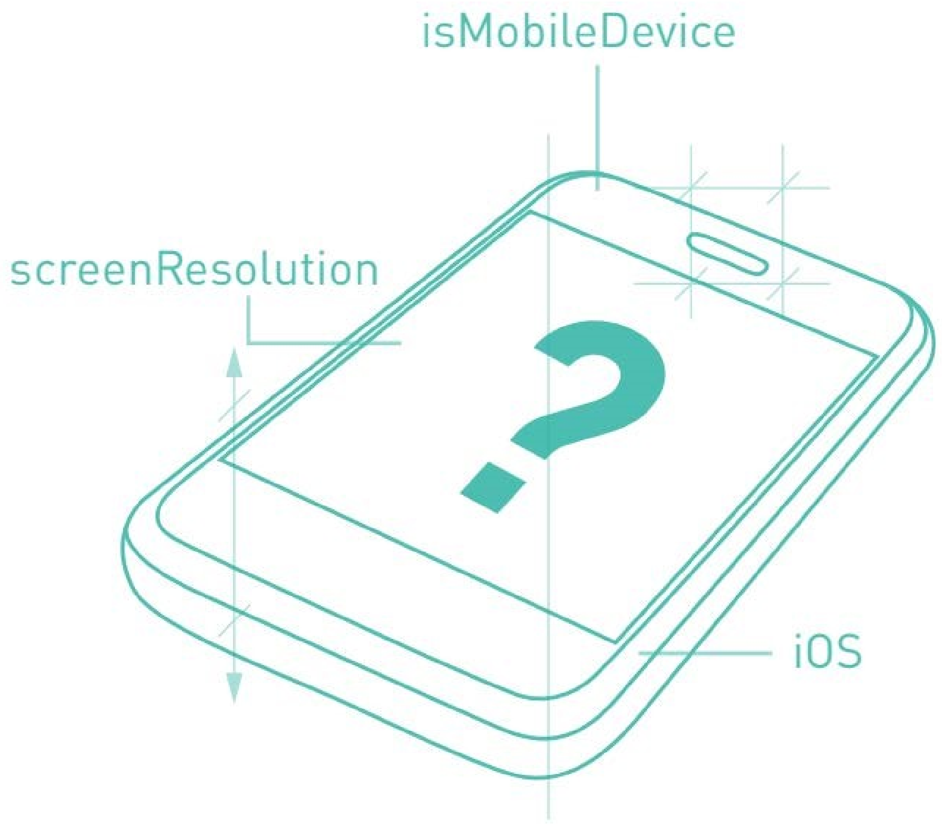
DeviceAtlas

THE COMPLETE GUIDE TO USER-AGENT STRINGS

WHAT THEY ARE, HOW TO READ THEM, AND HOW TO USE THEM.



Contents

What is a User-Agent string?

A User-Agent (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 has been a standard part of web architecture for over 25 years, ever since the specification for HTTP 1.0 was published, 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. These use cases were actually anticipated by Tim Berners Lee and the original authors of the HTTP specification in the early 1990s.

Even more information, such as device model, screen resolution, CPU and storage capacity can be returned when the User-Agent string is mapped to an additional set of data, which should of course be done in real-time. This is the advantage of using DeviceAtlas for your device intelligence requirements.

Anatomy of a User-Agent string

Use of the User-Agent string is specified in the HTTP standard as described in RFC 1945.

In fact, the User-Agent string has been part of the HTTP standard since the very first version over 25 years ago, 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 defining their 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*. [RFC 1945]

The specification defines how User-Agent strings are structured as follows:

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

> The User-Agent field-value consists of one or more product identifiers, each followed by zero or more comments, 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 "/" character with some optional information in brackets. The tokens are typically listed by significance, however this is completely left up to 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-Agent strings used by a Samsung Galaxy S8 Active and a macOS-based computer using a Safari browser:

```
Mozilla/5.0 (Linux; Android 7.0; SM-G892A Build/NRD90M; wv)
AppleWebKit/537.36 (KHTML, like Gecko)
Version/4.0 Chrome/60.0.3112.107 Mobile Safari/537.36
```

```
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
```

How does User-Agent string parsing work in device intelligence?

From a technical point of view, examining the User-Agent string is not difficult. The HTTP User-Agent header is included in every web request and is accessible in different ways depending on the programming language used. In JavaScript, you can get a User-Agent string using navigator.userAgent.

