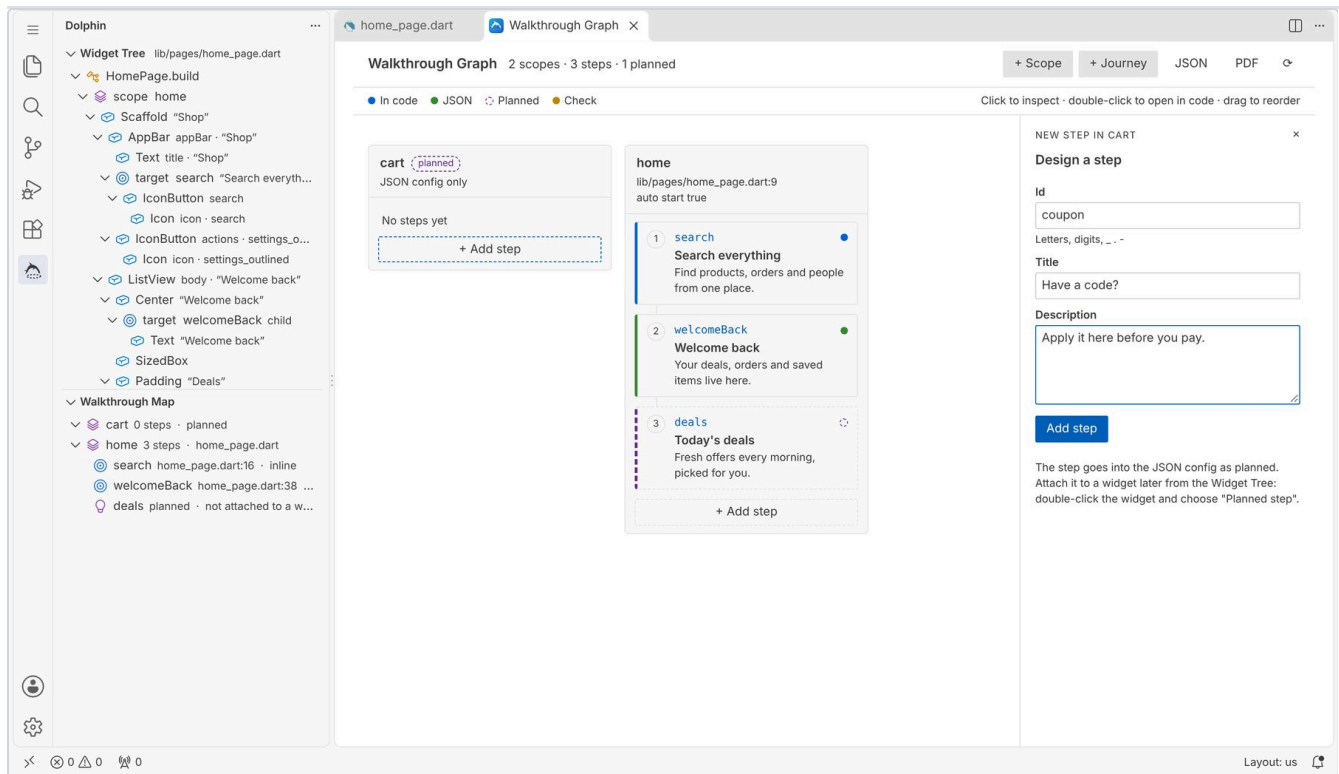


Figure 11.2 Designing a step on a page that is not built yet.

Attaching planned work to the code

When the page is built, connect the plan to real widgets. The content stays in the JSON.

- 1 **Attach the page.** Open it, double-click its page widget in the Widget Tree and choose **Planned scope...**, then pick the scope.
- 2 **Attach each step.** Double-click a widget, choose **Planned step...** and pick from the list. The widget is wrapped with that id.
- 3 **Check the graph.** The card turns from dashed purple to green, and double-click now jumps to the widget.

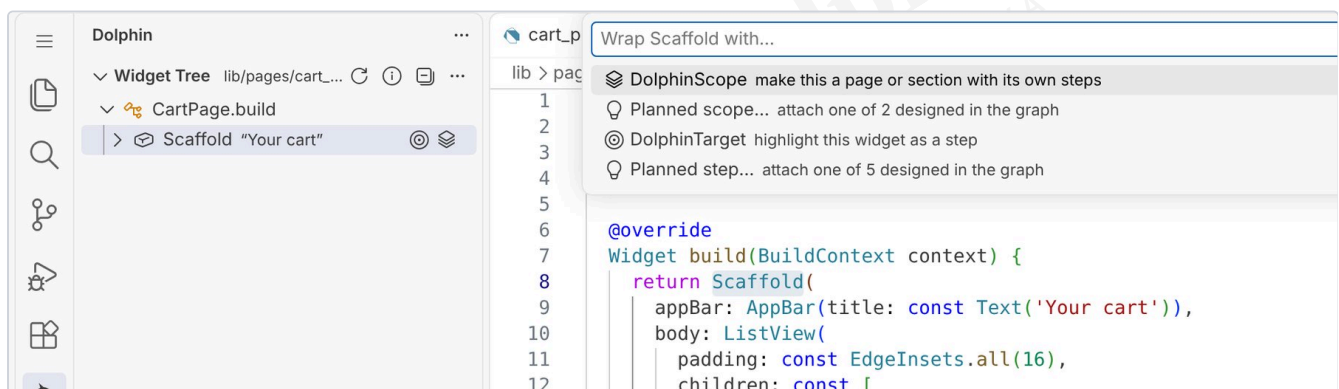


Figure 11.3 Planned scope... and Planned step... in the wrap menu, with how many are waiting.

