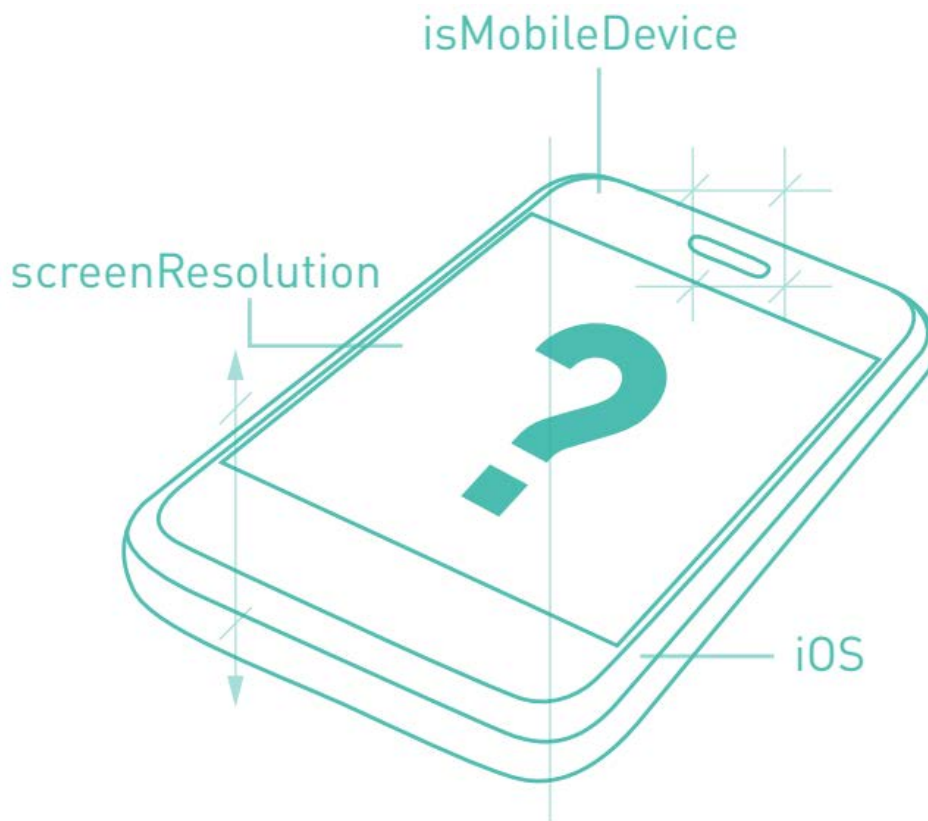


THE COMPLETE GUIDE TO USER AGENTS

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



CONTENTS

03 WHAT IS A USER AGENT?

04 HOW DOES USER AGENT PARSING WORK?

05 WHAT CAN YOU DO WITH USER AGENTS?

06 BUILDING A REGEX DEVICE DETECTION SOLUTION

07 THIRD-PARTY CLOUD BASED SOLUTION

08 LOCALLY INSTALLED DEVICE DETECTION

09 MOST COMMON USER AGENTS

What is a User-Agent?

A User-Agent (UA) 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 carrier ID, 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.

This is the advantage of using DeviceAtlas for your detection requirements.

Anatomy of a User-Agent

Use of the User-Agent string is specified in the standards on HTTP here - [RFC 1945](#).

In fact, the UA 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 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.

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 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 UAs used by a Samsung Galaxy S8 and a Mac OS X-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 parsing work in device detection?

From a technical point of view examining the User-Agent is not difficult. You can get a User-Agent string using `navigator.userAgent` in JavaScript or the `HTTP_USER_AGENT` variable.

Many companies use a regex approach to analyze the User-Agent. Again this relies on pattern or string matching to identify keywords which might identify the underlying device. 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.

