# Лекция №3

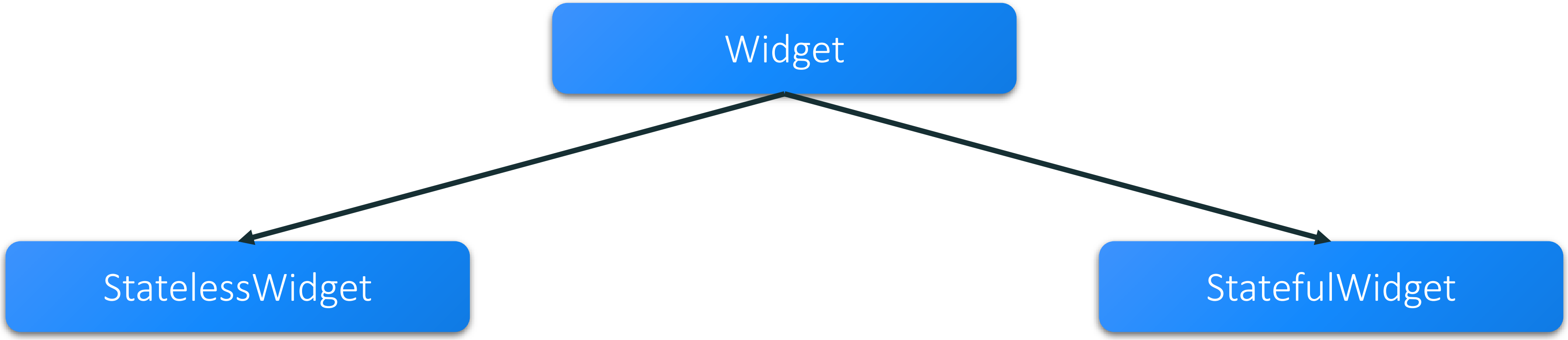
Widgets

# Everything is widget

# Widget

describe what their view should look like given their current configuration and state. (описывает представление view в соответствии с конфигурацией и состоянием)

Widget is immutable(неизменяемый)



Widget without state

Ex.:

- Text()
- Container()
- FlatButton()
- Etc.

Widget with some state

Ex.:

- Image()
- Checkbox()
- Form()
- Slider()
- Etc.

# Custom widget

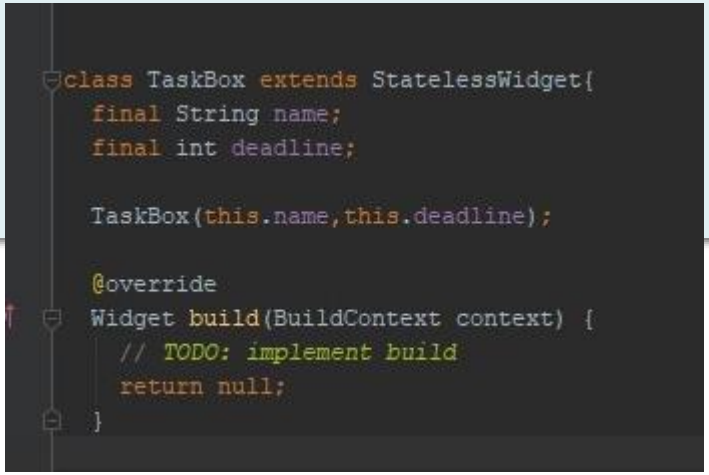
Stateless Widget

class NotStupidName extened StatelessWidget

Create constructor

Override build()

Ex:



```
class TaskBox extends StatelessWidget{
  final String name;
  final int deadline;

  TaskBox(this.name,this.deadline);

  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return null;
  }
```

Stateful Widget

class NotStupidName extened StatefulWidget

Create constructor

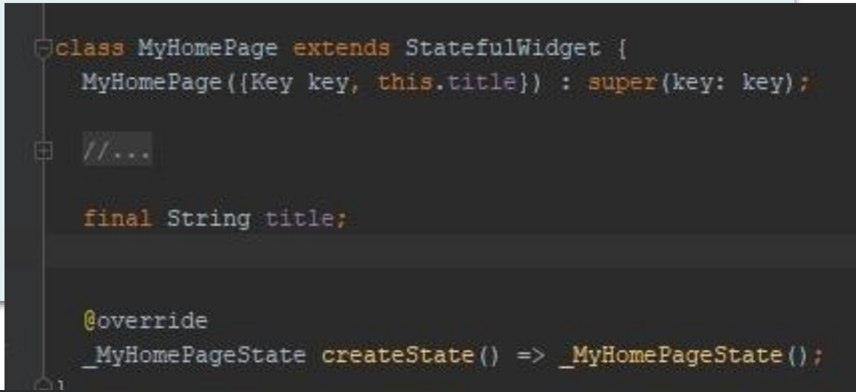
Override createState()

Create class extends State<T>

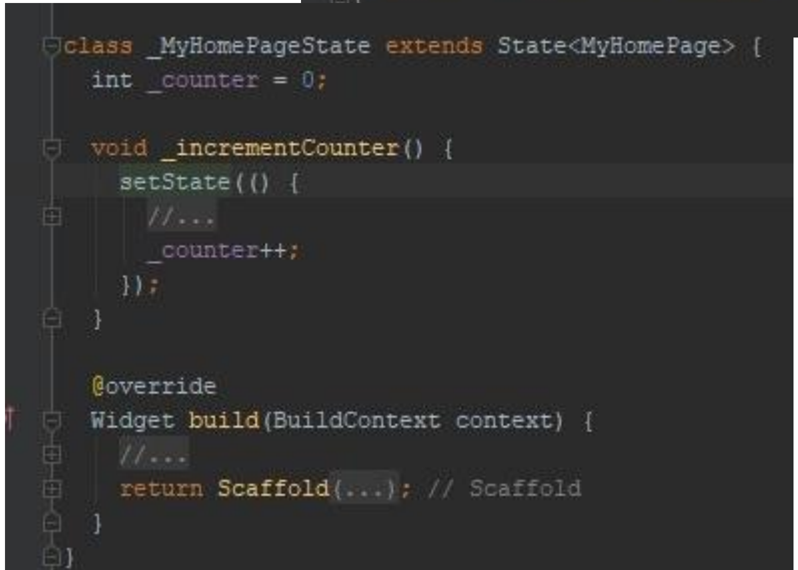
Override build()

setState()

