

@starting-style debugging

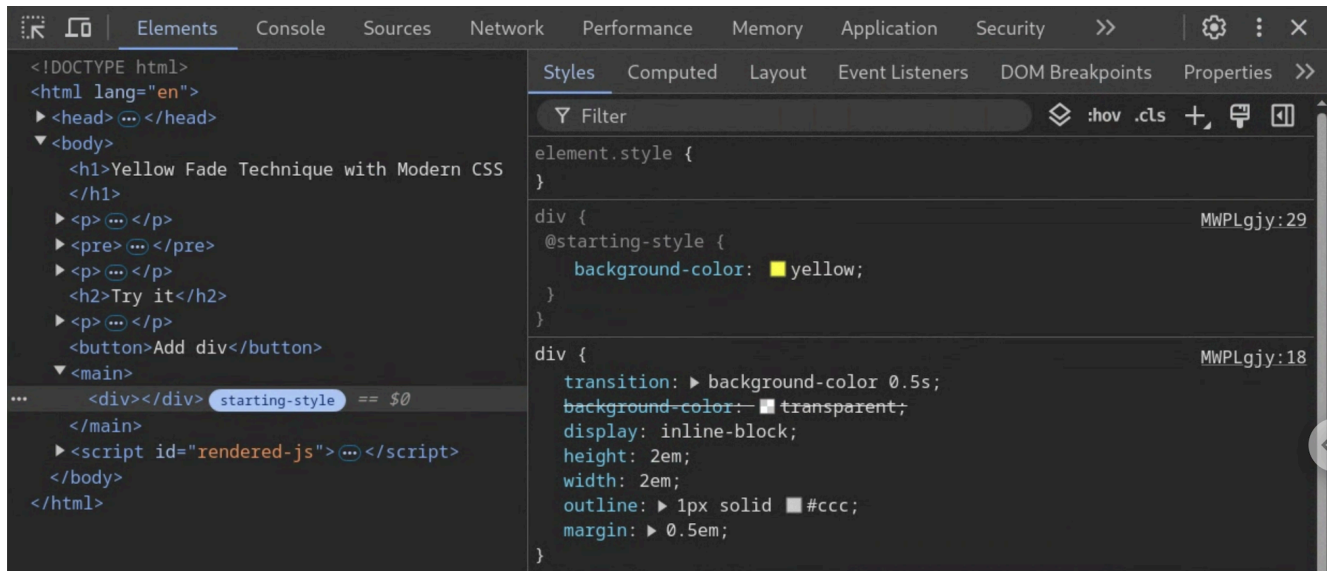
Attention: Externally visible, non-confidential

Author: Bramus Van Damme

Status: Accepted

Short Link: go/chrome-devtools:starting-style-debugging-design or
<https://bit.ly/devtools-starting-style-debugging-design>

Created: 2024-12-09 / **Last Updated:** 2025-09-22



One-page overview

Summary

Give authors control over whether an element should render with its `@starting-style` style rules applied or not so that they can see, inspect, and edit these CSS rules.

Platforms

All

Team

Chrome DevTools

Tracking issue

crbug.com/40278160

Value proposition

CSS `@starting-style` is a CSS feature that allows authors to define styles for an element before it has received the first style update. When setting those targeted properties to transition with CSS transitions, authors can use these starting-styles to create [smooth entry effects](#).

The feature is part of Interop 2024 and became Baseline Newly Available in 2024. It [shipped in Chrome 117](#). Demo: <https://codepen.io/bramus/pen/MWPLgjy>

Currently, these `@starting-style` rules are not surfaced to developers through Chrome DevTools. Authors are therefore left in the dark when it comes to debugging these.

This Design Doc describes a solution to this problem, allowing authors to inspect, edit, and see the `@starting-style` rules of an element.

Code affected

Elements panel related code in devtools-frontend + CDP + `inspector_css_agent` on the Blink/Chromium side.

Signed off by

Name	Write (not) LGTM in this column
pfaffe@chromium.org (DevTools Eng)	LGTM
alexrudenko@chromium.org (DevTools Eng)	LGTM
futhark@chromium.org (Blink Eng)	LGTM
pjmclachlan@chromium.org (PM)	

Terminology

- Starting Styles: CSS Rules or CSS Declarations that are applied using `@starting-style`
- Affected by Starting Styles: When there are Starting Styles targeted towards the element
- Starting Style State: When an element is rendered with its Starting Styles applied.

User stories

Know if an element is affected by Starting Styles or not

User goal:

As a front-end developer, I want to easily know if an element is affected by Starting Styles or not.

