YSC 8K

Usage & Protocol Reference

Driver version 1.11.0 · Generated 2026-08-01

Applies to 8K V1 (single-chip) and 8K V2 (dual-chip)

Contents

1. Using the Box

What this box does

Simply put, it is an "inline wire": your mouse/keyboard plugs into the box first, and the box plugs into the PC. The PC believes it is directly connected to your mouse/keyboard (it works exactly the same), but the box sits in between — which is what enables recoil control, rapid-fire macros, mouse curves, jitter and other adjustments in the software.

Wiring

Legacy (single-chip box): there is only one USB port — plug it straight into the PC; your mouse/keyboard goes into the port on the other side of the box.

New (dual-chip box / 8K V2): there are two USB ports, left and right — plug your real mouse/keyboard into the **right** port, then run a cable from the **left** port to the PC. Simple rule: right = real mouse, left = PC.

Once wired up, the PC will show an extra mouse plus a serial (COM) port — the driver software uses this port to send commands to the box. If either side is plugged in wrong, the mouse will not move or the PC will not recognize it.

Driver software panels

Panel	What it does
Serial Connection	Pick the box's COM port and click Connect. Commands only work after connecting.
KmNet	Control the box from another computer over the network; includes a self-test.
Monitor	Live view of mouse buttons and movement reported by the box (click "Enable Upload" first).
Macros / Jitter / Mouse Curve / Gamepad Mapping	Graphical settings for recoil control, rapid fire, feel, etc. — set and save to apply.
Docs	This document, plus a command tester (send commands and watch the replies).
Firmware Update (legacy)	Upgrades legacy single-chip boxes; no programmer needed.
8K V2 Firmware Update	Upgrades the new dual-chip box (flash left and right once each).
Debug	Packet-capture tooling for development / troubleshooting; normal users rarely need it.

First four steps

1. Wire up the box, plug it into the PC, open the driver software.
2. In "Serial Connection", select the box's COM port and click Connect (the top-right corner shows connected).
3. To confirm the link: open "Monitor" and click "Enable Upload", move the mouse and watch for data; or send a command in the Docs command tester and see whether the box replies.
4. For recoil / macros / curves, go to the matching panel, set and save; for firmware upgrades, go to the matching firmware panel (legacy: "Firmware Update"; new: "8K V2 Firmware Update").

2. Serial Communication Protocol

2.1 Basic parameters

Item	Value
Baud rate	Firmware whitelist: 115200 / 230400 / 460800 / 921600 / 1M / 1.5M / 2M / 3M; 4M is an experimental extension that hardware may not support stably. Switch dynamically via cmd:133 or km.baud.
Data bits	8
Stop bits	1
Parity	None
Flow control	No hardware flow control
DMA	One DMA channel each for USART1 TX/RX; IDLE interrupt triggers frame reception.

2.2 Frame format

```
<START>[2-byte big-endian length][JSON payload]<END>
```

YSC protocol frame: <START> (7 B) + 2-byte big-endian total length + JSON payload + <END> (5 B). MAKCU protocol: plain text km.command(args)\r\n, responses end with \r\n>>>. There is also a binary baud frame: 0xDE 0xAD 0x05 0x00 0xA5 + 4-byte little-endian baud (no text markers at all).

2.3 YSC command list

CMD_MOUSE_MOVE (30) — Smooth mouse movement

```
{"cmd":30,"x":100,"y":50,"c":1,"rx":1.0,"ry":1.0,"rc":0}
```

Smooth movement. x/y is the total target displacement, c is the number of steps (one share per step). Optional rx/ry are physical-mouse X/Y axis deceleration factors (float, default 1.0 = no deceleration); rc is the deceleration duration in milliseconds (int16, default 0 = off). The firmware converts by the current mouse polling rate (1K = 1 ms/report, 8K = 1 ms/8 reports), so rc=50 lasts 50 ms at either rate. Deceleration acts on physical mouse input rather than this command's own displacement, and is typically used for recoil control: shrink pull-down input while firing to counter recoil. The firmware uses a lightweight JSON scanner — no cJSON malloc/free — making it suitable for high-frequency calls.

