Flutter for web developers

Contents

This page is for users who are familiar with the HTML and CSS syntax for arranging components of an application's UI. It maps HTML/CSS code snippets to their Flutter/Dart code equivalents.

The examples assume:

- The HTML document starts with `<!DOCTYPE html>`, and the CSS box model for all HTML elements is set to `border-box`, for consistency with the Flutter model.

  content_copy
  ```
  {
    box-sizing: border-box;
  }
  ```

- In Flutter, the default styling of the “Lorem ipsum” text is defined by the `bold24Roboto` variable as follows, to keep the syntax simple:

  ```
  TextStyle bold24Roboto = TextStyle(
    color: Colors.white,
    fontSize: 24,
    fontWeight: FontWeight.w900,
  );
  ```

  content_copy

How is react-style, or *declarative*, programming different than the traditional imperative style? For a comparison, see Introduction to declarative UI.

Performing basic layout operations

The following examples show how to perform the most common UI layout tasks.

Styling and aligning text

Font style, size, and other text attributes that CSS handles with the font and color properties are individual properties of a `TextStyle` child of a `Text` widget.

In both HTML and Flutter, child elements or widgets are anchored at the top left, by default.

```
<div class="greybox">
    Lorem ipsum
</div>

.greybox {
      background-color: #e0e0e0; /* grey 300 */
      width: 320px;
      height: 240px;
      font: 900 24px Georgia;
    }
```
content_copy

```
var container = Container( // grey box
  child: Text(
    "Lorem ipsum",
    style: TextStyle(
      fontSize: 24,
      fontWeight: FontWeight.w900,
      fontFamily: "Georgia",
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```
content_copy

Setting background color

In Flutter, you set the background color using a `Container`'s `decoration` property.

The CSS examples use the hex color equivalents to the Material color palette.

```
<div class="greybox">
  Lorem ipsum
</div>

.greybox {
      background-color: #e0e0e0;  /* grey 300 */
      width: 320px;
      height: 240px;
      font: 900 24px Roboto;
    }
```
content_copy

```
var container = Container( // grey box
  child: Text(
    "Lorem ipsum",
    style: bold24Roboto,
  ),
  width: 320,
  height: 240,
  decoration: BoxDecoration(
    color: Colors.grey[300],
  ),
);
```

content_copy

Centering components

A `Center` widget centers its child both horizontally and vertically.

To accomplish a similar effect in CSS, the parent element uses either a flex or table-cell display behavior. The examples on this page show the flex behavior.

```
<div class="greybox">
  Lorem ipsum
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
```

content_copy

```
var container = Container( // grey box
  child: Center(
    child: Text(
      "Lorem ipsum",
      style: bold24Roboto,
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```
content_copy

Setting container width

To specify the width of a `Container` widget, use its `width` property. This is a fixed width, unlike the CSS max-width property that adjusts the container width up to a maximum value. To mimic that effect in Flutter, use the `constraints` property of the Container. Create a new `BoxConstraints` widget with a `minWidth` or `maxWidth`.

For nested Containers, if the parent's width is less than the child's width, the child Container sizes itself to match the parent.

content_copy

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
  width: 100%;
  max-width: 240px;
}
```

content_copy

```
var container = Container( // grey box
  child: Center(
    child: Container( // red box
      child: Text(
        "Lorem ipsum",
        style: bold24Roboto,
      ),
      decoration: BoxDecoration(
        color: Colors.red[400],
      ),
      padding: EdgeInsets.all(16),
      width: 240, //max-width is 240
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```

Manipulating position and size

The following examples show how to perform more complex operations on widget position, size, and background.

Setting absolute position

By default, widgets are positioned relative to their parent.

To specify an absolute position for a widget as x-y coordinates, nest it in a `Positioned` widget that is, in turn, nested in a `Stack` widget.

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  position: relative;
}
.redbox {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
  position: absolute;
  top: 24px;
  left: 24px;
}
```

```
var container = Container( // grey box
  child: Stack(
    children: [
      Positioned( // red box
        child:  Container(
          child: Text(
            "Lorem ipsum",
            style: bold24Roboto,
          ),
          decoration: BoxDecoration(
            color: Colors.red[400],
          ),
          padding: EdgeInsets.all(16),
        ),
        left: 24,
        top: 24,
      ),
    ],
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```

content_copy

Rotating components

To rotate a widget, nest it in a `Transform` widget. Use the `Transform` widget's `alignment` and `origin`properties to specify the transform origin (fulcrum) in relative and absolute terms, respectively.

For a simple 2D rotation, in which the widget is rotated on the Z axis, create a new `Matrix4` identity object and use its `rotateZ()` method to specify the rotation factor using radians (degrees × π / 180).

