Layout Widgets


Layout Widgets
Single-child
Align
AspectRatio
Baseline
Center
ConstrainedBox
Container
Padding
SizedBox
Transform
CustomSingleChildLayout
Expanded
FittedBox
FractionallySizedBox
IntrinsicHeight & Width
LimitedBox
Offstage
OverflowBox
SizedOverflowBox
Multi-child
Column
CustomMultiChildLayout
Flow
GridView
IndexedStack
LayoutBuilder
ListBody
ListView
Row
Stack
Table
Wrap

Align

Constructors

Align({Key key, AlignmentGeometry alignment: Alignment.center, double widthFactor, double heightFactor, Widget child})

Creates an alignment widget. [...]

Properties

alignment → AlignmentGeometry

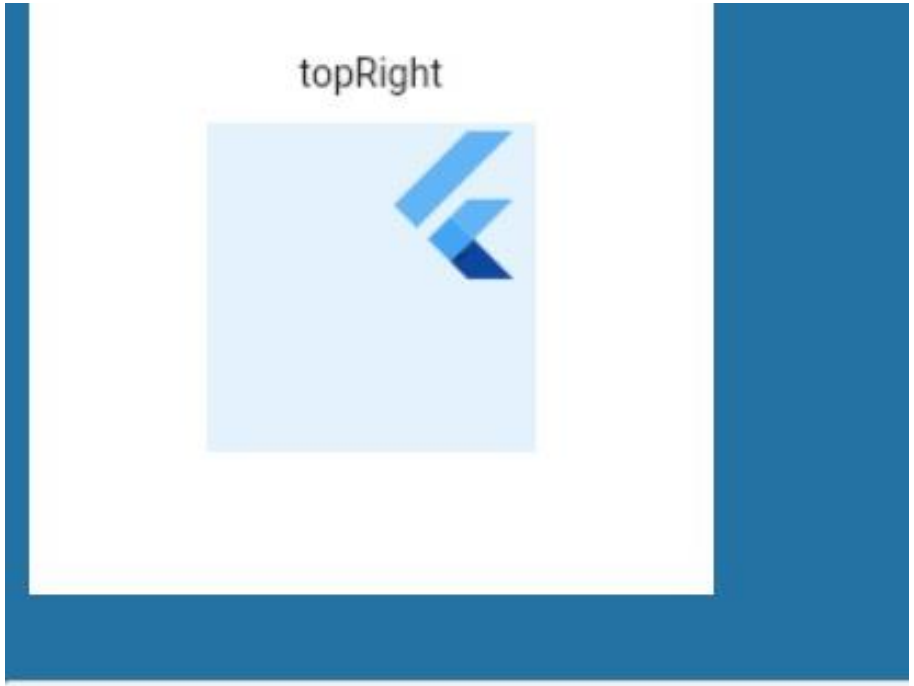
How to align the child. [...]

heightFactor → double

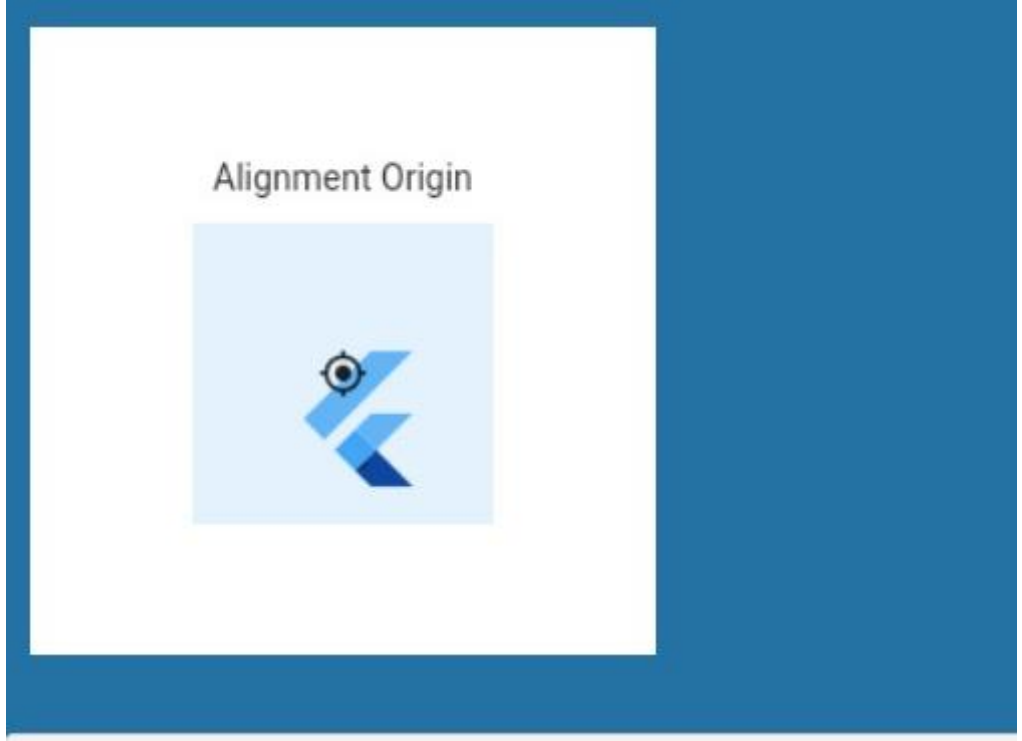
If non-null, sets its height to the child's height multiplied by this factor. [...]

widthFactor → double

If non-null, sets its width to the child's width multiplied by this factor. [...]




```
Center(
  child: Container(
    height: 120.0,
    width: 120.0,
    color: Colors.blue[50],
    child: Align(
      alignment: Alignment.topRight,
      child: FlutterLogo(
        size: 60,
      ),
    ),
  ),
```




```
Center(
  child: Container(
    height: 120.0,
    width: 120.0,
    color: Colors.blue[50],
    child: Align(
      alignment: Alignment(0.2, 0.6),
      child: FlutterLogo(
        size: 60,
      ),
    ),
  ),
```

Flutter

AspectRatio

Constructors

AspectRatio({Key key, @required double aspectRatio, Widget child})

Creates a widget with a specific aspect ratio. [...]

