Newtonian Docs

Placeables

A program will always have exactly one root positionless placeable (defined below).

Basic Components

A placeable is defined using several of the following components (depending on the placeable type, some do not apply)

```
Key ( PropertyBody ) : PositionTag {
  PlaceableBody
}
```

Key

A name that identifies the type of placeable

PropertyBody

It includes the properties that define how the placeable should be drawn. Properties are listed as pairs `propertyKey: propertyValue` separated by commas.

```
Key (
  propertyKey1: propertyValue1,
  propertyKey2: propertyValue2,
  ...
  propertyKeyN: propertyValueN
) :PositionTag { PlaceableBody }
```

⚠️ If the same property is defined more than once for the same placeable, the last definition will be the one to be used (Although this is not an error, a warning is displayed upon code generation).

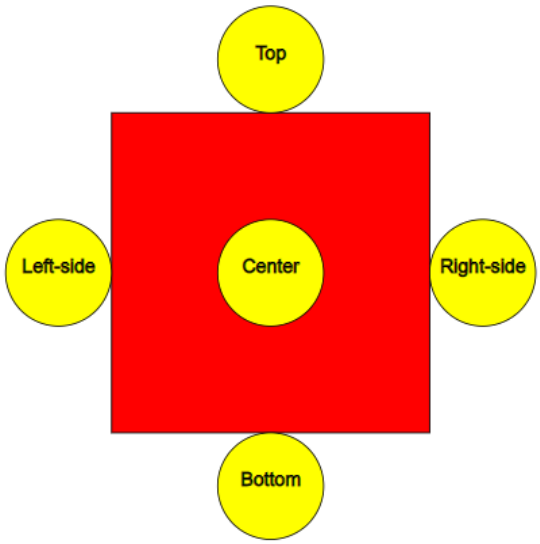
PositionTag

It defines the position from which the placeable is drawn with respect of the placeable's parent. This tag can not be used when the placeable is the root of the program (it has no parent), or when its parent is of type Alignment. These type of placeables are called `positionless placeables` .

The possible values this tag can take are:

`Top` , `Bottom` , `Left-side` , `Right-side` , `Center` (default)

```
Block(height: 3, width: 3, color: red) {
    Ball(color: yellow, label: "Top"): Top,
    Ball(color: yellow, label: "Bottom"): Bottom,
    Ball(color: yellow, label: "Left-side"): Left-side,
    Ball(color: yellow, label: "Right-side"): Right-side,
    Ball(color: yellow, label: "Center"): Center,
}
```



PlaceableBody

It contains a list of placeables separated by commas that will all be children of the current placeable.

```
Key ( PropertyBody ) :PositionTag {
  Pl1Key ( Pl1PropertyBody ) : Pl1PositionTag {
    Pl1PlaceableBody
  },
  ...
  PlNKey ( PlNPropertyBody ) : PlNPositionTag {
    PlNPlaceableBody
```

```
  }
}
```

PositionlessPlaceableBody

We will refer as `PositionlessPlaceableBody` to a PlaceableBody that contains PositionlessPlaceables (placeables that do not contain their PositionTag)

```
Key ( PropertyBody ) :PositionTag {
  Pl1Key ( Pl1PropertyBody ) {
    Pl1PlaceableBody
  },
  ...
  PlNKey ( PlNPropertyBody ) {
    PlNPlaceableBody
  }
}
```

Alignment

Structure

```
Key [PositionTag] [PositionlessPlaceableBody]
```

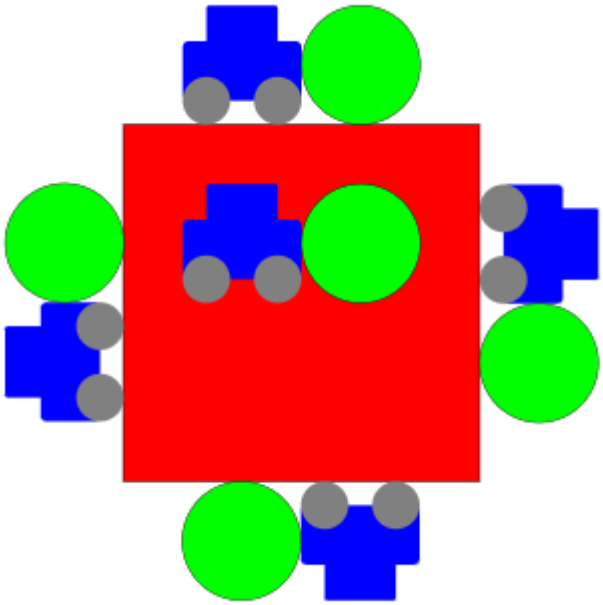
🚨 An alignment placeable cannot be a direct child of another alignment placeable. (Error)

