WiCOS64

REMOTE STORAGE

Admin Guide & Technical Documentation

Version v1.0.1.33 - as of 2026-02-04

- Server + Admin UI + W64Tool
- Disk image support (D64/D71/D81) - read/write (optional)
- D81: 1581 partitions ("DIRs") + optional auto-resize (repack)
- MKDIR/RMDIR: create/delete images + create/delete D81 partitions
- Bootstrap & Discovery (HTTP/UDP) + optional MAC extras
- Token management, quotas & trash mode
- Ops Playground (CLI) incl. history

Document information

This document covers installation, operation, administration and the technical background of the WiCOS64 Remote Storage Server, including the Admin UI, the Windows tray app and the CLI tool **w64tool** (state: v1.0.1.33).

Note

WiCOS64 is a remote storage server for the Commodore 64 (C64) in combination with a WiC64 module. This document is intended as a reference: you can read it from start to finish, or jump directly to the chapters you need for a specific task.

Audience

- Users running the server inside a home network (LAN).
- Developers writing their own C64 programs/tools against the Remote Storage API.
- Advanced users who want to configure tokens, quotas, disk images, or discovery/bootstrap.

Table of contents

1 Overview

The WiCOS64 Remote Storage Server provides the C64 - via a WiC64 module - with a virtual file system on the network. The client does not talk to SMB/NFS directly. Instead, it uses a compact binary Remote Storage API that is transported via HTTP POST.

Optionally, disk images (**.d64**/**.d71**/**.d81**) can be mounted transparently as directories. Depending on the feature toggles, images are not only readable but can also be written, renamed, copied and deleted.

1.1 Components

- **Server** (**wicos64-server.exe**): provides the Remote Storage API, manages tokens/quotas, optional trash mode, and serves disk images as virtual directories.
- **Admin UI**: web interface for status, configuration, tokens & quotas, live log, diagnostics and the Ops Playground.
- **Windows tray app** (**wicos64-tray.exe**): convenient start/stop/restart and quick access to Admin UI/logs/config (not a Windows service).
- **w64tool** (CLI): smoke-test/debug tool (CAPS/PING/LS/APPEND/HASH/SEARCH).
- **C64 client**: any client (driver/BASIC extension/program) that uses the Remote Storage API.

1.2 Typical workflows

- **Bootstrapping**: the client finds the server (discovery or IP scan) and fetches its configuration via **/bootstrap**.
- **File access**: **LS**/**STAT**/**READ_RANGE** for directories and files.
- **Writing**: **WRITE_RANGE**/**APPEND**/**MKDIR**/**RM**/**RMDIR**/**CP**/**MV** (as long as the server/token is not in read-only mode).
- **Disk images**: **.d64**/**.d71**/**.d81** appear as directories. Depending on toggles, reading/copying is possible, optionally also writing, deleting, renaming and copying between images.
- **D81 subdirectories**: 1581 "DIRs" are real partitions (they must consist of contiguous tracks). Optional auto-resize can run a repack when space is low.
- **Admin**: token management, quotas, feature toggles, live log & playground.

2 Installation & operation

This chapter describes the recommended setup on Windows (win32). Linux/macOS works in principle as well, as long as you build the appropriate binaries yourself.

2.1 Requirements

- WiC64 in the C64 (or emulator) with working network configuration.
- The C64 can reach the server via HTTP (typically port 8080).
- For UDP discovery: UDP port 6464 inside the LAN (optional).
- Write permissions in **base_path** and (if enabled) in **trash_dir**.

2.2 Directory structure

Recommended layout of the release bundle (example):

```
wicos64-server.exe
wicos64-tray.exe
w64tool.exe
config/
  config.json
  config.example.json   (optional)
img/
  wicos64.ico
wicos64-data/
  dev1/
  dev2/
logs/
  server.log
```

Note: By default the server expects **config\config.json** next to the executable. For backwards compatibility, an existing **config.json** in the EXE directory is still accepted.

