Description

Title

TraceLoop Telemetry Feed Integration Specifications

The following sections provide dedicated integration specifications for six physiological telemetry feeds in the TraceLoop system. Each integration is structured according to TraceLoop's canonical architecture (Sections 2–17 of the system specification) and details the data interfaces, schema, control logic (Full vs. Lite implementations), safety/guard-rails, UI integration, and implementation effort. These integrations are designed to fit into TraceLoop's relational factor table and arbitration engine (FIG. 3 and FIG. 4 of the TraceLoop documentation), using the standard fifteen-column loop schema (Table 1) and cross-channel rule it um

(priority_over, mutually_exclusive, requires_ok, synergy_with, etc.). New figures are introduced for novel sensor modules (e.g. PLR, Proning) with call-out descriptions, and references are made to existing TraceLoop figures (such as FIG. 3 relational schema, FIG. 4 arbitration flowchart, and FIG. 11 factor table) where applicable.

Anti-Xa (Lab-Based Coagulation Feedback)

Supported Feed Formats & Sources: Anti-Xa (anti-factor Xa) levels can enter TraceLoop via multiple interfaces. The most common source is the hospital lab system, which can send Anti-Xa results as HL7 v2 ORU messages (e.g. ORU^R01 result feeds) after a blood draw. In a Full implementation, more continuous monitoring is possible via an **arterial-line lab-on-chip sensor** that resides in-line with an arterial catheter. This microfluidic sensor can directly measure heparin activity (Anti-Xa) in near real-time. The arterial-line sensor module would likely communicate over a local bus (e.g. I²C/SPI on a bedside controller or a serial RS-232/USB link if it's an external device), or it could publish readings via MQTT if designed as a networked IoT sensor. Bluetooth Low Energy (BLE) is less common for lab analytes but could be used for a portable point-of-care coagulation tester that pairs with the TraceLoop hub. In summary, **HL7** is used for lab-reported values (discrete, intermittent updates), whereas a dedicated sensor would use a **streaming interface** (embedded bus or serial) for continuous data. All sources funnel into TraceLoop's sensor ingestion pipeline with proper timestamping and validation.

Payload Schema: Regardless of source, the Anti-Xa feed payload is defined with fields for:

- `antiXa_value` – the measured level (e.g. in IU/mL), numeric (float).
- `units` – typically "IU/mL" for Anti-Xa activity.
- `timestamp` – ISO 8601 format or UNIX epoch for when the sample was taken (lab time or sensor read time).
- `quality_flag` – (optional) any quality indicator (e.g. lab result status or sensor confidence).
- **Frequency:** Lab feeds may arrive as needed (every few hours to daily), whereas a continuous sensor could stream at a lower frequency (e.g. every few minutes, or on significant changes) to avoid noise. For example, an arterial-line chip might update Anti-Xa every 5–15 minutes with a new measurement.

The feed is integrated into TraceLoop's data model as a **Hematology sensor modality**, likely mapped to a *Hemato* category in the factor table (since Anti-Xa relates to anticoagulation). In FIG. 11 (Factor-Row Table), each sensor-driven loop is one row. The Anti-Xa feed corresponds to a factor row with a unique ID (e.g. `ANTI_XA`) and a human-readable label ("Anti-Xa Level"). The **conflict_group** for Anti-Xa's loop depends on its actuator: here it is associated with the protamine infusion pump, so we assign it to the protamine pump's conflict domain (e.g. `PUMP_PROT`). This means the Anti-Xa control loop will contend only with loops using that same actuator (the protamine delivery system) in the arbitration engine. (Note: the heparin infusion itself would be a separate loop on a different pump channel – see below – so physical conflict is minimal; the priority logic rather than conflict_group handles their interaction.)

Control Logic Path (Full vs. Lite): In **Full** closed-loop mode, TraceLoop uses Anti-Xa feedback to automatically regulate anticoagulation and reversal agents. The control logic follows a rule-based threshold approach combined with risk-weighted arbitration:

- When Anti-Xa is **sub-therapeutic** (below a lower threshold, e.g. <0.3 IU/mL), the system will **initiate or increase heparin infusion** to reach therapeutic levels. This is handled by a separate factor (e.g. `HEPARIN_INFUSE`) which would have its own row (likely conflict_group of the heparin pump, e.g. `IV_PUMP_3`). That loop might have a rule like "if ANTI_XA < 0.3, then start or titrate heparin at X Units/kg/hr". When Anti-Xa is **above the upper threshold** (e.g. >0.7 IU/mL or evidence of overdose), the Anti-Xa loop triggers **reversal**: it activates the protamine pump to deliver a neutralizing dose. For example, the rule might specify "if Anti-Xa > 0.7 IU/mL, then stop heparin and give protamine 1 mg per 100 U of heparin administered". These threshold values can be adjusted per protocol, but 0.3/0.7 IU/mL are typical for heparin therapy ranges.
- The **factor table definitions** capture this logic via cross-channel relationships. The Anti-Xa loop (driving protamine) is assigned a high harm_severity (since uncontrolled anticoagulation is critical, likely CRITICAL severity with time_to_harm = minutes). It is given a `priority_over` edge

over the heparin infusion loop. Meanwhile, the heparin infusion loop includes a `requires_ok = ANTI_XA` gating field, meaning it will only run if the Anti-Xa loop is not indicating overdose. In practice, this gating ensures heparin infusion is **blocked whenever Anti-Xa is critically high** and the protamine loop is active. The engine will topologically sort these so that the Anti-Xa/protamine action has precedence, preventing a scenario where heparin continues running during an overdose reversal. This is exactly the interaction described in the TraceLoop documentation: *"ANTI_XA (risk 17.5) has priority_over = HEPARIN_PROTAMINE; HEPARIN_PROTAMINE has requires_ok = ANTI_XA… The engine ranks ANTI_XA first and the gate prevents heparin from running until anti-Xa ≤ 0.7 IU."*. (Here `HEPARIN_PROTAMINE` refers to the heparin infusion loop; it's gated until the Anti-Xa loop signals levels are safe.)

- In **Lite** (non-autonomous) mode, the Anti-Xa feed is used for decision support and alarms rather than direct actuation. The system will monitor the feed and, when thresholds are crossed, generate recommendations or alerts. For example, if Anti-Xa < 0.3, TraceLoop Lite could display a suggestion like "Anti-Xa low – consider starting heparin at 5 U/kg/hr" and highlight the heparin pump control. If Anti-Xa is high (>0.7), it might alarm and recommend "Hold heparin; prepare protamine dose of 1 mg per 100 U heparin given." The **factor row still exists** in Lite, but instead of auto-executing the action, it may be flagged with a mode that requires human confirmation (e.g. an internal `auto_execute` flag off). The control logic path thus stops at a **UI prompt**. The underlying conflict relationships (priority and gating) still apply to prioritize the alerts – for instance, an Anti-Xa high alert would suppress any prompt to continue heparin. The risk scoring from the factor table can be used to rank this alert's priority on the interface (making sure a critical anticoagulation alert is top-tier).

Actuator Integration (Full): In Full mode, the Anti-Xa loop directly controls an **infusion pump actuator** for protamine (and indirectly controls the heparin pump via stop commands). The protamine dosing follows a micro-dosing strategy to avoid overshoot – for instance, delivering protamine in increments and verifying the effect on Anti-Xa on subsequent lab draws or sensor readings. A **rate envelope** may be defined: e.g. at most X mg/min of protamine to avoid rapid swings in coagulation. In this case, a simple proportional dosing equation could be used (e.g. compute required protamine based on the estimated heparin in circulation, and infuse over 10–20 minutes). The heparin infusion itself is also an actuator pathway, but that loop might already exist in the system (e.g. as part of a hemodynamic support factor). The coordination of actuators is handled through TraceLoop's conflict resolution – here, because protamine and heparin use different pumps, they are in different conflict_groups (no direct hardware conflict) but logically one should **not** run while the other is counteracting it. This is ensured by the priority_over + requires_ok rules described above rather than by physical locking. The system also logs each actuation for audit (e.g., "ANTI_XA loop commanded PROTAMINE_PUMP at 10 mg over 5 min" which is chained into the audit log).

Guard-Rails, Conflict Graph & Fallback: The Anti-Xa integration benefits from TraceLoop's multi-layer safety architecture. At the **local loop level (L1 guard)**, the Anti-Xa feed is validated (e.g. ignoring obviously spurious lab values or sensor noise) and checked against safe ranges. For example, if the sensor suddenly reports an impossibly high Anti-Xa, the system might flag a sensor error and refrain from massive protamine dosing. Next, the cross-channel **conflict graph** ensures no conflicting

commands: since Anti-Xa and heparin loops target the same physiology but via different actuators, TraceLoop uses the graph rules to prevent antagonistic actions in the same cycle. The *requires_ok* gating effectively serves as a **guard-rail**: it is a compiled safety rule that ensures heparin (anticoagulant) cannot run if the Anti-Xa loop has determined coagulation is too low (over-anticoagulated). This deterministic gating is logged and auditable. Additionally, because both loops share a clinical goal (coagulation balance), the system might mark them as *mutually_exclusive* in a soft sense – though priority/gating already handles it, a mutually_exclusive edge could be added to double-insure they never execute simultaneously. In the **risk arbitration** (FIG. 4 process), the Anti-Xa loop carries a higher risk score than routine heparin infusion, so it will naturally preempt in the priority queue for the pump or for action sequencing. This was demonstrated in the example JSON log where Anti-Xa ("winner") outranks heparin ("loser") with risk 17.5 vs 15.

Fallback modes: If for any reason the Anti-Xa closed-loop must be aborted (for example, repeated sensor failures or a clinician manually disables it), the system will revert to manual control. The **failsafe hierarchy** (FIG. 5) includes an automatic fallback to a safe state – here the safe state would be to hold heparin (to avoid inadvertent overdose) and wait for clinician input. An L5 manual override is always available, allowing a clinician to BLOCK the Anti-Xa loop or FORCE it for one cycle. For instance, a clinician might force a protamine dose despite Anti-Xa not yet reading high if clinical context demands – this would be done via the override mechanism with proper logging. The system's watchdogs (FIG. 8) would monitor that the protamine pump responds correctly; any failure to deliver or hardware fault raises an F_DRIFT or fault flag as per the general actuator verification logic.