Properties

aspectRatio → double

The aspect ratio to attempt to use. [...]

Flutter

Baseline

A widget that positions its child according to the child's baseline.

Constructors

Baseline({Key key, @required double baseline, @required TextBaseline baselineType, Widget child})
Creates a widget that positions its child according to the child's baseline. [...]

Properties

baseline → double
The number of logical pixels from the top of this box at which to position the child's baseline.

baselineType → TextBaseline
The type of baseline to use for positioning the child.

ConstrainedBox

```
ConstrainedBox(
  constraints: const BoxConstraints.expand(),
  child: const Card(child: Text('Hello World!')),
)
```

Constructors

ConstrainedBox({Key key, @required BoxConstraints constraints, Widget child})

Creates a widget that imposes additional constraints on its child. [...]

Properties

constraints → BoxConstraints

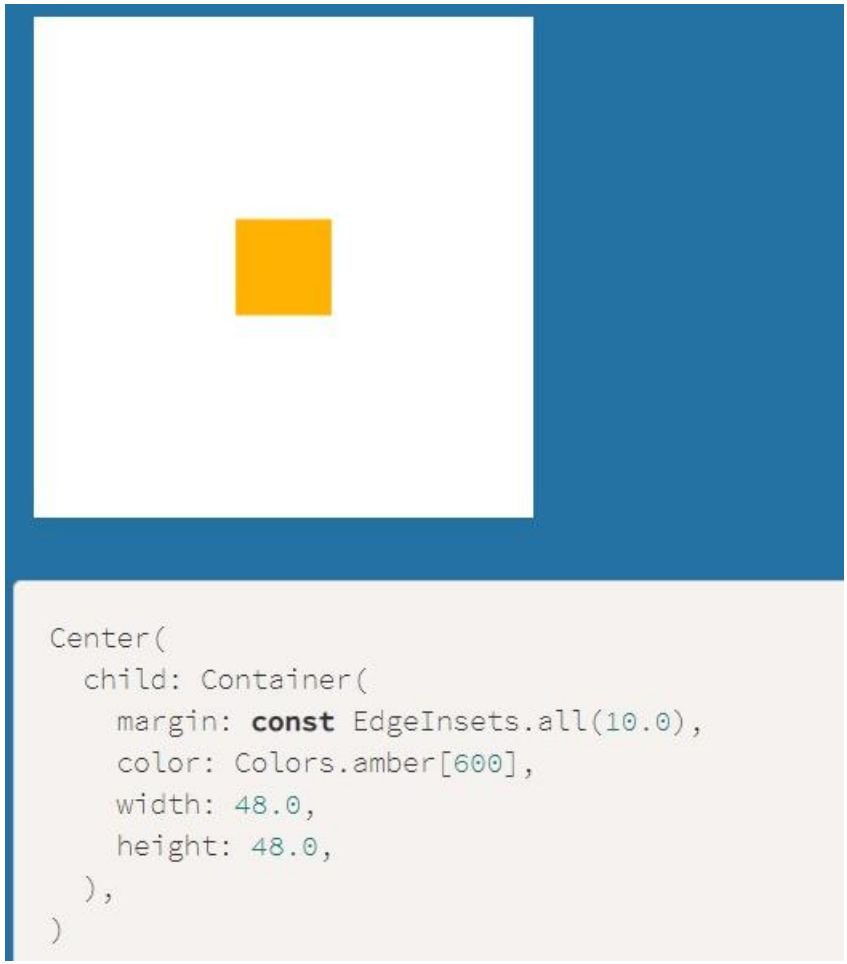
The additional constraints to impose on the child.

Flutter

Container

Constructors

Container({Key key, AlignmentGeometry alignment, EdgeInsetsGeometry padding, Color color, Decoration decoration, Decoration foregroundDecoration, double width, double height, BoxConstraints constraints, EdgeInsetsGeometry margin, Matrix4 transform, Widget child})



Properties

alignment → AlignmentGeometry. Align the child within the container. [...]
child → Widget. The child contained by the container. [...]
constraints → BoxConstraints. Additional constraints to apply to the child. [...]
decoration → Decoration. The decoration to paint behind the child. [...]
foregroundDecoration → Decoration. The decoration to paint in front of the child.
margin → EdgeInsetsGeometry. Empty space to surround the decoration and child.
padding → EdgeInsetsGeometry. Empty space to inscribe inside the decoration. The child, if any, is placed inside
transform → Matrix4. The transformation matrix to apply before painting the container.

Flutter

FittedBox

Constructors

FittedBox({Key key, BoxFit fit: BoxFit.contain, AlignmentGeometry alignment: Alignment.center, Widget child})

Creates a widget that scales and positions its child within itself according to fit. [...]

Properties

alignment → AlignmentGeometry

How to align the child within its parent's bounds. [...]

fit → BoxFit

How to inscribe the child into the space allocated during layout.