CMD_MOUSE_MOVE_TOW (31) — Directional additive movement

```
{"cmd":31,"x":100,"y":50,"c":1,"rx":1.0,"ry":1.0,"rc":0}
```

Directional movement that stacks with cmd:30 (displacements add up in the HID report). Fields are identical to cmd:30. When cmd:30 and cmd:31 both set deceleration (rc>0), cmd:31 takes precedence.

CMD_MOUSE_BUTTON (32) — Button with auto-release

```
{"cmd":32,"b":1,"c":10}
```

Simulates a button press, auto-released after being reported c times. b is the button bitmask: 1 = left, 2 = right, 4 = middle.

CMD_MOUSE_BUTTON_STATE (33) — Set button state directly

```
{"cmd":33,"b":1,"s":1}
```

Sets button state directly (immediate effect, no report count). s=1 press, s=0 release. b uses the same bitmask as cmd:32.

CMD_ChangeUploadStatus (34) — Upload switch

```
{"cmd":34,"status":1}
```

Enables / disables HID data upload. status>0 enables, status=0 disables.

CMD_MOUSE_WHEEL_HOLD (35) — Hold wheel

```
{"cmd":35,"w":-1,"c":100}
```

Wheel-hold command. w is the wheel value (int8; -1 = down, 1 = up; other values are clamped to ±1); c is the duration (uint32, milliseconds). The device keeps sending the wheel value for the given duration.

CMD_MACRO_SET (36) — Configure a button macro

```
{"cmd":36,"s":0,"e":1,"t":3,"p":1,"w":-1,"i":30,"d":10,"ij":5,"dj":2}
```

Configures a macro slot. s = slot (0-7), e = enable (0/1), t = trigger mask, p = suppress mask, w = wheel value (int8), i = wheel trigger interval period in ms (optional; 0 = continuous, no interval), d = wheel-on duration per period in ms (optional, ≤ i; 0 = full period), ij = interval random jitter in ms (optional; the actual interval each period is randomized within i ± ij), dj = duration random jitter in ms (optional; the actual duration is randomized within d ± dj). Button bits: 1 = left, 2 = right, 4 = middle, 8 = side-back, 16 = side-forward. With i>0 the wheel pulses on the period beat (e.g. i=30, d=10, ij=5, dj=2 means fire roughly every 30 ms ±5, each pulse lasting about 10 ms ±2). Response {code:200,message:"macro_set",data:""}.

CMD_MACRO_GET (37) — Query button macros

```
{"cmd":37,"s":-1}
```

Queries macro configuration. s=-1 returns all slots (array); s=0..7 returns a single slot. Response fields: s/e/t/p/w/i/d/ij/dj.

CMD_MACRO_RESET (38) — Reset macros

```
{"cmd":38}
```

Resets all macro configurations to firmware defaults (all disabled). Response {code:200,message:"macro_reset"}.

CMD_JITTER_SET (39) — Jitter (recoil-track) configuration

```
{"cmd":39,"e":1,"t":1,"ax":10,"fx":0,"py":5,"fy":0}
```

Configures mouse jitter. e = enable (0/1), t = trigger (t=1 means trigger on fire / button), ax = X amplitude, fx = X frequency, py = Y amplitude, fy = Y frequency. Save/query results are returned asynchronously via debug_response (message:"jitter").

CMD_JITTER_GET (40) — Query jitter

```
{"cmd":40}
```

Queries the current jitter configuration. Response via debug_response (message:"jitter") with {e,t,ax,fx,py,fy}.

CMD_JITTER_RESET (41) — Reset jitter

```
{"cmd":41}
```

Resets the jitter configuration to defaults (all zeroed and disabled).

CMD_MOUSE_CURVE_SET (42) — Mouse curve configuration

```
{"cmd":42,"e":1,"p":2,"n":4,"d":0,"j":15}
```

Configures the mouse movement curve. e = enable (0/1), p = curve profile, n = segments, d = duration (ms; 0 = profile default), j = jitter amplitude. Save/query results are returned asynchronously via debug_response (message:"mouse_curve").

