Audion Browsers Portable - user guide

Contents

- How the window works
- First run
- Download Official Files
- Which browsers the DLL builder supports, and why
- What is inside a build
- Updating
- The certificates
- Build settings
- The working folders
- From the command line
- Worth knowing

This program makes portable browsers: ones that live in a folder, start from anywhere, and are never installed into Windows. Bookmarks, passwords and tabs stay inside that folder, so it travels on a flash drive or goes to someone else as it is.

How the window works

The program is opened with `Start.exe`.

Under the header stands a row of five large buttons, the sections: `INSTALL`, `DOWNLOADS`, `UPDATE`, `CERTIFICATE`, `SERVICE`. Press a section and its commands, with their settings, are on the left at once — there is nowhere to descend into.

A command with settings is a form. The button that runs it stands at the right end of the head of the form, filled with colour and named after the action: `BUILD`, `DOWNLOAD`, `UPDATE`. The rarely needed

settings lie under the heading `ADVANCED` — in sight, nothing has to be unfolded. A command without settings is a large button with a line of what it does.

Every choice is made with buttons: the chosen one is filled, the rest stay outlined. A browser is a button with its logo in the colour of that logo, so the row is read at a glance.

Captions are short. The explanation — what will happen, what a wrong answer costs — appears in the tooltip when the pointer rests on the control.

On the right are the working folders and the journal. `SOURCE` is where finished builds lie; `TARGET` is where new builds, archives and downloaded files go. The journal shows everything a command says; a download is one live line — how much has come, how much there is, and how fast. At the bottom is the status line with a clock and the `STOP` button.

At the top right are the language (`RU` / `EN`) and the theme, four of them. Both change at once, without a restart.

First run

1. Section `SERVICE` → `7-ZIP`. Nothing can be unpacked without it.
2. Section `INSTALL` → pick the browsers with the buttons and press `BUILD`.

Each browser is 120–240 MB of download and about half a gigabyte on disk. The builds appear in the Target folder (`output\Portable`); start one with `<Browser> Portable.cmd` in its root.

Building the same browser again is safe: when its build is already in the Target folder, only `App` inside it is replaced. The profile in `Data`, the cache and anything you put into that folder yourself stay as they were.

Choose a Target with a short path. Windows does not open files whose full path is longer than 260 characters, and a browser will not start from a folder that deep. The build is still made, and the program warns how many of the browser's files ended up past that limit: move the folder higher up — it is whole.

If one browser fails, the others are still built — the failures are listed at the end.

Download Official Files

Open `DOWNLOADS`, choose browsers and architecture, then press `DOWNLOAD`. The original files are saved under `Browser Downloads` in the Target folder. This command needs no 7-Zip and never runs

or unpacks a package or replaces DLLs.

- **Cent Browser**: portable self-extracting EXE, x86/x64.
- **LibreWolf**: portable ZIP, x64/ARM64.
- **Vivaldi**: installer, x86/x64/ARM64. Open it yourself and choose Advanced → Install Standalone (official instructions).
- **DuckDuckGo**: regular Windows installer, x64. This is an ordinary installation.

Existing verified downloads are reused after a SHA256 check. DuckDuckGo is downloaded to a temporary file on every run: an equal hash keeps the saved copy, and a different hash replaces it with the current installer under the original file name. Errors are listed per browser and do not stop the other downloads. Zen is omitted because its current official Windows release has no portable package (checked 2026-10-01).

Which browsers the DLL builder supports, and why

Those without a self-updating portable build of their own:

- **Google Chrome** — no portable version exists;
- **Yandex Browser** — no portable version, and the Russian CA certificates are already built into it;
- **Brave** — the portable builds are third-party and do not update;
- **Chromium-Gost** — speaks the GOST encryption that government portals need;
- **Ungoogled Chromium** — Chromium without Google's services.

The DLL builder excludes **Vivaldi**, **Cent Browser** and **Opera** — each has its own portable install that updates itself. Vivaldi and Cent Browser are available from the separate `DOWNLOADS` section. **Thorium** was tried and removed: 32-bit builds only, and store extensions do not install in it.

What is inside a build

Folder or file	What it is
`App`	The browser itself. Replaced wholesale on update.
`Data`	Your profile: bookmarks, passwords, tabs, extensions.
`Cache`	Cache. Safe to delete.
`Certificates`	The certificates and two files — install and remove.

Folder or file	What it is
`<Browser> Portable.cmd`	Starts the browser.
`Portable-Build.json`	Which versions are inside.

Updating

The `UPDATE` section. It has two parts, switched by the buttons over the form: `BROWSER` — check and update a whole build, `PORTABILITY LIBRARY` — replace the library alone and leave the browser as it is.