UI Chip Behavior: On the user interface, the Anti-Xa loop is represented as a **therapy chip** or tile in the TraceLoop dashboard (likely under a Hematology or Coagulation category). In **Full mode**, this chip would display the current Anti-Xa value and its status (e.g. "Anti-Xa: 0.8 IU/mL – High" in red). When the loop is actively intervening, the chip would be **green** (indicating it's running normally), possibly with an icon showing an infusion in progress. If a safety threshold is crossed, it might blink or turn red to indicate a critical state. If a clinician taps the chip (or the linked override dial), they could suspend the loop – doing so would turn the chip amber (Override active) and start a countdown as per the override rules. The chip could also present controls like "Confirm Protamine Dose" or "Override and Continue Heparin" depending on context. In **Lite mode**, the Anti-Xa chip likely appears as an indicator and alert source. It might show the value and a colored status (green for in-range, yellow for slightly off, red for out-of-range). If an action is recommended, the chip may flash and offer a **"Suggestion"** button – e.g., "Anti-Xa low: Recommend heparin 5 U/kg/hr – [Administer]?" which a clinician can press to carry out the action manually. The Lite chip thus does not automatically dose but guides the user. In both modes, the chip would be integrated into the **explainability layer** – if the user drills down, they can see a log of "Anti-Xa reading X triggered protamine at time Y".

Implementation Effort: *Full integration* of Anti-Xa feedback requires moderate code and device-side work. On the code side, adding the Anti-Xa loop means creating new factor entries in the database (for Anti-Xa and for heparin infusion logic, if not already present) and implementing the threshold-triggered actions. This is relatively straightforward since it's rule-based (no complex algorithm beyond threshold checks) – many components (priority logic, gating, override) are leveraged from the existing engine. The main code lift is ensuring the HL7 lab interface or sensor driver is reliable and mapping results into the loop in real-time. If using the arterial-line sensor, device-side lift is higher: developing or integrating the

sensor firmware, calibrations, and bus communication. TraceLoop's hardware module would need to interface with that (likely via an MCU analog front-end or digital interface). If sticking to lab results via HL7, device-side lift is minimal (the lab system and interface engine configuration, which is a known quantity). *Lite integration* is easier: mostly UI/UX development to display values and recommendations, plus configuration of alerts. Code changes to the core engine are minor for Lite (you might flag the Anti-Xa loop as "advisory only"). Overall, Anti-Xa integration builds upon existing infusion control patterns in TraceLoop, meaning **medium effort for Full (mostly integration and testing)** and **low to medium for Lite**. Regulatory-wise, closed-loop anticoagulation will demand thorough verification due to high risks, so testing effort is significant (simulate various Anti-Xa scenarios to ensure correct priority arbitration, as per ISO-14971 risk management).

Head-of-Bed Angle (HOB)

Supported Feed Formats & Sources: The head-of-bed (HOB) angle is typically provided by bed systems or posture sensors. Many modern ICU beds have angle sensors for the backrest, which can interface with external systems. **HL7** is commonly used for device data via the Patient Care Device (PCD) profiles – for example, a bed might send periodic ORU messages with the current backrest angle or alarms if the angle is below a threshold (per VAP prevention protocols). In fact, some integrated hospital platforms send an HL7 alarm when HOB falls below 30°pubmed.ncbi.nlm.nih.govpmc.ncbi.nlm.nih.gov. If direct HL7 from the bed is not available, an **integration gateway** (like a device aggregator) can poll the bed or use the bed's API. Older bed models may output analog/serial data – e.g., a bed could have an **RS-232** maintenance port providing angle, or a simple analog voltage proportional to angle that can be read via an ADC. In the absence of a smart bed interface, a **retrofit BLE inclinometer** can be attached to the bed frame. For example, a BLE sensor on the head section can transmit angle data at intervals (this would be a custom hardware addition). Another modern approach is using a small **IoT device with MQTT**: a microcontroller with an accelerometer that publishes the bed angle to an MQTT topic on the hospital network. All of these sources provide the same essential data (angle in degrees), and TraceLoop's device abstraction would normalize it. We anticipate a polling frequency of perhaps 0.1 Hz to 1 Hz (once every 1–10 seconds) for HOB angle updates, as rapid changes are rare and not needed at high frequency.

Payload Schema: The HOB angle feed payload includes:

- `angle_deg` – numeric value of bed head elevation in degrees (0° = flat supine, 90° = full upright).
- `timestamp` – when the reading was taken (to correlate with events, ISO 8601).
- `bed_id` – identifier if multiple beds or a network feed (to ensure association with correct patient, relevant in multi-bed systems).

- valid_range – (optional metadata) allowed range flag, e.g., "OK" if within safe range, or "LOW" if below threshold, if the device pre-flags it.

- **Frequency:** Typically on change or periodic (e.g., every 5 seconds). Alarms could be event-driven (immediate message when angle falls out of range).

Within TraceLoop's factor table (FIG. 11 schema), the HOB angle corresponds to a **non-pharmacological device loop**. We might classify it under a *Ventilation Support* category (since it relates to VAP prevention and respiratory care) or a *Safety* category. The factor ID could be HOB_ANGLE. It will have a **conflict_group** representing the bed actuator (for Full implementations where the system might adjust the bed). We can define a new conflict group, e.g., BED or BED_MOTOR, to namespace bed position commands. All bed-related loops (HOB angle, proning, PLR if it involves bed adjustment) would share this conflict_group so that only one bed movement occurs at a time. If the bed is not automated in Lite mode, the conflict_group still exists but no actual output is sent – it may simply be used to prevent simultaneous bed-related suggestions. The *sensor_modality* for HOB would be "Angle Sensor" or similar.

Control Logic Path (Full vs. Lite): In **Full** mode, the TraceLoop system actively ensures the HOB angle stays within a target range (commonly ≥30° for ventilated patients to reduce aspiration risk[pubmed.ncbi.nlm.nih.gov](pubmed.ncbi.nlm.nih.gov)). The control loop is relatively simple: it monitors angle_deg. If the angle drops below a threshold (e.g. 30°), after a short grace period it will **issue a command to the bed** to elevate the head section to the target angle (say 30° or a configurable 30–45° range). Many ICU beds support remote adjustment via integrated systems or could be outfitted with actuators that accept commands. The factor table for HOB_ANGLE would encode a threshold rule (trigger condition angle_deg < 30 degrees) and an associated action "raise bed to 30°". This could be encoded as a pair of thresholds in the factor's parameters (e.g., "low threshold 30°, high threshold none" – since we only care about a minimum angle in this scenario). The **priority** of this loop is mainly patient safety (moderate risk of VAP if violated), but not as acute as, say, an active drug loop. Likely harm_severity might be set to MODERATE with a time_to_harm of hours (since a low HOB angle increases pneumonia risk over hours). Thus, it would have a risk score that ensures it gets addressed promptly but it might not override life-critical loops. In practice, however, because adjusting the bed doesn't conflict with, e.g., drug delivery, it can operate concurrently – conflict arises only if another bed movement loop is in play.

For cross-loop logic, HOB angle might have **mutually_exclusive** relationships with certain other factors:

- If a **Proning** loop is active (patient is being turned prone), the HOB angle rule may be temporarily suspended or considered incompatible (because during proning the bed might be flat by necessity). We could set mutually_exclusive between HOB_ANGLE and PRONING loops so they don't try to act at cross purposes (one trying to raise head while another is flipping the patient).

- HOB angle might also be set to not interfere with **CPR** or emergency loops (e.g., if a code blue loop requires bed flat, it would override HOB angle). This can be handled by conflict_group (if code uses bed actuator) or by a priority rule (code blue loop would simply have higher priority on

the bed device).
Other than those, HOB angle loop can run independently.

In **Lite** mode, the system does not move the bed automatically but will alert clinicians if the HOB angle is out of range. The logic in the factor row still triggers at 30°, but instead of an automatic command, it might produce an **alert**: e.g., "Head-of-bed angle low (20°) – Elevate head to ≥30°" on the user interface. The system may also interface with hospital alert systems: for instance, sending a notification to the bedside nurse's phone or to the central station if the bed is flat for more than a few minutes without a documented reason. This is similar to existing VAP prevention protocols. If the bed system itself provides an alarm (some beds have built-in angle alarms), TraceLoop can ingest that and simply propagate it via its UI (ensuring it appears on the TraceLoop dashboard alongside other loops). The factor row might be marked as "advisory" in Lite mode, meaning no actuator output is generated – only UI alerts and perhaps an entry in the explainability log that the condition was detected. The logic path thus goes: sensor input → threshold check → event flag → UI notification, without entering the arbitration queue for actuators (since no actual actuation in Lite).

Actuator Integration (Full only): In a full integration scenario, controlling the bed requires an actuator interface. If the bed is networked and has an open API or is connected to a bed controller, TraceLoop would send a command like "Set HOB Angle = 30°" or "Raise head section by X degrees". This might be done via a vendor-specific protocol or a middleware (some hospital systems have bed integration platforms). The actuator in this case is the **bed motor** for the head section. Micro-dosing is not exactly applicable here, but *rate limiting* is relevant – the bed should not jolt the patient, so the command might specify a slow movement or the bed's internal logic handles it. We define safe envelopes such as raising no faster than the bed's standard speed and perhaps checking load (to ensure no obstructions). The HOB loop would also possibly include a **deadband or hysteresis** – e.g., if the target is 30°, maybe command at 28° to 32° to avoid chattering if sensor oscillates around 30. In terms of code, adding bed actuation means implementing a driver or message sender. If a direct network command is unavailable, a possible implementation is using a small IoT actuator that physically presses the bed's angle-up button (an out-of-scope hack but conceivable). For the spec, we assume a proper interface exists (like a digital command through the bed's system). No other actuators are involved with HOB, and adjusting bed angle does not consume any shared resource except the bed's own controls (hence conflict_group separation).

