LTP Serial Protocol v2 Specification

Overview

This document specifies a bidirectional serial protocol for communication between an LTP Serial Sink (host) and a microcontroller (MCU) that drives LED strips. The protocol is designed to be:

- **Simple** - Easy to implement on resource-constrained microcontrollers (Arduino Uno, Nano)
- **Efficient** - Binary format for pixel data to maximize throughput
- **Reliable** - Checksums and acknowledgments for error detection
- **Bidirectional** - MCU can report status, geometry, and errors back to host
- **Extensible** - Version negotiation and reserved command space for future features

Design Goals

1. **Arduino Uno Compatible** - Must work within 2KB RAM, 32KB flash constraints
2. **High Throughput** - Support 300+ pixels at 30+ FPS on 115200 baud, 1000+ on USB
3. **Low Latency** - Minimal protocol overhead for real-time applications
4. **Self-Describing** - MCU reports its capabilities, geometry, and configuration
5. **Robust** - Graceful handling of transmission errors and buffer overflows
6. **Tear-Free Display** - Explicit separation of data transfer and display update

Protocol Versions

Version	Status	Description
1.0	Current	Text-based unidirectional protocol
2.0	This Spec	Binary bidirectional protocol

Physical Layer

Transport Types

The protocol supports multiple physical transports:

Transport	Typical Speed	Notes
Hardware UART	9600-921600 baud	Traditional serial, speed-limited
USB CDC (Virtual COM)	1-12 Mbps	Baud rate setting often ignored
USB Native	12-480 Mbps	Full USB speed, no baud limits
Bluetooth Serial	115200-921600	Wireless, higher latency

USB CDC Note: Many Arduino boards (Leonardo, Micro, Pro Micro, Teensy, ESP32-S2/S3) use native USB. The "baud rate" setting is typically ignored - data transfers at full USB speed (1+ Mbps effective for CDC). The host should detect USB devices and not artificially limit throughput based on the configured baud rate.

Serial Parameters (UART Transport)

Parameter	Default	Range	Notes
Baud Rate	115200	9600-921600	Negotiable during handshake
Data Bits	8	8	Fixed
Parity	None	None	Fixed
Stop Bits	1	1	Fixed
Flow Control	None	None/RTS-CTS	Optional hardware flow control

Timing

Parameter	Value	Notes
Inter-byte timeout	10ms	Max gap between bytes in a packet
Response timeout	100ms	Max time to wait for ACK/response
Keepalive interval	5s	Optional heartbeat when idle

Throughput Expectations

Transport	60 pixels	300 pixels	1000 pixels
115200 baud	60 FPS	12 FPS	3 FPS
921600 baud	480 FPS	96 FPS	28 FPS
USB CDC	500+ FPS	200+ FPS	100+ FPS

Assumes RGB format (3 bytes/pixel), full frames, minimal overhead

---

Buffered Display Model

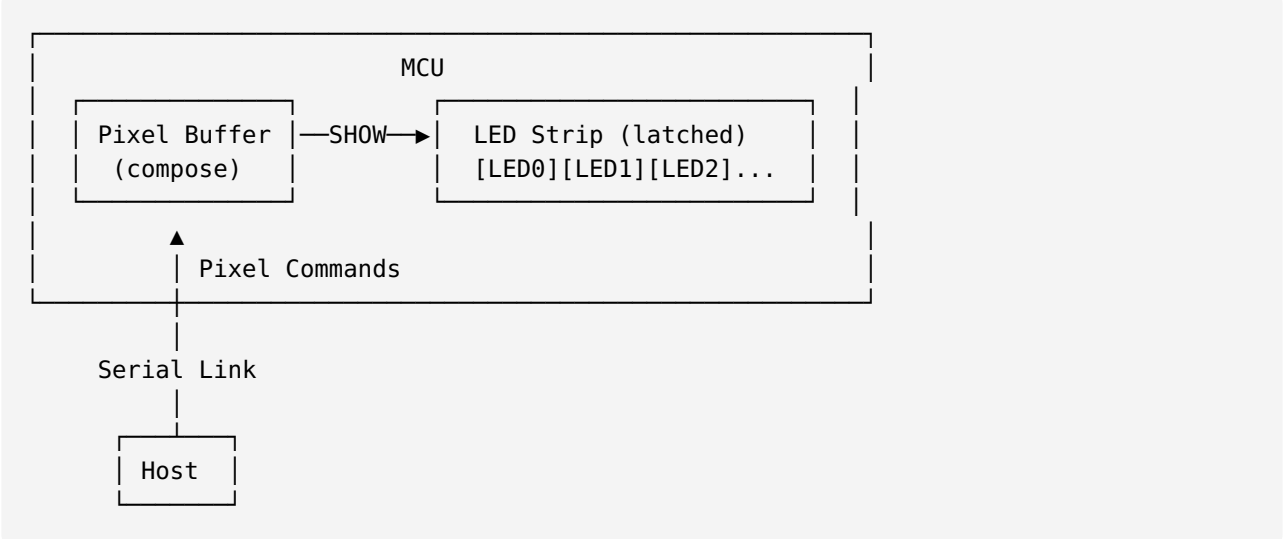
A key design principle of this protocol is the **separation of data transfer from display update**. This enables tear-free rendering regardless of the device's processing speed or the serial link bandwidth.