`CHECK` and `UPDATE` stand side by side in the head of one form and work from the same settings: look first, then change.

`CHECK` shows what has been published next to the version of your build. Nothing is downloaded.

A new Chrome reaches users by degrees, and Google does not put it into the installer at once. So the comparison is with the version already served to everyone, and a build made from the very installer Google holds right now counts as current — there is nothing to download yet.

`UPDATE` replaces only the browser inside the build - the settings file of the portability library (INI) stays as you left it, and a standard one is written only when the library is changed - and leaves `Data` and `Cache` alone, so the profile stays. A build whose browser and wrapper are both current is skipped without downloading.

No browser has to be ticked. With nothing ticked under `Browsers`, every build found in Source is taken: each one says what browser it holds. Ticks are for narrowing that down — updating only Yandex, for example.

There can be several builds. Put as many builds into Source as you like — different browsers, or several of the same one — each in a folder of its own, and they are updated in one run. A failure with one does not stop the rest.

The update happens where the build lies. Point Source at its folder — a flash drive, a network share, wherever it lives — and it is updated in place. Nothing has to be copied, and the Target folder is not used here: that one is for new builds.

With no build in Source, the program looks in `output\Portable` — at what it made itself. If a ticked browser is not there either, the update ends with an error and says where it looked: "done" means something was really updated, or was current already.

Source may also point straight at one build — at its own folder. It is updated in place, together with its profile, and is not renamed.

A build can come from elsewhere. Put a portable browser made anywhere into Source: the folder can be called anything, and the browser can sit in `Chrome\`, in `App\Chrome-bin\`, or right in that folder. The program finds the browser by its executable and the profile by its `Local State` file, and the update brings the build to its own layout: `App`, `Data`, `Cache`. The profile is only renamed into place — never copied and never deleted. `CHECK` says beforehand that the layout is not standard.

Nothing that lay beside the browser is deleted. Only the browser itself is replaced. Your own folders and files stay where they were, and what is in the way of the new layout — the previous packer's launchers and its settings — is moved into `Old files` inside the build. Delete that folder yourself once you are sure nothing in it is needed.

The folder takes the standard name (for example `Google Chrome Portable`) when it is the only build of that browser in Source. Several builds of one browser keep their own names — those are how you tell them apart. The folder Source points at directly is not renamed either.

The browser has to be closed: otherwise the update of that build stops before the download, having touched nothing. Chromium-Gost and Ungoogled Chromium cannot be told apart by their files: such a build is recognised by its folder name (`Chromium-Gost Portable`, `Ungoogled Chromium Portable`) or by the tick — tick one of the two, and a folder of any name is taken to be that one. A tick does not take another browser: when `chrome.exe` calls itself Cent Browser, Supermium or Iron, the build is left as it was, and the journal says the program does not build that browser.

A folder of your own beside the browser is not touched, even one named like a date — `10.10.2026` or `2025.03.01`: only a folder that holds the browser's files is taken for its version folder.

`UPDATE DLL ONLY` refreshes the selected library in existing builds. Select the browsers, library and architecture (`Auto` reads each browser executable). Source accepts a build or a parent folder containing several builds: they are found the way `CHECK` and `UPDATE` find them — at any depth and under any folder name, every build of a ticked browser. A build made elsewhere, whose browser is not in `App` yet, is first brought to the program's layout with `UPDATE`. No browser installer is downloaded and the whole `App` is not replaced. The same library keeps its INI; switching libraries creates the standard INI. If a file is in use, changes are rolled back: close the browser and retry.

`ADVANCED` accepts optional HTTP(S) URLs of full Chrome/Yandex installers. Blank uses the official source; a custom file is fetched afresh during build/update. A custom Chrome version is known after unpacking. `CHECK` reports an unknown version and `UPDATE` does not skip it based on Google's stable release API.

`Disable Yandex updater` is on by default during build and update. It removes only `service_update.exe` inside Yandex's `App`, including an already-current build. Other browsers and profile files are untouched. Off keeps the updater in a newly assembled `App`; restoring a previously removed file requires a rebuild.

The certificates

Russian state sites are signed by an authority Windows does not trust out of the box, so such a site opens with a security warning. The steps are separate so that nothing happens by itself.

While building and updating — the checkbox `State site certificates`. It puts the files and two shortcuts, install and revoke, into the build. It installs nothing. Unticked, it removes nothing from a build: files put there earlier stay.

Before it is used, every certificate file is compared by its thumbprint with the one written into the program. A file of the same name and other contents is not used: it is set aside under another name, and the certificate is downloaded anew.

