PWAs on Oculus Quest 2

- The Oculus Browser
  - User agent
  - User interface
  - Refresh rate and device pixel ratio
- PWAs in Oculus Browser and the Oculus Store
  - State sharing
  - PWA window sizes
  - Interacting with PWAs
  - Permissions for PWAs
  - Debugging PWAs via Chrome DevTools
  - App discovery
- Exemplary PWAs on the Oculus Quest 2
  - PWAs by Meta
  - PWAs by other developers
- Creating PWAs for Oculus
  - Web App Manifest requirements
  - Packaging PWAs with Bubblewrap CLI
  - Packaging PWAs with Oculus Platform Utility
  - Packaging PWAs with PWABuilder
  - Installing PWAs with ADB
  - App submission
- Testing multi-tab apps
- An Oculus version of SVGcode
- Immersive 3D WebXR PWAs
  - Hand tracking
  - Augmented/Mixed Reality (Passthrough)
- Conclusions
- Acknowledgements

The Oculus Quest 2 is a virtual reality (VR) headset created by Oculus, a division of Meta. It is the successor to the company's previous headset, the Oculus Quest. The device is capable of running as both a standalone headset with an internal, Android-based operating system, and with Oculus-compatible VR software running on a desktop computer when connected over USB or Wi-Fi. It uses the Qualcomm Snapdragon XR2 system on a chip with 6 GB of RAM. The Quest 2's display is a singular fast-switch LCD panel with 1,832 × 1,920 pixels per eye resolution that runs at a refresh rate of up to 120 Hz.

Oculus Quest 2 device with controllers.

The Oculus Browser

Currently there are three browsers available for the Oculus Quest 2: Wolvic, a successor to Firefox Reality, and the built-in Oculus Browser. This article focuses on the latter. The Oculus website introduces the Oculus Browser as follows.

"*Oculus Browser provides support for the latest web standards and other technologies to help you create VR experiences on the web. Today's 2D web sites work great in Oculus Browser because it's powered by the Chromium rendering engine. It's further optimized for Oculus headsets to get the best performance and to enable web developers take advantage of the full potential of VR with new APIs, like WebXR. Through WebXR, we're opening the doors to the next frontier of the web.*"

Oculus Browser with three browser windows open.

User agent

The browser's user agent string at the time of writing is as follows.

```
Mozilla/5.0 (X11; Linux x86_64; Quest 2)
AppleWebKit/537.36 (KHTML, like Gecko)
OculusBrowser/18.1.0.2.46.337441587
SamsungBrowser/4.0
Chrome/95.0.4638.74
VR
Safari/537.36
```

As you can see, the current version `18.1.0.2.46.337441587` of the Oculus Browser is based on Chrome `95.0.4638.74`, that is only one version behind the current stable version of Chrome, which is `96.0.4664.110`. If the user switches to mobile mode, `VR` changes to `Mobile VR`.

Oculus Browser About page.

User interface

The browser's user interface (shown above) has the following features (top row from left to right):

- Back button
- Reload button
- Site information
- URL bar
- Create bookmark button
- Resize button with narrow, medium, and wide options as well as a zoom feature
- Request mobile website button
- Menu button with the following options:
  - Enter private mode
  - Close all tabs
  - Settings
  - Bookmarks
  - Downloads
  - History
  - Clear browsing data

The bottom row includes the following features:

- Close button
- Minimize button
- Three dots button with back, forward, and reload options

Refresh rate and device pixel ratio

For Oculus Quest 2, Oculus Browser renders both 2D web page content and WebXR at a 90 Hz refresh rate. When watching fullscreen media, Oculus Browser optimizes the device refresh rate based on the frame rate of the video, for example, 24 fps. The Oculus Quest 2 has a device pixel ratio of 1.5 for crisp text.

PWAs in Oculus Browser and the Oculus Store

On October 28, 2021, Jacob Rossi, Product Management Lead at Meta (Oculus), shared that PWAs were coming to Oculus Quest and Oculus Quest 2. In the following, I describe the PWA experience on Oculus and explain how to build, sideload, and test your PWA on the Oculus Quest 2.

State sharing

Login state is shared between Oculus Browser and PWAs, allowing users to seamlessly switch between the two. Naturally, Facebook Login is supported out of the box. The Oculus Browser includes a password manager that allows users to store and share their passwords securely between the browser and installed app experiences.

PWA window sizes

Browser windows and windows of installed PWAs can be freely resized by the user. The height can vary between 625 px and 1,200 px. The width can be set between 400 px and 2,000 px. The default dimensions are 1,000 × 625 px.