Flutter

Expanded

Constructors

Expanded({Key key, int flex: 1, @required Widget child})

Creates a widget that expands a child of a Row, Column, or Flex so that the child fills the available space along the flex widget's main axis.

const

Properties

child → Widget

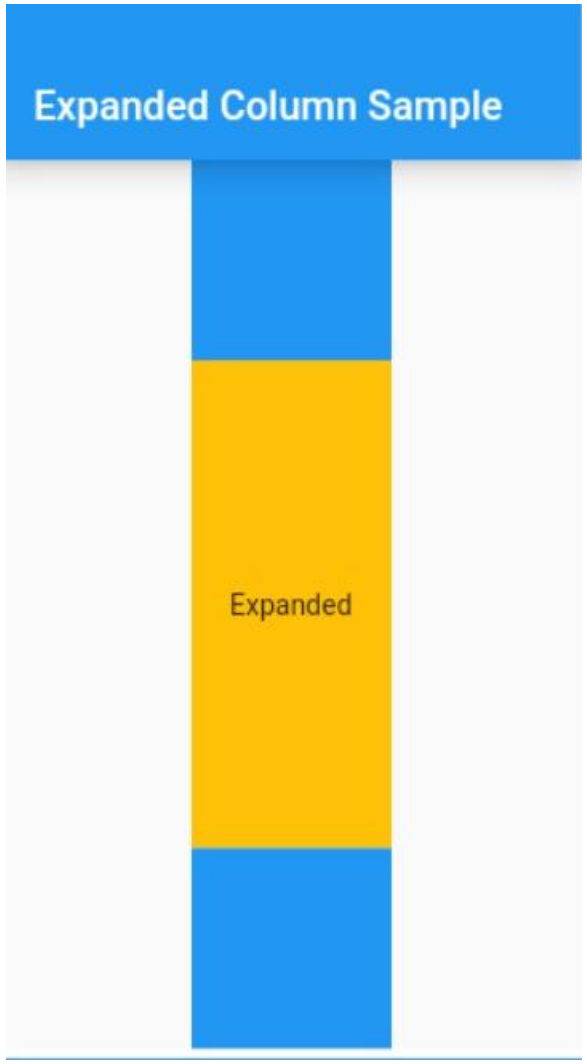
The widget below this widget in the tree. [...]

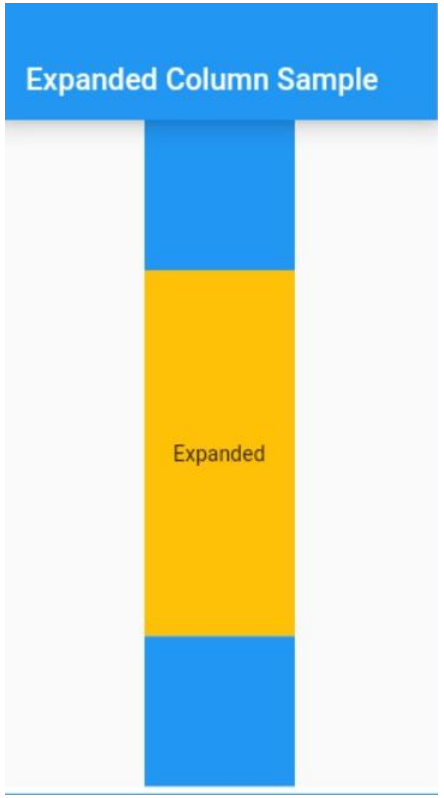
fit → FlexFit

How a flexible child is inscribed into the available space. [...]

flex → int

The flex factor to use for this child [...]






```
Widget build(BuildContext context) {
 return Scaffold(
    appBar: AppBar(
     title: Text('Expanded Column Sample’),
        ),
     body: Center(
         child: Column(
            children: <Widget>[
             Container(
                 color: Colors.blue,
                 height: 100,
                 width: 100,
             ),
             Expanded(
                 child: Container(
                     color: Colors.amber,
                     width: 100,
                     ),
             ),
             Container(
                color: Colors.blue,
                height: 100,
                width: 100,
             ),
            ]
```

Flutter

CustomSingleChildLayout

A widget that defers the layout of its single child to a delegate.

Constructors

CustomSingleChildLayout({Key key, @required SingleChildLayoutDelegate delegate, Widget child})

Creates a custom single child layout. [...]

Properties

delegate → SingleChildLayoutDelegate

The delegate that controls the layout of the child.

```
class _MySingleChildLayoutDelegate extends SingleChildLayoutDelegate {
  _MySingleChildLayoutDelegate({@required this.widgetSize});

  final Size widgetSize;

  @override
  BoxConstraints getConstraintsForChild(BoxConstraints constraints) {
    //we expand the layout to our predefined sizes
    return BoxConstraints.expand(width: 120.0, height: 120.0);
  }

  @override
  Offset getPositionForChild(Size size, Size childSize) {
    return Offset(widgetSize.width / 4, widgetSize.height / 4);
  }

  @override
  bool shouldRelayout(_MySingleChildLayoutDelegate oldDelegate) {
    return widgetSize != oldDelegate.widgetSize;
  }
}
```

FractionallySizedBox

Constructors

FractionallySizedBox({Key key, AlignmentGeometry alignment: Alignment.center, double widthFactor, double heightFactor, Widget child})

Creates a widget that sizes its child to a fraction of the total available space. [...]

Properties

alignment → AlignmentGeometry

How to align the child. [...]

heightFactor → double

If non-null, the fraction of the incoming height given to the child. [...]

widthFactor → double

If non-null, the fraction of the incoming width given to the child. [...]

Flutter

IntrinsicHeight/IntrinsicWidth

A widget that sizes its child to the child's intrinsic height.

A widget that sizes its child to the child's intrinsic width.

```
113        child:Column(
114          children: <Widget>[
115            IntrinsicHeight(
116              child: Row(
117                children: <Widget>[
118                  Text('hey'),
119                  VerticalDivider(
120                    color: Colors.black,
121                    width: 100,), // VerticalDivider
122                  Text('Dude',style: TextStyle(fontSize: 80),)
123                ], // <Widget>[]
124              ), // Row
125            ), // IntrinsicHeight
126            IntrinsicWidth(
127              stepWidth: 8,
128              stepHeight: 0.1,
129              child: Row(
130                children: <Widget>[
131                  Text('hey'),
132                  VerticalDivider(
133                    color: Colors.black,
134                    width: 100,), // VerticalDivider
135                  Text('Dude',style: TextStyle(fontSize:20),)
136                ], // <Widget>[]
137              ), // Row
138            ), // IntrinsicWidth
139          ], // <Widget>[]
140        ), // Column
141      ), // Center
```

LimitedBox

Constructors

LimitedBox({Key key, double maxWidth: double.infinity, double maxHeight: double.infinity, Widget child})

Creates a box that limits its size only when it's unconstrained. [...]

