# Container Queries and new CSS to go along with them

The CSS Containment Module Level 3 has a formal definition of containers, container queries and container relative length units.

This post contains both parts that are supported in Chromium browsers behind a flag and others that are under development. **It is not production code and should not be used as such**.

## Defining the container

Before we use container relative length units we need to define a container that the units will refer to.

Containers allow you to create finer grained level of sizing for components that will be more specific than regular units (that refer to a window) or viewport units that refer to the viewport (usually the same as the window)

If you set a container, then you can choose to size the components based on the container's size rather than the viewport or Window.

The following table shows the different values for the `contain` property. Some of these values were defined in the level 2 CSS containment module so I indicate the spec version they were defined in.

| Types of Containment | Spec Introduced in | Description |
|---|---|---|
| **Size** | Level 2 | size: The size of the element can be computed without checking its children, the element dimensions are independent of its contents. |
| **Layout** | Level 2 | The internal layout of the element is totally isolated from the rest of the page, it's not affected by anything outside and its contents cannot have any effect on the ancestors. |
| **Style** | Level 2 | The effects of counters and quotes cannot escape this element, so they are isolated from the rest of the page. |

| Types of Containment | Spec Introduced in | Description |
|---|---|---|
| **Paint** | Level 2 | Descendants of the element cannot be displayed outside its bounds, nothing will overflow this element (or if it does it won't be visible). |
| **Inline-size** | Level 3 | Descendants of the element will only respond to changes on the element's width, the inline axis in left-to-write languages. |
| **Block-size** | Level 3 | Descendants of the element will only respond to changes on the element's height, the block axis in left-to-write languages. |

There are also two compound values that you can use as shorthand for defining multiple types of containment.

- **content**: Which is equivalent to contain: layout paint style.
- **strict**: This is equivalent to contain: layout paint size style.

An example of a container ready to handle container relative sizes could look like this:

```
.card-container {
  contain: layout content inline-size style;
}
```

This container uses the new `inline-size` and the `layout` and `style` containment values from the level 2 specification to indicate the expected behavior for the children of the container.

We then define the elements that we want to use inside the container.

```
.card {
  display: flex;
  flex-direction: column;
  flex-wrap: wrap;

  gap: 1rem;
}
```

```
.card--thumb {
  aspect-ratio: 1 / 1;
  flex: 0 0 150px;
  background-color: #663399;
  border-radius: 7px;
}

.card-title {
  font-weight: bold;
  font-size: 1.5rem;
  margin-bottom: 0.5rem;
}

.card--content {
  flex: 1;
}
```

We then use one or more `@container` at-rule to define the changes that we want to make based on the size of the container.

```
@container (min-width: 400px) {
  .card {
    flex-direction: row;
  }

  .thumb {
    flex: 0 0 100px;
    align-self: flex-start;
  }
}


@container (min-width: 600px) {
  .card--thumb {
    flex: 0 0 150px;
  }
}
```

```css
@container (min-width: 800px) {
  .card {
    position: relative;
    justify-content: center;
    align-items: center;
    min-height: 350px;
  }

  .thumb {
    position: absolute;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    opacity: 0.25;
  }

  .content {
    position: relative;
    flex: unset;
  }

  .title {
    font-weight: bold;
    font-size: 1.5rem;
    margin-bottom: 0.5rem;
  }

  .desc {
    max-width: 480px;
    margin-left: auto;
    margin-right: auto;
    text-align: center;
    color: #222;
  }
}
```

# Container type

**Note:**

The properties described below are different than the contain properties in the CSS Containment Module Level 2. You can use `container-type` without using `contain`.

The `container-type` property establishes the element as a query container for the purpose of container queries, allowing style rules styling its descendants to query various aspects of its sizing, layout, and style and respond accordingly.

The available values are:

`size` : Establishes a query container for size queries on both the inline and block axis. Applies layout, style, and size containment to the principal box.

