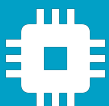




Data sheet - User manual



Contents

1	Introduction	3
1.1	Features	3
2	Installation	4
2.1	Installation on Windows	5
2.2	Installation on Linux	5
2.2.1	Linux on ARM64	5
2.3	Installation on macOS	6
3	Using the application	7
3.1	Connecting to a device	8
3.2	Connecting to a DRS1	9
3.2.1	On Linux	9
3.2.2	On macOS	9
3.2.3	On Windows	10
3.3	Sending AT commands	11
3.4	The log view	12
3.5	Recording a session to a file	12
3.6	Editing device settings	13
3.7	Exporting and importing a configuration	14
3.8	Updating the firmware	16
3.9	Application information	17
4	Disclaimer	18

1 Introduction

Aerobits APP is a desktop application for configuring and updating Aerobits devices over a serial (UART/USB) connection. It provides a terminal for AT commands, a table-based editor for every setting the device exposes, and a firmware updater - all without needing a separate terminal emulator or command-line tools.

1.1 Features

- **Quick start with the products of AEROBITS company**
- **Cross platform** - Windows, Linux, Linux ARM and macOS
- **Automatic detection of connected serial ports**
- **Terminal for AT commands with a resendable command history**
- **Settings editor listing every parameter with its description and permitted values**
- **Export and import of a device configuration as a file**
- **Restore of the device's default settings**
- **Firmware update from an .aff file, with live progress**
- **Optional session log written to a file for diagnostics and bug reports**

For more information please contact support@aerobits.pl.

2 Installation

The application is distributed as a ZIP archive per platform. There is no installer and nothing is written outside the folder you unpack it into, apart from the optional log file described in [Recording a session to a file](#) (page 12).

Download the archive matching your system from the release page:

System	Archive	Run
Windows (64-bit)	serial_app_at_commands_windows-vX.Y.Z.zip	Aerobits_APP.exe
Linux (64-bit, x86)	serial_app_at_commands_linux-vX.Y.Z.zip	Aerobits_APP
Linux (ARM64)	serial_app_at_commands_arm-vX.Y.Z.zip	Aerobits_APP
macOS (Apple Silicon)	serial_app_at_commands_macos-vX.Y.Z.zip	Aerobits_APP.app

2.1 Installation on Windows

Unpack **serial_app_at_commands_windows-vX.Y.Z.zip** and open the resulting folder. Start the application by double-clicking **Aerobits_APP.exe**.

Keep the whole folder together - the executable loads the libraries and the firmware updater from the files next to it, so moving **Aerobits_APP.exe** out on its own will not work.

Windows drivers for the USB-to-serial converter used by Aerobits devices are installed automatically on any current Windows version. The device then appears as a COMx port in the **Port** list.

2.2 Installation on Linux

Unpack the archive and run the application from the folder:

```
unzip serial_app_at_commands_linux-vX.Y.Z.zip
cd artifacts_lin
./Aerobits_APP
```

If the file is not executable after unpacking, set the flag once:

```
chmod +x ./Aerobits_APP
```

Serial ports appear as `/dev/ttyUSB0` or `/dev/ttyACM0`. On most distributions these belong to the `dialout` group, and a normal user cannot open them until they are a member of it:

```
sudo usermod -a -G dialout $USER
```

Log out and back in for the change to take effect, otherwise the application reports a permission error when connecting.

2.2.1 Linux on ARM64

The ARM64 archive is unpacked the same way, but a ZIP does not carry Unix permissions, so both the application and the firmware updater arrive without their execute flag. Set both before the first run:

```
unzip serial_app_at_commands_arm-vX.Y.Z.zip
cd artifacts_arm
chmod +x ./Aerobits_APP _internal/deps/upd_app_bbb
./Aerobits_APP
```

Note:

The two are needed at different moments. Without the flag on **Aerobits_APP** nothing starts at all. Without it on **_internal/deps/upd_app_bbb** the application runs and connects normally, and only **Firmware Update** fails - the updater is a separate program the application launches, so its flag is not needed until then.