Properties

maxHeight → double

The maximum height limit to apply in the absence of a BoxConstraints.maxHeight constraint.

maxWidth → double

The maximum width limit to apply in the absence of a BoxConstraints.maxWidth constraint.

Flutter

Offstage

A widget that lays the child out as if it was in the tree, but without painting anything, without making the child available for hit testing, and without taking any room in the parent.

Animations continue to run in offstage children, and therefore use battery and CPU time, regardless of whether the animations end up being visible.

Offstage can be used to measure the dimensions of a widget without bringing it on screen (yet). To hide a widget from view while it is not needed, prefer removing the widget from the tree entirely rather than keeping it alive in an Offstage subtree.

Constructors

Offstage({Key key, bool offstage: true, Widget child})

Creates a widget that visually hides its child.

const

Properties

offstage → bool

Whether the child is hidden from the rest of the tree. [...]

final

OverflowBox

A widget that imposes different constraints on its child than it gets from its parent, possibly allowing the child to overflow the parent.

Constructors

OverflowBox({Key key, AlignmentGeometry alignment: Alignment.center, double minWidth, double maxWidth, double minHeight, double maxHeight, Widget child})

Creates a widget that lets its child overflow itself.

Properties

alignment → AlignmentGeometry

How to align the child. [...]

maxHeight → double

The maximum height constraint to give the child. Set this to null (the default) to use the constraint from the parent instead.

maxWidth → double

The maximum width constraint to give the child. Set this to null (the default) to use the constraint from the parent instead.

minHeight → double

The minimum height constraint to give the child. Set this to null (the default) to use the constraint from the parent instead.

minWidth → double

The minimum width constraint to give the child. Set this to null (the default) to use the constraint from the parent instead.

Padding

Constructors

Padding({Key key, @required EdgeInsetsGeometry padding, Widget child})

Creates a widget that insets its child. [...]

Properties

padding → EdgeInsetsGeometry

The amount of space by which to inset the child.

EdgeInsets

Constructors

EdgeInsets.all(double value)
Creates insets where all the offsets are value. [...]

EdgeInsets.fromLTRB(double left, double top, double right, double bottom)
Creates insets from offsets from the left, top, right, and bottom.

EdgeInsets.fromWindowPadding(WindowPadding padding, double devicePixelRatio)
Creates insets that match the given window padding. [...]
EdgeInsets.only({double left: 0.0, double top: 0.0, double right: 0.0, double bottom: 0.0})
Creates insets with only the given values non-zero. [...]

EdgeInsets.symmetric({double vertical: 0.0, double horizontal: 0.0})
Creates insets with symmetrical vertical and horizontal offsets. [...]

SizedBox

```
SizedBox(
  width: 200.0,
  height: 300.0,
  child: const Card(child: Text('Hello World!')),
)
```

A box with a specified size.

Constructors

SizedBox({Key key, double width, double height, Widget child})

Creates a fixed size box. The width and height parameters can be null to indicate that the size of the box should not be constrained in the corresponding dimension.

SizedBox.expand({Key key, Widget child})

Creates a box that will become as large as its parent allows.

SizedBox.fromSize({Key key, Widget child, Size size})

Creates a box with the specified size.

SizedBox.shrink({Key key, Widget child})

Creates a box that will become as small as its parent allows.

Flutter

SizedOverflowBox

A widget that is a specific size but passes its original constraints through to its child, which may then overflow.

Constructors

SizedOverflowBox({Key key, @required Size size, AlignmentGeometry alignment: Alignment.center, Widget child})

Creates a widget of a given size that lets its child overflow. [...]

Properties

alignment → AlignmentGeometry

How to align the child. [...]

size → Size

The size this widget should attempt to be.

Transform

Constructors

Transform({Key key, @required Matrix4 transform, Offset origin, AlignmentGeometry alignment, bool transformHitTests: true, Widget child})

Creates a widget that transforms its child. [...]

Transform.rotate({Key key, @required double angle, Offset origin, AlignmentGeometry alignment: Alignment.center, bool transformHitTests: true, Widget child})

Creates a widget that transforms its child using a rotation around the center. [...]

Transform.scale({Key key, @required double scale, Offset origin, AlignmentGeometry alignment: Alignment.center, bool transformHitTests: true, Widget child})

Creates a widget that scales its child uniformly. [...]

Transform.translate({Key key, @required Offset offset, bool transformHitTests: true, Widget child})

Creates a widget that transforms its child using a translation. [...]

Properties

alignment → AlignmentGeometry

The alignment of the origin, relative to the size of the box. [...]

origin → Offset

The origin of the coordinate system (relative to the upper left corder of this render object) in which to apply the matrix. [...]

transform → Matrix4

The matrix to transform the child by during painting.

transformHitTests → bool

Whether to apply the transformation when performing hit tests.

