WiCOS64 Remote Storage API

WiC64 / HTTP POST

Specification v0.2.2 (compatible with v0.1 header format, RPC version = 1)

Last updated: 2025-12-30

This specification is optimized for the WiCOS64 project:

- small, robust transfers (chunking)
- consistent behavior for the VFS device `net:`
- suitable for a “system URL” (external commands/scripts under `/bin`, etc.)
- easy to implement on the backend, yet extensible without breaking the protocol

Terminology: MUST = mandatory, SHOULD = recommended, MAY = optional.

Table of Contents

0. Goals and Scope

0.1 Objective

The server provides a persistent, hierarchical file store (folders + files) that can be used by WiCOS64 via the WiC64 adapter like a remote drive.

0.2 Core Requirements (v0.2.2 "Core")

A server is considered v0.2.2 Core compliant if it implements at least:

- 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, but defined in the protocol):

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

1. Transport (WiC64 / HTTP POST)

1.1 Endpoint

- HTTP POST to a fixed API URL (example):
- `https://your-server.tld/wicos64/api`
- Request body and response body are binary RPC messages (see Section 4).

1.2 Content-Type

- Client SHOULD send: `Content-Type: application/octet-stream`
- Server SHOULD respond: `Content-Type: application/octet-stream`

1.3 HTTP Status vs. App Status (IMPORTANT: unambiguous!)

Goal: even on application errors, WiCOS64 should always receive a parseable `W64F` response.

MUST:

- If the server received a request body of at least 10 bytes, it MUST always return HTTP 200 and send a `W64F` response (also for `BAD_REQUEST`).
- Application errors are signaled exclusively via the `status` field in the `W64F` response header (optionally with `err_msg` in the payload).

HTTP 4xx/5xx are allowed ONLY if the server cannot safely build a `W64F` response, e.g.:

- Body < 10 bytes (header cannot be read)
- connection/gateway failure before the body is received
- internal server failure that prevents building the response (should be rare)

If the request is >= 10 bytes but, for example, has the wrong magic or an implausible `payload_len`, etc.:

- The server still responds with HTTP 200 + `W64F(status=BAD_REQUEST)` (+ optional `err_msg`).

1.4 Timeouts / Chunking

- For v0.2.2 the server SHOULD allow a maximum “useful” chunk size of 4096..8192 bytes.
- The exact maximum is negotiated via `CAPS` (`max_chunk`).

1.5 No Automatic Compression (practically important!)

Many web servers compress automatically (gzip/br). That would destroy the binary format.

MUST:

- The server MUST not apply any `Content-Encoding` compression to RPC responses.
- `Content-Encoding` MUST either be absent or be `identity`.
- Recommendation: set `Cache-Control: no-transform` for the endpoint.

1.6 Auth / Token (clearly defined)

Because the WiC64 does not reliably support freely configurable HTTP headers, v0.2.2 uses:

- Token is passed as a query parameter in the API URL, e.g.
- `https://your-server.tld/wicos64/api?token=ABC123`

Server rules:

- If authentication is active and the token is missing/invalid: HTTP 200 + `W64F(status=ACCESS_DENIED)`.
- The server SHOULD not log the token in clear text.

2. Data Types and Encoding

2.1 Endianness

All integers are little endian.

2.2 Strings

String = `u16 len` + `len` bytes, without a null terminator.

2.3 String Encoding / Allowed Characters

- v0.2.2 standard: ASCII (0x20..0x7E).

The server MUST reject (`INVALID_PATH`) if:

- control characters (< 0x20) or DEL (0x7F) are present
- a null byte is present

On the WiCOS64 side, spaces/quotes are still limited; therefore the recommendation is:

- filenames SHOULD consist of `A-Z 0-9 . _ -`
- other ASCII characters MAY be allowed, but shell comfort is not guaranteed

2.4 Case Policy

- The server MUST treat paths case-insensitively (ASCII folding).
- The server SHOULD return names in `LS`/`STAT` in canonical uppercase (so C64 & IEC feel the same).
- (Optional) If the server wants to preserve case: that's fine, but matching remains case-insensitive.

