

The TourBox **.tb** preset format

Everything here was derived from a TourBox Elite on Windows with Console 5.11.3, by writing files, importing them, reading the control list back, and re-exporting. Each claim carries a status:

- **confirmed** — observed directly on the device or in Console's own UI
- **inferred** — deduced from structure or arithmetic, not observed
- **unresolved** — known to exist, not decoded

Nothing below is guessed. Where a value is unknown it says so.

1. Container — confirmed

```
.tb → gzip (deflate level 6, MTIME = 0, OS = 0xFF, no FNAME/FEXTRA)
      → XML <TableTransfer>, 2-space indent, no XML declaration
          <configBytes> base64 → 3350-byte control table
          <presetName>   display name
          <presetConf>   section layout (which control ids appear in which group)
          <hudList>      HUD geometry and visible item ids
          <shortCuts>    optional <ShortDesc> dictionary, see §7
```

Writing the gzip header this way makes the round trip byte-identical. Verified on four independent vendor exports.

2. Control table — confirmed

```
3350 bytes = 22-byte header + 256 records × 13 bytes
record[0] == record index on all 256 slots (self-validating; assert on load)
```

Header. **header[1]** is **0x02** everywhere; **header[2..21]** are zero. **header[0]** is the preset's slot number inside Console, assigned on import (*inferred*): three probes written with **0x01** came back

as `0x08`, `0x11`, `0x12`. A generator can write anything there.

3. Record — confirmed

```
[0]      control id (equals the index)
[1]      flags: 0x08 marks a rotational control
[2]      reserved (zero in every sample)
[3..7]   action A
[8..12]  action B – the second rotation direction
```

4. Action block

```
[0] modifier mask   (when the kind is keyboard)
[1] action kind
[2] parameter       (non-zero for some non-keyboard kinds)
[3] reserved        (zero except three vendor blocks holding 63)
[4] key code within the kind
```

Correction to earlier descriptions. Byte [1] is *not* a two-valued "key page". Across 581 vendor presets it takes **41 distinct values**. It selects the kind of action. Only kinds 0 and 1 are keyboard.

Kind	Meaning	Status
0	ASCII character	confirmed
1	special key, compact vendor enum	confirmed
everything else	mouse, plugin commands, TourMenu, macros	unresolved

A kind-0 block whose code is 0 and whose byte [0] is 32 or 128 is *not* a keyboard action despite the kind.

5. Modifiers — confirmed

Bit	Key
0x02	Shift
0x04	Alt
0x08	Ctrl

Bits 0x01, 0x10, 0x20, 0x40, 0x80 are **unresolved** — they never appear on a keyboard action in any vendor preset.

Two independent lines agree. The vendor's own Photoshop preset binds the four Prime Four buttons to bare modifiers, and Console renders them [CTRL], [SHIFT], [ALT] against bytes holding 8, 2 and 4. Separately, the shipped shortcut dictionaries use the same encoding, and publicly known shortcuts pin each bit — m=8 s=Z labelled "Отменить" is Ctrl+Z, m=12 s=C labelled "Размер холста..." is Ctrl+Alt+C.

This contradicts the upstream project YongHee-Kim/tourbox-preset, which documents a bare Alt binding as 0x02. On this device and firmware 0x02 is Shift and Alt is 0x04. Anyone generating presets from the upstream table will emit Shift where they meant Alt.

6. Special keys (kind 1) — confirmed

Each name below was written into a probe, imported, and read back from Console's control list under that name.

Code	Key		Code	Key
1	Up		20	F11
2	Down		21	F12
3	Left		22	F13
4	Right		23	F14
5	PageUp		24	F15

Code	Key		Code	Key
6	PageDown		25	F16
7	Home		26	F17
8	End		27	F18
9	Insert		28	F19
10	F1		29	F20
11	F2		32	F23
12	F3		42	Numpad *
13	F4		43	Numpad +
14	F5		45	Numpad -
15	F6		46	Numpad .
16	F7		47	Numpad /
17	F8		48–57	Numpad 0–9
18	F9			
19	F10			

Function keys run F1–F24 across codes 10–33. Numpad digits are `48 + digit`.

Red rows. Codes 30, 31, 33, 34–41, 44, 58, 59, 60 are accepted and stored, but Console paints them red — its marker for a key it will not deliver. Re-exporting and diffing shows *records identical*, so red is a display verdict, not a rejection. Their names were unreadable at screenshot resolution, so they are left unnamed here even where arithmetic makes them obvious (30, 31, 33 fall inside the F-key run; 44 sits where Windows places `VK_SEPARATOR`).

Codes above 60 were never probed.

7. The `<ShortDesc>` dictionary — what it is and is not

Vendor presets ship a dictionary of `<m>` modifier mask, `<s>` key value, `<d>` human label. Values above `0x01000000` are Qt::Key constants.

It is a catalogue of **the application's shortcuts**, not a label list for the device's bindings. It cannot be used to name kind-1 codes: intersecting its candidates for compact code 3 yields `{Backspace, Return}`, while code 3 is confirmed `Left`. Its real value is §5 — `<m>` shares the control table's modifier encoding.