Many companies use a regular expression ("regex") approach to analyse the User-Agent header. This approach relies on pattern or string matching to identify keywords which might identify the underlying device. A typical regex approach would look for the presence of 'iPhone' or 'Android' in the User-Agent, but the accuracy concerns are many. Telling Android tablets and phones apart is an obvious weakness, and the presence of the iPhone token may be just about as useful as the Mozilla token, which is included in many User-Agent strings, even those that have nothing to do with the Mozilla Foundation's software

As User-Agent strings do not conform to any standard pattern, this technique is prone to failure and is not future proof. Furthermore, the regex technique will not detect when it fails, but simply misclassify a device silently. Also, regex rules can get complex very quickly, and complex regular expressions are notoriously difficult to maintain. Volume is also a challenge. DeviceAtlas adds 30-40 new devices every single day, so to stay current, you would need to constantly update your regex rules as new devices, browsers and OSs are released, and then run tests to see if the solution still works well. At some point, this becomes a costly maintenance job, and, over time, a real risk that you are mis-detecting or failing to detect much of your traffic. If you decide to tackle device detection yourself, you should therefore make sure you have the capacity to deal with the volume and complexity of new devices.

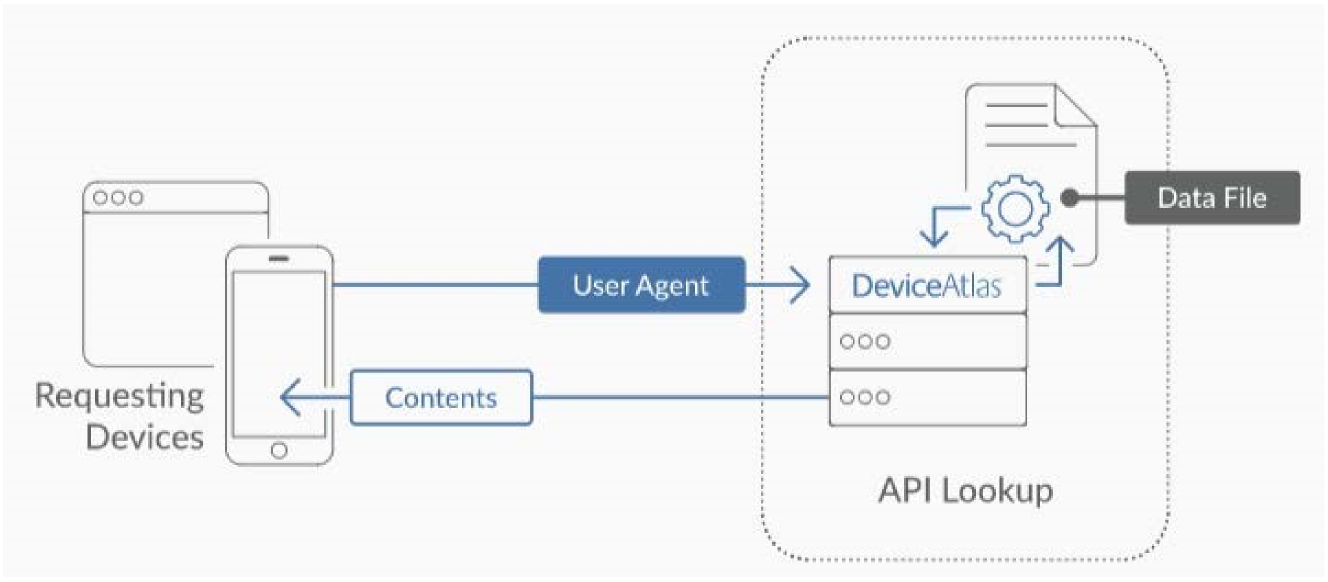
Accurately parsing User-Agent strings is one problem. The real difficulty is in staying on top of the constantly shifting sands of the device, browser, OS market with potentially millions of permutations when things like language and locale or side-loaded browsers are layered on. This is where a good device detection solution really pays off.

There are two prerequisites for device detection.

- That the User-Agent lookup happens extremely quickly and
- That the 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 being flexible enough to accommodate new variants as they arise. The reason that this is difficult is that there are millions of variants and new User-

Agent strings are being created all the time. Every new device, browser, browser version, OS or app can create new and previously unseen User-Agent string



In this regard, not all approaches to device detection are created equal—the bad ones will have inaccurate data and return false positives, meaning that you may think you are getting a correct result, but in reality you are receiving default (and useless) values for unknown User-Agent strings. Some approaches hog server resources because of their unsophisticated and messy APIs and codebases.

DeviceAtlas uses a Patricia Trie data structure to determine the properties of a device in the quickest and most efficient way. The resulting combination of speed, accuracy and memory efficiency is the reason why major companies rely on established solutions built on proven and patented technology like DeviceAtlas.

