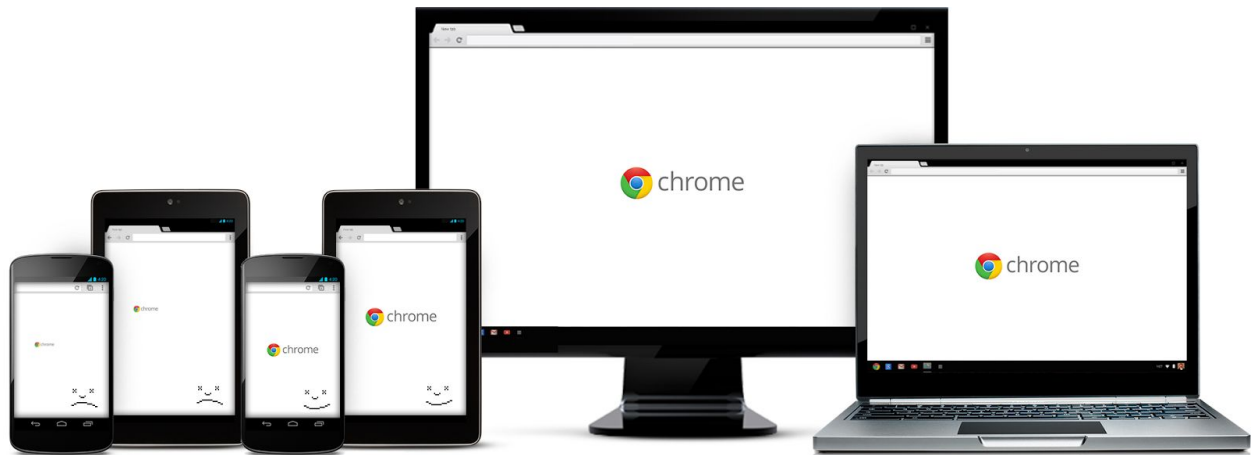


Chromium's Text Autosizer

skobes@google.com (Steve Kobes)

pdr@google.com (Philip Rogers)

last updated: 30 Jul 2014



The Text Autosizer is a component of the [Blink](#) rendering engine that is responsible for boosting font sizes on small screens. (Text autosizing is also known as "font boosting" or "font inflation".)

Text autosizing normally runs only on Chrome for Android. However, it can be applied in desktop Chrome by enabling device emulation in Dev Tools.

Unlike the previous implementation of text autosizing which required two layout passes, the new text autosizer is "single-pass", running concurrently with layout.

The text autosizer incorporates "fingerprinting", which uses heuristics to detect similarities in page elements. This improves consistency of the output on comment forums such as Hacker News, which have historically been handled poorly by text autosizing.

The text autosizer was enabled in trunk on 29 Apr 2014 in [r172846](#), and launched with M36 to Beta channel on 30 May and stable channel on 21 Jul. The code lives primarily in [renderer/core/layout/text_autosizer.cc](#). Tracking bug: <https://crbug.com/302005>.

Background

The primary reason to boost font sizes is that a page is *wider than the device*.



layout width of <http://en.wikipedia.org/> - 980 CSS pixels

screen width of portrait mode Nexus 5 -
360 density-independent pixels (DIPs)



Chrome for Android shows the page zoomed out, so that the entire width of the page fits on the screen. But because it is zoomed out, the text is too small to read without boosting.

Note that we have measured the screen width in *density-independent pixels* (DIPs) to ensure consistency across devices. The ratio of physical pixels to DIPs is given by the "device pixel ratio" (`window.devicePixelRatio`), and is often 2 or 3 on high-density screens.

Instead of boosting the font size, we could display the page at 100% (1 CSS pixel = 1 DIP). Then the text would be legible, but the user would need to scroll horizontally to read it, which is a very unpleasant experience.

Alternatively, we could reduce the [layout viewport](#) to match the screen width of the device. But this often destroys the appearance of a page that was designed for a desktop browser. Here's <http://en.wikipedia.org/> at a layout width of 360 CSS pixels:



Note the unpleasant overlap of elements in the header. Furthermore there is still horizontal scrolling because the page content overflows the initial containing block.

To avoid such results, Blink applies a default minimum width of 980 pixels to the layout viewport through the user-agent stylesheet in <Source/core/css/viewportAndroid.css>.