Ex:



```
class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  //...

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```



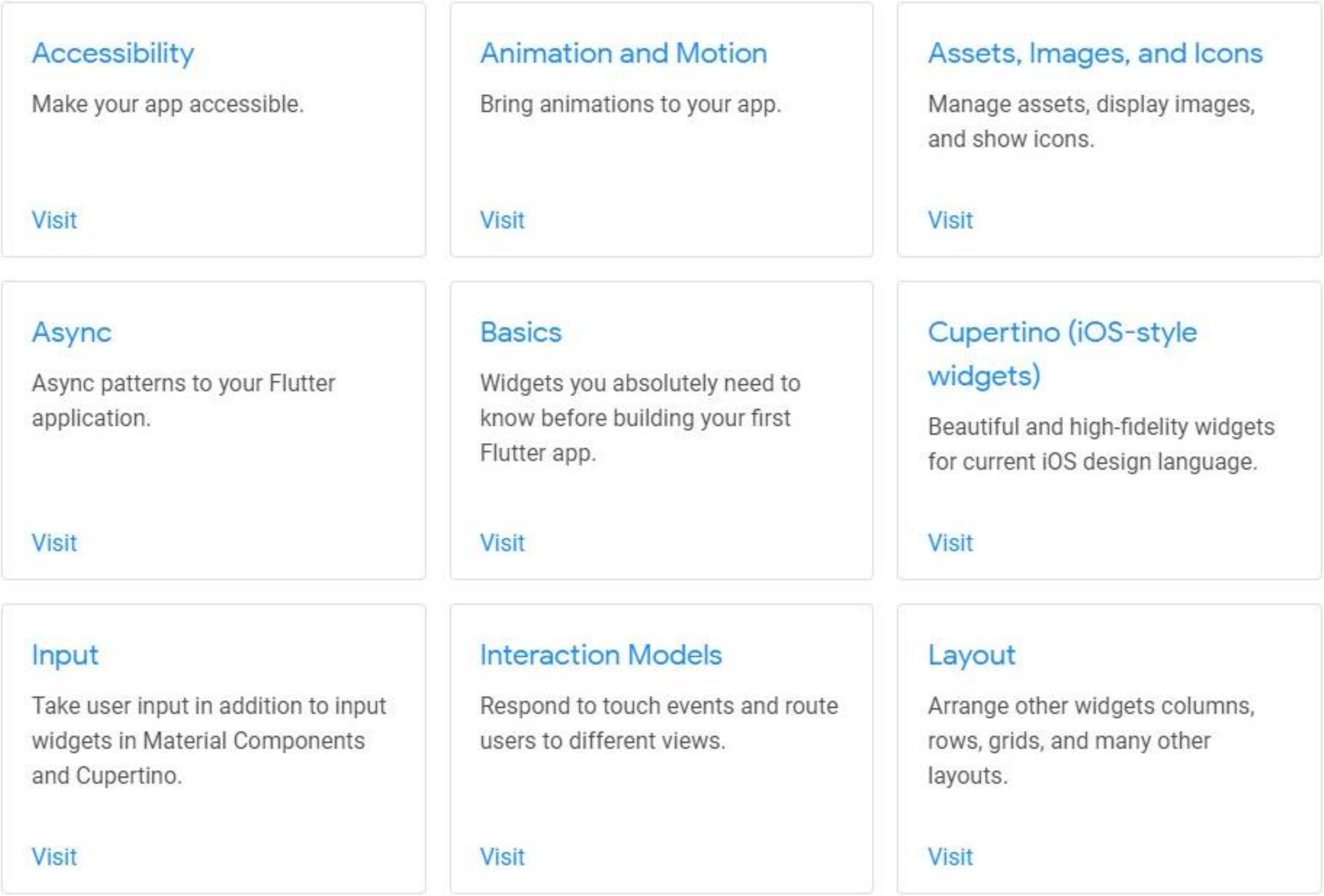
```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      //...
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    //...
    return Scaffold(...); // Scaffold
  }
}
```

# Рекомендации

Используйте  StatelessWidget всегда когда можете обойтись без StatefulWidget

Минимизируйте количество childs в StatefulWidget

# Widget types



### Accessibility

Make your app accessible.

Visit

### Animation and Motion

Bring animations to your app.

Visit

### Assets, Images, and Icons

Manage assets, display images, and show icons.

Visit

### Async

Async patterns to your Flutter application.

Visit

### Basics

Widgets you absolutely need to know before building your first Flutter app.

Visit

### Cupertino (iOS-style widgets)

Beautiful and high-fidelity widgets for current iOS design language.

Visit

### Input

Take user input in addition to input widgets in Material Components and Cupertino.

Visit

### Interaction Models

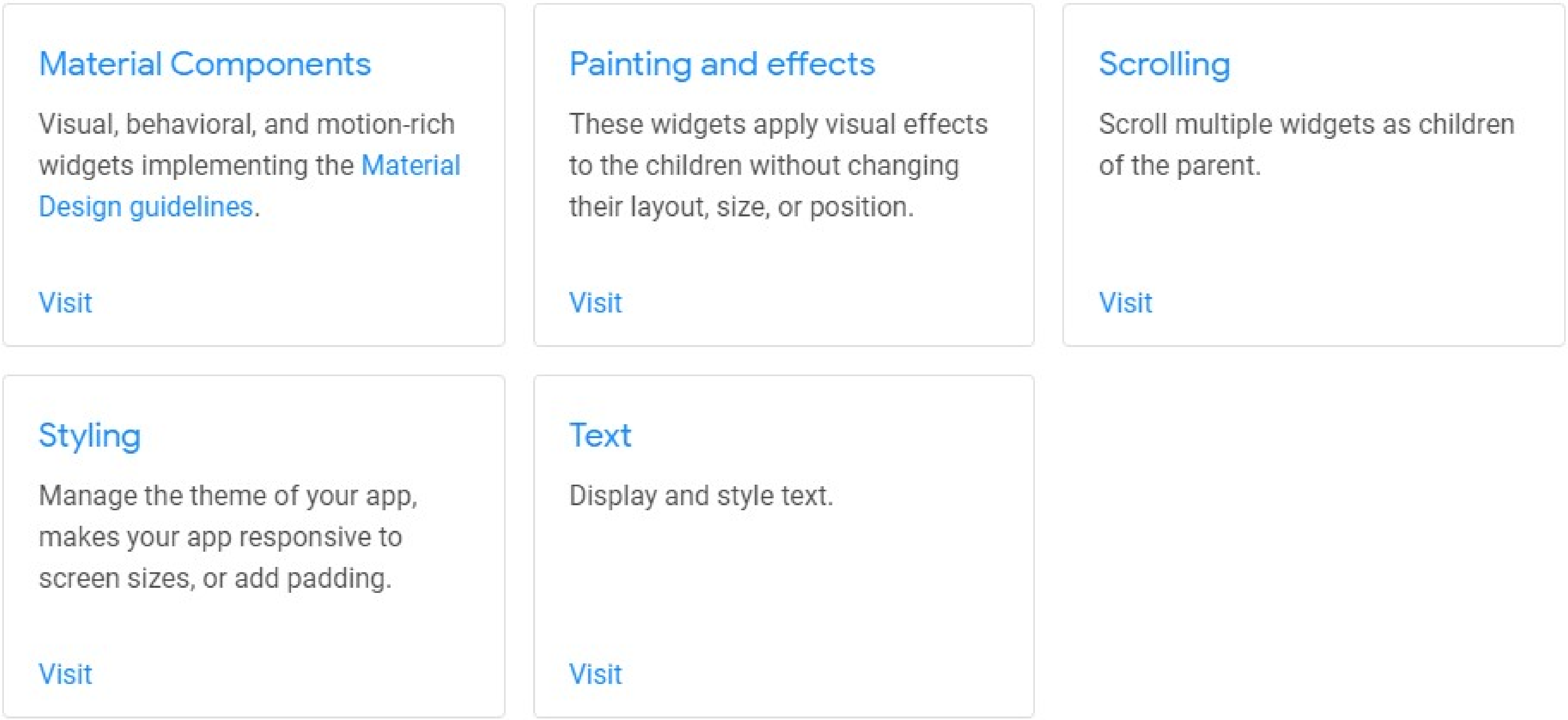
Respond to touch events and route users to different views.

Visit

### Layout

Arrange other widgets columns, rows, grids, and many other layouts.

Visit



### Material Components

Visual, behavioral, and motion-rich widgets implementing the Material Design guidelines.

Visit

### Painting and effects

These widgets apply visual effects to the children without changing their layout, size, or position.

Visit

### Scrolling

Scroll multiple widgets as children of the parent.

Visit

### Styling

Manage the theme of your app, makes your app responsive to screen sizes, or add padding.

Visit

### Text

Display and style text.

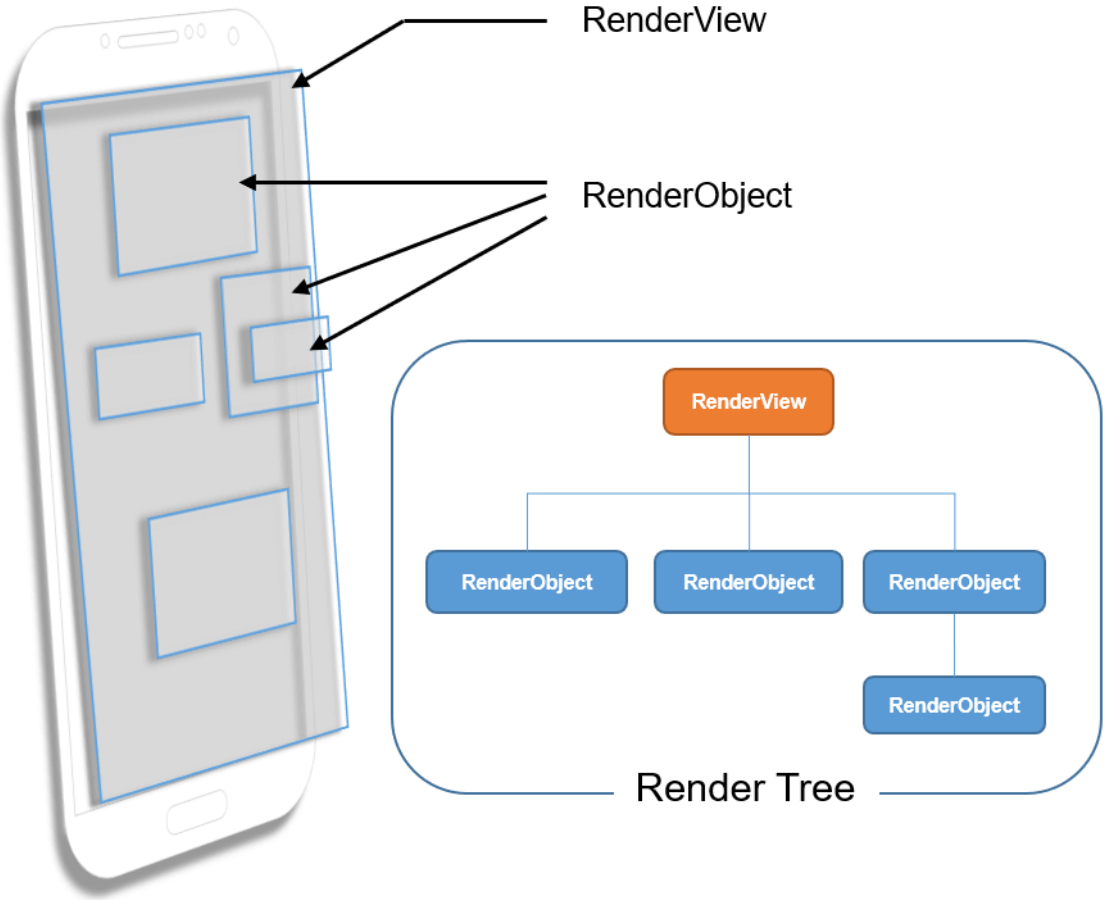
Visit

# RenderObjects

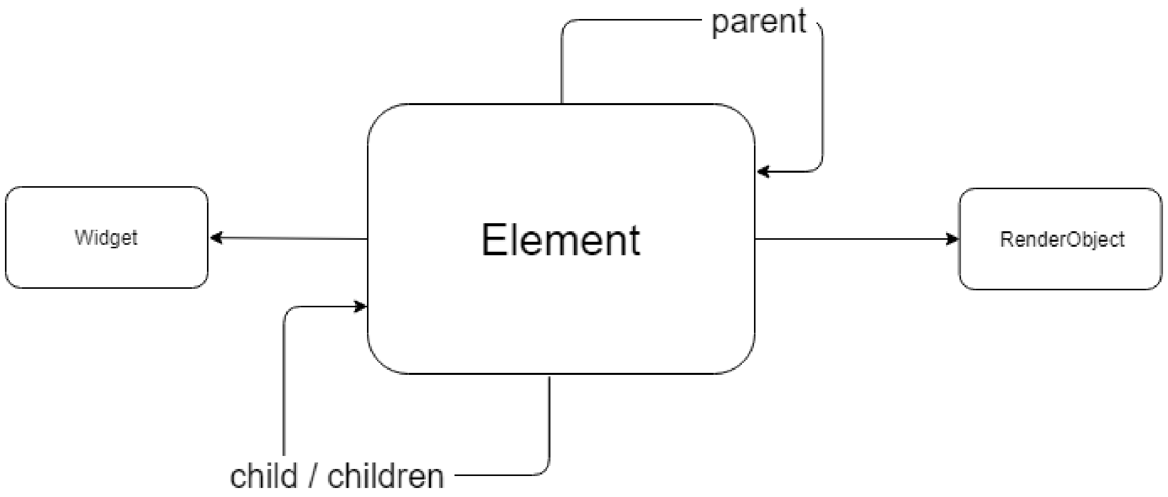
- Большой объект

- Дерево визуализации Flutter - это низкоуровневая система компоновки и рисования, основанная на сохраненном дереве объектов, наследуемых от RenderObject. Большинству разработчиков, использующих Flutter, не нужно напрямую взаимодействовать с деревом рендеринга. Вместо этого большинству разработчиков следует использовать виджеты, построенные с использованием дерева рендеринга.

-изменяемый



# Elements

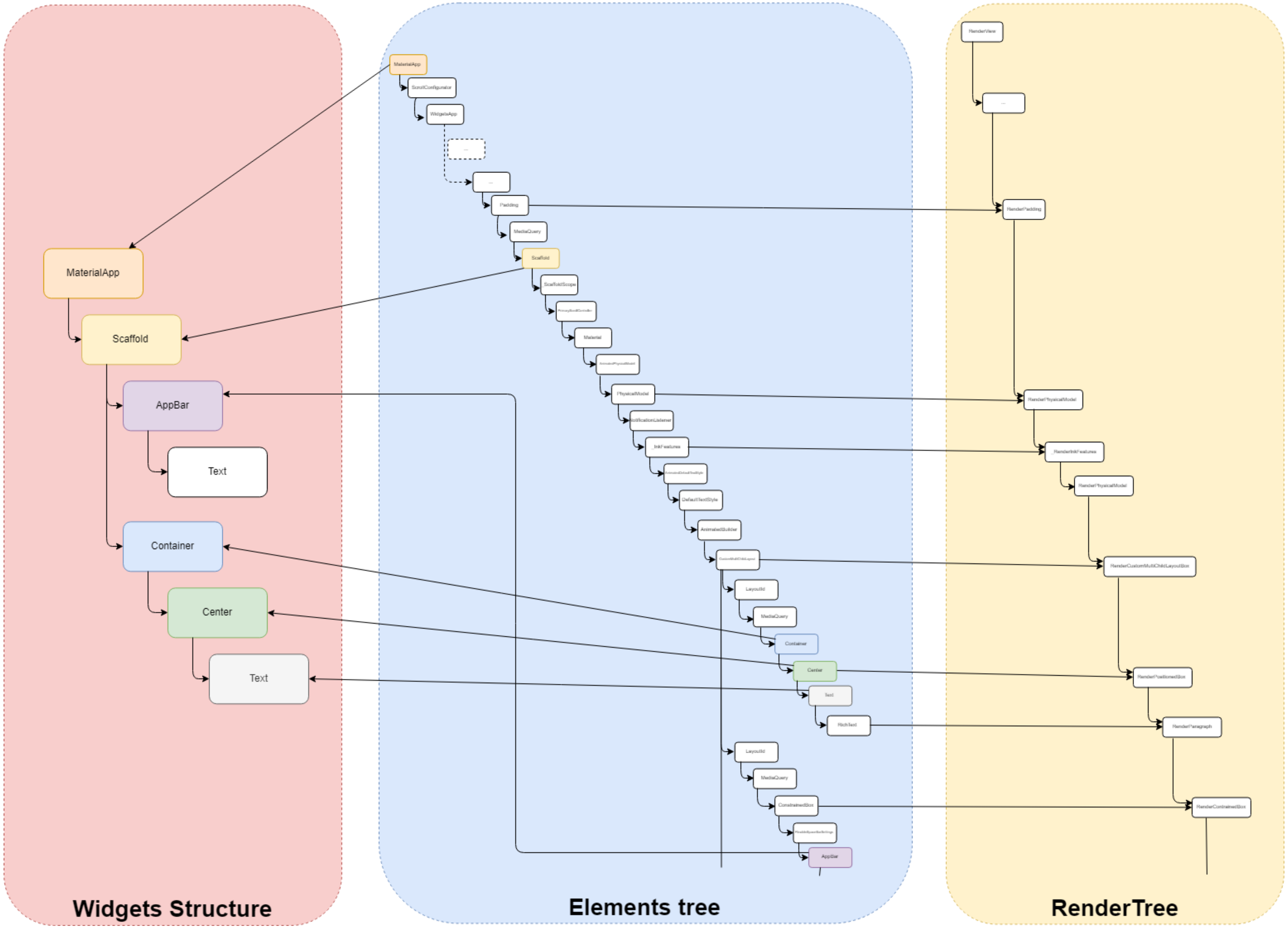


***Каждому** виджету соответствует **один***

*элемент. Элементы связаны друг с другом*

*и образуют дерево. Следовательно **элемент** является ссылкой на что-то в дереве*

***Элементы** определяют, как части отображаемых блоков связаны друг с другом*


MaterialApp
Scaffold
AppBar
Text
Container
Center
Text
Widgets Structure
MaterialApp
ScrollConfigurator
WidgetsApp
...
...
Padding
MediaQuery
Scaffold
_ScaffoldScope
PrimaryScrollController
Material
AnimatedPhysicalModel
PhysicalModel
NotificationListener
_InkFeatures
AnimatedDefaultTextStyle
DefaultTextStyle
AnimatedBuilder
CustomMultiChildLayout
LayoutId
MediaQuery
Container
Center
Text
RichText
LayoutId
MediaQuery
ConstrainedBox
FlexibleSpaceBarSettings
AppBar
Elements tree
RenderView
...
RenderPadding
RenderPhysicalModel
_RenderInkFeatures
RenderPhysicalModel
RenderCustomMultiChildLayoutBox
RenderPositionedBox
RenderParagraph
RenderConstrainedBox
RenderTree

# BuildContext

A handle to the location of a widget in the widget tree

This class presents a set of methods that can be used from StatelessWidget.build methods and from methods on State objects.

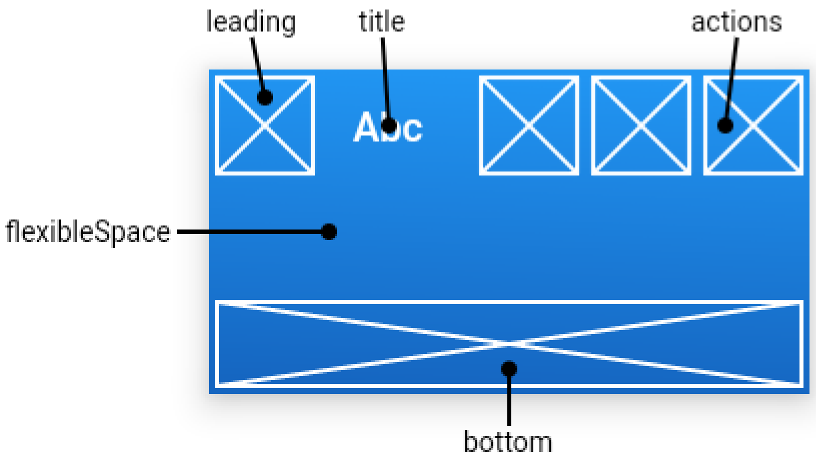
*BuildContext* соответствует элементу, связанному с виджетом, а также местоположению виджета в дереве

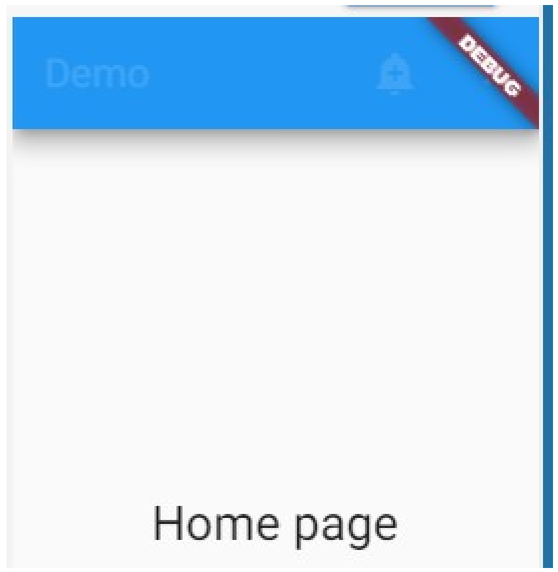
# Basic Widgets

- Appbar
- Column
- Container
- FlutterLogo
- Icon
- Image
- Placeholder
- RaisedButton
- Row
- Scaffold
- Text

# Appbar

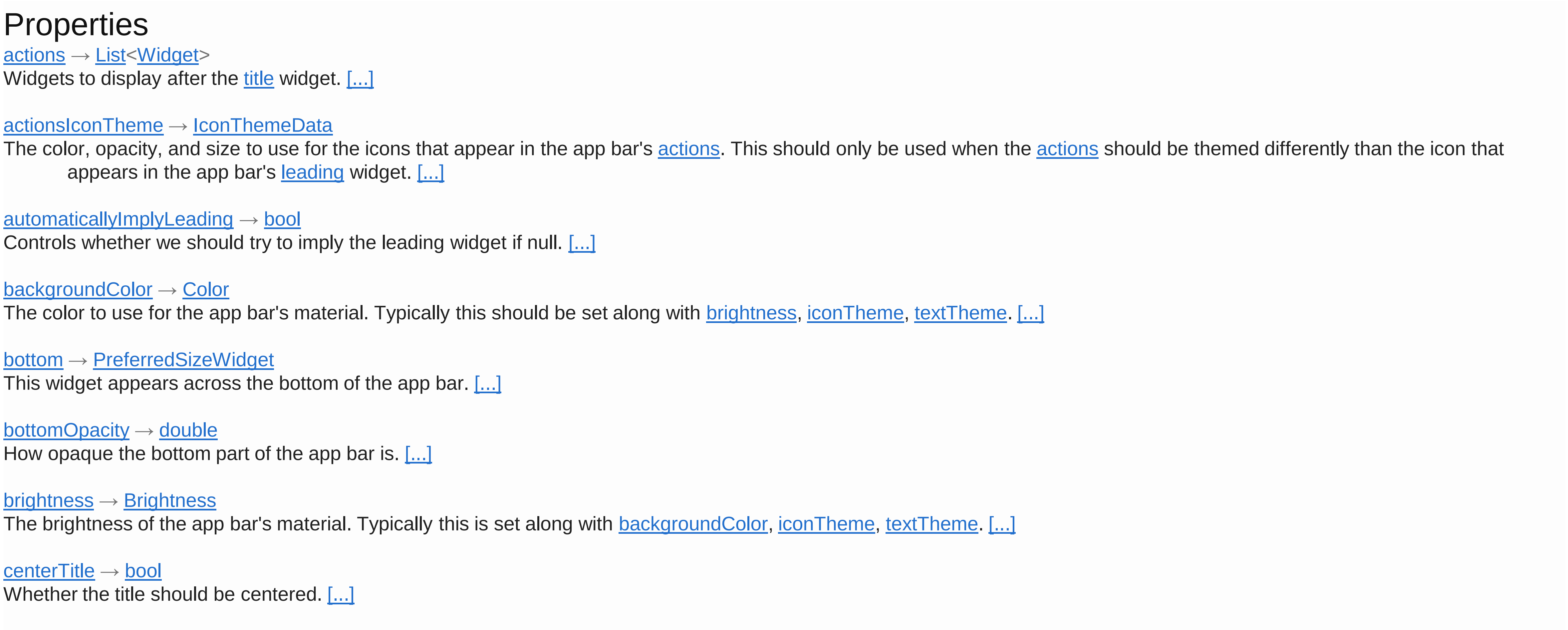
```
appBar: AppBar(
      title: const Text('Demo’),
       centerTitle:true,
      actions: <Widget>[
              //some widgets
       ],
    ),
```





Demo

Home page



# Properties

actions → List<Widget>
Widgets to display after the title widget. [...]

actionsIconTheme → IconThemeData
The color, opacity, and size to use for the icons that appear in the app bar's actions. This should only be used when the actions should be themed differently than the icon that appears in the app bar's leading widget. [...]

automaticallyImplyLeading → bool
Controls whether we should try to imply the leading widget if null. [...]

backgroundColor → Color
The color to use for the app bar's material. Typically this should be set along with brightness, iconTheme, textTheme. [...]

bottom → PreferredSizeWidget
This widget appears across the bottom of the app bar. [...]

bottomOpacity → double
How opaque the bottom part of the app bar is. [...]

brightness → Brightness
The brightness of the app bar's material. Typically this is set along with backgroundColor, iconTheme, textTheme. [...]

centerTitle → bool
Whether the title should be centered. [...]

elevation → double
The z-coordinate at which to place this app bar relative to its parent. [...]

flexibleSpace → Widget
This widget is stacked behind the toolbar and the tab bar. It's height will be the same as the app bar's overall height. [...]

iconTheme → IconThemeData
The color, opacity, and size to use for app bar icons. Typically this is set along with backgroundColor, brightness, textTheme. [...]

leading → Widget
A widget to display before the title. [...]

preferredSize → Size
A size whose height is the sum of kToolbarHeight and the bottom widget's preferred height. [...]

primary → bool
Whether this app bar is being displayed at the top of the screen. [...]

shape → ShapeBorder
The material's shape as well its shadow. [...]

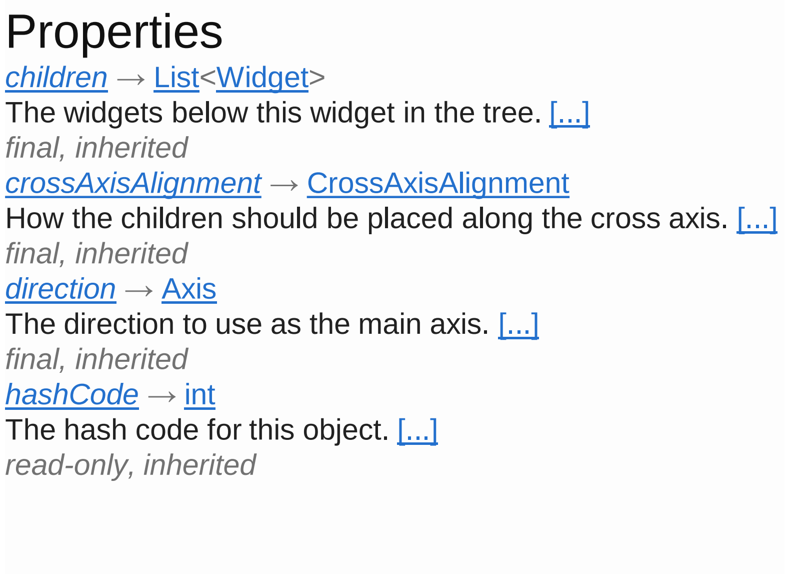
textTheme → TextTheme
The typographic styles to use for text in the app bar. Typically this is set along with brightness backgroundColor, iconTheme. [...]

title → Widget
The primary widget displayed in the app bar. [...]

titleSpacing → double
The spacing around title content on the horizontal axis. This spacing is applied even if there is no leading content or actions. If you want title to take all the space available, set this value to 0.0. [...]

toolbarOpacity → double
How opaque the toolbar part of the app bar is. [...]

# Column



## Properties

children → List<Widget>

The widgets below this widget in the tree. [...]

*final, inherited*

crossAxisAlignment → CrossAxisAlignment

How the children should be placed along the cross axis. [...]

*final, inherited*

direction → Axis

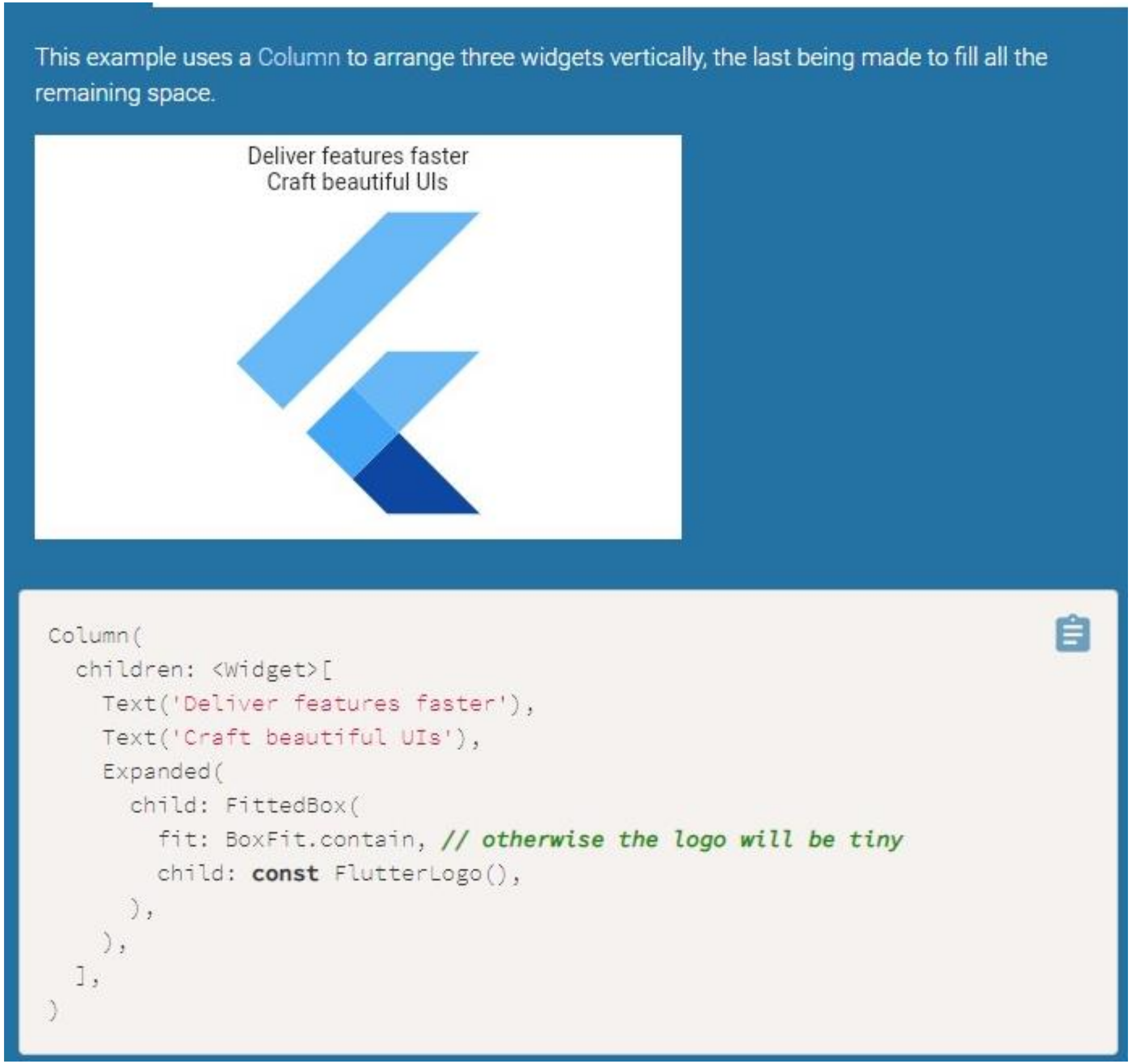
The direction to use as the main axis. [...]

*final, inherited*

hashCode → int

The hash code for this object. [...]

*read-only, inherited*



This example uses a Column to arrange three widgets vertically, the last being made to fill all the remaining space.

Deliver features faster
Craft beautiful UIs