Benefits of User Agent string analysis

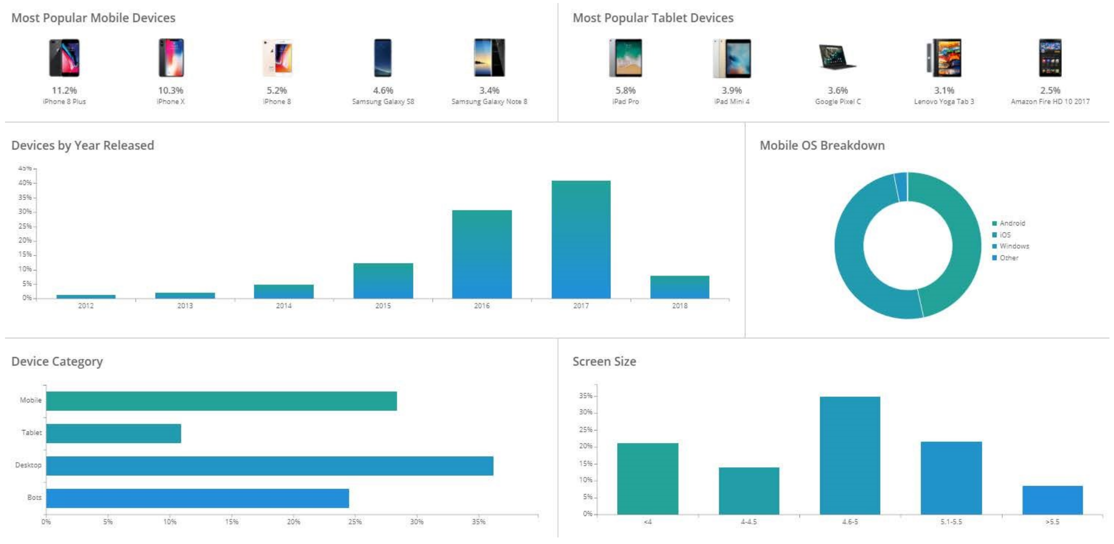
Increased Conversions - Content Optimization

A well-implemented User-Agent string analysis strategy allows you to adapt content dynamically, so that each visitor gets an optimal viewing experience. Whether the device used is a smartphone, tablet, desktop, or a low-end device, you only get one chance to make a first impression, so you should make it as positive as possible.

Another factor in optimizing for increased conversions is page load/weight. By parsing a User-Agent string, you can learn how large a device's screen is, for example, and send an appropriately sized image to the device. This cuts down on the potential customer's load time, and can also save them data if on a metered connection. The same improvement also increases the reach of your content by ensuring that people with compromised connectivity or less performant devices also get to experience your content, not just well-connected people in rich countries.. One size does not fit all in a global marketplace.

Getting the most out of User-Agent string 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 content this in action, view our article on Adaptive Web Design In Action.



Reporting/Analytics

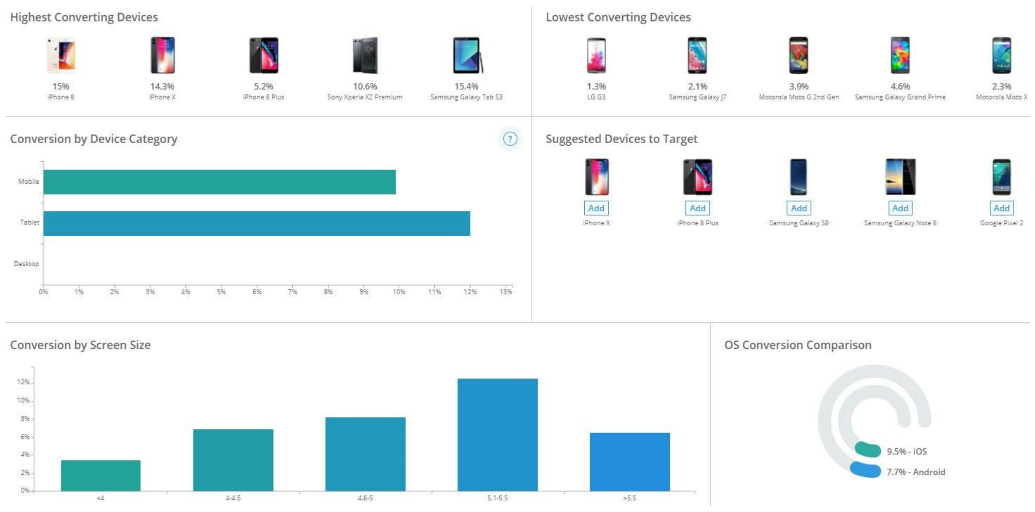
After the fact reporting and analysis of User-Agent visits can also inform future decisions and strategies, as the information you now have sheds light on the most granular of scenarios.