CMD_MOUSE_CURVE_GET (43) — Query mouse curve

```
{"cmd":43}
```

Queries the current mouse curve configuration. Response via debug_response (message:"mouse_curve") with {enabled,profile,segments,duration,jitter}.

CMD_MOUSE_CURVE_RESET (44) — Reset mouse curve

```
{"cmd":44}
```

Resets the mouse curve to defaults (profile=2, segments=4, duration=0, jitter=15).

CMD_JUMP_TO_IAP (50) — Enter Bootloader

```
{"cmd":50}
```

Writes the calibration word 0x5AA55AA5 to 0x0808FFFC (after erasing) and triggers a system reset. The device first replies {code:200,message:"entering_iap"}, then reboots into IAP Bootloader mode awaiting firmware upgrade.

CMD_GET_VERSION (132 / 0x84) — Query build date

```
{"cmd":132}
```

Queries the firmware build date (__DATE__ macro). Response {code:200,message:"version",data:"..."}.

CMD_SET_BAUDRATE (133 / 0x85) — Switch baud rate dynamically

```
{"cmd":133,"baud":3000000}
```

Dynamically switches the USART1 baud rate. The firmware does not validate against the whitelist and configures the hardware USART directly — so only send the 8 whitelisted rates (115200–3000000). Response {code:200,message:"switching"}; the device reconfigures USART immediately, and the host must switch baud rate in sync.

2.4 MAKCU command list

Mouse control

Command	Format	Notes
Move	km.move(x,y[,seg])	Mouse move; seg = steps 1..512 (values beyond are clamped).
Left button	km.left(1/0)	1 = press, 0 = release, 2 = silent release (skips the button-push queue).
Right button	km.right(1/0)	Same as left.
Middle button	km.middle(1/0)	Same as left.
Side button 1	km.ms1(1/0)	Same as left.
Side button 2	km.ms2(1/0)	Same as left.
Wheel	km.wheel(-1/1)	Scroll one notch; accepts ±1 only (other values are clamped).

Axis / button lock

Command	Notes
km.lock_mx(0/1) / km.lock_mx+ / km.lock_mx-	Lock the X axis (positive / negative directions can be locked separately).
km.lock_my(0/1) / km.lock_my+ / km.lock_my-	Lock the Y axis, same as above.
km.lock_ml/mr/mm/ms1/ms2(0/1)	Lock physical buttons so their input has no effect.
km.invert_x/y(0/1)	Invert the X or Y axis direction.
km.swap_xy(0/1)	Swap X/Y axis mapping.

Capture mode

Command	Notes
km.catch_ml(0/1)	Left-button capture mode: 0 = auto, 1 = manual.

Command	Notes
km.catch_mm(0/1)	Middle-button capture mode.
km.catch_mr(0/1)	Right-button capture mode.
km.catch_ms1(0/1)	Side-button 1 capture mode.
km.catch_ms2(0/1)	Side-button 2 capture mode.

System commands

Command	Notes
km.version()	Returns km.version(km.MAKCU).
km.echo(0/1)	Command echo switch (default on).
km.baud(rate)	Switch baud rate (rate >= 115200, no whitelist check); no argument returns the current baud rate.
km.buttons(mode,period)	Button-push mode: mode=0 off, mode>0 on; period = 1..1000 ms.
km.serial("xxx") / km.serial(0)	Set / reset the device serial number (max 32 bytes, escape sequences supported); no argument returns the current serial.
km.reboot()	Soft-reset the device; reboots immediately after replying km.reboot().
km.scroll_side(0/1)	When enabled, side button 2 triggers wheel events.
km.scroll_lr(0/1)	When enabled, pressing left + right / side buttons together triggers the wheel.

Binary baud-rate switch (special)

Besides text commands, the firmware supports a 9-byte binary baud-switch frame: 0xDE 0xAD 0x05 0x00 0xA5 + 4-byte little-endian baud. This frame bypasses MAKCU text parsing and reconfigures USART directly. It is commonly used for recovery when the current baud rate is unknown.

