The Device Memory API

By Philip Walton

Engineer at Google working on the Web Platform

The range of capabilities of devices that can connect to the web is wider today than it's ever been before. The same web application that's served to a high-end desktop computer may also be served to a low-powered phone, watch, or tablet, and it can be incredibly challenging to create compelling experiences that work seamlessly on any device.

The Device Memory API is a new web platform feature aimed at helping web developers deal with the modern device landscape. It adds a read-only property to the `navigator` object, `navigator.deviceMemory`, which returns how much RAM the device has in gigabytes, rounded down to the nearest power of two. The API also features a Client Hints Header, `Device-Memory`, that reports the same value.

Note: The JavaScript API is new in Chrome 63, while the Client Hints header has been enabled by default in Chrome since version 61.

The Device Memory API gives developers the ability to do two primary things:

- Make runtime decisions about what resources to serve based on the returned device memory value (e.g. serve a "lite" version of an app to users on low-memory devices).
- Report this value to an analytics service so you can better understand how device memory correlates with user behavior, conversions, or other metrics important to your business.

Tailoring content dynamically based on device memory

If you're running your own web server and are able to modify the logic that serves resources, you can conditionally respond to requests that contain the `Device-Memory` Client Hints Header.

```
GET /main.js HTTP/1.1
Host: www.example.com
Device-Memory: 0.5
Accept: */*
```

With this technique you can create one or more versions of your application script(s) and respond to requests from the client conditionally based on the value set in the **`Device-Memory`** header. These versions don't need to contain completely different code (as that's harder to maintain). Most of the time the "lite" version will just exclude features that may be expensive and not critical to the user experience.

Using the Client Hints Header

To enable the **`Device-Memory`** header, either add the Client Hints **`<meta>`** tag to the **`<head>`** of your document:

```
<meta http-equiv="Accept-CH" content="Device-Memory">
```

Or include "Device-Memory" in your server's **`Accept-CH`** response headers:

```
HTTP/1.1 200 OK
Date: Thu Dec 07 2017 11:44:31 GMT
Content-Type: text/html
Accept-CH: <strong>Device-Memory</strong>, Downlink, Viewport-Width
```

This tells the browser to send the **`Device-Memory`** header with all sub-resource requests for the current page.

In other words, once you've implemented one of the options above for your website, if a user visits on a device with 0.5 GB of RAM, all image, CSS, and JavaScript requests from this page will include the **`Device-Memory`** header with the value set to "0.5", and your server can respond to such requests however you see fit.

For example, the following Express route serves a "lite" version of a script if the **`Device-Memory`** header is set and its value is less than 1, or it serves a "full" version if the browser doesn't support the **`Device-Memory`** header or the value is 1 or greater:

```
app.get('/static/js/:scriptId', (req, res) => {
  // Low-memory devices should load the "lite" version of the component.
  // The logic below will set `scriptVersion` to "lite" if (and only if)
  // `Device-Memory` isn't undefined and returns a number less than 2.
  const scriptVersion = req.get('Device-Memory') < 1 ? 'lite' : 'full';

  // Respond with the file based on `scriptVersion` above.
  res.sendFile(`./path/to/${req.params.scriptId}.${scriptVersion}.js`);
});
```

Note: The `Accept-CH` header works well for a page's subresource, but it doesn't help if you want to conditionally serve page contents based on device capabilities. To address this, the **`Accept-CH-Lifetime`** header (coming soon to Chrome) instructs browsers to include the specified client hints headers in all subsequent requests from this origin for the number of seconds specified by the header value.

Using the JavaScript API

In some cases (like with a static file server or a CDN) you won't be able to conditionally respond to requests based on an HTTP header. In these cases you can use the JavaScript API to make conditional requests in your JavaScript code.

The following logic is similar to the Express route above, except it dynamically determines the script URL in the client-side logic:

```
// Low-memory devices should load the "lite" version of the component.
// The logic below will set `componentVersion` to "lite" if (and only if)
// deviceMemory isn't undefined and returns a number less than 1.
const componentVersion = navigator.deviceMemory < 1 ? 'lite' : 'full';

const component = await import(`path/to/component.${componentVersion}.js`);
component.init();
```

While conditionally serving different versions of the same component based on device capabilities is a good strategy, sometimes it can be even better to not serve a component at all.

In many cases, components are purely enhancements. They add some nice touches to the experience, but they aren't required for the app's core functionality. In these cases, it may be wise to not load such components in the first place. If a component intended to improve the user experience makes the app sluggish or unresponsive, it's not achieving its goal.

With any decision you make that affects the user experience, it's critical you measure its impact. It's also critical that you have a clear picture of how your app performs today.

Understanding how device memory correlates with user behavior for the current version of your app will better inform what action needs to be taken, and it'll give you a baseline against which you can measure the success of future changes.