You may currently only report on broad categories like mobile/desktop/tablet. By adding hundreds of additional data points by using User-Agent strings, you can get a much closer look at the individual devices and any friction experienced when navigating through your website or app.

Enhanced Ad-targeting

It is important to make sure that your ads reach the right people at the right time.

Device detection is an essential ingredient to make this possible, and analysing the User-Agent string of each device allows you to ensure that 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.



For anyone involved in the online advertising space, buyers and sellers both, ad fraud is clearly a concern. DeviceAtlas is used by some major players in the ad-tech space, such as Adjust and SpotX to detect and combat ad fraud. We will discuss this further in the next section.

Bot Detection

Bots and web crawlers are not all evil, but those that do not exist to help you can be refused at the door.

By spotting an incoming bot's User-Agent string, you can avoid wasting resources on serving them content intended for a human audience.

Conversely, we all realize the benefits from search bots from Google and Bing, so treating those differently once identified by their User-Agent string allows you to benefit from good bots, while reducing the potential resource drain caused by the malicious variety.

User Agent Client Hints

In addition to the User-Agent string, DeviceAtlas now uses all HTTP headers, including Client Hints headers, to ensure maximum insights into device and browser capabilities in web traffic. Visit our HTTP Header Parser to see more.

How does the HTTP Header Parser work?

The HTTP Header Parser is used to identify a device if the Client Hints headers are populated and will fall back to identify a User-Agent string if they are not. To leverage the benefits from Client Hints, businesses will experience an increased development time while still requiring a solution for unsupported devices. A multi-indexed device intelligence solution, such as DeviceAtlas, can prove to be a more efficient and effective option.

We continue to evolve our service in line with Google's plans to deprecate and freeze User-Agent strings in Chrome.

Benefits of DeviceAtlas support of full HTTP Header analysis

- DeviceAtlas enables businesses to engage users on any connected device, extending the business reach both geographically and across market segments. We support Client Hints in addition to User-Agent strings using real-time analysis, which maximises coverage across browsers.
- DeviceAtlas support of the UA-CH model header provides continuity in a changing landscape. Our service is a safe, robust and forward-looking choice for businesses seeking deep intelligence on the connected devices accessing their services and offerings.
- Market leaders use DeviceAtlas to ensure that the consumer experience of their brand is consistent across all devices, from entry level to premium, to maximise engagement and loyalty. Now supporting UA-CH in addition to User-Agent string parsing

Third-party cloud-based device detection

Choosing a dedicated third-party device detection solution might be a viable choice, especially for larger high-traffic websites implementing content adaptation based on numerous devices' features.

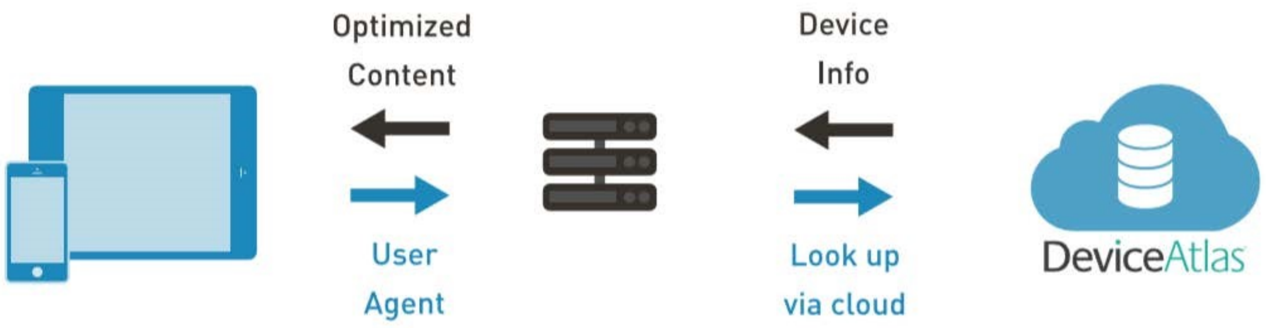
There are two options for the third-party device detection: cloud-based and a locally-deployed solution.

With cloud-based detection, data is delivered on demand for specific device headers submitted via the Cloud API. To integrate a cloud-based detection in your website you need to download the API and insert a code snippet into your website's code. The third-party service 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 the regex solution due to the fact that no manual updates are required. A third-party up-to-date device database also means a higher level of accuracy.