Interacting with PWAs

PWAs can be controlled with the Oculus left and right controllers, Bluetooth mice and keyboards, and via hand tracking. Scrolling works via the thumb sticks on the Oculus controllers, or by pinching the thumb and the index finger and moving in the desired direction. To select something, the user can point and pinch.

Permissions for PWAs

Permissions in Oculus Browser work pretty much the same way as in Chrome. The state is shared between apps running in the browser and installed PWAs, so users can switch between the two experiences without having to grant the same permissions again.

Albeit many permissions are implemented, not all features are supported. For example, while requesting the geolocation permission succeeds, the device never actually gets a location.

Similarly, the various hardware APIs like WebHID, Web Bluetooth, etc. all pass feature detection, but don't actually show a picker that would let the user pair the Oculus with a hardware device. I suppose feature detectability of APIs will be refined once the browser matures.

Permissions in Oculus Browser.

Debugging PWAs via Chrome DevTools

After enabling Developer Mode, debugging PWAs on Oculus Quest 2 works exactly as described in Remote debug Android devices.

1. On the Oculus device, browse to the desired site in Oculus Browser.
2. Launch Google Chrome on your computer and navigate to `chrome://inspect/#devices`.
3. Find the Oculus device in question, which will be followed by a set of Oculus Browser tabs currently open on the device.
4. Click **inspect** on the desired Oculus Browser tab.

Inspecting an app running on the Oculus Quest 2 with Chrome DevTools.

App discovery

People can use the browser itself or the Oculus Store to discover PWAs. Just like with any other browser, installed PWAs also work in Oculus Browser as websites running in a tab. When a user visits a site, the Oculus Browser will help them discover the app if (and only if) it is available in the Oculus Store. For users that already have the app installed, Oculus Browser will help them easily switch to the app if they desire.

Currently the `BeforeInstallPrompt` event will *not* fire in Oculus Browser, despite feature detection reporting it to be supported.

Oculus Browser inviting the user in a prompt to install the MyEmail app.

Exemplary PWAs on the Oculus Quest 2

PWAs by Meta

Multiple Meta divisions have created PWAs for the Oculus Quest 2, for example Instagram and Facebook. These PWAs run in standalone app windows that don't have a URL bar and that can be freely resized.

Facebook Oculus Quest 2 app.

Instagram Oculus Quest 2 app

PWAs by other developers

At the time of this writing, there is a small but growing number of PWAs for the Oculus Quest 2 on the Oculus Store. Spike lets users experience all the essential work tools like email, chat, calls, notes, tasks, and to-dos from their inbox in a virtual environment hub right in the Spike app.

Spike Oculus Quest 2 app.

Another example is Smartsheet, a dynamic workspace that provides project management, automated workflows, and rapid building of new solutions.

More PWAs like Slack, Dropbox, or Canva are coming, as teased in a video featuring Jacob Rossi that was released in the context of Facebook's Connect conference in 2021.

Creating PWAs for Oculus

Meta outlined the required steps in their documentation. In general, PWAs that are installable in Chrome should oftentimes work out of the box on Oculus.

Web App Manifest requirements

There are some important differences compared to Chrome's installability criteria and the Web App Manifest spec. For example, Oculus only supports left-to-right languages at the moment, whereas the Web App Manifest spec enforces no such constraints. Another example is `start_url`, which Chrome strictly requires for an app to be installable, but which on Oculus is optional. Oculus offers a command line tool that lets developers create PWAs for the Oculus Quest 2, which allows them to pass the missing (or override the existing) parameters in the Web App Manifest.

Name	Description
`name`	(Required) The name of the PWA. Currently Oculus only supports left-to-right languages for the name.
`display`	(Required) Either `"standalone"` or `"minimal-ui"`. Oculus currently doesn't support any other values.
`short_name`	(Required) A shorter version of the app name, if needed.
`scope`	(Optional) The URL or paths that should be considered as part of the app.
`start_url`	(Optional) The URL to show at app launch.

Oculus has a number of optional **proprietary** Web App Manifest fields that can be used to customize the PWA experience.

Name	Description
`ovr_package_name`	(Optional) Sets the package name of the APK generated for the PWA. This must be in reverse domain name notation, e.g., `"com.company.app.pwa"`. If not set, developers must provide a package name to the command line tool with the (then required) parameter `--package-name`.
`ovr_multi_tab_enabled`	(Optional) If `true`, this boolean field will give the PWA a tab bar similar to Oculus Browser. In multi-tab PWAs, internal links that target a new tab (`target="_new"` or `target="_blank"`) will open in new tabs within the PWA window. This differs from single-tab PWAs where such links would open to a Oculus Browser window. **This feature is currently being standardized as Tabbed Application Mode.**
`ovr_scope_extensions`	(Optional) Allows a PWA to include more web pages within the scope of the web application. It consists of a JSON dictionary containing extension URLs or wildcard patterns. **This feature is currently being standardized as Scope Extensions for Web Apps.**

Packaging PWAs with Bubblewrap CLI

Bubblewrap is an open source set of libraries and a command line tool (CLI) for Node.js. Bubblewrap is developed by the Google Chrome team, to help developers generate, build, and sign an Android project that launches your PWA as a Trusted Web Activity (TWA).

