NABU RetroNET Command Protocol Specification

DJ Sures

Version: 2026.02.14.01

Derived from: RetroNET-FileStore.[ch], RetroNET-IAControl.h, RetroNET-CPMDrive.h, NABU-LIB.[ch] (NABU-LIB)

Scope: Binary command protocol carried over the NABU Internet Adapter HCCA link.

Table of Contents

1. **Architecture and Transport**
   1.1 Roles
   1.2 Physical/Link Layer (HCCA)
   1.3 Two execution modes
2. **Data Types and Encoding Rules**
   2.1 Integer endianness
   2.2 Strings
   2.3 Filenames and casing
   2.4 Return values and sentinel values
3. **Command Format**
4. **RetroNET File Store Commands**
   4.1 Constants (flags)
   4.2 FileDetailsStruct response layout (83 bytes)
   4.3 File/Directory command reference
      - 0xA3 Open file (get/assign handle)
      - 0xA7 Close handle
      - 0xA8 File size by filename
      - 0xE7 Read by filename (no handle)
      - 0xE8 Replace bytes by filename (no handle)
      - 0xA4 Size by handle
      - 0xA5 Read by handle (random access)
      - 0xA9 Append
      - 0xAA Insert
      - 0xAB Delete range
      - 0xB0 Empty file (truncate to 0)
      - 0xAC Replace bytes by handle (overwrite-in-place)
      - 0xAD Delete physical file
      - 0xAE Copy

- 0xAF Move
- 0xB1 List directory
- 0xB2 List item details (by index)
- 0xB3 File details by filename
- 0xB4 File details by handle
- 0xB5 Sequential read
- 0xB6 Seek
- 0xDC Count lines in text file
- 0xDD Get specific line as text

5. **RetroNET TCP Client Commands**
    - 0xD0 Open TCP connection
    - 0xD1 Close TCP handle
    - 0xD2 Bytes available to read (TCP handle)
    - 0xD3 Read from TCP handle
    - 0xD4 Write to TCP handle
6. **RetroNET TCP Server Commands (IA-hosted server)**
    - 0xD5 Connected client count
    - 0xD6 Bytes available to read (server)
    - 0xD7 Read from TCP server channel
    - 0xD8 Write to TCP server channel
7. **Printer / Punch-Out Output**
    - 0xDA Printer output (one byte)
    - 0xDB Punch-out output (one byte)
8. **CP/M Drive Manager Commands**
    - 0xDE Build drive
    - 0xDF Extract drive

9. **IA Control Command Group**

9.1 Channel / Parent / Child browsing

- 0xBA/0x00 Get parent count
- 0xBA/0x01 Get parent name
- 0xBA/0x0D Get parent description
- 0xBA/0x18 Get child count (extended)
- 0xBA/0x19 Get child name (extended)
- 0xBA/0x1A Set selection (extended)
- 0xBA/0x1B Get child description (extended)
- 0xBA/0x1C Get child author (extended)
- 0xBA/0x1D Get child icon tile color
- 0xBA/0x1E Get child icon tile pattern

9.2 News and logs

- 0xBA/0x07 Get current news content
- 0xBA/0x0A Get IA log tail
- 0xBA/0x0B Get news title
- 0xBA/0x0C Get news date
- 0xBA/0x0E Get news count
- 0xBA/0x0F Get news content by id
- 0xBA/0x10 Get news title by id
- 0xBA/0x11 Get news date by id
- 0xBA/0x12 Get news icon tile color by id
- 0xBA/0x13 Get news icon tile pattern by id

9.3 Adapter info / time

- 0xBA/0x14 Get adapter OS
- 0xBA/0x15 Get current date/time string

- 0xBA/0x16 Get adapter version string
- 0xBA/0x17 Is new adapter version available?

10. **Worked Examples**
    10.1 Read a whole file in blocks (handle-based)
    10.2 List directory entries
    10.3 Simple TCP client request/response
11. **Implementation Guidance and Gotchas**
    11.1 Read exactly what the command returns
    11.2 Length-prefixed reads defensively
    11.3 Close handles
    11.4 Uppercase filenames
    11.5 Sequential read for streaming
    11.6 Not found vs directory

---

1) Architecture and Transport

1.1 Roles

- **NABU (client):** Issues commands over the HCCA interface and reads responses immediately.
- **Internet Adapter (IA / server):** Executes filesystem, TCP, “channels/news”, and CPM-drive operations; returns binary responses.