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
  transform: rotate(15deg);
}
```
content_copy

```
var container = Container( // gray box
  child: Center(
    child:  Transform(
      child:  Container( // red box
        child: Text(
          "Lorem ipsum",
          style: bold24Roboto,
          textAlign: TextAlign.center,
        ),
        decoration: BoxDecoration(
          color: Colors.red[400],
        ),
        padding: EdgeInsets.all(16),
      ),
      alignment: Alignment.center,
      transform: Matrix4.identity()
        ..rotateZ(15 * 3.1415927 / 180),
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```
content_copy

Scaling components

To scale a widget up or down, nest it in a [`Transform`] widget. Use the Transform widget's `alignment` and `origin` properties to specify the transform origin (fulcrum) in relative or absolute terms, respectively.

For a simple scaling operation along the x-axis, create a new [`Matrix4`] identity object and use its `scale()`method to specify the scaling factor.

When you scale a parent widget, its child widgets are scaled accordingly.

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
  transform: scale(1.5);
}
```

content_copy

```
var container = Container( // gray box
  child: Center(
    child:  Transform(
      child:  Container( // red box
        child: Text(
          "Lorem ipsum",
          style: bold24Roboto,
          textAlign: TextAlign.center,
        ),
        decoration: BoxDecoration(
          color: Colors.red[400],
        ),
        padding: EdgeInsets.all(16),
      ),
      alignment: Alignment.center,
      transform: Matrix4.identity()
        ..scale(1.5),
     ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```

content_copy

Applying a linear gradient

To apply a linear gradient to a widget's background, nest it in a `Container` widget. Then use the `Container`widget's `decoration` property to create a `BoxDecoration` object, and use `BoxDecoration`'s `gradient`property to transform the background fill.

The gradient "angle" is based on the Alignment (x, y) values:

- If the beginning and ending x values are equal, the gradient is vertical (0° | 180°).
- If the beginning and ending y values are equal, the gradient is horizontal (90° | 270°).

Vertical gradient

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  padding: 16px;
  color: #ffffff;
  background: linear-gradient(180deg, #ef5350, rgba(0, 0, 0, 0)
80%);
}
```

content_copy

```
var container = Container( // grey box
  child: Center(
    child: Container( // red box
      child: Text(
        "Lorem ipsum",
        style: bold24Roboto,
      ),
      decoration: BoxDecoration(
        gradient: LinearGradient(
          begin: const Alignment(0.0, -1.0),
          end: const Alignment(0.0, 0.6),
          colors: <Color>[
            const Color(0xffef5350),
            const Color(0x00ef5350)
          ],
        ),
      ),
      padding: EdgeInsets.all(16),
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```

Horizontal gradient

content_copy

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  padding: 16px;
  color: #ffffff;
  background: linear-gradient(90deg, #ef5350, rgba(0, 0, 0, 0)
80%);
}
```

```
var container = Container( // grey box
  child: Center(
    child: Container( // red box
      child: Text(
        "Lorem ipsum",
        style: bold24Roboto,
      ),
      decoration: BoxDecoration(
        gradient: LinearGradient(
          begin: const Alignment(-1.0, 0.0),
          end: const Alignment(0.6, 0.0),
          colors: <Color>[
            const Color(0xffef5350),
            const Color(0x00ef5350)
          ],
        ),
      ),
      padding: EdgeInsets.all(16),
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```

Manipulating shapes

The following examples show how to make and customize shapes.

Rounding corners

To round the corners of a rectangular shape, use the `borderRadius` property of a `BoxDecoration` object. Create a new `BorderRadius` object that specifies the radii for rounding each corner.

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* gray 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
  border-radius: 8px;
}
```

content_copy

```
var container = Container( // grey box
  child: Center(
    child: Container( // red circle
      child: Text(
        "Lorem ipsum",
        style: bold24Roboto,
      ),
      decoration: BoxDecoration(
        color: Colors.red[400],
        borderRadius: BorderRadius.all(
          const Radius.circular(8),
        ),
      ),
      padding: EdgeInsets.all(16),
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```

content_copy

Adding box shadows

In CSS you can specify shadow offset and blur in shorthand, using the box-shadow property. This example shows two box shadows, with properties:

- `xOffset: 0px, yOffset: 2px, blur: 4px, color: black @80% alpha`
- `xOffset: 0px, yOffset: 06x, blur: 20px, color: black @50% alpha`

In Flutter, each property and value is specified separately. Use the `boxShadow` property of `BoxDecoration`to create a list of [`BoxShadow`]() widgets. You can define one or multiple `BoxShadow` widgets, which can be stacked to customize the shadow depth, color, and so on.

content_copy

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.8),
              0 6px 20px rgba(0, 0, 0, 0.5);
}
```