One example of this type of solution is the DeviceAtlas cloud-based detection service 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.

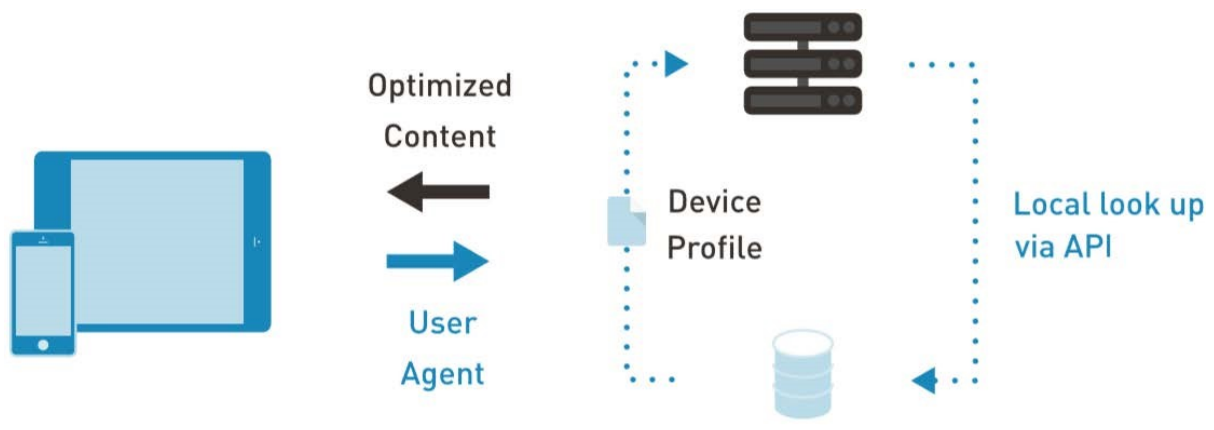


Locally-installed device detection

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 need to download a device data file from your detection provider and deploy the provider's API into your environment. It is best to set automatic, script-based downloads and updates of the file to ensure the most up-to-date data is in use.

In DeviceAtlas's case, the device data is available in a highly compressed JSON format which minimizes the server footprint. It can either be downloaded manually or obtained via an automated script. The data file consists of a JSON structure offering extremely fast lookups with a minimal footprint.



Examples of common User-Agent strings

There are millions of User-Agent string combinations, given that User-Agent strings change with both the software and hardware of a device. For example, a Chrome browser on an iPhone 13 will introduce itself using a different User-Agent string than a Safari browser on the same phone.

Every device type, including phones, tablets, desktops, may come with its own UA that makes it possible to detect this device for any purpose. Interestingly bots and crawlers also come with their unique UAs. Here is a list of example User-Agent strings for a selection of popular device types. If you would like to learn more about these devices, or any others that you might have a User-Agent string for, just copy and paste the strings into our User-Agent testing tool. You will then see all the properties of the detected device.

Android Mobile User-Agent strings

Samsung Galaxy A02Mozilla/5.0 (Linux; Android 11; Samsung SM-A025G) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.166 Mobile Safari/535.19

Samsung Galaxy A42 5G

Mozilla/5.0 (Linux; Android 11; SM-A426U) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.105 Mobile Safari/537.36

Samsung Galaxy F22

Mozilla/5.0 (Linux; Android 7.0; SM-G930VC Build/NRD90M; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/58.0.3029.83 Mobile Safari/537.36

Samsung Galaxy M12

Mozilla/5.0 (Linux; Android 11; SM-M127N) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/80.0.3987.119 Mobile Safari/537.36

Samsung Galaxy S21 5G

Mozilla/5.0 (Linux; Android 11; SM-G9910) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.181 Mobile Safari/537.36

Samsung Galaxy S21 Ultra 5G

Mozilla/5.0 (Linux; Android 11; SM-G998W) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Mobile Safari/537.36

Samsung Galaxy S21 + 5G

Mozilla/5.0 (Linux; Android 6.0.1; Nexus 6P Build/MMB29P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.83 Mobile Safari/537.36

Motorola Moto G10 Power

Mozilla/5.0 (Linux; Android 11; moto g(10) power Build/RRBS31.Q1-3-34-1-2; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/89.0.4389.105 Mobile Safari/537.36

Motorola One 5G ace

Mozilla/5.0 (Linux; Android 10; motorola one 5G ace Build/QZK30.Q4-40-55; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/88.0.4324.152 Mobile Safari/537.36

Motorola Moto E7 Power

Mozilla/5.0 (Linux; Android 10; moto e(7) power Build/QOM30.255-12; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/85.0.4183.101 Mobile Safari/537.36