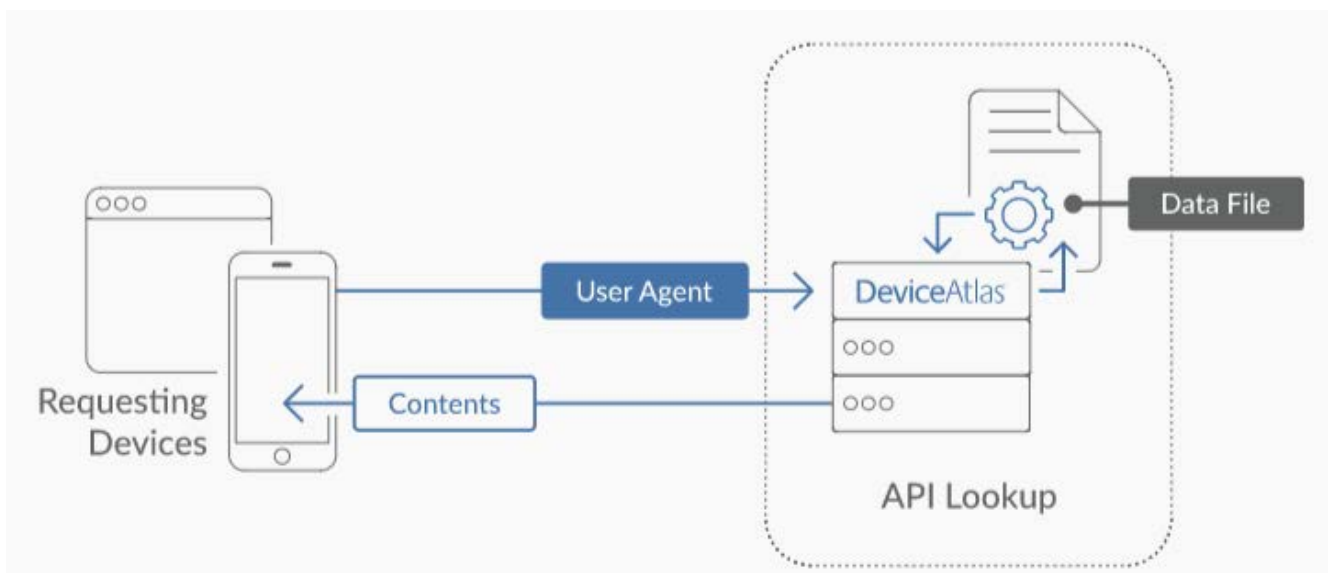
As User-Agent strings do not conform to any standard pattern, this technique is prone to failure and is not future proof. You would need to constantly update your regex rules as new devices, browsers and OS's are released, and then run tests to see if the solution still works well. At some point, this becomes a costly maintenance job, and, over time, a real risk that you are mis-detecting or failing to detect much of your traffic.

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



In this regard not all approaches to device detection are created equal—the bad ones will have inaccurate data, return false positives—you may think you are getting a correct result, but an inferior solution may return default values for unknown UAs. 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. This is the reason why major companies rely on established solutions built on proven and patented technology like DeviceAtlas.

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 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**

But speed should never come at the cost of accuracy.

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, speed is something you just can't have enough of.

The 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 exceedingly 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 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.

Benefits of User Agent analysis

Increased Conversions - Content Optimization

A well implemented User Agent 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, 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 wait time, and can also save them data if 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 content in action, view our article on [Adaptive Web Design In Action](#).



Reporting/Analytics

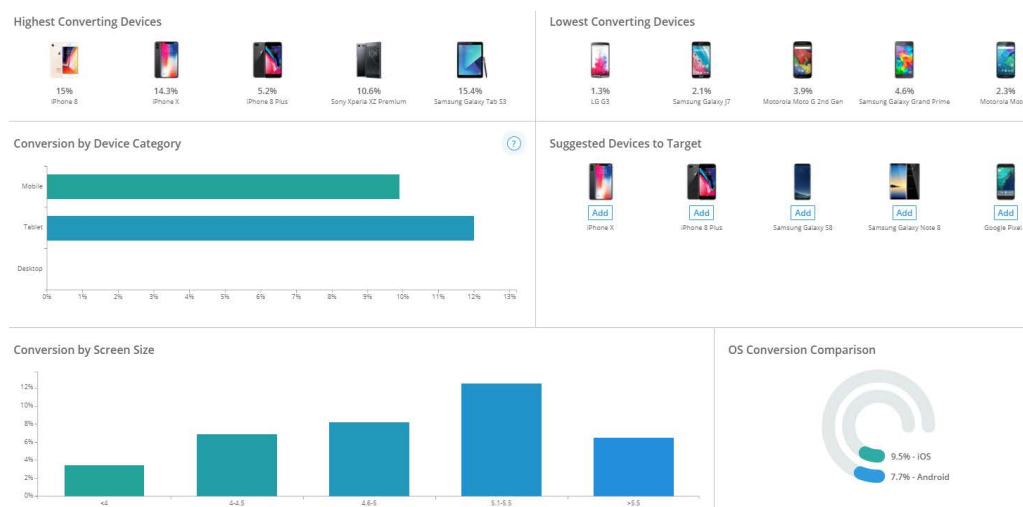
After the fact reporting and analysis of User Agent visits can also inform future decisions and strategies, as the information you now have sheds light on the most granular of scenarios.

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

Enhanced Ad-targeting

It's 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 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.



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.

Bot Detection

Bots and web crawlers aren't all evil, but those that aren't there to help you can be refused at the door.

By spotting an incoming bot User Agent, 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 allows you benefit from good bots, but reduce the potential resource drain caused by the malicious variety.