The Problem

Without explicit display control, the host doesn't know when the MCU will push data to the LEDs. If the MCU updates LEDs while still receiving data, observers may see partial frames ("tearing").

The Solution: Explicit SHOW Command

Modern addressable LEDs (WS2812, SK6812, APA102, etc.) have internal latches - each LED chip holds its color value until new data is shifted in. The "display buffer" is effectively distributed across the LED chips themselves, not in MCU RAM.



Flow: 1. Host sends pixel data (PIXEL_FRAME, PIXEL_SET_RANGE, etc.) 2. MCU writes data to its **pixel buffer** (LEDs unchanged) 3. Host sends SHOW command when frame is complete 4. MCU shifts data out to LED strip 5. LEDs latch new values atomically 6. Observer sees complete frame, no tearing

Key insight: The MCU only needs a single pixel buffer. The LEDs themselves provide the "front buffer" by latching and holding their values until the next SHOW.

Memory Considerations

Single buffer per strip:

Pixels	RGB (3 bytes)	RGBW (4 bytes)
60	180 bytes	240 bytes
120	360 bytes	480 bytes
300	900 bytes	1200 bytes
500	1500 bytes	2000 bytes

For Arduino Uno (2KB RAM), practical limits: - ~500 RGB pixels with minimal overhead - ~400 RGBW pixels

Auto-Show Mode

Devices may support an AUTO_SHOW configuration option: - When enabled, display updates automatically after each complete PIXEL_FRAME - Useful for simple setups where the host sends complete frames each time - SHOW command is still accepted but becomes a no-op

---

Packet Format

All communication uses a consistent packet structure:

```
+-------+-------+--------+------+---------+----------+
| START | FLAGS | LENGTH | CMD  | PAYLOAD | CHECKSUM |
+-------+-------+--------+------+---------+----------+
| 1     | 1     | 2      | 1    | 0-1024  | 1        |
+-------+-------+--------+------+---------+----------+
```

Fields

Field	Size	Description
START	1 byte	Sync byte: `0xAA`
FLAGS	1 byte	Packet flags (see below)
LENGTH	2 bytes	Payload length (little-endian, 0-1024)
CMD	1 byte	Command code
PAYLOAD	0-1024 bytes	Command-specific data
CHECKSUM	1 byte	XOR of all bytes from FLAGS to end of PAYLOAD

FLAGS Byte

```
Bit 7: Reserved (0)
Bit 6: Reserved (0)
Bit 5: Reserved (0)
Bit 4: COMPRESSED - Payload uses RLE compression
Bit 3: CONTINUED - More packets follow (fragmentation)
Bit 2: RESPONSE - This is a response packet
Bit 1: ACK_REQ - Request acknowledgment
Bit 0: ERROR - Error response (RESPONSE must also be set)
```

Checksum Calculation

Simple XOR checksum for minimal MCU overhead:

```
uint8_t checksum = 0;
for (int i = 1; i < packet_length - 1; i++) {  // Skip START, include up to PAYLOAD
    checksum ^= packet[i];
}
```

Command Reference

Command Categories

Range	Category	Direction
0x00-0x0F	System/Control	Both

Range	Category	Direction
0x10-0x1F	Query/Info	Host → MCU
0x20-0x2F	Query Response	MCU → Host
0x30-0x3F	Pixel Data	Host → MCU
0x40-0x4F	Configuration	Host → MCU
0x50-0x5F	Events/Status	MCU → Host
0xF0-0xFF	Reserved	-

System Commands (0x00-0x0F)

0x00 NOP (No Operation)

Keepalive/ping packet. No payload.

Request: `AA 02 0000 00 02` **Response:** ACK if ACK_REQ set

0x01 RESET

Request MCU to reset/reinitialize. No payload.

