DeviceAtlas®

The complete guide to User-Agents

What they are, how to read them, and how to use them

Contents

What is a User-Agent?

A User-Agent (also known as UA string) is an alphanumeric string that identifies the 'agent' or program making a request to a web server for an asset such as a document, image or web page. It is a standard part of web architecture and is passed by all web requests in the HTTP headers.

The User-Agent string is very useful because it gives you information about the software and hardware running on the device making the request. You can make important decisions on how to handle web traffic based on the User-Agent string, ranging from simple segmentation and redirection, to more complex content adaptation and device targeting decisions.

Even more information, such as screen resolution, CPU and storage capacity can be returned when the User-Agent string is mapped to an additional set of data, returned in real-time.

The User-Agent string is one element in the set of HTTP Headers, which form the handshaking process between the browser and the web server. These consist of request headers and response headers. Other request headers which are used to understand the user device or context include the Accept Header, which identifies the language and locale setting of the browser. This allows the web server to know the end user's preferred language, so if content is available in this language it can be served by default.

Anatomy of a User-Agent

Use of the User-Agent string is specified in the original HTTP standard, RFC 1945.

The User-Agent string has been part of the HTTP standard since the very first version, and has been retained in every update since, right up to HTTP/2. These standards make recommendations on what should be in the User-Agent string as well as describing its purpose"

The "User-Agent" header field contains information about the User-Agent originating the request, which is often used by servers to help identify the scope of reported

interoperability problems, to work around or tailor responses to avoid particular User-Agent limitations, and for analytics regarding browser or operating system use.
A User-Agent SHOULD send a User-Agent field in each request unless specifically configured not to do so.

How it is constructed is defined to a degree:

> User-Agent = product *(RWS (product / comment))

Product tokens are explained in more detail as:

> The User-Agent field-value consists of one or more product identifiers, each followed by zero or more comments (Section 3.2 of [RFC7230]), which together identify the User-Agent software and its significant subproducts. By convention, the product identifiers are listed in decreasing order of their significance for identifying the User-Agent software.

Each product identifier consists of a name and optional version.

Product tokens are used to allow communicating applications to identify themselves by software name and version. Most fields using product tokens also allow sub-products which form a significant part of the application to be listed, separated by white space.

Each product token includes a product name and its version separated by a "/" sign with some optional information in brackets. The tokens are typically listed by significance, however this is completely left up to the software publisher. Tokens can be used to send browser-specific information and to acquire device specific information from the device's ROM, such as the model ID, operating system and its version, etc.

Here are two examples of User-Agents used by a Samsung Galaxy S22 and a macOS computer using the Safari browser:

```
Mozilla/5.0 (Linux; Android
12; SM-S9010 Build/
QP1A.190711.020; wv)
AppleWebKit/537.36 (KHTML,
like Gecko) Version/4.0
Chrome/80.0.3987.119 Mobile
Safari/537.36
```

```
Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_15_7)
AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/16.2
Safari/605.1.15
```

The Web & Apps Landscape

At the birth of the web in the early 1990s the only viable web clients ran on personal computers. Mobile devices followed around a decade later. Today you can access the web on a wide variety of hardware types including smart watches, VR headsets, smart speakers, games consoles, televisions and refrigerators. Adding to the richness of the web is the fact that a lot of web pages are now accessed from applications ("apps") that are not web browsers as such, at least not in the normal sense—think of messaging clients, social networking apps and so on.

This richness has led to an explosion in the information that can be conveyed in the User-Agent string. Consider the example User-Agent string mentioned in RFC1945, the specification for the first version of the web:

```
CERN-LineMode/2.15
libwww/2.17b3
```

This says that the browser in question is CERN's LineMode browser, built on a library called libwww. This was an adequately descriptive User-Agent string for the day—there was simply less to be said as there were only a few web browsers in existence and just one device type that could run them—the PC.

But in today's world wide web, plied by all manner of device types, running thousands of different web-capable applications, it's no longer sufficiently descriptive to enable the use cases of the User-Agent string as envisaged by Tim Berners Lee:

- for statistical purposes
- tracing of protocol violations
- automated recognition of User-Agents for the sake of tailoring responses to avoid particular User-Agent limitations

A typical User-Agent string from a mobile device today might look like this:

```
Mozilla/5.0 (iPhone; CPU
iPhone OS 14_5 like Mac OS
X) AppleWebKit/605.1.15
(KHTML, like Gecko)
Mobile/15E148
MicroMessenger/7.0.18
(0x17001220) NetType/4G
Language/en
```

This is the WeChat "super app" running on an iPhone. The level of detail in this User-Agent string is a reflection of the fact that there are far more considerations for publishers on today's web than there were in 1990. To enable a given URL

to work on a vast range of device types, screen sizes, input methods and connectivity levels sometimes requires content tailoring, and that tailoring needs to be informed by data.

User-Agent Client Hints

Computer scientist Andrew Tanenbaum once quipped that "The nice thing about standards is that you have so many to choose from."

Unfortunately we are now in this position with the User-Agent header. Enshrined in open standards since the early nineties, the User-Agent header is now being buffeted by change in the form of a proposal from Google. The proposal is called User-Agent Client Hints (UA-CH) and is the biggest change to the User-Agent header since the dawn of the web.