Current Issues:

1. Looking at the Elements Panel, it does not show which elements have Starting Styles. Authors need to observe the elements on the page to see if they have entry effects or not
2. Even if DevTools were to show the Starting Styles in the styles tab (see next user story), you would still need to click around in the tree to figure out which elements have Starting Styles.

See the Starting Styles that apply to an element

User goal:

As a front-end developer, I want to see which Starting Styles are declared and what the effect of those Starting Styles is.

Current Issues:

1. DevTools does not show the Starting Styles that apply to an element.
2. Because Starting Styles are ephemeral, developers can only see the visual outcome of these styles while the elements are transitioning.
3. To retrigger these visual effects, authors need to delete and reinsert the element they are inspecting.

Edit the Starting Styles

User goal:

As a front-end developer, I want to alter the Starting Styles that apply to an element.

Current Issues:

1. DevTools does not show the Starting Styles that apply (~ applied) to an element in the Styles Tab, so they cannot be edited.

Design

A solution to debugging `@starting-style` should have the following qualities:

- Correctness:
The rules that you see in DevTools should match with what is shown on the page.
- Easy Mental Model:
Authors should not have to manually put certain things together or imagine things to understand what is (or will be) going on.
- Discoverability:
It should be easy for authors to discover elements with Starting Styles declared on it.

Based on these three traits, the following 3-part solution is proposed:

1. Starting Styles Pill: In the elements tree, decorate elements that have Starting Styles with a pill to indicate that the element has `@starting-style` rules.
2. Starting Style State Toggle: In the elements tree, upon clicking the starting-style pill, the element can be forced into its Starting Style State. Clicking it again will revert back to the original state. These changes need to be animated so that the author can see the effect of their changes in the transition.
3. Show Starting Styles: When an element is forced into its Starting Style State, show the corresponding Starting Styles in the Styles Tab which authors can then manipulate + also see the visual impact because the element is forced into rendering in its Starting Style State.

Implementing this requires 4 changes:

1. CDP: Add a command to the backend to force an element to render with its Starting Styles.
2. Blink: Conditionally collect Starting Styles in InspectorCSSAgent and pass these to DevTools.
3. Elements Tree: Add a pill to indicate whether an element is affected by starting styles. Clicking the pill forces an element into its Starting Style State.
4. Styles Tab: Show the Starting Styles when the element's matched styles include them.

TL;DR

1 - On the backend, we're going to add a new method `forceStartingStyle` that can force an element into rendering with its Starting Styles applied.

2 - In Blink, when collecting styles, we're going to track whether an Element is affected by Starting Styles or not. Additionally, when the element is forced into its Starting Style State we are effectively going to collect these `@starting-style` rules and render the element with its Starting Styles. Both the `affectedByStartingStyles` and (when collected) the `StartingStyles` themselves are passed to DevTools.

3 - On the frontend, we're going to show a pill next to an element when it is affected by Starting Styles. Clicking the pill toggle-invokes the `forceStartingStyle` for that element.

4 - On the frontend, we're going to extend the Styles Tab to show the `@starting-style` rules when an element's reported matched styles include any Starting Style Rules.

1. Add command to backend to force an element to render with its Starting Styles.

CL: <https://crrev.com/c/6077714> (Merged)

We're going to add a new CDP method: `forceStartingStyle` that can be used to force a specific node to render in its Starting Style Mode. The Starting Style Mode can be toggled on and off.

For this `InspectorCSSAgent` will keep track of which nodes are forced into their Starting Style Mode in a new `HashSet` named `NodeIdToForcedStartingStyle`. The `HashSet` gets cleared when the `InspectorCSSAgent`'s non-persistent data gets reset.

To link things together a new probe `ForceStartingStyle` will be created. This probe will be taken into account in `CSSAnimations::CalculateTransitionUpdate` so that the element can render in its starting style state.

Protocol changes

- Addition of the `forceStartingStyle` command

```
None
# Ensures that the given node is in its starting-style state.
command forceStartingStyle
  parameters
    # The element id for which to force the starting-style state.
    DOM.NodeId nodeId
    # Boolean indicating if this is on or off.
    boolean forced
```

2. Keep track of an element having Starting Styles pass and conditionally collect these Starting Styles in `InspectorCSSAgent` to pass to DevTools.

CLs: <https://crrev.com/c/5916506> (Merged) + <https://crrev.com/c/6077714> (Merged) + <https://crrev.com/c/6165202>

We are going to adjust the `ElementRuleCollector::CollectMatchingRulesForListInternal` so that it collects the starting style rules for an element when the probe `ForceStartingStyle` for an element is enabled.