3. Command Reference

3.1 YSC protocol

Mouse movement (cmd:30)

```
{"cmd":30,"x":100,"y":50,"c":1,"rx":1.0,"ry":1.0,"rc":0}
```

Smooth mouse movement. x/y is the total target displacement, c is the number of steps. Optional rx/ry/rc enable physical-mouse deceleration (recoil control): rc is the deceleration duration in milliseconds; the firmware converts it by the current mouse polling rate (1K = 1 report/ms, 8K = 8 reports/ms), so rc=N lasts N ms on either kind of mouse.

Parameter	Type	Description
x	int16	X-axis target displacement (negative = left)
y	int16	Y-axis target displacement (negative = up)
c	int16	Number of steps (1 = immediate, >1 = spread smoothly)
rx	float	Optional. Physical-mouse X-axis deceleration factor, default 1.0
ry	float	Optional. Physical-mouse Y-axis deceleration factor, default 1.0
rc	int16	Optional. Deceleration duration in ms (polling-rate adaptive), default 0 = off

Directional movement (cmd:31)

```
{"cmd":31,"x":100,"y":50,"c":1,"rx":1.0,"ry":1.0,"rc":0}
```

Directional movement that stacks with cmd:30 (displacements add up in the HID report). Fields are identical to cmd:30. When cmd:30 and cmd:31 both decelerate (rc>0), cmd:31 takes precedence.

Parameter	Type	Description
x	int16	X-axis target displacement
y	int16	Y-axis target displacement
c	int16	Number of steps
rx	float	Optional. Physical-mouse X-axis deceleration factor
ry	float	Optional. Physical-mouse Y-axis deceleration factor
rc	int16	Optional. Deceleration duration in ms (polling-rate adaptive)

Mouse button (cmd:32)

```
{"cmd":32,"b":1,"c":10}
```

Simulates a button press, auto-released after being reported c times. b is the button bitmask (1 = left, 2 = right, 4 = middle).

Parameter	Type	Description
b	uint8	Button bitmask (1 = left, 2 = right, 4 = middle)
c	uint32	Number of reports to sustain

Button state (cmd:33)

```
{"cmd":33,"b":1,"s":1}
```

Sets button state directly (immediate effect, no report count). s=1 press, s=0 release.

Parameter	Type	Description
b	uint8	Button bitmask
s	uint8	0 = release, 1 = press

Toggle upload state (cmd:34)

```
{"cmd":34,"status":1}
```

Enables / disables HID data upload. Once enabled, the device reports mouse data (buttons + movement) back over the serial port.

Wheel hold (cmd:35)

```
{"cmd":35,"w":-1,"c":100}
```

Wheel-hold command. w is the wheel value (±1 only), c is the duration in milliseconds.

Parameter	Type	Description
w	int8	Wheel value (-1 = down, 1 = up; other values clamped)
c	uint32	Duration (milliseconds)

Set macro (cmd:36)

```
{"cmd":36,"s":0,"e":1,"t":3,"p":1,"w":-1,"i":30,"d":10}
```

Configures a button macro. s = slot (0-7), e = enable, t = trigger mask, p = suppress mask, w = wheel value, i = interval period in ms (0 = continuous), d = on-duration per period in ms. Button bits: 1 = left, 2 = right, 4 = middle, 8 = side-back, 16 = side-forward.

Parameter	Type	Description
s	uint8	Slot (0-7)
e	uint8	Enable (0/1)
t	uint8	Trigger button bitmask
p	uint8	Suppress button bitmask
w	int8	Wheel value
i	uint16	Wheel trigger interval period (ms); 0 = continuous, no interval
d	uint16	Wheel-on duration per period (ms), ≤ i; 0 = full period

Query macro (cmd:37)

```
{"cmd":37,"s":-1}
```

Queries macro configuration. s=-1 queries all slots; s=0-7 queries a single slot.