1.2 Physical/Link Layer (HCCA)

The protocol is transported via the **HCCA** port handshake implemented in NABU-LIB.c (hcca_writeByte, hcca_readByte) and, for high-throughput transfers, “interruptless” direct routines implemented in RetroNET-FileStore.c (hcca_Di*).

Key operational detail:

- Many RetroNET commands are implemented in a “transaction” style:
  Write command + arguments → read response bytes immediately.
- Some commands stream payload bytes directly from the IA as fast as possible (busy-waiting the AY status bits and reading IO_HCCA).

1.3 Two execution modes

RetroNET uses two patterns:

1. **Buffered/interrupt-driven HCCA** (hcca_writeByte, hcca_readByte, hcca_readBytes)
    - Uses an RX ring-buffer filled by an interrupt handler.
    - Simpler, good for small replies.
2. **Interruptless / Direct I/O (“Di”)** (hcca_DiFocusInterrupts, hcca_DiWriteByte, hcca_DiReadByte, etc.)
    - Disables interrupts and “focuses” on HCCA RX/TX for performance (large reads like file blocks).
    - You’ll see this used for file reads and some queries where speed matters.

Recommendation:
Use **Di mode** for bulk reads/writes and variable-length streaming reads. Use buffered mode for small control ops (close handle, printer, etc.).

---

2) Data Types and Encoding Rules

2.1 Integer endianness

All multi-byte integers are **little-endian**.

- uint16_t: low byte, then high byte
- uint32_t / int32_t: least-significant byte first

2.2 Strings

Most commands that accept a string use:

- **Length-prefixed string**: uint8_t len followed by len bytes (not NUL-terminated).
- Some helper routines exist that write NUL-terminated strings, but RetroNET command payloads in these files primarily use length-prefixed strings.

2.3 Filenames and casing

All files stored on the IA Server will be in UPPERCASE.

That means:

- The IA may normalize to uppercase.
- For portability, **uppercase filenames on the NABU side** before sending (especially if you're implementing your own client without the library helpers).

2.4 Return values and sentinel values

Where file size or handle size is returned:

- -1 is commonly used for "not available / not downloaded / directory" depending on context.
- -2 is used by the IA for **"does not exist"**

---

3) Command Format

RetroNET commands are generally:

REQUEST:

[CMD: uint8]

[ARGS...]

RESPONSE:

[Fixed fields] or [Length + payload] or [Streaming bytes]

For performance, there is **no explicit framing length** at the transport level; the client must know exactly how many bytes to read based on the command's defined response format. The performance of RetroNET is due to this approach.

---

4) RetroNET File Store Commands

These commands provide file and directory operations through the IA's RetroNET storage.

4.1 Constants (flags)

Open flags (fileFlag):

- OPEN_FILE_FLAG_READONLY = 0b00000000
- OPEN_FILE_FLAG_READWRITE = 0b00000001

Copy/Move flags:

- COPY_MOVE_FLAG_NO_REPLACE = 0b00000000
- COPY_MOVE_FLAG_YES_REPLACE = 0b00000001

List flags (bitmask):

- FILE_LIST_FLAG_INCLUDE_FILES = 0b00000001
- FILE_LIST_FLAG_INCLUDE_DIRECTORIES = 0b00000010

Seek origin:

- RN_SEEK_SET = 1
- RN_SEEK_CUR = 2
- RN_SEEK_END = 3

4.2 FileDetailsStruct response layout (83 bytes)

Used by:

- 0xB2 (list item)
- 0xB3 (details by filename)
- 0xB4 (details by handle)

Binary layout:

Offset	Size	Type	Name	Notes
0	4	int32	FileSize	>=0 file; -1 folder; -2 not found
4	2	uint16	CreatedYear	
6	1	uint8	CreatedMonth	
7	1	uint8	CreatedDay	
8	1	uint8	CreatedHour	24-hour
9	1	uint8	CreatedMinute	
10	1	uint8	CreatedSecond	
11	2	uint16	ModifiedYear	
13	1	uint8	ModifiedMonth	
14	1	uint8	ModifiedDay	
15	1	uint8	ModifiedHour	24-hour
16	1	uint8	ModifiedMinute	
17	1	uint8	ModifiedSecond	
18	1	uint8	FilenameLen	max 64
19	64	bytes	Filename	may contain unused trailing bytes

Derived booleans used by the library:

- IsFile = (FileSize >= 0)

- Exists = (FileSize != -2)

4.3 File/Directory command reference

CMD 0xA3 — Open file (get/assign handle)

Request:

0xA3

filenameLen (uint8)

filename[filenameLen]

fileFlag (uint16)

fileHandle (uint8) ; 0xFF = IA assigns one

Response:

handle (uint8)

Behavior notes:

- If the target is a local “File:” path and it doesn’t exist, IA creates a 0-byte file.
- If caller supplies a handle already in use, IA may assign a new one and return it.
- URL types supported by comment contract:
    - ftp://...
    - http://...
    - https://...
    - local file paths with optional drive and backslashes

CMD 0xA7 — Close handle

Request:

0xA7

handle (uint8)

Response: none

CMD 0xA8 — File size by filename (does not auto-create)

Request:

0xA8

filenameLen (uint8)

filename[filenameLen]

Response:

size (int32) ; -1 or -2 may indicate not present depending on IA behavior

Use this when you want existence/size without opening/creating.

CMD 0xE7 — Read by filename (no handle)

Request:

0xE7

filenameLen (uint8)

filename[filenameLen]

readOffset (uint32)

readLength (uint16)

Response:

toRead (uint16)

data[toRead] ; streamed

Notes:

- Uses direct streaming in the library for speed.
- Returns 0 when EOF reached or error.

CMD 0xE8 — Replace bytes by filename (no handle)

Request:

0xE8

filenameLen (uint8)

filename[filenameLen]

fileOffset (uint32)

dataLen (uint16)

data[dataLen]

Response: none (library does not read one)

Notes:

- Intended for overwrite-in-place (not insert/append).
- For HTTP/HTTPS/FTP sources: comment contract says writes are non-persistent for remote URLs.

CMD 0xA4 — Size by handle

Request:

0xA4

handle (uint8)

Response:

size (int32)

Notes:

- For local files opened by handle, size reflects current stored data.
- For URL handles, may be -1 until downloaded/realized on IA.

CMD 0xA5 — Read by handle (random access)

Request:

0xA5

handle (uint8)

readOffset (uint32)

readLength (uint16)

Response:

toRead (uint16)

data[toRead] ; streamed

CMD 0xA9 — Append

Request:

0xA9

handle (uint8)

dataLen (uint16)

data[dataLen]

Response: none

CMD 0xAA — Insert (shifts subsequent bytes)

Request:

0xAA

handle (uint8)

fileOffset (uint32)

dataLen (uint16)

data[dataLen]

Response: none

CMD 0xAB — Delete range

Request:

0xAB

handle (uint8)

fileOffset (uint32)

deleteLen (uint16)

Response: none

CMD 0xB0 — Empty file (truncate to 0)

Request:

0xB0

handle (uint8)

Response: none

CMD 0xAC — Replace bytes by handle (overwrite-in-place)

Request:

0xAC

handle (uint8)

fileOffset (uint32)

dataLen (uint16)

data[dataLen]

Response: none

CMD 0xAD — Delete physical file

Request:

0xAD

filenameLen (uint8)

filename[filenameLen]

Response: none

Notes:

- If file is open via a handle, IA will close it.

CMD 0xAE — Copy

Request:

0xAE

srcLen (uint8)

src[srcLen]

dstLen (uint8)

dst[dstLen]

copyMoveFlag (uint8) ; replace? see flags above

Response: none

CMD 0xAF — Move

Request:

0xAF

srcLen (uint8)

src[srcLen]

dstLen (uint8)

dst[dstLen]

copyMoveFlag (uint8)

Response: none

CMD 0xB1 — List directory

Request:

0xB1

pathLen (uint8)

path[pathLen]

wildcardLen (uint8)

wildcard[wildcardLen]

fileListFlags (uint8) ; bitmask include files/dirs

Response:

count (uint16)

Follow-up: call 0xB2 for each item index 0..count-1.