The `InspectorCSSAgent::FillAncestorData` method also gets adjusted so that it too takes the Starting Style Rules from the `parent_rule` into account in order to pass the data over CDP to DevTools. This is done by adding an `InspectorCSSAgent::CollectStartingStylesFromRule` method and adjusting the Chrome DevTools Protocol to support this new CSSRule type.

Finally `InspectorStyleSheet` also gets adjusted to take `kStartingStyleRule` into account when collecting and building rules. This is modeled similar to how Container Rules are collected.

`ElementRuleCollector::CollectMatchingRulesForListInternal` also gets adjusted to track whether an element has starting styles, even when not collecting them. When Starting Styles are detected, the `Element` is marked to be affected by Starting Styles. This flag is stored internally in a new `ElementRareDataVector` field and gets sent to DevTools as part of the `getComputedStyleForNode` call.

This flag gets sent as part `getComputedStyleForNode` instead of `getMatchedStylesForNode` because the `hasStartingStyles` flag needs to be read for every element shown in the Elements Tree, and not just for the selected element.

To be able to write `inspector-protocol` tests, the `css-helper.js` will be adjusted to also output `startingStyles` if any are present on the `rule`.

Protocol changes

- Addition of a new `StartingStyleRule CSSRuleType`

```
None
# Enum indicating the type of a CSS rule, used to represent the order of a style
rule's ancestors.
# This list only contains rule types that are collected during the ancestor rule
collection.
experimental type CSSRuleType extends string
enum
  MediaRule
  SupportsRule
  ContainerRule
  LayerRule
  ScopeRule
  StyleRule
  StartingStyleRule
```

- Introduction of a `CSSStartingStyleRule` at-rule descriptor

```
None
# CSS Starting Style at-rule descriptor.
experimental type CSSStartingStyle extends object
properties
  # The associated rule header range in the enclosing stylesheet (if
  # available).
  optional SourceRange range
  # Identifier of the stylesheet containing this object (if exists).
  optional StyleSheetId styleSheetId
```

- Addition of a `startingStyles` property of the type `CSSStartingStyleRule` on the `CSSRule` type

```
None
type CSSRule extends object
properties
  ...

  # @starting-style CSS at-rule array.
  # The array enumerates @starting-style at-rules starting with
  # the innermost one, going outwards.
  experimental optional array of CSSStartingStyle startingStyles
```

- Addition of `affectedByStartingStyles` on the `Node` type

None

```
# DOM interaction is implemented in terms of mirror objects that represent the actual
DOM nodes.
# DOMNode is a base node mirror type.

type Node extends object
  properties
    # Node identifier that is passed into the rest of the DOM messages as the `nodeId`.
Backend
  # will only push node with given `id` once. It is aware of all requested nodes and
  will only
  # fire DOM events for nodes known to the client.
  nodeId nodeId
  # The id of the parent node if any.
  optional nodeId parentId

  ...

  experimental optional boolean isScrollable
  experimental optional boolean affectedByStartingStyles
```

3. Element Tree: Add a clickable starting-style pill

CL: <https://chromium-review.googlesource.com/c/devtools/devtools-frontend/+5923558>

We introduce a new type `STARTING_STYLE` in the `AdornerManager`. In `ElementsTreeElements.updateStyleAdorners` we will conditionally add the adorer next to the element.

The adorer is only shown when the element is affected by Starting Styles, as exposed on the `Node` through the `affectedByStartingStyles` flag.

The click action on the adorer is set to toggle the Starting Styles on the element by invoking the CDP `forceStartingStyle` command.

4. Styles Tab: Show Starting Styles when enabled

CL: <https://chromium-review.googlesource.com/c/devtools/devtools-frontend/+5923558>

(Same as #3)

We will introduce a new `CSSStartingStyle` class to represent a Starting Style Rule. The `CSSRule` class gets extended to include a `startingStyles` property that is an array of `CSSStartingStyle` instances. This property gets populated from the `payload`.

When creating the ancestor rules in `StylePropertiesSection.createAncestorRules`, support for the `Protocol.CSS.CSSRuleType.StartingStyleRule` ruleType gets added so that it creates a new `ElementsComponents.CSSQuery.CSSQuery()` component with its title set to `@starting-style`.

This `@starting-style` component only gets shown when the element is forced into its Starting Style State. This is done by checking the contents of the `startingStyles`.

Limitations

1. When toggling a class that suddenly makes a **@starting-style** rule apply to an element is not caught.

Consider the following CSS:

```
CSS
div {
  background: lime;
  transition: background 0.2s ease-in;
}

.special {
  @starting-style {
    background: yellow;
  }
}
```

When having a **div** in the document it will show up in the tree without a Starting Style pill. Upon adding the **.special** class to it, it will still show up in the tree without a Starting Style pill.