Meta Quest Browser currently doesn't fully support TWA, but starting from version 1.18.0, Bubblewrap supports packaging PWAs for Meta Quest devices.

It can generate *universal* APK files that open a TWA on regular Android devices and the Meta Quest Browser on Meta Quest devices.

Assuming you have Node.js installed, Bubblewrap CLI can be installed with the following command:

```
npm i -g @bubblewrap/cli
```

When running Bubblewrap for the first time, it will offer to automatically download and install the required external dependencies—Java Development Kit (JDK) and Android SDK Build Tools.

To generate a Meta Quest compatible Android project that wraps your PWA, run the `init` command with the `--metaquest` flag and follow the wizard:

```
bubblewrap init --manifest="https://your.web.app/manifest.json" --metaquest
```

Once the project has been generated, build and sign it by running:

```
bubblewrap build
```

This will output a file called `app-release-signed.apk`. This file can be installed on the device or published to the Meta Quest Store, Google Play Store or any of the other Android app distribution platforms.

Pro tip: you can also use the `--chromeosonly` flag in addition to the `--metaquest` flag to make APK files compatible not only with Meta Quest and regular Android devices, but also with ChromeOS devices.

Packaging PWAs with Oculus Platform Utility

Oculus Platform Utility is the official command line tool developed by Meta for publishing apps for Oculus Rift and Meta Quest devices.

It also allows to package PWAs for Meta Quest devices with the `create-pwa` command and publish them to the Meta Quest Store and App Lab.

Set the output file name via the `-o` parameter and the path to Android SDK via the `--android-sdk` parameter.

Point the tool at the live URL of the web app manifest via the `--web-manifest-url` parameter.

If you don't have a manifest on your live PWA or wish to override the live manifest, you can still generate an APK for your PWA using a local manifest file and the `--manifest-content-file` parameter.

To leave the manifest as pure as possible, use the `--package-name` parameter with a value in reverse domain name notation (for example, `com.company.app.pwa`), rather than adding the proprietary `ovr_package_name` field to the manifest.

```
ovr-platform-util create-pwa -o output.apk --android-sdk ~/bin/android-10 --manifest-content-file manifest.json --package-name com.company.app.pwa
```

APK files generated by Oculus Platform Utility are only compatible with Meta Quest devices and cannot be run on regular Android devices. Also they can only be published to the Meta Quest Store and App Lab.

Packaging PWAs with PWABuilder

Using PWABuilder is in the authors' view the easiest and therefore recommended way to package PWAs for Meta Quest at the moment.

PWABuilder is an open source project developed by Microsoft, that allows developers to package and sign their PWAs for publishing to various stores, including Microsoft Store, Google Play Store, App Store, and Meta Quest Store.

Packaging PWAs with PWABuilder is as easy as entering the URL of a PWA, entering/editing the metadata for the app, and clicking the **Generate** button.

PWABuilder gives developers the choice of what tool under the hood to use for packaging PWAs for Meta Quest devices.

You can choose the **Meta Quest** option to use the Oculus Platform Utility.

PWABuilder packaging options.

Pro tip: PWABuilder uses a slightly modified version of Oculus Platform Utility that generates much smaller APK files by removing unused code and resources.

You can choose the **Android** option to use the Bubblewrap and select the **Meta Quest compatible** checkbox.

Packaging PWAs with PWABuilder using Bubblewrap.

Pro tip: since PWABuilder is a PWA itself that works everywhere there is a browser, you can open it in the Meta Quest Browser and perform all the steps right on your Meta Quest.

Installing PWAs with ADB

To enable Android Debug Bridge (ADB) on the Meta Quest device, you must enable developer mode in the companion mobile app. Before you can put your device in developer mode, you must belong to (or have created) a developer organization in the Oculus Developer Center.

After creating the APK file, you can sideload it to the Meta Quest device using the ADB via USB or Wi-Fi:

```
adb install app-release-signed.apk
```

If you use the Bubblewrap CLI for packaging PWAs, it provides a convenient alias command to sideload the APK file:

```
bubblewrap install
```

Pro tip: you can use Android Web Toolbox to sideload APK files. It's an open source web app that allows you to access all ADB functionality right from the browser even from another Android device. No installation or drivers are required thanks to the Web USB API.