Motorola Moto G50

Mozilla/5.0 (Linux; Android 11; moto g(50) Build/RRFS31.Q1-59-76-2; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/92.0.4515.159 Mobile Safari/537.36 EdgW/1.0

Motorola moto g play (2021)

Mozilla/5.0 (Linux; Android 10; moto g play (2021)) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.127 Mobile Safari/537.36

Huawei Mate 40 Pro

Mozilla/5.0 (Linux; Android 10; NOH-NX9) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.120 Mobile Safari/537.36

Huawei Nova 8 Pro 5G

Mozilla/5.0 (Linux; U; Android 10; zh-cn; BRQ-AN00 Build/HUAWEIBRQ-AN00) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/77.0.3865.120 MQQBrowser/11.9 Mobile Safari/537.36 COVC/045717

Nokia C1 Plus

Mozilla/5.0 (Linux; Android 10; Nokia C1 Plus Build/QP1A.190711.020; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/81.0.4044.138 Mobile Safari/537.36

Nokia G10

Mozilla/5.0 (Linux; Android 11; Nokia G10) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.88 Mobile Safari/537.36

Nokia G20

Mozilla/5.0 (Linux; Android 11; Nokia G20) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.88 Mobile Safari/537.36

Vivo V21

Mozilla/5.0 (Linux; Android 11; V2108) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.106 Mobile Safari/537.36

Vivo Y20 2021

Mozilla/5.0 (Linux; Android 10; moto e(7) power Build/QOM30.255-12; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/85.0.4183.101 Mobile Safari/537.36

Vivo X60

Mozilla/5.0 (Linux; Android 11; V2045) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/74.0.3729.136 Mobile Safari/537.36

Google Pixel 5

Mozilla/5.0 (Linux; Android 11; Pixel 5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.111 Mobile Safari/537.36

Google Pixel 4a

Mozilla/5.0 (Linux; Android 11; Pixel 4a Build/RP1A.200720.005; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/84.0.4147.111 Mobile Safari/537.36

Xiaomi Poco X3 Pro

Mozilla/5.0 (Linux; Android 11; M2102J20SG Build/RKQ1.200826.002; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/87.0.4280.141 Mobile Safari/537.36

Xiaomi Poco M3 Pro 5G

Mozilla/5.0 (Linux; Android 11; M2103K19PG) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.101 Mobile Safari/537.3

Xiaomi Mi 11 UltraAnchor

Mozilla/5.0 (Linux; Android 11; M2102K1G) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.101 Mobile Safari/537.36

iPhone User-Agent strings

Below are examples of User-Agent strings used by the most recent iPhone devices. As Apple do not pass much information through the User-Agent strings, these strings alone do not allow us to differentiate between iPhone models. However, with the DeviceAtlas client-side component, you can classify these user agents to return the correct device model.

Apple iPhone SE (Firefox)

Mozilla/5.0 (iPhone; CPU iPhone OS 14_7_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) FxiOS/36.0 Mobile/15E148 Safari/605.1.15

Apple iPhone 12 Pro Max (Firefox)

Mozilla/5.0 (iPhone; CPU iPhone OS 14_7_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) FxiOS/36.0 Mobile/15E148 Safari/605.1.15

Apple iPhone XS Max (Firefox)

Mozilla/5.0 (iPhone; CPU iPhone OS 12_0 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) FxiOS/13.2b11866 Mobile/16A366 Safari/605.1.15

Apple iPhone 12 Mini (Chrome)

Mozilla/5.0 (iPhone; CPU iPhone OS 14_7 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) CriOS/92.0.4515.90 Mobile/15E148 Safari/604.1

Apple iPhone 11 (Safari)

Mozilla/5.0 (iPhone; CPU iPhone OS 14_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.1 Mobile/15E148 Safari/604.1

Apple iPhone 12 Mini (Safari)

Mozilla/5.0 (iPhone; CPU iPhone OS 14_7_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1

Apple iPhone 12 Pro (Safari)

Mozilla/5.0 (iPhone; CPU iPhone OS 14_7_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1

Apple iPhone 12 (Safari)

Mozilla/5.0 (iPhone; CPU iPhone OS 14_7_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Mobile/15E148 Safari/604.1

Apple iPhone SE (2nd Generation) (Safari)

Mozilla/5.0 (iPhone12,8; U; CPU iPhone OS 13_0 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/14A403 Safari/602.1

Apple iPhone 12 Pro (Safari)