```
Column(
  children: <Widget>[
    Text('Deliver features faster'),
    Text('Craft beautiful UIs'),
    Expanded(
      child: FittedBox(
        fit: BoxFit.contain, // otherwise the logo will be tiny
        child: const FlutterLogo(),
      ),
    ),
  ],
)
```

*key* → Key
Controls how one widget replaces another widget in the tree. [...]
*final, inherited*

*mainAxisAlignment* → MainAxisAlignment
How the children should be placed along the main axis. [...]
*final, inherited*

*mainAxisSize* → MainAxisSize
How much space should be occupied in the main axis. [...]
*final, inherited*

*runtimeType* → Type
A representation of the runtime type of the object.
*read-only, inherited*

*textBaseline* → TextBaseline
If aligning items according to their baseline, which baseline to use.
*final, inherited*

*textDirection* → TextDirection
Determines the order to lay children out horizontally and how to interpret `start` and `end` in the horizontal direction. [...]
*final, inherited*

*verticalDirection* → VerticalDirection
Determines the order to lay children out vertically and how to interpret `start` and `end` in the vertical direction. [...]
final, inherited

# Container

## Properties

alignment → AlignmentGeometry
Align the child within the container. [...]
*final*

child → Widget
The child contained by the container. [...]
*final*

constraints → BoxConstraints
Additional constraints to apply to the child. [...]
*final*

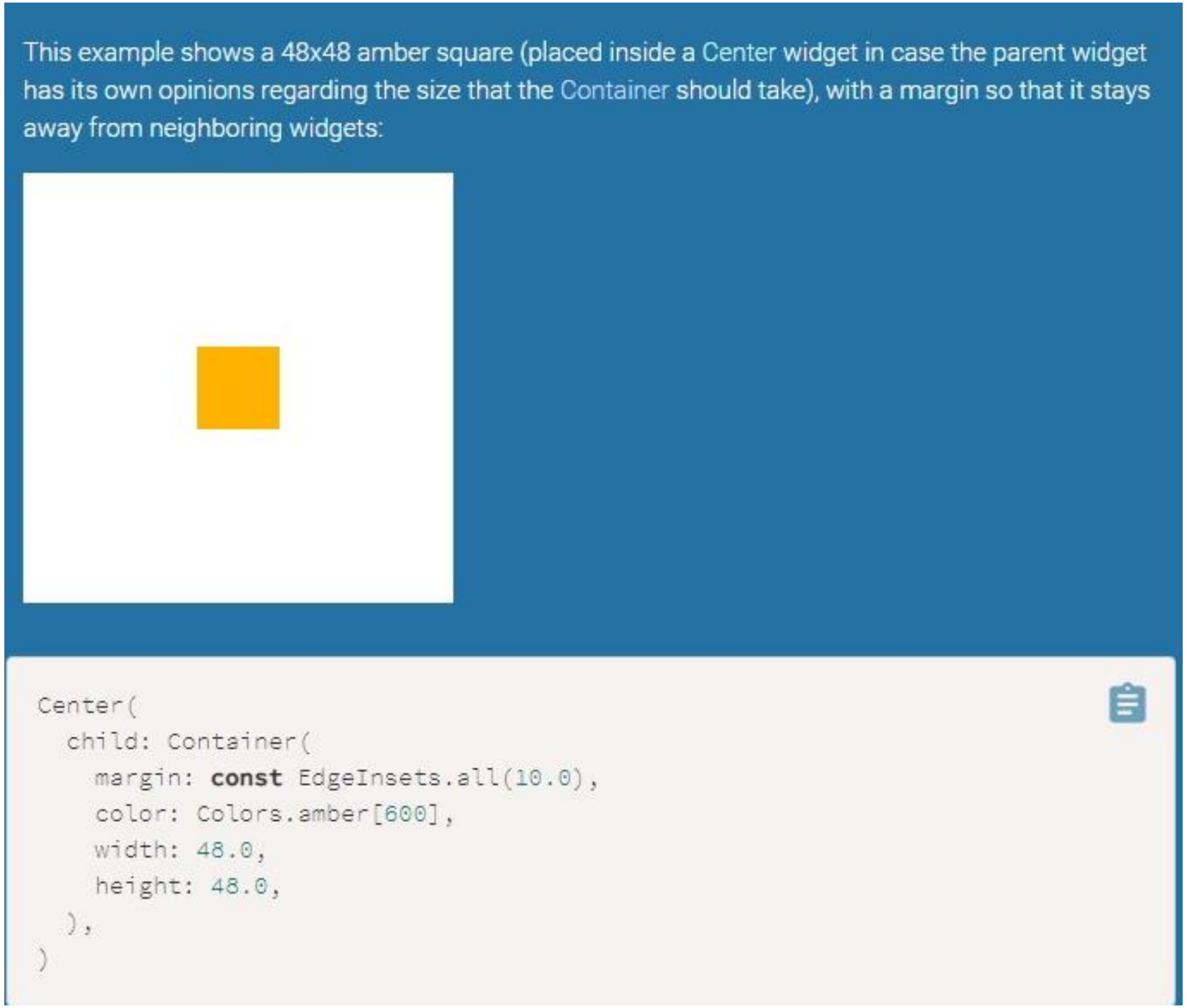
decoration → Decoration
The decoration to paint behind the child. [...]
*final*

foregroundDecoration → Decoration
The decoration to paint in front of the child.
*final*

margin → EdgeInsetsGeometry
Empty space to surround the decoration and child.
*final*

padding → EdgeInsetsGeometry
Empty space to inscribe inside the decoration. The child, if any, is placed inside this padding. [...]
*final*

transform → Matrix4
The transformation matrix to apply before painting the container.
*final*



This example shows a 48x48 amber square (placed inside a Center widget in case the parent widget has its own opinions regarding the size that the Container should take), with a margin so that it stays away from neighboring widgets:

```
Center(
  child: Container(
    margin: const EdgeInsets.all(10.0),
    color: Colors.amber[600],
    width: 48.0,
    height: 48.0,
  ),
)
```

Flutter

# Icon

## Properties

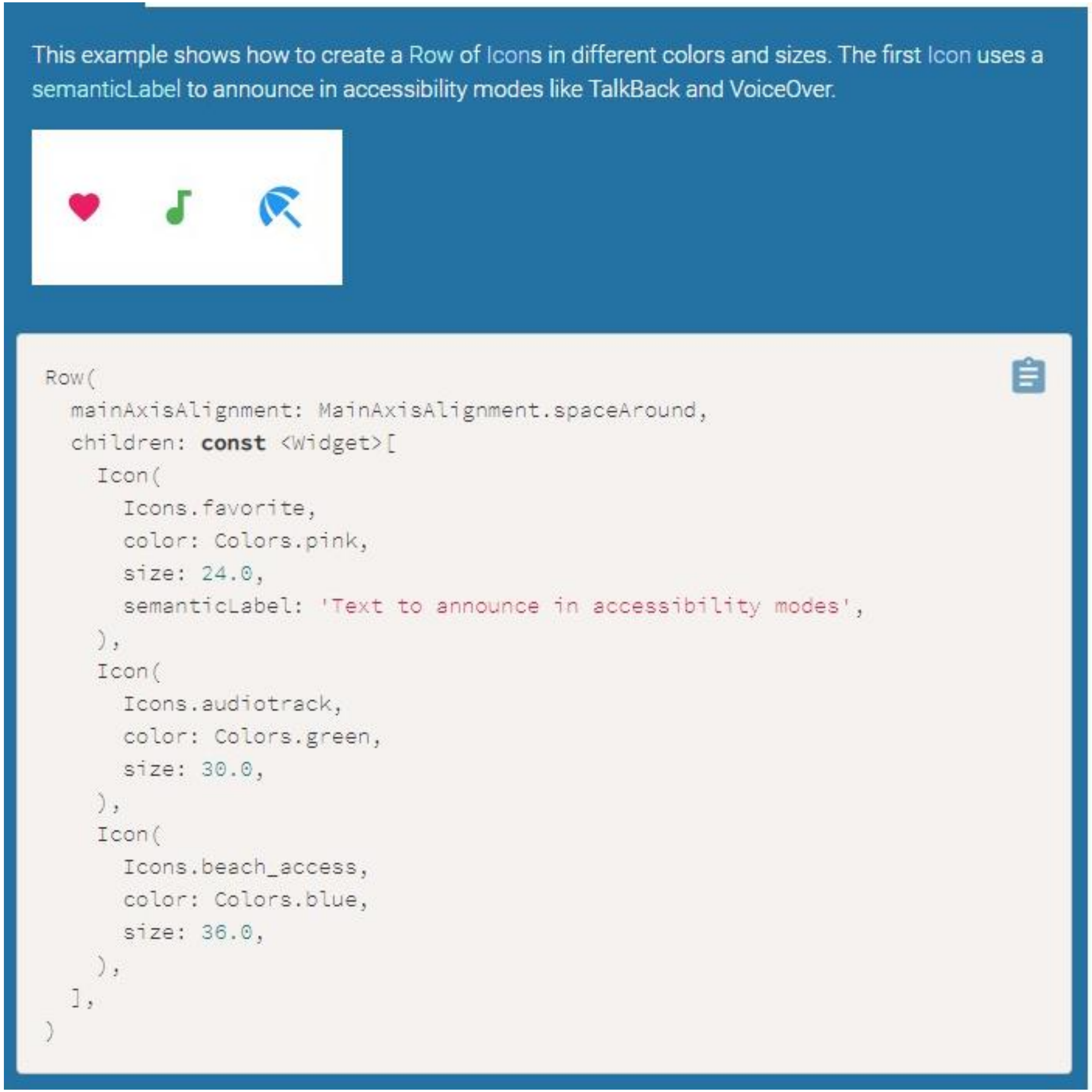
color → Color
The color to use when drawing the icon. [...]
*final*

icon → IconData
The icon to display. The available icons are described in Icons. [...]
*final*

semanticLabel → String
Semantic label for the icon. [...]
*final*

size → double
The size of the icon in logical pixels. [...]
*final*

textDirection → TextDirection
The text direction to use for rendering the icon. [...]
*final*



This example shows how to create a Row of Icons in different colors and sizes. The first Icon uses a semanticLabel to announce in accessibility modes like TalkBack and VoiceOver.