UA-CH is a proposal to reduce the amount of information conveyed by the User-Agent string and instead utilize of a set of optional Client Hints headers, the most detailed of which need to be explicitly requested by the web server in order to be sent by the browser rather than in every request, as is the case with the UA. The stated purpose of the UA-CH proposal is to reduce the prevalence of passive fingerprinting. DeviceAtlas has publicly questioned the evidence for the problem solved by the proposal, pointing to a lack of any concrete evidence for widespread passive fingerprinting on the web. Despite posing this question in 2021 there hasn't yet been a credible answer, two years later.

Status

UA-CH are now in place in Chrome releases since early 2023 (Chrome 110), Microsoft Edge and some other Chromium-based browsers. By contrast, Apple, Mozilla and Brave have not adopted the proposal. The net result is that developers now have to contend with two very different approaches to identifying clients on the web and the situation is unlikely to improve any time soon.

How User-Agent Strings Serve Us

The venerable User-Agent string is used to improve experiences on the internet every day but it is done so seamlessly that nobody notices anymore. The following sections list some of the ways that User-Agent strings are used every day on the Internet:

Content adaptation

Content adaptation is a widely practised technique to improve user experiences. Most major websites offer different experiences depending on the device used to visit them. Sometimes the differences are subtle, sometimes they are major. There are several use cases for this but all require that the server needs to be aware at HTML serving time of the type of device in use.

Device-specific pages

Many websites serve different content for mobile and desktop devices. Serving different HTML and resources allows for much richer device adaptation than is possible with responsive design, which focuses on cosmetic aspects. As an example, the number of products displayed on an e-Commerce site might vary depending on whether the user is interacting with a phone, tablet or desktop device.

For these cases, serving device-specific sites to users can be helpful.

Low-powered devices

Some sites serve different content to low-powered devices that cannot handle CPU-intensive tasks, large video or high resolution images. Such content adaptation typically uses the device model information that's integrated in the User-Agent string for this purpose.

Browser bug workarounds

Many webapps work around device and browser limitations or bugs by tailoring the code as needed.

Browser feature tailoring

It's possible for websites to tailor features to particular browsers, and this can be achieved by maintaining a list of available features for particular browsers and versions. In some cases this technique is used to improve performance by only sending browser polyfills when required.

Operating system integration

Some websites change links to OS-specific ones such as Android intent links to improve the user experience.

Download of appropriate binary executables

Websites often propose the right binary to the user by default. The right binary executable for the current user depends on a few factors such as the operating system, its version, its bitness, as well as the CPU architecture.

Vulnerability checking

Some environments will chose to inspect browser and operating system versions to protect users when there are known vulnerabilities. The User-Agent string allows this check to be performed in a web environment.

Debugging

One of the original intents of the User-Agent string was for debugging purposes. The User-Agent header allows problematic devices or browsers to be identified in server log files thus allowing the problem to be addressed.

Spam filtering & bot detection

About half of all web traffic is driven by non-human users. Many websites will chose to control this traffic by utilizing rules on their web servers. As an example, known-bad bots can be blocked from accessing content.

User login notification

Many web applications now notify the user when their credentials are used to log in on a new device. This device and browser information helps the user to decide if the login attempt matches one they made themselves.

Advertising

Advertising is the de facto micropayments model for the web, also enabling many apps to be made available in free versions. In a world where privacy regulations are constraining the targeting ability of advertisers, knowledge of a user's device can help ensure that ads are more relevant. As a very simple example, the User-Agent string can be used to ensure that only apps that run on a particular device are advertised on it, or that accessories are only advertised to devices that are relevant.

Device inventory maintenance

A challenge for IT operations departments, where BYOD policies are in place, is to ensure that staff devices are up to date. A simple means to do this is through parsing of User-Agent strings presented by employee devices when connecting to network resources. This can be done in advance of a login to allow for blocking of devices with unpatched operating systems or out of date browsers.

Subscriber whitelist maintenance

Subscribers to commercial services may wish to access them from multiple devices. By identifying the devices used by the subscriber, a whitelist of user-approved devices which are authorised to access the services can be maintained. Devices from outside of the whitelist could be subject to additional security verification (2FA for example) in order to authenticate the subscriber fully.

OpenRTB and the User-Agent header

Knowledge of the user's device has always been part of the OpenRTB specification. The standard expressly includes a ua attribute in bid requests to allow informed decisions to be made about the device in question. This allows for ads to be targeted based on the device a user is holding.

The IAB recently published an updated version of the OpenRTB specification. The updated document, a revision to version 2.6 of the specification, now addresses the landscape change brought about by Google's decision to progressively reduce the content of the User-Agent string in Chrome in favour of User-Agent Client Hints (UA-CH).

This update provides welcome clarity for the OpenRTB ecosystem since Google's change would have threatened to weaken the ability of the OpenRTB protocol to support targeting by device type and other non-PII characteristics.

