WiCOS64 Remote Storage API (WiC64/ HTTP POST)

Spezifikation v0.2.2 (kompatibel zu v0.1 Header-Format, RPC-Version=1)

Stand: 2025-12-30

Diese Spezifikation ist für das WiCOS64-Projekt optimiert:

- kleine, robuste Transfers (Chunking)
- einheitliches Verhalten für VFS-Device net:
- geeignet für „System-URL“ (externe Commands/Scripte unter /bin etc.)
- backendseitig einfach zu implementieren, aber erweiterbar ohne Protokollbruch *Begriffe: MUST = muss, SHOULD = sollte, MAY = optional.*

---

0. Ziele und Scope

0.1 Ziel

Der Server stellt einen persistenten, hierarchischen Dateispeicher (Ordner + Dateien) bereit, der von WiCOS64 ueber den WiC64 Adapter wie ein Remote-Laufwerk genutzt wird.

0.2 Kernanforderungen (v0.2.2 "Core")

Ein Server gilt als v0.2.2 Core-konform, wenn er mind. implementiert:

- CAPS (0x0E)
- STATFS (0x0F)
- LS (0x01)
- STAT (0x02)
- READ_RANGE (0x03)
- WRITE_RANGE (0x04)
- MKDIR (0x06)
- RMDIR (0x07)
- RM (0x08)
- CP (0x09)
- MV (0x0A)

Optional (nice-to-have, aber im Protokoll definiert):

- APPEND (0x05)
- SEARCH (0x0B)
- HASH (0x0C)
- PING (0x0D) (legacy)

---

1. Transport (WiC64 / HTTP POST)

1.1 Endpoint

- HTTP POST auf eine feste API-URL (Beispiel):
- `https://dein-server.tld/wicos64/api`
- Request-Body und Response-Body sind binäre RPC-Nachrichten (siehe Abschnitt 4).

1.2 Content-Type

- Client SHOULD senden: `Content-Type: application/octet-stream`
- Server SHOULD antworten: `Content-Type: application/octet-stream`

1.3 HTTP Status vs. App-Status (WICHTIG: eindeutig!)

Ziel: WiCOS64 soll bei App-Fehlern immer eine parsebare W64F-Response bekommen.

MUST:

- Wenn der Server einen Request-Body von mind. 10 Bytes empfangen hat, dann MUST er immer HTTP 200 liefern und eine W64F Response senden (auch bei BAD_REQUEST).
- App-Fehler werden ausschließlich über `Status` im W64F-Response-Header signalisiert (optional mit `err_msg` in der Payload).

HTTP 4xx/5xx sind NUR erlaubt, wenn der Server keine W64F-Response sicher bauen kann, z.B.:

- Body < 10 Bytes (Header nicht lesbar)
- Verbindungs-/Gatewayfehler vor Body-Empfang
- interner Serverfehler, der das Antwortbauen verhindert (sollte selten sein)

Wenn der Request zwar >=10 Bytes ist, aber z.B. Magic falsch, payload_len unplausibel o.ä.:

- Server antwortet trotzdem HTTP200 + W64F(status=BAD_REQUEST) (+ optional err_msg).

1.4 Timeouts / Chunking

- Der Server SHOULD für v0.2.2 einen maximalen "useful" Chunk von 4096..8192 Bytes erlauben.
- Das exakte Maximum wird per CAPS ausverhandelt (`max_chunk`).

1.5 Keine automatische Kompression (praktisch wichtig!)

Viele Webserver komprimieren automatisch (gzip/br). Das würde das Binformat zerstören.

MUST:

- Server MUST keine Content-Encoding-Kompression auf RPC-Responses anwenden.
- `Content-Encoding` MUST entweder fehlen oder `identity` sein.
- Empfehlung: für den Endpoint `Cache-Control: no-transform` setzen.

1.6 Auth / Token (klar definiert)

Da der WiC64 keine frei konfigurierbaren HTTP Header garantiert, gilt für v0.2.2:

- Token wird in der API-URL als Query-Parameter übergeben, z.B.

```
https://dein-server.tld/wicos64/api?token=ABC123
```

Server-Regeln:

- Wenn Auth aktiv ist und Token fehlt/ungültig: HTTP200 + W64F(status=ACCESS_DENIED).
- Server SHOULD Token nicht im Klartext loggen.