```
var container = Container( // grey box
  child: Center(
    child: Container( // red box
      child: Text(
        "Lorem ipsum",
        style: bold24Roboto,
      ),
      decoration: BoxDecoration(
        color: Colors.red[400],
        boxShadow: [
          BoxShadow (
            color: const Color(0xcc000000),
            offset: Offset(0, 2),
            blurRadius: 4,
          ),
          BoxShadow (
            color: const Color(0x80000000),
            offset: Offset(0, 6),
            blurRadius: 20,
          ),
        ],
      ),
      padding: EdgeInsets.all(16),
    ),
  ),
  width: 320,
  height: 240,
  decoration: BoxDecoration(
    color: Colors.grey[300],
  ),
  margin: EdgeInsets.only(bottom: 16),
);
```

Making circles and ellipses

Making a circle in CSS requires a workaround of applying a border-radius of 50% to all four sides of a rectangle, though there are basic shapes.

While this approach is supported with the `borderRadius` property of `BoxDecoration`, Flutter provides a `shape` property with `BoxShape` enum for this purpose.

```
<div class="greybox">
  <div class="redcircle">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* gray 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redcircle {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
  text-align: center;
  width: 160px;
  height: 160px;
  border-radius: 50%;
}
```
content_copy

```
var container = Container( // grey box
  child: Center(
    child: Container( // red circle
      child: Text(
        "Lorem ipsum",
        style: bold24Roboto,
        textAlign: TextAlign.center,
      ),
      decoration: BoxDecoration(
        color: Colors.red[400],
        shape: BoxShape.circle,
      ),
      padding: EdgeInsets.all(16),
      width: 160,
      height: 160,
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```

Manipulating text

The following examples show how to specify fonts and other text attributes. They also show how to transform text strings, customize spacing, and create excerpts.

Adjusting text spacing

In CSS you specify the amount of white space between each letter or word by giving a length value for the letter-spacing and word-spacing properties, respectively. The amount of space can be in px, pt, cm, em, etc.

In Flutter, you specify white space as logical pixels (negative values are allowed) for the `letterSpacing`and `wordSpacing` properties of a [`TextStyle`] child of a `Text` widget.

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
  letter-spacing: 4px;
}
```

content_copy

```
var container = Container( // grey box
  child: Center(
    child: Container( // red box
      child: Text(
        "Lorem ipsum",
        style: TextStyle(
          color: Colors.white,
          fontSize: 24,
          fontWeight: FontWeight.w900,
          letterSpacing: 4,
        ),
      ),
      decoration: BoxDecoration(
        color: Colors.red[400],
      ),
      padding: EdgeInsets.all(16),
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```

content_copy

Making inline formatting changes

A `Text` widget lets you display text with some formatting characteristics. To display text that uses multiple styles (in this example, a single word with emphasis), use a `RichText` widget instead. Its `text` property can specify one or more `TextSpan` widgets that can be individually styled.

In the following example, "Lorem" is in a `TextSpan` widget with the default (inherited) text styling, and "ipsum" is in a separate `TextSpan` with custom styling.

content_copy

```
<div class="greybox">
  <div class="redbox">
    Lorem <em>ipsum</em>
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
}
 .redbox em {
  font: 300 48px Roboto;
  font-style: italic;
}
```

content_copy

```
var container = Container( // grey box
  child: Center(
    child: Container( // red box
      child:  RichText(
        text: TextSpan(
          style: bold24Roboto,
          children: <TextSpan>[
            TextSpan(text: "Lorem "),
            TextSpan(
              text: "ipsum",
              style: TextStyle(
                fontWeight: FontWeight.w300,
                fontStyle: FontStyle.italic,
                fontSize: 48,
              ),
            ),
          ],
        ),
      ),
      decoration: BoxDecoration(
        color: Colors.red[400],
      ),
      padding: EdgeInsets.all(16),
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```

Creating text excerpts

An excerpt displays the initial line(s) of text in a paragraph, and handles the overflow text, often using an ellipsis. In HTML/CSS an excerpt can be no longer than one line. Truncating after multiple lines requires some JavaScript code.

In Flutter, use the `maxLines` property of a `Text` widget to specify the number of lines to include in the excerpt, and the `overflow` property for handling overflow text.

content_copy

```
<div class="greybox">
  <div class="redbox">
    Lorem ipsum dolor sit amet, consec etur
  </div>
</div>

.greybox {
  background-color: #e0e0e0; /* grey 300 */
  width: 320px;
  height: 240px;
  font: 900 24px Roboto;
  display: flex;
  align-items: center;
  justify-content: center;
}
.redbox {
  background-color: #ef5350; /* red 400 */
  padding: 16px;
  color: #ffffff;
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
}
```

content_copy

```
var container = Container( // grey box
  child: Center(
    child: Container( // red box
      child: Text(
        "Lorem ipsum dolor sit amet, consec etur",
        style: bold24Roboto,
        overflow: TextOverflow.ellipsis,
        maxLines: 1,
      ),
      decoration: BoxDecoration(
        color: Colors.red[400],
      ),
      padding: EdgeInsets.all(16),
    ),
  ),
  width: 320,
  height: 240,
  color: Colors.grey[300],
);
```