2.3 Installation on macOS

Unpack **serial_app_at_commands_macos-vX.Y.Z.zip** and drag **Aerobits APP.app** to your Applications folder, or run it from wherever you unpacked it.

The application is signed but not notarized by Apple, so a copy downloaded from the internet is held in quarantine and macOS refuses to open it on the first attempt. Release it once with:

```
xattr -dr com.apple.quarantine "Aerobits APP.app"
```

Alternatively, right-click the application and choose **Open**, then confirm - or allow it under **System Settings -> Privacy & Security -> Open Anyway**. This is only needed the first time.

macOS requires **version 15 (Sequoia) or newer**, and the build runs on Apple Silicon.

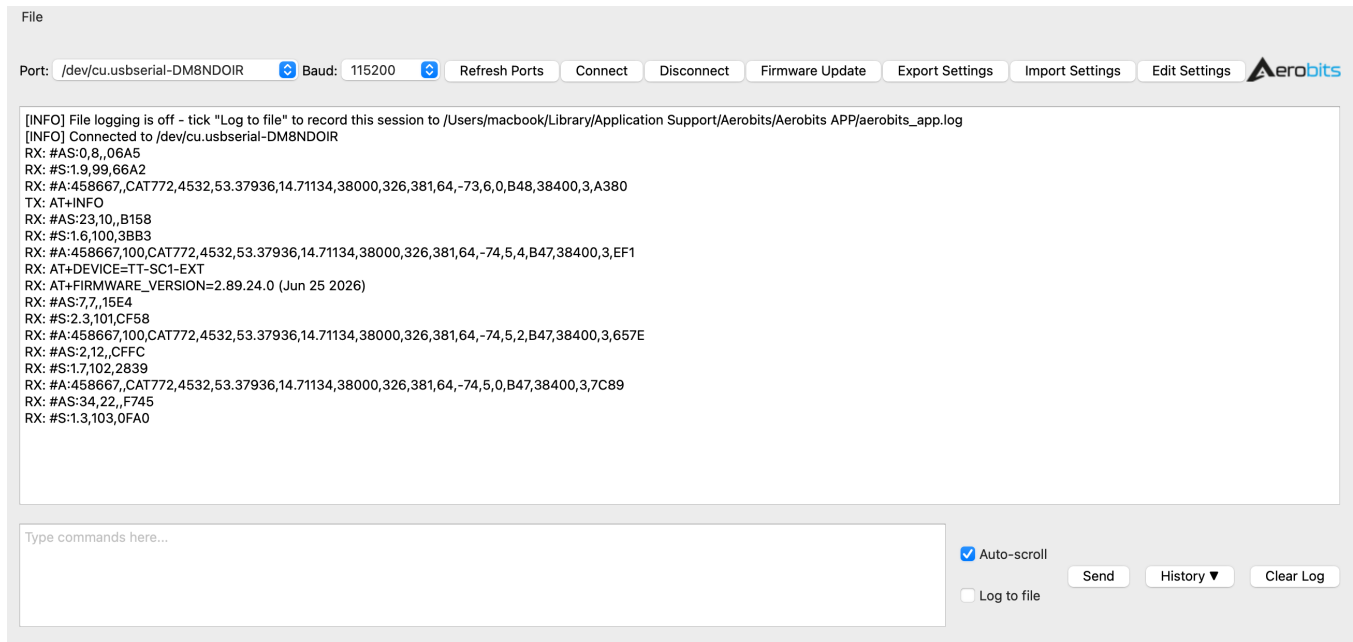
No driver installation is required. Aerobits devices appear as `/dev/cu.usbserial-*` in the **Port** list.

Note:

On macOS the menu bar is part of the system menu bar at the top of the screen, not inside the application window. **About Aerobits APP** and **Quit** are found under the bold **Aerobits APP** menu, and **Open log folder** under **File**.

3 Using the application

The main window is split into three areas: the connection bar at the top, the log view in the middle, and the command box with its buttons at the bottom.