Reset macros (cmd:38)

```
{"cmd":38}
```

Resets all macro configurations to firmware defaults.

Keyboard key (cmd:45)

```
{"cmd":45,"kc":4,"down":1}
```

Injects a single keyboard press/release. kc = HID keycode (0x04–0xE7; modifiers 0xE0–0xE7 = left/right Ctrl/Shift/Alt/GUI); down=1 press, 0 release. Only effective when the passthrough device exposes a keyboard interface; otherwise the device returns 404 no_keyboard. Call multiple times for multiple keys; release with down:0 or cmd:46. Real keystrokes still pass through normally (merged with injected ones).

Parameter	Type	Description
kc	uint8	HID keycode (0x04–0xE7; modifiers 0xE0–0xE7)
down	uint8	1 = press, 0 = release

Keyboard release-all (cmd:46)

```
{"cmd":46}
```

Releases all injected pressed keys. Ignored when there is no keyboard interface.

Enter Bootloader (cmd:50)

```
{"cmd":50}
```

Erases the calibration word and resets into IAP Bootloader mode. The device replies entering_iap first, then disconnects USB and waits for firmware upgrade. 8K V2 path: the device clears the anti-brick flag, writes BKP_DR1=0xA5A5, and re-enumerates after reset with serial number TOWMCUIAP — the COM number may change, so rescan ports.

Get version (cmd:0x84)

```
{"cmd":132}
```

Queries the firmware build date. Returns {code:200,message:"version",data:"build date"}. 8K V2 path: APP returns message="ysc-towmcu-L|R v1.0"; IAP returns data="YSC-IAP". Use the substring "-IAP" to tell the modes apart.

Switch baud rate (cmd:0x85)

```
{"cmd":133,"baud":3000000}
```

Dynamically switches the serial baud rate. The firmware does not validate the whitelist and configures the hardware directly — sending one of the 8 whitelisted rates (115200–3000000) is recommended.

Parameter	Type	Description
baud	uint32	Target baud rate

3.2 MAKCU protocol

Mouse movement

`km.move(100,50)`

Moves the mouse. Supports an optional segments parameter for stepped movement (range 1..512).

Parameter	Type	Description
x	int	X-axis offset
y	int	Y-axis offset
segments	int	Optional, number of steps (1–512)

Left button

`km.left(1)`

Left-button operation. 1 = press, 0 = release, 2 = silent release (does not enter the button-push queue).

Right button

`km.right(1)`

Right-button operation. Same parameters as the left button.

Middle button

`km.middle(1)`

Middle-button operation. Same parameters as the left button.

Side button 1 (ms1)

`km.ms1(1)`

Side-button 1 operation.

Side button 2 (ms2)

`km.ms2(1)`

Side-button 2 operation.

Wheel

`km.wheel(1)`

Scrolls the wheel one notch. Accepts -1 (down) or 1 (up) only; other values are clamped.

Lock X axis

```
km.lock_mx(1)
```

Locks X-axis movement. 1 = lock both positive and negative directions, 0 = unlock. Use lock_mx+ / lock_mx- to lock a single direction.

Lock Y axis

```
km.lock_my(1)
```

Locks Y-axis movement. Same as the X axis; supports lock_my+ / lock_my-.

Lock buttons

```
km.lock_ml(1)
```

Locks a button so the physical press has no effect. Supports lock_ml (left), lock_mr (right), lock_mm (middle), lock_ms1, lock_ms2.

Capture mode

```
km.catch_ml(1)
```

Button capture mode: 0 = auto, 1 = manual. Supports catch_ml / mm / mr / ms1 / ms2.

Invert axis

```
km.invert_x(1)
```

Inverts the direction of the given axis. Supports invert_x and invert_y.

Swap XY axes

```
km.swap_xy(1)
```

Swaps X/Y axis mapping.

Side-button wheel

```
km.scroll_side(1)
```

When enabled, side button 2 triggers wheel events.

Left+right wheel

```
km.scroll_lr(1)
```

When enabled, pressing left + right / side buttons together triggers the wheel.