Request: `AA 02 0000 01 03` **Response:** MCU resets, then sends HELLO

0x02 ACK (Acknowledgment)

Positive acknowledgment. Payload contains sequence number if tracking.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Acknowledged command code | | 1 | 1 | Sequence number (optional) |

0x03 NAK (Negative Acknowledgment)

Error response. Payload contains error code.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Failed command code | | 1 | 1 | Error code (see Error Codes) | | 2 | n | Optional error message (ASCII) |

0x04 HELLO

MCU announces itself after power-on or reset. Sent automatically.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Protocol version major | | 1 | 1 | Protocol version minor | | 2 | 2 | Firmware version (major.minor as BCD) | | 4 | 1 | Strip count (1-16) | | 5 | 2 | Total pixel count across all strips (little-endian) | | 7 | 1 | Default color format (see Color Formats) | | 8 | 1 | Capabilities flags byte 1 | | 9 | 1 | Capabilities flags byte 2 (if CAPS_EXTENDED set) | | 10 | 1 | Control count (number of advertised controls) | | 11 | 1 | Input count (number of advertised inputs) |

Capabilities Flags (Byte 1):

```
Bit 0: CAPS_BRIGHTNESS - Supports brightness control
Bit 1: CAPS_GAMMA - Supports gamma correction
Bit 2: CAPS_RLE - Supports RLE compression
Bit 3: CAPS_FLOW_CTRL - Supports hardware flow control
Bit 4: CAPS_TEMP_SENSOR - Has temperature sensor
Bit 5: CAPS_VOLT_SENSOR - Has voltage sensor
Bit 6: CAPS_SEGMENTS - Supports segment addressing
Bit 7: CAPS_EXTENDED - Extended capabilities in byte 2
```

Extended Capabilities Flags (Byte 2, optional if CAPS_EXTENDED set):

```
Bit 0: CAPS_FRAME_ACK - Can send frame acknowledgments
Bit 1: CAPS_PIXEL_READBACK - Supports reading back pixel values
Bit 2: CAPS_EEPROM - Has persistent configuration storage
Bit 3: CAPS_USB_HIGHSPEED - Native USB (ignore baud rate)
Bit 4: CAPS_MULTI_STRIP - Supports multiple LED strip outputs
Bit 5: CAPS_INPUTS - Has input devices (buttons, encoders, etc.)
Bit 6: Reserved
Bit 7: Reserved
```

Example: MCU with 60 RGB pixels, protocol 2.0, firmware 1.5, USB high-speed

```
AA 04 0009 04 02 00 15 00 3C 00 03 87 08 [checksum]
                                   ^^ ^^ caps byte 1 (0x87 = extended flag set)
                                      ^^ caps byte 2 (0x08 = USB high-speed)
```

0x05 SHOW

Trigger display update - shift pixel buffer to LED strip.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 2 | Frame number (optional, for tracking/acknowledgment) |

Behavior: - MCU shifts its pixel buffer out to the LED strip - LEDs latch the new values - If FRAME_ACK is enabled, MCU sends FRAME_ACK response

Usage: - Send all pixel data for a frame (PIXEL_FRAME, PIXEL_SET_RANGE, etc.) - Send SHOW to make changes visible - This ensures tear-free updates even on slow serial links

---

Query Commands (0x10-0x1F)

0x10 GET_INFO

Request device information.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Info type (see below) |

Info Types: | Value | Type | Response | |-------|------|----------| | 0x00 | All | Full device info | | 0x01 | Version | Protocol and firmware versions | | 0x02 | Strips | Strip configuration | | 0x03 | Status | Current state, temperature, voltage | | 0x04 | Controls | Advertised control definitions | | 0x05 | Stats | Frame count, error count, uptime | | 0x06 | Inputs | Advertised input definitions |

0x11 GET_PIXELS

Request current pixel values (for debugging/verification).

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID | | 1 | 2 | Start index (little-endian) | | 3 | 2 | Count (little-endian, 0 = all) |

Response: PIXEL_RESPONSE with requested range

0x12 GET_CONTROL

Request current value of a control.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Control ID |

Response: CONTROL_RESPONSE with control value

0x13 GET_STRIP

Request configuration of a specific strip.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID (0-15, or 0xFF for all) |

Response: STRIP_RESPONSE with strip configuration

0x14 GET_INPUT

Request current state of an input.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Input ID (0-255, or 0xFF for all) |

Response: INPUT_RESPONSE with input state

Query Response Commands (0x20-0x2F)

0x20 INFO_RESPONSE

Response to GET_INFO.

Payload varies by info type requested:

Type 0x00 (All): | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Protocol version major | | 1 | 1 | Protocol version minor | | 2 | 2 | Firmware version (BCD) | | 4 | 1 | Strip count | | 5 | 2 | Total pixel count | | 7 | 1 | Default color format | | 8 | 1 | Capabilities flags byte 1 | | 9 | 1 | Capabilities flags byte 2 | | 10 | 1 | Control count | | 11 | 16 | Device name (ASCII, null-terminated) |

Type 0x02 (Strips): | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip count | | 1 | n | Strip definitions (see Strip Definition below) |

Strip Definition (8 bytes per strip): | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID | | 1 | 2 | Pixel count | | 3 | 1 | Color format | | 4 | 1 | LED type (0=WS2812, 1=SK6812, 2=APA102, etc.) | | 5 | 1 | Data pin | | 6 | 1 | Clock pin (0 if N/A) | | 7 | 1 | Flags (bit 0: reversed) |