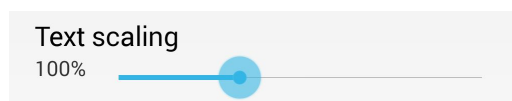
A webpage that supports narrower viewports can override this by specifying its own viewport parameters. For example, mobile-optimized or "responsive" websites often include a meta viewport tag such as

```
<meta name="viewport" content="width=device-width">
```

This results in a layout width equal to the device width, and usually disables text autosizing on the page (but see below for the exception).

Accessibility Text Scaling

The second reason for font boosting is to implement the effects of the "Text scaling" slider on the Accessibility preferences page.



The position of this slider acts as a multiplier on the effects of the text autosizer.

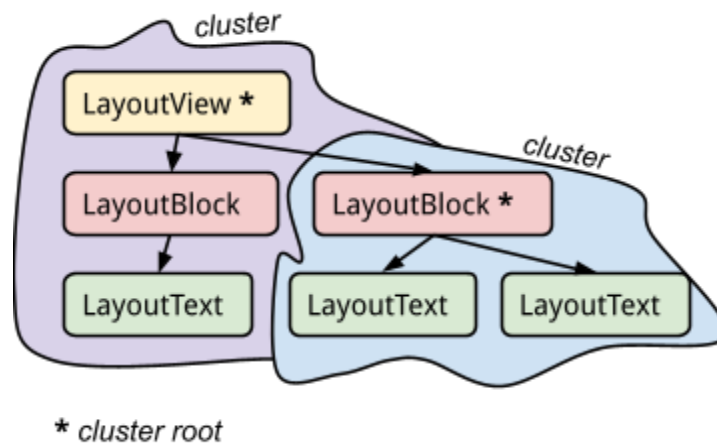
If the slider is > 100%, some text autosizing may be performed even on a page with width=device-width in its meta viewport tag.

If the slider is < 100%, the text autosizer applies less boosting than it normally would.

Clusters

A "cluster" is a subset of the layout tree in which a particular text autosizing multiplier is applied. This multiplier is called the "cluster multiplier". A cluster is identified by a block known as its "cluster root". The text autosizer divides the page into clusters by classifying certain blocks as cluster roots when they enter layout.

Every LayoutObject "belongs to" exactly one cluster, namely the cluster associated with the LayoutObject's nearest cluster-root ancestor in the layout tree. The text autosizer "inflates" (boosts the font of) every text node using the multiplier of the cluster it belongs to.



Clusters allow inflation to be proportional to the width of the column in which the text is laid out. Wider columns are more inflated, and narrower columns are less inflated. This serves two purposes:

- It helps preserve the structure of the page (overinflating a narrow column is likely to produce unpleasant wrapping of text or ugly layout artifacts).
- It optimizes for legibility after the user has zoomed into the column using pinch gestures or double-tapping. The zoom level that allows the column to fill the screen is the maximum zoom at which the text in that column can be read without having to scroll horizontally.

Suppressing Clusters

Clusters are also created to suppress autosizing in a region of the page. This is called a "suppressing cluster". Generally we prevent parts of the page from autosizing for several reasons, such as:

- Blocks with explicit height in CSS are not autosized, to avoid the text overflowing the declared height.
- Form controls are not autosized.
- Rows of links, such as navigation headers, are not autosized.
- Blocks marked "white-space: nowrap" in CSS are not autosized.
- Clusters that contain only a small amount of text (less than about 4 lines of text) are not autosized.

Formulas

Computed Font Size

The computed font size of autosized text is given by:

```
if specifiedSize ≤ 16:  
    computedSize = specifiedSize × clusterMultiplier  
  
if 16 ≤ specifiedSize ≤ (32 × clusterMultiplier – 16):  
    computedSize = (specifiedSize / 2) + (16 × clusterMultiplier – 8)  
  
if specifiedSize ≥ (32 × clusterMultiplier – 16):  
    computedSize = specifiedSize
```

Here, *specifiedSize* is the font size that would be used in the absence of autosizing based on the CSS and markup on the page, and *clusterMultiplier* is the multiplier of the cluster the text belongs to. The formula for determining *clusterMultiplier* is given in the next section. All units are in CSS pixels.

This formula can be summarized as: the cluster multiplier is fully applied up to a specified size of 16px, with a linear backoff beyond this threshold. The computed size is never less than the specified size.