With the newest update, the OpenRTB specification now explicitly defines how to populate the ua attribute and the sua attribute in the case where a browser supports User-Agent Client Hints:

> ***"For backwards compatibility, exchanges are recommended to always populate ua with the User-Agent string, when available from the end user's device, even if an alternative representation, such as the User-Agent Client-Hints, is available and is used to populate sua. No inferred or approximated User-Agents are expected in this field.***
>
> ***If a client supports User-Agent Client Hints, and sua field is present, bidders are recommended to rely on sua for detecting device type, browser type and version and other purposes that rely on the User-Agent information, and ignore ua field. This is because the ua may contain a frozen or reduced User-Agent string."***

User-Agent String Evolution

Privacy Concerns

User-Agent strings are no longer the privacy issue that they're sometimes made out to be. There are two primary reasons for this:

1. The User-Agent string isn't as useful for fingerprinting as it used to be
2. Fewer entities in the ecosystem now have access to the User-Agent string

Fingerprinting

In the past there were legitimate concerns that the browser User-Agent string was a significant source of entropy for fingerprinting users. Here is an example User-Agent string from around ten years ago:

```
Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1;
SV1; Tablet PC 1.7; .NET CLR
1.0.3705; .NET CLR 1.1.4322;
InfoPath.1; Alexa Toolbar;
.NET CLR 2.0.50727)
```

In this case Internet Explorer 6 is not only saying that there are two browser plugins installed (InfoPath and the Alexa Toolbar), it also lists three separate versions of the .NET common language runtime with very granular version numbers. This is a highly specific set of information that, in combination with IP address, would make it very likely that a user is individually targettable.

Here are some more examples from the past:

- `Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506; .NET CLR 1.1.4322; InfoPath.2; Zango 10.3.75.0)`
- `Opera/9.10 (Windows NT 5.1; U; MEGAUPLOAD 1.0; pl)`

Note the Zango and Megaupload plugins and their version numbers..

This level of detail simply isn't present in mainstream browsers anymore. Let's consider three of the most common browsers today:

- **Chrome**
  `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36`

- **Safari**
  ```
  Mozilla/5.0 (Macintosh;
  Intel Mac OS X 10_15_7)
  AppleWebKit/605.1.15 (KHTML,
  like Gecko) Version/16.4.1
  Safari/605.1.15
  ```
- **Edge**
  ```
  Mozilla/5.0 (Macintosh;
  Intel Mac OS X 10_15_7)
  AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/112.0.0.0
  Safari/537.36
  Edg/112.0.1722.68
  ```

These modern User-Agent strings tell us very little apart from the browser version, operating system name and version. **Firefox** reveals even less (it omits the least significant version number):

```
Mozilla/5.0 (Macintosh;
Intel Mac OS X 10.15;
rv:109.0) Gecko/20100101
Firefox/113.0
```

Given that most modern browsers and operating systems are automatically kept up to date, these version numbers will be meaningless for fingerprinting purposes.

Access to the User-Agent string

For the User-Agent string to be used for fingerprinting, one needs access to it. Since 2010, when the Firesheep plugin exposed just how much information was freely available from sniffing session cookies on LAN traffic, the web has rapidly moved to HTTPS by default. Google now reports that over 90% of page loads in Chrome happen over HTTPS.

For the User-Agent string in particular, this means that no network intermediaries have access to the information that a person's browser emits. To all intents and purposes, this information is now impossible to obtain as a man-in-the-middle. This limits access to the User-Agent string to websites that the user chooses to connect with, and any linked third-party origins. In their role as network intermediaries, the LAN, ISP and mobile operator no longer have access to the User-Agent string. This change alone greatly limits the scope for passive fingerprinting on the web.

Making use of the User-Agent header

How does User-Agent parsing work in device detection?

From a technical point of view, examining the User-Agent is not particularly complex and can be done using navigator.userAgent in JavaScript or the HTTP User-Agent header field made available by web servers.

Many companies use a regular expression ("regex") approach to analyze User-Agents which relies on pattern or string matching to find keywords that can identify the underlying device. A typical regex approach would look for the presence of 'iPhone' or 'Android' in the User-Agent, however this can result in data inaccuracies. Being able to differentiate between Android tablets and phones is an obvious weakness and the presence of the iPhone token may be just about as useful as the Mozilla token.

As User-Agent strings do not conform to any standard pattern, this technique is prone to failure and is not future-proof. Regex rules would constantly need to be updated as new devices, browsers and operating systems are released, as well as running endless tests to confirm whether the solution is still working correctly. At some point, this becomes a costly maintenance job, and, over time, a real risk that a significant proportion of traffic is being mis-detected.

Accurately parsing User-Agents is one problem, but the real difficulty is being able to stay on top of the constantly shifting sands of the device, browser and OS market. This becomes even more difficult with millions of permutations when language and locale, or sideloaded browsers are added to the mix. More recently, User-Agent Client Hints have necessitated looking at many additional HTTP headers to identify a device where previously just the User-Agent header would have sufficed.

This is where having a good device detection solution really pays off.

There are two prerequisites for device detection:

- Header lookups happen extremely quickly
- Device identification is highly accurate

This involves accurately mapping all possible User-Agent strings for a particular device and having an API that can accurately and quickly return the information while also being flexible enough to accommodate new variants as they arise.