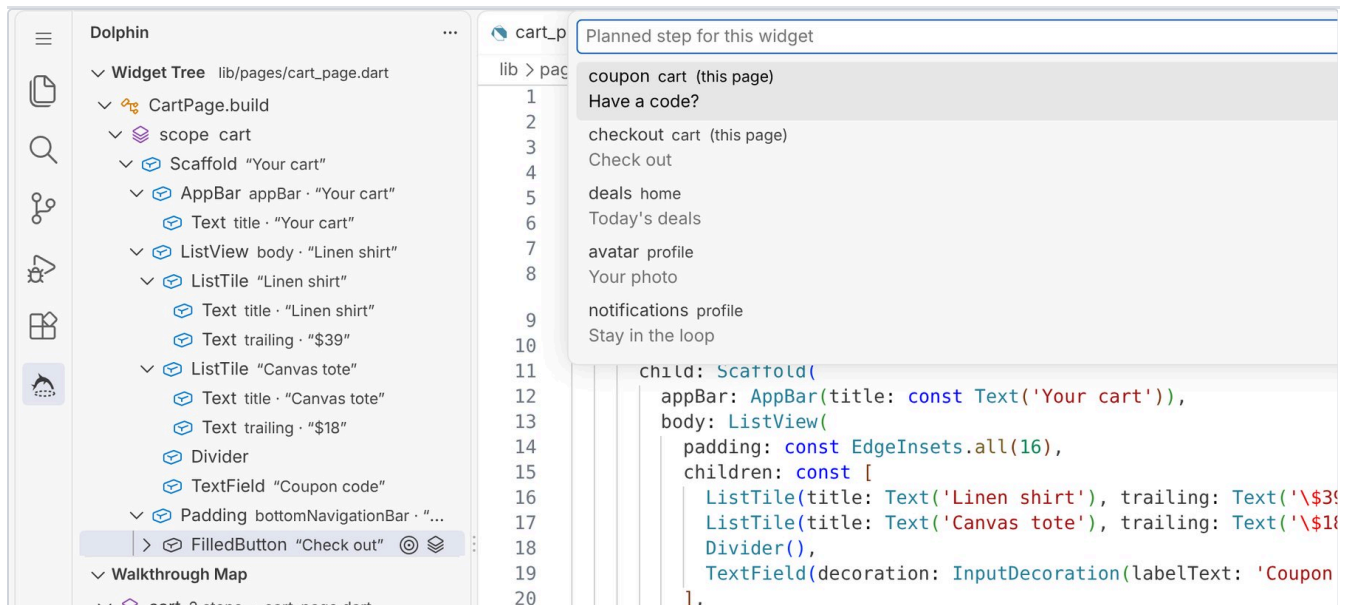


Figure 11.4 The planned steps of every page, with their titles. Steps of the current page come first.

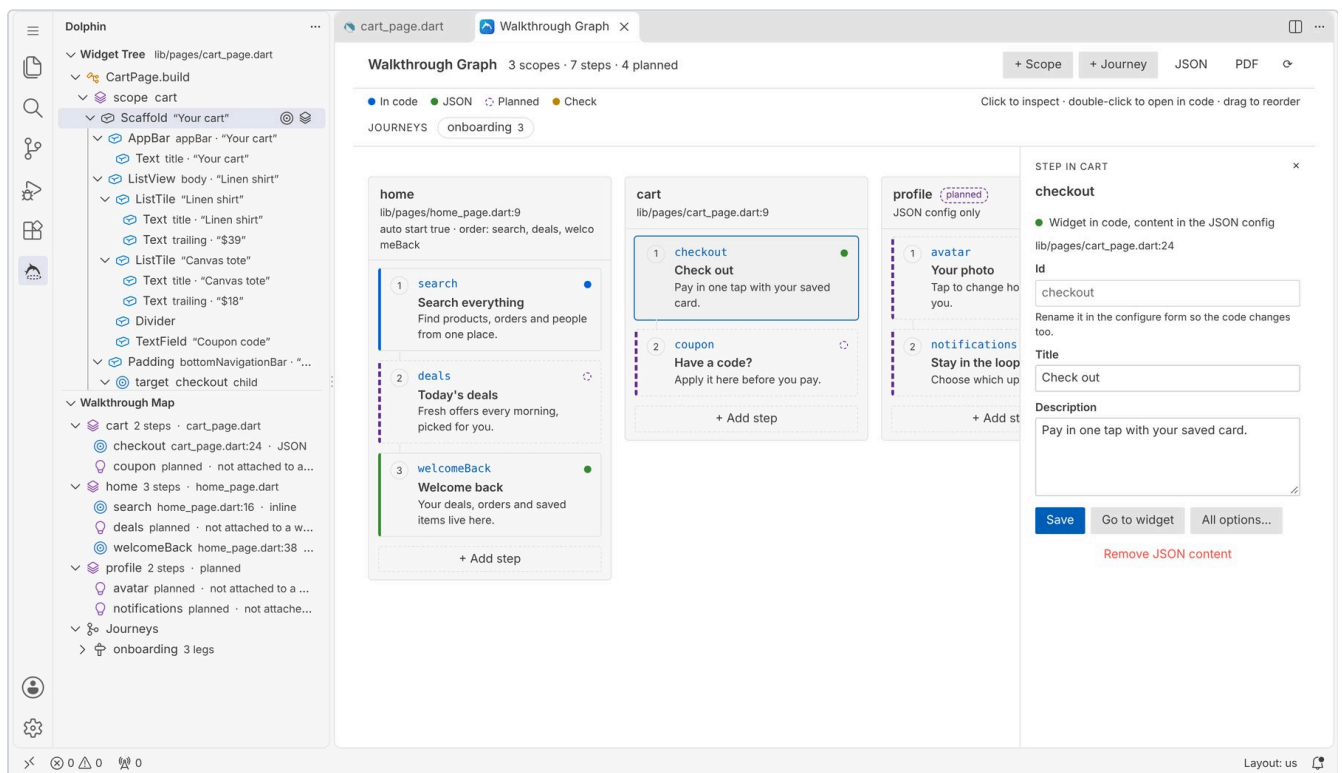


Figure 11.5 After attaching: `checkout` is wrapped in code and its content stays in JSON.