3. Paths / Normalization / Sandbox

3.1 Path Rules

- Separator: `/`
- Root is `/`
- No backslashes
- Max path length: 255 bytes (MUST for v0.2.2; can be confirmed in `CAPS`)

3.2 Normalization (MUST)

The server MUST perform the following normalization/validation:

- `//` → `/` (collapse multiple slashes)
- remove `/./`
- trailing slash: `/dir` and `/dir/` are identical
- `..` is forbidden (`INVALID_PATH`)
- the path MUST remain inside the user root (sandbox), otherwise `INVALID_PATH` or `ACCESS_DENIED`

3.3 User Sandbox

- Each request is resolved against a user root (e.g. derived from the token).
- The server MUST prevent access outside this root.

4. RPC Format (POST body)

Each HTTP POST body contains exactly one RPC message.

4.1 Request Header (Client → Server)

Offset	Size	Field	Description
0	4	magic	ASCII `"W64F"`
4	1	version	1 (RPC version)
5	1	op	Opcode
6	1	flags	Option bits (operation-specific)
7	1	reserved	MUST be 0
8	2	payload_len	`u16` LE

4.2 Response Header (Server → Client)

Offset	Size	Field	Description
0	4	magic	`"W64F"`
4	1	version	echo request version
5	1	op_echo	echo request op (or 0xFF if request is not valid)
6	1	status	App status (0 = OK)
7	1	reserved	MUST be 0
8	2	payload_len	`u16` LE

4.3 General Payload Rules (MUST)

- `payload_len` MUST be consistent with the body.
- If `payload_len` does not match the actual length:
  - the server responds with HTTP 200 + `W64F(status=BAD_REQUEST)` (+ optional `err_msg`).
- If `status` ≠ 0, the server MAY provide an optional debug message as payload:
- `string err_msg`
- The client may ignore it.

5. Application Status Codes

Code	Name	Meaning
0	OK	Success
1	NOT_FOUND	Path does not exist
2	NOT_A_DIR	Path is not a directory
3	IS_A_DIR	Path is a directory
4	ALREADY_EXISTS	Destination already exists
5	DIR_NOT_EMPTY	Directory not empty (RMDIR without recursive)
6	ACCESS_DENIED	Token/policy does not allow operation
7	INVALID_PATH	Invalid path (.., forbidden characters, outside sandbox)
8	RANGE_INVALID	Offset/length invalid
9	TOO_LARGE	Request/response too large (chunk too large)
10	NOT_SUPPORTED	Opcode/flag not implemented
11	BUSY	Resource locked / server busy
12	BAD_REQUEST	Invalid payload format (lengths do not match)
13	INTERNAL	Internal server error

6. Opcodes and Payload Layouts

All u16/u32: little endian. Strings: `u16 + bytes`.

0x0E CAPS (required for v0.2.2 Core)

Purpose: negotiate features/limits without trial-and-error.

Request payload: empty

Response payload:

Field	Type	Description
max_chunk	u16	Max data bytes per READ_RANGE/WRITE_RANGE chunk (e.g. 4096)
max_payload	u16	Max payload_len (excluding 10-byte header), e.g. 16384
max_path	u16	Max path bytes (recommended 255)
max_name	u16	Max name bytes (recommended 64)
max_entries	u16	Max entries for LS/SEARCH per response (recommended 50)
features_lo	u32	Feature bits (see below)
server_time_unix	u32	UTC seconds (0 if unknown)
server_name	string	Optional name/build

Feature bits (`features_lo`):

Bit	Name	Meaning
0	FEAT_STATFS	STATFS implemented
1	FEAT_APPEND	APPEND implemented
2	FEAT_SEARCH	SEARCH implemented
3	FEAT_HASH_CRC32	HASH CRC32 implemented
4	FEAT_HASH_SHA1	HASH SHA1 implemented
5	FEAT_MKDIR_PARENTS	MKDIR -p (flag PARENTS)
6	FEAT_RMDIR_RECURSIVE	RMDIR -r (flag RECURSIVE)
7	FEAT_CP_RECURSIVE	CP recursive
8	FEAT_OVERWRITE	CP/MV overwrite
9	FEAT_ERRMSG	Error payload `err_msg` is provided

0x0F STATFS

Purpose: space info for status line / cache strategy.