The reason that this can be so difficult to execute correctly is because there are millions of variants already in existence with new User-Agents being created all the time. Every new device, browser, browser version, OS or app can create a new and previously unseen User-Agent.

In this regard, not all approaches to device detection are created equal—some will have inaccurate data or return false positives—you may think you have a correct result, but an inferior solution may return default values or fallback devices for unknown User-Agents. Some approaches can also max out server resources because of unsophisticated and messy APIs and codebases.

DeviceAtlas uses a set of Patricia trie data structure to determine the properties of a device in the quickest and most efficient way. This is the reason why major companies rely on established solutions that are built on proven and patented technology.

Patented Technology

In the device intelligence world, speed is everything **but accuracy should never be sacrificed.**

Our patented algorithm allows DeviceAtlas to achieve both speed and accuracy without any tradeoffs by combining some uniquely useful characteristics:

- It is extremely fast
- It allows for perfect accuracy
- It has a very light memory and data file footprint

Whether you're running a real time bidding (RTB) platform where the entire auction process takes place in 100 milliseconds, or an analytics platform churning through trillions of requests, or a website running on a lowly VPS, speed is consistently needed.

Our algorithm allows for the extremely high speeds afforded by a Patricia trie, but with the flexibility to accommodate the reality that it's not possible to have prior knowledge of all devices on the market.

In computer science, trie structures are often used in search-like use cases such as spell checkers and predictive text—despite their apparent complexity, this approach works exceedingly well. To this end, we have incorporated some improvements into the traditional Patricia trie.

Firstly, the approach considers the User-Agent string as groups of tokens, allowing us to skip insignificant characters. Secondly, our approach does not rely on the traditional left-to-right ordering of the User-Agent string to achieve perfect results.

Benefits of User-Agent analysis

Increased Conversions - Content Optimization

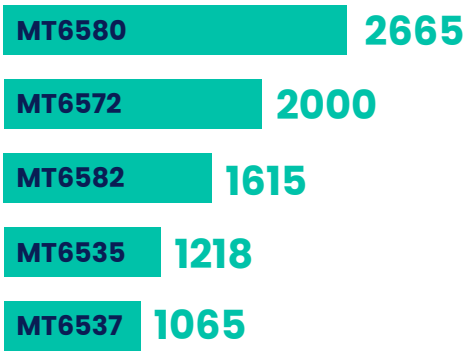
A well implemented User-Agent analysis strategy allows you to adapt content dynamically to ensure that each visitor has an optimal viewing experience. Whether the visiting device is a smartphone, tablet, desktop, a high end or low end device, getting the first impression right is critically important.

Another factor in optimizing for increased conversions is page load/ weight. By parsing a User-Agent, you can learn how big the screen is, for example, and send an appropriately sized image to the device. This will cut down on a potential customer's wait time and can also save data if they are on a metered connection.

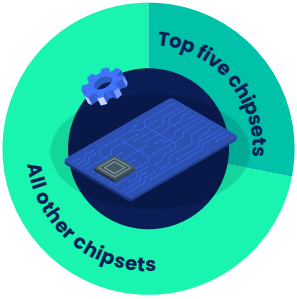
Getting the most out of User-Agent analysis helps you become fully aware of the changing patterns of device usage, which can inform content, design and business decisions.

For some examples of this content in action, view our article on adaptive web design in action, as well as a recent analysis we did on adaptive vs responsive website design in eCommerce.

Top five chipsets used...



... account for **28%** of all chipsets used!



Reporting/Analytics

After the fact reporting and analysis of User-Agent visits can also inform future decisions and strategies—this new informationΩcan shed light on the most granular of scenarios.

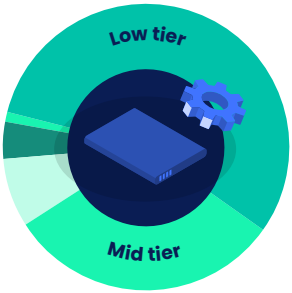
Your existing reporting may only focus on mobile/desktop/tablet, but adding hundreds of additional data points with User-Agents will provide a much closer look at individual devices. This level of granularity can offer key insights such as friction associated with specific devices navigating through your website.

Enhanced Ad-targeting

In advertising, as a rule of thumb, it's important to ensure that your ads reach the right people at the right time. Device detection is an essential ingredient in making this possible, and analyzing the User-Agent of each device allows you to ensure you reach the desired audience. Up to date device information can power campaign management interfaces so that users can create campaigns based on a wide range of device characteristics such as device type (phone, tablet, set top box and 47 other device types), device tier (entry-level to premium tier), year released and so on.

For anyone involved in the online advertising space, buyers and sellers both, ad fraud is clearly a concern. DeviceAtlas is used by major players in the ad-tech space, such as The Trade Desk, Yahoo, SpotX and Magnite.

56% of all hardware used is considered **low tier**



Almost **10%** of all phones have a screen size of..

5 inches

Bot Detection

We all realize the benefits of search bots from Google and Bing, but being able to distinguish between those and malicious bots accessing your content means that you can avoid wasting resources intended for a human audience. Treating good and bad bots differently once identified by their User-Agent allows you to benefit from good bots, while also reducing the potential resource drain caused by the malicious variety.