2.3 Start (CLI)

For debugging, starting the server from the CLI is convenient (console output). In the release bundle, a "no console" variant (Windows GUI subsystem) is commonly used as well.

```
.\wicos64-server.exe

# explicitly specify config path
.\wicos64-server.exe -config .\config\config.json

# optional: open browser after start
.\wicos64-server.exe -open-admin
```

Windows tip: In "no-console" builds, Windows logs to **.\logs\server.log** by default (if no console is visible).

2.4 Start (Windows tray app)

The tray app does not install a Windows service. It is a normal Windows application. It can start/stop/restart the server and provides quick links to the Admin UI. The tray icon is preferably loaded from **.\img\wicos64.ico** (fallback: legacy location in the EXE directory).

```
.\wicos64-tray.exe
```

3 Security, tokens & access rules

The server is designed for operation inside a LAN. A token (similar to an API key) is the primary access protection. In addition, there are optional mechanisms such as MAC policy, quotas and a separate bootstrap token.

3.1 Token types

- **API token**: sent with every API request as query parameter (**?token=...**).
- **Token list**: **config.json** can define multiple tokens with their own roots/policies.
- **Bootstrap token**: optional token required for **/bootstrap** (see config: **bootstrap.token**).
- **Legacy**: older configuration fields (**token** / **token_roots**) are still supported.

3.2 Quotas

Quotas can be configured per token and limit e.g. the maximum storage usage (**quota_bytes**). This protects against directories filling up accidentally. In addition, you can set a per-file limit (**max_file_bytes**).

3.3 Read-only mode

If **global_read_only** is enabled (or a token has **read_only=true**), write operations (WRITE_RANGE, APPEND, MKDIR, RM, RMDIR, CP/MV with a write target) are rejected. This is useful for demo servers or public shares.

4 Technical background: C64 <-> server communication

WiC64 provides HTTP functions to the C64. In practice it works like this:

- The client sets a URL (HTTP POST URL) - e.g. **http://<server>:8080/wicos64/api?token=...**.
- The client sends binary payload (HTTP POST data). This payload is the Remote Storage request format (**W64F**).
- The server responds with **Content-Type: application/octet-stream** and a binary response block.

Important: The HTTP status is intentionally always **200**, as long as the server can respond technically. The actual success/error logic is stored in the binary response (W64F status code + optional **err_msg**).

4.1 Request/response - rough structure (v0.2.2)

Every request starts with the magic **W64F**, followed by version/opcode fields and a payload length. All multi-byte numbers are little endian (LE) unless explicitly stated otherwise. For the exact binary layouts, see the API spec; this guide provides a simplified overview:

Request header (client -> server)

Offset	Size	Field	Description
0	4	magic	ASCII "W64F"
4	1	proto_ver	Protocol version
5	1	op	Operation (LS, READ_RANGE, CP, ...)
6	1	flags	Operation-specific flags (bitset)
7	1	reserved	Must be 0
8	2	payload_len	Payload length (bytes, u16 LE)

Response header (server -> client)

Offset	Size	Field	Description
0	4	magic	ASCII "W64F"
4	1	proto_ver	Echo of request protocol version
5	1	op_echo	Echo of op (or 0xFF on invalid request)
6	1	status	Status code (OK / NOT_FOUND / ...)
7	1	reserved	Must be 0
8	2	payload_len	Response payload length (bytes, u16 LE)

From byte 10 onward, an optional error text (**err_msg**) and/or operation-specific payload data may follow.

4.2 Why HTTP 200 on errors?

WiC64 primarily returns a payload to the C64 and can only evaluate HTTP status codes in a limited way. Therefore, WiCOS64 uses a stable HTTP transport (always 200) and stores the real result status in the W64F status code. This simplifies debugging and prevents misleading WiC64 error messages.

5 Remote Storage API - reference (practice-oriented)

This section describes the most important operations. For the exact binary layouts, see the API spec (Remote Storage API v0.2.2). This guide focuses on behavior, flags and examples.

