Baseline features you can use today

- The dialog element
- Individual CSS transform properties
- New viewport units
- Deep copy in JavaScript
- The :focus-visible pseudo-class
- The TransformStream interface
- Wrap up

https://www.youtube.com/watch?v=x9rh0Du4Czg

The web is always evolving and browsers update constantly to give developers new tools to innovate on the platform. Things that previously required helper libraries become part of the web platform and supported on all browsers, along with shorter or easier ways to code design elements.

While this constant evolution and adaptation is helpful, it can also bring confusion. It can be difficult to navigate all these updates. As developers, we have questions like, "When will all browser engines support this new feature?" "When can I actually start using those features in my production code?" "For how long should we support older browsers?"

The true answer is "it depends". What is needed and what is usable really depends on your user base, tech stack, your team structure, and supported devices. But, one thing we all agree, is that the current landscape of the web can make it difficult to make those decisions.

The Chrome team is collaborating with other browser engines and the web community to bring more clarity. This includes our work on projects like Interop 2023 which helps to improve interoperability of a set of key features. But what about features landed in the past few years? Are experimental features we learned about two years ago ready to use?

In this post, I want to highlight some features that are now available to all major browser engines for the past two major versions. This is the cut-off point where we feel that the majority of developers will feel a feature is safe to use, and is the feature set we're calling Baseline. For more,

please see the announcement of Baseline here.

The dialog element

The `<dialog>` element is a new HTML element to create dialog prompts such as popups and modals.

To use it, define the modal element, then open the dialog by calling the `showModal()` method on the dialog element.

```
<dialog id="d">
  <form method="dialog">
    <p>Hi, I'm a dialog.</p>
    <button>ok</button>
  </form>
</dialog>

<button onclick="d.showModal()">
  Open Dialog
</button>
```

As a native HTML element, features like focus management, tab tracking, and keeping the stacking context are built in. For more, read Building a dialog component.

Individual CSS transform properties

Using CSS transforms is a performant way to add a movement to your site.
You might be familiar with writing CSS transforms with three properties in one line.
With individual transform properties you can now specify transform properties individually. This comes in handy when you are writing complex keyframe animations.

```
.target {
  translate: 50% 0;
  rotate: 30deg;
  scale: 1.2;
```

```
}
```

For an in-depth explanation of this change, read Finer grained control over CSS transforms with individual transform properties.

New viewport units

On mobile, viewport size is influenced by the presence or absence of dynamic toolbars. Sometimes you have a URL bar and navigation toolbar visible, but sometimes those toolbars are completely retracted.
The actual size of viewport will be different depending on whether the toolbars are visible or not. New viewport units like `svh` and `lvh` give web developers finer control when designing for the mobile. You can learn more in the article The large, small, and dynamic viewport units.

Deep copy in JavaScript

In the past, to create a deep copy of an object with no reference to the original object, a popular hack was `JSON.stringify` combined with `JSON.parse`. This was such a common hack that V8 (Chrome's javascript engine) aggressively improved the performance of this trick. But, you don't need this hack anymore with `structuredClone`.

```
const original = {id: 0, prop: {name: "Tom"}}

/* Old hack */
const deepCopy = JSON.parse(JSON.stringify(original));

/* New way */
const deepCopy = structuredClone(original);
```

Please refer to Deep-copying in JavaScript using structuredClone for more details.

The `:focus-visible` pseudo-class

As web developers, we all are familiar with that "focus ring" that shows up when you are navigating a page with a keyboard or clicking on input elements. It is a necessary feature for accessibility but sometimes it gets in the way of the visual design for different users. The `:focus-visible` CSS

pseudo-class checks if the browser believes that the focus should be visible. You can now specify styles for only when focus should be visible.

```
/* focus with tab key */
:focus-visible {
    outline: 5px solid pink;
}

/* mouse click */
:focus:not(:focus-visible) {
    outline: none;
}
```

Check out the Focus section on Learn CSS for more information.

The TransformStream interface

The TransformStream interface of the Streams API makes it possible to pipe streams into one another.

For example you can stream data from one place, then compress the data stream to another location as the data gets passed. The article Streaming requests with the fetch API walks you through this sample use case.

Wrap up

There are many more features that recently became interoperable and stable to use on the web platform. Going forward we will work with the WebDX Community Group to bring more clarity to these Baseline feature sets. Please check out web.dev/baseline for ongoing details.

If you want to stay up-to-date, our team is publishing articles when a feature becomes interoperable, and publish monthly updates on what's going on the web platform from experimental features to newly interoperable.

We are always curious if what we are doing helps you, or if you need different kinds of support, so please reach out to us at WebDX Community Group.

□□□ #3
□ □□□ □□□ 2026-10-31 21:39:29 UTC

 cocobook-sync □□□ 2026-11-04 17:36:34 UTC