The computed size is available to script via `getComputedStyle(element)['font-size']`, or in the computed styles section of the Elements tab in Dev Tools.

Cluster Multiplier

The cluster multiplier is generally computed as follows:

$$\text{clusterMultiplier} = \max(1, \text{textScalingSlider} \times \text{systemFontScale} \times \text{clusterWidth} / \text{screenWidth})$$

where:

- *clusterWidth* is roughly the width in CSS pixels of the cluster's text. In some cases this is obtained by measuring the layout width of the cluster root element. In other cases it is obtained by measuring the deepest block that contains all of the cluster's text nodes ("deepest block containing all text" or DBCAT). The cluster width is also clamped to be \leq the layout width.
- *screenWidth* is the width of the device in density-independent pixels (DIPs). This varies with orientation (portrait vs. landscape).
- *textScalingSlider* comes from the "text scaling" slider on the Accessibility settings page. Its default value is 1 (or 100%).
- *systemFontScale* is the *fontScale* field of the Android [Configuration](#) object, which reflects the user's system-wide selection in Settings > Display > Font size. It also defaults to 1.

(The fact that a meta viewport with *width=device-width* disables autosizing can be derived from this formula. Under default accessibility settings, *textScalingSlider* = *systemFontScale* = 1. If layout width = *screenWidth*, and *clusterWidth* \leq layout width, then we have *clusterWidth* / *screenWidth* \leq 1, and *clusterMultiplier* = 1.)

The value of (*textScalingSlider* \times *systemFontScale*) is passed into Blink through the [accessibilityFontScaleFactor](#) setting.

A suppressing cluster has a multiplier of 1.

Device Scale Adjustment

The device scale adjustment is an additional "fudge factor" that is applied to websites that have no explicit meta viewport or @viewport (i.e., most websites designed for desktop browsers).

When the device scale adjustment is applied, the formula for the cluster multiplier becomes:

$$\text{clusterMultiplier} = \max(1, \text{deviceScaleAdjustment} \times \text{textScalingSlider} \times \text{systemFontScale} \times \text{clusterWidth} / \text{screenWidth})$$

The device scale adjustment ranges from 1.05 to 1.3 depending on the screen size. It is computed in [chrome_content_browser_client.cc](#), and passed into Blink through the [deviceScaleAdjustment](#) setting.

The device scale adjustment was introduced to compensate for poor legibility on tablets caused by differences in viewing distance. A legible font size on a phone would appear smaller on a tablet despite being equal in DIPs, due to the tablet being held further from the user's eyes. For more background, see <http://crbug.com/229151>.

Since M32, the device scale adjustment is disabled when the page specifies explicit viewport parameters using meta viewport or @viewport. This ensures that the page author can reliably disable autosizing with width=device-width (unless the user has boosted the accessibility text scaling slider past 100%).

Alternative proposals for applying the device scale adjustment are explored in [Autosizing Fudge Factor Behavior Proposals](#).

Implementation

Each [Document](#) owns a lazily constructed instance of [TextAutosizer](#), which is responsible for inflating the text and keeping track of the information it needs.

Inflation during layout

To avoid having multiple layout passes, inflation must occur *during* layout, not before or after. This is because:

- the amount of inflation depends on the width of a containing block (the cluster width), and
- the result of the inflation may alter the height of that block, since the larger text occupies more space.

Text autosizing hooks into the layout treewalk through `TextAutosizer::LayoutScope`, and its subclass `TextAutosizer::TableLayoutScope`. This allows inflation of each text node to occur after its containing block has computed its width, but before that block's children enter layout.

The simplified model is:

1. `LayoutBlockFlow::layoutBlockFlow` calculates the block's width
2. `TextAutosizer::LayoutScope` ctor inflates the text inside the block (based on the width)
3. `layoutBlockFlow` lays out the block's children (using the inflated font size)
4. `layoutBlockFlow` calculates the block's height (using the sizes of its children)

The text autosizer inflates a `LayoutText` node by writing the cluster multiplier into the text's `RenderStyle` using `RenderStyle::setTextAutosizingMultiplier`. This updates the `FontDescription` with the appropriate computed font size, and calls `Font::update` to update the underlying font object.

The cluster stack

The text autosizer keeps a stack of Cluster objects representing the clusters that have entered layout. The TextAutosizer::LayoutScope ctor looks at each block that enters layout to decide whether it is a cluster root. If it is, it creates a new Cluster object and pushes it onto the stack, where it stays until the LayoutScope destructor pops it off.