Renaming. A planned step can be renamed in the inspector. A step that already has a widget is renamed in its configure form, so the code changes too.

12 Journeys

A journey walks the user across several pages in a fixed order, for example home, then cart, then profile.

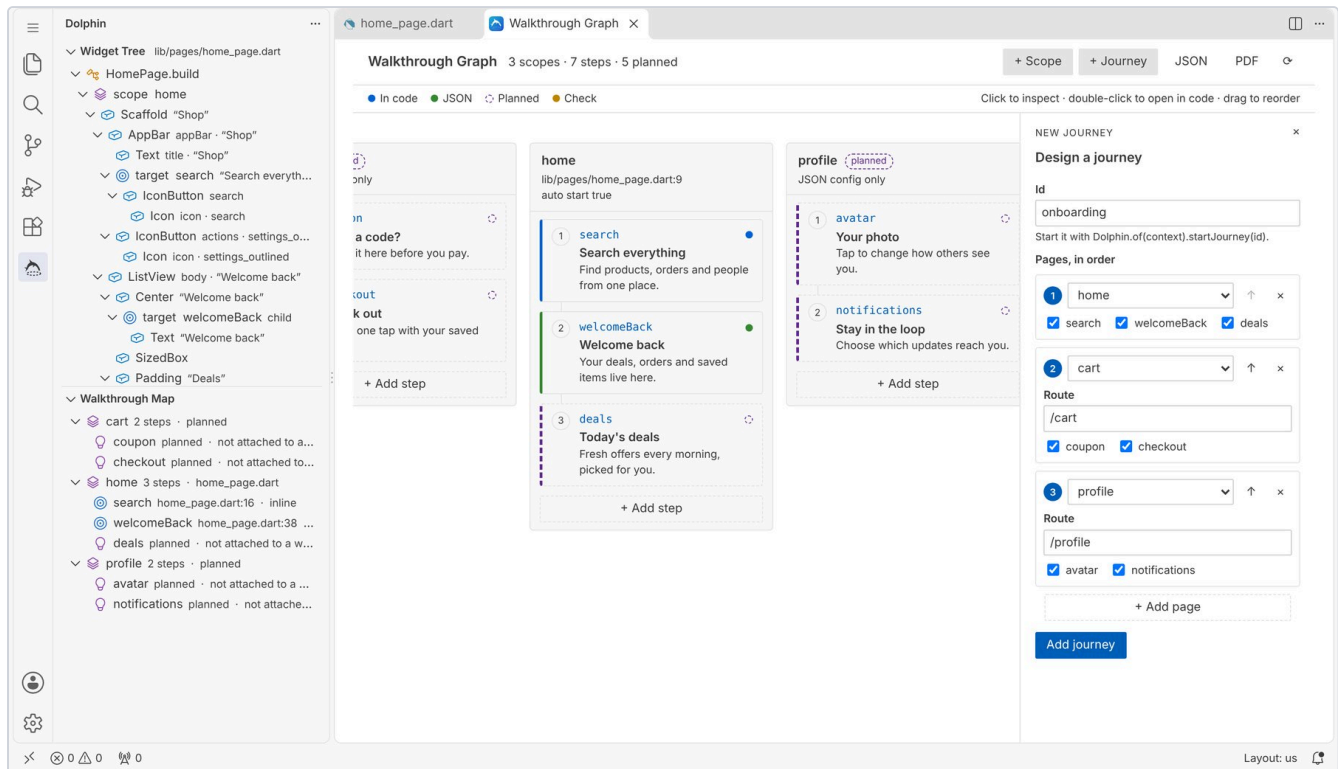


Figure 12.1 The journey editor in the graph. Each page is a leg with its steps and route.

- 1 Click **+ Journey** in the graph, or **Add journey** on the Walkthrough Map.
- 2 Name it, for example `onboarding`.
- 3 Add pages in order. For each page choose the steps to show (none ticked means all pending steps) and the route that opens it, such as `/cart`.
- 4 Click **Add journey**. It is saved under `journeys` in the JSON config.