Guard-Rails, Conflict Interactions & Fallback: The HOB angle loop mainly acts as a preventative safety measure. Guard-rails are straightforward: a **local guard** might enforce that the bed is not raised beyond safe limits (e.g., not beyond 45° if that's max for patient's condition to avoid sliding or hypotension). If a command to adjust the bed fails (bed unresponsive or mechanically blocked), the system will go into a fallback mode by raising an alarm to human staff – essentially, it cannot correct the situation automatically, so it escalates. Conflict-wise, as discussed, the primary interactions are with proning or certain procedures:

- **Mutual Exclusion:** Marking HOB and Proning as mutually exclusive loops so they are not both active. If a proning session is initiated (perhaps via a PRONING factor going active), the HOB loop can automatically suspend its enforcement (perhaps by an internal flag or simply by not triggering while proning loop's state = "prone"). After proning is done (patient supine again), the

HOB loop would resume.

- **Requires_OK:** Alternatively, we could use a gating approach: e.g., HOB loop could have a requires_ok = SUPINE_POSITION if we define a factor that is true only when patient is supine. Or the proning loop could set a global state. This gating would effectively turn off HOB enforcement unless the patient is supine.

- **Priority Over:** If HOB angle adjustment and some other bed movement request occur simultaneously, a decision is needed. For instance, if a Passive Leg Raise loop (which also uses bed motors) is triggered at the same time as HOB low, which takes precedence? Likely the urgent one for patient care (PLR is diagnostic and short, whereas HOB low could be tolerated for a minute). We might set relative priority scores (risk for PLR is low – just diagnostic, whereas risk for low HOB is moderate – aspiration risk). Thus the scheduler might rank raising HOB over performing a PLR at the same instant. However, since these loops are in the same conflict_group (BED), the arbitration will choose the higher risk one first. We would set **harm_severity** for HOB loop at maybe 3 (moderate) and PLR maybe 1 (low), so Risk(HOB) > Risk(PLR) and HOB wins the bed control when needed.

- **Guard-rail rules:** If the patient's condition contraindicates elevating HOB (for example, certain spinal injuries might require flat bed), the loop should be easily disabled. This could be achieved by a clinician entering an order that sets the HOB loop's state to "off" or by an override with indefinite duration. TraceLoop's policy point could allow a flag to suspend that loop for medical reasons (documented). Under the hood, this would be an *override BLOCK* at the factor level.

- The HOB loop doesn't directly involve drug delivery, so it does not interplay with FIG. 8 watchdog timers beyond standard system reliability. If bed data stops updating (sensor disconnect), a **timeout** can be set (e.g., no angle data for >30s triggers a comms fault alarm). The fallback in that case is again to notify staff that bed angle is unknown – they should manually verify positioning.

If the system had tried to raise the bed and it didn't move (maybe a mechanical fault), **fallback** is manual adjustment: an alarm "Bed angle adjustment failed" would sound, prompting immediate human intervention. The system might also mark the BED actuator in a fault state so no further automated attempts occur until reset (to avoid motor strain). In summary, the system will always fail safe – a low bed angle will never go unnoticed; either the system fixes it or alarms persist until resolved, consistent with safety layer escalation.

UI Chip Behavior: In the TraceLoop UI, HOB angle might be represented as a **small tile indicator** under ventilator or safety parameters. For example, a simple icon of a bed with a numeric angle display. In **Full mode**, the chip would be interactive in that it shows if the loop is actively controlling the bed. If the angle is above threshold, the chip might be green or normal, showing "HOB: 35° ✓". If it falls below threshold, the chip would turn red or flash, showing "HOB: 20° ↻ (Correcting)" indicating that the system is correcting it (the rotating arrow iconography might suggest an automatic adjustment in progress). Once the bed is raised, it returns to green. The clinician can tap this chip to see details or to override. Overriding

here might mean "do not raise bed for now" (for instance, if there's a medical reason to keep patient flat temporarily). Activating an override would turn the chip amber and perhaps display a countdown (if a temporary override) or a lock icon (if indefinite). In **Lite mode**, the chip would primarily serve as an alert. It could still display the current angle continuously (which is useful information). If the angle drops too low, it might change color (yellow if slightly low, red if significantly <30°). The text might say "HOB: 20° (Low!)" and an alert icon. The nurse can acknowledge the alert via the UI – perhaps tapping it marks it acknowledged (but it stays red until angle is fixed). In Lite, since the system isn't actively changing anything, the chip might also allow the user to document a reason or trigger actions. For example, tapping could provide quick buttons: "Elevate to 30°" which if pressed might send a command to an integrated bed *or* simply record that the nurse did it. (If no bed integration, the nurse physically raises the bed and the sensor update will soon reflect the change, turning the chip green again). The chip's state changes also propagate to remote monitors – e.g., the central dashboard could show a warning if any patient's HOB is low, as part of the VAP bundle compliance view.

Implementation Effort: Implementing HOB angle integration is a relatively **low-to-medium effort** on the software side, but **device-side effort** depends on bed connectivity. For *Lite mode*, most of the work is ingesting the data. If using HL7 from an existing system, development is largely configuring the interface engine to parse the ORU message (which contains an angle observation or alarm). If using a BLE or IoT sensor, a small driver to read that (via BLE gateway or MQTT client) must be written – modest complexity. The control logic for threshold check is trivial. UI additions (displaying angle, generating an alert) are straightforward as well. For *Full mode*, additional challenges include writing a command interface to the bed. Many hospital beds do not have open APIs, so this could involve vendor-specific SDKs or creative solutions (perhaps via a PLC that drives the bed motors). Assuming an API exists, code must be written to format a command (which might itself be an HL7 command or a REST call, etc.). Safety testing is needed to ensure the bed moves correctly on command and stops at the desired angle (feedback loop tuning might be needed if the bed reports angle in real-time while moving). On the factor table side, adding HOB_ANGLE as a new factor row with appropriate columns is simple. Ensuring the mutual exclusion with proning is set requires adding a cross-reference row in the relations table (which is again straightforward data entry as per FIG. 3 ER schema). From a regulatory perspective, making this automated could push TraceLoop into the realm of device control for patient positioning – this is usually considered lower risk than drug delivery, but still needs verification (e.g., ensure no harm like dislodging lines when raising bed). The **code lift** is moderate mainly if bed control is implemented; otherwise, just reading the sensor is light. The **device lift** could range from none (if bed already integrated) to high (developing a hardware solution to press bed buttons or a custom bed interface). For a first version, the Lite approach (alert only) might be done first given its simplicity, and the Full actuation could be added when a bed with remote control is available. Overall, HOB angle integration leverages traceLoop's existing alerting and factor framework with minimal modifications, aligning with patient safety goals recommended in ICU guidelines (HOB 30–45° to prevent VAPpubmed.ncbi.nlm.nih.gov).

Passive Leg Raise (PLR) Response

Supported Feed Formats & Sources: The Passive Leg Raise (PLR) test is a dynamic maneuver rather than a single sensor output, but its integration into TraceLoop involves capturing both the *execution of the maneuver* and the *hemodynamic response*. There are a few ways this can be achieved:

- **Bed Sensors/Interface:** If the bed is motorized, the system can initiate and detect a PLR via bed angle sensors. For example, the bed's head can be lowered and legs raised to perform a PLR (often done by moving from a 45° semi-recumbent to flat with legs elevated ~45°)litfl.com. The feed here could be the **bed position** (angles of head and leg sections). A combination of a drop in head angle to 0° and raise in leg section to 45° indicates a PLR is in progress. The bed can either report a specific "PLR mode activated" if it has presets, or TraceLoop can infer it from angle changes. This would likely use the same bed interface as HOB (HL7 device data or a bed API).

- **Hemodynamic Monitor Feeds:** The crucial part of PLR is measuring the patient's response – typically an increase in stroke volume or arterial pulse pressure indicates fluid responsivenesslitfl.com. TraceLoop would ingest data from continuous cardiac output monitors or arterial lines. Common devices: pulse contour analysis (e.g., FloTrac/Vigileo), bioreactance (NICOM), or echocardiography outputs. Many of these provide values like **Cardiac Output (CO), Stroke Volume (SV), Pulse Pressure (PP)** in real time. Standard formats include **HL7 PCD-01 messages** from patient monitors or vendor-specific serial protocols. For example, an advanced ICU monitor could output SV every 10 seconds; a NICOM might output an "SV change %" reading. If those are not directly available, TraceLoop could calculate it from raw signals (like capturing an arterial line waveform via an analog input or digital feed and computing pulse pressure, though that's complex and likely the monitor does it).

- **Manual Input or Trigger:** In absence of automation, a clinician can indicate in the system that a PLR is being done (e.g., pressing a "Perform PLR" button on the UI). This could start a timer and prompt the user to observe certain readings. The outcome (positive/negative) might be manually entered or selected, which then feeds into the TraceLoop logic. This "feed" is more of a user-driven event.

- **BLE or Wireless Sensors:** Conceivably, a future bed or device might have a dedicated PLR detection sensor. However, it likely boils down to the above components (bed angles + hemodynamics).

Thus, PLR integration draws from **multiple sources**: bed position (for the maneuver) and cardiovascular measurements (for the effect). TraceLoop can treat the PLR as a compound sensor: it knows a PLR test is underway (by command or detection) and then reads the CO/PP values. If those sources use HL7 or other protocols, the system listens on those channels. For instance, an HL7 message from a cardiac output monitor might contain stroke volume values every 15 seconds. The system might sample values just before PLR and during the PLR (30–90 seconds into the maneuverlitfl.com) to compute the delta. MQTT could also be used if an IoT sensor publishes a "PLR result" after completion (though not common now). RS-232 could be relevant for older monitors (some hemodynamic monitors output data on serial ports).