CMD 0xB2 — List item details (by index)

Request:

0xB2

fileItemIndex (uint16)

Response:

- FileDetailsStruct (83 bytes layout above)

CMD 0xB3 — File details by filename

Request:

0xB3

filenameLen (uint8)

filename[filenameLen]

Response:

- FileDetailsStruct (83 bytes)

CMD 0xB4 — File details by handle

Request:

0xB4

handle (uint8)

Response:

- FileDetailsStruct (83 bytes)

CMD 0xB5 — Sequential read

Request:

0xB5

handle (uint8)

readLength (uint16)

Response:

toRead (uint16)

data[toRead] ; streamed

Notes:

- Reads from an internal file pointer maintained by IA.
- Combine with 0xB6 seek to reposition.

CMD 0xB6 — Seek (for sequential read pointer)

Request:

0xB6

handle (uint8)

offset (int32)

seekOption (uint8) ; RN_SEEK_SET/CUR/END

Response:

newPosition (int32)

Notes:

- IA clamps to [0, fileSize]; returned position reflects the actual pointer after clamping.

CMD 0xDC — Count lines in text file

Request:

0xDC

handle (uint8)

Response:

lineCount (uint16)

CMD 0xDD — Get specific line as text

Request:

0xDD

handle (uint8)

lineNumber (uint16)

Response:

toRead (uint16)

lineBytes[toRead] ; streamed

Notes:

- Line splitting uses IA/OS newline convention.
- Provide a sufficiently large buffer on the NABU side.

---

5) RetroNET TCP Client Commands

These provide a simple TCP socket abstraction via “handles”.

CMD 0xD0 — Open TCP connection

Request:

0xD0

hostnameLen (uint8)

hostname[hostnameLen]

port (uint16)

handle (uint8) ; 0xFF = IA assigns one

Response:

handle (uint8) ; 0xFF indicates failure to connect

CMD 0xD1 — Close TCP handle

Request:

0xD1

handle (uint8)

Response: none

CMD 0xD2 — Bytes available to read (TCP handle)

Request:

0xD2

handle (uint8)

Response:

available (int32) ; -1 indicates disconnected/no connection

CMD 0xD3 — Read from TCP handle

Request:

0xD3

handle (uint8)

readLength (uint16)

Response:

toRead (int32) ; -1 indicates disconnected

data[toRead] ; streamed if toRead > 0

CMD 0xD4 — Write to TCP handle

Request:

0xD4

handle (uint8)

dataLen (uint16)

data[dataLen]

Response:

wrote (int32) ; -1 indicates disconnected

---

6) RetroNET TCP Server Commands (IA-hosted server)

These talk to a TCP server implemented inside the IA, configured via IA settings.

CMD 0xD5 — Connected client count

Request:

0xD5

Response:

clientCount (uint8)

CMD 0xD6 — Bytes available to read (server)

Request:

0xD6

Response:

available (uint8)

CMD 0xD7 — Read from TCP server channel

Request:

0xD7

readLength (uint8)

Response:

toRead (uint8)

data[toRead] ; streamed if toRead > 0

CMD 0xD8 — Write to TCP server channel

Request:

0xD8

dataLen (uint8)

data[dataLen]

Response: none

---

7) Printer / Punch-Out Output

These are “character output” helpers that forward bytes to host-side sinks.

CMD 0xDA — Printer output (one byte)

Request:

0xDA

c (uint8)

Response: none

CMD 0xDB — Punch-out output (one byte)

Request:

0xDB

c (uint8)

Response: none

---

8) CP/M Drive Manager Commands

These control IA features related to building/extracting CP/M drives.