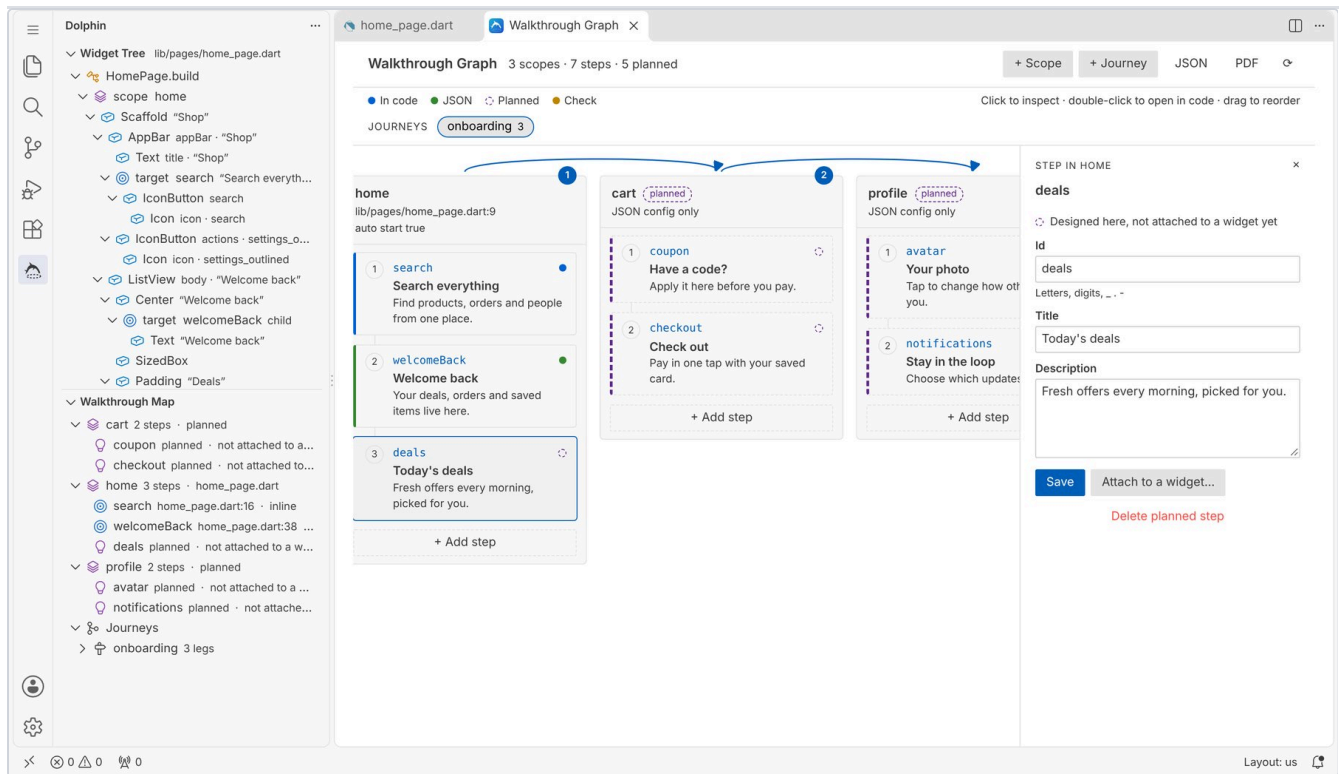


Figure 12.2 Click a journey to see its path. Steps it skips are faded.

Start it from your app:

```
Dolphin.of(context).startJourney('onboarding');
```

Routes. To open each page, dolphin needs your `navigatorKey`, or a **journey navigator** for routers such as `go_router`. Both are in the controller form (section 8).

13 Exporting the map as PDF

Share the whole walkthrough with design, product or QA as a PDF: every scope with its settings, every step with its title, description and location, every journey, and every open problem.

- 1 **Start the export** with the PDF icon on the Walkthrough Map, the **PDF** button in the graph, or **Dolphin: Export walkthrough map as PDF...**
- 2 **Choose where to save it.** The default is `walkthrough-map.pdf` in the project folder.
- 3 **Open it or Show in folder** from the message that follows.

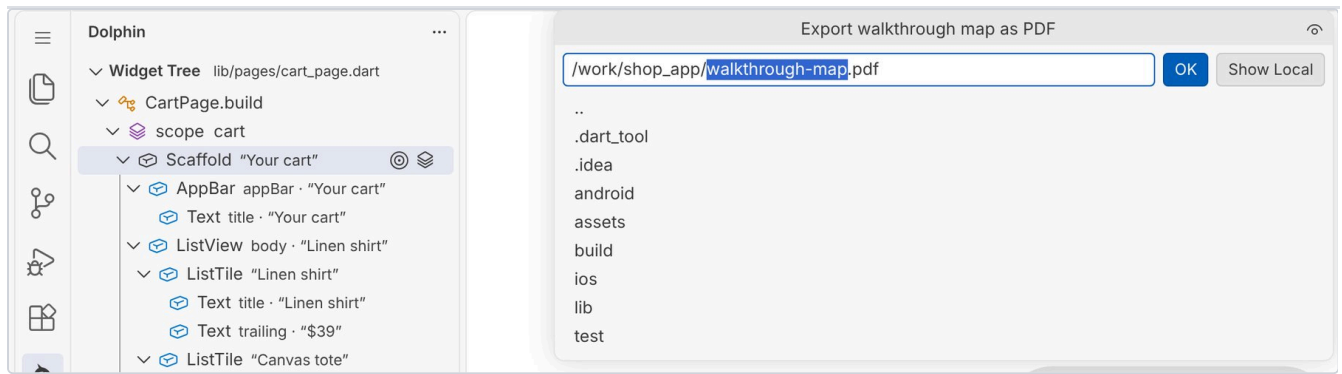


Figure 13.1 The save dialog.



Figure 13.2 The map is saved.

Planned steps are listed as *planned, no widget yet*, so the PDF also works as a to-do list for the developers.

14 Editor helpers

Small helpers in the Dart and JSON editors, all switchable in Settings.