Type 0x03 (Status): | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | State (0=idle, 1=running, 2=error) | | 1 | 1 | Current brightness (0-255) | | 2 | 2 | Temperature (°C × 10, signed, 0x7FFF = N/A) | | 4 | 2 | Voltage (mV, 0xFFFF = N/A) | | 6 | 1 | Error code (0 = no error) |

Type 0x04 (Controls): | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Control count | | 1 | n | Control definitions (see Control Definition below) |

Type 0x05 (Stats): | Offset | Size | Description | |--------|------|-------------| | 0 | 4 | Frames received (little-endian) | | 4 | 4 | Frames displayed | | 8 | 4 | Bytes received | | 12 | 2 | Checksum errors | | 14 | 2 | Buffer overflows | | 16 | 4 | Uptime (seconds) |

Type 0x06 (Inputs): | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Input count | | 1 | n | Input definitions (see Input Definition below) |

0x21 PIXEL_RESPONSE

Response to GET_PIXELS.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID | | 1 | 2 | Start index | | 3 | 2 | Pixel count | | 5 | n | Pixel data (3 or 4 bytes per pixel) |

0x22 CONTROL_RESPONSE

Response to GET_CONTROL.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Control ID | | 1 | n | Control value (type depends on control) |

0x23 STRIP_RESPONSE

Response to GET_STRIP.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip count in response | | 1 | n | Strip definitions (8 bytes each, see Strip Definition above) |

0x24 CONTROLS_LIST

Full list of advertised controls.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Control count | | 1 | n | Control definitions |

Control Definition (variable length): | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Control ID (0-255) | | 1 | 1 | Control type (see Control Types) | | 2 | 1 | Flags (bit 0: read-only, bit 1: persistent) | | 3 | 1 | Name length | | 4 | n | Name (ASCII, not null-terminated) | | 4+n | m | Type-specific data (see below) |

Control Types:

Value	Type	Type-Specific Data
0x01	BOOL	1 byte: current value (0/1)
0x02	UINT8	3 bytes: current, min, max
0x03	UINT16	6 bytes: current, min, max (little-endian)
0x04	INT8	3 bytes: current, min, max (signed)
0x05	INT16	6 bytes: current, min, max (signed, little-endian)

Value	Type	Type-Specific Data
0x06	ENUM	1 byte: current + 1 byte: option count + n bytes: option strings (length-prefixed)
0x07	STRING	1 byte: max length + 1 byte: current length + n bytes: current value
0x08	COLOR	3 bytes: current RGB value
0x09	ACTION	0 bytes (trigger-only, no value)

Example Controls:

A device might advertise these controls:

```
Control 0: "Brightness" (UINT8, 0-255, current=200)
Control 1: "Gamma" (UINT8, 10-30, current=22)  // 10 = 1.0, 22 = 2.2
Control 2: "Idle Timeout" (UINT16, 0-3600, current=300)  // seconds
Control 3: "Test Mode" (ENUM: "Off", "Rainbow", "White", "Red")
Control 4: "Save Settings" (ACTION)
Control 5: "Auto Brightness" (BOOL, current=false)
```

0x25 INPUT_RESPONSE

Response to GET_INPUT.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Input count in response | | 1 | n | Input states (variable length per input type) |

Input State Entry: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Input ID | | 1 | 1 | Input type | | 2 | n | Current value (type-dependent, see Input Types) |

0x26 INPUTS_LIST

Full list of advertised inputs.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Input count | | 1 | n | Input definitions |

Input Definition (variable length): | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Input ID (0-255) | | 1 | 1 | Input type (see Input Types) | | 2 | 1 | Flags (bit 0: inverted, bit 1: debounced, bit 2: pull-up enabled) | | 3 | 1 | Name length | | 4 | n | Name (ASCII, not null-terminated) | | 4+n | m | Type-specific configuration (see below) |

Input Types:

Value	Type	Config Data	Event Data
0x01	BUTTON	1 byte: GPIO pin	1 byte: state (0=released, 1=pressed)
0x02	ENCODER	2 bytes: pin A, pin B	1 byte: delta (signed, clicks since last report)
0x03	ENCODER_BTN	3 bytes: pin A, pin B, btn pin	2 bytes: delta (signed) + state
0x04	ANALOG	1 byte: ADC channel + 2 bytes: threshold	2 bytes: value (0-65535)
0x05	TOUCH	1 byte: touch channel + 2 bytes: threshold	1 byte: state (0=untouched, 1=touched)
0x06	SWITCH	1 byte: GPIO pin	1 byte: state (0=off, 1=on)
0x07	MULTI_BUTTON	1 byte: button count + n bytes: pins	1 byte: bitmask of pressed buttons

Example Inputs:

A device might advertise these inputs:

```
Input 0: "Play/Pause" (BUTTON, pin 2, debounced)
Input 1: "Volume" (ENCODER, pins 3/4)
Input 2: "Mode Select" (ENCODER_BTN, pins 5/6/7)
Input 3: "Light Sensor" (ANALOG, ADC0, threshold=100)
Input 4: "Touch Pad" (TOUCH, channel 0, threshold=50)
```

Pixel Data Commands (0x30-0x3F)

Range Convention: All ranges in this protocol use **exclusive end indices**, like Python's `range()`. A range of `[start, end)` includes `start` but excludes `end`. For example, `start=0, end=30` addresses pixels 0-29 (30 pixels total).