Section `CERTIFICATE` → `ADD TO BUILD` saves the two certificate files and install/revoke commands under `Certificates` in selected existing builds. Point Source at a build or its parent folder. Chrome is selected by default. Browser/profile files and Windows trust remain unchanged.

Section `CERTIFICATE` → `INSTALL` — adds them to your own Windows account's store, which every Chromium browser reads. No administrator rights; other users of the machine see no change.

`REVOKE` — removes exactly those two certificates by fingerprint; anything else stays.

`CHECK` shows whether they are installed and changes nothing.

Yandex Browser needs none of this: the certificates are already in it.

Build settings

Portability. The default library is Proxy library (x86/x64), with every optional switch in `App/version.ini` set to `0`. Alternatives are Chrome++ and Vivaldi++, both supporting x86/x64/ARM64. A failed release lookup or download falls back to the archive of the selected library kept from an earlier download (in `Portable\_archives` of the Target folder), verified again. The generated launcher starts in `App` so relative `Data` and `Cache` paths stay inside the build. Registry cleanup applies only to Chrome++.

Wrapper architecture. Leave it at `AUTO`. The program reads whether the browser is 32- or 64-bit and uses a matching DLL from the selected library. This is not a formality: a 32-bit browser with a 64-bit wrapper starts as if nothing were wrong, but its profile goes to the system and the build stops being portable.

Clean registry with Chrome++. Applies only to Chrome++. Off by default. With it on, the browser wipes its own registry branch when it exits. That branch is shared with an installed copy of the same browser, so turn it on only where that browser is not installed.

Pack into an archive. Turn it on when the builds are to be handed over: one file instead of a folder. The format sits next to it — `ZIP` opens anywhere, `7Z` is smaller but needs 7-Zip on the other side. On update the archive is named after the build's folder, so several builds of one browser give an archive each. The archive of an updated build holds the browser, its launcher and the certificates - **without the profile and the cache**: bookmarks, passwords and history stay with you, in the build itself.

Stop. The `STOP` button in the status line. A command stops by itself, at its next safe point: a download is cut, unpacking and packing are ended, a build that has not been touched stays as it was, and the operation ends as stopped. If the browser is already being put in place, that one replacement is finished and the remaining builds are not started. An archive is either whole or not there — no half of an archive is left under the real name.

Closing the window while a command runs asks first and does the same: it stops the command and closes when the command has stopped.

Keep working files (under `ADVANCED`). Unpacked installers stay in `Portable\_tmp` in the Target folder — useful when a build failed and the reason has to be found. The downloaded files themselves lie beside it either way, in `Portable\_archives`: one per browser, older versions are removed by themselves.

Work around Defender. The portability library — `version.dll` — is sometimes taken for a threat by Windows Defender right in the middle of a build. With this on, the Source and Target folders are excluded from scanning for the length of the operation and put back the moment it ends; Windows asks for permission once. Where Defender is off, the checkbox does nothing and asks nothing. If an operation was cut short and the exclusion stayed, `SERVICE` → `DEFENDER: REMOVE EXCLUSION` takes it back — from the folders of earlier runs too.

The working folders

`SOURCE` and `TARGET` are at the top right. At every start they are the `input` and `output` folders of the program itself; another folder can be picked with the button, typed in, or dropped onto the window.

- The clock button opens the history of paths; the pin keeps a path in it.
- `RESET` returns both folders to `input` and `output`.
- `LIST` shows in the journal what lies in the Source.
- `CLEAR` deletes the **contents** of the chosen folder — the folder itself stays. It asks first what to clear and names the paths. What is deleted goes to the Recycle Bin; a flash drive and a network share have none, and the question says so beforehand. A folder outside the program is asked about separately: it may hold builds with somebody's profile. The root of a drive, the program itself and its own service folders cannot be cleared.

`CLEAR WORKSPACE` in the header removes the working material: the downloaded installers and what was unpacked from them (`Portable\_archives` and `Portable\_tmp` in the Target folder). Finished builds are not touched; the next build downloads what it needs again.

From the command line

Everything a button does, a command does too — with the same settings:

```
browsers_cli.cmd describe
browsers_cli.cmd run --operation browsers_build --set browsers=chrome,brave --target
D:\Portable
browsers_cli.cmd run --operation browsers_check --source E:\Flash\Browsers
browsers_cli.cmd run --operation browsers_update --source E:\Flash\Browsers
```

`describe` prints every section, command and setting with its name. The command the window runs an operation with is written into the journal at every start — it can be copied from there. Exit codes: `0` done, `1` failed or done in part, `130` stopped, `2` the command was written wrong, and then nothing is run.

Worth knowing

The first start of any build takes longer — the browser is laying out its profile.

A portable build and an installed browser of the same name run side by side without disturbing each other.