Flutter

Transform power

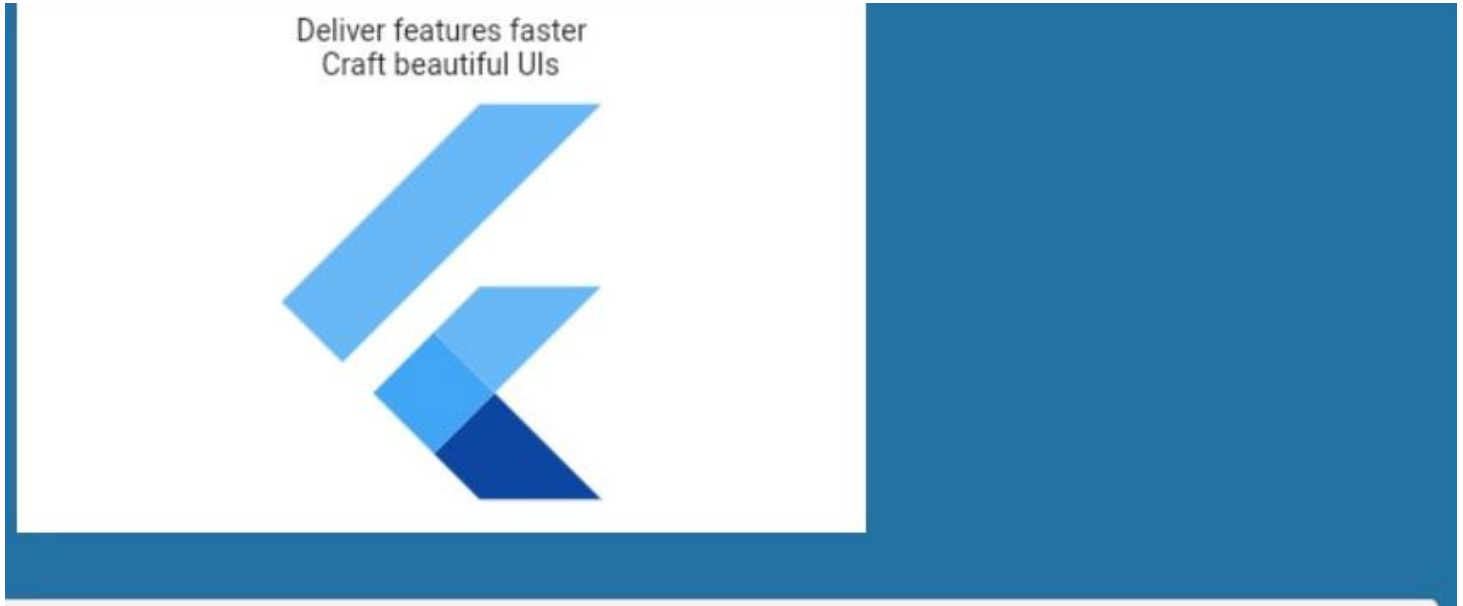
```
class ImageRotate extends StatefulWidget {
  @override
  _MetalHead createState() => new _MetalHead();
}

class _MetalHead extends State<ImageRotate>
    with SingleTickerProviderStateMixin {
  AnimationController animationController;

  @override
  void initState() {
    super.initState();
    animationController = new AnimationController(
      vsync: this,
      duration: new Duration(seconds: 4),
    ); // AnimationController
    animationController.repeat();
  }
```

```
@override
Widget build(BuildContext context) {
  return Stack(
    alignment: Alignment.center,
    children:  <Widget>[
    new Container(
      color: Colors.white,
      width: 400,
      height: 400,
      alignment: Alignment.center,
      child:Padding(
        padding: EdgeInsets.fromLTRB(0, 120, 0, 0),
        child: new Image.asset('images/body.png')
      ) // Padding
    ), // Container
    new Container(
      alignment: Alignment.center,
      child: new AnimatedBuilder(
        animation: animationController,
        child: new Container(
          height: 150.0,
          width: 150.0,
          child: new Image.asset('images/head.png'),
        ), // Container
        builder: (BuildContext context, Widget _widget)
          return new Transform.rotate(
            angle: animationController.value * 242,
            child: _widget,
          ); // Transform.rotate
```

Column




```
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(),
      ),
    ),
  ],
)
```

Constructors

Column({Key key,
 MainAxisAlignment mainAxisAlignment: MainAxisAlignment.start,
MainAxisSize mainAxisSize: MainAxisSize.max,
CrossAxisAlignment crossAxisAlignment: CrossAxisAlignment.center,
TextDirection textDirection,
 VerticalDirection verticalDirection: VerticalDirection.down,
TextBaseline textBaseline,
 List<Widget> children: const []})

Creates a vertical array of children. [...]

CustomMultiChildLayout

A widget that uses a delegate to size and position multiple children.

Constructors

CustomMultiChildLayout({Key key, @required MultiChildLayoutDelegate delegate, List<Widget> children: const []})

Creates a custom multi-child layout. [...]

Flow

A widget that sizes and positions children efficiently, according to the logic in a FlowDelegate.
Flow layouts are optimized for repositioning children using transformation matrices.

Constructors

Flow({Key key, @required FlowDelegate delegate, List<Widget> children: const []})

Creates a flow layout. [...]

