WiCOS64

REMOTE STORAGE

Admin Guide & Technische Dokumentation

Version v1.0.1.33 - Stand 2026-02-04

- Server + Admin UI + W64Tool
- Disk-Image Support (D64/D71/D81) - read/write (optional)
- D81: 1581-Partitionen ("DIRs") + optional Auto-Resize (Repack)
- MKDIR/RMDIR: Images anlegen/löschen + D81-Partitionen anlegen/löschen
- Bootstrap & Discovery (HTTP/UDP) + optional MAC-Extras
- Token-Management, Quotas & Trash-Mode
- Ops Playground (CLI) inkl. History

Dokument-Infos

Dieses Dokument beschreibt Installation, Betrieb, Administration und die technischen Hintergründe des WiCOS64 Remote Storage Servers inklusive Admin-UI, Windows-Tray-App und dem CLI-Tool **w64tool** (Stand: **v1.0.1.33**).

Hinweis

WiCOS64 ist ein Remote-Storage-Server für den Commodore 64 (C64) in Kombination mit einem WiC64-Modul. Dieses Dokument ist als Nachschlagewerk gedacht: Du kannst es von vorne lesen - oder gezielt einzelne Kapitel für konkrete Aufgaben nutzen.

Zielgruppe

- Anwender, die den Server im Heimnetz betreiben
- Entwickler, die eigene C64-Programme/Tools gegen die Remote-Storage-API schreiben
- Fortgeschrittene Nutzer, die Token, Quotas, Disk-Images oder Discovery/Bootstrap konfigurieren wollen

Inhaltsverzeichnis

1 Überblick

Der WiCOS64 Remote Storage Server stellt dem C64 - über ein WiC64-Modul - ein virtuelles Dateisystem im Netzwerk bereit. Der Client spricht dabei nicht direkt mit SMB/NFS, sondern nutzt eine kompakte binäre Remote-Storage-API, die über HTTP POST transportiert wird.

Optional können Disk-Images (.d64/.d71/.d81) transparent als Verzeichnisse gemountet werden. Je nach Feature-Toggles sind Images nicht nur lesbar, sondern können auch geschrieben, umbenannt, kopiert und gelöscht werden.

1.1 Komponenten

- **Server** (wicos64-server.exe): stellt die Remote-Storage-API bereit, verwaltet Tokens/Quotas, optional Trash-Mode und bedient Disk-Images als virtuelle Verzeichnisse.
- **Admin UI**: Web-Oberfläche für Status, Konfiguration, Tokens & Quotas, Live-Log, Diagnose und Ops Playground.
- **Windows Tray App** (wicos64-tray.exe): Komfort-Start/Stop/Restart, schneller Zugriff auf Admin UI/Logs/Config (kein Windows-Service).
- **w64tool** (CLI): Smoke-Test/Debug-Tool (CAPS/PING/LS/APPEND/HASH/SEARCH).
- **C64-Client**: beliebiger Client (Treiber/BASIC-Erweiterung/Programm), der die Remote-Storage-API nutzt.

1.2 Typische Workflows

- **Bootstrapping**: Client findet den Server (Discovery oder IP-Scan) und holt sich die Konfiguration über /bootstrap.
- **Dateizugriff**: LS/STAT/READ_RANGE für Verzeichnisse und Dateien.
- **Schreiben**: WRITE_RANGE/APPEND/MKDIR/RM/RMDIR/CP/MV (sofern nicht im Read-Only Mode).
- **Disk-Images**: .d64/.d71/.d81 erscheinen als Verzeichnisse. Je nach Toggle sind Lesen/Kopieren möglich, optional auch Schreiben, Löschen, Rename und Copy zwischen Images.
- **D81-Unterverzeichnisse**: 1581-'DIRs' sind echte Partitionen (müssen zusammenhängende Tracks haben). Optionales Auto-Resize kann bei Platzmangel ein Repack durchführen.
- **Admin**: Token-Management, Quotas, Feature-Toggles, Live-Log & Playground.

2 Installation & Betrieb

Dieses Kapitel beschreibt den empfohlenen Betrieb unter Windows (win32). Linux/macOS funktionieren prinzipiell analog, sofern Du die passenden Binaries selbst baust.

2.1 Voraussetzungen

- WiC64 im C64 (oder Emulator) mit funktionierender Netzwerk-Konfiguration
- Zugriff vom C64 auf den Server per HTTP (typisch Port 8080)
- Für Discovery per UDP: Port 6464 im LAN (optional)
- Schreibrechte im base_path und ggf. in trash_dir

2.2 Verzeichnisstruktur

Empfohlenes Layout des Release-Bundles (Beispiel):

```
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
```

Hinweis: Der Server erwartet standardmäßig **config\config.json** neben der EXE. Zur Abwärtskompatibilität wird eine vorhandene **config.json** im EXE-Ordner weiterhin akzeptiert.