Strip Addressing: All pixel commands include a strip ID. Use `0xFF` to address all strips simultaneously (for commands that support it).

0x30 PIXEL_SET_ALL

Set all pixels on a strip (or all strips) to the same color.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID (0-15, or 0xFF for all strips) | | 1 | 3-4 | Color value (RGB or RGBW) |

Example: Set all pixels on all strips to red

```
AA 00 0004 30 FF FF 00 00 [checksum]
              ^^ strip 0xFF = all strips
```

0x31 PIXEL_SET_RANGE

Set a range of pixels on a strip to the same color.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID (0-15) | | 1 | 2 | Start index (little-endian) | | 3 | 2 | End index (exclusive, little-endian) | | 5 | 3-4 | Color value |

Example: Set pixels 0-29 on strip 0 to green (start=0, end=30)

```
AA 00 0008 31 00 00 00 1E 00 00 FF 00 [checksum]
              ^^ strip 0
```

0x32 PIXEL_SET_INDEXED

Set specific pixels by index with individual colors on a strip.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID (0-15) | | 1 | 2 | Pixel count | | 3 | n | Pixel entries: 2-byte index + 3/4-byte color |

Example: On strip 0, set pixel 5 to red, pixel 10 to blue

```
AA 00 000D 32 00 02 00 05 00 FF 00 00 0A 00 00 00 FF [checksum]
              ^^ strip 0
```

0x33 PIXEL_FRAME

Full frame of sequential pixel data for a strip (most efficient for animations).

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID (0-15) | | 1 | 2 | Start index (usually 0) | | 3 | 2 | Pixel count | | 5 | n | Raw pixel data (3 or 4 bytes per pixel) |

Note: For strips > ~340 RGB pixels, use multiple PIXEL_FRAME packets with CONTINUED flag, followed by SHOW.

0x34 PIXEL_FRAME_RLE

RLE-compressed frame data for a strip (efficient for patterns with repeated colors).

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID (0-15) | | 1 | 2 | Start index | | 3 | 2 | Total pixel count (uncompressed) | | 5 | n | RLE-encoded data |

RLE Encoding:

```
Each run: [count] [R] [G] [B] (or [R] [G] [B] [W] for RGBW)
- count = 1-255 pixels of the same color
- count = 0 indicates end of data (optional)
```

Example: Strip 0: 30 red pixels followed by 30 blue pixels

```
AA 04 000B 34 00 00 00 3C 00 1E FF 00 00 1E 00 00 FF [checksum]
              ^^ strip 0
```

0x35 PIXEL_DELTA

Delta update - only changed pixels since last frame on a strip.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID (0-15) | | 1 | 2 | Change count | | 3 | n | Changes: 2-byte index + 3/4-byte color |

Usage: Host tracks changes and sends only modified pixels. More efficient than full frames for subtle animations.

Configuration Commands (0x40-0x4F)

0x40 SET_CONTROL

Set an advertised control value.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Control ID | | 1 | n | Control value (type and size depend on control) |

Response: ACK on success, NAK with error code on failure.

Example: Set brightness (control 0) to 200

```
AA 00 0002 40 00 C8 [checksum]
              ^^ ^^ control 0, value 200
```

Example: Trigger "Save Settings" action (control 4)

```
AA 00 0001 40 04 [checksum]
              ^^ control 4 (ACTION type, no value needed)
```

0x41 SET_STRIP

Configure a strip's parameters.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Strip ID | | 1 | 2 | Pixel count | | 3 | 1 | Color format | | 4 | 1 | LED type | | 5 | 1 | Data pin | | 6 | 1 | Clock pin (0 if N/A) | | 7 | 1 | Flags |

Response: ACK on success, NAK on failure.

Standard Controls

Devices SHOULD advertise these standard controls when applicable:

ID	Name	Type	Description
0	Brightness	UINT8 (0-255)	Global brightness
1	Gamma	UINT8 (10-30)	Gamma × 10 (22 = 2.2)
2	Idle Timeout	UINT16 (0-65535)	Seconds before auto-off
3	Auto Show	BOOL	Auto-display after PIXEL_FRAME
4	Frame Ack	BOOL	Send FRAME_ACK after display
5	Status Interval	UINT16	Status report interval (seconds)

Device-specific controls should use IDs 16 and above.