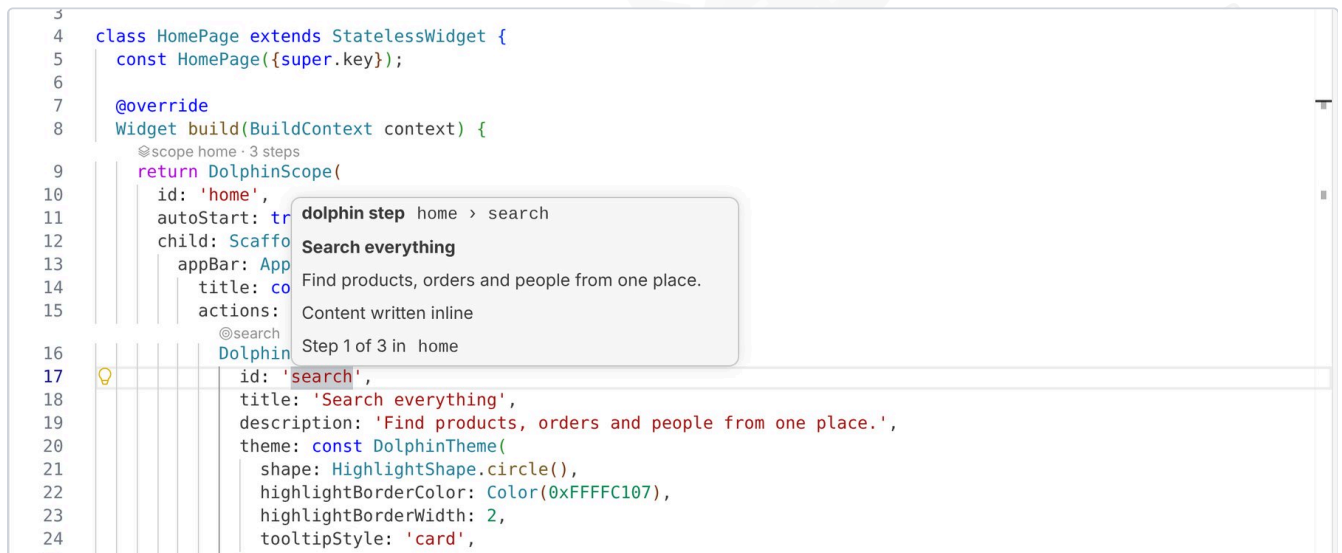


Figure 14.1 Hover on a step id to see its title, description, where its content lives and its position in the scope.

HELPER	WHAT IT DOES
Configure links	Above each dolphin widget: scope home · 3 steps, target search, Configure controller . Click to open the form.

Light bulb menu	On any widget: wrap with DolphinTarget, DolphinScope or Dolphin. On a dolphin widget: configure or remove it.
Hover	On a step id: content, storage and position.
Completion	Scope ids, tooltip styles and condition names inside dolphin widgets.
Checks	Duplicate ids in a scope, steps with no content, tooltip styles and conditions that were never registered, a missing <code>Dolphin</code> root, a target with no child, and JSON entries with no widget.
JSON schema	Completion, descriptions and validation for every key in the config file.

Snippets

PREFIX	INSERTS
<code>dscope</code>	A page or section as a walkthrough scope
<code>dtarget</code>	A widget highlighted as a step
<code>dtargetid</code>	A target whose content lives in the JSON config
<code>dstep</code>	Full step data
<code>dtheme</code>	A theme override
<code>droot</code>	The controller and the Dolphin widget above MaterialApp
<code>dreplay</code> , <code>dreset</code>	Show a page's walkthrough again, or forget its progress
<code>djourney</code>	Start a journey from the JSON config
<code>denabled</code>	Turn every walkthrough on or off
<code>dbuilder</code> , <code>dcondition</code>	A named tooltip design or condition usable from JSON