---

2. Datentypen und Encoding

2.1 Endianness

Alle Integer sind Little Endian.

2.2 Strings

String = `u16 len` + `len` Bytes, ohne Null-Terminator.

2.3 String-Encoding / erlaubte Zeichen

- v0.2.2 Standard: ASCII (0x20..0x7E).
- Server MUST ablehnen (INVALID_PATH), wenn:
- Control-Chars (<0x20) oder DEL (0x7F) vorkommen
- Null-Byte vorkommt
- WiCOS64-seitig sind Leerzeichen/Quotes noch begrenzt; daher gilt als Empfehlung:
- Dateinamen SHOULD aus `A-Z 0-9 . _ -` bestehen.
- Andere ASCII-Zeichen MAY erlaubt sein, sind aber für Shell-Komfort nicht garantiert.

2.4 Case-Policy

- Server MUST Pfade case-insensitive behandeln (ASCII-Folding).
- Server SHOULD Namen im LS/STAT in kanonischer Großschreibung zurückgeben (damit C64 & IEC sich gleich anfühlen).
- (Optional) Wenn der Server Case-preserving sein will: ok, aber Matching bleibt case-insensitive.

---

3. Pfade / Normalisierung / Sandbox

3.1 Pfadregeln

- Separator: `/`
- Root ist `/`
- Keine Backslashes
- Max. Pfadlänge: 255 Bytes (MUST für v0.2.2, kann in CAPS bestätigt werden)

3.2 Normalisierung (MUST)

Der Server MUST folgende Normalisierung/Validierung durchführen:

- `//` -> `/` (mehrfache Slashes kollabieren)
- `/./` entfernen
- Trailing slash: `/dir` und `/dir/` sind identisch
- `..` ist verboten (INVALID_PATH)
- Pfad MUST im Benutzer-Root bleiben (Sandbox), andernfalls INVALID_PATH oder ACCESS_DENIED

3.3 Benutzer-Sandbox

- Jeder Request wird gegen ein User-Root aufgelöst (z.B. anhand Token).
- Der Server MUST verhindern, dass außerhalb dieses Roots zugegriffen wird.

---

4. RPC-Format (POST-Body)

Jeder HTTP POST Body enthält genau eine RPC-Nachricht.

4.1 Request Header (Client -> Server)

Offset	Size	Feld	Beschreibung
0	4	magic	ASCII "`W64F`"
4	1	version	1 (RPC-Version)
5	1	op	Opcode
6	1	flags	Optionsbits (op-spezifisch)
7	1	reserved	MUST 0
8	2	payload_len	u16 LE

4.2 Response Header (Server -> Client)

Offset	Size	Feld	Beschreibung
0	4	magic	"`W64F`"
4	1	version	echo request version
5	1	op_echo	echo request op (oder `0xFF` wenn Request nicht valide)
6	1	status	App-Status (0 = OK)
7	1	reserved	MUST 0
8	2	payload_len	u16 LE

4.3 Allgemeine Payload-Regeln (MUST)

- `payload_len` MUST konsistent zum Body sein.
- Wenn `payload_len` nicht zur realen Länge passt:
- Server antwortet HTTP200 + W64F(status=BAD_REQUEST) (+ optional err_msg).
- Bei status != 0 MAY der Server als Payload eine optionale Debug-Message liefern:
- `string err_msg`
- Client darf sie ignorieren.

5. App-Statuscodes

Code	Name	Bedeutung
0	OK	Erfolg
1	NOT_FOUND	Pfad existiert nicht
2	NOT_A_DIR	Pfad ist keine Directory
3	IS_A_DIR	Pfad ist eine Directory
4	ALREADY_EXISTS	Ziel existiert bereits
5	DIR_NOT_EMPTY	Dir nicht leer (bei RMDIR ohne rekursiv)
6	ACCESS_DENIED	Token/Policy erlaubt Operation nicht
7	INVALID_PATH	Ungültiger Pfad (.., verbotene Zeichen, außerhalb Sandbox)
8	RANGE_INVALID	Offset/Länge ungültig
9	TOO_LARGE	Request/Response zu groß (Chunk zu groß)
10	NOT_SUPPORTED	Opcode/Flag nicht implementiert
11	BUSY	Ressource gesperrt / Server beschäftigt
12	BAD_REQUEST	Payload-Format ungültig (Längen passen nicht)
13	INTERNAL	Interner Fehler (Server)

6. Opcodes und Payload-Layouts

Alle u16/u32: Little Endian. Strings: u16+bytes.