Payload Schema: The PLR "feed" can be represented in TraceLoop as an event with associated measurements:

- `PLR_trigger` – a boolean or state (e.g., `INITIATED` at time X, `COMPLETED` at time Y). This could be implicit if we model PLR as a factor that can be "active" during the test.
- `SV_change` – the change in stroke volume (or CO) due to PLR, typically expressed as a percentage.
- `PP_change` – alternatively or additionally, the change in pulse pressure (mmHg or %) due to PLR.
- `responsive` – a derived boolean or categorical result (e.g., `RESPONSIVE` vs `NON_RESPONSIVE` to fluid). This is determined by whether the SV or PP increase exceeds a threshold (commonly around +10% SV or +10% PP signifies fluid responsiveness[ittfi.com]).
- `timestamps` – of baseline measurement (pre-PLR) and during PLR peak.
- `quality` – optional field if the signal was adequate (e.g., arrhythmia during PLR might invalidate results).

In the factor table, we might create a factor `PLR_TEST` that represents the passive leg raise maneuver and its outcome. This factor would likely be categorized under *Hemodynamic* or *Fluid Management*. It might not have a continuous numeric value to track like other sensors, but rather an event-driven state. One way is to treat the detection of responsiveness as a boolean sensor output that becomes true if criteria are met. The **conflict_group** for PLR is the bed (since physically raising legs uses the bed motors) – same as HOB, we assign `BED` for actuation. The PLR_TEST factor would have an associated actuator action (performing the leg raise) and then rely on sensor data to complete. We might also have *no direct actuator* if the clinician initiates it, but to align with Full integration, we assume the system can do it. The **thresholds** here are not static like a number, but rather conditions: we define threshold in terms of SV increase (e.g., threshold 10% for positive response). This can be encoded in the factor's rule logic as: *if PLR initiated and SV_change ≥ 10% → set PLR_TEST outcome = TRUE (positive)*. TraceLoop's architecture might handle this via either a small script in the loop or by splitting it into two factors (one to initiate PLR, one to detect response). However, for spec simplicity, consider it one factor with multi-step logic.

Control Logic Path: In **Full implementation**, TraceLoop can autonomously conduct a PLR test and use the result to guide fluid therapy:

1. **Initiation:** The system decides when a PLR is needed – typically if there are signs of possible hypovolemia or before giving a fluid bolus, a PLR can be done to assess responsiveness. This decision could be encoded as a trigger in another factor (e.g., a shock factor might trigger PLR if blood pressure is low and patient is not fluid overloaded). Alternatively, a caregiver can initiate it

via UI and let the system handle the details.

2. **Actuation (Bed Control):** TraceLoop sends commands to adjust the bed: lower the head to horizontal and raise the legs to ~45°litfl.com. This is done by the PLR_TEST factor taking control of the BED conflict_group. While PLR is active (let's say it sets an internal state "PLR_ACTIVE"), other bed movements are locked out by conflict (so HOB loop is temporarily not allowed, etc.). A timer is started (the maximal effect is around 1 minute into PLRlitfl.com).

3. **Measurement:** The system continuously (or at least at baseline and 1 minute) reads the stroke volume or cardiac output from the connected monitor. It will compare the value pre-PLR to during-PLR. If the **stroke volume increases by ≥10%** (or a similar threshold) during the maneuver, the test is considered positivelitfl.com. If using pulse pressure as a surrogate, a ≥10% rise in pulse pressure can be usedlitfl.com. TraceLoop could compute this in real-time: e.g., baseline CO = 4.5 L/min, during PLR CO = 5.0 L/min is +11% -> positive.

4. **Outcome Logic:** The PLR_TEST factor will set a boolean outcome: e.g., an internal variable plr_responsive = TRUE if positive, else false. This result can be stored or forwarded as a sensor value in the factor table (some systems might treat it as a virtual sensor reading like "FluidResponsive: Yes/No").

5. **Post-PLR Action:** If positive (meaning patient likely will benefit from fluids), TraceLoop can automatically proceed to give a fluid bolus in Full mode. Here is where integration with an **actuator loop for fluids** happens. We might have a factor IV_FLUID_BOLUS that is normally gated until a positive PLR. Using TraceLoop semantics, we can set requires_ok = PLR_TEST on the fluid bolus factor – so it will only run if PLR_TEST yielded positive (TRUE). Additionally, we might have priority_over edges such that if PLR is positive, it takes priority in decision-making for volume expansion over other interventions (for example, over starting a vasopressor, if both were options). The system could then automatically start a measured fluid infusion (say 500 mL over 20 minutes) through the infusion pump (with conflict_group perhaps IV_PUMP_FLUID).

    - If the PLR is negative (no significant SV increase), the system (in Full mode) will refrain from giving a fluid bolus. Instead, it might trigger a different pathway (e.g., if patient is hypotensive but not fluid-responsive, maybe it suggests/increases vasopressors or inotropes). We can implement that logic with another factor or adjusting risk scores such that the vasopressor loop now wins priority since fluid loop is gated off.

    - In effect, PLR_TEST acts as a **gate/qualifier** for fluid therapy. This gating is exactly what the requires_ok field is designed for: we set the fluid-infusion rule to require PLR_TEST. The PLR_TEST factor itself doesn't necessarily have a continuous actuation after the bed movement; it's more of a transient diagnostic action that yields a condition (so its requires_ok wouldn't be used by others, rather its truth value is).

6. **Completion:** After the measurement window (usually ~2 minutes total), the system will return the bed to the previous position (head up again). The PLR_TEST factor releases the BED conflict_group and goes into a standby state (until another PLR is needed).

During this process, arbitration ensures the PLR action doesn't conflict with anything else. Because PLR uses the bed motor, any simultaneous demand for the bed (HOB adjustment or proning) will be queued. According to our conflict_group BED, only one can execute at a time. PLR_TEST would likely have a moderate risk priority – not as high as a life-critical therapy, but since it's short and diagnostic, we want it to execute when invoked without undue delay (perhaps harm_severity = LOW, time_to_harm = n/a because it's not causing harm itself, but we might assign a pseudo-risk or use manual priority boosting to ensure it runs promptly when requested).

In **Lite mode**, TraceLoop will assist the clinician in performing and interpreting a PLR, but not do it hands-free:

- The clinician initiates a PLR test (via UI button or simply manually does it and then marks an event on the UI). The system might provide on-screen guidance: "Performing PLR: ensure head at 0°, legs at 45°". It may even interface with the bed if possible to automate the position (semi-automated).

- The system times the maneuver and fetches the needed data from monitors. It could display a small trend graph of stroke volume or blood pressure during the PLR interval.

- After 1 minute, the system calculates the changes and then **displays the result**: e.g., "PLR Completed: Stroke Volume +15% – Positive. Suggest giving 500 mL fluid challenge." or "PLR Negative – fluid bolus not indicated, consider alternatives." This recommendation would appear on the UI and possibly as part of a decision support notification.

- In Lite mode, the system does not automatically infuse fluids; it leaves the decision to the clinician but provides the analysis. It might highlight a "Give Fluid" button if positive, which the clinician can confirm to start an infusion manually (or via integrated pump if they accept, which blurs the line toward automation but still clinician-initiated).

- If the PLR was initiated due to a suggestion (like an alert "Patient hypotensive, consider PLR"), the system will tie the outcome to that suggestion and either clear the alert (if fluid given or not needed) or escalate to another suggestion (if PLR negative, maybe "Consider starting norepinephrine").

Actuator Integration: In Full mode, the actuator is primarily the **bed motor controls** (for adjusting patient position) and secondarily the **IV fluid pump** (for delivering fluids if positive). The bed control aspect is similar to HOB integration: the software sends commands to set bed angles. This requires the bed's actuators to be addressable (which might be achieved if the bed has an interface for preset positions like Trendelenburg, etc.). Alternatively, TraceLoop could achieve PLR by controlling both head and leg

actuators separately: command head-down and leg-up. This should be done in a coordinated fashion (some beds might have a single command for "leg raise" or one for "Trendelenburg" which is head-down whole-body – but PLR is typically done by bending at hips, not tilting the entire body). The integration should ensure **smooth and synchronous motion** – possibly sending both commands near-simultaneously. There might be a micro-control loop verifying the angles reached (if the bed feeds back current angle, TraceLoop can monitor to know when target is reached and then start the clock for measurement). The **IV pump for fluid** would be engaged only if needed: if the fluid bolus loop is enabled by PLR, it will then have its own dosing logic. We might incorporate a **micro-dose equation** for fluids: e.g., "administer 4 mL/kg over 10 minutes" or a fixed volume with a maximum rate. It could also ensure not to overshoot—perhaps giving in steps: 250 mL then reassess, etc. For safety, if the patient's cardiorespiratory status changes during the bolus (e.g., signs of overload), a guard rail would halt it.

Guard-Rail Layer, Conflicts & Fallback: Several safety considerations accompany PLR integration:

- **Patient Selection Guard:** PLR should not be done in certain cases (e.g., if patient has raised intracranial pressure or spinal injury, PLR could be harmful)littl.com. A guard rule could be implemented such that the PLR_TEST factor has a requires_ok dependency on, say, NO_CONTRAINDICATIONS. This could be an aggregate condition that is false if ICP is high or spinal precautions are active. In practical terms, an ICU team would know when not to do PLR, but the system can provide an extra check (for example, if an intracranial pressure loop is active and critical, PLR loop will not run).

- **Mutual Exclusion:** PLR (bed movement) is mutually exclusive with proning, as well as with any active procedure that requires patient to be still (imagine a situation like an ongoing dialysis or a central line placement – the system should avoid sudden moves). We can mark PLR as mutually_exclusive with PRONING as we did for HOB, and perhaps with a CPR loop if that exists (if CPR in progress, obviously no PLR).