15 Settings and commands

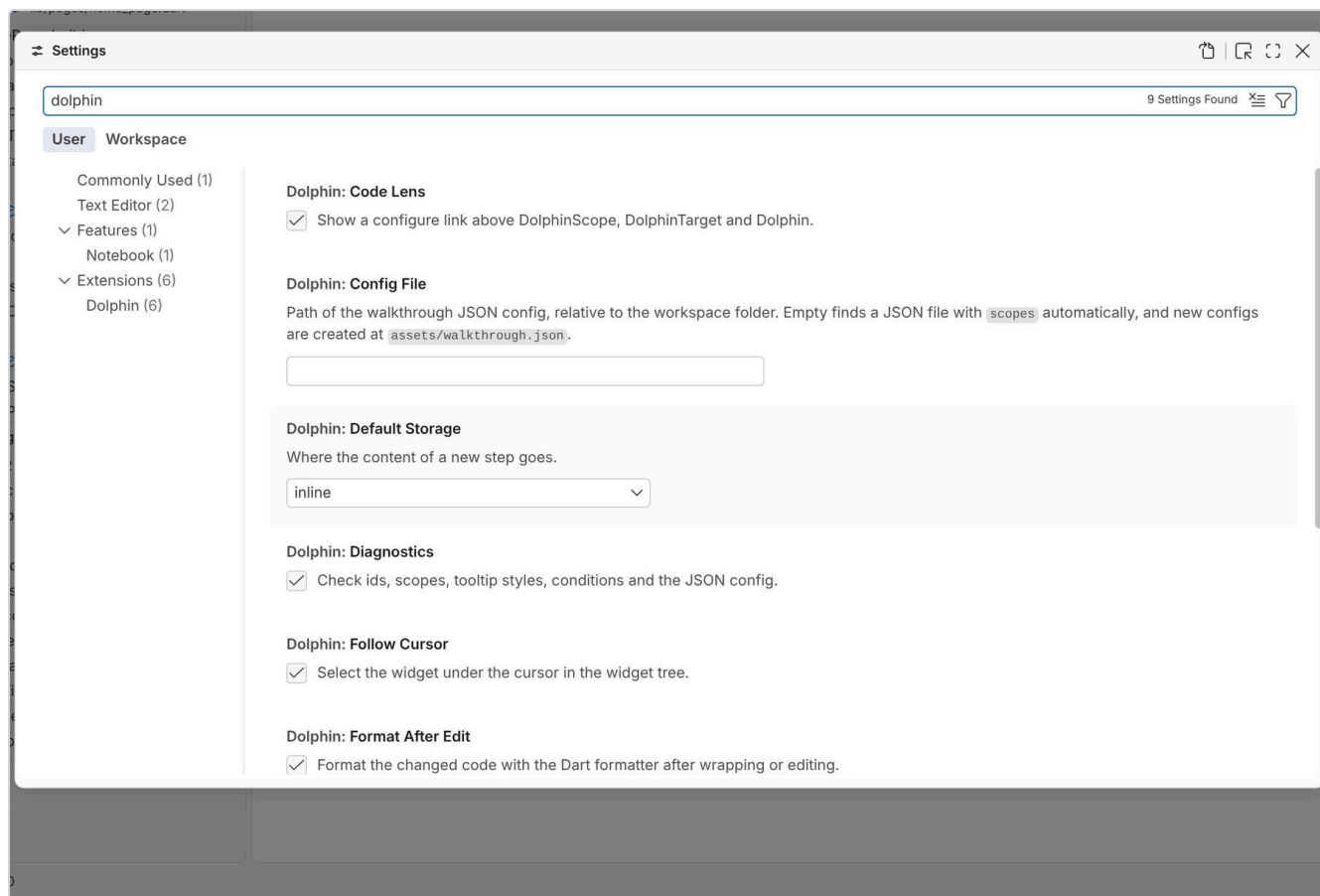


Figure 15.1 Open Settings (`Ctrl+,`) and search for **dolphin**.

SETTING	DEFAULT	DESCRIPTION
<code>dolphin.configFile</code>	empty	Path of the JSON config. Empty finds it automatically.
<code>dolphin.defaultStorage</code>	inline	Where the content of a new step goes: <code>inline</code> or <code>json</code> .
<code>dolphin.formatAfterEdit</code>	true	Format the changed code after wrapping or editing.
<code>dolphin.followCursor</code>	true	Select the widget under the cursor in the tree.
<code>dolphin.codeLens</code>	true	Show the configure links above dolphin widgets.
<code>dolphin.diagnostics</code>	true	Check ids, scopes, styles, conditions and the config.

Commands

DOLPHIN: ...	WHAT IT DOES
Wrap with DolphinTarget...	Wrap the widget under the cursor as a step. <code>Ctrl+Alt+T</code> , <code>Cmd+Alt+T</code> on Mac.
Wrap with DolphinScope...	Wrap the widget under the cursor as a page or section.
Wrap with Dolphin (app root)...	Wrap the app and set up the controller.
Configure...	Open the form for the dolphin widget under the cursor.
Remove wrapper (keep child)	Take a dolphin widget away and keep its child.

Open JSON config / Create JSON config	Open the config, or create one.
Load JSON config at app startup	Change <code>main()</code> to load the config.
Add journey...	Add a cross-page journey to the config.
Open walkthrough graph	Lanes, steps, journeys and design mode.
Export walkthrough map as PDF...	Save the whole map as a PDF.
Add dolphin to pubspec.yaml	Run <code>flutter pub add dolphin</code> .
About Dolphin	Links, numbers and the version your project uses.
Refresh	Read the project again.

16 Dark theme and About Dolphin

Every view and panel follows your VS Code colour theme: light, dark and high contrast.