```
Row(
  mainAxisAlignment: MainAxisAlignment.spaceAround,
  children: const <Widget>[
    Icon(
      Icons.favorite,
      color: Colors.pink,
      size: 24.0,
      semanticLabel: 'Text to announce in accessibility modes',
    ),
    Icon(
      Icons.audiotrack,
      color: Colors.green,
      size: 30.0,
    ),
    Icon(
      Icons.beach_access,
      color: Colors.blue,
      size: 36.0,
    ),
  ],
)
```

# Image

## Properties

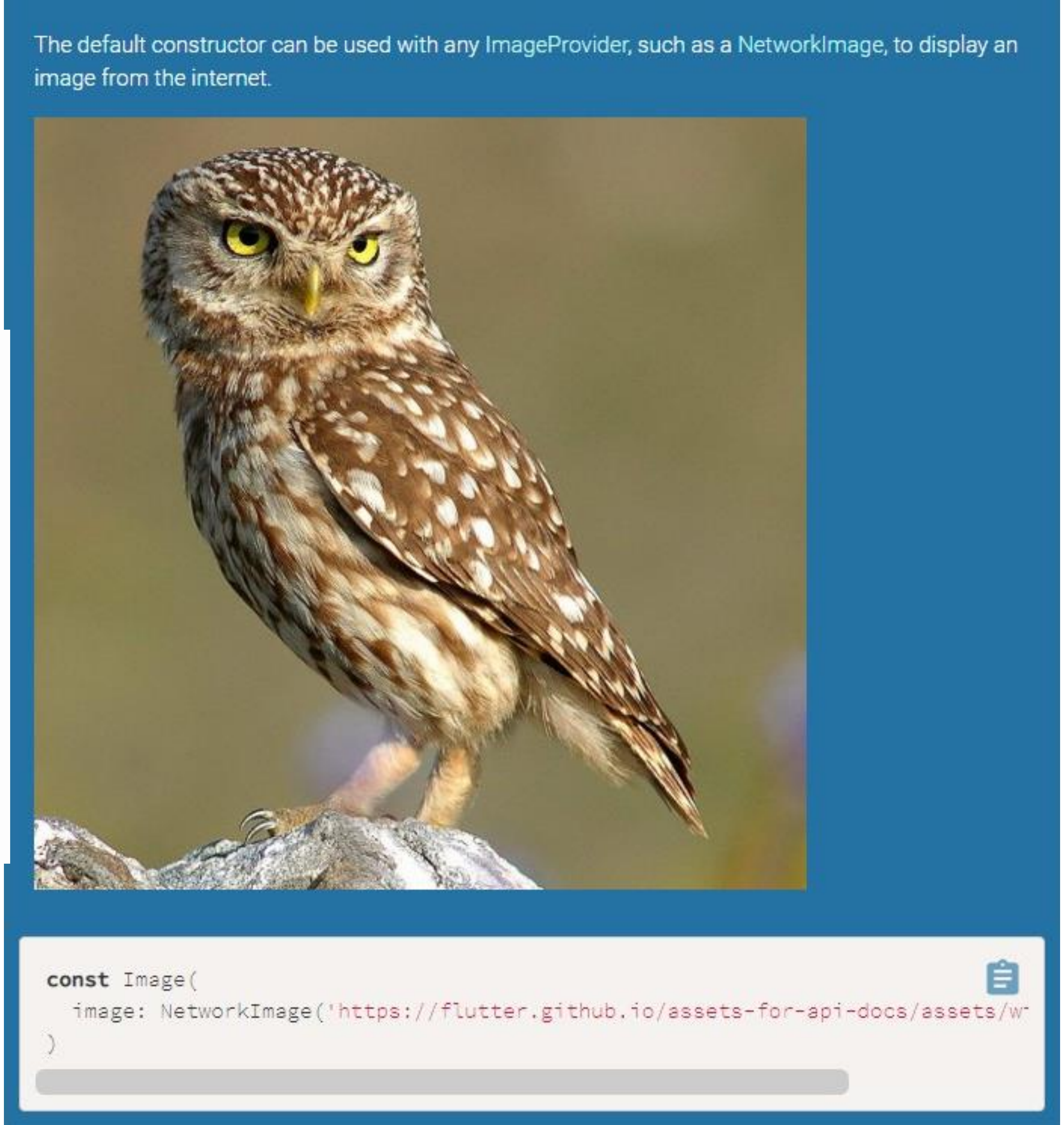
alignment → AlignmentGeometry
How to align the image within its bounds. [...]
*final*

centerSlice → Rect
The center slice for a nine-patch image. [...]
*final*

color → Color
If non-null, this color is blended with each image pixel using colorBlendMode.
*final*

colorBlendMode → BlendMode
Used to combine color with this image. [...]
*final*

excludeFromSemantics → bool
Whether to exclude this image from semantics. [...]
*final*



The default constructor can be used with any ImageProvider, such as a NetworkImage, to display an image from the internet.

