JS FLEXOR
LIGHTWEIGHT JAVASCRIPT
LAYOUT LIBRARY

Flexor Documentation

Welcome to Flexor

Flexor is a lightweight tool that makes building web layouts easier and faster. It uses a single attribute called `data-flexor` on your HTML elements to control how they look and behave on different screens. Whether you're creating a simple webpage or a complex app, Flexor simplifies the process of arranging content so it looks good everywhere.

This documentation will guide you through everything you need to know about Flexor: how to install it, use it, everything. Let's get started!

Table of Contents

Getting Started

To use Flexor, you just need a web browser and a text editor. It works with plain HTML, but it also plays nicely with tools like React or Vue if you use those.

Installation

1. Download the Files: Copy the flexor.js and flexor.css files from this documentation into your project folder.
2. Link Them in Your HTML: Add these lines to your HTML file's <head> and <body> sections:

```html
<head>
  <link rel="stylesheet" href="flexor.css">
</head>
<body>
  <!-- Your content goes here -->
  <script src="flexor.js"></script>
</body>
```

3. **Start Using It:** Add the data-flexor attribute to any HTML element, and Flexor will take care of the rest.

First Example

Here's a quick example to see Flexor in action:

```html
<div data-flexor="flex row gap-10px stack-500px">
  <div style="background: lightblue;">Box 1</div>
  <div style="background: lightgreen;">Box 2</div>
  <div style="background: lightcoral;">Box 3</div>
</div>
```

This creates a horizontal row of three boxes with 10 pixels of space between them. When the screen gets smaller than 500 pixels wide, the boxes stack vertically like a tower.

Basic Usage

Flexor works by reading the data-flexor attribute on an HTML element and turning it into a layout. You tell it what to do with a few words, and it arranges your content accordingly.

Simple Row Layout

```html
<div data-flexor="flex row gap-1rem">
  <div>Item 1</div>
  <div>Item 2</div>
  <div>Item 3</div>
</div>
```

This makes a row of items with 1 rem (a unit based on your text size, about 16 pixels by default typically) of space between them. Think of it as laying out books side by side on a shelf with a little gap between each.

Column Layout

```html
<div data-flexor="flex col gap-10px">
  <div>Top</div>
  <div>Middle</div>
  <div>Bottom</div>
</div>
```

This stacks items vertically with 10 pixels between each of them.

Syntax Explained

The `data-flexor` attribute is a list of words that tell Flexor how to arrange your content. Here's what each part means:

1. Mode: The first word decides the layout type. Right now, it's always `flex`, which uses a system called Flexbox (a technical term for a way to arrange things flexibly, like organizing a row of toys that can bend or stack). In the future, we might add `grid` for more complex layouts.
   - Example: `flex`
2. Direction: The second word says whether items go side by side (`row`) or up and down (`col`).
   - Example: `row` or `col`
3. Proportions: Numbers after the direction set how much space each item gets compared to others. For instance, `1 2 1` means the middle item is twice as wide as the others in a row.
   - Example: `1 2 1`
   - Note: Skip this if you use `smart` (explained later).
4. Gap: Use `gap-<size>` to add space between items. You can use pixels (`10px`), rem (`1rem`), or other units. It's like putting spacers between books on a shelf.
   - Example: `gap-10px`

5. Stack At: Use `stack-<size>` to make a row turn into a column when the screen gets smaller than a certain width. You can use preset names like `mobile` (600px) or custom sizes like `500px`.
   - Example: `stack-500px`
6. Align: Use `align-<option>` to position items vertically in a row or horizontally in a column. Options are `start`, `center`, or `end`. It's like deciding if toys sit at the top, middle, or bottom of a shelf.
   - Example: `align-center`
7. Justify: Use `justify-<option>` to spread items horizontally in a row or vertically in a column. Options include `start`, `center`, `end`, `space-between`, `space-around`, or `space-evenly`. Think of it as arranging chairs with even gaps or pushing them to one side.
   - Example: `justify-space-between`
8. Wrap: Add `wrap` to let items move to a new line if they don't fit, like wrapping a long line of toys onto the next shelf.
   - Example: `wrap`
9. Smart: Add `smart` to let Flexor adjust item sizes based on their content, like giving more room to a big book on a shelf.
   - Example: `smart`

Full Example

```html
<div data-flexor="flex row 1 2 1 gap-10px stack-500px align-center justify-space-between wrap smart">
    <div>Short</div>
    <div>Longer content here</div>
    <div>Medium</div>
  </div>
```

This creates a row where the middle item is bigger due to `smart`, with 10px gaps, centered vertically, spaced out horizontally, wrapping if needed, and stacking below 500px.