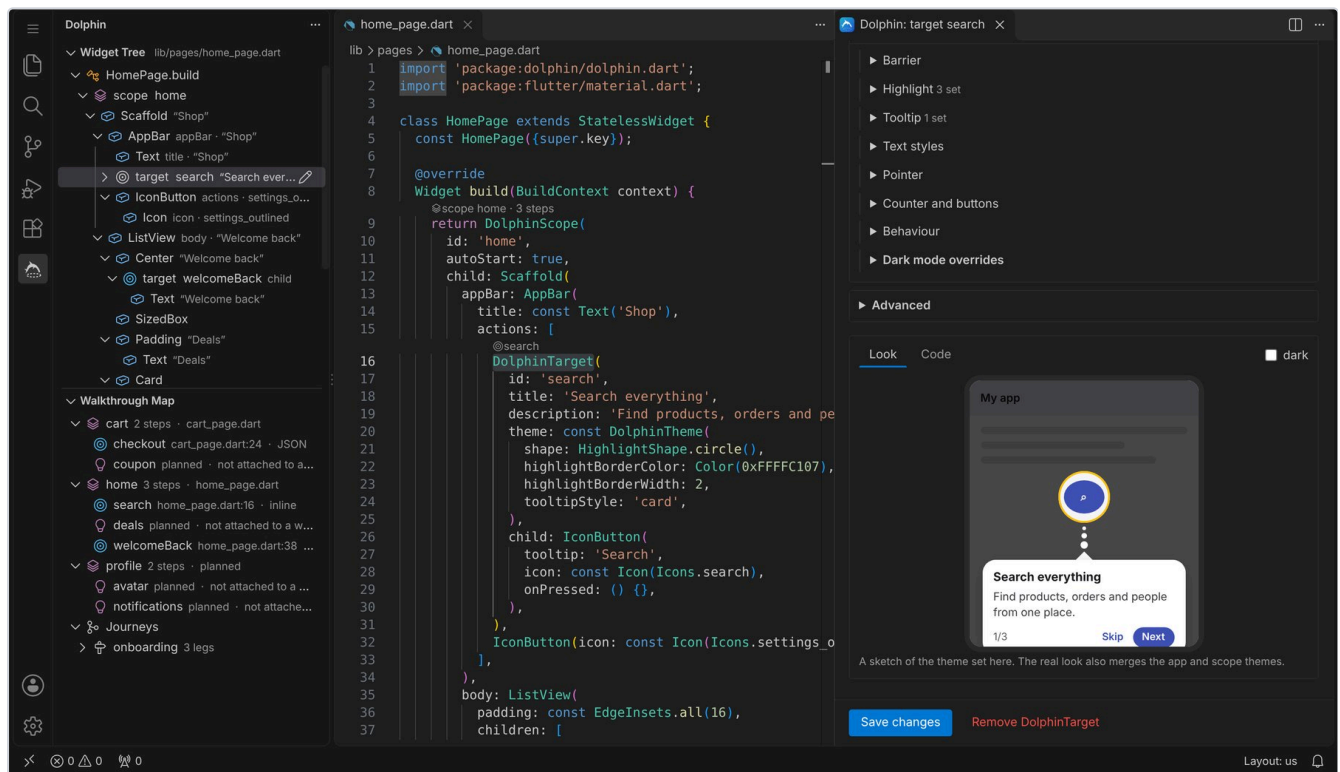


Figure 16.1 The configure panel in a dark theme.

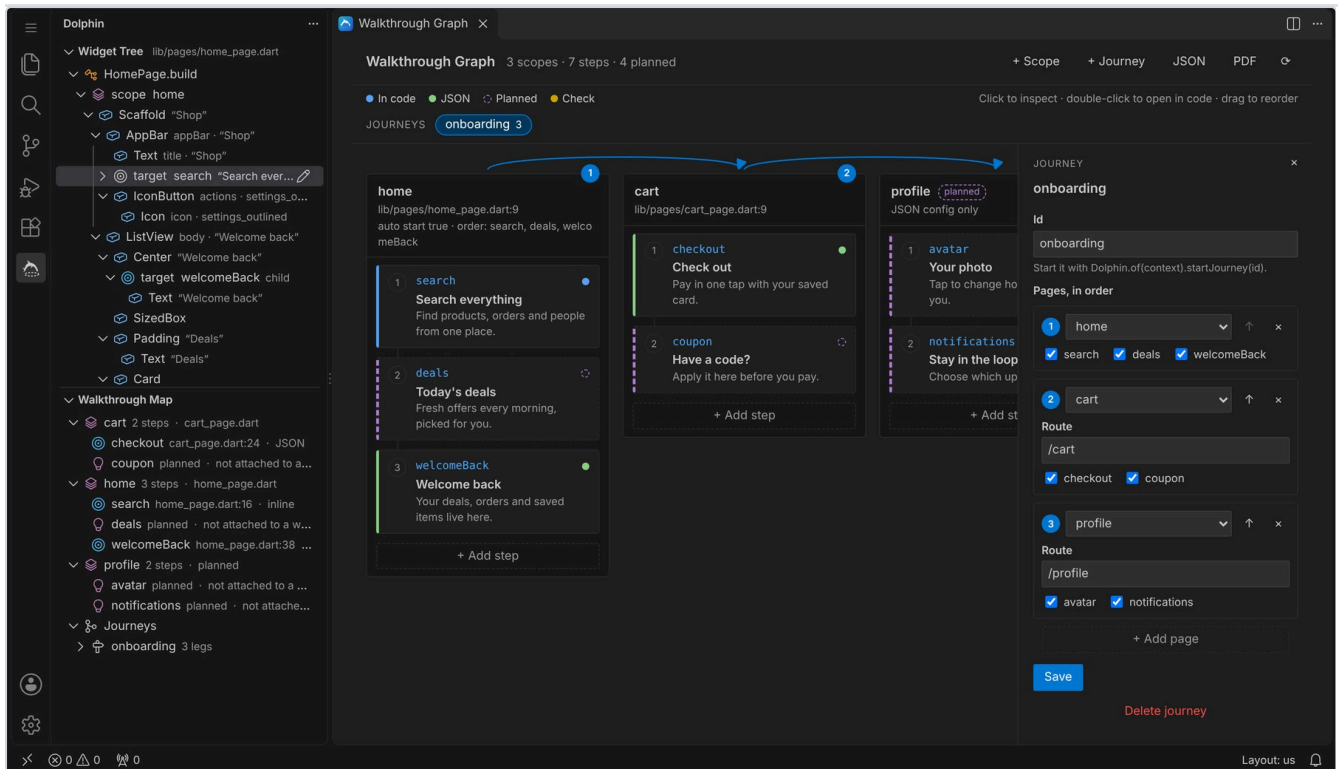


Figure 16.2 The walkthrough graph in a dark theme.

About Dolphin

Open it with the **i** button on the Widget Tree, or **Dolphin: About Dolphin**. It shows the package and extension versions, the dolphin version your project uses from `pubspec.lock`, and live numbers from `pub.dev`: downloads in the last 30 days, likes and pub points. Once the extension is on the Marketplace, its installs and rating appear too.