3.1 Connecting to a device

1. Connect the device to a USB port.
2. Pick it from the **Port** list. The list is filled in when the application starts; use **Refresh Ports** if the device was plugged in afterwards.
3. Select the **Baud** rate the device is configured for. Aerobits devices leave the factory at **115200**, which is the default here. The other supported rates are 57600, 921600 and 3000000.
4. Press **Connect**.

Once connected, everything the device sends appears in the log view prefixed with **RX:**, and everything the application sends is shown as **TX:**. Press **Disconnect** to release the port.

Each entry in the **Port** list carries the description the adapter reports alongside the port name - **ACM0 - Aerobits**, for example - so the right device can be picked without connecting to it first. Open the list to see them; the name on its own is shown once a port is selected.

Note:

If a wrong baud rate is selected the device still appears to connect, but the log fills with unreadable characters. Disconnect, pick the correct rate and connect again.

3.2 Connecting to a DRS1

A DRS1 is reached over the network rather than through a USB-to-serial adapter, so it does not show up as a `/dev/ttyUSB0`. On Linux and macOS you can bridge it to a virtual serial port with **socat**, which the application then lists like any other port.

3.2.1 On Linux

Install socat if it is not already present:

```
sudo apt install socat
```

Create the virtual port, giving the link a name that starts with `/tmp/tty`:

```
socat pty,link=/tmp/ttyV0,b115200,raw,echo=0 tcp:<drs1-address>:<drs1-port>
```

3.2.2 On macOS

Install socat with [Homebrew](#):

```
brew install socat
```

Then create the virtual port:

```
socat -d -d -d pty,link=/tmp/ttyV0,rawer,echo=0 tcp:<drs1-address>:<drs1-port>
```

Note:

Two differences from the Linux command. There is no `b115200`: the `b<rate>` options are a Linux-only part of socat, and a macOS build rejects the command with unknown option "`b115200`". And `rawer` is used instead of `raw`, which switches off the remaining line processing that would otherwise alter the data.

`-d -d -d` only raises socat's own logging, which is useful while setting the link up; leave it out once it works.

Leave that command running. In the application press **Refresh Ports**: `/tmp/ttyV0` appears in the **Port** list, and you select it and press **Connect** exactly as for a directly attached device.

Note:

The name matters. The application looks in `/tmp` for entries whose name begins with `tty`, so a link created anywhere else, or called anything else, is not found. Create only one at a time - with several present the application cannot tell them apart.

Note:

The **Baud** setting in the application has no effect on a virtual port, and neither does the speed given to socat. A virtual port has no UART behind it to clock bytes out, so the setting is recorded but never applied. The speed that actually matters is the one configured on the DRS1's own serial interface.

Start socat before pressing **Refresh Ports**. The application looks for the port for about two seconds and then gives up, so a link created afterwards is only picked up by pressing **Refresh Ports** again.

Stopping socat removes `/tmp/ttyV0`. Press **Disconnect** in the application first, then stop socat with `Ctrl+C`.

3.2.3 On Windows

Everything above applies to Linux and macOS only. Windows has no equivalent of a pty, so socat cannot conjure up a port for the application to open. The port has to exist beforehand: a virtual COM port driver is installed to emulate a pair of linked ports, socat is pointed at one of them, and the application connects to the other - which Windows presents as an ordinary COMx.

Selecting it in the **Port** list is then no different from a directly attached device.

Note:

The emulation step depends on the driver used, so the procedure is not repeated here. Follow the DRS1 documentation for the tool to install, how to create the port pair and which port to give to socat.

3.3 Sending AT commands

Type a command into the box at the bottom and press **Send**. The command is transmitted with a trailing carriage return and line feed, and the reply appears in the log view.

Useful commands to start with:

Command	Meaning
AT+INFO	Display device type and firmware version
AT+HELP	List every setting with its description
AT+CONFIG=2	Enter CONFIG mode, required before changing settings
AT+CONFIG=0	Return to RUN mode
AT+REBOOT	Restart the device

Note:

Most settings commands are rejected in RUN mode with AT+ERROR (Command unavailable in RUN state). Send AT+CONFIG=2 first, or use the settings editor described below, which handles the mode change for you.