CMD 0xDE — Build drive

Request:

0xDE

driveLetter (uint8) ; 0=A … 15=P

Response:

len (uint8)

message[len] ; read into 128-byte buffer per library contract

---

CMD 0xDF — Extract drive

Request:

0xDF

driveLetter (uint8) ; 0=A … 15=P

Response:

len (uint8)

message[len]

Design note from code:
These return a string message (not a structured status).

---

9) IA Control Command Group

IA Control commands share a common top-level command byte:

CMD 0xBA — IA Control multiplexer

Request header:

0xBA

subcommand (uint8)

[subcommand args...]

Response: depends on subcommand (often length + data)

9.1 Channel / Parent / Child browsing

Sub 0x00 — Get parent count

Request:

0xBA 0x00

Response:

count (uint8)

Sub 0x01 — Get parent name

Request:

0xBA 0x01

parentId (uint8)

Response:

len (uint8)

name[len]

Buffer guidance: 64 bytes.

Sub 0x0D — Get parent description

Request:

0xBA 0x0D

parentId (uint8)

Response:

len (uint8)

desc[len]

Buffer guidance: library clears 255 bytes (so plan for up to 255).

Sub 0x18 — Get child count (extended, uint16)

Request:

0xBA 0x18

parentId (uint8)

Response:

count (uint16)

Sub 0x19 — Get child name (extended)

Request:

0xBA 0x19

parentId (uint8)

childId (uint16)

Response:

len (uint8)

name[len]

Sub 0x1A — Set selection (extended)

Request:

0xBA 0x1A

parentId (uint8)

childId (uint16)

Response: none

Sub 0x1B — Get child description (extended)

Request:

0xBA 0x1B

parentId (uint8)

childId (uint16)

Response:

len (uint8)

desc[len]

Buffer guidance: 256 bytes.

Sub 0x1C — Get child author (extended)

Request:

0xBA 0x1C

parentId (uint8)

childId (uint16)

Response:

len (uint8)

author[len]

Buffer guidance: 16 bytes.

Sub 0x1D — Get child icon tile color (32 bytes)

Request:

0xBA 0x1D

parentId (uint8)

childId (uint16)

Response:

color[32]

Sub 0x1E — Get child icon tile pattern (32 bytes)

Request:

0xBA 0x1E

parentId (uint8)

childId (uint16)

Response:

pattern[32]

9.2 News and logs

Sub 0x07 — Get current news content

Request:

0xBA 0x07

Response:

len (uint16)

content[len]

Buffer guidance: 512 bytes.

Sub 0x0A — Get IA log tail

Request:

0xBA 0x0A

Response:

len (uint16)

log[len]

Buffer guidance: 512 bytes.

Sub 0x0B — Get news title

Request:

0xBA 0x0B

Response:

len (uint8)

title[len]

Buffer guidance: 64 bytes.

Sub 0x0C — Get news date

Request:

0xBA 0x0C

Response:

len (uint8)

date[len]

Buffer guidance: 20 bytes.

Sub 0x0E — Get news count

Request:

0xBA 0x0E

Response:

count (uint8)

Sub 0x0F — Get news content by id

Request:

0xBA 0x0F

id (uint8)

Response:

len (uint16)

content[len]

Sub 0x10 — Get news title by id

Request:

0xBA 0x10

id (uint8)

Response:

len (uint8)

title[len]

Sub 0x11 — Get news date by id

Request:

0xBA 0x11

id (uint8)

Response:

len (uint8)

date[len]

Sub 0x12 — Get news icon tile color by id (32 bytes)

Request:

0xBA 0x12

id (uint8)

Response:

color[32]

Sub 0x13 — Get news icon tile pattern by id (32 bytes)

Request:

0xBA 0x13

id (uint8)

Response:

pattern[32]

9.3 Adapter info / time

Sub 0x14 — Get adapter OS

Request:

0xBA 0x14

Response:

os (uint8)

Values:

- 0 Windows
- 1 MacOS
- 2 Linux
- 99 Unknown

Sub 0x15 — Get current date/time string (formatted)

Get the current date/time as a string in the specified format

Send blank dateformat with 0 bytes for a default string.