AUTO_SHOW Behavior: - When enabled, the MCU automatically displays after receiving a complete PIXEL_FRAME - The SHOW command is still accepted but becomes a no-op - Useful for simple setups with single complete frame transmissions - Default is disabled (explicit SHOW required)

0x42 SAVE_CONFIG

Save current configuration to EEPROM/flash. No payload.

0x43 LOAD_CONFIG

Load configuration from EEPROM/flash. No payload.

0x44 RESET_CONFIG

Reset configuration to factory defaults. No payload.

0x45 SET_SEGMENT

Define a segment for addressing portions of the strip.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Segment ID (0-15) | | 1 | 2 | Start index | | 3 | 2 | Pixel count | | 5 | 1 | Flags (reverse, mirror, etc.) |

Event/Status Commands (0x50-0x5F)

0x50 STATUS_UPDATE

MCU reports status change (sent asynchronously).

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Status type | | 1 | n | Status data |

Status Types: | Value | Type | Data | |-------|------|------| | 0x01 | Ready | None - MCU ready to receive | | 0x02 | Busy | None - MCU processing | | 0x03 | Error | Error code + message | | 0x04 | Temperature | int16 (°C × 10) | | 0x05 | Voltage | uint16 (mV) | | 0x06 | Buffer | uint8 (% full) |

0x51 FRAME_ACK

MCU acknowledges frame display (optional, when enabled).

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 2 | Frame number | | 1 | 2 | Display timestamp (ms since boot, lower 16 bits) |

0x52 ERROR_EVENT

MCU reports an error condition.

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Error code | | 1 | n | Error details (optional) |

0x53 INPUT_EVENT

MCU reports an input state change (sent asynchronously when input changes).

Payload: | Offset | Size | Description | |--------|------|-------------| | 0 | 1 | Input ID | | 1 | 1 | Input type | | 2 | 2 | Timestamp (ms since boot, lower 16 bits) | | 4 | n | Event data (type-dependent, see Input Types) |

Event Data by Input Type:

Type	Event Data	Description
BUTTON	1 byte	0=released, 1=pressed
ENCODER	1 byte (signed)	Delta clicks since last event
ENCODER_BTN	2 bytes	Delta (signed) + button state
ANALOG	2 bytes	Current value (little-endian)
TOUCH	1 byte	0=untouched, 1=touched
SWITCH	1 byte	0=off, 1=on
MULTI_BUTTON	1 byte	Bitmask of currently pressed buttons

Usage Notes:

- INPUT_EVENT is sent unsolicited when input state changes
- Events include a timestamp for precise timing (e.g., measuring button hold duration)
- For ENCODER type, delta represents clicks since last report (positive = clockwise)
- For ANALOG type, events are sent when value crosses the configured threshold
- Host can also poll input state using GET_INPUT command
- Multiple rapid events may be batched in implementation (send latest state)

Example: Button 0 pressed

```
AA 00 0004 53 00 01 34 12 01 [XOR]
              ^^ ^^ input ID 0, type BUTTON
                    ^^ ^^ timestamp 0x1234
                          ^^ state: pressed
```

Example: Encoder 1 rotated 3 clicks clockwise

```
AA 00 0004 53 01 02 78 56 03 [XOR]
              ^^ ^^ input ID 1, type ENCODER
```

```
                    ^^ ^^ timestamp 0x5678
                          ^^ delta: +3
```

Error Codes

Code	Name	Description
0x00	OK	No error
0x01	CHECKSUM_ERROR	Packet checksum mismatch
0x02	INVALID_COMMAND	Unknown command code
0x03	INVALID_LENGTH	Payload length mismatch
0x04	INVALID_PARAMETER	Parameter out of range
0x05	BUFFER_OVERFLOW	Receive buffer overflow
0x06	PIXEL_OVERFLOW	Pixel index out of range
0x07	BUSY	MCU busy, try again
0x08	NOT_SUPPORTED	Feature not supported
0x09	TIMEOUT	Operation timed out
0x0A	HARDWARE_ERROR	Hardware failure
0x0B	CONFIG_ERROR	Configuration invalid

Color Formats

Value	Format	Bytes/Pixel	Description
0x03	RGB	3	Red, Green, Blue
0x04	RGBW	4	Red, Green, Blue, White
0x13	GRB	3	Green, Red, Blue (WS2812 native)
0x14	GRBW	4	Green, Red, Blue, White

Value	Format	Bytes/Pixel	Description
0x23	BGR	3	Blue, Green, Red
0x33	BRG	3	Blue, Red, Green