2.3 Starten (CLI)

Für Debugging ist der CLI-Start praktisch (mit Console-Ausgabe). Im Release-Bundle ist zusätzlich eine 'no console' Variante (Windows GUI Subsystem) üblich.

```
.\wicos64-server.exe

# explizit
.\wicos64-server.exe -config .\config\config.json

# optional: Browser nach Start öffnen
.\wicos64-server.exe -open-admin
```

Windows-Tipp: In 'no-console' Builds wird unter Windows standardmäßig nach **.\logs\server.log** geloggt (falls keine Konsole sichtbar ist).

2.4 Starten (Windows Tray App)

Die Tray-App startet keinen Service, sondern ist ein normales Windows-Programm. Sie kann den Server starten/stoppen/restarten und verlinkt die Admin-UI.

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

Das Tray-Icon wird bevorzugt aus **.\img\wicos64.ico** geladen (Fallback: alte Position im EXE-Ordner).

3 Sicherheit, Tokens & Zugriffsregeln

Der Server ist für den Betrieb im LAN gedacht. Ein Token (ähnlich einem API-Key) ist der primäre Zugriffsschutz. Zusätzlich gibt es optionale Mechanismen wie MAC-Policy, Quotas und einen separaten Bootstrap-Token.

3.1 Token-Typen

- **API Token**: wird bei jedem API-Request als Query-Parameter gesendet (?token=...).
- **Token-Liste**: in config.json können mehrere Tokens mit eigenen Roots/Policies gepflegt werden.
- **Bootstrap Token**: optionaler Token, der für /bootstrap benötigt wird (siehe config: bootstrap.token).
- **Legacy**: ältere Konfigurationsfelder (token / token_roots) werden weiterhin unterstützt.

3.2 Quotas

Quotas sind pro Token konfigurierbar und begrenzen z.B. die maximale Speicherbelegung (quota_bytes). Das schützt vor 'aus Versehen' volllaufenden Verzeichnissen. Zusätzlich kann ein per-File-Limit (max_file_bytes) gesetzt werden.

3.3 Read-Only Mode

Wenn global_read_only aktiv ist (oder ein Token read_only=true hat), werden schreibende Operationen (WRITE_RANGE, APPEND, MKDIR, RM, RMDIR, CP/MV mit schreibendem Ziel) abgewiesen. Das ist nützlich für Demo-Server oder öffentliche Freigaben.

4 Technischer Hintergrund: Kommunikation C64 ↔ Server

Das WiC64 stellt dem C64 HTTP-Funktionen bereit. In der Praxis läuft es so ab:

- Der Client setzt eine URL (HTTP POST URL) - z.B. http://:8080/wicos64/api?token=...
- Der Client sendet binäre Nutzdaten (HTTP POST data). Diese Nutzdaten sind das Remote-Storage-Request-Format (W64F).
- Der Server antwortet mit Content-Type: application/octet-stream und einem binären Response-Block.

Wichtig

Der HTTP-Status ist absichtlich immer 200, solange der Server technisch antworten kann. Die eigentliche Fehler-/Statuslogik steckt im binären Response (W64F-Statuscode + optional err_msg).

4.1 Request/Response - grobe Struktur (v0.2.2)

Jeder Request beginnt mit dem Magic W64F, gefolgt von Versions-/Opcode-Feldern und einer Payload-Länge. Alle Multi-Byte-Zahlen sind little endian (LE), sofern nicht ausdrücklich anders erwähnt. Die exakten Details stehen in der API-Spec; hier ist eine vereinfachte Übersicht:

Request Header (Client → Server)

Offset	Größe	Feld	Beschreibung
0	4	magic	ASCII 'W64F'
4	1	proto_ver	Protokollversion
5	1	op	Operation (LS, READ_RANGE, CP, ...)
6	1	flags	Op-spezifische Flags (bitset)
7	1	reserved	muss 0 sein
8	2	payload_len	Länge der Payload (Bytes, u16 LE)

Response Header (Server → Client)

Offset	Größe	Feld	Beschreibung
0	4	magic	ASCII 'W64F'
4	1	proto_ver	Protokollversion
5	1	op_echo	Echo des Ops (oder 0xFF bei Fehlern)
6	1	status	Statuscode (OK/NOT_FOUND/...)
7	1	reserved	muss 0 sein

8	2	`payload_len`	Länge der Response-Payload (Bytes, u16 LE)

Ab Byte 10 folgen optional ein Fehlertext (err_msg) und/oder op-spezifische Payload-Daten.