5.1 Basic ops

- **PING**: connectivity test. Expects OK and optional server info.
- **CAPS**: which features/flags does the server support? (important for clients)
- **STATFS**: file system info: free space, limits, etc.

Examples using **w64tool**:

```
.\w64tool.exe -url http://192.168.28.40:8080/wicos64/api?token=DEV1 ping
.\w64tool.exe -url http://192.168.28.40:8080/wicos64/api?token=DEV1 caps
```

5.2 File system ops

- **LS**: lists a directory. The server returns entries (name, type, size, ...).
- **STAT**: metadata for a path (file/directory).
- **READ_RANGE**: reads bytes (offset/len) from a file - ideal for streaming/loading.
- **WRITE_RANGE**: writes bytes (offset/len) into a file (if not read-only).
- **APPEND**: appends bytes to a file.
- **MKDIR** / **RMDIR**: create/delete directories (optional: parents/recursive).
- **RM**: delete a file (optional: trash mode).

Path note

The path separator is **/**. The server normalizes paths (e.g. collapsing double slashes) and prevents path traversal (**..**) outside the **base_path** sandbox.

5.3 COPY/MOVE (CP/MV)

CP copies files or directories. **MV** moves/renames. Both operations support flags such as overwrite/recursive (if enabled on the server).

- **Copy inside the file system**: file or directory (with **-r**).
- **Image -> file system**: extract files from disk images (also via wildcard/bulk).
- **File system -> image**: write files into images (requires **disk_images_write_enabled=true**).
- **Image -> image**: copy between images (e.g. D81 -> D71) is possible; exact rules depend on the target format.

Examples:

```
# Copy a single file (overwrite on)
cp -o /dev1/hello.prg /dev2/hello.prg

# recursive: copy a whole directory (if enable_cp_recursive)
cp -r /games/ /backup/games/

# Copy a file out of a D81 (image is treated like a directory)
cp /images/SONIC.D81/TS /dev1/TS
```

```
# Copy into an image root (disk_images_write_enabled required)
cp /dev1/INTRO /images/WORK.D64

# Copy into a D81 partition (if TESTDIR exists)
cp /ZUHAUS /images/SONIC.D81/TESTDIR
```

5.4 Bulk copy (wildcard) - strict

Bulk copy is an extension of **CP**: if the source path contains a wildcard (***** or **?**), it is interpreted as a pattern. The server expands the pattern server-side and copies all matches.

Strict rule: wildcards are only allowed in the last path segment (e.g. **/usr/*** or **/disk.d81/PRG?.***). A wildcard in any earlier segment is rejected with **BAD_REQUEST**.

```
# Bulk copy: all files from /usr to /bin
cp -o /usr/* /bin/

# Bulk extract: out of a disk image
cp -o /images/SONIC.D81/* /dev1/sonic/
```

Recommended flags: **-o** (overwrite) and **-r** (recursive). For bulk copy without **-r**, directories in the source match set are ignored.

6 Disk images as directories (D64/D71/D81)

WiCOS64 can present disk images transparently as directories. This means: a path like **/images/DISK.D64** is treated like a folder. The client uses normal operations such as LS/STAT/READ_RANGE/CP/MV.

6.1 Supported formats

- **.d64** (1541)
- **.d71** (1571 - two sides)
- **.d81** (1581 - including subdirectories/partitions)

6.2 Read vs. write (feature toggles)

Disk images are only active if **disk_images_enabled** is enabled (globally and for the token). If enabled, matching files appear as directories.

- **Read** (default): LS/STAT/READ_RANGE/CP out of the image.
- **Write** (optional): if **disk_images_write_enabled=true**, then writing into images, delete, rename/MV and copy into images are allowed.
- **Auto-resize** (optional, mainly D81): if **disk_images_auto_resize_enabled=true**, the server may run a repack to make space in D81 partitions when capacity is low.