Based on analysis of the HTTP headers, good bots can be identified since they self-declare in the User-Agent string. However, some User Agents seek to mask their identity by using a generic or different User-Agent string. In order to identify these scenarios, it is necessary to look at additional signals such as those obtained via a JavaScript library.

Approaches to User-Agent parsing

There are many approaches to availing of the information available from the User-Agent header, and the following sections discuss some of the options available:

Building a regex device detection solution

In some cases it is feasible to build a device detection solution based on regex, which is essentially a pattern matching scheme utilizing a number of well-known mobile browser User-Agent string snippets. The use cases of this approach might include websites where simple mobile or tablet redirection is sufficient, or where highly precise detection of properties is not really important.

A regex-based device detection can be built with different programming languages. As referenced below in the case of Adjust, there are some limitations that you should be aware of before exploring this option further:

Accuracy – Regex User-Agent matching can work well in the general sense, but it will inevitably fail to recognize some devices correctly, e.g. certain Android tablets. While it's possible to add more specific patterns to the regex to match edge-cases by matching specific model numbers, this has the disadvantage of reducing the performance of the detection, and will have knock-on performance implications for your site.

Performance – Regular expressions can be slow to execute, particularly for complicated patterns. If performance is a key factor, then this is not the way to go.

Maintenance – The regex patterns will need to be updated regularly to keep up to date with new devices that may not be already covered.

Device capabilities – If you need anything more than simple traffic routing (mobile/tablet/desktop)

such as knowing device properties like screen size, memory limit, HTML5 support, then the regex solution may not be suitable.

We spoke with one of our customers, Adjust, in January 2023 about their decision to move away from in-house User-Agent parsing using a regex solution:

"We were writing a lot of it [code] on our own, which meant that we had a long and perpetually outdated list of Regexes to look at the User-Agents and try to make sense of them. [...] We had a very tough time distinguishing between Android tablets and phones, which is kind of important for us because we provide this data to clients."

Locally deployed and cloud-based device detection

Choosing a commercial device detection solution might be a better choice than a home-grown solution, especially for larger, high-trafficked websites that implement content adaptations use cases such as advertising and analytics where very high throughputs and low latency are required.

There are two options for third-party device detection solution:

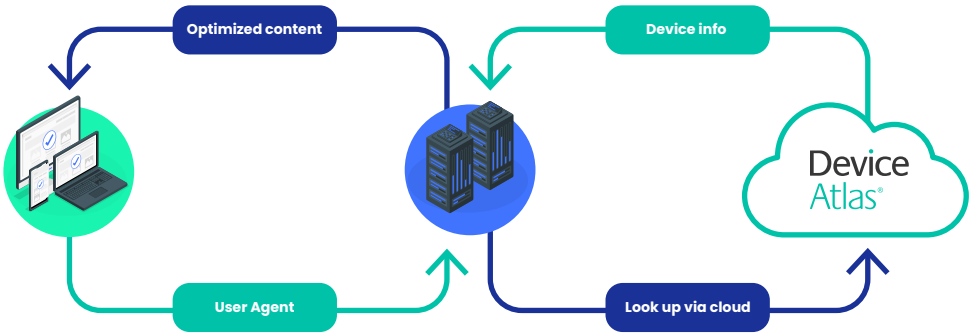
1. Cloud-based
2. Locally-deployed

With cloud-based detection, data is delivered on demand for specific device headers submitted via a Cloud API. To integrate a cloud-based detection on your website you would need to download the API and insert a code snippet into your website's code. The third-party service then adds the capability of identifying and handling traffic from any device category to your website via an up-to-date database of all the latest devices.

Cloud-based device detection is easier to implement and maintain than a home-grown solution due to the fact that no manual updates are required. An up-to-date, third-party device database also means a higher level of accuracy.

One example of this type of solution is the DeviceAtlas cloud-based detection service which is based on API calls to the DeviceAtlas servers. You can check implementation examples in different coding environments here.

The basic version of DeviceAtlas cloud-based detection is available to try for free.

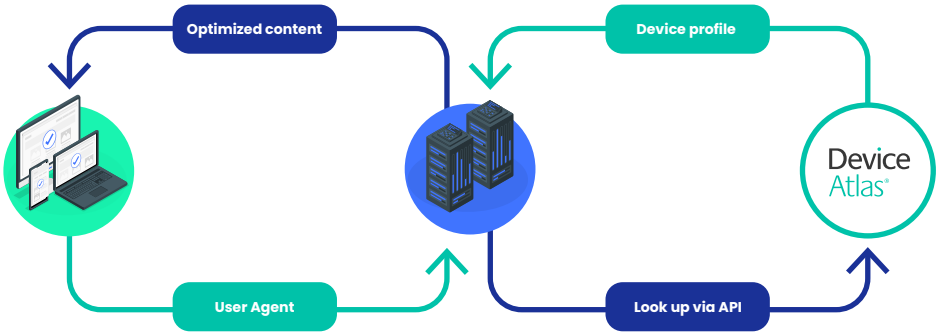


Locally-installed device detection

Some of the largest websites preclude dependency on any third-party services, and thus deploy only locally installed device detection solutions.