Tracking device memory with analytics

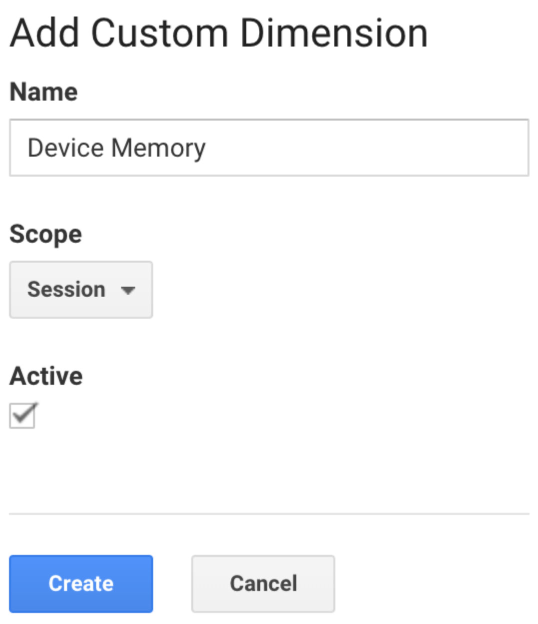
The Device Memory API is new, and most analytics providers are not tracking it by default. Fortunately, most analytics providers give you a way to track custom data (for example, Google Analytics has a feature called Custom Dimensions), that you can use to track device memory for you users' devices.

Using a custom device memory dimension

Using custom dimensions in Google Analytics is a two-step process.

1. Set up the custom dimension in Google Analytics
2. Update your tracking code to `set` the device memory value for the custom dimension you just created.

When creating the custom dimension, give it the name "Device Memory" and choose a scope of "session" since the value will not change during the course of a user's browsing session:


Add Custom Dimension

Name: Device Memory

Scope: Session

Active: ✓

Next update your tracking code. Here's an example of what it might look like. Note that for browsers that don't support the Device Memory API, the dimension value will be "(not set)".

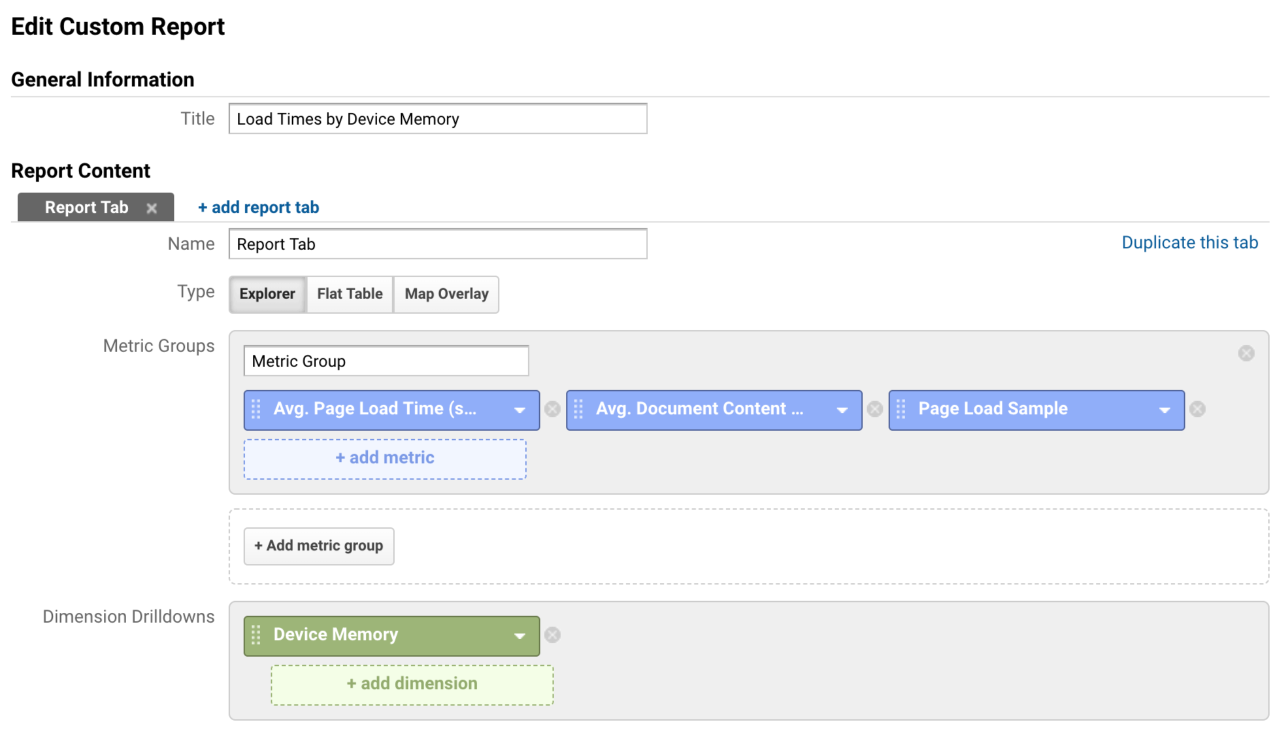
```
// Create the tracker from your tracking ID.
// Replace "UA-XXXXX-Y" with your Google Analytics tracking ID.
ga('create', 'UA-XXXXX-Y', 'auto');

// Set the device memory value as a custom dimension on the tracker.
// This will ensure it gets sent with all future data to Google Analytics.
// Note: replace "XX" with the index of the custom dimension you created
// in the Google Analytics admin.
```

```
ga('set', 'dimensionXX', navigator.deviceMemory || '(not set)');

// Do any other other custom setup you want...

// Send the initial pageview.
ga('send', 'pageview');
```