Row

It will align all its children placeables “horizontally” (see example).

```
Block(color: red, height: 6, width: 6){
    Row: Left-side{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
    },
    Row: Right-side{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
    },
    Row: Bottom{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
```

```
    },
    Row: Top{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
    },
    Row: Center{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
    }
}
```

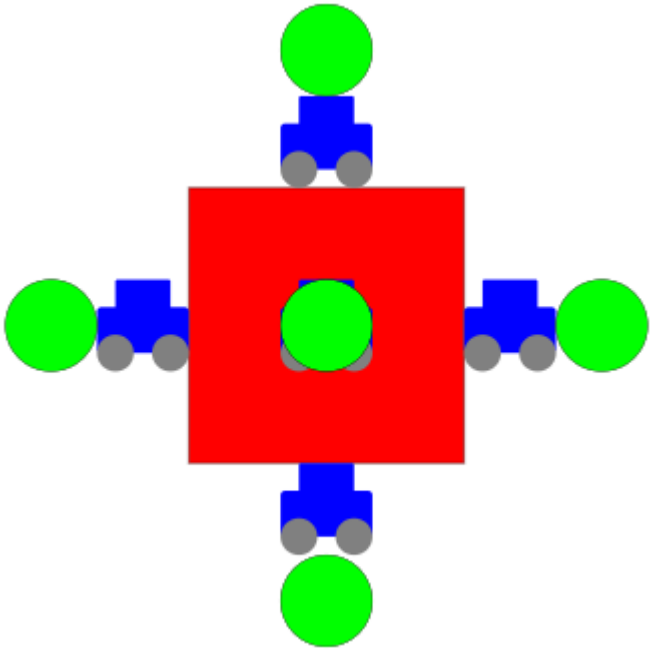


Column

It will align all its children placeables “vertically” (see example).

```
Block(color: red, height: 6, width: 6){
    Column: Left-side{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
    },
    Column: Right-side{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
    },
    Column: Bottom{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
    },
    Column: Top{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
    },
```

```
    Column: Center{
        Car(color:blue, height: 2, width: 2),
        Cylinder(color: green, radius: 1)
    }
}
```



Objects

Structure

```
Key [PropertyBody] [PositionTag] [PlaceableBody]
```

Block/Car

Properties:

- `color: color_t`
- `label: string`
- `width: number`
- `height: number`

Cylinder/Ball

Properties:

- `color: color_t`
- `label: string`
- `radius: number`

HorizontalPlane/VerticalPlane

Properties:

- `color: color_t`
- `label: string`
- `angle: number`
- `angle-label: string`
- `length: number`
- `friction: boolean`
- `visible: boolean`

Linear Objects

Structure

```
Key [PropertyBody] [PositionTag]
```

Arrow

Properties:

- `color: color_t`
- `label: string`
- `angle: number`
- `angle-label: string`
- `length: number`
- `direction: direction_t`
- `double-arrow: boolean` → this makes the arrow have a tip on both ends.

- `reverse-arrow: boolean` → this reverses the direction of the arrow meaning that the tip will be on the start rather than the end of the arrow.

Rope/Spring

Properties:

- `color: color_t`
- `label: string`
- `length: number`

Spacer

A spacer is a blank space with the length specified, useful to leave a space between objects aligned with a row or column.

Properties

- `length: number`

Property Value Types

Number

Numeric value that can be either an `int` or `float`.

> 🚨 In the case of `angle`, its value must be between `-360` and `360`.
> For `length`, `radius`, `height`, `width` their value must be non-negative.

String

A character string between double quotation marks: `"string"`. It works with most characters of the Basic Miltilingual Plane of Unicode, including from regular ASCII characters to greek characters or mathematical symbols. Unfortunately, it does not support emojis for the time being.

None of the other types values should go between double quotation marks, since this is used to identify a string.

Boolean

Can only take two values: `true` or `false`

Color Type

It can be either one of the preset colors:

- `red` → rgb(255,0,0)
- `green` → rgb(0,255,0)
- `blue` → rgb(0,0,255)
- `black` → rgb(0,0,0)
- `yellow` → rgb(255,255,0)
- `orange` → rgb(255,140,0)
- `purple` → rgb(128,0,128)

or a custom color represented in hexadecimal preceeded by `#`. For example: `#FFAABB`

Direction Type

It can take one of the 8 following values:

Relative to the current parent angle:

- `right`
- `up`
- `left`
- `down`

Absolute direction, regardless of the parent angle:

- `east`
- `north`
- `west`
- `south`