To implement locally-installed detection you must download a device data file from your solution provider and deploy their API into your environment. It is best to set automatic, script-based downloads and updates of the file to ensure that the most up-to-date data is in use.

In DeviceAtlas's case, the device data is available in a highly compressed JSON format offering extremely fast lookups with a minimal footprint, and can be downloaded manually or obtained via an automated script.



Examples of common User-Agents

There are millions of combinations given which User-Agents can change depending on the software and hardware. For example, a Chrome browser on an iPhone 14 will introduce itself using a different User-Agent than a Safari browser on the same phone.

Every device type, including phones, tablets, desktops, may come with its own User-Agent, making it possible to detect for any purpose. Interestingly, bots and crawlers also come with their own unique User-Agents. Below is a list of User-Agents for different device types that can be detected.

If you would like to learn more about any of these devices, just copy and paste the string into our User-Agent testing tool and you'll find all the properties of that detected device.

Index of common User-Agents

Android Mobile User-Agents

Device	User-Agent
Samsung Galaxy S23	Mozilla/5.0 (Linux; Android 13; SM-S911B) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Mobile Safari/537.36 Dalvik/2.1.0 (Linux; U; Android 13; SM-S911B Build/TP1A.220624.014)
Samsung Galaxy S21 Ultra 5G	Mozilla/5.0 (Linux; Android 11; SM-G998U) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.93 Mobile Safari/537.36 Dalvik/2.1.0 (Linux; U; Android 12; SM-G998U Build/ SP1A.210812.016)
Samsung Galaxy Z Flip4	Mozilla/5.0 (Linux; Android 12; SM-F721U Build/SP2A.220305.013; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/103.0.5060.71 Mobile Safari/537.36 Mozilla/5.0 (Linux; Android 12; SM-F721U) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.0.0 Mobile Safari/537.36
Samsung Galaxy Note 20	Mozilla/5.0 (Linux; Android 10; SM-N980F Build/QP1A.190711.020; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/80.0.3987.119 Mobile Safari/537.36 Dalvik/2.1.0 (Linux; U; Android 10; SM-N980F Build/QP1A.190711.020)
Google Pixel 7	Mozilla/5.0 (Linux; Android 13; Pixel 7 Build/TD1A.220804.009. A2; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/103.0.5060.71 Mobile Safari/537.36 Dalvik/2.1.0 (Linux; U; Android 13; Pixel 7 Build/TD1A.220804.009. A2)
Sony Xperia 5 IV	Mozilla/5.0 (Linux; Android 12; XQ-CQ62 Build/64.0.H.11.9; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/107.0.5304.141 Mobile Safari/537.36 Mozilla/5.0 (Linux; Android 12; XQ-CQ62) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Mobile Safari/537.36
HTC Desire 22 pro	Mozilla/5.0 (Linux; Android 12; HTC Desire 22 pro Build/ SKQ1.220201.001; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/103.0.5060.129 Mobile Safari/537.36 Mozilla/5.0 (Linux; Android 12; HTC Desire 22 pro) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Mobile Safari/537.36

iPhone User-Agents

Below are some examples of User-Agent strings used by the most recent iPhone devices. As Apple does not pass much info through the User-Agent, version numbers don't allow us to differentiate between iPhone models.

However, with the DeviceAtlas client-side component component, these User-Agents can be classified and the correct device model returned.

Device	User-Agent
iPhone 14 Pro Max	Mozilla/5.0 (iPhone15,3; U; CPU iPhone OS 16_0 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/20A362 Safari/602.1
iPhone 14 Pro	Mozilla/5.0 (iPhone15,2; U; CPU iPhone OS 16_0 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/20A362 Safari/602.1
iPhone 14 Plus	Mozilla/5.0 (iPhone14,8; U; CPU iPhone OS 16_0 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/20A362 Safari/602.1
iPhone 14	Mozilla/5.0 (iPhone14,7; U; CPU iPhone OS 16_0 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/20A362 Safari/602.1
iPhone 13 Pro Max	Mozilla/5.0 (iPhone14,3; U; CPU iPhone OS 15_0 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/19A346 Safari/602.1
iPhone 12	Mozilla/5.0 (iPhone13,2; U; CPU iPhone OS 14_0 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/15E148 Safari/602.1
iPhone 12 Mini	Mozilla/5.0 (iPhone13,1; U; CPU iPhone OS 14_0 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/15E148 Safari/602.1
iPhone 11	Mozilla/5.0 (iPhone12,1; U; CPU iPhone OS 13_0 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/15E148 Safari/602.1

Desktop User-Agents

Device	User-Agent
Windows 10-based PC using Edge browser	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 Edge/12.246
Chrome OS-based laptop using Chrome browser (Chromebook)	Mozilla/5.0 (X11; CrOS x86_64 8172.45.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.64 Safari/537.36
Mac OS X-based computer using a Safari browser	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9
Windows 7-based PC using a Chrome browser	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111 Safari/537.36
Linux-based PC using a Firefox browser	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:15.0) Gecko/20100101 Firefox/15.0.1