- **Priority Management:** PLR has diagnostic benefit but if a higher priority event occurs (say patient goes into cardiac arrest mid-PLR), the system must abort the PLR immediately. The failsafe monitors (like an code blue detection or a asystole alarm from ECG) would trigger a higher priority factor that preempts bed control. Since the bed is same conflict group, a high priority factor (CPR positioning factor, conflict_group=BED with critical risk) would seize control, possibly flatting the bed for CPR. The scheduler (FIG. 4/7 logic) ensures the highest risk command wins in a conflict group. So PLR would automatically be interrupted. We should design PLR factor to gracefully handle interruption: if aborted, it returns bed to safe position if possible or just stops and flags incomplete test.

- **Data Validation:** The hemodynamic response must be interpreted carefully. TraceLoop's filtering should ensure that the rise in SV is due to PLR and not artifact. E.g., if the patient coughed or had an arrhythmia, the CO reading might spike or drop unrelatedly. A guard could check signal quality (many monitors provide a confidence or variability indicator). If data quality is poor, TraceLoop might extend the measurement time or repeat PLR automatically once more before concluding.

- **Fallback Modes:** If the bed does not respond to the command to raise legs, or the monitor data is unavailable (e.g., CO monitor not connected), the system will fall back to not executing the automated PLR. It will alert the clinician: "Unable to perform PLR – bed control failure" or "Cardiac output data not available, manual assessment required." In such a case, the clinician may perform a manual PLR and maybe observe blood pressure themselves (outside the system). The system essentially steps out and perhaps logs that the attempt failed.

- If PLR is done but yields an indeterminate result (some patients have borderline changes), the system could either categorize it as negative (with a note of uncertainty) or prompt a repeat. A guard-rail could limit repeats (e.g., not do more than 2 PLRs in 15 minutes to avoid patient discomfort or measurement confounding).

- **Conflict Graph Integration:** The PLR factor's existence in the factor graph allows it to coordinate with therapy loops. For example, we mentioned gating of fluid bolus. Also, synergy could be considered: If a fluid bolus loop and a vasopressor loop are both candidates to treat hypotension, and PLR is positive, the fluid loop should proceed and perhaps have **priority_over** starting a vasopressor (since we know fluid is likely effective). Conversely, if PLR is negative, a vasopressor loop might get priority. This can be achieved by dynamically adjusting risk or by encoding two alternative factors: e.g., SHOCK_FLUID vs SHOCK_PRESSOR where PLR's result influences which is allowed. Implementation-wise, a simple way is to have a rule: if PLR responsive, set a context variable that boosts fluid loop risk score a bit and lowers pressor, if not, do opposite. But since the factor table is static at compile-time, the gating approach is more straightforward (fluid loop requires PLR positive; pressor loop could require PLR negative in a protocolized environment). These cross-channel rules ensure no tug-of-war between fluid and pressor – a classic synergy/antagonism situation resolved by explicit logic rather than ad-hoc decisions.

UI Chip Behavior: The PLR functionality will have a representation on the UI, though it might be slightly different from continuous loops. Possibly a **"PLR Test"** button or chip in the hemodynamics section. In Full mode, the chip might show status: e.g., when idle it shows "PLR: Ready". When a PLR is running (either automatically triggered or manually initiated), the chip could indicate "PLR: In Progress…" with perhaps an animation or countdown (since it's a short-term action). After completion, it might briefly display the result "PLR: Positive" or "Negative" with a color (green for positive meaning good response to fluids, or maybe blue; negative could be gray). Because PLR is diagnostic, not an ongoing therapy, the chip might then reset to idle state or show the last result for a while. The user can likely click the chip to view details like the SV values before/after (e.g., "SV increased from 60 mL to 70 mL (+17%)"). In Lite mode, the PLR chip would be more interactive for the clinician. It could have a **two-state UI**: an idle state "Perform PLR" (like a button to start), and a post-test state with the result. If the system cannot auto-adjust the bed in Lite, pressing the button might present instructions: "Please lay patient flat and raise legs. Press Continue when done." The clinician follows that, then the system gathers data and after a minute shows the outcome on the chip. The result might flash or highlight if positive and potentially enable a "Recommend Fluid" action. Perhaps the chip transforms into a prompt: "PLR positive – press here to give fluids" in Lite, whereas in Full it would have done it and just inform "Fluids

administered".

For override behavior, there isn't a traditional "override" for PLR since it's not a continuous loop. However, if the system auto-initiated a PLR and the clinician needs to cancel (maybe they notice a contraindication), they could hit a cancel on the chip. In Full mode, that would send commands to abort and return bed to prior position (like an emergency stop). The chip would likely turn amber or red if an override/cancellation occurred, indicating that the automated test was halted by user (and maybe log that event). The UI also logs the PLR outcome to the patient's record or notes for reference. In summary, the PLR chip is a hybrid of an action button and a status indicator, guiding the user through the maneuver in Lite, and providing transparency in Full.

Figure 12 – Passive Leg Raise Response Integration: *This schematic diagram illustrates the PLR integration. The sequence (annotated by circled numbers) begins with the TraceLoop controller (210) issuing bed position commands (1) to lower the head and elevate the legs (via the Bed Actuator ⓯ in conflict group "BED"). The cardiovascular sensors (e.g., arterial line or NICOM, 902) provide stroke volume data to the TraceLoop rule engine (210) during the maneuver (2). The PLR factor module (marked "PLR_TEST") computes the change in stroke volume (3) and sets a flag if ≥10% increase (positive response). Downstream, the Fluid Bolus loop (in IV Pump conflict group) is gated by the PLR flag (4): if positive, the fluid infusion command (242) is released to the IV pump actuator; if negative, that command path stays blocked and alternative therapy loops (e.g., vasopressor 244) may take priority. The diagram also shows the UI indicator (on the bedside interface 220) updating the clinician about the PLR status and result throughout the process (5). This integration leverages the core TraceLoop arbitration engine (FIG. 4) to ensure the bed movement and fluid actuation occur in a safe, coordinated manner, with all events logged to the audit chain (FIG. 9).*

Implementation Effort: The PLR integration is **moderately complex** because it spans sensing, actuation, and logic. On the software side, implementing the logic to calculate responsiveness is straightforward arithmetic, but integrating two data sources (bed and hemodynamic) and timing them requires careful synchronization (the system needs to mark baseline and follow-up times, which involves some state machine in code). Writing the control for bed actuators is similar effort to HOB (likely using the same bed interface, so once that is done, adding PLR is incremental). The bigger lifts are testing and ensuring reliability of detection. If interfacing with monitors via HL7, some development is needed to parse and fetch the specific parameters (like identifying which HL7 message field is stroke volume – this could vary by vendor but is often standardized via IEEE 11073 nomenclature or similar). If the monitors aren't already feeding TraceLoop, that integration must be done (which might benefit other loops too, so it's a shared effort). The risk engine and factor table require new entries: one for PLR, one for fluid bolus if not existing. That's mostly configuration but ensuring the cross-rule gating in the DAG is correct and doesn't introduce cycles (the compile-time validator will check for cycles). We must ensure PLR_TEST doesn't depend on something that depends back on it. Given our design (PLR triggers fluid but fluid doesn't trigger PLR), it's acyclic. The **device-side effort**: if the bed and monitors are modern and can integrate, it's more about software integration. If not, one might need to deploy additional sensors – e.g., if no continuous CO monitor is present, one might consider noninvasive options (like a portable ultrasound linked to system – but that's highly advanced and likely outside this scope). Ideally, a minimally invasive monitor is assumed. Testing will involve trying PLR in various conditions (with known outcomes) to fine-tune threshold and timing. *Full mode* code must handle real-time data during PLR (maybe using a buffer of the last 5 cardiac cycles to average stroke volume). This real-time aspect is

a bit heavier than typical threshold rules, but within capability (perhaps using a small routine called by the 10 ms engine cycle to update PLR state – which is fine since 1-min timescales are large relative to 10 ms). *Lite mode* mainly requires UI and some guidance logic – comparatively easier once the data is there. One important aspect is **user training/communication**: The clinicians must trust the PLR automation. The spec should eventually include user overrides and ensure any automated bed movement gives a clear alert (no surprises to a caregiver in the room). Given these considerations, the PLR integration is a medium lift feature, enhancing TraceLoop's decision-making for fluid management significantly by adding an evidence-based test for fluid responsivenesslitfl.com.

Train-of-Four (TOF) Count/Ratio (Neuromuscular Blockade Monitoring)

Supported Feed Formats & Sources: Train-of-Four (TOF) monitoring is used to assess the depth of neuromuscular blockade by delivering four nerve stimuli in quick succession and measuring muscle response. The outputs are typically the **TOF count** (0–4 twitches) and the **TOF ratio** (ratio of the fourth twitch to the first twitch amplitude). Sources for TOF data in an ICU:

- **Bedside Neuromuscular Monitors:** Many ICU patient monitors or stand-alone devices (like TwitchView, TOF-Watch, or others) provide quantitative TOF measurements. These devices often connect to the patient's ulnar nerve (stimulation electrodes) and have an accelerometer or electromyography sensor on the thumb to detect twitches. Modern systems output the TOF ratio as a percentage or decimal (e.g., 0.6 or 60%). Integration can be via **HL7 PCD** messages from the patient monitor if the monitor has a module for TOF (for example, some monitoring systems will include TOF ratio in their vitals data stream). If a standalone device is used, it may output data via a serial port or USB (which could be parsed) or even via Bluetooth to a tablet.

- **Serial/Analog Data:** Older devices might provide an analog output (like a voltage proportional to twitch force) or require interpretation of lights. However, devices like TOF-Watch SX can connect to a computer. In absence of a direct digital feed, a custom microcontroller could theoretically stimulate and read an accelerometer – in fact, TraceLoop's design listing "Peripheral Nerve TOF – Thumb accelomyograph" suggests an in-house or integrated approach. This would be an embedded solution: an MCU triggers nerve stimulation (via a peripheral nerve stimulator circuit) and reads the acceleration of the thumb (via an accelerometer sensor). The result is computed onboard and fed to TraceLoop's main controller (e.g., via CAN bus or analog input).

- **BLE wireless sensor:** There are emerging wireless EMG patches or accelerometers for TOF that could send data via BLE. For example, a small wearable on the thumb that connects to a central station. If available, TraceLoop could get that data through a BLE gateway service.

