Interaction to Next Paint (INP) becomes a Core Web Vital on March 12. Start making your websites more responsive to user input today. Learn how.
(https://web.dev/blog/inp-cwv-march-12?utm_source=web.dev&utm_medium=banner&utm_campaign=inp-cwv)

First Contentful Paint (FCP)

Philip Walton
(https://twitter.com/philwalton) (https://github.com/philipwalton) (https://philipwalton.com/)

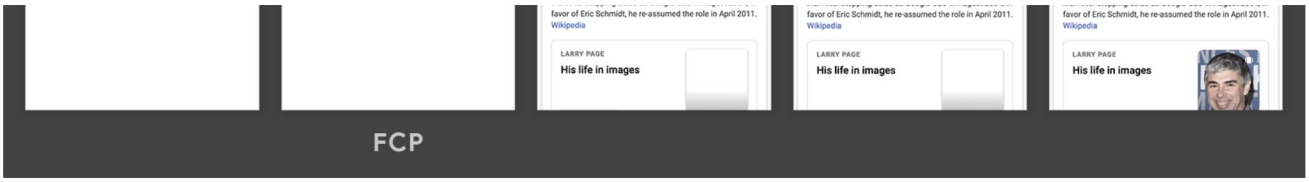
Browser Support				
	60	79	84	14.1

Source (https://developer.mozilla.org/docs/Web/API/PerformancePaintTiming)

First Contentful Paint (FCP) is an important, user-centric metric for measuring perceived load speed (/articles/user-centric-performance-metrics#types_of_metrics). It marks the first point in the page load timeline where the user can see anything on the screen. A fast FCP helps reassure the user that something is happening (/articles/user-centric-performance-metrics#defining_metrics).

FCP measures the time from when the user first navigates to the page to when any part of the page's content is rendered on the screen. For this metric, "content" refers to text, images (including background images), `<svg>` elements, or non-white `<canvas>` elements.

Key Point: FCP includes any unload time from the previous page, connection setup time, redirect time, and Time To First Byte (TTFB) (/articles/ttfb)], which can be significant when measured in the field and can lead to differences between field and lab measurements.

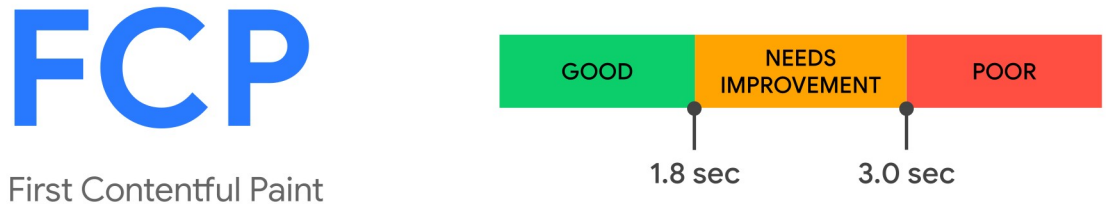


In this load timeline, FCP happens in the second frame, because that's when the first text and image elements are rendered to the screen.

Not all the content renders when the first content element renders. This is an important distinction to make between FCP and Largest Contentful Paint (LCP) (/articles/lcp), which measures when the page's main contents have finished loading.

What is a good FCP score?

To provide a good user experience, sites must have an FCP of 1.8 seconds or less. To ensure that you're hitting this target for most of your users, a good threshold to measure is the 75th percentile of page loads, segmented across mobile and desktop devices.



Good FCP values are 1.8 seconds or less. Poor values are greater than 3.0 seconds.

How to measure FCP

FCP can be measured in the lab (/articles/user-centric-performance-metrics#in_the_lab) or in the field (/articles/user-centric-performance-metrics#in_the_field), and it's available in the following tools:

Field tools

- PageSpeed Insights (https://pagespeed.web.dev/)
- Chrome User Experience Report (https://developer.chrome.com/docs/crux/)

Lab tools

- Lighthouse (https://developer.chrome.com/docs/lighthouse/overview/)
- Chrome DevTools (https://developer.chrome.com/docs/devtools/)
- PageSpeed Insights (https://pagespeed.web.dev/)

Measure FCP in JavaScript

To measure FCP in JavaScript, use the Paint Timing API (https://w3c.github.io/paint-timing/). The following example shows how to create a `PerformanceObserver` (https://developer.mozilla.org/docs/Web/API/PerformanceObserver) that listens for a `paint` entry with the name `first-contentful-paint` and logs it to the console.

```
new PerformanceObserver((entryList) => {
  for (const entry of entryList.getEntriesByName('first-contentful-paint')) {
    console.log('FCP candidate:', entry.startTime, entry);
  }
}).observe({type: 'paint', buffered: true});
```

Warning: This code shows how to log the `first-contentful-paint` entry to the console. Measuring FCP in JavaScript is more complicated.