Every command that is sent is remembered for the session. Press **History** to pick an earlier command and put it back into the command box, ready to edit and resend:

AT+HELP
AT+INFO

3.4 The log view

Auto-scroll keeps the newest line in sight. Scrolling up to read something older suspends following the tail until you scroll back to the bottom, so the view is not yanked away while you are reading.

Clear Log empties the view. It does not affect the log file.

3.5 Recording a session to a file

Tick **Log to file** to write everything shown in the log view to a file as well. This is off by default, so a device streaming continuously does not fill the disk unless you ask for it. Ticking and unticking it during a session pauses and resumes recording; the file records where that happened, so a gap is not mistaken for lost data.

Lines in the file carry a timestamp, which the view does not show. The file grows to 5 MB, then rotates, keeping five older files.

File -> Open log folder opens the folder holding it. The location is:

System	Path
Windows	%APPDATA%\Aerobits\Aerobits APP\aerobits_app.log
Linux	~/.local/share/Aerobits/Aerobits APP/aerobits_app.log
macOS	~/Library/Application Support/Aerobits/Aerobits APP/aerobits_app.log

Please attach this file when reporting a problem to support@aerobits.pl.

3.6 Editing device settings

With the device connected, press **Edit Settings**. The application switches the device into CONFIG mode, reads every supported parameter and shows them grouped by category:

Read-only values are greyed out. Edit the Value column to change a setting.

Group	Setting	Value	Description
SYSTEM	AT+BAUDRATE	0	Baudrate of serial interface (0:115200, 1:921600, 2:3000000, 3:57600)
SYSTEM	AT+BOOT	0	Is firmware in bootloader mode
SYSTEM	AT+CONFIG	2	CONFIG mode (0:disable, 1:baudrate 115200, 2:baudrate as set)
SYSTEM	AT+DEVICE	TT-SC1-EXT	Device type's name
SYSTEM	AT+FIRMWARE_VERSION	2.89.24.0 (Jun 25 2026)	Device's firmware version
SYSTEM	AT+LOCK	0	Device in CONFIG mode (0:no lock, 1:lock)
SYSTEM	AT+MAVLINK_INPUT_ENABLED	0	Enable MAVLINK input (0:disable, 1:enable all, 2:enable except GNSS)
SYSTEM	AT+SERIAL_NUMBER	09-0001109	Device's serial number
SYSTEM	AT+SYSTEM_LOG	0	System logs (0:disable, 1:enable)
SYSTEM	AT+SYSTEM_STATISTICS	CSV	None, CSV, JSON
ADSB	AT+ADSB_RX_PROTOCOL_DECODED	CSV	ADS-B decoded protocol: None, CSV, Mavlink, JSON, GDL90, ASTERIX
ADSB	AT+ADSB_RX_PROTOCOL_INC	0	Reporting mode of decoded ADS-B targets (0:once per second,always, 1:once per second,if data updated, 2:immediately,only after position update)
ADSB	AT+ADSB_RX_PROTOCOL_RAW	NONE	None, HEX, BEAST, JSON, HEXd
ADSB	AT+ADSB_STATISTICS	CSV	None, CSV, JSON

Restore Defaults Cancel Save to Device

Each row shows the parameter group, the AT command that sets it, its current value and a description listing the permitted values. Values that the device reports as read-only - such as the device type, serial number and firmware version - are greyed out and cannot be edited.

Reading them takes a few seconds: the application asks the device about every setting in turn, so that it knows which ones may be changed. Progress is shown while it does.

To change a setting, click its **Value** cell and type the new value. Press **Save to Device** to write every changed row, or **Cancel** to discard the changes. Only rows you actually edited are written.

Every command sent and every reply received is printed in the log view, so a configuration change can be read back afterwards or attached to a support request. A value the device will not accept - one outside its permitted range, for example - is reported there and in a warning window rather than being counted as saved.

Restore Defaults loads the device's factory configuration. It asks for confirmation first, because the current configuration is lost.

The device is returned to RUN mode when the editor closes, whichever button you used.