Warning: write/auto-resize can modify images and can be destructive. Use this inside a LAN, with backups, and consider enabling write features only for trusted clients (per token).

6.3 Feature toggles: global & per token

Disk image features can be enabled/disabled globally and per token. Per-token write/auto-resize flags are optional overrides. If a token flag is not set, the global value is used.

```
{
  "disk_images_enabled": true,
  "disk_images_write_enabled": false,
  "disk_images_auto_resize_enabled": false,
  "tokens": [
    { "name": "dev1", "token": "DEV1", "root": "dev1", "disk_images_enabled": true },
    { "name": "public", "token": "PUBLIC", "root": "pub", "disk_images_enabled": false }
  ]
}
```

6.4 Special case: MKDIR/RMDIR on image files

If disk images as directories are enabled (**disk_images_enabled=true**), the following special cases apply:

- **mkdir "test.d64"** creates an empty D64 image (analog for .d71/.d81).
- **rmdir "test.d64"** deletes the image file (optionally via trash mode instead of hard delete).
- If **disk_images_enabled=false**, ***.d64/*.d71/*.d81** are treated as normal files - MKDIR/RMDIR will not apply to them.
- Creating/deleting images is only possible on the normal host file system (not "inside" a mounted image).

```
# create empty image
mkdir /images/test.d64

# delete image
rmdir /images/test.d64
```

6.5 1581 subdirectories (.d81) and auto-resize

For **.d81**, subdirectories are shown as real directories - technically these are 1581 partitions. **Important:** these partitions must consist of contiguous tracks (contiguous allocation).

- Read/list also in nested partitions (e.g. **/TOOLS.D81/UTILS/FASTCOPY**).
- Write into partitions (if **disk_images_write_enabled=true**).
- MKDIR/RMDIR inside a D81 creates/deletes partitions (DIRs).
- Recursive import of a host directory as a partition via **CP -r** (repack).
- Optional auto-resize: if space is low, the server can repack the image and grow partitions, as long as the tree has enough free space up to the root.

Practical examples:

```
# switch into a D81
cd /images/TOOLS.D81
ls

# switch into a partition
cd /images/TOOLS.D81/UTILS
ls

# create a new partition (write toggle required)
mkdir /images/TOOLS.D81/UTILS/NEWDIR

# copy a file into the partition (write toggle required)
cp /dev1/NEWTOOL /images/TOOLS.D81/UTILS

# recursive: import a host directory as a new partition (repack)
cp -r -o /dev1/MYDIR/ /images/TOOLS.D81/UTILS/MYDIR
```

Performance: For READ_RANGE, the server reads only the required sectors/blocks from the image. This is much more efficient than extracting the whole image and also works for streaming loaders.

Rename protection (edge case)

To avoid accidentally turning an image into a normal file (or vice versa), the default behavior is: a **.d64**/**.d71**/**.d81** cannot be renamed via **MV** to a name without that extension, and a normal file cannot be renamed into **.d64**/**.d71**/**.d81**. Optionally, this can be enabled per token (**disk_images_allow_rename_convert=true**).

7 Find the server automatically (discovery & bootstrap)

To avoid hard-coding an IP address on the C64 client, there are two mechanisms: discovery (UDP) and bootstrap (HTTP). In some networks, UDP is blocked - then an IP scan can be used as a fallback.

7.1 HTTP bootstrap

Bootstrap is a normal HTTP endpoint (default: **/wicos64/bootstrap**). The client passes a **cfg** (profile name) and optionally a MAC address. The server returns a configuration that the client then uses for API access.

```
# Example URL (GET)
http://192.168.28.40:8080/wicos64/bootstrap?cfg=RETROBOOT&mac=001122334455
```

MAC format: the server normalizes MAC addresses server-side. If a client sends 24 hex characters (WiC64 firmware quirk), it is reduced to 12 characters.