This ensures that the text autosizer can always access the "current cluster" by peeking at the top of the cluster stack.

Note that clusters only exist during layout. The Cluster objects are owned by the cluster stack, and the cluster stack is empty when layout finishes.

Updating page info

The text autosizer is notified via updatePageInfo() when any input to the cluster multiplier computation changes. These are:

- screen width in DIPs
- layout width of the top-level frame in CSS pixels
- accessibility settings (text scaling slider, system font scale)
- value of [Settings::textAutosizingEnabled\(\)](#) - this defaults to true on Android and false on desktop, but can be toggled by Dev Tools or layout tests

The text autosizer caches the last-seen values of these inputs in the TextAutosizer::PageInfo object. If updatePageInfo() decides that the cluster multipliers are invalid as a result of one of these changes, it will mark all the text nodes as needing relayout so that the multipliers are updated.

Fingerprinting and superclusters

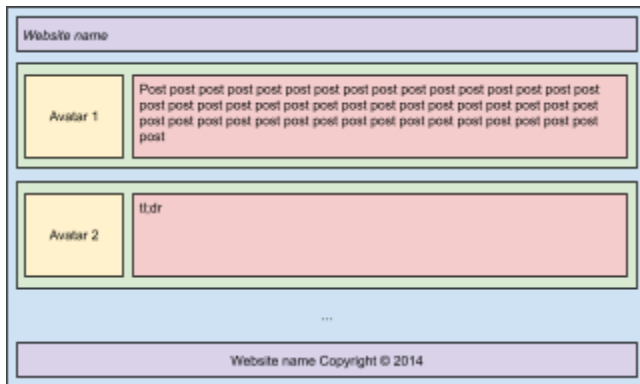
During style recalculation, TextAutosizer::record() computes "fingerprints" (hashes) for blocks on the page. Fingerprints are used for recognizing similarities between clusters on websites such as forums, where each post or comment may be a separate cluster. The mapping between blocks and their fingerprints is maintained by TextAutosizer::FingerprintMapper.

During layout, if there are two or more clusters whose roots share a common fingerprint, a "supercluster" is created. Superclusters enforce a uniform multiplier and uniform suppression status (e.g., if at least one cluster has enough text to be autosized, the entire supercluster is permitted to autosize).

Fingerprint source data

To compute a fingerprint for a `LayoutObject`, the text autosizer initializes a `FingerprintSourceData` struct with the corresponding element's tag name and several of its computed style properties, then uses `WTF::StringHasher` to compute an unsigned integer hash value from the bytes in this struct. This value is the fingerprint.

In addition to computed style values, the `FingerprintSourceData` struct stores the fingerprint of the *parent* element, recursively computed and cached by the `FingerprintMapper`. This means that the fingerprint incorporates styles from all the elements in the node's ancestor chain.