Reporting on Device Memory data

Once the device memory dimension is `set` on the tracker object, all data you send to Google Analytics will include this value. This will allow you to break down any metric you want (e.g. page load times, goal completion rate, etc.) by device memory to see if there are any correlations.

Since device memory is a custom dimension rather than a built-in dimension, you won't see it in any of the standard reports. To access this data you'll have to create a custom report. For example, the configuration for a custom report that compares load times by device memory might look like this:


Edit Custom Report

General Information

Title: Load Times by Device Memory

Report Content

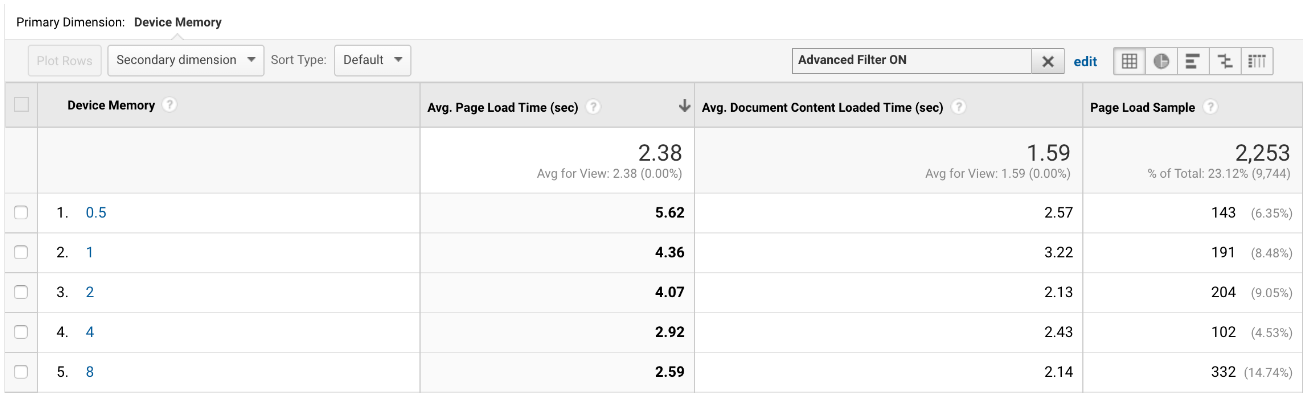
Name: Report Tab

Type: Explorer

Metric Groups: Metric Group — Avg. Page Load Time (s..., Avg. Document Content ..., Page Load Sample

Dimension Drilldowns: Device Memory

And the report it generates might looks like this:



	Device Memory	Avg. Page Load Time (sec)	Avg. Document Content Loaded Time (sec)	Page Load Sample
		2.38 Avg for View: 2.38 (0.00%)	1.59 Avg for View: 1.59 (0.00%)	2,253 % of Total: 23.12% (9,744)
1.	0.5	5.62	2.57	143 (6.35%)
2.	1	4.36	3.22	191 (8.48%)
3.	2	4.07	2.13	204 (9.05%)
4.	4	2.92	2.43	102 (4.53%)
5.	8	2.59	2.14	332 (14.74%)

Once you're collecting device memory data and have a baseline for how users are experiencing your application across all ranges of the memory spectrum, you can experiment with serving different resources to different users (using the techniques described in the section above). Afterwards you'll be able to look at the results and see if they've improved.

Wrapping up

This post outlines how to use the Device Memory API to tailor your application to the capabilities of your users' devices, and it shows how to measure how these users experience your app.

While this post focuses on the Device Memory API, most of the techniques described here could be applied to any API that reports device capabilities or network conditions.

As the device landscape continues to widen, it's more important than ever that web developers consider the entire spectrum of users when making decisions that affect their experience.

Last updated July 2, 2018.