Optionally, the server can append additional **KEY=VALUE** lines per MAC address to the bootstrap response (e.g. client-specific defaults). Old clients ignore unknown keys - backwards compatibility is preserved.

7.2 UDP discovery (WDP1)

Discovery uses UDP on port **6464**. The client sends a DISCOVER packet (magic **"WDP1"**), the server responds unicast with an OFFER. The OFFER contains e.g. server IP, HTTP port and feature flags.

Security: discovery should be used only inside a LAN. If **lan_only=true**, the server ignores packets from non-private IPs. A rate limit reduces spam.

7.3 Fallback: IP scan

If UDP is blocked and no DNS name is available, the client can try to find the server via an IP scan. Typically **/bootstrap** or **/api** is tested over a port range.

Note: on the C64 this scan can take noticeably longer depending on network size and timeouts. A fixed IP via DHCP reservation is often the simplest solution.

8 Admin UI - usage & features

The Admin UI is an optional web interface. It is active only if **enable_admin_ui** is set in **config.json**. By default it listens on the same port as the API (e.g. :8080) and is reachable at **/admin**.

8.1 Enable & access

```
{
  "enable_admin_ui": true,
  "admin_user": "admin",
  "admin_password": "change-me",
  "admin_allow_remote": false
}
```

Security: set your own password and enable **admin_allow_remote** only if you really need to access the Admin UI from another machine in the LAN. For maximum safety: keep **admin_allow_remote=false** and use the Admin UI only locally on the server.

8.2 Dashboard / status

On the dashboard you can see e.g. version/build, listen address, API endpoint, base path, free space and which features are enabled.

8.3 Config - form view & JSON view

- **Form view**: convenient for common settings (toggles, ports, paths, limits).
- **JSON view**: full control; ideal for copy/paste or larger changes.

Important: When saving, the JSON configuration is written as a whole. Fields that are not displayed in the UI must not get lost - the UI should round-trip the full JSON.

New: in the config view you can optionally manage the bootstrap extension per MAC ("MAC->Extra").

8.4 Tokens & quotas

In the token manager you can create, edit and delete tokens. Typical per-token settings:

- Name + token string
- Root folder (relative to **base_path**)
- Quota/limits (**quota_bytes**, **max_file_bytes**)
- Feature toggles (e.g. **disk_images_enabled**, **disk_images_write_enabled**, **disk_images_auto_resize_enabled**)
- Read-only (optional)
- Optional: **disk_images_allow_rename_convert** (only when explicitly desired)

8.5 Live log

The live log shows recent requests including time, IP, op name, status, duration and bytes. Filters allow you to show only LS or only CP, for example.

8.6 Ops Playground (mini CLI)

The Ops Playground is a small command line inside the Admin UI. It lets you test operations without changing anything on the C64. Internally it uses the same dispatch as real client requests, including token context (permissions/feature toggles/limits).

- Select a token context (so path mapping and permissions match).
- Command history (history dropdown).
- Output including status/err_msg and, when useful, structured payload.
- "Send selected log entry to Playground": take a selected live log entry and copy it into the Playground.

Examples (Playground):

```
ping
caps
ls /
ls /images
stat /dev1/hello.prg
read_range /dev1/hello.prg 0 256
cp -o /images/WORK.D64/INTRO /dev1/INTRO
```

8.7 Jobs: trash / tmp cleanup

Optionally, the server can move deleted files into a trash directory. A cleanup job can periodically empty the trash. Similarly, there is a tmp cleanup for temporary files under **/.tmp** (or the internal tmp directory).

9 Configuration (config.json) - reference

The configuration is a JSON file. The default path is **config/config.json** next to the server EXE (fallback: legacy **config.json** in the EXE directory). You can always specify the path explicitly via **-config**.