Set Top Box User-Agents

Device	User-Agent
Apple TV (2022)	AppleTV14,1/16.1
Chromecast with Google TV (4K)	Mozilla/5.0 (X11; Linux armv7l) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.90 Safari/537.36 CrKey/1.17.46278 Mozilla/5.0 (X11; Linux aarch64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.225 Safari/537.36 CrKey/1.56.500000 DeviceType/Chromecast
Minix NEO X39	Mozilla/5.0 (Linux; Android 7.1.2; NEO_X39) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.99 Safari/537.36
Amazon Fire TV Stick 4K Max	Mozilla/5.0 (Linux; Android 9; AFTKA) AppleWebKit/537.36 (KHTML, like Gecko) Silk/92.2.11 like Chrome/92.0.4515.159 Safari/537.36
Amazon Fire TV Cube	Mozilla/5.0 (Linux; Android 9; AFTR) AppleWebKit/537.36 (KHTML, like Gecko) Silk/98.6.10 like Chrome/98.0.4758.136 Safari/537.36
Chromecast	Mozilla/5.0 (CrKey armv7l 1.5.16041) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.0 Safari/537.36
Roku Ultra	Roku 4640X/DVP-7.70 (297.70E04154A)
Minix NEO X5	Mozilla/5.0 (Linux; U; Android 4.2.2; he-il; NEO-X5-116A Build/JDQ39) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Safari/534.30
Amazon 4K Fire TV	Mozilla/5.0 (Linux; Android 5.1; AFTS Build/LMY47O) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/41.99900.2250.0242 Safari/537.36
Google Nexus Player	Dalvik/2.1.0 (Linux; U; Android 6.0.1; Nexus Player Build/MMB29T)
Apple TV 5th Gen 4K	AppleTV6,2/11.1
Apple TV 4th Gen	AppleTV5,3/9.1.1

Bots and Crawlers User-Agents

For a more in-depth list of User-Agent strings related to web crawlers and bots, check out this article.

Device	User-Agent
Google Bot	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Bing Bot	Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)
Yahoo! Bot	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)

Game Consoles User-Agents

Device	User-Agent
Sony Playstation 5	Mozilla/5.0 (PlayStation; PlayStation 5/2.26) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.0 Safari/605.1.15
Xbox Series X	Mozilla/5.0 (Windows NT 10.0; Win64; x64; Xbox; Xbox Series X) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.82 Safari/537.36 Edge/20.02
Nintendo Wii U	Mozilla/5.0 (Nintendo WiiU) AppleWebKit/536.30 (KHTML, like Gecko) NX/3.0.4.2.12 NintendoBrowser/4.3.1.11264.US
Xbox One S	Mozilla/5.0 (Windows NT 10.0; Win64; x64; XBOX_ONE_ED) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.79 Safari/537.36 Edge/14.14393
Xbox One	Mozilla/5.0 (Windows Phone 10.0; Android 4.2.1; Xbox; Xbox One) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2486.0 Mobile Safari/537.36 Edge/13.10586
Playstation 4	Mozilla/5.0 (PlayStation 4 3.11) AppleWebKit/537.73 (KHTML, like Gecko)
Playstation Vita	Mozilla/5.0 (PlayStation Vita 3.61) AppleWebKit/537.73 (KHTML, like Gecko) Silk/3.2
Nintendo 3DS	Mozilla/5.0 (Nintendo 3DS; U; ; en) Version/1.7412.EU

Tablet User-Agents

Device	User-Agent
iPad Air (2020)	Mozilla/5.0 (iPad13,1; iPad; U; CPU OS 14 like Mac OS X) AppleWebKit/602.2.14 (KHTML, like Gecko) Mobile/16E227
Samsung Galaxy Tab S6 5G	Mozilla/5.0 (Linux; Android 9; SAMSUNG SM-T866N) AppleWebKit/537.36 (KHTML, like Gecko) SamsungBrowser/11.1 Chrome/75.0.3770.143 Safari/537.36
Samsung Galaxy Tab A7 10.4	Mozilla/5.0 (Linux; Android 12; SM-T509) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36
Sony Xperia Z4 Tablet LTE	Mozilla/5.0 (Linux; Android 7.0; SGP771 Build/32.3.A.2.33; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/58.0.3029.83 Safari/537.36
Fire HD 10 Plus	Mozilla/5.0 (Linux; Android 9; KFTRPWI) AppleWebKit/537.36 (KHTML, like Gecko) Silk/92.2.11 like Chrome/92.0.4515.159 Safari/537.36
Lenovo Tab P11	Dalvik/2.1.0 (Linux; U; Android 10; Lenovo TB-J606F Build/ QKQ1.200730.002)
iPad Pro 12.9 (2022)	Mozilla/5.0 (iPad14,5; U; CPU OS 16_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/11.3 Mobile/20B82 Safari/602.1
Google Pixel C	Mozilla/5.0 (Linux; Android 7.0; Pixel C Build/NRD90M; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/52.0.2743.98 Safari/537.36
Sony Xperia Z4 Tablet	Mozilla/5.0 (Linux; Android 6.0.1; SGP771 Build/32.2.A.0.253; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/52.0.2743.98 Safari/537.36
Nvidia Shield Tablet K1	Mozilla/5.0 (Linux; Android 6.0.1; SHIELD Tablet K1 Build/MRA58K; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/55.0.2883.91 Safari/537.36
Samsung Galaxy Tab S3	Mozilla/5.0 (Linux; Android 7.0; SM-T827R4 Build/NRD90M) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.116 Safari/537.36