This is because the element was already present in the DOM, so Blink does not consider the **@starting-style** rules to build the [before-change styles](#).

2. Toggling **@starting-style** on pseudos does not work

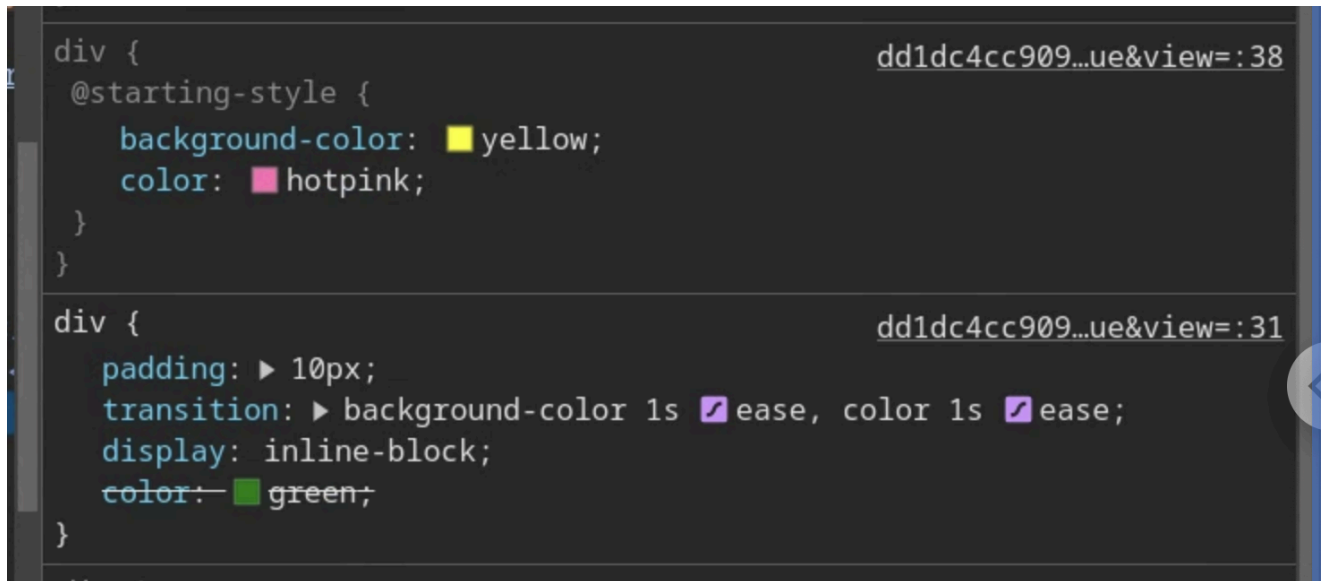
TODO

Alternatives considered

A bunch of alternatives were considered. These were discarded because they don't meet the three required qualities listed in [the Design section of this doc](#).

1. Always show @starting-style in the styles pane

In this solution, DevTools gets a simple adjustment to show the @starting-style rules the backend collected.



Pros:

- Discoverability: It's easy to see if an element has @starting-styles applied or not, but only after selecting it.
- Ease of use: Authors can immediately edit the @starting-styles.

Cons:

- Correctness: this representation is only correct while the element has its starting-styles applied. When inspecting elements much later in the page's lifetime, the @starting-style declarations are irrelevant, so DevTools would show them incorrect info.
- Mental Model:
 - Always showing these styles might make it harder for authors to know which declarations are *currently* applied and which are not. In the screenshot above, the `color: green` declaration is always stricken out, even when that element is no longer being rendered in its starting style state.
 - Editing the @starting-style has no effect on the currently visible element, so authors can only guess what the introduced changes look like.

2. Always show @starting-style in the styles pane + adjust resolution in DevTools

This is a variant of approach 1, in which DevTools gets adjusted to not consider @starting-style rules when striking out declarations. This is also what Firefox does.