Building a regex device detection solution

In some cases it is feasible to build a device detection based on regex. It 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 coding languages.

There are, however, some limitations that you should be aware of before exploring this option further:

Accuracy – Regex User-Agent matching can work well in the general case, but will inevitably fail to recognize some portion of devices correctly, e.g. certain Android tablets. And while it is 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 for you, 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), for instance if you need to know device properties such as screen size, memory limit, HTML5 support, then the regex solution may not be suitable.

Third-party cloud based device detection

Choosing a dedicated third-party device detection solution might be a better 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.

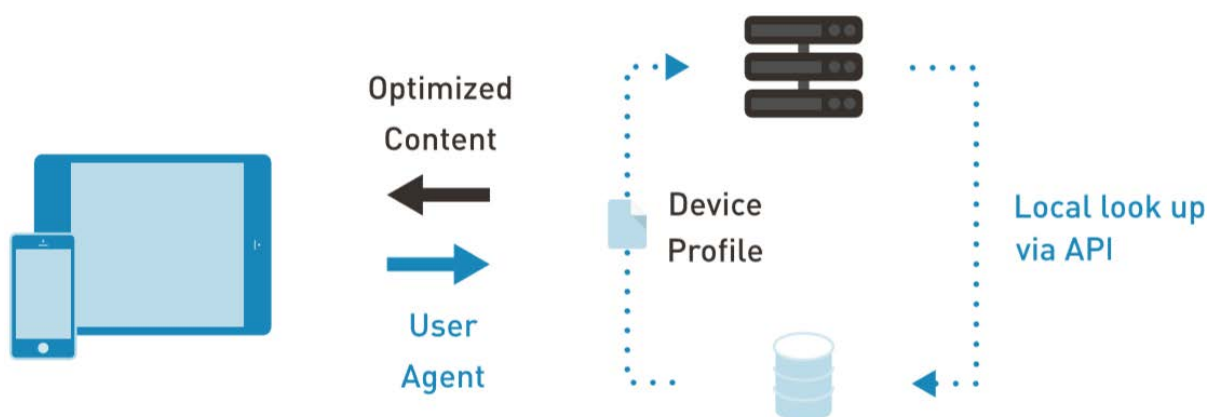


Locally-installed device detection

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 minimizing 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-Agents

There are millions of User-Agent combinations given that UAs change with the software and hardware. For example, a Chrome browser on an iPhone 7 will introduce itself using a different UA 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-Agents for different device types that can be detected. If you'd like to learn more on these devices, just copy and paste the UAs to our [User-Agent testing tool](#). You'll see all the properties of a detected device.

Android Mobile User Agents

Samsung Galaxy S8

```
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
```

Samsung Galaxy S7

```
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 S7 Edge

```
Mozilla/5.0 (Linux; Android 6.0.1; SM-G935S Build/MMB29K; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/55.0.2883.91 Mobile Safari/537.36
```

Samsung Galaxy S6

```
Mozilla/5.0 (Linux; Android 6.0.1; SM-G920V Build/MMB29K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.98 Mobile Safari/537.36
```

Samsung Galaxy S6 Edge Plus

```
Mozilla/5.0 (Linux; Android 5.1.1; SM-G928X Build/LMY47X) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.83 Mobile Safari/537.36
```

Google/Nexus 6P

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

Sony Xperia XZ

Mozilla/5.0 (Linux; Android 7.1.1; G8231 Build/41.2.A.0.219; ww) AppleWebKit/537.36 (KHTML, like Gecko)
Version/4.0 Chrome/59.0.3071.125 Mobile Safari/537.36

Sony Xperia Z5

Mozilla/5.0 (Linux; Android 6.0.1; E6653 Build/32.2.A.0.253) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.98 Mobile Safari/537.36

HTC One X10

Mozilla/5.0 (Linux; Android 6.0; HTC One X10 Build/MRA58K; ww) AppleWebKit/537.36 (KHTML, like Gecko)
Version/4.0 Chrome/61.0.3163.98 Mobile Safari/537.36

HTC One M9

Mozilla/5.0 (Linux; Android 6.0; HTC One M9 Build/MRA58K) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.98 Mobile Safari/537.36

iPhone User Agents

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

However, with [DeviceAtlas client-side](#), you can classify these user agents to return correct device model.

Apple iPhone X

```
Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11.0 Mobile/15A372 Safari/604.1
```

Apple iPhone 8

```
Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1.34 (KHTML, like Gecko) Version/11.0 Mobile/15A5341f Safari/604.1
```

Apple iPhone 8 Plus

```
Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11.0 Mobile/15A5370a Safari/604.1
```

Apple iPhone 7

```
Mozilla/5.0 (iPhone9,3; U; CPU iPhone OS 10_0_1 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/14A403 Safari/602.1
```

Apple iPhone 7 Plus

```
Mozilla/5.0 (iPhone9,4; U; CPU iPhone OS 10_0_1 like Mac OS X) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Mobile/14A403 Safari/602.1
```

Apple iPhone 6

```
Mozilla/5.0 (Apple-iPhone7C2/1202.466; U; CPU like Mac OS X; en) AppleWebKit/420+ (KHTML, like Gecko) Version/3.0 Mobile/1A543 Safari/419.3
```

Desktop User Agents

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

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

```
Roku4640X/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/LMY470) 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

We've compiled a more in-depth list of User-Agent strings of the [most active web crawlers and bots](#).

