FAQ: Non-Accessibility tools and features can cause crashes, slowness, lockups, jank and memory bloat by turning on accessibility

A number of tools and utilities utilize platform accessibility APIs, causing performance degradation and in some cases, crashes or lockups in Chromium, which includes Chrome, Edge and Electron applications. Both engineers and users of these tools tend to be unaware of the implications of turning on accessibility support.

Video demonstration of the issue.

Why do accessibility APIs degrade performance?

The Chromium accessibility (a11y) architecture requires marshaling web content and DOM changes across multiple processes, mapping to platform-specific accessibility APIs on each operating system. Along the way, multiple copies of the data must be created, along with some complex computations. Because accessibility today's APIs were designed before browsers went to the "one-tab-per-process" model, they are synchronous, meaning that they require immediate results, and they request these results from the browser process, not the content process.

Almost everything in the content, down to the bounding rectangle of each character, must be sent across the process boundary just in case it's needed. In many cases, the information is actually not needed at all.

This requires a lot of memory, CPU cycles, and battery.

The stability of Chromium is not usually affected, but as it runs additional code paths, it can cause additional crashes to surface.

What are some examples of non-a11y tools that do this?

These are tools that we think might turn on accessibility. The list is by no means complete. Please help us add/correct tools if you have better information:

1. Mac tools to help move windows via keyboard commands:
    - Rectangle
    - Magnet
    - Moom
    - Divvy

    - uBar
    - Yabai
    - Witch
2. Form fillers / password managers
    - TextExpander (https://textexpander.com/)
    - Hyland OnBase application enabler
    - LastPass
    - 1Password
    - Microsoft Authenticator
3. Screen clickers (for task automation)
    - Auto clicker
4. Antivirus
    - AVG
    - Avast
5. Call Recorder for Android
6. Windows Text Cursor Indicator
7. ManicTime (https://www.manictime.com/) - see https://crbug.com/1430958
8. Grammarly

Is accessibility on in my Chromium-based browser?

Visit chrome://accessibility in your address bar and see if any boxes are checked. If anything is checked, accessibility is on and the browser is performing extra work.

How can I fix this on my local machine?

Option 1. Run Chrome/Edge/etc. with --disable-renderer-accessibility.

Manually disabling accessibility this way may have an impact on some of the tools you are using, but should improve performance and memory usage.
Once accessibility is been disabled, it will not come back on. You should not see any boxes checked in chrome://accessibility.

Simply unchecking the “Web accessibility” box at chrome://accessibility is not a permanent change and accessibility features may get turned on again later.

Option 2. Run Chrome/Edge/etc. with –force-renderer-accessibility=form-controls

Using --force-renderer-accessibility=form-controls.

This limits the amount of data we put into the accessible tree to only what password managers care about (form controls and text), making it more performant and memory efficient.

The browser will still be compatible with password managers but incompatible with assistive technology such as screen readers.

Making the command line flag permanent

To permanently force these modes, you can add these command line flag to your browser's launcher on your desktop.

On Mac:

1. Quit Google Chrome first if you currently have it opened.
2. Do one of the following to open a terminal
    a. Click the Launchpad icon in the Dock, type Terminal in the search field, then click Terminal.
    b. In Finder , open the /Applications/Utilities folder, then double-click Terminal.
3. /Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --disable-renderer-accessibility

On Windows:

1. <Please add details here>

What's next?

Please note that manually disabling accessibility may have an impact on some tools you're using. If you notice any tool to stop working, please provide details at <where to send users>?

Is accessibility a setting? Or, what is it?

Accessibility is not a setting. It's more like a mode that is turned on when needed. When the browser automatically detects that a running piece of software, perhaps a tool or utility of some kind, needs information about what's happening in the Chrome window, they can use the platform's accessibility API. Once the tool does this, Chrome wakes up all the accessibility code, and continually updates the main browser process with almost everything going on inside of content.

When you peek at chrome://accessibility to see if accessibility is on, you are not seeing a setting. You are checking to see if the need for accessibility was auto detected, because a tool on the system asked for it.

When you run your browser with --disable-renderer-accessibility, you prevent the auto detection from even happening in the first place, and accessibility won't run no matter what.

Can accessibility be turned off after it's used?

Only by restarting the browser, or manually via chrome://accessibility. The browser itself will not automatically do this (yet). We'd like to fix that -- see mitigations at the bottom of this FAQ.

Can a11y be used partially, just enough for the given tool?

With the exception of screen readers, most tools that use accessibility don't need the vast majority of information being computed and copied between processes. A natural question is, whether we can reduce the information to just what's needed.

Unfortunately, we don't currently have a great way of automatically detecting what kind of software is asking to use the accessibility API, or what information will be requested by it, so currently this mode must be turned on manually via –force-renderer-accessibility=form-controls.

What's being done to mitigate the issue?

At some point in the future, you shouldn't need to run with --disable-renderer-accessibility.

For those interested in the technical details, here are some proposed mitigations that are either being worked on or discussed:

1. Reduce jank/lockups by breaking up accessibility serialization (make it interruptible) [Not being pursued because of concerns about atomicity of changes, and potentially causing flakiness]
2. Mitigate Perf Impact of accessibility in non-screenreader scenarios. Add additional AXModes for a11y tools that only need a partial accessibility tree, e.g. only send focusable objects to a password manager [Work underway]
3. Auto-disable accessibility when not in use (will ship in 99, available as a flag now)
4. Move Accessibility deserialization off main thread
5. Experiments with asynchronous accessibility APIs (this could help greatly reduce what information needs to be marshaled between processes). This will only help with new ATs.
6. Move more computations out of Blink to browser, where they only need to be computed on demand. Helps a lot with text editing, will help screen reader users who currently can get long delay between announcements while navigating document. Work underway.
7. Use profiling to find problematic areas in SendPendingAccessibilityEvents and elsewhere
8. Coalesce the event/dirty-object queue. Work to serialize fewer nodes overall — for example, we currently serialize the common ancestor of all changes .. is that the most performant option?
9. Can we avoid turning on web contents accessibility if a tool is only accessing chrome UI, like the address bar?
10. Add force accessibility off as a Chrome flag, setting, or enterprise option