Pros:

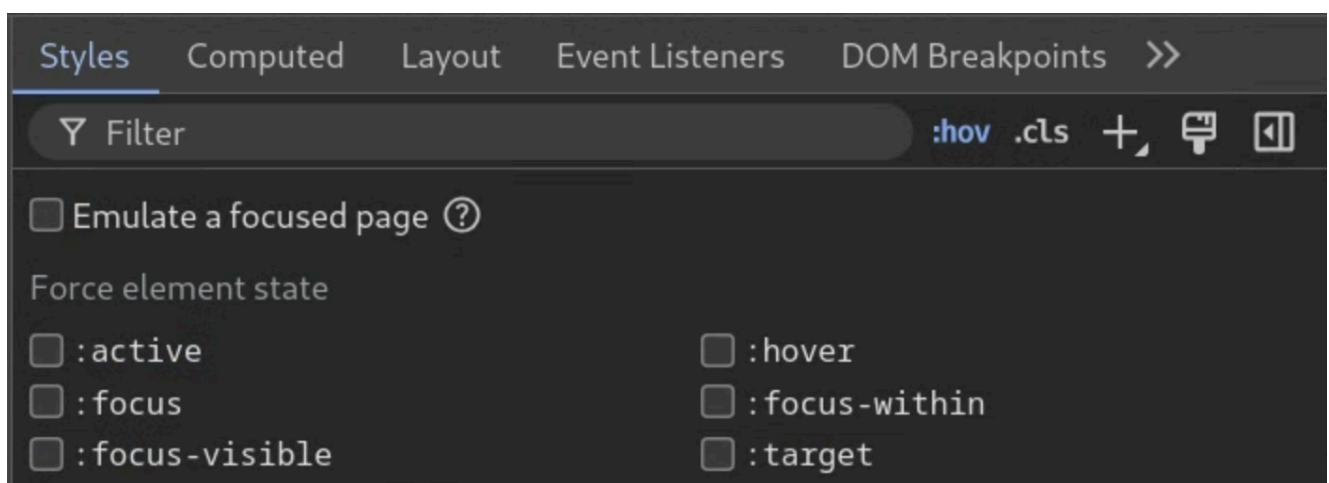
- Discoverability: It's easy to see if an element has `@starting-styles` applied or not.
- Correctness: The problem introduced by solution 1 has been circumvented.
- Ease of use: Authors can immediately edit the `@starting-styles`.

Cons:

- Mental Model:
 - Authors need to manually run a cascade in their head to know which declarations are *currently* applied and which are not.
 - Editing the `@starting-style` has no effect on the currently visible element, so authors can only guess what the introduced changes look like.

3. Allow `@starting-style` toggling trough `:hov` menu

This approach extends the `:hov` menu with an extra option to show the `@starting-style` rules. When enabled, the `@starting-style` declarations get shown in the styles pane.



Pros:

- Mental Model: Authors are explicitly opting in to showing the `@starting-style`, so they (should) know what is shown in the styles pane only defines the entry-effect of the inspected element.

Cons:

- Mental Model: Editing the `@starting-style` has no effect on the currently visible element, so authors can only guess what the introduced changes look like.
- Discoverability: It's not easy to see if an element had `@starting-styles` applied or not.
- Correctness: Showing the `@starting-styles` when enabled is only correct when the element has its `@starting-styles` applied. When inspecting elements much later in the page's lifetime, the `@starting-style` declarations are irrelevant. To fix this, the solution part of option 2 also needs to be implemented.
- Ease of use: It takes a few clicks before you can edit the `@starting-styles`.

4. Allow `@starting-style` toggling trough `:hov` menu + Force Element State

Same as approach 3, but this one also forces the inspected element into the `@starting-style` state.

Pros:

- Mental Model:
 - Authors are explicitly opting in to showing the `@starting-style`

- Editing the `@starting-style` has a visual effect on the currently visible element (state)
- Correctness: By also forcing the element into the `@starting-style` state, DevTools striking out other declarations applied to that element is correct.

Cons:

- Discoverability: It's not easy to see if an element had `@starting-styles` applied or not.
- Ease of use: It takes a few clicks before you can edit the `@starting-styles`.

5. Show `@starting-style` in its own section in the styles pane

Similar to `@keyframes`, a dedicated section in the styles pane to show the `@starting-styles` can be added.

This approach would require approach 2 to be also implemented (ignore when striking through).

Pros:

- Mental Model: Because the `@starting-styles` are shown separately, they – including inherited ones – are all grouped and authors can c
- Correctness: The problem introduced by solution 1 has been circumvented.
- Ease of use: Authors can immediately edit the `@starting-styles`.

Cons:

- Mental Model: Editing the `@starting-style` has no effect on the currently visible element, so authors can only guess what the introduced changes look like. Can be solved by adding a "Force state" button to the section.
- Discoverability: Authors most likely will need to scroll down before they can reach this section.

Rollout plan

The idea is to initially launch this behind a feature flag to gather feedback from authors.

Core principle considerations

See first part of the [Design](#) section.

Testing plan

???

Follow-up work

???