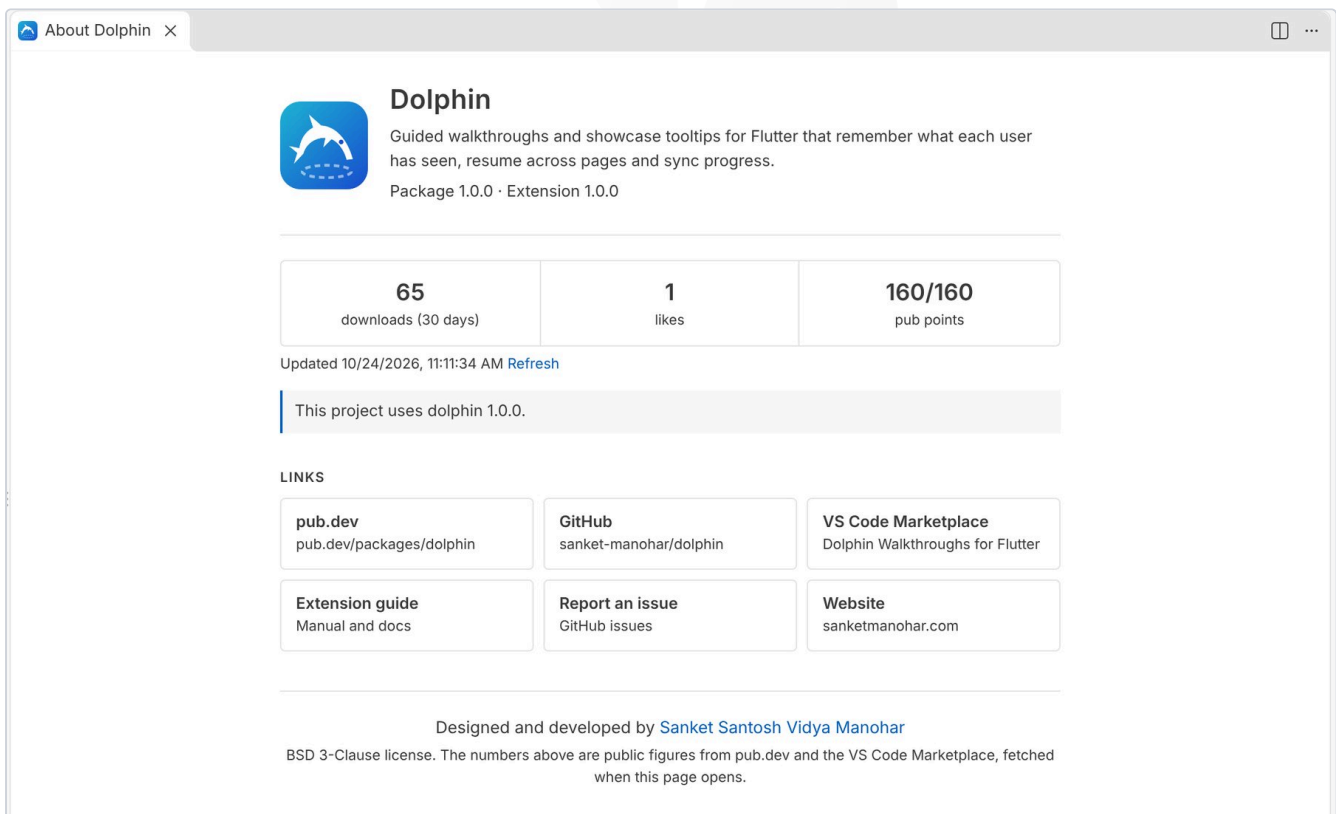


Figure 16.3 About Dolphin, with links to `pub.dev`, GitHub, the VS Code Marketplace, this guide online, issues and `sanketmanohar.com`.

Offline. The numbers are kept for an hour. Without a connection the page still opens with its links, and **Refresh** tries again.

17 Troubleshooting

PROBLEM	WHAT TO DO
The Widget Tree is empty	Open a <code>.dart</code> file and click into the editor. The tree shows the active Dart file only. Use Refresh after large outside changes such as a git checkout.
A panel stays blank	Close the panel and open it again. If it repeats, run Developer: Reload Window . An error message, if any, is shown in the panel itself.
The walkthrough does not start	Check that <code>Dolphin</code> wraps your app (section 8) and that the page has a <code>DolphinScope</code> with Auto start on. Steps already seen do not show again; use <code>drreset</code> while testing.
JSON content is not used	Make sure the config is listed under <code>flutter: assets:</code> and loaded in <code>main()</code> . Run Load JSON config at app startup .
“Planned” steps in the map	They are in the JSON but on no widget yet. Attach them with Planned step... (section 11).
The code is not formatted	Install the Dart extension, or turn off <code>dolphin.formatAfterEdit</code> .
A field I wrote by hand is gone from the form	It is still in the code. Values the form cannot show, such as variables, are kept as they are.
About shows no numbers	pub.dev could not be reached. The page works offline; try Refresh later.

Windows, macOS and Linux

- The extension has no native parts and runs the same on every system.
- Files with Windows line endings (CRLF) are read and written correctly.
- Paths are shown relative to the project, with forward slashes, on every system.
- Panels load their scripts only from the extension and re-send their data when they open, so they do not stay blank after VS Code restores them.

Undo. Every wrap, edit and removal is a normal editor change. `Ctrl+Z` takes it back, and your version control shows exactly what changed.

18 License and disclaimer

Dolphin Walkthroughs for Flutter is free to use under the BSD 3-Clause License. The full text is in the `LICENSE` file that comes with the extension.

No liability. The extension is provided “as is”, without warranty of any kind. The author is not liable for any damage or loss arising from its use. Review the changes it makes as you would any other code change before you commit them.

No data collected. The extension does not collect, store or send any personal data, code or usage data. It works offline. The About page reads only public download and like counts from pub.dev and the Visual Studio Marketplace, when you open it.

Links

RESOURCE	ADDRESS
Package on pub.dev	pub.dev/packages/dolphin
Package source	github.com/sanket-manohar/dolphin
Documentation	sanket-manohar.github.io/dolphin
Extension page	sanket-manohar.github.io/dolphin/vscode.html
VS Code Marketplace	marketplace.visualstudio.com · sanket-manohar.dolphin-walkthroughs
Author	sanketmanohar.com

Designed and developed by **Sanket Santosh Vidya Manohar**.