9.1 Minimal example

```
{
  "listen": ":8080",
  "endpoint": "/wicos64/api",
  "base_path": "./wicos64-data",
  "tokens": [
    { "token": "WICOS64_DEV1", "name": "C64 #1", "root": "dev1" }
  ],
  "enable_admin_ui": true,
  "admin_user": "admin",
  "admin_password": "change-me"
}
```

9.2 Basic settings

Field	Description
listen	Listen address (Host:Port). Example ':8080' or '127.0.0.1:8080'.
endpoint	API path. Default: /wicos64/api.
base_path	Base directory for token roots (relative roots are resolved below it).
server_name	String returned via CAPS (client info).

9.3 Tokens (new, recommended)

The new token structure is an array **tokens**. Per entry:

Field	Description
token	Token string sent by the client.
name	Display name (optional, Admin UI only).
root	Root directory (relative to base_path or absolute).
enabled	Optional; if false, the token is disabled.
read_only	If true, write operations are disabled for this token.
quota_bytes	Optional quota limit.
max_file_bytes	Optional per-file limit.
disk_images_enabled	Optional override for disk_images_enabled.
disk_images_write_enabled	Optional override for disk_images_write_enabled.
disk_images_auto_resize_enabled	Optional override for disk_images_auto_resize_enabled (D81 auto-resize).
disk_images_allow_rename_convert	Optional: allow renaming between 'image' and 'normal file' (default: false).

9.4 Legacy token configuration (still supported)

For older setups, the fields **token** (single token) and **token_roots** (map token->root) are still supported. If **tokens** is not empty, that list takes precedence.

9.5 Global policies

Field	Description
global_read_only	Switches the entire server into read-only mode.
global_quota_bytes	Optional global quota (in addition to token-specific quota_bytes).
global_max_file_bytes	Optional global per-file limit.

9.6 Limits (DoS protection)

These limits are communicated to the client via CAPS and are enforced server-side:

Field	Default	Meaning
max_payload	16384	Max payload (HTTP body) in bytes.
max_chunk	4096	Max chunk size for READ/WRITE/APPEND.
max_path	255	Max path length (string).
max_name	64	Max segment length (file/dir name).
max_entries	50	Max entries per LS (paging for large directories).

9.7 Feature toggles

Field	Description
enable_mkdir_parents	Enables MKDIR -p (mkdir -p).
enable_rmdir_recursive	Enables RMDIR -r (rm -r).
enable_cp_recursive	Enables CP -r (copy directories).
enable_overwrite	Enables overwrite flags for WRITE_RANGE/CP/MV.
enable_errmsg	If enabled, the server also returns err_msg (text) in the response.

9.8 Disk images

Field	Description
disk_images_enabled	Mounts .d64/.d71/.d81 as directories.
disk_images_write_enabled	Allows write/delete/rename inside mounted images (potentially destructive).
disk_images_auto_resize_enabled	Allows auto-resize/repack for D81 partitions when space is low.

9.9 Bootstrap (optional)

Bootstrap returns API_URL + token for a client - optionally via MAC mapping.

Field	Description
bootstrap.enabled	Enables /bootstrap.
bootstrap.token	Shared secret (config token) required for /bootstrap.
bootstrap.lan_only	Only LAN IPs may use /bootstrap.
bootstrap.unknown_mac_policy	What to do with unknown MACs: 'deny' or 'legacy'.
bootstrap.mac_tokens	Map MAC (AABBCCDDEEFF) -> token.
bootstrap.mac_extra	Optional: map MAC -> extra KEY=VALUE lines appended to the response.

9.10 Discovery (optional)

Field	Description
discovery.enabled	Enables UDP discovery.
discovery.udp_port	UDP port (default 6464).
discovery.lan_only	Only LAN IPs may use discovery.
discovery.rate_limit_per_sec	Rate limit per source IP.

9.11 Trash & cleanup (optional)

Field	Description
trash_enabled	If true, RM/RMDIR and overwrite operations are moved to trash_dir instead
trash_dir	Path relative to the token root (default: .TRASH).
trash_cleanup_enabled	If true, a job removes old trash entries.
tmp_cleanup_enabled	Removes temporary files under /.tmp (or the internal tmp) periodically.