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

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

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

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

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)

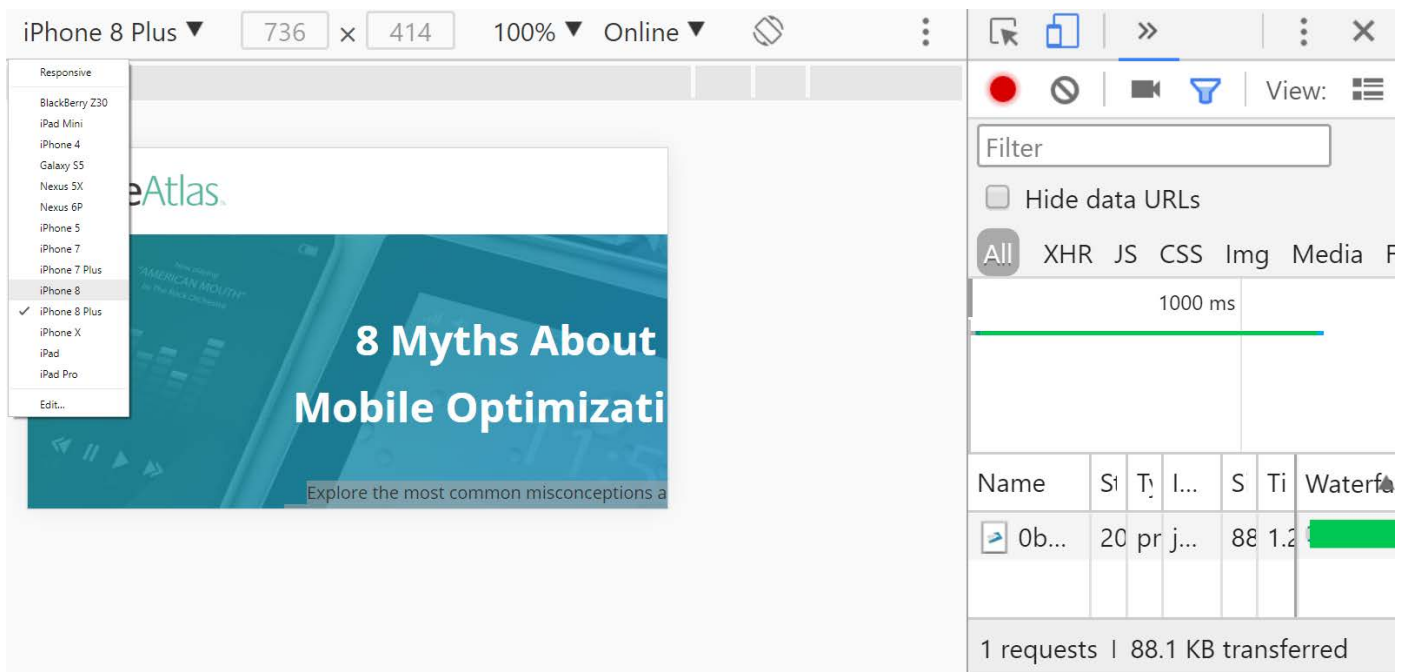
Changing your User-Agent

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.](#)



The screenshot shows a browser window in responsive design mode. The top bar indicates the device is set to "iPhone 8 Plus" with a resolution of 736x414 and 100% zoom. The page content is displayed in a mobile layout, featuring a blue header with the "eAtlas" logo and a main section titled "8 Myths About Mobile Optimizati". Below the title, there is a sub-header "Explore the most common misconceptions a".

The developer tools panel is open on the right side. The "Filter" input is empty. The "Hide data URLs" checkbox is unchecked. The "All" filter is selected, and the "XHR" tab is active. A network request is visible with a duration of 1000 ms. The network panel table shows the following data:

Name	St	T	L...	S	Ti	Waterfa
Ob...	20	pr	j...	88	1.2	

At the bottom of the network panel, it shows "1 requests | 88.1 KB transferred".

Conclusion

At first glance, leveraging the User Agent 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 UAs, 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'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 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 →](#)