

Install and Launch

[Русский](#) · [User Guide](#) · [Reference](#)

Contents

- [Requirements](#)
- [Launching](#)
- [Building the Environment](#)
- [Building the Window](#)
- [Dependencies](#)
- [Project Folders](#)
- [Maintenance](#)
- [Before a Large Run](#)

The project is portable: it is not installed into the system, writes nothing to the registry, and lives entirely inside its own folder. Copying the folder to another machine is a sufficient "install".

Requirements

Windows. The program targets Windows: moving columns by address and converting old `.doc` / `.xls` files work through Word and Excel.

Excel. Needed by the "Move columns by address" command — it is what preserves the formulas, formatting, and row order of the primary workbook. Word is needed only to convert `.doc` files. Every other command works without them.

Internet. Needed once to build the environment, and occasionally afterwards to download the OKTMO registry.

A built portable project needs nothing else: the window lives in `App\` together with its own .NET runtime, and Python lives in the `runtime` folder. Neither .NET nor PowerShell has to be installed on the machine.

Launching

Way	What it does
<code>Start.exe</code>	Open the program window
<code>App\AudionAddressProcessor.exe</code>	The same, directly
<code>address_cli.cmd</code>	The console command: the same operations without a window (see the Reference)
<code>builder_main.cmd</code>	Portable environment build

`Start.exe` starts `App\AudionAddressProcessor.exe` and steps aside. A second start from the same folder does not open a second window — it brings the first one to the front.

The window finds the program folder by where it lies (`App\` sits in the root) and calls the Python engine from `runtime\`: `runtime\python.exe`, then `runtime\python\python.exe`. Every operation runs as a process of its own, so STOP and closing the window leave behind neither Python nor Excel nor Word.

The window's settings live in `config\window_settings.json` (size, language, theme, font size), the history of paths in `config\path_history.json`. On the first start the window opens in Russian; English is switched on with the EN button and remembered.

If the window does not open

Common causes:

- **No `App\`.** `Start.exe` says what builds it: `install\Build-App.cmd`.
- **The environment is not built.** The window opens, but its journal says there is no project Python and operations do not start — build the environment (`builder_main.cmd`, step `[01]` or `[02]`).
- **The window crashed.** The details are written into `logs\gui-crash-TIME.log`, the course of the day into `logs\gui-DATE.log`.

To check the window–command–manifest chain without showing the window:

```
App\AudionAddressProcessor.exe --self-check=check.json
```

The window builds every section, compares them with the manifest and writes into the file how many operations and fields it drew; exit code 0 means everything is in place.

Building the Environment

Needed only if you received the project without a `runtime` folder or are rebuilding it. The build refreshes the components — Python and its packages; it does not touch the window (`App\`).

`builder_main.cmd` is the build menu:

Item	What it does
[01] PYTHON ENV CMD	Build the portable environment (the main path)
[02] PYTHON ENV PS	The same through the program's own PowerShell (<code>App\AudionPs.exe</code>)
[05] WINDOW	Rebuild <code>App\</code> from <code>Source\</code> — only after the window sources changed
[09] PORTABLE OFFLINE	Install from the local <code>wheelhouse</code> , no network
[70] CLEAN INSTALL CACHE	Clear the build cache and what building the window leaves behind; the built <code>runtime</code> and <code>App\</code> stay
[71] VERIFY / DOCTOR	Check the environment, the console command and the built window

Installing without the internet

If the project has a `wheelhouse` folder with the dependency wheels, the environment is built from it: `install\install_portable_offline.cmd` or item `[09] PORTABLE OFFLINE`.

Checking

`[71] VERIFY / DOCTOR` — or `install\verify_portable_env.cmd` — checks the interpreter and the import of every dependency, that the console command reads the manifest, that `Start.exe` and `App\` are in place and that the window draws every operation and field of the manifest. It is the first thing to run when something does not work.

Building the Window

The window is a program on Avalonia 12 and .NET 10; its sources are in `Source\`. The built window lies in `App\` and travels with a release as it is, so an ordinary environment build does not rebuild it.

Rebuild it after the sources in `Source\` change, or when the manifest gets something the window cannot draw yet:

```
install\Build-App.cmd
```

It needs the .NET 10 SDK. The script runs the window checks (`tests\test_window_avalonia.py`), builds the window and the program's own PowerShell `AudionPs.exe`, checks what was built (`--self-check`), and only then replaces `App\`: a failed build leaves the previous working copy in place. The number comes from `config\version.json`.

`Start.exe` is built separately: `install\Build-StartLauncher.cmd`.

Dependencies

Package	What for
<code>pandas</code>	Table operations
<code>openpyxl</code>	Reading and writing XLSX
<code>rapidfuzz</code>	Approximate comparison of street names
<code>pymorphy3</code> , <code>pymorphy3-dicts-ru</code>	Case forms of settlement names
<code>PyMuPDF</code>	The text layer of PDF
<code>pywin32</code>	Working with Excel and Word
<code>pyyaml</code>	The manifest and settings files
<code>tqdm</code>	The course of long stages
<code>setuptools</code> , <code>wheel</code> , <code>packaging</code>	Building the environment
<code>pytest</code>	Tests

The window depends on no Python package. The license texts of all third-party components are in `licenses/`.

Project Folders

Folder	What is in it	May be cleaned
<code>App</code>	The built window and the program's own PowerShell	No, rebuilt from <code>Source</code>
<code>Source</code>	The window sources	No
<code>system_core</code>	Source code of the engine and the console command	No
<code>config</code>	Settings, manifest, window layout, OKTMO keys and pins	No
<code>data</code>	OKTMO registry and the region reference	The registry downloads again
<code>runtime</code>	Portable Python	Rebuilt
<code>wheelhouse</code>	Wheels for the offline install	Yes
<code>input</code>	Source files	Yes
<code>output</code>	Results	Yes
<code>report</code>	Run reports	Yes
<code>logs</code>	Operation and window journals	Yes
<code>workspace</code>	Intermediate files	Yes
<code>release</code>	Result archives	Yes
<code>docs</code>	Documentation	No
<code>licenses</code>	Licenses of third-party components	No
<code>tests</code>	Tests	No

The working folders are first created by `install\init_folders.cmd`.

Maintenance

Project cleanup. `cleanup_project.cmd` removes accumulated artifacts: caches, temporary conversions, journals, reports, the Rosstat snapshot `data\rosstat` (it downloads again) and the rebuildable `runtime` and `wheelhouse`. Code, settings, documentation, sources, the built window `App\` and approved references are kept. Mind that the cleanup empties both `input` and `output` — take what you need out of them first.

CMD file encoding. After editing any `.cmd`, run `install\Check-CmdEncoding.cmd`: the files must stay UTF-8 without BOM with CRLF line endings. Otherwise the Cyrillic in the menus turns into garbage.

Updating dependencies. `install\Update-Requirements-Lock.cmd` regenerates the lock file.

The OKTMO registry is updated not from here but with a button in the window — see [OKTMO](#).

Before a Large Run

Check the environment (`VERIFY / DOCTOR`), work on copies of important tables, and keep the sources, the reference and the results in different folders.

Make the first run on a new data set on ten rows whose answer is known in advance.