```
const Image(
  image: NetworkImage('https://flutter.github.io/assets-for-api-docs/assets/w
)
```

filterQuality → FilterQuality
Used to set the FilterQuality of the image. [...]
*final*
fit → BoxFit
How to inscribe the image into the space allocated during layout. [...]
*final*
frameBuilder → ImageFrameBuilder
A builder function responsible for creating the widget that represents this image. [...]
*final*
gaplessPlayback → bool
Whether to continue showing the old image (true), or briefly show nothing (false), when the image provider changes.
*final*
height → double
If non-null, require the image to have this height. [...]
*final*
image → ImageProvider
The image to display.
*final*
loadingBuilder → ImageLoadingBuilder
A builder that specifies the widget to display to the user while an image is still loading. [...]
*final*
matchTextDirection → bool
Whether to paint the image in the direction of the TextDirection. [...]
*final*
repeat → ImageRepeat
How to paint any portions of the layout bounds not covered by the image.
*final*
semanticLabel → String
A Semantic description of the image. [...]
*final*
width → double
If non-null, require the image to have this width. [...]
*final*

Flutter

# Placeholder

## Properties

color → Color

The color to draw the placeholder box.

*final*

fallbackHeight → double

The height to use when the placeholder is in a situation with an unbounded height. [...]

*final*

fallbackWidth → double

The width to use when the placeholder is in a situation with an unbounded width. [...]

*final*

strokeWidth → double

The width of the lines in the placeholder box.

*final*

Flutter

# RaisedButton

## Properties

*animationDuration* → Duration
Defines the duration of animated changes for shape and elevation. [...]
*final, inherited*

*autofocus* → bool
True if this widget will be selected as the initial focus when no other node in its scope is currently focused. [...]
*final, inherited*

*child* → Widget
The button's label. [...]
*final, inherited*

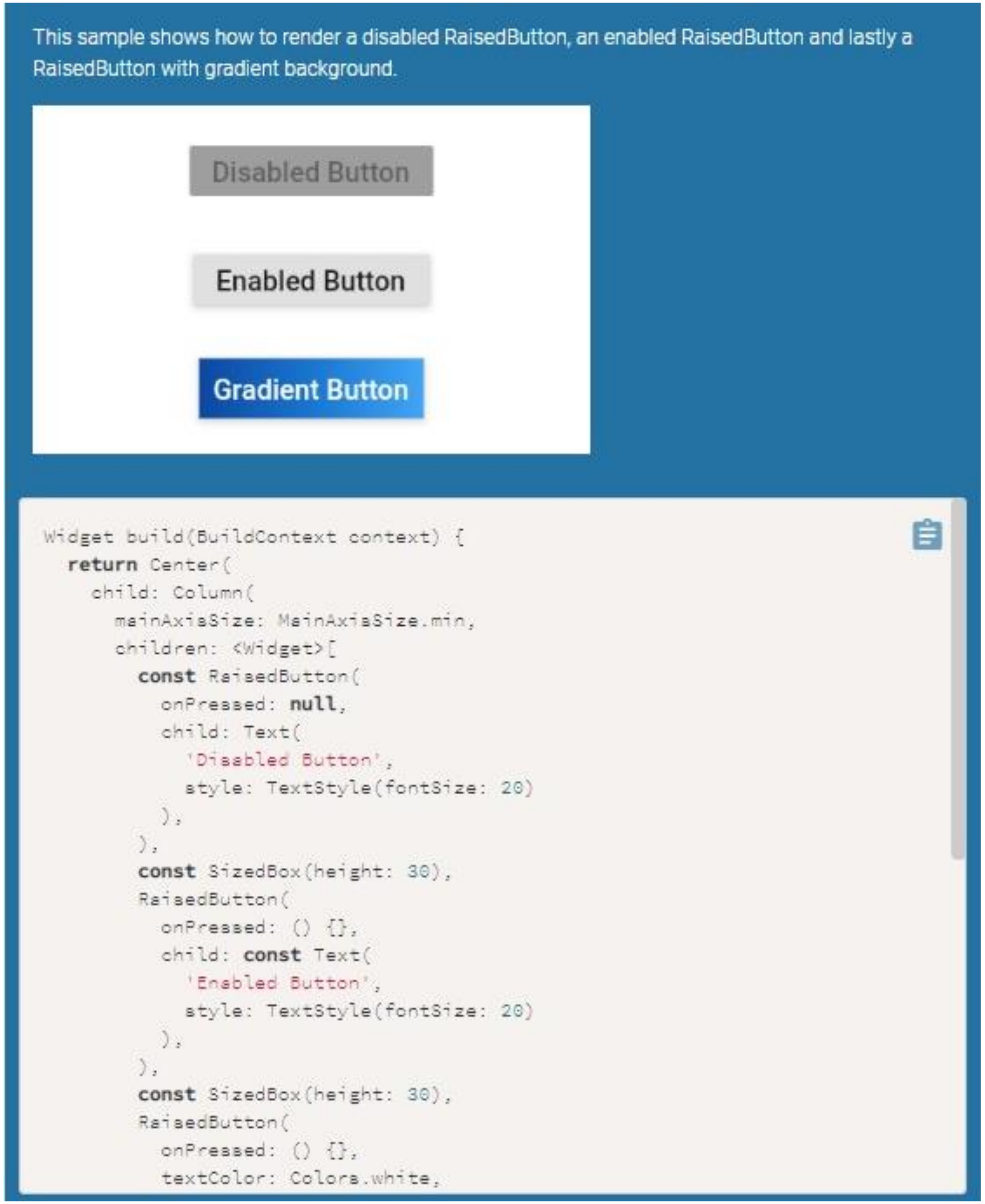
*clipBehavior* → Clip
The content will be clipped (or not) according to this option. [...]
*final, inherited*

*color* → Color
The button's fill color, displayed by its Material, while it is in its default (unpressed, enabled) state. [...]
*final, inherited*

*colorBrightness* → Brightness
The theme brightness to use for this button. [...]
*final, inherited*

*disabledColor* → Color
The fill color of the button when the button is disabled. [...]
*final, inherited*

*disabledElevation* → double
The elevation for the button's Material relative to its parent when the button is not enabled. [...]
*final, inherited*



This sample shows how to render a disabled RaisedButton, an enabled RaisedButton and lastly a RaisedButton with gradient background.