4.2 Warum HTTP 200 bei Fehlern?

Das WiC64 liefert dem C64 primär einen Payload zurück und kann HTTP-Statuscodes nur eingeschränkt auswerten. Deshalb nutzt WiCOS64 einen stabilen HTTP-Transport (immer 200) und legt den echten Ergebnisstatus in den W64F-Statuscode. Das erleichtert Debugging und verhindert 'falsche' WiC64-Fehlerbilder.

5 Remote-Storage API - Referenz (praxisorientiert)

Hier sind die wichtigsten Operationen beschrieben. Für die exakten Binärlayouts siehe zusätzlich die API-Spec (Remote Storage API v0.2.2). In diesem Guide liegt der Fokus auf Verhalten, Flags und Beispielen.

5.1 Grundops

- **PING**: Verbindungstest. Erwartet OK und optional Server-Infos.
- **CAPS**: Welche Features/Flags unterstützt der Server? (wichtig für Clients)
- **STATFS**: Infos zum Dateisystem: freier Platz, Limits, etc.

Beispiele mit 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 Dateisystemops

- **LS**: Listet ein Verzeichnis. Server liefert Einträge (Name, Type, Size, ...).
- **STAT**: Metadaten zu einem Pfad (Datei/Verzeichnis).
- **READ_RANGE**: Liest Bytes (offset/len) aus einer Datei - ideal für Stream/Loader.
- **WRITE_RANGE**: Schreibt Bytes (offset/len) in eine Datei (wenn nicht read-only).
- **APPEND**: Hängt Bytes an eine Datei an.
- **MKDIR / RMDIR**: Verzeichnis anlegen / löschen (optional: parents/recursive).
- **RM**: Datei löschen (optional: Trash-Mode).

Hinweis zu Pfaden

Pfadseparator ist '/'. Der Server normalisiert Pfade (z.B. doppelte Slashes) und verhindert Path Traversal ('..') außerhalb des base_path.

5.3 COPY/MOVE (CP/MV)

CP kopiert Dateien oder Verzeichnisse. MV verschiebt/renamed. Beide Ops unterstützen Flags wie overwrite/recursive (wenn serverseitig aktiviert).

- Copy im Filesystem: Datei oder Directory (mit -r).
- Image → Filesystem: Dateien aus Disk-Images extrahieren (auch per Wildcard/Bulk).
- Filesystem → Image: Dateien in Images schreiben (wenn disk_images_write_enabled=true).

- Image → Image: Copy zwischen Images (z.B. D81 → D71) ist möglich; die konkreten Regeln hängen vom Zielformat ab.

Beispiele:

```
# Einzelne Datei kopieren (overwrite an)
cp -o /dev1/hello.prg /dev2/hello.prg

# rekursiv: komplettes Verzeichnis (wenn enable_cp_recursive)
cp -r /games/ /backup/games/

# Datei aus D81 (Image wird als Verzeichnis behandelt)
cp /images/SONIC.D81/TS /dev1/TS

# Datei ins Image-Root schreiben (disk_images_write_enabled erforderlich)
cp /dev1/INTRO /images/WORK.D64

# Datei in eine D81-Partition schreiben (wenn TESTDIR existiert)
cp /ZUHAUS /images/SONIC.D81/TESTDIR
```

5.4 Bulk Copy (Wildcard) - strict

Bulk Copy ist eine Erweiterung von CP: Wenn der Source-Pfad ein Wildcard (* oder ?) enthält, wird er als Muster interpretiert. Der Server expandiert das Muster serverseitig und kopiert alle Treffer.

Strict-Regel: Wildcards sind nur im letzten Pfadsegment erlaubt (z.B. /usr/* oder /disk.d81/PRG?.*). Ein Wildcard in einem früheren Segment wird mit BAD_REQUEST abgelehnt.

```
# Bulk Copy: alle Dateien aus /usr nach /bin
cp -o /usr/* /bin/

# Bulk Extract: aus Disk-Image heraus
cp -o /images/SONIC.D81/* /dev1/sonic/
```

Empfohlene Flags: -o (overwrite) und -r (rekursiv). Bei Bulk Copy ohne -r werden Verzeichnisse in der Source-Menge ignoriert.

6 Disk-Images als Verzeichnisse (D64/D71/D81)

WiCOS64 kann Disk-Images transparent als Verzeichnisse darstellen. Das heißt: Ein Pfad wie /images/DISK.D64 wird beim Zugriff wie ein Ordner behandelt. Der Client nutzt normale LS/STAT/READ_RANGE/CP/MV.

6.1 Unterstützte Formate

- .d64 (1541)
- .d71 (1571 - zwei Seiten)
- .d81 (1581 - inklusive Unterverzeichnissen/Partitionen)

6.2 Lesen vs. Schreiben (Feature Toggles)

Disk-Images sind nur aktiv, wenn disk_images_enabled (global und für den Token) eingeschaltet ist. Dann erscheinen passende Dateien als Verzeichnisse.