Sideloaded apps appear in an **Unknown Sources** section in the app drawer.

App submission

Submission and consideration for the Oculus Store is only available if you have been approved as an Meta Quest Store developer. Distribution of PWAs via App Lab is not currently available. The Meta team will share more on when and how you can submit a PWA to App Lab soon.

Uploading and submitting PWAs to the Oculus Store is covered in detail in the Oculus Developer Center docs.

Apart from submitting apps to the Oculus Store, developers can also distribute their apps via platforms like SideQuest directly to consumers safely and securely, without requiring store approval. This allows them to get an app directly to end users, even if it is early in development, experimental, or aimed at a unique audience.

Testing multi-tab apps

To test multi-tab apps, I created a little test PWA that demonstrates the various link features: namely opening a new in-PWA tab, staying on the current tab, opening a new browser window, and opening in a WebView staying on the current tab. Create a locally installable copy of this app by running the commands below on your machine.

```
ovr-platform-util create-pwa -o test.apk --android-skd ~/bin/android-10 --web-manifest-url
https://tomayac.github.io/oculus-pwa-test/manifest.json --package-name com.example.pwa
adb install test.apk
```

Here's a screencast of the test app.

https://www.youtube.com/watch?v=3ZlxCjW9rtg

An Oculus version of SVGcode

To take the instructions for a spin, I created an Oculus version of my most recent PWA, SVGcode. You can download the resulting APK file `output.apk` from my Google Drive. If you want to investigate the package further, I have a decompiled version, too. Find the build instructions in `package.json`.

Using the app on Oculus works fine, including the ability to open and save files. The Oculus Browser doesn't support the File System Access API, but the fallback approach helps. The only thing that didn't function is pinch-zooming. My expectation was for it to work by pressing the trigger button on both controllers and then moving the controllers in opposed directions. Other than that, everything else was performant and responsive, as you can see in the embedded screencast.

https://www.youtube.com/watch?v=Gjc0IR17kAk

Immersive 3D WebXR PWAs

PWA support on Oculus Quest is not limited to flat 2D apps. Developers can build immersive 3D experiences for VR using the WebXR API.

Wondering how various prompts (PWA install, permission requests, notifications) are handled from within VR, if at all?

Here's a screencast of User Agent Prompts test from the Immersive Web Working Group's WebXR Tests.

As you can see, entering VR mode requires the user's permission. Permissions are asked once per origin. Requesting permissions leaves the immersive mode. Notifications are currently not supported.

Hand tracking

You can use your hands to interact with PWAs in immersive mode thanks to the WebXR Hand Input API and Meta's AI-based hand-tracking system.

Here's a screencast of Hand Tracking Sample from the Immersive Web Working Group's WebXR Samples.

Augmented/Mixed Reality (Passthrough)

As announced at Meta Connect 2022, Meta Quest Browser has added support for WebXR Augmented Reality (AR), also known as Mixed Reality (MR), on Meta Quest 2 and Meta Quest Pro devices.

Let's check a slightly modified A-Frame starter example with scaled-down models and hidden sky and plane for augmented reality.

A-Frame is an open source web framework for building 3D/VR/AR experiences entirely out of declarative, reusable custom HTML elements that are easy to read, understand, and copy-and-paste.

https://glitch.com/edit/#!/a-frame-hello-world-ar

Here's a screencast of this demo on Meta Quest 2.

Meta Quest 2 has monochrome cameras, so the passthrough is in grayscale, while Meta Quest Pro has color cameras.

Conclusions

PWAs on Oculus Quest 2 are a lot of fun and very promising. The endless virtual canvas that lets users scale their screen to whatever fits the current task best has a lot of potential to change the way we work in the future. While typing in VR with hand tracking is still in its infancy and, at least for me, doesn't work very reliably yet, it works well enough for entering URLs or typing short texts.

What I like the most about PWAs on the Oculus Quest 2 is that they are just regular PWAs that can be used unchanged in a browser tab or through a thin APK wrapper without any platform-specific APIs. Targeting multiple platforms with the same code has never been easier. Here's to PWAs in VR and AR on the web. The future is bright!

Acknowledgements

Oculus Quest 2 photo by Maximilian Prandstätter on Flickr. Oculus Store images of Instagram, Facebook, Oculus Browser, and Spike apps as well as app discoverability illustration and hand

tracking animation courtesy of Meta. Hero image by Arnau Marín i Puig. This post was reviewed by Joe Medley.

□□□ #3
□ □□□ □□□ 2026-10-31 22:12:34 UTC
□ cocobook-sync □□□ 2026-11-04 17:37:54 UTC