9.12 Compat toggles

Field	Description
compat.fallback_prg_extension	READ/STAT/HASH optionally try '.PRG' if a file is not found without extensi
compat.wildcard_load	READ/STAT/HASH allow '*' and '?' in the last segment (Commodore-style).

Note: The wildcard compat toggle applies only to read-like operations. Write operations never accept wildcards.

10 w64tool - CLI tool

w64tool is a small CLI tool for smoke-testing the API from a PC. It is intentionally simple and works well to verify token/URL/server reachability.

10.1 Usage

```
# Help / commands
.\w64tool.exe -h

# Set URL including token
.\w64tool.exe -url "http://127.0.0.1:8080/wicos64/api?token=WICOS64_DEV1" caps
```

Currently supported commands (excerpt): **caps**, **ping**, **ls**, **append**, **hash**, **search**.

10.2 Typical tests

```
# Connection / token test
.\w64tool.exe -url "http://127.0.0.1:8080/wicos64/api?token=WICOS64_DEV1" ping

# Check features / limits
.\w64tool.exe -url "http://127.0.0.1:8080/wicos64/api?token=WICOS64_DEV1" caps

# List root
.\w64tool.exe -url "http://127.0.0.1:8080/wicos64/api?token=WICOS64_DEV1" ls /

# Test disk image mounts (if disk_images_enabled=true)
.\w64tool.exe -url "http://127.0.0.1:8080/wicos64/api?token=WICOS64_DEV1" ls /images
.\w64tool.exe -url "http://127.0.0.1:8080/wicos64/api?token=WICOS64_DEV1" ls /images/TOOLS.D81
```

For more extensive operations (CP/MV/WRITE_RANGE/READ_RANGE), the Ops Playground in the Admin UI is currently the more practical tool.

11 Windows tray app

The tray app is a helper program for Windows. It is aimed at users who want to start/stop the server conveniently without keeping a console open.

11.1 Features

- Start/stop/restart server (depending on status and permissions).
- Open Admin UI: opens **/admin** in the browser (if Admin UI is enabled).
- Reload config (soft): reloads **config.json** without a full process restart.
- Open config: opens **config.json** in the editor.
- View server.log: opens **logs/server.log** (especially important for "no-console" builds).

11.2 Config and icon path

Default config: the tray app uses **config\config.json** next to the EXE by default. Alternatively you can pass **-config**. For backwards compatibility, an existing **config.json** in the EXE directory is accepted as well.

Icon: preferred **img\wicos64.ico** (fallback: **wicos64.ico** in the EXE directory).

11.3 Typical problems

- **Server does not start:** check logs (**logs/server.log**). Often caused by port conflicts or an invalid config.json.
- **Admin UI does not open: enable_admin_ui** must be true; **admin_allow_remote** influences URL selection.
- **Tray shows "no base URL":** check **listen** in config.json (e.g. ':8080').
- **Firewall:** Windows Firewall may block incoming connections (HTTP 8080, optionally UDP 6464).

12 Troubleshooting & FAQ

This chapter summarizes the most common issues and their solutions.

12.1 "payload_len mismatch" / BAD_REQUEST

If WiC64 and the server interpret payload lengths differently, the live log often shows **payload_len mismatch**. Possible causes:

- Firmware/emulator sends an extra byte.
- Multipart form-data is not correctly "unwrapped" (server-side).
- Client miscalculated the length (header vs. body).

In the live log you can see **request_bytes**/**resp_bytes** and **err_msg** (if **enable_errmsg** is enabled).

12.2 "Failed to open connection"

This is a WiC64 HTTP issue (DNS, gateway, firewall, wrong IP/port). Check:

- Server is running and listening on the correct IP/port.
- URL in the client is correct (no typo like 192.268.x.x).
- Windows Firewall allows incoming connections.