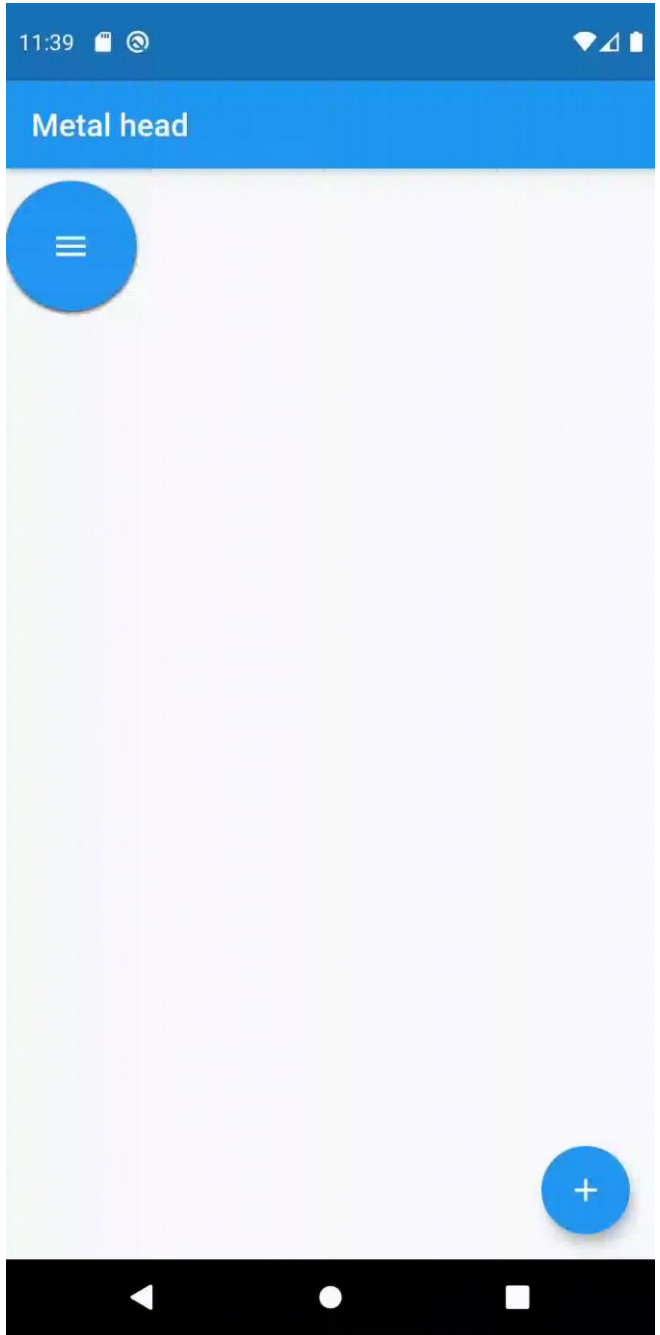
Flow.unwrapped({Key key, @required FlowDelegate delegate, List<Widget> children: const []})

Creates a flow layout. [...]

Properties

delegate → FlowDelegate

The delegate that controls the transformation matrices of the children.



```
class FlowMenu extends StatefulWidget {
  @override
  _FlowMenuState createState() => _FlowMenuState();
}

class _FlowMenuState extends State<FlowMenu> with SingleTickerProviderStateMixin {
  AnimationController menuAnimation;
  IconData lastTapped = Icons.notifications;
  final List<IconData> menuItems = <IconData>[
    Icons.home,
    Icons.new_releases,
    Icons.notifications,
    Icons.settings,
    Icons.menu,

  ];

  void _updateMenu(IconData icon) {
    if (icon != Icons.menu)
      setState(() => lastTapped = icon);
  }
```

```
@override
void initState() {
  super.initState();
  menuAnimation = AnimationController(
    duration: const Duration(milliseconds: 250),
    vsync: this,
  );
}
@override
Widget build(BuildContext context) {
  return Container(
    child: Flow(
      delegate: FlowMenuDelegate(menuAnimation: menuAnimation),
      children: menuItems.map<Widget>((IconData icon) => flowMenuItem(icon)).toList(),
    ),
  );
}
```

```
Widget flowMenuItem(IconData icon) {
  final double buttonDiameter = MediaQuery.of(context).size.width / menuItems.length;
  return Padding(
    padding: const EdgeInsets.symmetric(vertical: 8.0),
    child: RawMaterialButton(
      fillColor: lastTapped == icon ? Colors.amber[700] : Colors.blue,
      splashColor: Colors.amber[100],
      shape: CircleBorder(),
      constraints: BoxConstraints.tight(Size(buttonDiameter, buttonDiameter)),
      onPressed: () {
        _updateMenu(icon);
        menuAnimation.status == AnimationStatus.completed
            ? menuAnimation.reverse()
            : menuAnimation.forward();
      },
      child: Icon(
        icon,
        color: Colors.white,
        size: 25.0,
      ),
    ),
  );
}
```

```
class FlowMenuDelegate extends FlowDelegate {
  FlowMenuDelegate({this.menuAnimation}) : super(repaint: menuAnimation);

  final Animation<double> menuAnimation;

  @override
  bool shouldRepaint(FlowMenuDelegate oldDelegate) {
    return menuAnimation != oldDelegate.menuAnimation;
  }

  @override
  void paintChildren(FlowPaintingContext context) {
    double dx = 0.0;
    for (int i = 0; i < context.childCount; ++i) {
      dx = context.getChildSize(i).width * i;
      context.paintChild(
        i,
        transform: Matrix4.translationValues(
          dx * menuAnimation.value,
          0,
          0,
        ),
      );
```

GridView

A scrollable, 2D array of widgets.

Constructors

GridView({Key key, Axis scrollDirection: Axis.vertical, bool reverse: false, ScrollController controller, bool primary, ScrollPhysics physics, bool shrinkWrap: false, EdgeInsetsGeometry padding, @required SliverGridDelegate gridDelegate, bool addAutomaticKeepAlives: true, bool addRepaintBoundaries: true, bool addSemanticIndexes: true, double cacheExtent, List<Widget> children: const [], int semanticChildCount})
Creates a scrollable, 2D array of widgets with a custom SliverGridDelegate. [...]

GridView.builder({Key key, Axis scrollDirection: Axis.vertical, bool reverse: false, ScrollController controller, bool primary, ScrollPhysics physics, bool shrinkWrap: false, EdgeInsetsGeometry padding, @required SliverGridDelegate gridDelegate, @required IndexedWidgetBuilder itemBuilder, int itemCount, bool addAutomaticKeepAlives: true, bool addRepaintBoundaries: true, bool addSemanticIndexes: true, double cacheExtent, int semanticChildCount})
Creates a scrollable, 2D array of widgets that are created on demand. [...]