8. Control id map — confirmed

47 slots. Slot 59 carries the rotational flag, belongs to no section, and should be left alone.

Console is localised, so the same control has a different label depending on the interface language. English names are what this project uses everywhere — manifests, tools, documentation:

English	Русский (Console)	Physically
Knob	Ручка	round ridged wheel, centre right
Scroll	Колесо	vertical wheel top left, like a mouse wheel
Dial	Диск	large flat disc, bottom left — the biggest control
Tall	Длинная	tall button on the right side
Side	Боковая	narrow button on the left edge
Top	Поперечная	wide flat button across the top
Short	Короткая	small rounded button below Tall
D-pad	Вверх / Вниз / Левый / Правый	four small keys at the bottom
C1 / C2	C1 / C2	two round buttons, upper right
Tour	Tour	teardrop button, lower left

The two easiest to confuse are **Scroll** (Колесо) and **Dial** (Диск) — both round in Russian. Scroll is the little mouse-style wheel; Dial is the big flat disc.

Slot	Control		Slot	Control
0	Tall		20–23	Side + D-pad ↑↓←→
1	Side		24	Tall ×2

Slot	Control		Slot	Control
2	Top		25	Top + Tall
3	Short		26	Tall + Short
4	Knob		27	Side + Tall
5–8	Tall/Short/Top/Side + Knob		28	Short ×2
9	Scroll		29	Top + Short
10	Scroll press		30	Side + Short
11–14	Tall/Short/Top/Side + Scroll		31	Top ×2
15	Dial		32	Side + Top
16–19	D-pad ↑↓←→		33	Side ×2
34	C1		36, 37	Tall + C1, Tall + C2
35	C2		43–46	Top + D-pad ↑↓←→
42	Tour		55, 56	Knob press, Dial press
			57, 58	Short + C1, Short + C2

Cross-check: `<presetConf>` lists the Prime Four combinations in the order `[33, 31, 24, 28, 32, 27, 30, 25, 29, 26]`, and Console's UI lists them in exactly that order.

Dial combinations

The Rotating Section exposes Prime Four combinations for the Knob and the Scroll, but **not** for the Dial — there is no `Side+Dial` row among the 47 slots above.

That is a limit of the standard section, not of the hardware. Console's **Custom Section** lets you build a combination by hand, and the vendor's own presets use it: across the corpus, custom entries live in slots **62, 72, 135, 149, 150, 153, 155, 158**, listed in `<presetConf>` under `Custom Section.html`.

Every vendor example there is a button combination — flags clear, action A only, no second direction — so none of them shows what a Dial combination looks like. Until one is created in Console and exported, the translator cannot address `Side+Dial`, and it says so rather than picking a neighbouring slot.

To settle it: create the combination in Console's Custom Section, export, and diff against the same preset without it. The slot that changed is the answer.

9. Not decoded

- **Mouse buttons and drag.** The wheel is done (\$4a); clicks and press-and-hold are not.
- **TourMenu, macros, plugin commands** (the `[PS] ...` entries).
- Kind-1 code 0, which appears in vendor presets alongside an otherwise empty block.
- Modifier bits `0x01`, `0x10`, `0x20`.

A generator that emits kinds 0, 1 and 2 — any ASCII character, any special key, the mouse wheel, each with any combination of Ctrl, Alt and Shift, on any of the 47 slots, at a chosen turn speed and haptic strength — needs none of it.

4a. Mouse wheel — confirmed

Action kind 2 is the mouse wheel: **code 4 up, code 5 down**. Console renders such a row as "Колесико мыши вверх / вниз", and with a modifier byte set, as "ALT+Колесико мыши".

Code 0 also appears in vendor presets — Photoshop puts it on the Scroll — but a slot holding it reads as unassigned in Console. Not usable.

This is what makes pointer-driven adjustment work: a rotation sending the wheel moves whatever control the mouse is hovering, so Lightroom, Lumetri and Resolve sliders can be turned without any per-parameter shortcut existing.

4b. Turn speed and haptic strength

Both live in the flags byte of a rotation, one 2-bit field each. Across every vendor preset the byte only ever takes 0x00, 0x01, 0x02, 0x04, 0x05, 0x06, 0x08, 0x09, 0x0A — which is exactly two 2-bit fields and nothing else.

Bits	Meaning	Values
<code>0x03</code>	turn speed — <i>inferred</i>	0 normal, 1 slow, 2 slower
<code>0x0C</code>	haptic strength — <i>confirmed</i>	0x00 off, 0x04 light, 0x08 normal

Haptic is confirmed by probe6: the indicator icon in Console's control list differs on exactly the two rows where the probe changed those bits, and nowhere else. Speed is inferred from the same corpus range and still wants a read-back from the per-rotation settings panel.

10. How this was established

The method is reproducible on any Elite:

1. Export an empty preset from Console. Confirm a byte-identical round trip.
2. Generate a probe that writes known values into chosen slots.
3. Import it into Console **as a new preset**.
4. Read the control list — Console names each row, which names the code.
5. Re-export and diff — records identical means the value is stored verbatim.

Step 4 gives meaning, step 5 gives validity. Both are needed: a code can survive the round trip and still be one Console refuses to deliver.