12.3 Disk image listing is empty

- **disk_images_enabled** is disabled globally or for the token.
- The file extension is not recognized as an image (only .d64/.d71/.d81).
- Image is corrupted or not a real D64/D71/D81.

12.4 Writing into disk images is denied

If writing/copying into images is rejected with ACCESS_DENIED/NOT_SUPPORTED:

- **disk_images_write_enabled** is still false globally or per token.
- Token is read-only or **global_read_only** is enabled.
- **enable_overwrite** is off, but the client tries overwrite/truncate.
- You are trying to write/create an image inside another mounted image (not allowed).

12.5 D81 partition full / auto-resize

With **.d81**, partitions (DIRs) can run out of space. If **disk_images_auto_resize_enabled** is enabled, the server tries a repack when space is low. If it is disabled, you typically get TOO_LARGE or a disk-image-specific error.

- Enable auto-resize (globally or per token) if you want the feature.
- Alternatively delete/move files or reorganize the image externally.
- Note: repack can be I/O intensive (the image is rewritten).

12.6 Bulk copy reports "too large"

With bulk copy, the response payload can become large if many matches are reported. The server limits payloads via **max_payload**. Solution: increase **max_payload** or run bulk operations in smaller batches.

Appendix A - Status codes (excerpt)

The W64F status code is stored in the response header (byte 6). In the live log it is also shown as text. The complete list is in the API spec; here is a practical excerpt:

Name	Code	Meaning
OK	0x00	Operation successful.
NOT_FOUND	0x01	Path does not exist.
NOT_A_DIR	0x02	Path is not a directory.
IS_A_DIR	0x03	Path is a directory (operation expects a file).
ALREADY_EXISTS	0x04	Target already exists (e.g. MKDIR on a file).
DIR_NOT_EMPTY	0x05	Directory not empty (RMDIR without recursive).
ACCESS_DENIED	0x06	Token invalid or permission denied (read-only, disk_images_write_disabled, quota
INVALID_PATH	0x07	Invalid path or outside root.
RANGE_INVALID	0x08	Invalid offset/length for READ_RANGE/WRITE_RANGE.
TOO_LARGE	0x09	Payload/file/quota exceeded.
NOT_SUPPORTED	0x0A	Operation or flag not enabled (feature toggle).
BUSY	0x0B	Server is busy (e.g. parallel writes).
BAD_REQUEST	0x0C	Invalid payload/parameters or wildcard rules violated.
INTERNAL	0x0D	Internal server error (see err_msg).

Appendix B - Example sequences

B.1 Typical client start (bootstrap + PING)

1) Client -> GET /wicos64/bootstrap?cfg=RETROBOOT&mac;=001122334455
2) Server -> returns URL: http://192.168.28.40:8080/wicos64/api?token=DEV1
3) Client sets POST URL to this URL
4) Client sends PING (W64F request)
5) Server responds OK (W64F response)

B.2 Mount / access disk image (read)

1) ls /images
2) ls /images/TOOLS.D81
3) cd /images/TOOLS.D81/UTILS
4) ls
5) cp /images/TOOLS.D81/UTILS/FASTCOPY /dev1/FASTCOPY

B.3 Writing into disk images (optional)

Prerequisite: **disk_images_write_enabled=true** (globally or per token).

1) mkdir /images/test.d64 # create empty D64
2) cp /dev1/INTRO /images/test.d64 # write file into image root
3) mv -o /images/test.d64/INTRO /images/test.d64/INTRO2
4) rm /images/test.d64/INTRO2
5) rmdir /images/test.d64 # delete image

Appendix C - Wildcard examples

Wildcard syntax is similar to DOS/Unix: ***** = any number of characters, **?** = exactly one character. For bulk copy, strict mode applies: wildcards only in the last segment.

```
# OK
cp -o /usr/* /bin/
cp -o /images/WORK.D64/DEMO* /dev1/

# BAD_REQUEST (wildcard in a middle segment)
cp /u*/tools/* /bin/
```