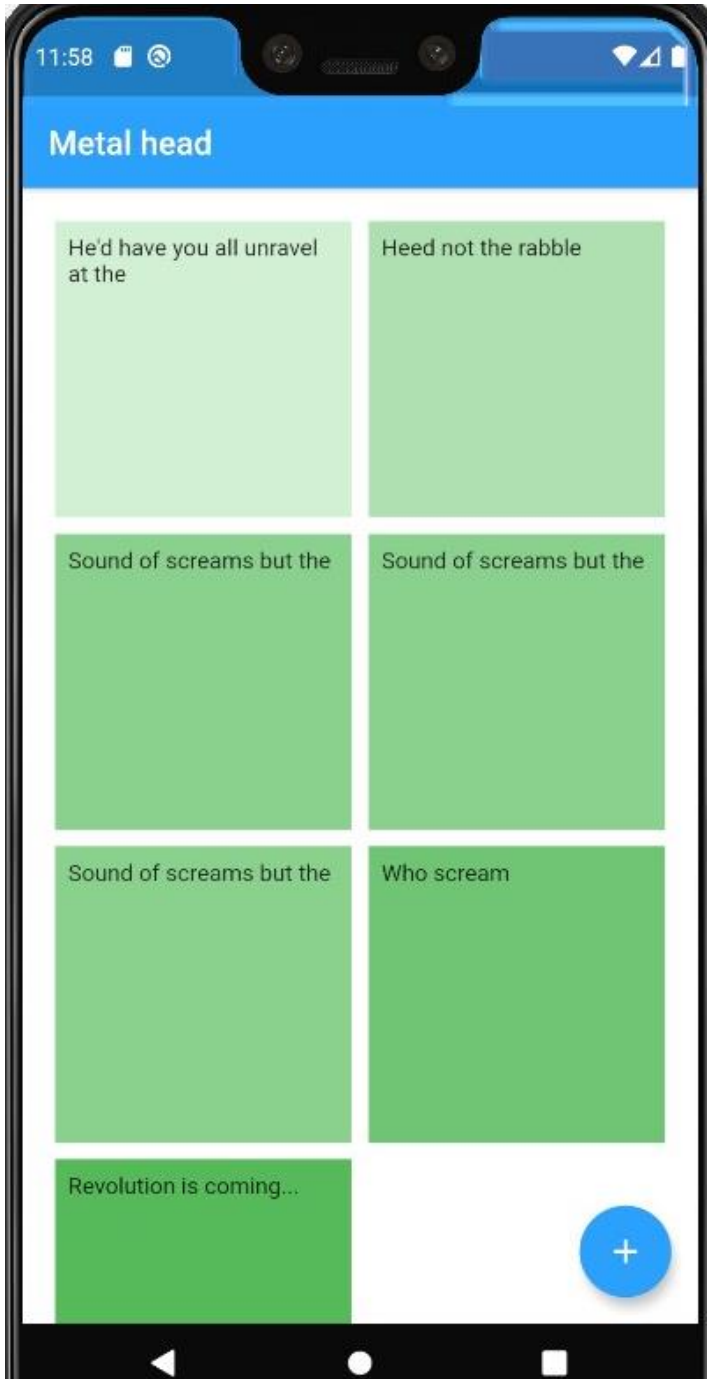
GridView.count({Key key, Axis scrollDirection: Axis.vertical, bool reverse: false, ScrollController controller, bool primary, ScrollPhysics physics, bool shrinkWrap: false, EdgeInsetsGeometry padding, @required int crossAxisCount, double mainAxisSpacing: 0.0, double crossAxisSpacing: 0.0, double childAspectRatio: 1.0, bool addAutomaticKeepAlives: true, bool addRepaintBoundaries: true, bool addSemanticIndexes: true, double cacheExtent, List<Widget> children: const [], int semanticChildCount, DragStartBehavior dragStartBehavior: DragStartBehavior.start})
Creates a scrollable, 2D array of widgets with a fixed number of tiles in the cross axis. [...]

GridView.custom({Key key, Axis scrollDirection: Axis.vertical, bool reverse: false, ScrollController controller, bool primary, ScrollPhysics physics, bool shrinkWrap: false, EdgeInsetsGeometry padding, @required SliverGridDelegate gridDelegate, @required SliverChildDelegate childrenDelegate, double cacheExtent, int semanticChildCount, DragStartBehavior dragStartBehavior: DragStartBehavior.start})
Creates a scrollable, 2D array of widgets with both a custom SliverGridDelegate and a custom SliverChildDelegate. [...]

GridView.extent({Key key, Axis scrollDirection: Axis.vertical, bool reverse: false, ScrollController controller, bool primary, ScrollPhysics physics, bool shrinkWrap: false, EdgeInsetsGeometry padding, @required double maxCrossAxisExtent, double mainAxisSpacing: 0.0, double crossAxisSpacing: 0.0, double childAspectRatio: 1.0, bool addAutomaticKeepAlives: true, bool addRepaintBoundaries: true, bool addSemanticIndexes: true, List<Widget> children: const [], int semanticChildCount, DragStartBehavior dragStartBehavior: DragStartBehavior.start})
Creates a scrollable, 2D array of widgets with tiles that each have a maximum cross-axis extent. [...]

```
Widget _simpleGridView(){
  return  CustomScrollView(
    primary: false,
    slivers: <Widget>[
      SliverPadding(
        padding: const EdgeInsets.all(20),
        sliver: SliverGrid.count(
          crossAxisSpacing: 10,
          mainAxisSpacing: 10,
          crossAxisCount: 2,
          children: <Widget>[
            Container(...), // Container
            Container(...), // Container
            Container(...), // Container
            Container(...), // Container
            Container(...), // Container
            Container(...), // Container
            Container(...), // Container

          ], // <Widget>[]
        ), // SliverGrid.count
      ), // SliverPadding
    ], // <Widget>[]
  ); // CustomScrollView
}
```



IndexedStack

A Stack that shows a single child from a list of children.

Constructors

```
IndexedStack({
Key key,
AlignmentGeometry alignment: AlignmentDirectional.topStart,
TextDirection textDirection,
StackFit sizing: StackFit.loose,
int index: 0,
List<Widget> children: const []})
```

Creates a Stack widget that paints a single child. [...]

Properties

index → int

The index of the child to show.