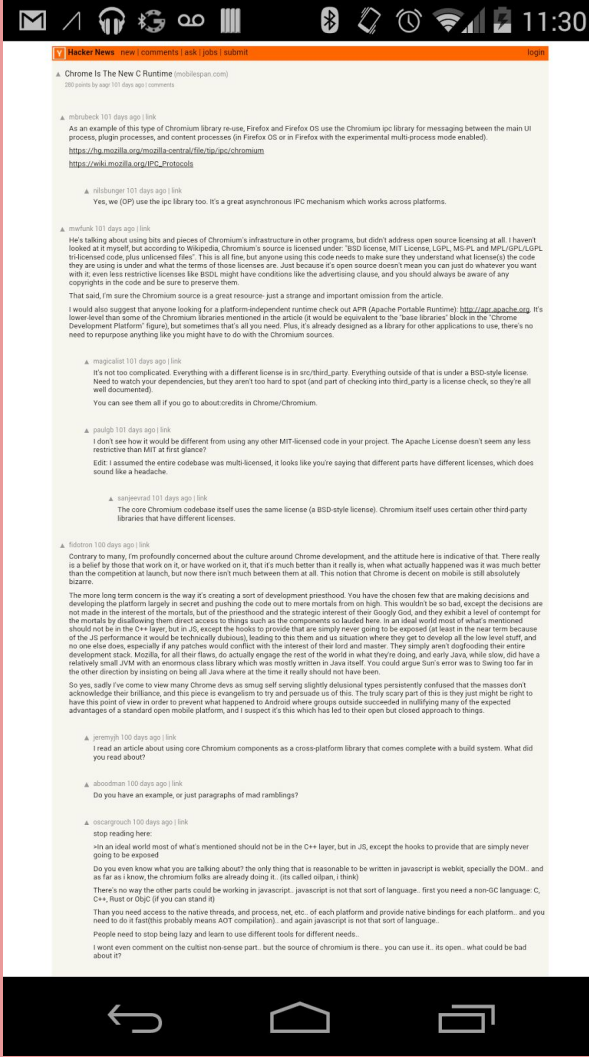
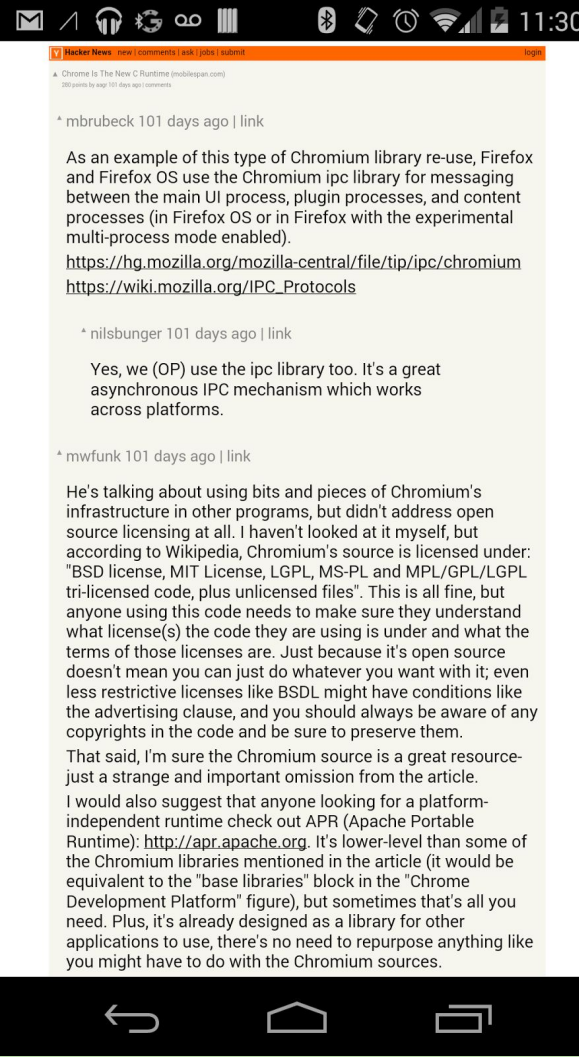
Fingerprint example

1. `body`
2. `body>div,color(purple),font(italic)`
3. `body>div,color(green)>div,color(red)`
4. `body>div,color(green)>div,color(red)`
5. `body>div,color(purple),font(small)`

Blocks 3 and 4 share a fingerprint. If these blocks end up creating clusters, they will also share a supercluster.

Below are some screenshots and timing statistics of forum websites Reddit and Hacker News rendered in Chrome for Android, showing improved consistency and performance with the new TextAutosizer.



	
<p>Robohornet Pro Benchmark: 5.44s</p>	<p>Robohornet Pro Benchmark: 4.94s</p>
<p>Wikipedia ww2 desktop layout: 4013ms</p>	<p>Wikipedia ww2 desktop layout: 2393ms</p>
<p>Reddit.com layout: 375ms</p>	<p>Reddit.com layout: 261ms</p>

Related Links

Background:

- [Seeing the world through high DPI](#) (video) - John Mellor, Google I/O 2013
- [A tale of two viewports](#) - explains layout viewports vs. visual viewports
- [An intro to meta viewport and @viewport](#) - some nice examples from Opera

Other browsers' implementations of text autosizing:

- [Mozilla's implementation](#) with two good writeups:
 - <http://dbaron.org/log/20111126-font-inflation>
 - <http://jwir3.wordpress.com/2012/07/30/font-inflation-fennec-and-you>
- [Apple's implementation](#) (two-pass, similar to Chrome's original implementation)

Fingerprinting:

- [Tackling Text Autosizing of forums](#) - original writeup on ancestry hashing by Tim Volodine