```
Widget build(BuildContext context) {
  return Center(
    child: Column(
      mainAxisSize: MainAxisSize.min,
      children: <Widget>[
        const RaisedButton(
          onPressed: null,
          child: Text(
            'Disabled Button',
            style: TextStyle(fontSize: 20)
          ),
        ),
        const SizedBox(height: 30),
        RaisedButton(
          onPressed: () {},
          child: const Text(
            'Enabled Button',
            style: TextStyle(fontSize: 20)
          ),
        ),
        const SizedBox(height: 30),
        RaisedButton(
          onPressed: () {},
          textColor: Colors.white,
```

*disabledTextColor* → Color
The color to use for this button's text when the button is disabled. [...]
*final, inherited*

*elevation* → double
The z-coordinate at which to place this button relative to its parent. [...]
*final, inherited*

*enabled* → bool
Whether the button is enabled or disabled. [...]
*read-only, inherited*

*enableFeedback* → bool
Whether detected gestures should provide acoustic and/or haptic feedback. [...]
*final, inherited*

*focusColor* → Color
The fill color of the button's Material when it has the input focus. [...]
*final, inherited*

*focusElevation* → double
The elevation for the button's Material when the button is enabled and has the input focus. [...]
*final, inherited*

*focusNode* → FocusNode
An optional focus node to use as the focus node for this widget. [...]
*final, inherited*

*hashCode* → int
The hash code for this object. [...]
*read-only, inherited*

*height* → double
The vertical extent of the button. [...]
*final, inherited*

*highlightColor* → Color
The highlight color of the button's InkWell. [...]
*final, inherited*

*highlightElevation* → double
The elevation for the button's Material relative to its parent when the button is enabled and pressed. [...]
*final, inherited*

*hoverColor* → Color

The fill color of the button's Material when a pointer is hovering over it. [...]

*final, inherited*

*hoverElevation* → double

The elevation for the button's Material when the button is enabled and a pointer is hovering over it. [...]

*final, inherited*

*key* → Key

Controls how one widget replaces another widget in the tree.

*final, inherited*

*materialTapTargetSize* → MaterialTapTargetSize

Configures the minimum size of the tap target. [...]

*final, inherited*

*minWidth* → double

The smallest horizontal extent that the button will occupy. [...]

*final, inherited*

*onHighlightChanged* → ValueChanged<bool>

Called by the underlying InkWell widget's InkWell.onHighlightChanged callback. [...]

*final, inherited*

*onLongPress* → VoidCallback

The callback that is called when the button is long-pressed. [...]

*final, inherited*

*onPressed* → VoidCallback

The callback that is called when the button is tapped or otherwise activated. [...]

*final, inherited*

*padding* → EdgeInsetsGeometry

The internal padding for the button's child. [...]

*final, inherited*

*runtimeType* → Type

A representation of the runtime type of the object.

*read-only, inherited*

*shape* → ShapeBorder

The shape of the button's Material. [...]

*final, inherited*

*splashColor* → Color

The splash color of the button's InkWell. [...]

*final, inherited*

*textColor* → Color

The color to use for this button's text. [...]

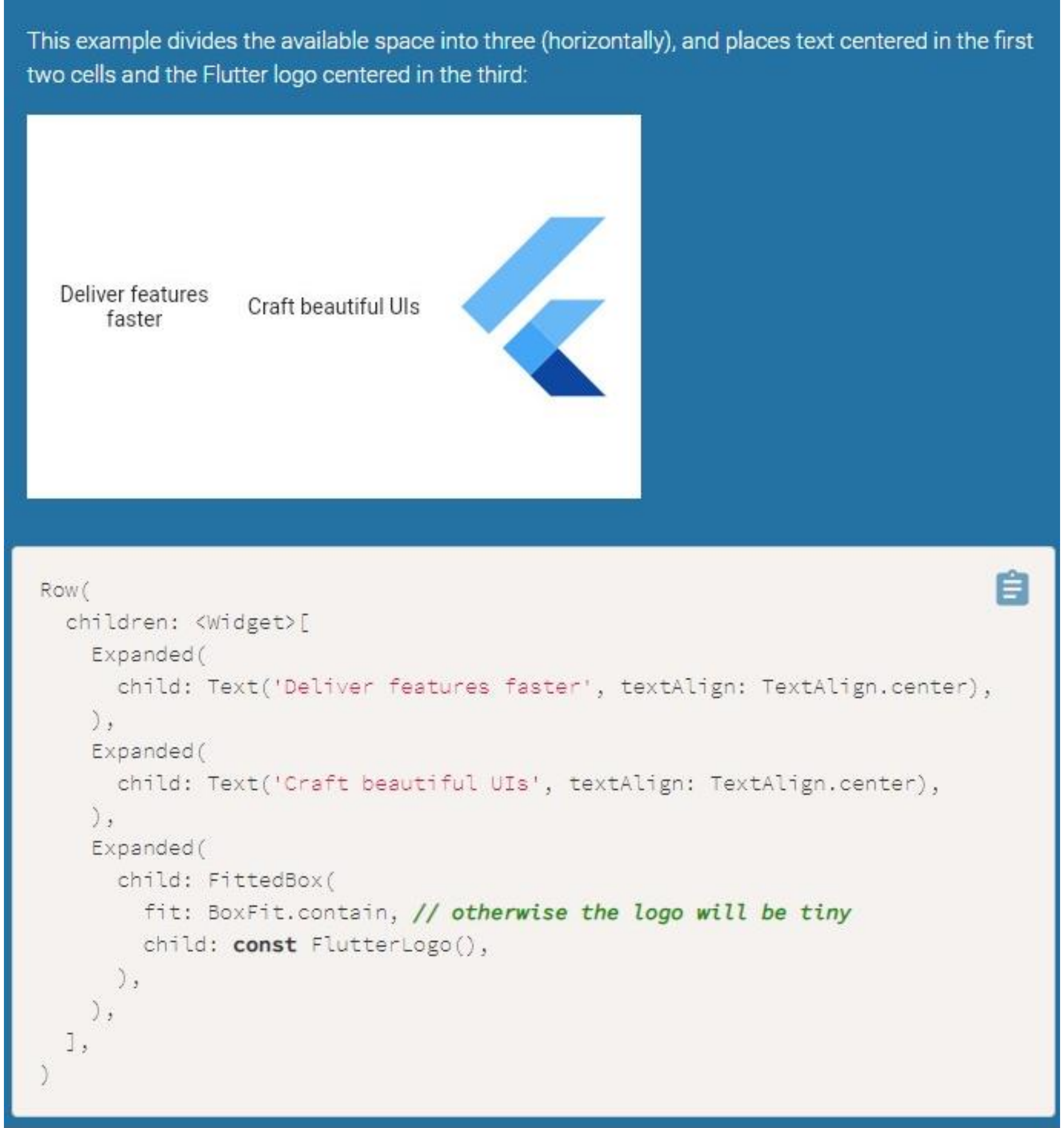
*final, inherited*

*textTheme* → ButtonTextTheme

Defines the button's base colors, and the defaults for the button's minimum size, internal padding, and shape. [...]

*final, inherited*

# Row



This example divides the available space into three (horizontally), and places text centered in the first two cells and the Flutter logo centered in the third:

```
Row(
  children: <Widget>[
    Expanded(
      child: Text('Deliver features faster', textAlign: TextAlign.center),
    ),
    Expanded(
      child: Text('Craft beautiful UIs', textAlign: TextAlign.center),
    ),
    Expanded(
      child: FittedBox(
        fit: BoxFit.contain, // otherwise the logo will be tiny
        child: const FlutterLogo(),
      ),
    ),
  ],
)
```

## Properties

*children* → List<Widget>

The widgets below this widget in the tree. [...]

*final, inherited*

*crossAxisAlignment* → CrossAxisAlignment

How the children should be placed along the cross axis. [...]

*final, inherited*

*direction* → Axis

The direction to use as the main axis. [...]

*final, inherited*

*hashCode* → int

The hash code for this object. [...]

*read-only, inherited*

*key* → Key

Controls how one widget replaces another widget in the tree. [...]

*final, inherited*

*mainAxisAlignment* → MainAxisAlignment

How the children should be placed along the main axis. [...]

*final, inherited*

*mainAxisSize* → MainAxisSize

How much space should be occupied in the main axis. [...]

*final, inherited*

*runtimeType* → Type

A representation of the runtime type of the object.

*read-only, inherited*

*textBaseline* → TextBaseline

If aligning items according to their baseline, which baseline to use.

*final, inherited*

*textDirection* → TextDirection

Determines the order to lay children out horizontally and how to interpret `start` and `end` in the horizontal direction. [...]

*final, inherited*

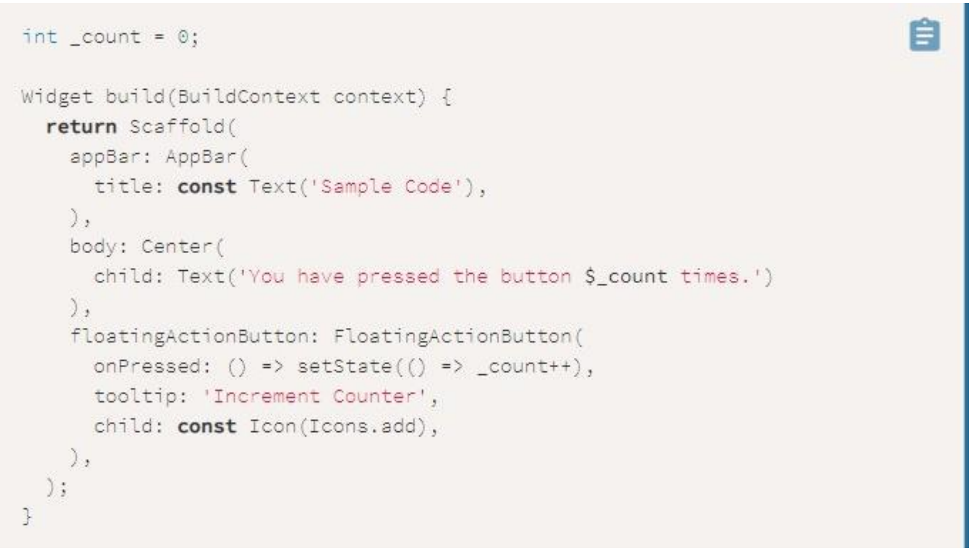
*verticalDirection* → VerticalDirection

Determines the order to lay children out vertically and how to interpret `start` and `end` in the vertical direction. [...]

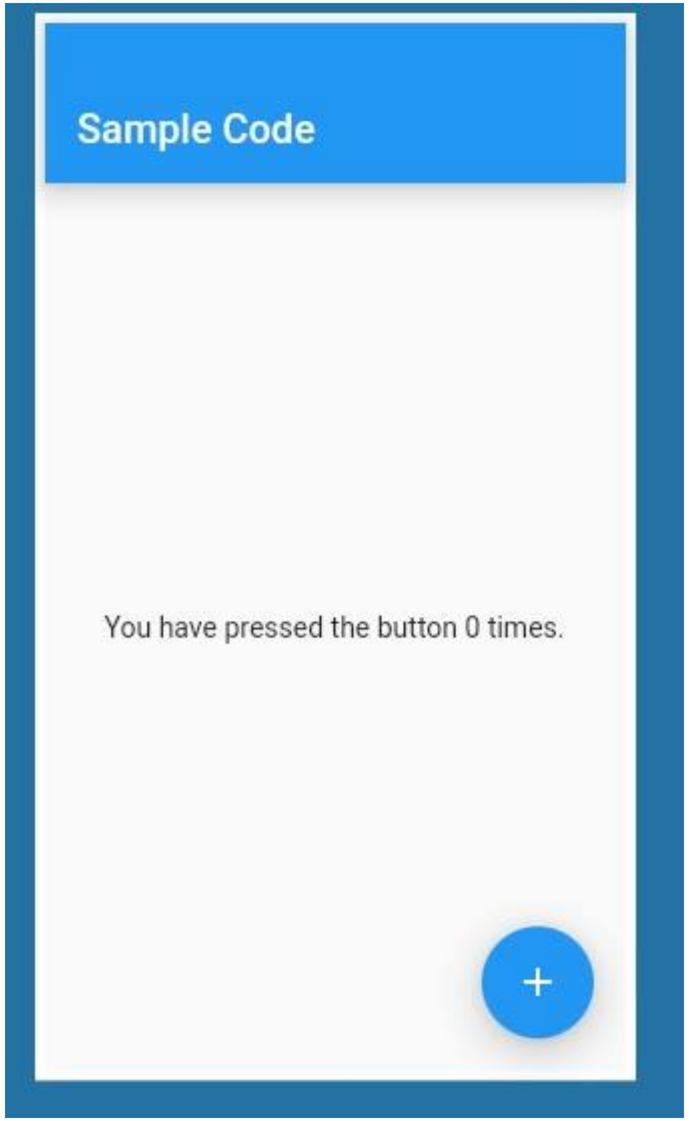
*final, inherited*

# Scafold

Implements the basic material design visual layout structure.