- **Read (Standard)**: LS/STAT/READ_RANGE/CP aus dem Image heraus.
- **Write (optional)**: Wenn disk_images_write_enabled=true, sind zusätzlich Schreiben in Images, Delete, Rename/MV sowie Copy in Images erlaubt.
- **Auto-Resize (optional, primär D81)**: Wenn disk_images_auto_resize_enabled=true, kann der Server bei Platzmangel in D81-Partitionen ein Repack durchführen, um Platz zu schaffen.

> **Achtung**
>
> Write/Auto-Resize können Images verändern und sind potentiell destruktiv. Nutze das im LAN, mit Backups, und ggf. per Token nur für vertrauenswürdige Clients.

6.3 Feature Toggle: global & pro Token

Disk-Image Features können global und pro Token ein-/ausgeschaltet werden. Pro Token sind die Write/Auto-Resize Flags optional (Override). Nicht gesetzte Token-Flags übernehmen den globalen Wert.

```
{
  "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 Besonderheit: MKDIR/RMDIR auf Image-Dateien

Wenn Disk-Images als Verzeichnisse aktiviert sind (disk_images_enabled=true), gelten folgende Sonderfälle:

- **mkdir** "test.d64" erstellt ein leeres D64-Image (analog für .d71/.d81).
- **rmdir** "test.d64" löscht die Image-Datei (optional: Trash-Mode statt hart löschen).
- Wenn disk_images_enabled=false, werden *.d64/*.d71/*.d81 als normale Dateien behandelt - MKDIR/RMDIR greifen dann nicht.
- Das Erstellen/Löschen von Images ist nur auf dem normalen Filesystem möglich (nicht 'innerhalb' eines gemounteten Images).

```
# leeres Image erstellen
mkdir /images/test.d64

# Image löschen
rmdir /images/test.d64
```

6.5 1581-Unterverzeichnisse (.d81) und Auto-Resize

Bei .d81 werden Unterverzeichnisse als echte Directories dargestellt - technisch sind das 1581-Partitionen. Wichtig: Diese Partitionen müssen aus zusammenhängenden Tracks bestehen (contiguous allocation).

- Lesen/Listen auch in verschachtelten Partitionen (z.B. /TOOLS.D81/UTILS/FASTCOPY).
- Schreiben in Partitionen (wenn disk_images_write_enabled=true).
- MKDIR/RMDIR innerhalb einer D81 legt/leöscht Partitionen (DIRs).
- Rekursives Importieren eines Host-Verzeichnisses als Partition via CP -r (Repack).
- Optionales Auto-Resize: Bei Platzmangel kann der Server das Image neu packen (Repack) und Partitionen vergrößern, sofern der Baum bis zum Root genug freien Platz hat.

Praxisbeispiele:

```
# in D81 wechseln
cd /images/TOOLS.D81
ls

# in Partition wechseln
cd /images/TOOLS.D81/UTILS
ls

# neue Partition anlegen (Write Toggle erforderlich)
mkdir /images/TOOLS.D81/UTILS/NEWDIR

# Datei in Partition hinein kopieren (Write Toggle erforderlich)
cp /dev1/NEWTOOL /images/TOOLS.D81/UTILS

# rekursiv: Host-Verzeichnis als neue Partition importieren (Repack)
cp -r -o /dev1/MYDIR/ /images/TOOLS.D81/UTILS/MYDIR
```

Performance: Für READ_RANGE liest der Server nur die benötigten Sektoren/Blocks aus dem Image. Das ist deutlich effizienter als komplette Image-Extraktion und funktioniert auch für Streaming-Loader.

Rename-Schutz (Edge Case)

Damit ein Image nicht aus Versehen 'zum normalen File' wird (oder umgekehrt), gilt standardmäßig: Ein .d64/.d71/.d81 kann nicht per MV in einen Namen ohne diese Endung umbenannt werden, und eine normale Datei kann nicht in .d64/.d71/.d81 umbenannt werden. Optional kann das pro Token freigeschaltet werden (disk_images_allow_rename_convert=true).

7 Server automatisch finden (Discovery & Bootstrap)

Damit ein C64-Client nicht hart eine IP eintragen muss, gibt es zwei Mechanismen: Discovery (UDP) und Bootstrap (HTTP). In manchen Netzwerken ist UDP gesperrt - dann bleibt als Fallback z.B. ein IP-Scan.

7.1 HTTP Bootstrap

Bootstrap ist ein normaler HTTP-Endpoint (Default: /wicos64/bootstrap). Der Client übergibt ein cfg (Profilname) und optional die MAC-Adresse. Der Server liefert daraufhin eine Konfiguration zurück, die der Client dann für API-Zugriffe verwendet.

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

MAC-Format: Der Server normalisiert MAC-Adressen serverseitig. Wenn ein Client 24 Hex-Zeichen sendet (WiC64-Firmware-Eigenheit), wird das auf 12 Zeichen reduziert.

Optional kann der Server pro MAC zusätzliche KEY=VALUE-Zeilen an die Bootstrap-Antwort anhängen (z.B. für Client-spezifische Defaults). Alte Clients ignorieren unbekannte Keys - die Abwärtskompatibilität bleibt damit erhalten.

7.2 UDP Discovery (WDP1)

Discovery läuft per UDP auf Port 6464. Der Client sendet ein DISCOVER-Paket (Magic 'WDP1'), der Server antwortet unicast mit einem OFFER. Der OFFER enthält u.a. die Server-IP, den HTTP-Port und Feature-Flags.

Sicherheit: Discovery sollte nur im LAN genutzt werden. Wenn lan_only=true, ignoriert der Server Pakete von nicht-privaten IPs. Ein Rate-Limit reduziert Spam.

7.3 Fallback: IP Scan

Wenn UDP im Netz blockiert ist und kein DNS-Name zur Verfügung steht, kann der Client per IP-Scan versuchen, den Server zu finden. Dabei wird typischerweise /bootstrap oder /api auf einem Portbereich getestet.

Hinweis: Auf dem C64 kann dieser Scan - je nach Netzgröße und Timeouts - spürbar länger dauern. Eine feste IP per DHCP-Reservation ist daher oft der einfachste Weg.

8 Admin UI - Bedienung & Funktionen

Die Admin-UI ist eine optionale Web-Oberfläche. Sie wird nur aktiv, wenn enable_admin_ui in der config.json gesetzt ist. Standardmäßig lauscht sie auf dem gleichen Port wie die API (z.B. :8080) und ist unter /admin erreichbar.

8.1 Aktivieren & Zugang

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

Sicherheit: Setze ein eigenes Passwort und schalte admin_allow_remote nur ein, wenn Du die Admin-UI wirklich von einem anderen Rechner im LAN erreichen willst. Für maximalen Schutz: admin_allow_remote=false und Admin-UI nur lokal am Server nutzen.

8.2 Dashboard / Status

Im Dashboard siehst Du u.a. Version/Build, Listen-Adresse, API-Endpoint, Basis-Pfad, freien Speicher und die aktivierten Features.

8.3 Config - Form View & JSON View

- **Form View**: komfortabel für übliche Einstellungen (Toggle, Ports, Pfade, Limits)
- **JSON View**: volle Kontrolle; ideal für Copy/Paste oder größere Änderungen

Wichtig: Beim Speichern wird die JSON-Konfiguration vollständig geschrieben. Felder, die in der UI nicht angezeigt werden, dürfen nicht verloren gehen - die UI sollte das komplette JSON round-trip-fähig behandeln.

Neu: In der Config-Ansicht kann optional auch die Bootstrap-Erweiterung pro MAC ("MAC→Extra") gepflegt werden.

8.4 Tokens & Quotas

Im Token-Manager kannst Du Tokens anlegen, bearbeiten und löschen. Pro Token sind typische Einstellungen:

- Name + Token-String
- Root-Folder (relativ zu base_path)
- Quota/Limits (quota_bytes, max_file_bytes)
- Feature Toggles (z.B. disk_images_enabled, disk_images_write_enabled, disk_images_auto_resize_enabled)
- Read-Only (optional)
- Optional: disk_images_allow_rename_convert (nur wenn bewusst gewünscht)

8.5 Live Log

Das Live-Log zeigt die letzten Requests inkl. Zeit, IP, Op-Name, Status, Dauer und Bytes. Über Filter kannst Du z.B. nur LS oder nur CP anzeigen.

8.6 Ops Playground (Mini CLI)

Der Ops Playground ist eine kleine Kommandozeile direkt in der Admin-UI. Damit kannst Du Operationen gezielt testen, ohne am C64 etwas zu ändern. Der Playground nutzt intern denselben Dispatch wie echte Client-Requests, inkl. Token-Kontext (Rechte/Feature-Toggles/Limits).

- Auswahl eines Token-Kontexts (damit Pfad-Mapping und Rechte stimmen)
- Kommando-History (↑/↓ oder Dropdown)
- Ausgabe inkl. Status/err_msg und - wenn sinnvoll - strukturierter Payload
- "Send selected log entry to Playground": markierten Live-Log-Eintrag als Playground-Command übernehmen

```
# Beispiele (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

Optional kann der Server gelöschte Dateien in einen Trash verschieben. Ein Cleanup-Job kann den Trash periodisch leeren. Ähnlich gibt es einen Tmp-Cleanup für temporäre Dateien unter /.tmp (oder dem internen tmp).