0x0E CAPS (Pflicht für v0.2.2 Core)

Zweck: Feature-/Limit-Aushandlung ohne Rateversuche.

Request Payload: leer Response Payload:

Feld	Typ	Beschreibung
max_chunk	u16	Max. Datenbytes pro READ_RANGE/WRITE_RANGE Chunk (z.B. 4096)
max_payload	u16	Max. `payload_len` (ohne 10-Byte Header), z.B. 16384
max_path	u16	Max. Pfadbytes (Empfehlung 255)
max_name	u16	Max. Namebytes (Empfehlung 64)
max_entries	u16	Max. Einträge für LS/SEARCH pro Response (Empfehlung 50)
features_lo	u32	Feature-Bits (siehe unten)
server_time_unix	u32	UTC seconds (0 wenn unbekannt)
server_name	string	optionaler Name/Build

Feature-Bits (features_lo):

Bit	Name	Bedeutung
0	FEAT_STATFS	STATFS implementiert
1	FEAT_APPEND	APPEND implementiert
2	FEAT_SEARCH	SEARCH implementiert
3	FEAT_HASH_CRC32	HASH CRC32 implementiert
4	FEAT_HASH_SHA1	HASH SHA1 implementiert
5	FEAT_MKDIR_PARENTS	MKDIR -p (Flag PARENTS)
6	FEAT_RMDIR_RECURSIVE	RMDIR -r (Flag RECURSIVE)
7	FEAT_CP_RECURSIVE	CP rekursiv
8	FEAT_OVERWRITE	CP/MV Overwrite
9	FEAT_ERRMSG	Fehlerpayload err_msg wird geliefert

0x0F STATFS

Zweck: Platz-Info für Statuszeile / Cache-Strategie.

Request Payload:

Feld	Typ	Beschreibung
path	string	optional; wenn leer -> "/"

Response Payload:

Feld	Typ	Beschreibung
total_bytes	u32	Gesamt (0 wenn unbekannt)
free_bytes	u32	frei (0 wenn unbekannt)
used_bytes	u32	benutzt (0 wenn unbekannt)

Server-Regeln:

- Wenn `path` leer: behandeln wie `/`.
- Wenn `path` nicht existiert: NOT_FOUND.

0x01 LS

Listet Einträge eines Verzeichnisses (paginierbar).

Request Payload:

Feld	Typ	Beschreibung
path	string	Directory (leer -> "/")
start_index	u16	0 fuer erste Seite
max_entries	u16	10..50 empfohlen (Server darf clampen)

Response Payload:

Feld	Typ	Beschreibung
count	u16	Anzahl Eintraege in dieser Seite
entries[]	struct	count-mal
next_index	u16	0xFFFF wenn Ende, sonst start_index+count

LS Entry struct:

- `u8 type` (0=file, 1=dir)
- `u32 size` (bei dir 0)
- `u32 mtime_unix`
- `string name`

Server-Regeln:

- MUST stabile Sortierung liefern (empfohlen: Name, case-insensitive).
- SHOULD Namen als Großschrift ausgeben (Abschnitt 2.4).
- Wenn `path` ein File ist: status=NOT_A_DIR.
- Wenn `path` nicht existiert: status=NOT_FOUND.
- Wenn `start_index` >= Anzahl Einträge: status=OK, `count=0`, `next_index=0xFFFF`.

0x02 STAT

Metadaten für einen Pfad.

Request Payload:

- `path string` (leer -> "/")

Response Payload:

- `type u8` (0=file,1=dir)
- `size u32`
- `mtime_unix u32`

0x03 READ_RANGE (klar definierte EOF/Range-Regeln)

Liest einen Bereich aus einer Datei.