```
int _count = 0;

Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Sample Code'),
    ),
    body: Center(
      child: Text('You have pressed the button $_count times.')
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () => setState(() => _count++),
      tooltip: 'Increment Counter',
      child: const Icon(Icons.add),
    ),
  );
}
```



Sample Code

You have pressed the button 0 times.

# Properties

appBar → PreferredSizeWidget

An app bar to display at the top of the scaffold.

*final*

backgroundColor → Color

The color of the Material widget that underlies the entire Scaffold. [...]

*final*

body → Widget

The primary content of the scaffold. [...]

*final*

bottomNavigationBar → Widget

A bottom navigation bar to display at the bottom of the scaffold. [...]

*final*

bottomSheet → Widget

The persistent bottom sheet to display. [...]

*final*

drawer → Widget

A panel displayed to the side of the body, often hidden on mobile devices. Swipes in from either left-to-right (TextDirection.ltr) or right-to-left (TextDirection.rtl) [...]

*final*

drawerDragStartBehavior → DragStartBehavior

Determines the way that drag start behavior is handled. [...]

*final*

drawerEdgeDragWidth → double

The width of the area within which a horizontal swipe will open the drawer. [...]

*final*

drawerScrimColor → Color

The color to use for the scrim that obscures primary content while a drawer is open. [...]

*final*

endDrawer → Widget

A panel displayed to the side of the body, often hidden on mobile devices. Swipes in from right-to-left (TextDirection.ltr) or left-to-right (TextDirection.rtl) [...]

*final*

extendBody → bool

If true, and bottomNavigationBar or persistentFooterButtons is specified, then the body extends to the bottom of the Scaffold, instead of only extending to the top of the bottomNavigationBar or the persistentFooterButtons. [...]

*final*

extendBodyBehindAppBar → bool

If true, and an appBar is specified, then the height of the body is extended to include the height of the app bar and the top of the body is aligned with the top of the app bar. [...]

*final*

floatingActionButton → Widget

A button displayed floating above body, in the bottom right corner. [...]

*final*

floatingActionButtonAnimator → FloatingActionButtonAnimator

Animator to move the floatingActionButton to a new floatingActionButtonLocation. [...]

*final*

floatingActionButtonLocation → FloatingActionButtonLocation

Responsible for determining where the floatingActionButton should go. [...]

*final*

persistentFooterButtons → List<Widget>

A set of buttons that are displayed at the bottom of the scaffold. [...]

*final*

primary → bool

Whether this scaffold is being displayed at the top of the screen. [...]

*final*

resizeToAvoidBottomInset → bool

If true the body and the scaffold's floating widgets should size themselves to avoid the onscreen keyboard whose height is defined by the ambient MediaQuery's MediaQueryData.viewInsets `bottom` property. [...]

*final*

# Text

## Properties

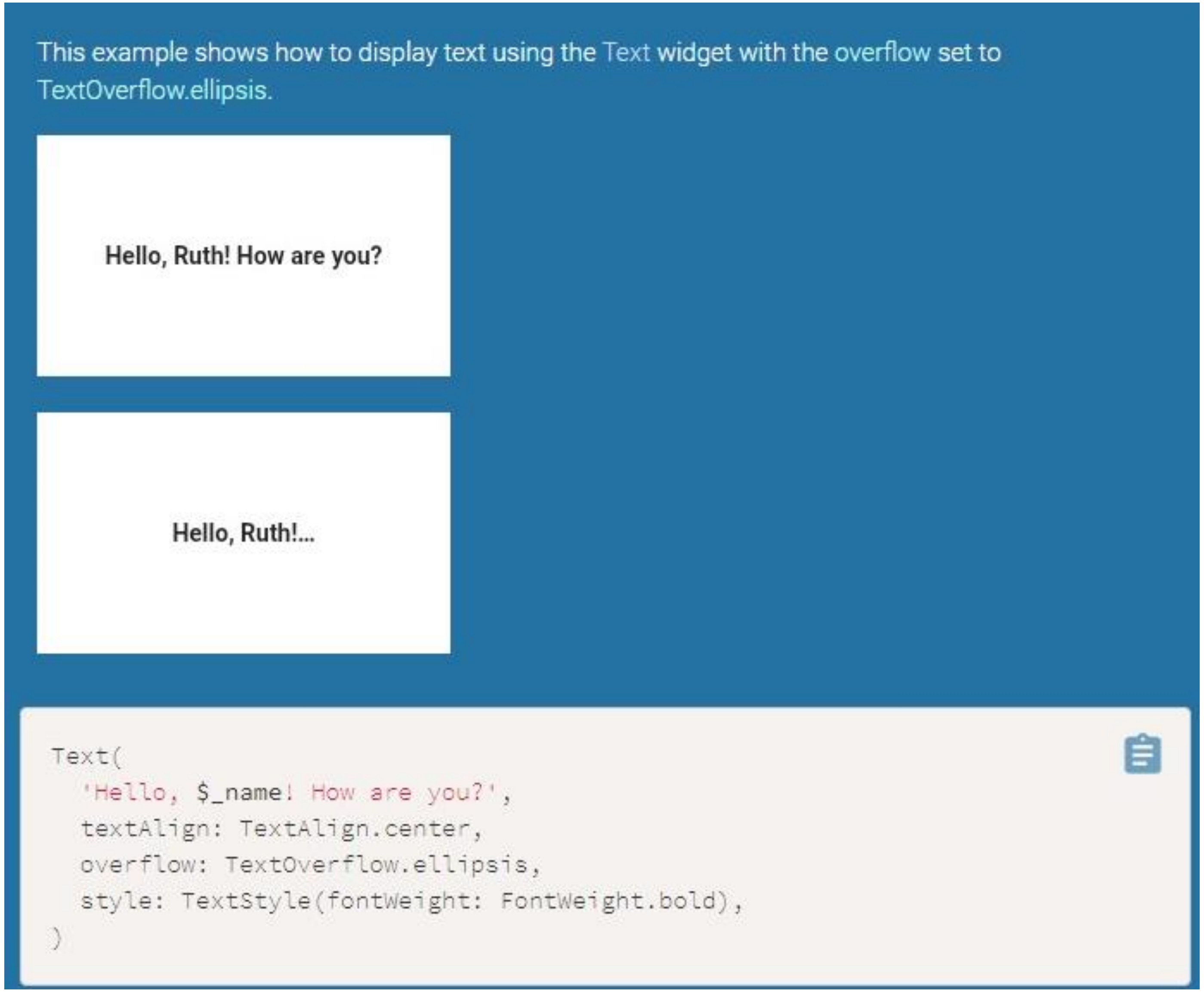
data → String
The text to display. [...]
*final*

locale → Locale
Used to select a font when the same Unicode character can be rendered differently, depending on the locale. [...]
*final*

maxLines → int
An optional maximum number of lines for the text to span, wrapping if necessary. If the text exceeds the given number of lines, it will be truncated according to overflow. [...]
*final*

overflow → TextOverflow
How visual overflow should be handled.
*final*

semanticsLabel → String
An alternative semantics label for this text. [...]
*final*



This example shows how to display text using the Text widget with the overflow set to TextOverflow.ellipsis.

Hello, Ruth! How are you?

Hello, Ruth!...

```
Text(
  'Hello, $_name! How are you?',
  textAlign: TextAlign.center,
  overflow: TextOverflow.ellipsis,
  style: TextStyle(fontWeight: FontWeight.bold),
)
```

softWrap → bool
Whether the text should break at soft line breaks. [...]
*final*
strutStyle → StrutStyle
The strut style to use. Strut style defines the strut, which sets minimum vertical layout metrics. [...]
*final*
style → TextStyle
If non-null, the style to use for this text. [...]
*final*
textAlign → TextAlign
How the text should be aligned horizontally.
*final*
textDirection → TextDirection
The directionality of the text. [...]
*final*
textScaleFactor → double
The number of font pixels for each logical pixel. [...]
*final*
textSpan → InlineSpan
The text to display as a InlineSpan. [...]
*final*
textWidthBasis → TextWidthBasis
Defines how to measure the width of the rendered text.
*final*