Request payload:

Field	Type	Description
path	string	Optional; if empty → `"/"`

Response payload:

Field	Type	Description
total_bytes	u32	Total (0 if unknown)
free_bytes	u32	Free (0 if unknown)
used_bytes	u32	Used (0 if unknown)

Server rules:

- If `path` is empty: treat as `/`.
- If `path` does not exist: `NOT_FOUND`.

0x01 LS

Lists entries of a directory (paginated).

Request payload:

Field	Type	Description
path	string	Directory (empty → "/")

start_index	u16	0 for first page
max_entries	u16	10..50 recommended (server may clamp)

Response payload:

Field	Type	Description
count	u16	Number of entries in this page
entries[]	struct	Repeated `count` times
next_index	u16	0xFFFF if end; otherwise `start_index + count`

LS entry struct:

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

Server rules:

- MUST provide stable sorting (recommended: name, case-insensitive).
- SHOULD output names in uppercase (Section 2.4).
- If `path` is a file: `NOT_A_DIR`.
- If `path` does not exist: `NOT_FOUND`.
- If `start_index` ≥ number of entries: `status=OK`, `count=0`, `next_index=0xFFFF`.

0x02 STAT

Metadata for a path.

Request payload:

Field	Type	Description
path	string	Empty → `"/"`

Response payload:

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

0x03 READ_RANGE (clearly defined EOF/range rules)

Reads a range from a file.

Request payload:

Field	Type

path	string
offset	u32
length	u16

Response payload:

- raw bytes (0..length bytes)
- `payload_len` in the response header is the actual number of bytes

MUST rules:

- If `path` is a directory: `IS_A_DIR`.
- If `path` does not exist: `NOT_FOUND`.
- If `length` > `max_chunk`: `TOO_LARGE`.
- If `offset == size`: `OK` and the response payload has 0 bytes (EOF).
- If `offset > size`: `RANGE_INVALID`.

0x04 WRITE_RANGE (clearly defined range/chunk rules)

Writes bytes into a file at an offset.

Flags:

- Bit 0 `TRUNCATE`: truncate file to 0 before writing
- Bit 1 `CREATE`: create file if it does not exist

Request payload:

Field	Type
path	string
offset	u32
data_len	u16
data	bytes[data_len]

Response payload: empty (`payload_len = 0`)

MUST rules:

- `data_len` MUST exactly match the number of remaining payload bytes; otherwise `BAD_REQUEST`.
- If `data_len` > `max_chunk`: `TOO_LARGE`.
- If `TRUNCATE` is set: `offset` MUST be 0; otherwise `BAD_REQUEST`.
- If the file does not exist and `CREATE` is not set: `NOT_FOUND`.
- If `path` is a directory: `IS_A_DIR`.
- Offset semantics (no sparse writes in v0.2.2):

- `offset > size` → `RANGE_INVALID`
- `offset == size` is allowed (append via WRITE_RANGE)
- `offset < size` overwrites bytes starting at `offset`

0x05 APPEND (optional)

Flags:

- Bit 1 `CREATE` (like WRITE_RANGE)

Request payload:

- `path string`
- `data_len u16`
- `data bytes`

Response: empty

Idempotency note: APPEND is not idempotent on retries. WiCOS64 should prefer WRITE_RANGE.

0x06 MKDIR

Flags:

- Bit 0 `PARENTS`: auto-create parent directories (`-p`) (only if `FEAT_MKDIR_PARENTS`)

Request: `path string`

Response: empty

Error rules:

- If `path` already exists and is a directory: `OK`.
- If `path` already exists and is a file: `ALREADY_EXISTS`.

0x07 RMDIR

Flags:

- Bit 0 `RECURSIVE`: recursive delete (`-r`) (only if `FEAT_RMDIR_RECURSIVE`)

Request: `path string`

Response: empty

Error rules:

- If `path` is a file: `NOT_A_DIR`.
- If the directory is not empty and `RECURSIVE` is not set: `DIR_NOT_EMPTY`.

0x08 RM

Deletes a file.

Request: `path string`

Response: empty

Error rules:

- If `path` is a directory: `IS_A_DIR`.