In this example, the logged `first-contentful-paint` entry tells you when the first contentful element was painted. However, in some cases this entry isn't valid for measuring FCP.

The following section lists the differences between what the API reports and how the metric is calculated.

Differences between the metric and the API

- The API dispatches a `first-contentful-paint` entry for pages loaded in a background tab, but those pages should be ignored when calculating FCP. First paint timings are considered only if the page was in the foreground the entire time.
- The API doesn't report `first-contentful-paint` entries when the page is restored from the back/forward cache (/articles/bfcache#impact_on_core_web_vitals), but FCP should be

to the parent frame for aggregation.

- The API measures FCP from navigation start, but for prerendered pages (https://developer.chrome.com/docs/web-platform/prerender-pages), FCP should be measured from `activationStart` (https://developer.mozilla.org/docs/Web/API/PerformanceNavigationTiming/activationStart) because that corresponds to the FCP time as experienced by the user.

Instead of memorizing all these subtle differences, developers can use the `web-vitals` JavaScript library (https://github.com/GoogleChrome/web-vitals) to measure FCP, which handles these differences for you where possible (except in iframes):

```
import {onFCP} from 'web-vitals';

// Measure and log FCP as soon as it's available.
onFCP(console.log);
```

Refer to the source code for `onFCP()` (https://github.com/GoogleChrome/web-vitals/blob/main/src/onFCP.ts) for a complete example of how to measure FCP in JavaScript.

Note: In some cases, such as cross-origin iframes, you can't measure FCP in JavaScript. For details, refer to the limitations (https://github.com/GoogleChrome/web-vitals#limitations) section of the `web-vitals` library.

How to improve FCP

To learn to improve FCP for a specific site, you can run a Lighthouse performance audit and pay attention to any specific opportunities (https://developer.chrome.com/docs/lighthouse/performance/#opportunities) or diagnostics (https://developer.chrome.com/docs/lighthouse/performance/#diagnostics) the audit suggests.

To learn how to improve FCP in general (for any site), refer to the following performance guides:

- Eliminate render-blocking resources (https://developer.chrome.com/docs/lighthouse/performance/render-blocking-resources/)
- Minify CSS (https://developer.chrome.com/docs/lighthouse/performance/unminified-css/)

- Preconnect to required origins (https://developer.chrome.com/docs/lighthouse/performance/uses-rel-preconnect/)
- Reduce server response times (TTFB) (/articles/ttfb)
- Avoid multiple page redirects (https://developer.chrome.com/docs/lighthouse/performance/redirects/)
- Preload key requests (https://developer.chrome.com/docs/lighthouse/performance/uses-rel-preload/)
- Avoid enormous network payloads (https://developer.chrome.com/docs/lighthouse/performance/total-byte-weight/)
- Serve static assets with an efficient cache policy (https://developer.chrome.com/docs/lighthouse/performance/uses-long-cache-ttl/)
- Avoid an excessive DOM size (https://developer.chrome.com/docs/lighthouse/performance/dom-size/)
- Minimize critical request depth (https://developer.chrome.com/docs/lighthouse/performance/critical-request-chains/)
- Ensure text remains visible during webfont load (https://developer.chrome.com/docs/lighthouse/performance/font-display/)
- Keep request counts low and transfer sizes small (https://developer.chrome.com/docs/lighthouse/performance/resource-summary/)

Changelog

Occasionally, bugs are discovered in the APIs used to measure metrics, and sometimes in the definitions of the metrics themselves. As a result, changes must sometimes be made, and these changes can show up as improvements or regressions in your internal reports and dashboards.

To help you manage this, all changes to either the implementation or definition of these metrics are surfaced in this Changelog (http://bit.ly/chrome-speed-metrics-changelog).

If you have feedback for these metrics, provide it in the web-vitals-feedback Google group (https://groups.google.com/g/web-vitals-feedback).