- **MQTT/Networked devices:** If the TOF device is network-capable (like sending results to a central system), it might publish to a topic or have an API. This is less common, but not

impossible with newer tech (some anesthesia monitoring solutions allow network integration).

- **Manual entry:** In low-tech scenarios, a clinician might perform a TOF using a handheld stimulator and visually/feel the twitches, then enter the count into the system. While not ideal for continuous closed-loop, TraceLoop Lite could allow manual input of TOF count as a data point.

Payload Schema: The TOF feed payload typically includes:

- `TOF_count` – integer 0 to 4 indicating number of twitches observed.
- `TOF_ratio` – numeric (0.0–1.0 or 0–100%) indicating the ratio of the fourth to first twitch amplitude. This is a finer measure of blockade degree when some twitches are present (if count=4).
- `stimulation_current` – (optional) the mA used for stimulation (to ensure supramaximal stimulation; usually fixed, e.g., 30 mA).
- `timestamp` – when the measurement was taken. Some systems measure TOF periodically (e.g., every 15 minutes) or on demand. For closed-loop, more frequent (e.g., every 5 minutes or continuous algorithmic derivation from nerve stimulator can be used).
- Possibly `device_id` if multiple or for logging calibration, but not critical.

In TraceLoop's factor table, we might have a factor `TOF_RATIO` (the measured value) or a factor representing the paralysis control loop. Likely, we will have a factor that represents the control of a neuromuscular blocking agent (NMBA) infusion, which *uses* the TOF measurement as input. For clarity, define `TOF_MONITOR` as a sensor factor (no direct actuator, just feeding data) and `PARALYSIS_CONTROL` as the factor that actually controls the drug pump. However, since the question lists TOF count/ratio as a feed to integrate, we focus on how that feed is utilized in TraceLoop logic.

The **conflict_group** for the controlling loop will be the infusion pump delivering the paralytic (e.g., if cisatracurium infusion is on pump channel IV_PUMP_4, conflict_group might be `IV_PUMP_4` or a more generic `NMB_PUMP`). The sensor itself (TOF monitor) doesn't occupy a conflict_group except we may treat it as part of an overall ventilator support group logically. But since it doesn't actuate, conflict_group for a pure sensor factor could be a dummy or omitted – however, every factor row requires one. We might tie the sensor factor to the same conflict_group as the pump for simplicity or to a "MONITOR" group. The more straightforward approach is to have a single factor that encompasses both sensing and actuation ("maintain TOF at target by adjusting infusion"). That factor would have the pump's conflict_group. We will describe as if one factor does the closed-loop control, since the TraceLoop architecture allows a factor to have sensor input fields and to produce an actuator command.

Control Logic Path (Full vs. Lite):

In **Full closed-loop mode**, the integration of TOF aims to automate titration of neuromuscular blockade (paralytic drugs) to a desired level and ensure safe reversal:

- **Target Setting:** First, decide the target TOF level. This could be scenario-dependent. For a fully paralyzed patient (e.g., severe ARDS to facilitate ventilation), target might be 0/4 twitches initially. For lighter sedation or maintenance, target might be 1–2 twitches or a TOF ratio perhaps around 0.1–0.4 (10–40%). Often, an ICU strategy is to maintain 1–2 twitches out of 4 for adequate paralysis but not excessive drug. We will assume a target of 2/4 or 3/4 twitches for moderate blockade, unless deep paralysis is explicitly needed. The factor can have this target encoded or dynamically set by another input (like a physician order could adjust it).

- **Closed-loop Control:** The PARALYSIS_CONTROL factor reads the current TOF ratio. If the ratio is **below target** (meaning the patient is more paralyzed than needed, e.g., TOF ratio 0.0 and target 0.3), it will **decrease or pause the NMBA infusion** to allow some recovery. If the ratio is **above target** (patient regaining muscle function, e.g., TOF ratio 0.8 when target is 0.4 during an ongoing case where some block is desired), it will **increase the infusion rate** or give a bolus if needed to deepen the block. Essentially, this forms a feedback loop controlling drug rate based on TOF error. A simple control algorithm might be:

    - If TOF count < target count: possibly do nothing or decrease infusion slowly to avoid overdosing (since already 0 twitches means fully paralyzed).

    - If TOF count > target: increase infusion to achieve more blockade.

    - If TOF ratio exists and target is expressed in ratio, use that proportionally.

- TraceLoop may implement either a rule-based step controller (like incremental changes) or a more continuous PID-like controller. Given the system, a rule table could be encoded, e.g.:

    - "If TOF count = 4 (full muscle function) and paralytic is ordered, then give bolus or increase rate significantly."

    - "If TOF count = 3, slightly increase rate."

    - "If TOF count = 1, perhaps decrease rate slightly (as patient is quite blocked)."

    - "If 0 twitches (TOF count 0), consider stopping infusion until some twitch returns (to avoid accumulation)."

- Fine control would use the ratio: e.g., target ratio 0.9 at end of therapy (for extubation readiness, guidelines say ensure TOF ratio >0.9 before extubationxavant.com). During maintenance, target ratio might be lower. The factor can smoothly adjust infusion within safe bounds.

- **Factor Table Entries:** The PARALYSIS_CONTROL factor would have conflict_group = the pump delivering NMBA. It might have a requires_ok from something like a sedation factor (discussed below) and possibly a priority_over or synergy relation with ventilation. For example, *requires_ok = VENTILATOR_CONTROL* could ensure paralysis only active if ventilator is in controlled mode (you wouldn't paralyze if the vent isn't providing breaths). This might be too granular, but logically, usually one only paralyzes when on mechanical ventilation, which in a closed-loop might equate to requiring that the ventilation loop is active. We can also incorporate *synergy*: sedation and paralysis loops should run together, as sedation must deepen when paralysis is on (to ensure no awareness). So PARALYSIS_CONTROL could have synergy_with = SEDATION_CONTROL meaning schedule them together (the engine could co-emit commands to sedation and paralysis to keep timing aligned).

- **Interaction with Extubation readiness:** A critical aspect is ensuring that when we want to wean or extubate the patient, the paralytic has worn off sufficiently (TOF ratio > 0.9 criterionxavant.com). So, when the system or clinician decides to stop paralysis (for example, during daily sedation interruption or SBT trial), the PARALYSIS_CONTROL factor will switch to a weaning mode. This means turning off the infusion and monitoring the TOF ratio rise. The factor might even trigger **reversal agents** if needed (like suggamadex or neostigmine) – though in ICU, suggamadex could be used if available. If we had a reversal loop, say NMB_REVERSAL, that might be invoked if TOF ratio isn't reaching 0.9 in a needed timeframe. That could be set as a gating: e.g., an extubation factor requires TOF_MONITOR to be >0.9. If not, it could either delay extubation or automatically give reversal (depending on automation level).

- **Full automation details:** For the drug infusion, we need to ensure safe limits (the factor will incorporate guard-rails for min/max infusion rate based on protocol). Also, do not overshoot: the system should avoid 0 twitches for too long if not needed; at the same time avoid inadequate blockade if indicated (prevent patient movement if it could be dangerous, like during delicate surgeries or severe ARDS where movement can cause harm). TraceLoop's risk scoring would assign a risk to inadequate paralysis (could risk ventilator asynchrony) vs overparalysis (risk of prolonged weakness). These might be moderate-level and balanced.

In **Lite mode**, the system will monitor TOF and provide guidance to clinicians on how to adjust the NMBA dose:

- It will display the current TOF count/ratio in real-time on the UI.

- It may have rules that trigger suggestions: e.g., "TOF 4/4 – patient not paralyzed, consider increasing cisatracurium infusion" or "TOF 0/4 for 30 min – consider reducing paralytic dose to avoid overdose."

- If extubation or SBT is planned, the system can alert "TOF ratio only 0.5 – recovery inadequate for extubation; recommend waiting or giving reversal agent"xavant.com.

- The actual adjustment of infusion is left to the clinician: either they push a button that the system might provide ("titrate up 20%" etc.) or they do it at the pump and maybe log it.

- The factor table for Lite mode still has entries (maybe with auto_execute off). For example, it might still compute an optimal dose in the background but not apply it, instead showing it as a recommendation ("Suggested rate: 8 mg/h").

Actuator Integration (Full): The key actuator is the **infusion pump** for the neuromuscular blocker (e.g., cisatracurium, vecuronium infusion). TraceLoop would interface with the infusion controller to set the rate. This could be the same mechanism it uses for other IV pumps: likely a digital command over a pump integration (HL7 PCD message or a proprietary pump SDK). Micro-dosing: Typically, NMBA infusions are continuous, but the concept of micro-dosing here might mean incremental adjustments. For safety, any automated increases might be done in small steps (e.g., increase by 5–10% at a time, then recheck TOF after a few minutes, because these drugs have some lag). The system could also administer a **bolus** dose if the situation requires rapid paralysis (for example, if a sudden need arises to secure airway or in OR scenarios). Rate envelopes: ensure infusion does not exceed max recommended (like no more than X μg/kg/min or mg/h per protocol). The system might incorporate patient weight or use the dose per weight in calculations. Also, if using an agent like atracurium or cisatracurium which have spontaneous degradation, the system might have to account for infusion durations and metabolite buildup (though that is advanced detail). The closed-loop essentially functions as a PID controller with the TOF ratio as feedback. Gains could be tuned based on the reversibility window column in FIG. 11 (the concept of faster or slower control response based on how quickly an overshoot can be reversed). For example, since deep paralysis is reversible with an antidote (sugammadex for aminosteroids or just time for others), the system might allow a slightly aggressive control knowing reversal is possible, but not too aggressive to avoid hemodynamic side effects.