Communication Flow

Initialization Sequence

```
Host                              MCU
  |                                 |
  |        [Power On/Reset]         |
  |                                 |
  |<-------- HELLO ----------------|  MCU announces capabilities
  |                                 |
  |-------- GET_INFO (0x00) ------>|  Host requests full info
  |                                 |
  |<------- INFO_RESPONSE ---------|  MCU responds with details
  |                                 |
  |-------- SET_CONFIG ----------->|  Host configures as needed
  |                                 |
  |<-------- ACK -----------------|  MCU acknowledges
  |                                 |
  |-------- PIXEL_FRAME --------->|  Host sends pixel data
  |                                 |
  |-------- SHOW ---------------->|  Host triggers display
  |                                 |
```

Normal Operation

```
Host                              MCU
  |                                 |
  |-------- PIXEL_FRAME --------->|  Frame 1
  |-------- SHOW ---------------->|
  |                                 |
  |-------- PIXEL_DELTA --------->|  Frame 2 (only changes)
  |-------- SHOW ---------------->|
  |                                 |
  |-------- PIXEL_FRAME_RLE ----->|  Frame 3 (compressed)
  |-------- SHOW ---------------->|
  |                                 |
  |<------- STATUS_UPDATE --------|  MCU reports temperature
  |                                 |
```

Error Recovery

```
Host                              MCU
  |                                |
  |-------- PIXEL_FRAME --------->|  Corrupted packet
  |                                |
  |<-------- NAK (CHECKSUM) ------|  MCU reports error
  |                                |
  |-------- PIXEL_FRAME --------->|  Host retransmits
  |                                |
  |-------- SHOW ---------------->|
  |                                |
```

Implementation Guidelines

Arduino Memory Considerations

Minimum Buffer Sizes: - Receive buffer: 64 bytes (for commands) + pixel buffer - Pixel buffer: `pixel_count × bytes_per_pixel` (e.g., 60×3 = 180 bytes) - Transmit buffer: 32 bytes (for responses)

Memory Optimization: - Process packets incrementally (don't buffer entire frame) - Use PROGMEM for constant strings - Single pixel buffer (no double-buffering on small MCUs)

Example Memory Layout (Arduino Uno - 2KB RAM):

```
Receive buffer:    128 bytes
Pixel buffer:      360 bytes (120 RGB pixels)
Transmit buffer:    32 bytes
Variables:         100 bytes
Stack:             ~400 bytes
Available:       ~1000 bytes
Maximum pixels:    ~250 (with minimal buffers)
```

Recommended Baud Rates (UART)

Pixel Count	Min Baud	Recommended	Max FPS
1-60	38400	115200	60+
61-150	115200	230400	30+
151-300	230400	460800	30+

Pixel Count	Min Baud	Recommended	Max FPS
301-600	460800	921600	20+

Note: For USB CDC devices, baud rate is typically ignored. The host should detect `CAPS_USB_HIGHSPEED` and transmit at full speed.

Bandwidth Calculations

UART at 115200 baud (11,520 bytes/sec effective):

Format	60 pixels	150 pixels	300 pixels
Full frame (RGB)	186 bytes	456 bytes	906 bytes
Full frame max FPS	61 FPS	25 FPS	12 FPS
RLE (50% compression)	93 bytes	228 bytes	453 bytes
RLE max FPS	123 FPS	50 FPS	25 FPS

USB CDC (~1 Mbps effective):

Format	300 pixels	600 pixels	1000 pixels
Full frame (RGB)	906 bytes	1806 bytes	3006 bytes
Full frame max FPS	1100 FPS	550 FPS	330 FPS

Actual FPS limited by LED refresh rate (~400-800 Hz for WS2812) and MCU processing

Host Implementation Notes

1. **Adaptive Protocol Selection:**
2. Use PIXEL_FRAME for animations with many changes
3. Use PIXEL_DELTA for subtle changes
4. Use PIXEL_FRAME_RLE for patterns with solid regions
5. Use PIXEL_SET_ALL/PIXEL_SET_RANGE for fills
6. **Transport Detection:**
7. Check `CAPS_USB_HIGHSPEED` in device capabilities
8. If set, ignore configured baud rate and send at full speed
9. If not set, respect baud rate and add inter-packet delays if needed

10. **Flow Control:**
11. Monitor MCU buffer status if reported via STATUS_UPDATE
12. Implement backoff on NAK responses
13. Use hardware flow control for high-speed UART operation
14. **Error Handling:**
15. Retry on checksum errors (max 3 times)
16. Re-sync on persistent errors (send NOP, wait for ACK)
17. Log errors for diagnostics

MCU Implementation Notes

1. **Packet Parsing:**
2. State machine for byte-by-byte parsing
3. Validate checksum before processing
4. Discard incomplete packets on timeout
5. Process pixel data incrementally (write to buffer as received)
6. **Pixel Buffer:**
7. Single pixel array in MCU RAM
8. Pixel commands write to this buffer
9. Buffer is NOT pushed to LEDs until SHOW command
10. LEDs hold their previous values (internal latches) until updated
11. **Display Update (SHOW):**
12. On SHOW, shift entire pixel buffer to LED strip
13. LEDs latch new values atomically as data arrives
14. Previous LED values visible until update completes
15. This provides tear-free display without MCU double-buffering
16. **Display Timing:**
17. Buffer pixel data until SHOW received
18. If AUTO_SHOW enabled, display after complete PIXEL_FRAME
19. Track frame number for debugging and FRAME_ACK
20. **Status Reporting:**