Tablet
User-Agents (Continued)

Device	User-Agent
Samsung Galaxy Tab A	Mozilla/5.0 (Linux; Android 5.0.2; SAMSUNG SM-T550 Build/LRX22G) AppleWebKit/537.36 (KHTML, like Gecko) SamsungBrowser/3.3 Chrome/38.0.2125.102 Safari/537.36
Amazon Kindle Fire HDX 7	Mozilla/5.0 (Linux; Android 4.4.3; KFTHWI Build/KTU84M) AppleWebKit/537.36 (KHTML, like Gecko) Silk/47.1.79 like Chrome/47.0.2526.80 Safari/537.36
LG G Pad 7.0	Mozilla/5.0 (Linux; Android 5.0.2; LG-V410/V41020c Build/ LRX22G) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/34.0.1847.118 Safari/537.36

E-Readers User-Agents

Device	User-Agent
Amazon Kindle 4	Mozilla/5.0 (X11; U; Linux armv7l like Android; en-us) AppleWebKit/531.2+ (KHTML, like Gecko) Version/5.0 Safari/533.2+ Kindle/3.0+
Amazon Kindle 3	Mozilla/5.0 (Linux; U; en-US) AppleWebKit/528.5+ (KHTML, like Gecko, Safari/528.5+) Version/4.0 Kindle/3.0 (screen 600x800; rotate)

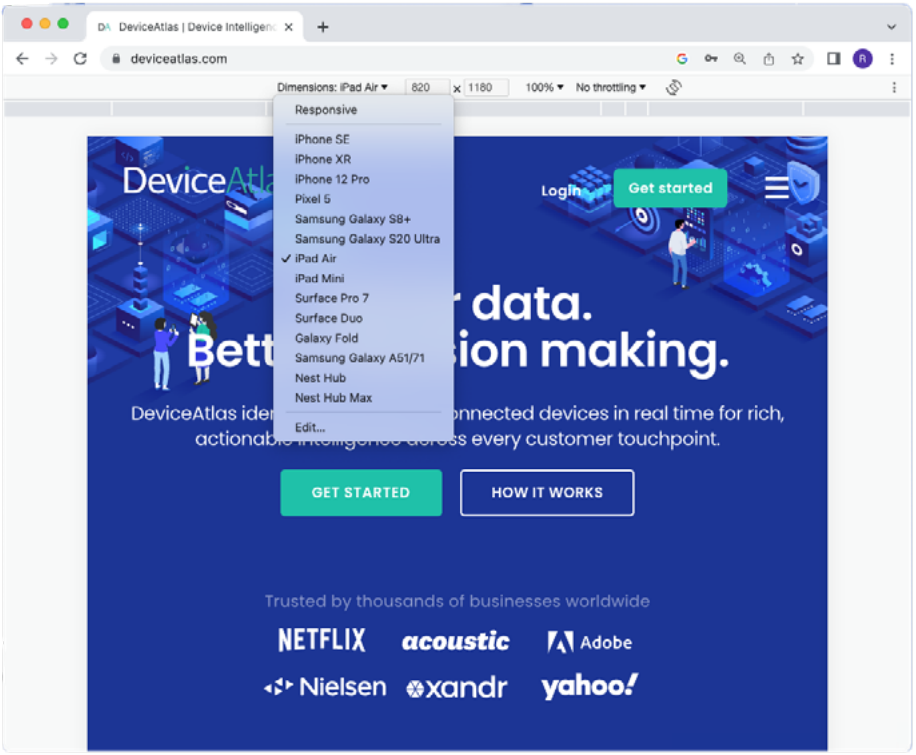
Changing your User-Agent

Looking to test mobile websites in your desktop browser?

Or, maybe you need to test page weight and load times in the mobile environment?

These tasks are easily done by changing the browser's default User-Agent header.

Click here to learn a few simple methods for switching User-Agents in desktop browsers.



Conclusion

The humble User-Agent header has been around since the dawn of the web and, despite recent landscape changes, it continues to serve us today.

At first glance, leveraging the User-Agent seems like an easy way to segment traffic and optimize your content to increase engagement on all devices. However, the tricky part lies in handling a constantly evolving set of User-Agents when new devices are identified every single day. This results in ineffective and quickly obsolete analytics and reporting, planning and optimizing, as well as a difficulty in managing costs.

For companies operating at scale that lack the resources to deal with this in-house, it's worth investing in a high performance device intelligence solution like DeviceAtlas.

Start detecting all devices accessing your content across all environments

DeviceAtlas is a high-speed, high-performance, low-server footprint device detection solution used by some of the largest companies in the online space.

The most common use-cases are:

- Optimizing UX and conversion rates for all connected devices
- Improving web performance
- Targeting ads
- Analyzing web and app traffic

DeviceAtlas allows you to target any of the 220 device properties to build fine-grained content optimization and detailed reports on web traffic. Get started with a free trail to test DeviceAtlas in your environment.

Get started

Learn more

Online: deviceatlas.com

or email: info@deviceatlas.com