9 Konfiguration (config.json) - Referenz

Die Konfiguration ist eine JSON-Datei. Standardpfad ist **config/config.json** neben der Server-EXE (Fallback: legacy config.json im EXE-Ordner). Du kannst den Pfad jederzeit explizit per -config setzen.

9.1 Minimalbeispiel

```
{
  "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 Basis-Einstellungen

Feld	Beschreibung
`listen`	Listen-Adresse (Host:Port). Beispiel ':8080' oder '127.0.0.1:8080'.
`endpoint`	API Pfad. Default: /wicos64/api.
`base_path`	Basis-Verzeichnis für Token-Roots (relative Roots werden darunter aufgelöst).
`server_name`	String, der über CAPS zurückgegeben wird (Client-Infos).

9.3 Tokens (neu, empfohlen)

Die neue Token-Struktur ist ein Array **tokens**. Pro Eintrag:

Feld	Beschreibung
`token`	Token-String, der vom Client gesendet wird.
`name`	Anzeigename (optional, nur für Admin UI).
`root`	Root-Verzeichnis (relativ zu base_path oder absolut).
`enabled`	Optional; wenn false, ist der Token deaktiviert.
`read_only`	Wenn true, sind Schreib-Operationen für diesen Token deaktiviert.
`quota_bytes`	Optionales Quota-Limit.

`max_file_bytes`	Optionales per-File-Limit.
`disk_images_enabled`	Optionaler Override für disk_images_enabled.
`disk_images_write_enabled`	Optionaler Override für disk_images_write_enabled.
`disk_images_auto_resize_enabled`	Optionaler Override für disk_images_auto_resize_enabled (D81 Auto-Resize).
`disk_images_allow_rename_convert`	Optional: erlaubt bewusstes Umbenennen zwischen 'Image' und 'normaler Datei' (Default: false).

9.4 Legacy Token-Konfiguration (weiterhin unterstützt)

Für alte Setups existieren weiterhin die Felder token (Single-Token) und token_roots (Map token->root). Wenn tokens nicht leer ist, hat diese Liste Vorrang.

9.5 Globale Policies

Feld	Beschreibung
`global_read_only`	Schaltet den kompletten Server in Read-Only.
`global_quota_bytes`	Optionales globales Quota (zusätzlich zu token-spezifischem quota_bytes).
`global_max_file_bytes`	Optionales globales per-File-Limit.

9.6 Limits (DoS-Schutz)

Diese Limits werden über CAPS an den Client kommuniziert und serverseitig geprüft:

Feld	Default	Bedeutung
`max_payload`	16384	Max. Payload (HTTP body) in Bytes.
`max_chunk`	4096	Max. Chunk-Größe für READ/WRITE/APPEND.
`max_path`	255	Max. Pfadlänge (String).
`max_name`	64	Max. Segmentlänge (Dateiname/Ordnername).
`max_entries`	50	Max. Anzahl Einträge pro LS (Paging für große Ordner).

9.7 Feature Toggles

Feld	Beschreibung
`enable_mkdir_parents`	Aktiviert MKDIR -p (mkdir -p).
`enable_rmdir_recursive`	Aktiviert RMDIR -r (rm -r).
`enable_cp_recursive`	Aktiviert CP -r (Verzeichnis kopieren).
`enable_overwrite`	Aktiviert overwrite-Flags für WRITE_RANGE/CP/MV.

`enable_errmsg`	Wenn aktiv, liefert der Server zusätzlich err_msg (Text) im Response.

9.8 Disk Images

Feld	Beschreibung
`disk_images_enabled`	Mountet .d64/.d71/.d81 als Verzeichnisse.
`disk_images_write_enabled`	Erlaubt Schreiben/Delete/Rename in gemounteten Images (potentiell destruktiv).
`disk_images_auto_resize_enabled`	Erlaubt Auto-Resize/Repack für D81-Partitionen bei Platzmangel.

9.9 Bootstrap (optional)

Bootstrap liefert API_URL + Token für einen Client - optional per MAC-Zuordnung.

Feld	Beschreibung
`bootstrap.enabled`	Schaltet /bootstrap ein.
`bootstrap.token`	Shared Secret (Config-Token), das für /bootstrap benötigt wird.
`bootstrap.lan_only`	Nur LAN IPs dürfen /bootstrap nutzen.
`bootstrap.unknown_mac_policy`	Was passiert bei unbekannter MAC: 'deny' oder 'legacy'.
`bootstrap.mac_tokens`	Map MAC (AABBCCDDEEFF) -> Token.
`bootstrap.mac_extra`	Optional: Map MAC -> Extra KEY=VALUE (Zeilen), die an die Antwort angehängt werden.

9.10 Discovery (optional)

Feld	Beschreibung
`discovery.enabled`	Schaltet UDP Discovery ein.
`discovery.udp_port`	UDP Port (Default 6464).
`discovery.lan_only`	Nur LAN IPs dürfen Discovery nutzen.
`discovery.rate_limit_per_sec`	Rate-Limit pro Source-IP.

9.11 Trash & Cleanup (optional)

Feld	Beschreibung
`trash_enabled`	Wenn true, werden RM/RMDIR und Overwrite-Operationen nach trash_dir verschoben statt hart gelöscht.

`trash_dir`	Pfad relativ zum Token-Root (Default: .TRASH).
`trash_cleanup_enabled`	Wenn true, räumt ein Job alte Trash-Einträge weg.
`tmp_cleanup_enabled`	Räumt temporäre Dateien unter /.tmp (oder dem internen tmp) weg.

9.12 Compat Toggles

Feld	Beschreibung
`compat.fallback_prg_extension`	READ/STAT/HASH versuchen optional '.PRG', wenn ohne Extension nicht gefunden.
`compat.wildcard_load`	READ/STAT/HASH erlauben '*' und '?' im letzten Segment (Commodore-Style).

10 w64tool - CLI Tool

w64tool ist ein kleines CLI zum Smoke-Testen der API vom PC aus. Es ist bewusst simpel gehalten und eignet sich gut, um Token/URL/Server-Erreichbarkeit zu prüfen.

10.1 Usage