Some settings only take effect after the device restarts. If one of those was changed, the device reboots as the editor closes. The application waits for it to come back - following it if it reappears under a different port name - and reconnects on its own, so there is nothing to do but wait for the log view to report it.

3.7 Exporting and importing a configuration

Export Settings writes the device's current configuration to a file, and **Import Settings** applies a previously exported file to a connected device. This is the quickest way to give several devices an identical configuration, or to keep a record of how a device was set up before changing it.

Both buttons require a connected device and open a file chooser for the .json file to write or read.

The exported file is plain JSON and can be read or edited in any text editor. A configuration exported from a TT-SC1-EXT looks like this:

```
{
  "settings": {
    "device": {
      "any": {
        "AT+BAUDRATE=": "0",
        "AT+MAVLINK_INPUT_ENABLED=": "0",
        "AT+SYSTEM_LOG=": "0",
        "AT+SYSTEM_STATISTICS=": "CSV",
        "AT+ADSB_RX_PROTOCOL_DECODED=": "CSV",
        "AT+ADSB_RX_PROTOCOL_INC=": "0",
        "AT+ADSB_RX_PROTOCOL_RAW=": "NONE",
        "AT+ADSB_STATISTICS=": "CSV"
      }
    },
    "special": {
      "AT+BOOT=": "0",
      "AT+CONFIG=": "2",
      "AT+DEVICE=": "TT-SC1-EXT",
      "AT+FIRMWARE_VERSION=": "2.89.24.0 (Jun 25 2026)",
      "AT+LOCK=": "0",
      "AT+SERIAL_NUMBER=": "09-0001109"
    }
  },
  "version": 1
}
```

Each key is the AT command that sets the value, including the trailing =, so a line reads back as the command that would be sent - AT+SYSTEM_STATISTICS=CSV. The exact set of keys depends on the device type and its firmware; the example above is only what one particular device reports.

The two groups are treated differently:

device / any

The adjustable settings. **Import Settings** writes every entry in this group to the device.

special

Everything that is recorded but not written back, for one of two reasons. Some entries identify the device or report its state rather than configure it - device type, firmware version, serial number, bootloader and config mode. Others are settings the device does not keep when it restarts, such as the lock state or a callsign; writing those back from a file would restore a value the device itself would have discarded. Both are **ignored on import**, so a file exported from one device will not overwrite another one's identity or reinstate something that was only ever temporary.

The application asks the device which of its settings it will accept and which it stores permanently, so the split above follows the device in front of it rather than a fixed list. A setting your device treats differently may therefore appear in the other group.

Import Settings reports what the device accepted. Anything it refuses is listed in the log view and in a warning window, so a file that does not suit the device cannot quietly appear to have been applied.

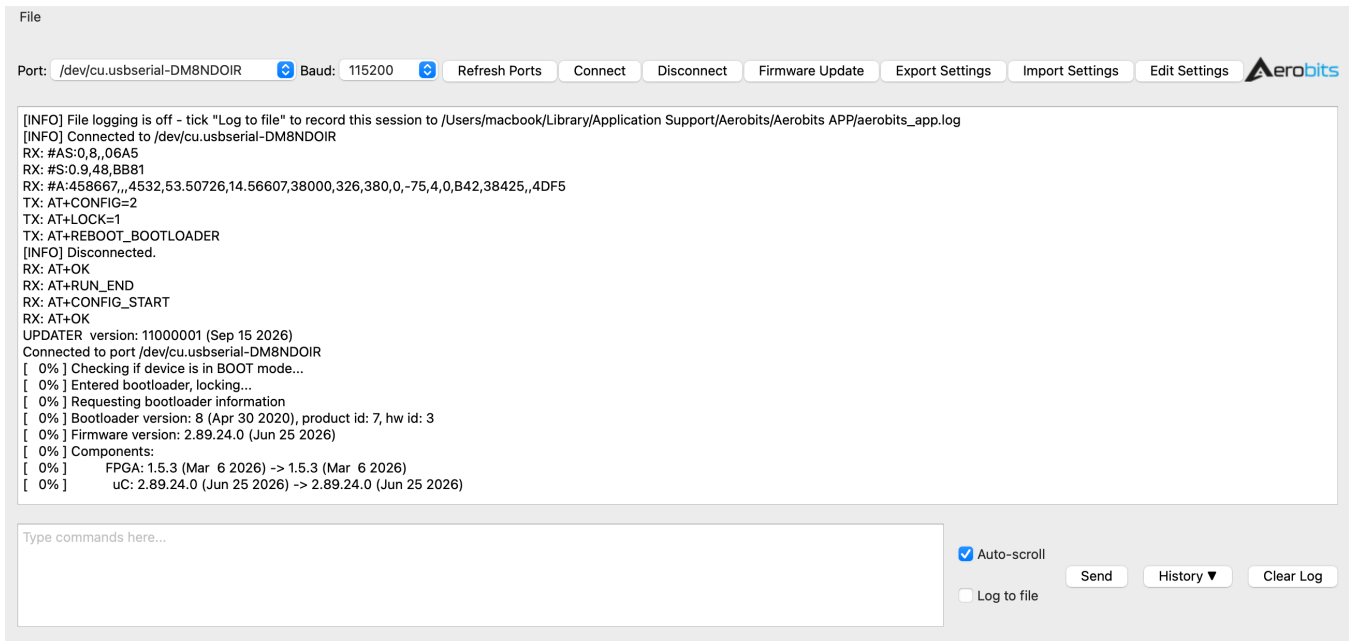
Note:

Importing does not check that the file came from the same device type. A file exported from a different product may contain commands the target device does not recognise, which it rejects one by one. Keep exports for different device types in separate files.

3.8 Updating the firmware

Firmware is supplied by Aerobits as an .aff file.

1. Connect to the device as usual.
2. Press **Firmware Update** and select the .aff file.
3. The application locks the device, reboots it into its bootloader and starts the updater. Progress is reported in the log view.



Before writing anything the updater reports the bootloader version and each component's current and new version, so you can confirm the file matches the device. The update finishes with `Operation completed successfully`, after which the device leaves bootloader mode and the application reconnects on its own.

Warning:

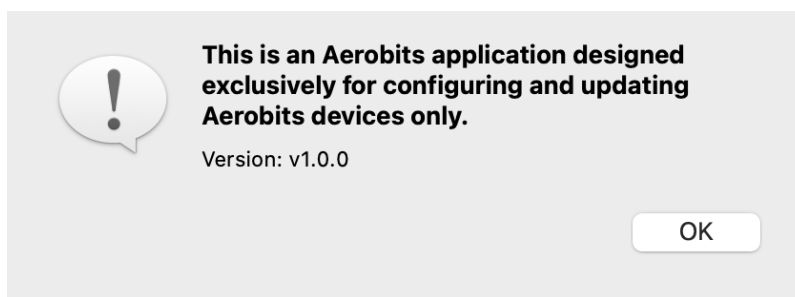
Do not unplug the device or close the application while an update is running. An interrupted update can leave the device unable to start.

Note:

The .aff file must match the device type. The updater checks the product and hardware identifiers and refuses a file intended for a different product.

3.9 Application information

File -> About Aerobits APP reports the application version. On macOS this entry is in the **Aerobits APP** menu instead.



4 Disclaimer

Information contained in this document is provided solely in connection with Aerobits products. Aerobits reserves the right to make changes, corrections, modifications or improvements to this document, and to products and services described herein at any time, without notice. All Aerobits products are sold pursuant to our own terms and conditions of sale. Buyers are solely responsible for the choice, selection and use of the Aerobits products and services described herein, and Aerobits assumes no liability whatsoever, related to the choice, selection or use of Aerobits products and services described herein. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services, it shall not be deemed a license granted by Aerobits for use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering use, in any manner whatsoever, of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN AEROBITS TERMS AND CONDITIONS OF SALE, AEROBITS DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO USE AND/OR SALE OF AEROBITS PRODUCTS INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED AEROBITS REPRESENTATIVE, AEROBITS PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE.

Information in this document supersedes and replaces all previously supplied information.

© 2024 Aerobits - All rights reserved