Request Payload:

Feld	Typ
path	string
offset	u32
length	u16

Response Payload:

- rohe Bytes (0..length Bytes)
- `payload_len` im Response-Header ist die tatsächliche Bytezahl

MUST Regeln:

- Wenn `path` eine Directory ist: status=IS_A_DIR.
- Wenn `path` nicht existiert: status=NOT_FOUND.
- Wenn `length > max_chunk`: status=TOO_LARGE.
- Wenn `offset == size`: status=OK und Response-Payload hat 0 Bytes (EOF).
- Wenn `offset > size`: status=RANGE_INVALID.

0x04 WRITE_RANGE (klar definierte Range/Chunk-Regeln)

Schreibt Bytes in eine Datei ab Offset.

Flags:

- Bit0 TRUNCATE: Datei vor Schreiben auf 0 kürzen
- Bit1 CREATE: Datei anlegen, falls nicht existiert

Request Payload:

Feld	Typ
path	string
offset	u32
data_len	u16
data	bytes[data_len]

Response Payload: leer (payload_len=0)

MUST Regeln:

- `data_len` MUST exakt zur Anzahl der verbleibenden Payload-Bytes passen, sonst BAD_REQUEST.
- Wenn `data_len > max_chunk`: status=TOO_LARGE.
- Wenn `TRUNCATE` gesetzt ist: `offset` MUST 0 sein, sonst BAD_REQUEST.
- Existiert die Datei nicht und `CREATE` ist nicht gesetzt: NOT_FOUND.
- Ist `path` eine Directory: IS_A_DIR.
- Offset-Semantik (kein Sparse Write in v0.2.2):
- `offset > size` => RANGE_INVALID
- `offset == size` ist erlaubt (append per WRITE_RANGE)
- `offset < size` überschreibt Bytes ab Offset

0x05 APPEND (optional)

Flags:

- Bit1 CREATE (wie WRITE_RANGE)

Request Payload:

- path string
- data_len u16
- data bytes

Response: leer

Hinweis Idempotenz: APPEND ist bei Retries nicht idempotent. WiCOS64 sollte bevorzugt WRITE_RANGE nutzen.

0x06 MKDIR

Flags:

- Bit0 PARENTS: Eltern automatisch anlegen (-p) (nur wenn FEAT_MKDIR_PARENTS)

Request: `path string`

Response: leer

Fehlerregeln:

- Wenn `path` bereits existiert und eine Directory ist: OK.
- Wenn `path` bereits existiert und eine Datei ist: ALREADY_EXISTS.

0x07 RMDIR

Flags:

- Bit0 RECURSIVE (-r) (nur wenn FEAT_RMDIR_RECURSIVE)

Request: `path string`

Response: leer

Fehlerregeln:

- Wenn `path` ein File ist: NOT_A_DIR.
- Wenn Dir nicht leer und RECURSIVE nicht gesetzt: DIR_NOT_EMPTY.

0x08 RM

Loescht Datei.

Request: `path string`

Response: leer

Fehlerregeln:

- Wenn `path` eine Directory ist: IS_A_DIR.

0x09 CP

Flags:

- Bit0 OVERWRITE
- Bit1 RECURSIVE (wenn src dir)

Request:

- `src_path string`
- `dst_path string`

Response: leer

Fehlerregeln:

- Wenn `src_path` eine Directory ist und RECURSIVE nicht gesetzt: IS_A_DIR.
- Wenn `dst_path` existiert und OVERWRITE nicht gesetzt: ALREADY_EXISTS.

0x0A MV

Flags:

- Bit0 OVERWRITE

Request:

- `src_path string`
- `dst_path string`

Response: leer

Fehlerregeln:

- Wenn `dst_path` existiert und OVERWRITE nicht gesetzt: ALREADY_EXISTS.

0x0B SEARCH (optional)

Flags:

- Bit0 CASE_INSENSITIVE
- Bit1 RECURSIVE
- Bit2 WHOLE_WORD (optional)

Request:

- base_path string
- query string
- start_index u16
- max_results u16 (10..30)
- max_scan_bytes u32 (0=default)

Response:

- count u16
- hits[] struct
- next_index u16 (0xFFFF Ende)

Hit struct:

- path string
- offset u32
- preview_len u16
- preview bytes[preview_len]

Server-Regeln:

- Wenn `start_index` >= Anzahl Treffer: status=OK, `count=0`, `next_index=0xFFFF`.