Mozilla/5.0 (iPhone13,3; 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

Desktop User-Agent strings

Linux-based PC using a Firefox browser

Mozilla/5.0 (X11; Linux x86_64; rv:93.0) Gecko/20100101 Firefox/93.0

macOS-based computer using a Safari browser

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.1 Safari/605.1.15

Windows 10-based PC using Edge browser

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.85 Safari/537.36 Edg/90.0.818.46

Chrome OS-based laptop using Chrome browser (Chromebook)

Mozilla/5.0 (X11; CrOS x86_64 13982.82.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.157 Safari/537.36

Set Top Box User-Agent strings

Apple TV 4K (6th Gen)

Mozilla/5.0 (AppleTV11,1; CPU OS 11.1 like Mac OS X; en-US)

Amazon Fire TV Stick Lite

Mozilla/5.0 (Linux; Android 9; AFTSS Build/PS7228; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/75.0.3770.143 Mobile Safari/537.36

Amazon Fire TV Stick (3rd Gen)

Mozilla/5.0 (Linux; Android 9; AFTSSS Build/PS7228; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/75.0.3770.143 Mobile Safari/537.36

TiVo Stream 4K

Mozilla/5.0 (Linux; U; Android 9.0.0; en-us; TIVO STREAM 4K Build/KOT49H)

Roku Express

RokuOS/3.0.0|Roku 3930X|STB|RokuOS 9.4.0 4200

Bots and Crawlers User-Agent strings

Google bot

Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; Googlebot/2.1; +http://www.google.com/bot.html) Chrome/92.0.4515.119 Safari/537.36

Bing bot

Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm) Anchor

Baiduspider

Mozilla/5.0 (compatible; Baiduspider-render/2.0; +http://www.baidu.com/search/spider.html)

Game Consoles User-Agent strings

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

Nintendo Switch

Mozilla/5.0 (Nintendo Switch; WifiWebAuthApplet) AppleWebKit/601.6 (KHTML, like Gecko) NF/4.0.0.5.10 NintendoBrowser/5.1.0.13343

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 NT 10.0; Win64; x64; Xbox; Xbox One) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2486.0 Safari/537.36 Edge/13.10586

Nvidia Shield

Mozilla/5.0 (Linux; Android 5.1; SHIELD Build/LMY47N; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/53.0.2785.97 Mobile Safari/537.36

Tablet User-Agent strings

Apple iPad

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.0 Safari/605.1.15

Note that, as with Apple iPhones, the user agent string alone does not contain sufficient information to identify iPad from Safari on a desktop device, let alone particular iPad models. However, with the DeviceAtlas client-side component, you can classify these user agents to return correct device model.

Samsung Galaxy Tab A7 Lite

Mozilla/5.0 (Linux; Android 11; SM-T227U Build/RP1A.200720.012; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/87.0.4280.141 Safari/537.36

Samsung Galaxy Tab S6 Lite

Mozilla/5.0 (Linux; Android 10; SM-610N Build/QP1A.190711.020; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/80.0.3987.119 Mobile Safari/537.36

Amazon Fire HD 10 (2021)

Mozilla/5.0 (Linux; Android 9; KFTRWI) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.88 Safari/537.36

Lenovo Tab E10

Mozilla/5.0 (Linux; Android 8.1.0; Lenovo TB-X104X Build/OPM1.171019.026; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/80.0.3987.162 Safari/537.36

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

Huawei MatePad Pro 12.6

Mozilla/5.0 (Linux; Android 10; WGR-W19 Build/HUAWEIWGR-W19; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/83.0.4103.106 Safari/537.36 Anchor

Acer Iconia Tab 10

Mozilla/5.0 (Linux; Android 10; ATAB1021E) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.88 Safari/537.36

Wristwatch User-Agent strings

Apple Watch SE

Mozilla/5.0 (Apple Watch5,9; CPU Apple Watch WatchOS like Mac OS X; WatchApp

Samsung Galaxy Watch Active2 (44mm)

Mozilla/5.0 (Linux; Android 9; SM-R825F Build/QP1A.190711.020; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/80.0.3987.119 Mobile Safari/537.36