Flutter

LayoutBuilder

Builds a widget tree that can depend on the parent widget's size

Constructors

LayoutBuilder({Key key, LayoutWidgetBuilder builder})

Creates a widget that defers its building until layout. [...]

const

Properties

builder → LayoutWidgetBuilder

Called at layout time to construct the widget tree. [...]

Flutter

ListView

A scrollable list of widgets arranged linearly.


```
final List<String> entries = <String>['A', 'B', 'C'];
final List<int> colorCodes = <int>[600, 500, 100];

ListView.builder(
  padding: const EdgeInsets.all(8),
  itemCount: entries.length,
  itemBuilder: (BuildContext context, int index) {
    return Container(
      height: 50,
      color: Colors.amber[colorCodes[index]],
      child: Center(child: Text('Entry ${entries[index]}')),
    );
  }
);
```

Constructors

ListView({Key key, Axis scrollDirection: Axis.vertical, bool reverse: false, ScrollController controller, bool primary, ScrollPhysics physics, bool shrinkWrap: false, EdgeInsetsGeometry padding, double itemExtent, bool addAutomaticKeepAlives: true, bool addRepaintBoundaries: true, bool addSemanticIndexes: true, double cacheExtent, List<Widget> children: const [], int semanticChildCount, DragStartBehavior dragStartBehavior: DragStartBehavior.start})

Creates a scrollable, linear array of widgets from an explicit List. [...]

ListView.builder({Key key, Axis scrollDirection: Axis.vertical, bool reverse: false, ScrollController controller, bool primary, ScrollPhysics physics, bool shrinkWrap: false, EdgeInsetsGeometry padding, double itemExtent, @required IndexedWidgetBuilder itemBuilder, int itemCount, bool addAutomaticKeepAlives: true, bool addRepaintBoundaries: true, bool addSemanticIndexes: true, double cacheExtent, int semanticChildCount, DragStartBehavior dragStartBehavior: DragStartBehavior.start})

Creates a scrollable, linear array of widgets that are created on demand. [...]

ListView.custom({Key key, Axis scrollDirection: Axis.vertical, bool reverse: false, ScrollController controller, bool primary, ScrollPhysics physics, bool shrinkWrap: false, EdgeInsetsGeometry padding, double itemExtent, @required SliverChildDelegate childrenDelegate, double cacheExtent, int semanticChildCount})

Creates a scrollable, linear array of widgets with a custom child model. [...]

ListView.separated({Key key, Axis scrollDirection: Axis.vertical, bool reverse: false, ScrollController controller, bool primary, ScrollPhysics physics, bool shrinkWrap: false, EdgeInsetsGeometry padding, @required IndexedWidgetBuilder itemBuilder, @required IndexedWidgetBuilder separatorBuilder, @required int itemCount, bool addAutomaticKeepAlives: true, bool addRepaintBoundaries: true, bool addSemanticIndexes: true, double cacheExtent})

Creates a fixed-length scrollable linear array of list "items" separated by list item "separators". [...]

Flutter

Row

A widget that displays its children in a horizontal array.


```
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(),
      ),
    ),
  ],
)
```

Constructors

```
Row({
Key key,
MainAxisAlignment mainAxisAlignment: MainAxisAlignment.start,
MainAxisSize mainAxisSize: MainAxisSize.max,
CrossAxisAlignment crossAxisAlignment: CrossAxisAlignment.center,
TextDirection textDirection,
VerticalDirection verticalDirection: VerticalDirection.down,
TextBaseline textBaseline,
List<Widget> children: const []})
```

Creates a horizontal array of children. [...]

Stack

A widget that positions its children relative to the edges of its box.

Constructors
Stack({Key key,
 AlignmentGeometry alignment: AlignmentDirectional.topStart,
TextDirection textDirection,
StackFit fit: StackFit.loose,
Overflow overflow: Overflow.clip,
List<Widget> children: const []})
Creates a stack layout widget. [...]

```
Stack(
  children: <Widget>[
    Container(
      width: 100,
      height: 100,
      color: Colors.red,
    ),
    Container(
      width: 90,
      height: 90,
      color: Colors.green,
    ),
    Container(
      width: 80,
      height: 80,
      color: Colors.blue,
    ),
  ],
)
```

Flutter

Table

A widget that uses the table layout algorithm for its children.

Constructors

```
Table({Key key,
List<TableRow> children: const [],
Map<int, TableColumnWidth> columnWidths,
 TableColumnWidth defaultColumnWidth: const FlexColumnWidth(1.0),
 TextDirection textDirection,
 TableBorder border,
TableCellVerticalAlignment defaultVerticalAlignment: TableCellVerticalAlignment.top,
 TextBaseline textBaseline})
```

Creates a table. [...]

Flutter

Wrap

A widget that displays its children in multiple horizontal or vertical runs.

Constructors

```
Wrap({Key key,
 Axis direction: Axis.horizontal,
 WrapAlignment alignment: WrapAlignment.start,
 double spacing: 0.0,
 WrapAlignment runAlignment: WrapAlignment.start,
 double runSpacing: 0.0,
 WrapCrossAlignment crossAxisAlignment: WrapCrossAlignment.start,
 TextDirection textDirection,
VerticalDirection verticalDirection: VerticalDirection.down,
 List<Widget> children: const []})
```

Creates a wrap layout. [...]

alignment → WrapAlignment How the children within a run should be placed in the main axis. [...]
crossAxisAlignment → WrapCrossAlignment How the children within a run should be aligned relative to each other in the cross axis. [...]
direction → Axis The direction to use as the main axis. [...]
runAlignment → WrapAlignment How the runs themselves should be placed in the cross axis. [...]
runSpacing → double How much space to place between the runs themselves in the cross axis. [...]
spacing → double How much space to place between children in a run in the main axis. [...]
textDirection → TextDirection Determines the order to lay children out horizontally and how to interpret start and end in the horizontal direction. [...]
verticalDirection → VerticalDirection Determines the order to lay children out vertically and how to interpret start and end in the vertical direction. [...]

Flutter