21. Send HELLO on boot/reset
22. Report errors immediately via NAK
23. Periodic status updates if enabled (STATUS_INTERVAL config)
24. Report buffer fullness to help host pace transmissions
25. **Range Handling:**
26. All ranges use exclusive end indices
27. `start=0, end=30` means pixels 0-29
28. Validate indices against pixel count, return NAK on overflow

---

Example Packets

HELLO from MCU

```
Protocol 2.0, Firmware 1.2, 1 strip with 60 RGB pixels, extended capabilities, 3 controls, 2 input
AA 04 000B 04 02 00 12 01 3C 00 03 87 08 03 02 [XOR]
                   ^^ ^^ ^^ strip count=1, total pixels=60
                            ^^ ^^ ^^ ^^ caps1, caps2, controls, inputs
```

Set All Pixels on All Strips to Red

```
AA 00 0004 30 FF FF 00 00 [XOR]
               ^^ strip ID 0xFF = all strips
```

Set Range 0-29 Green on Strip 0 (start=0, end=30)

```
AA 00 0008 31 00 00 00 1E 00 00 FF 00 [XOR]
               ^^ strip ID 0
```

Full Frame for Strip 0 (60 RGB pixels)

```
AA 00 00B9 33 00 00 00 3C 00 [180 bytes RGB data] [XOR]
               ^^ strip ID 0
```

RLE Frame for Strip 0 (30 red + 30 blue = 60 pixels)

```
AA 04 000B 34 00 00 00 3C 00 1E FF 00 00 1E 00 00 FF [XOR]
              ^^ strip ID 0
```

Set Brightness Control to 200

```
AA 00 0002 40 00 C8 [XOR]
              ^^ ^^ control ID 0 (Brightness), value 200
```

Get Device Info

```
AA 02 0001 10 00 [XOR]
```

ACK

```
AA 04 0002 02 33 00 [XOR]
```

(Acknowledging command 0x33, sequence 0)

NAK (Checksum Error)

```
AA 05 0002 03 33 01 [XOR]
```

(NAK for command 0x33, error code 0x01)

INPUT_EVENT (Button Pressed)

```
AA 00 0005 53 00 01 34 12 01 [XOR]
              ^^ ^^ input ID 0, type BUTTON (0x01)
                    ^^ ^^ timestamp 0x1234
                          ^^ state: pressed (1)
```

(MCU reports button 0 was pressed at timestamp 0x1234)

INPUT_EVENT (Encoder Rotated)

```
AA 00 0005 53 01 02 78 56 FD [XOR]
              ^^ ^^ input ID 1, type ENCODER (0x02)
```

```
                        ^^ ^^ timestamp 0x5678
                              ^^ delta: -3 (counter-clockwise)
```

(MCU reports encoder 1 was rotated 3 clicks counter-clockwise)

Migration from Protocol v1

Compatibility Mode

The host can detect protocol version by:

1. Send HELLO request and wait for response
2. If no response within 500ms, assume v1 protocol
3. Fall back to text-based commands

Feature Mapping

v1 Feature	v2 Equivalent
`0,29=0xFF0000`	PIXEL_SET_RANGE
`30=0x00FF00`	PIXEL_SET_INDEXED (single)
Full frame	PIXEL_FRAME
Change detection	PIXEL_DELTA
Run-length encoding	PIXEL_FRAME_RLE

Future Extensions

Reserved for future versions:

- **0x60-0x6F:** Animation commands (built-in patterns)
- **0x70-0x7F:** Multi-device synchronization
- **0x80-0x8F:** Firmware update protocol
- **0x90-0x9F:** Diagnostic commands
- **0xA0-0xAF:** Custom/vendor extensions

Reference Implementation

See the following files for reference implementations:

- `src/ltp_sink_serial/protocol_v2.py` - Python host implementation
- `arduino/ltp_serial_v2/` - Arduino library implementation

Revision History

Version	Date	Changes
2.0-draft	2025-01	Initial specification
2.0-draft2	2025-01	Added USB transport info, SHOW command, extended capabilities
2.0-draft3	2025-01	Corrected buffer model (LEDs have latches, MCU needs single buffer), exclusive end indices, removed deprecated SYNC
2.0-draft4	2026-01	Added strip addressing (strip ID in all pixel commands), control advertisement system, SET_CONTROL command, renumbered configuration commands
2.0-draft5	2026-01	Added input advertisement and event system (buttons, encoders, analog, touch), INPUT_EVENT command for unsolicited input reports