Smart Layouts

The `smart` option makes Flexor figure out how much space each item needs based on its content. Instead of you setting sizes like `1 2 1`, Flexor measures each item naturally (like weighing books to decide shelf space) and adjusts them automatically.

How It Works

When you add `smart`, Flexor:

1. Temporarily hides the layout to measure each item's natural size (width for rows, height for columns).
2. Sets those sizes as the basis for how much space each item gets.
3. Shows the layout again with the new sizes.

Example

```html
<div data-flexor="flex row gap-10px smart">
  <div>Hi</div>
  <div>This is a very long sentence that needs more space.</div>
  <div>Hello</div>
</div>
```

The middle item gets more space because it has more text, without you needing to guess the proportions.

Framework Integration

Flexor works great with frameworks like React and Vue. Here’s how to use it with special components that make it feel natural in those environments.

React Integration

Create a component to wrap Flexor:

```jsx
import React, { useEffect, useRef } from 'react';
import Flexor from './flexor.js';

const FlexorContainer = ({ children, config, ...props }) => {
  const ref = useRef(null);

  useEffect(() => {
    if (ref.current) {
      const dataFlexor = Flexor.generateConfig({ mode: 'flex', ...config });
      ref.current.setAttribute('data-flexor', dataFlexor);
      Flexor.applyTo(ref.current);
    }
  }, [config]);

  return <div ref={ref} {...props}>{children}</div>;
};

// Usage
export default function App() {
  return (
    <FlexorContainer config={{ direction: 'row', gap: '10px', smart: true }}>
      <div>Item 1</div>
      <div>Item 2 with more text</div>
      <div>Item 3</div>
    </FlexorContainer>
  );
}
```

Vue Integration

Create a Vue component:

```vue
<template>
  <div ref="container" :data-flexor="generatedConfig" v-bind="$attrs">
    <slot></slot>
  </div>
</template>

<script>
import { defineComponent, ref, computed, onMounted } from 'vue';
import Flexor from './flexor.js';

export default defineComponent({
  name: 'FlexorContainer',
  props: ['config'],
  setup(props) {
    const container = ref(null);
    const generatedConfig = computed(() => Flexor.generateConfig({ mode: 'flex', ...props.config }));

    onMounted(() => {
      if (container.value) {
        Flexor.applyTo(container.value);
      }
    });

    return { container, generatedConfig };
  }
});
</script>

<!-- Usage -->
<template>
  <flexor-container :config="{ direction: 'row', gap: '10px', smart: true }">
    <div>Item 1</div>
    <div>Item 2 with more text</div>
    <div>Item 3</div>
  </flexor-container>
</template>
```

These components let you use Flexor naturally in React or Vue, passing a configuration object instead of writing the data-flexor attribute directly.

Visual Editor Support

Flexor can work with a visual layout editor (a tool where you drag and drop items to design a layout). The Flexor.generateConfig function turns a setup from the editor into a data-flexor string.

Example Configuration

```javascript
const config = {
  mode: 'flex',
  direction: 'row',
  gap: '10px',
  stackAt: '500px',
  align: 'center',
  justify: 'space-between',
  wrap: true,
  smart: true
};
const dataFlexor = Flexor.generateConfig(config);
console.log(dataFlexor); // Outputs: "flex row gap-10px stack-500px align-center justify-space-between wrap smart"
```

How to Use It

Imagine a visual editor where you drag three boxes into a row, set a 10px gap, and click a “smart” checkbox. The editor builds the config object above, calls generateConfig, and applies the result to your HTML.

Plugins

Here’s an example plugin that logs layout details:

```javascript
Flexor.registerPlugin('logger', (container, config) => {
  console.log('Layout applied to:', container.tagName, 'with config:', config);
});
```

Whenever Flexor styles an element, this plugin prints details to the console.

Performance Tips

Flexor is designed to be fast, but here are tips to keep it running smoothly:

- **Use Smart Sparingly**: The `smart` option measures sizes, which takes a tiny bit of time. Use it only when you need content-based sizing.
- **Limit Containers**: If you have lots of Flexor elements, it might slow down the page load. Try to use it for main sections rather than every small box.
- **Test Without JS**: Flexor has a basic fallback in flexor.css, so check how your site looks if JavaScript is off (like testing a backup plan).

Troubleshooting

My Layout Doesn't Work

- Check the Attribute: Make sure `data-flexor` is spelled right and has valid words (e.g., `flex row` not `flex rows`).
- Load Order: Ensure `flexor.js` is loaded after your HTML content in the `<body>` tag.

It's Not Responsive

- Stack Value: Confirm your `stack-<size>` is correct (e.g., `stack-500px` not `stack-500`).
- CSS Loaded: Verify `flexor.css` is linked in your `<head>`.

JS FLEXOR
LIGHTWEIGHT JAVASCRIPT
LAYOUT LIBRARY