

The translator

Contents

- [1. The model](#)
- [2. Manifest](#)
- [3. What the translator refuses to do](#)
- [4. Beyond keystrokes](#)
- [5. What still cannot be expressed](#)
- [6. The loop](#)
- [7. Where the knowledge lives](#)

Turning a line a human can read into the bytes a TourBox stores.

```
"side+dpadLeft": "ctrl+home"
      |           |
      |           └─ modifiers 0x08, kind 0, code 0x24
      └─────────── slot 22, action A
```

That is the whole job. This document is the contract: what the translator promises, what it refuses to do, and where it stops.

1. The model

A preset is 256 slots. A slot holds up to two actions — **A** and **B** — where B is the second direction of a rotation. An action is a modifier mask plus one key.

So a binding is three questions:

Question	Answer comes from
Which slot?	the control name — <code>dial</code> , <code>c1</code> , <code>side+dpadLeft</code>

Question	Answer comes from
What does it send?	the shortcut — <code>ctrl+home</code>
One direction or two?	a string, or an <code>{a, b}</code> pair

Nothing else is needed for a keyboard preset, and the translator does nothing else.

2. Manifest

```
{
  "name": "Browser - General",
  "base": "vendor/Audion_Clean.tb",
  "bind": {
    "dial":      { "a": "ctrl+shift+tab", "b": "ctrl+tab" },
    "knobPress": "ctrl+l",
    "tallx2":    "escape",
    "id:26":     "p1:33"
  }
}
```

`name` is what Console shows. `base` is the empty preset used as a write template and defaults to `vendor/Audion_Clean.tb` — the untouched sections of that file (HUD geometry, section layout) pass through unchanged.

Shortcuts

Modifiers, then exactly one key, joined by `+`:

- **modifiers** — `ctrl`, `alt`, `shift`
- **a character** — `a`, `7`, `[`, `=`, ```
- **a named special key** — `left`, `pagedown`, `home`, `f12`, `numpad7` ... (45 of them, listed in [FORMAT.md §6](#))
- **a spelled-out control** — `space`, `enter`, `escape`, `tab`, `backspace`, `delete`
- **a raw code** — `p1:33` addresses a special key the table has no name for

`ctrl+shift+tab` is Ctrl+Shift+Tab. `alt+left` is Alt+Left arrow. Case does not matter and `numpad 7`, `numpad-7`, `numpad_7` all resolve.

Control names

Direct: `tall`, `side`, `top`, `short`, `knob`, `knobPress`, `scroll`, `scrollPress`, `dial`, `dialPress`, `dpadUp`, `dpadDown`, `dpadLeft`, `dpadRight`, `c1`, `c2`, `tour`.

Combinations: `tall+knob`, `short+knob`, `top+knob`, `side+knob` — same four with `+scroll` — `side+dpadUp` ... `side+dpadRight` — `top+dpadUp` ... `top+dpadRight` — `tallx2`, `shortx2`, `topx2`, `sidex2` — `top+tall`, `tall+short`, `side+tall`, `top+short`, `side+short`, `side+top` — `tall+c1`, `tall+c2`, `short+c1`, `short+c2`.

That is 47 slots, and it is all of them. `id:N` addresses a raw slot for calibration work.

There is no `side+dial`. The Dial is the one rotation with no combination slots. Ask for it and the translator stops with an error listing what exists — it will not quietly pick a neighbouring slot.

3. What the translator refuses to do

These are deliberate, and they are the point.

It never guesses a slot. An unknown control name is an error, not a best-effort match. A typo that silently lands on the wrong button would be discovered months later, on the device, mid-task.

It never invents a key code. Only codes read back from Console under their own name are in the table. Codes that exist but were never confirmed are reachable only through the explicit `p1:N` escape, where the intent is visible in the manifest.

It never writes a file it cannot read back. Every build re-parses its own output and compares the control table. A preset that fails that check is not written.

It does not touch what it was not asked about. Bindings you leave out stay exactly as they are in the base file.

4. Beyond keystrokes

Two more things are decoded and available in a manifest.

The mouse wheel. `wheelup` and `wheeldown` are shortcuts like any other and take modifiers:

```
"scroll":      { "a": "wheelup",      "b": "wheeldown" },
"side+scroll": { "a": "ctrl+wheelup", "b": "ctrl+wheeldown" }
```

A rotation sending the wheel moves whatever the pointer is over. That is how a Lightroom or Lumetri slider gets turned when no shortcut for it exists.

Turn speed and haptic strength, per rotation:

```
"dial": { "a": "wheelup", "b": "wheel down", "speed": "slower", "haptic": "off" }
```

speed is **normal**, **slow** or **slower**; **haptic** is **off**, **light** or **normal**. Three rotations carrying the same action at three speeds is how a preset offers coarse and fine without a mode switch.

5. What still cannot be expressed

- mouse buttons and press-and-hold drag
- TourMenu entries
- macros
- application plugin commands, the **[PS] ...** rows in vendor presets

See **FORMAT.md §9**. Where a preset in this repository had to work around one of these, its manifest says so in a **_deviation** field rather than pretending the mapping is what was asked for.

6. The loop

```
edit manifests/40-media-potplayer.json
python tools/tbmake.py manifests/40-media-potplayer.json
    → presets/40-media-potplayer.tb
import into Console as a NEW preset
```

Then check it, in this order:

1. **Read the control list.** Console names every row. The names should be the shortcuts you wrote. This catches a wrong slot instantly.
2. **Re-export and diff.** `python tools/tbfmt.py diff presets/x.tb exported.tb` Expect **RECORDS identical**. The header will differ at byte 0 — that is Console's slot number, not your data.
3. **Use it for an hour.** The layout is the part no tool can verify.

Step 1 tells you the mapping arrived. Step 2 tells you nothing was silently rewritten. Neither replaces step 3.

7. Where the knowledge lives

`calibration/calibration.json` is the single source of truth for every decoded fact, and every entry carries a status — `confirmed`, `inferred` or `unresolved`. The tables inside the tools are derived from it, never the other way round, and no fact is duplicated into a code comment where it could drift.

If you re-derive any of it on different hardware and get a different answer, the calibration file is what to correct — the method is in `FORMAT.md` [§10](#).