Examples:

"dddd, MMMM dd" outputs "Sunday, April 14"

"yyyy-MM-dd HH:mm:ss" outputs "2024-04-14 19:04:01"

"HH:mm:ss" outputs 19:04:01

"MM/dd/yyyy HH:mm" outputs 04/14/2024 19:04

yyyy: Four-digit year.

yy: Two-digit year.

MMMM: Full month name.

MMM: Abbreviated month name.

MM: Two-digit month (with leading zero).

M: One or two-digit month.

dddd: Full name of the day of the week.

ddd: Abbreviated name of the day.

dd: Two-digit day of the month (with leading zero).

d: One or two-digit day of the month.

HH: Two digits of hour (24-hour clock) with leading zeros.

H: One or two digits of hour (24-hour clock).

hh: Two digits of hour (12-hour clock) with leading zeros.

h: One or two digits of hour (12-hour clock).

mm: Two digits of minute with leading zeros.

m: One or two digits of minute.

ss: Two digits of second with leading zeros.

s: One or two digits of second.

fff: Three digits of milliseconds.

ff: Two digits of milliseconds.

f: One digit of milliseconds.

tt: AM or PM designator.

zzz: Time zone offset in hours and minutes.

zz: Time zone offset in hours.

z: Hours of time zone offset.

Request:

0xBA 0x15

formatLen (uint8)

format[formatLen] ; may be 0 for default

Response:

len (uint8)

dateTimeStr[len]

Buffer guidance: 64 bytes.

Sub 0x16 — Get adapter version string

Request:

0xBA 0x16

Response:

len (uint8)

version[len] ; example: "2024.04.19.00"

Buffer guidance: 14 bytes.

Sub 0x17 — Is new adapter version available?

Request:

0xBA 0x17

Response:

val (uint8) ; >0 => true

---

10) Worked Examples

10.1 Read a whole file in blocks (handle-based)

*Note: while getting the file size is one way to do this, simply reading until the read length is zero is more efficient.

Goal: OPEN → SIZE → READ in chunks → CLOSE

Pseudo-flow:

1. Open: 0xA3
2. Get size: 0xA4
3. Loop:
    - 0xA5 handle offset chunkLen → toRead + data
4. Close: 0xA7

Notes for NABU-side performance:

- Prefer chunk sizes that balance overhead vs memory (e.g., 256–2048 depending on your RAM budget).
- Use Di-mode style transfers for the read.

10.2 List directory entries

1. 0xB1 with:
    - pathLen + path
    - wildcardLen + wildcard (e.g., "*.TXT")
    - flags: include files/dirs
2. Read count (uint16)
3. For i=0..count-1: call 0xB2 with i and parse the 83-byte details.

10.3 Simple TCP client request/response

1. 0xD0 connect to "cloud.nabu.ca" port 1234 handle 0xFF
2. If returned handle != 0xFF:
    - write: 0xD4 handle len data...
    - read available: 0xD2 handle → available
    - read: 0xD3 handle desiredLen → toRead + data
3. close: 0xD1 handle

---

11) Implementation Guidance and “Gotchas”

11.1 Always read exactly what the command returns

There’s no envelope framing. If you under-read, your next command will “see” leftover bytes; if you over-read, you’ll block forever.

11.2 Use length-prefixed reads defensively

For responses that begin with a length (uint8 or uint16):

- Clamp to your buffer size.
- If the IA returns a length larger than expected, **read and discard the extra** (or stream into a ring buffer) to keep the link aligned.

11.3 Close handles (files and TCP)

The library notes that IA closes everything on an IA “INIT” event if the NABU resets, but you should still close:

- File handles via 0xA7
- TCP handles via 0xD1

11.4 Uppercase filenames

To avoid subtle mismatches across IA OS/filesystems:

- Convert filenames and wildcards to uppercase before sending.

11.5 Prefer sequential read for streaming

If you’re reading large files linearly:

- Use 0xB5 sequential reads + 0xB6 seek only when needed.
  It reduces the need to resend offsets each time and can simplify your client logic.

11.6 Distinguish “not found” vs “directory”

When parsing FileDetailsStruct.FileSize:

- -2: not found (Exists false)
- -1: directory (Exists true, IsFile false)
- >=0: file (Exists true, IsFile true)