```
# Hilfe / Kommandos
.\w64tool.exe -h

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

Aktuell unterstützte Commands (Auszug): caps, ping, ls, append, hash, search.

10.2 Typische Tests

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

# Features / Limits checken
.\w64tool.exe -url "http://127.0.0.1:8080/wicos64/api?token=WICOS64_DEV1" caps

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

# Disk-Image Mount testen (wenn 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
```

Für umfangreichere Ops (CP/MV/WRITE_RANGE/READ_RANGE) ist der Ops Playground in der Admin-UI aktuell das praktischere Werkzeug.

11 Windows Tray App

Die Tray App ist ein Hilfsprogramm für Windows. Sie richtet sich an Nutzer, die den Server bequem starten/stoppen wollen, ohne eine Konsole offen zu haben.

11.1 Funktionen

- Server starten/stoppen/restarten (je nach Status und Rechten).
- Open Admin UI: öffnet /admin im Browser (wenn Admin UI aktiv).
- Reload config (soft): lädt die config.json neu (ohne kompletten Prozess-Neustart).
- Open config: öffnet die config.json im Editor.
- View server.log: öffnet logs/server.log (besonders wichtig bei 'no-console' Builds).

11.2 Config- und Icon-Pfad

Default-Config: Die Tray App nutzt standardmäßig **config\config.json** neben der EXE. Du kannst alternativ -config angeben. Zur Abwärtskompatibilität wird eine vorhandene config.json im EXE-Ordner ebenfalls akzeptiert.

Icon: bevorzugt **img\wicos64.ico** (Fallback: wicos64.ico im EXE-Ordner).

11.3 Typische Probleme

- Server startet nicht: Logs prüfen (logs/server.log). Häufig sind Port-Konflikte oder ein ungültiges config.json.
- Admin UI öffnet nicht: enable_admin_ui muss true sein; admin_allow_remote beeinflusst die URL-Auswahl.
- Tray zeigt 'no base URL': listen in config.json prüfen (z.B. ':8080').
- Firewall: Windows-Firewall kann eingehende Verbindungen blockieren (HTTP 8080, optional UDP 6464).

12 Troubleshooting & FAQ

Hier sind die häufigsten Fehlerbilder und Lösungen zusammengefasst.

12.1 'payload_len mismatch' / BAD_REQUEST

Wenn WiC64 und Server unterschiedliche Payload-Längen interpretieren, erscheint im Live-Log oft 'payload_len mismatch'. Ursachen können sein:

- Firmware/Emulator sendet ein zusätzliches Byte
- Multipart-Form-Data wird nicht korrekt 'unwrapped' (serverseitig)
- Client hat sich bei der Länge verrechnet (Header vs. Body)

Im Live-Log siehst Du request_bytes/resp_bytes und die err_msg (wenn enable_errmsg aktiv ist).

12.2 'Failed to open connection'

Das ist ein WiC64-HTTP-Problem (DNS, Gateway, Firewall, falsche IP/Port). Prüfe:

- Server läuft und lauscht auf der richtigen IP/Port
- URL im Client ist korrekt (kein Tippfehler wie 192.268.x.x)
- Windows Firewall lässt eingehende Verbindungen zu

12.3 Disk-Image Listing leer

- disk_images_enabled ist global oder für den Token deaktiviert
- Dateiendung wird nicht als Image erkannt (nur .d64/.d71/.d81)
- Image ist beschädigt oder kein echtes D64/D71/D81

12.4 Schreiben in Disk-Images wird abgewiesen

Wenn Schreiben/CP ins Image mit ACCESS_DENIED/NOT_SUPPORTED abgelehnt wird:

- disk_images_write_enabled ist global oder pro Token noch false
- Token ist read_only oder global_read_only ist aktiv