Actuator integration also includes possibly a **stimulator** if TraceLoop directly performs nerve stimulation. If using an internal solution, the TraceLoop hardware would need a small module to generate the train-of-four stimulus pulses (usually 2 Hz train, 4 pulses of 0.2 ms duration, 0.5 seconds apart). That module would then measure response. However, that level of hardware integration is hinted by the "Analog Aux → TOF (1)" in the TraceLoop MX bus layout. If indeed implemented, the TraceLoop device itself would handle the entire TOF measurement cycle, making the feed essentially internal. Then the infusion adjustment is done by the main controller.

Guard-Rails, Conflict-Graph Interactions, and Fallback:

Safety is paramount because paralyzing a patient carries risks (awareness if not sedated, prolonged weakness, etc.).

- **Sedation Link (Requires/Mutual):** A guard condition should enforce that adequate **sedation is in place whenever paralysis is active**. This could be done via a `requires_ok = SEDATION_OK` on the paralysis loop. For instance, require that the sedation loop (propofol/remifentanil etc.) is active and at sufficient level (maybe defined as BIS below a threshold or RASS target met) before NMBA infusion is allowed. If sedation fails (e.g., infusion stops or BIS indicates patient waking),

TraceLoop should automatically pause NMBA infusion to avoid "awake paralysis." In practice, sedation and paralysis factors would be coupled: possibly marked as mutually_exclusive with something like an awake state or SBT state. One could also implement synergy: sedation and paralysis loops should be scheduled together and perhaps share priority.

- **Ventilator Synchrony:** The paralysis loop has a synergy with ventilator control because it eliminates patient effort. If the ventilator loop is trying an SBT (which requires the patient to breathe on their own), then paralysis is counterproductive and should be off. Therefore, when entering SBT readiness, the system must stop paralytics. We can enforce that by mutually_exclusive between the SBT trial factor and the paralysis factor, or by gating: e.g., requires_ok = NOT_SBT on paralysis or conversely, requires_ok = TOF_ratio>0.9 on SBT factor meaning it won't run if patient is still significantly paralyzed. In summary: SBT readiness factor will only become true if no blockade. The conflict graph should also ensure that if somehow both tried, the higher priority wins (likely SBT since it's about liberating from ventilator, would have high risk if done inappropriately).

- **Extubation Safety:** Before extubation, guidelines strongly recommend TOF ratio ≥ 0.9 (90%)xavant.comanesthesiologynews.com. TraceLoop could make this a **hard requirement** for an extubation or ventilator liberation factor. That is, an extubation loop might have requires_ok = TOF_RECOVERED (where that is true if TOF_ratio >=0.9). If that is not met, the extubation loop cannot proceed (the system would instead throw an alert and hold off). This is a guard-rail to prevent removing the breathing tube while patient may have residual paralysis, which can cause ventilatory failure. The integration of this with sedation pause (which often goes along with extubation trials) must be orchestrated carefully (the usual approach: stop NMBA infusion well in advance, let TOF recover, then do SBT with sedation off).

- **Drug Interaction Conflicts:** If multiple loops use the same pump or drug class, conflict_group covers it. Usually only one NMBA infusion is used at a time, so not an issue. But conflict might arise if one had both infusion and bolus on same channel – the system would schedule them appropriately (the infusion factor might temporarily pause infusion if a bolus factor (like a code dose of NMBA) fires, etc.).

- **Override / Fail-safe:** If for some reason the patient needs to be urgently reversed or if the loop malfunctions, a clinician can hit an override: BLOCK the paralysis factor (stopping infusion) and possibly manually give reversal. The system should immediately comply. Also, the watchdog (FIG. 8) monitors if pump commands are executed. If the pump doesn't stop when commanded (dangerous, infusion continuing), the system would escalate an alarm and could trigger hardware interlocks if available (like an emergency stop on pumps if integrated). Another fail-safe: if communication with TOF sensor is lost, the loop should not blindly continue high infusion – it should freeze or slowly titrate down and alert that it's running open-loop. Typically, losing feedback in a closed-loop should default to a safe output (probably maintaining last known good rate or a conservative lower rate).

- **Conflict resolution with other loops:** The paralysis loop has a certain harm severity attached. Over-paralysis can cause prolonged ICU weakness (myopathy) – that's a harm but not immediate. Under-paralysis can cause ventilator asynchrony or patient-ventilator dysynchrony or even accidental movement. The risk of the latter could be moderate in ARDS. So risk scoring might be moderate and time_to_harm might be minutes to hours. It will likely not override critical loops (like it won't override a blood pressure critical loop or an arrhythmia loop), but within its conflict_group (the pump) it's the only one typically. If the same pump channel is also used for sedation (hopefully not, they'd be separate pumps usually), they'd have separate conflict groups.

- We should mention that in the factor table sample, they even list an example: ensuring synergy with QT prolongation loop or others in some context. For neuromuscular, synergy with sedation was mentioned. We can also consider synergy with cooling (as hypothermia can affect TOF readings, but that's too detailed perhaps).

UI Chip Behavior: On the UI, the neuromuscular blockade loop would have a tile showing the current state of paralysis and allowing oversight:

- It might be labeled "Neuromuscular Blockade" or show the drug name e.g. "Cisatracurium". It would display **TOF count/ratio** prominently, since that's the key feedback. For example, "TOF: 2/4 (50%)" displayed in text.

- In **Full mode**, if the loop is active and maintaining paralysis, the chip would be green (normal operation) and might show an icon of a muscle or a paralysis symbol. The background might subtly indicate target vs actual (maybe a gauge or bar). The clinician can interact: for instance, if they want to lighten paralysis, they might reduce the target via the UI (e.g., set target to 3/4 twitches, effectively commanding the loop to allow more recovery). This would be a controlled parameter adjustment rather than an override. If the clinician hits an **override** (like a BLOCK), the chip might turn amber or red (depending on if it's safe state or if an alarm state). Amber might indicate "Paralysis loop paused by clinician". During that time, the system would cease infusion (the chip could show "HOLD" or something). They might do this if they suspect the patient is awake or for neurological exam.

- As the patient recovers neuromuscular function, the chip could change state. If the plan is to stop paralytics, once infusion is off, the chip might show progress of TOF ratio rising. It could highlight when threshold 0.9 is passed (maybe turning the chip border a different color or giving a checkmark "Recovered").

- If extubation is attempted while chip still shows <0.9 ratio, the UI would likely warn or even prevent confirming that in the interface (if integrated with extubation checklist).

- In **Lite mode**, the chip serves primarily as a monitor display and suggestion panel. It would show the same TOF data. Possibly color-coded: red if TOF ratio is low (meaning deep block) but maybe sedation issues? Actually, maybe use color to show compliance with target: e.g., blue when at target, yellow if above (not enough block) or below (too much block) target. In Lite,

there's no strict "target" since user is controlling, but the system can assume typical target of ~2 twitches and color accordingly.

- The chip could also show recommended actions in text or icon form. For example, a small up-arrow if it thinks you should increase dose, down-arrow if decrease. Clicking the chip might give more info: "TOF 4/4 – suggest increasing infusion by 0.5 mg/h to achieve 1-2 twitches." Or "TOF 0/4 for 1hr – consider holding infusion to allow some recovery."

- The user can also set the goal on the UI in Lite (e.g., "Desired twitches: 2/4"), so the system can calibrate its suggestions to that goal.

- Additionally, the UI can integrate sedation with this chip: perhaps show a lock icon if sedation is running, and warn if sedation is not present. For instance, if sedation chip shows RASS -1 (light sedation) but paralysis chip indicates paralyzed, the system should flash a warning across both (this might be an integrated alarm, or the paralysis chip could turn red with a message "WARNING: Patient may be awake while paralyzed!").

- For SBT readiness, the sedation and paralysis chips might coordinate: the SBT readiness chip (described later) would likely indicate "Cannot start trial: patient paralyzed" and that could reflect on the paralysis chip too.

- Logging: the chip's detail view would show a graph of TOF ratio over time, infusion rate over time, etc., to help clinicians see the trend (like whether blockade level is stable or accumulating).

- The override dial (if using the same override mechanism for all loops) might not be heavily used here except to pause. If a clinician wants to temporarily deepen block (like a manual bolus), they could either use the pump directly or perhaps a "Boost" button on UI if allowed. That would effectively override the normal logic for one cycle by forcing a higher infusion for a short period (something that could be done using the override FORCE command for one cycle).

Implementation Effort: Integrating TOF involves both hardware and software aspects:

- *Device integration:* If the ICU monitors provide TOF via HL7, it's relatively straightforward to parse and use (assuming HL7 messages with OBX segments for TOF ratio). If not, implementing a custom sensor like the accelerometer approach is more complex – requiring hardware development, which the TraceLoop MX design hints might already be planned (with analog input for TOF accelomyograph). Getting that calibrated (ensuring the system can measure twitch height accurately) and robust against noise is a significant engineering task. For a first integration, using existing commercial monitors would be easier.

- *Software logic:* The closed-loop control algorithm needs careful tuning. It's more complex than threshold rules (like Anti-Xa's straightforward triggers) because it's a continuous feedback loop that can oscillate if not tuned (e.g., too high infusion could overshoot 0 twitches for too long, too

low could cause oscillation of twitches coming back). The documentation suggests use of dynamic tuning via reversibility_window concept – we might leverage that to adjust how aggressively to dose (since paralysis is quickly reversible with drugs like sugammadex for aminosteroid NMBAs, one could classify it as a "short reversibility" scenario and thus allow faster changes, if that logic is implemented).

- *Integration into the factor graph:* We add the new factor row(s) with columns filled for conflict_group (pump), etc. We'll need to set up cross-channel edges (e.g., with sedation, SBT, extubation factors). Ensuring no cycles: sedation might require vent, vent requires not paralyzed for SBT – we have to be cautious but generally it should be acyclic. Likely sedation and paralysis loops don't form cycles because one doesn't require the other in both directions (maybe sedation doesn't require paralysis, only paralysis requires sedation).

- *Testing & Safety:* This integration touches patient safety intimately (over-paralysis and under-sedation can be catastrophic). So testing will be heavy: unit tests for logic (e.g., simulate scenario: TOF ratios changing, see if infusion commands follow correctly, etc.), simulation with pharmacokinetic models perhaps to verify stability, and of course real-world trials under clinician oversight. The regulatory burden is significant as it edges into full closed-loop anesthesia domain (which typically demands lots of validation).