0x0C HASH (optional)

Flags:

- Bit0 ALGO (0=CRC32,1=SHA1)

Request: `path string`

Response:

- bei CRC32: `u32`
- bei SHA1: 20 bytes

0x0D PING (legacy optional)

Request: leer

Response: string (z.B. "WICOS64-API 0.1") oder leer

---

7. Atomare Writes / Upload-Recipe (wichtig!)

Da WRITE_RANGE chunked ist, definiert v0.2.2 eine sichere Vorgehensweise:

Server SHOULD atomar umsetzen, Client SHOULD so senden:

1) Upload in temp:

- Ziel: `/.tmp/<name>.<rand>`
- erster Chunk: WRITE_RANGE(TRUNCATE|CREATE), offset=0
- weitere Chunks: WRITE_RANGE, offset+=len

2) Nach Erfolg: MV tmp -> final (OVERWRITE wenn gewünscht)
3) Bei Abbruch: RM tmp (best effort)

Damit entstehen keine "halb geschriebenen" Enddateien.

---

8. Verzeichnis- und Pagination-Semantik

- LS Pagination basiert auf start_index. Der Server MUST stabil sortieren.
- Wenn sich das Dir zwischen Seiten ändert, können Duplikate/Skips passieren; das ist akzeptabel.

---

9. Server-Verzeichnislayout (WiCOS64)

- `/bin` : System-Kommandos (externe Commands, Module)
- `/usr` : Benutzerprogramme, Scripts, Assets
- `/etc` : Konfiguration (optional)
- `/.tmp` : temporaer (Uploads, Cache)

Der Server MAY diese Verzeichnisse beim ersten Login initial anlegen.

---

10. Mapping WiCOS64 Shell/VFS -> API

WiCOS64-Kommando	API
ls [path]	LS
cd path	STAT + CWD lokal setzen
pwd	lokal
cat file	READ_RANGE Schleife
load file [$addr]	READ_RANGE Schleife
save file $addr $len	WRITE_RANGE Schleife (TRUNCATE im ersten Chunk)
mkdir/md dir	MKDIR
rmdir/rd dir	RMDIR
rm file	RM
cp src dst	CP
mv src dst	MV
xms/ramd info	lokal (bzw. STATFS für net:)

---

11. Implementierungshinweise für Backend

11.1 Robustheit (zusammengefasst)

- Keine HTML-Fehlerseiten im Body.
- Bei parsebarem Request (>=10 Bytes) immer HTTP200 + W64F-Response.
- Fehler werden über Status kodiert, optional err_msg als String.

11.2 Limits (empfohlen)

- max_chunk: 4096 (default), optional 8192
- max_payload: 16384 (oder kleiner, aber dann über CAPS melden)
- max_entries LS: 50
- max_name: 64
- max_path: 255
- SEARCH: serverseitig harte Limits (max_runtime / max_files / max_scan_bytes)

11.3 Security

- Token pro Benutzer; Root sandbox.
- Rate-Limit pro Token/IP.
- Optional Logging, aber Token nicht im Klartext loggen.

---

12. Kompatibilität zu v0.1

- Header/Grundopcodes entsprechen v0.1.
- v0.2.2 fügt CAPS/STATFS und Klarstellungen hinzu.
- Ein v0.1 Server kann CAPS/STATFS mit NOT_SUPPORTED beantworten.

---

13. Änderungen v0.2.1 -> v0.2.2 (Klarstellungen, kein Protokollbruch)

- MKDIR: wenn Ziel bereits existiert und eine Directory ist -> OK; wenn es eine Datei ist -> ALREADY_EXISTS.
- CP/MV: wenn `dst_path` existiert und OVERWRITE nicht gesetzt ist -> ALREADY_EXISTS.
- LS/SEARCH: wenn `start_index` >= Gesamtanzahl -> OK mit `count=0` und `next_index=0xFFFF` (Ende).