- enable_overwrite ist aus, aber Client versucht overwrite/truncate
- Du versuchst ein Image innerhalb eines anderen Images zu schreiben/zu erstellen (nicht erlaubt)

12.5 D81 Partition voll / Auto-Resize

Bei .d81 können Partitionen (DIRs) voll laufen. Wenn disk_images_auto_resize_enabled aktiv ist, versucht der Server bei Platzmangel ein Repack. Wenn das deaktiviert ist, bekommst Du

typischerweise TOO_LARGE oder einen disk-image-spezifischen Fehler.

- Auto-Resize aktivieren (global oder für den Token), wenn Du das Feature nutzen willst
- Alternativ: Dateien löschen/verschieben oder Image extern reorganisieren
- Beachte: Repack kann I/O-intensiv sein (Image wird neu geschrieben).

12.6 Bulk Copy meldet 'too large'

Bei Bulk Copy kann die Response-Payload sehr groß werden, wenn viele Treffer zurückgemeldet werden. Der Server begrenzt Payloads über max_payload. Lösung: max_payload erhöhen oder Bulk in kleineren Portionen ausführen.

Anhang A - Statuscodes (Auszug)

Der W64F-Statuscode steht im Response-Header (Byte 6). Im Live-Log wird er zusätzlich als Klartext angezeigt. Die vollständige Liste findest Du in der API-Spec; hier ein praxisnaher Auszug:

Name	Code	Bedeutung
`OK`	`0x00`	Operation erfolgreich.
`NOT_FOUND`	`0x01`	Pfad existiert nicht.
`NOT_A_DIR`	`0x02`	Pfad ist kein Verzeichnis.
`IS_A_DIR`	`0x03`	Pfad ist ein Verzeichnis (Operation erwartet Datei).
`ALREADY_EXISTS`	`0x04`	Ziel existiert bereits (z.B. MKDIR auf Datei).
`DIR_NOT_EMPTY`	`0x05`	Verzeichnis nicht leer (RMDIR ohne recursive).
`ACCESS_DENIED`	`0x06`	Token ungültig oder keine Rechte (Read-Only, disk_images_write_disabled, Quota, ...).
`INVALID_PATH`	`0x07`	Ungültiger Pfad oder Pfad außerhalb Root.
`RANGE_INVALID`	`0x08`	Ungültiger Offset/Length bei READ_RANGE/WRITE_RANGE.
`TOO_LARGE`	`0x09`	Payload/Datei/Quota überschritten.
`NOT_SUPPORTED`	`0x0A`	Operation oder Flag ist nicht aktiv (Feature Toggle).
`BUSY`	`0x0B`	Server gerade beschäftigt (z.B. parallele Writes).
`BAD_REQUEST`	`0x0C`	Ungültige Payload/Parameter oder Wildcard-Regeln verletzt.
`INTERNAL`	`0x0D`	Serverinterner Fehler (siehe err_msg).

Anhang B - Beispielsequenzen

B.1 Typischer Client-Start (Bootstrap + PING)

```
1) Client -> GET /wicos64/bootstrap?cfg=RETROBOOT&mac=001122334455
2) Server -> liefert URL: http://192.168.28.40:8080/wicos64/api?token=DEV1
3) Client setzt POST-URL auf diese URL
4) Client sendet PING (W64F Request)
5) Server antwortet OK (W64F Response)
```

B.2 Mount / Zugriff auf 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 Schreiben in Disk-Images (optional)

Voraussetzung: disk_images_write_enabled=true (global oder pro Token).

```
1) mkdir /images/test.d64          # leeres D64 anlegen
2) cp /dev1/INTRO /images/test.d64   # Datei in Image-Root schreiben
3) mv -o /images/test.d64/INTRO /images/test.d64/INTRO2
4) rm /images/test.d64/INTRO2
5) rmdir /images/test.d64            # Image löschen
```

Anhang C - Beispiele für Wildcards

Wildcard-Syntax ist angelehnt an DOS/Unix: * = beliebig viele Zeichen, ? = genau ein Zeichen.
Bei Bulk Copy gilt strict: Wildcards nur im letzten Segment.

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

# BAD_REQUEST (Wildcard in einem mittleren Segment)
cp /u*/tools/* /bin/
```