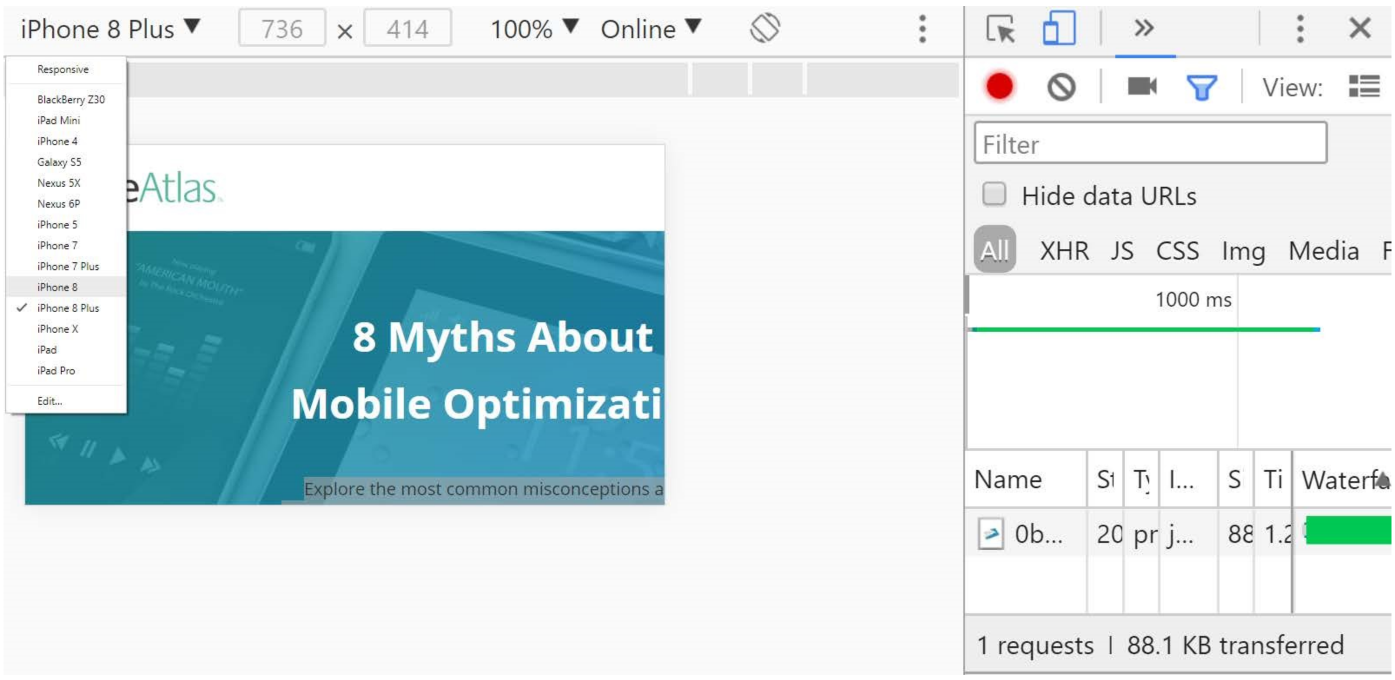
Huawei Watch 2

Mozilla/5.0 (Linux; Android 8.1; LEO-DLXXE Build/HONORLRA-AL00) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.111 HuaweiBrowser/9.1.1.308 Mobile Safari/537.36

Changing your User-Agent string

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 Use- Agent header.

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



iPhone 8 Plus ▼ 736 × 414 100% ▼ Online ▼

1 requests | 88.1 KB transferred

Patented Technology

In the device intelligence world, speed is everything, but accuracy can never be sacrificed. Our patented algorithm allows us to achieve both aims without trade-offs by combining some uniquely useful characteristics:

- It is extremely fast. But speed should never come at the cost of accuracy.
- It allows for perfect accuracy
- It has a very light memory and data file footprint

Whether you are running a real time bidding (RTB) platform where the entire auction process takes place in 100ms, or an analytics platform churning through trillions of requests, or a website running on a lowly VPS, you will not want to dispense with the speed of your solution.

The DeviceAtlas algorithm allows for the extremely high speeds afforded by a Patricia Trie, but with the flexibility to accommodate the reality that you cannot 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, they are an approach that works extremely well.

To this end we have incorporated some improvements into the traditional Patricia Trie. Firstly, it can handle non-perfect matches when necessary through our addition of pattern matching at key nodes in the trie where characters in the User-Agent string are not significant. This also removes unnecessary splaying of the trie, hugely decreasing the memory footprint of the loaded data.

Secondly, the data convection algorithm ensures that even a partial match of a User-Agent string will yield useful data.

Conclusion

At first glance, leveraging the User-Agent string seems like an easy way to segment traffic and optimize your content, leading to increased engagement on all devices.

The tricky part lies in handling a constantly evolving set of User-Agent strings, with new devices coming online daily. This makes planning, optimizing, analytics and reporting ineffective, quickly out of date and costly to manage.

For companies operating at scale that lack resources to deal with this in-house, it is 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 include:

- 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 180 device properties to build fine-grained content optimization and detailed reports on web traffic. Get started with a free trial to test DeviceAtlas in your environment.

GET STARTED >