`inline-size` : Establishes a query container for size queries on the container's own inline axis. Applies layout, style, and inline-size containment to the principal box.

`block-size` : Establishes a query container for size queries on the container's own block axis. Applies layout, style, and block-size containment to the principal box.

`style` : Establishes a query container for style queries.

`state` : Establishes a query container for state queries. They have been pushed to the next level of the CSS containment specification

# Container name

The container-name property specifies a list of query container names. These names can be used by `@container` rules to filter which query containers are targeted.

none : The query container has no query container name.

`<custom-ident>` : Specifies a query container name as an identifier.

`<string>` : Specifies a query container name as a `<string>` value; this computes to an identifier with the same value as the given `<string>`.

With `container-name` developers can specify which containers they want to target with `@container` queries and use the same container for multiple child elements.

```
main {
  container-type: size;
  container-name: page-layout;
}

.my-component {
  container-type: inline-size;
  container-name: component-library;
}

@container page-layout (block-size > 12em) {
  .card { margin-block: 2em; }
}

@container component-library (inline-size > 30em) {
  .card { margin-inline: 2em; }
}
```

## Container

This is a shorthand property to set both `container-type` and `container-name`. The values are separated by a slash `/`.

```
main {
  container: size / page-layout;
}

.my-component {
  container: inline-size / component-library;
}
```

# Container relative length units

These units are similar to viewport units except that they are relative to the dimensions of a the closes element that defined a container with `container-type`.

| unit | relative to |
|---|---|
| cqw | 1% of a query container's width |
| cqh | 1% of a query container's height |
| cqi | 1% of a query container's inline size |
| cqb | 1% of a query container's block size |
| cqmin | The smaller value of cqi or cqb |
| cqmax | The larger value of cqi or cqb |

We can use these units in a similar way to viewport units, but they will refer to the closest ancestor that defines a container.

```
aside, main {
  container-type: inline-size;
}

.container {
  h2 {
    font-weight: bolder;
    font-size: clamp(1.25rem, 3qw, 2rem);
    margin-bottom: clamp(0.5rem, 1.5qw, 1rem);
  }
}
```

## What Happens If We Use Query Units Without Defining Containers?

If no eligible query container is available, then use the small viewport size for that axis.

The browser will deal with query units as if they were viewport units.

Consider the following example:

```
h2 {
    font-size: clamp(1.25rem, 3qw, 2rem);
}
```

If there is no container that is defined for the <h2> or parents, the browser will consider 3qw as if it was 3% of the viewport width, most likely not what you want to do.

# Gotchas

**The @container API is not stable**, and is subject to syntax changes. If you try it out on your own, you may encounter bugs.

**Right now this only works in Chromium browsers** (Chrome, Edge, Opera, and others) if you enable the `#enable-container-queries` flag in `chrome://flags`.

Use the following links to track support on your browser:

- Chrome
- Firefox
- Safari

**Currently, you cannot use height-based container queries, using only the block axis**. In order to make grid children work with @container, you'll need to add a wrapper element. Despite this, adding a wrapper lets you still get the effects you want.

# Links and resources

- Container Queries
    - Next Gen CSS: @container
    - Container Queries are actually coming
    - Container Query Solutions with CSS Grid and Flexbox
    - Container Query Spec

        - CSS Tricks post on @container
        - Web.dev article on containment
- Container Relative Lengths
    - Container Relative Lengths in the CSS Containment Level 3 specification
    - Container Units Should Be Pretty Handy — CSS Tricks
    - CSS Container Query Units — Ahmad Shadeed
- Demos
    - Codepen — Miriam Suzanne
    - (forked) Codepen — Original Pen by Chris Coyier
    - Query Units - Card – Ahmad Shadeed
    - Query Units - Importance – Ahmad Shadeed
    - Query Units - Bio – Ahmad Shadeed
    - Simplified Container Query Demo — Una
    - Podcast Card Demo — Una
    - Plant Store Demo — Una
    - Baby Clothes Demo — Una