Button-push mode

```
km.buttons(1,10)
```

Enables button-push mode. Physical button state changes are reported actively. mode=0 disables; period = 1..1000 ms.

Parameter	Type	Description
mode	int	0 = off, >0 = on
period	int	Push period (ms, 1..1000)

Serial number

`km.serial("abc123")`

Sets / queries / resets the device serial number (max 32 bytes). Quoted argument = set, no argument = query, argument 0 = reset. C-style escapes are supported: newline / tab / backslash / quote / hex byte (backslash-x followed by two hex digits).

Echo control

`km.echo(1)`

Controls command echo. 1 = on, 0 = off. A GET returns the current state.

Switch baud rate

`km.baud(3000000)`

Dynamically switches the serial baud rate (rate >= 115200, no whitelist check). No argument returns the current baud rate.

Parameter	Type	Description
rate	int	Target baud rate (>= 115200)

Get version

`km.version()`

Queries firmware version information. Returns km.version(km.MAKCU).

Reboot device

`km.reboot()`

Soft-resets the device.

4. Device States & Responses

4.1 Operating modes

Mode	Description	How to identify
APP	Normal application running (passthrough / command processing).	Version query cmd:132 returns the application version string.
IAP	Bootloader, awaiting firmware upgrade.	cmd:132 returns YSC-IAP; V2 enumerates with serial number TOWMCUIAP.

4.2 Version strings

Version	Meaning
ysc-towmcu-L v1.0	V2 left-side APP (device side facing the PC)
ysc-towmcu-R v1.0	V2 right-side APP (host side reading the real device)
YSC-IAP	IAP Bootloader (shared by V1 / V2)
Build date (e.g. Jul 23 2026)	__DATE__ returned by the V1 APP via cmd:132

4.3 Response format

YSC protocol: JSON frame {"code":200,"message":"<name>","data":"<...>"}. code:200 means success; jitter / mouse-curve query results come back via the asynchronous event debug_response (message = jitter / mouse_curve).

MAKCU protocol: plain text; responses end with \r\n>>>.

4.4 Serial parameters

Data bits 8 / stop bits 1 / no parity / no flow control. Firmware baud whitelist: 115200 / 230400 / 460800 / 921600 / 1M / 1.5M / 2M / 3M; 4M is an experimental extension (may not be stable on all hardware). Switch dynamically via cmd:133 or km.baud.

5. V1 / V2 Documentation Consistency Check

Conclusion: the command sets and protocol semantics of the two versions are identical; the differences are only in physical transport and chip architecture. The item-by-item comparison follows.

Item	8K V1 (single-chip)	8K V2 (dual-chip)	Verdict
Command numbers	YSC cmd 30~44, 50, 132, 133	Same (command numbers fully identical)	Consistent
MAKCU protocol	km.* text protocol	Same	Consistent
Frame format	<START> [2B big-endian length] [JSON] <END>	Same	Consistent
Command transport	USART1 UART (3M/4M baud)	USB-CDC (VID 1A86 / PID FE0C, baud-agnostic)	Same semantics, different physical layer
Chip architecture	Single chip (USBHS Host + Device on one die)	Dual chip (Right = Host reading the real device / Left = Device facing the PC, inter-chip link)	Different architecture (transparent to the host)
IAP upgrade	UART 1.5 Mbaud	USB-CDC (serial number TOWMCUIAP)	Same semantics, different channel
Documentation completeness	—	—	See below

Corrections from this review

cmd 39~44 (jitter / mouse curve) were previously missing: the driver GUI (App.vue) already sends jitter and mouse-curve configurations with these commands, but the protocol documentation (i18n/docs/{zh,en}.js) did not list them. They have been added to the YSC command lists in zh.js / en.js, consistent with this PDF.

cmd:50 (IAP), cmd:132 (version) and cmd:133 (baud rate) are described identically in the V1/V2 documentation — no changes needed.

The MAKCU protocol (km.*) is identical across both versions.

This document is auto-generated by the YSC 8K driver · command semantics are governed by the device firmware implementation.