0x09 CP

Flags:

- Bit 0 `OVERWRITE`
- Bit 1 `RECURSIVE` (if `src` is a directory)

Request:

- `src_path string`
- `dst_path string`

Response: empty

Error rules:

- If `src_path` is a directory and `RECURSIVE` is not set: `IS_A_DIR`.
- If `dst_path` exists and `OVERWRITE` is not set: `ALREADY_EXISTS`.

0x0A MV

Flags:

- Bit 0 `OVERWRITE`

Request:

- `src_path string`
- `dst_path string`

Response: empty

Error rules:

- If `dst_path` exists and `OVERWRITE` is not set: `ALREADY_EXISTS`.

0x0B SEARCH (optional)

Flags:

- Bit 0 `CASE_INSENSITIVE`
- Bit 1 `RECURSIVE`
- Bit 2 `WHOLE_WORD` (optional)

Request:

Field	Type	Description

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

Response:

Field	Type	Description
count	u16	
hits[]	struct	
next_index	u16	0xFFFF end

Hit struct:

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

Server rules:

- If `start_index` ≥ number of hits: `status=OK`, `count=0`, `next_index=0xFFFF`.

0x0C HASH (optional)

Flags:

- Bit 0 `ALGO` (0=CRC32, 1=SHA1)

Request: `path string`

Response:

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

0x0D PING (legacy optional)

Request: empty

Response: `string` (e.g. "WICOS64-API 0.1") or empty

7. Atomic Writes / Upload Recipe (important!)

Because WRITE_RANGE is chunked, v0.2.2 defines a safe procedure.

The server SHOULD implement this atomically; the client SHOULD send:

1) Upload to a temporary path:

- Target: `/.tmp/<name>.<rand>`
- First chunk: `WRITE_RANGE(TRUNCATE|CREATE)`, `offset=0`
- Further chunks: `WRITE_RANGE`, `offset += len`

2) After success:

- `MV` temp → final (`OVERWRITE` if desired)

3) On abort:

- `RM` temp (best effort)

This prevents “half-written” final files.

8. Directory and Pagination Semantics

- LS pagination is based on `start_index`. The server MUST sort stably.
- If the directory changes between pages, duplicates/skips can happen; this is acceptable.

9. Server Directory Layout (WiCOS64)

- `/bin`: system commands (external commands, modules)
- `/usr`: user programs, scripts, assets
- `/etc`: configuration (optional)
- `/.tmp`: temporary (uploads, cache)

The server MAY create these directories on first login.

10. Mapping WiCOS64 Shell/VFS → API

WiCOS64 command	API
ls [path]	LS
cd path	STAT + set CWD locally
pwd	local
cat file	READ_RANGE loop
load file [$addr]	READ_RANGE loop
save file $addr $len	WRITE_RANGE loop (TRUNCATE in first chunk)
mkdir / md dir	MKDIR
rmdir / rd dir	RMDIR
rm file	RM

cp src dst	CP
mv src dst	MV
xms/ramd info	local (or STATFS for net:)

11. Backend Implementation Notes

11.1 Robustness (summary)

- No HTML error pages in the body.
- For a parseable request (>= 10 bytes), always HTTP 200 + `W64F` response.
- Errors are coded via `status`, optional `err_msg` as string.

11.2 Limits (recommended)

- `max_chunk`: 4096 (default), optionally 8192
- `max_payload`: 16384 (or smaller, but then report via CAPS)
- `max_entries` for LS: 50
- `max_name`: 64
- `max_path`: 255
- SEARCH: enforce hard server-side limits (`max_runtime` / `max_files` / `max_scan_bytes`)

11.3 Security

- One token per user; root sandbox.
- Rate limit per token/IP.
- Optional logging, but do not log tokens in clear text.

12. Compatibility with v0.1

- Header and basic opcodes correspond to v0.1.
- v0.2.2 adds CAPS/STATFS and clarifications.
- A v0.1 server can answer CAPS/STATFS with `NOT_SUPPORTED`.

13. Changes v0.2.1 → v0.2.2 (clarifications, no protocol break)

- MKDIR: if the target already exists and is a directory → OK; if it is a file → ALREADY_EXISTS.
- CP/MV: if `dst_path` exists and OVERWRITE is not set → ALREADY_EXISTS.
- LS/SEARCH: if `start_index` ≥ total count → OK with `count=0` and `next_index=0xFFFF` (end).