- *Lite mode development:* is simpler (just UI and suggestion rules) and could be done first to gather data. Possibly TraceLoop could run in advisory mode collecting TOF and infusion data for a while to build confidence before flipping to auto mode.

- Code-wise, implementing a multi-step rule (like a little state machine for blockade depth) might require extending the rules engine if it usually handles one-off triggers. However, the engine may allow continuous evaluation where at each cycle it computes a risk or score. For example, it could assign a "risk" number proportional to deviation from target block, to feed the arbitration. If patient is under-paralyzed vs target, risk of movement could be calculated and influence the priority. This numeric approach might complement the direct PI control by making the loop more or less urgent. Some of this might be outside the current engine's typical use (which is geared to selecting which loop fires, rather than continuous adjustment – though it can issue repeated commands and adjust intensities as part of the factor's parameters).

- Interfacing with pumps: must ensure the pump integration can handle frequent adjustments (some pump interfaces might have rate change latency or limitations).

- **Summary:** *Full integration* of TOF is a **high effort** because it's essentially adding a closed-loop control feature that is continuous and must coordinate with sedation and ventilation. *Lite integration* is **medium effort**, mostly interfacing and UI, since we can skip actuating and focus on data and recommendations. Device hardware integration can vary from low (if using HL7 from existing monitors) to high (if building a custom accelerometer solution). However, given that the TraceLoop MX concept lists TOF as one of the analog channels, much groundwork may already exist. We would cite, for example, that the system supports a thumb accel sensor for TOF

ratio and classifies it under neuromuscular block depth, confirming the architecture is prepared for this feed. Once implemented, this integration directly addresses anesthesia safety guidelines (ensuring TOF >0.9 at extubationxavant.com and avoiding residual blockade), which is a strong value-add for TraceLoop's comprehensive patient management.

Proning Position and Duration

Supported Feed Formats & Sources: "Proning" refers to turning a patient into the prone (face-down) position, often for extended periods to improve oxygenation in ARDS. Integrating proning status into TraceLoop involves knowing:

- **Position (Prone or Supine)**: This can come from sensors or system inputs. Modern ICU beds that support lateral rotation therapy or automated proning (e.g., specialty beds like Rotoprone) often have internal sensors or modes indicating patient orientation. If using such a bed, an interface (likely proprietary, possibly via a network or serial link) could provide a data point "Position = PRONE/SUPINE". Alternatively, an accelerometer on the patient (or bed) could deduce orientation. A small wearable IMU (inertial measurement unit) on the patient's chest could detect if the patient is facing down (orientation relative to gravity). This could transmit via **BLE** to the system. Another approach is using the bed's angle sensors: if the bed can tilt laterally or has rotation, those values might indirectly indicate prone (but usually prone is a manual process on a regular ICU bed, which might not have a sensor for "patient flipped").
  There's also the possibility of a manual input by clinicians: they could press a button in TraceLoop "Patient turned prone now" – essentially logging the event. That's less automatic but ensures the system is aware.

- **Duration Tracking**: Once prone, the key metric is how long they have been prone and ensuring they get flipped back after a certain time (commonly 16 hours prone, 8 hours supine in 24h cyclesess.nychhc.orgics.ac.uk). The system can start a timer when position=prone and keep track.

- **Bed Integration**: If using an advanced bed, it might actually perform the proning rotation or at least support it. For integration, some specialized beds might have an API call to start prone rotation (though typically prone positioning is manual with a team, aside from specialty devices). However, some lateral therapy beds tilt 40° which is not full prone but some beds (like Rotoprone) encapsulate and rotate the patient. If such a bed is integrated, it likely has a control system we could potentially interface with (maybe via a network command or physical remote). For safety, likely not automated by external systems currently, but we can envision future integration.

Given current practice, we assume manual proning by clinicians, with sensors to detect the state. So the feed is basically *patient orientation* plus some timer context. Data might come via **HL7** if hospital staff

document in EMR ("patient proned at 10:00") and that flows to TraceLoop (less real-time though). More directly, a connected bed might send an HL7 PCD event or alarm when prone positioning is engaged (some beds might have a mode selection that could be relayed). **MQTT** could be used if a custom orientation sensor publishes orientation data (similar to fall detection IoT devices, but repurposed). **RS-232** could apply if an older specialty bed's control unit can be accessed via serial for status.

Payload Schema: The proning feed can be structured as:

- `position` – a categorical value: e.g., `PRONE` or `SUPINE` (or possibly `LATERAL` if needed to capture intermediary, but primarily these two).
- `timestamp` – when the position was last changed (used to calculate duration).
- `session_duration` – an internally computed field indicating how long in the current position (the system can compute this from timestamp).
- `proning_order` – (optional) could be a flag if proning is prescribed for this patient, or a target daily duration (e.g., 16h).
- Possibly `rotation_angle` – if using a bed that continuously rotates, angle in degrees might be provided, but for our purposes just knowing prone vs supine is enough.
- `interruptions` – if the patient was supined briefly for procedures and then re-proned, the system might track that. But that's more of a log detail.

In the factor table, we can have a factor `PRONING` that represents the state of being prone as a binary state factor. Alternatively, it could be modeled with two factors: one for the action of turning prone (like a procedure factor) and one for monitoring the state/duration. But a simpler representation is a factor that is true when prone and false when supine. Since factors typically represent a loop, we can consider proning as a "therapeutic intervention" in itself (improving oxygenation). So `PRONING` could be treated akin to a loop that "runs" while patient is prone.

Conflict_group: If we consider the act of proning as using certain equipment or requiring certain resources, we might not have an actual device actuated by TraceLoop (unless a specialized bed). If not automating the turn, the conflict_group could be a dummy or something like `BED` again (since it involves bed orientation). However, we already used `BED` for HOB angle and PLR. We can reuse `BED` here as well because clearly you wouldn't want an automated bed movement to raise head or legs while in the process of proning. Marking them same conflict_group ensures no simultaneous contradictory bed moves. If proning is manual and not machine-actuated, one might think conflict_group isn't needed. But to keep the factor in schema, and to logically exclude other bed motions, we set conflict_group = `BED`. Then in Full automation, if a specialized proning bed exists, issuing a proning command would definitely use the bed actuators (maybe multiple motors coordinating tilt and rotation), so that fits.

Control Logic Path:

In **Full integration** (somewhat hypothetical since full automation of proning is rare outside specialized beds, but we describe possibilities):

- The system will decide when proning is indicated. Typically, guidelines say for severe ARDS with PaO2/FiO2 < 150, proning should be done for 12-16 hours/daynejm.orgics.ac.uk. So TraceLoop's respiratory loops (like an oxygenation loop) could trigger a proning action if oxygenation is poor despite high FiO2 and PEEP. We could implement a rule: e.g., *if P/F ratio < 150 and not already prone and no contraindications, then initiate proning*. This would be a trigger for the proning factor to turn "on" (true).

- Once the decision is made, if the bed is integrated and capable, TraceLoop could command the bed to rotate the patient to prone. In reality, such an action is complex and dangerous to automate without human supervision (lines can tangle, etc.). So perhaps Full mode here still requires human assistance: maybe the system issues a prompt "Begin proning now" to clinicians and monitors the process. Or if using a specialized proning bed (which encases the patient), the system might simply send a start command to that device and then rely on it to do the turn safely.

- For this spec, assume an advanced scenario: The bed can slowly rotate the patient 180° to prone when commanded, and TraceLoop can control that. Then the proning factor would lock the bed conflict_group for the duration of the procedure. It might have a substate for "in transition" versus "fully prone".

- Once prone, the factor PRONING is marked active. The system notes the start time.

- While prone, certain other adjustments are often made (e.g., adjust ventilator because prone positioning often improves oxygenation, possibly lower FiO2 needed; also pause tube feeds to reduce aspiration risk while turning, etc.). TraceLoop can coordinate these: For example, it might have synergy commands like automatically pausing enteral feeding pump (actuator action in conflict_group FEED_PUMP) when proning startsess.nychhc.org. This could be achieved by cross-factor rules: proning factor could have priority_over feed loop or directly command a pause.

- The system then needs to track duration. It could set a goal like 16 hours. There may be a countdown or timer in the logic (the factor can have a parameter of desired prone duration).

- After the duration, TraceLoop would prompt to supine the patient. In an automated bed scenario, it could command the bed back to supine. In manual scenario, it would alert staff "Proning session complete – time to return to supine."

- The factor then flips off once supine is achieved, and possibly starts a rest timer (like ensure 8 hours supine before next prone).

- If oxygenation worsens while supine, it might trigger another proning earlier. The loop could thus be iterative daily.

During proning, TraceLoop has to manage conflicts:

- As discussed, HOB angle enforcement loop should be off (which is fine if we set them mutually_exclusive or if proning simply has higher priority on the bed).

- PLR would be off limits in prone (can't raise legs easily in prone; plus typically not done).

- Many closed-loop actions might need retuning in prone (hemodynamics can shift; the system might need different threshold for blood pressure or sedation because prone patients often need deeper sedation). However, these are fine details often handled clinically rather than automated. But one can foresee synergy: sedation needs might increase when prone to tolerate it, so sedation loop could get a different target RASS when proning factor is true (this could be encoded by a rule like if PRONING true, then sedation target deeper – which could be done by synergy or gating on a different mode).

- Ventilator adjustments: some ventilator closed-loop might adapt (prone typically improves oxygenation by recruitment, so maybe vent FiO2 could be weaned faster or PEEP adjusted). The system could notice improved oxygenation and reduce FiO2 accordingly. That's just the usual vent loop doing its job, but it will see risk scores drop for hypoxemia and could then down-regulate support.

In **Lite mode**, since automation is minimal for proning:

- TraceLoop will act as a **scheduler and logger** for proning sessions. It can recommend proning when criteria are met: e.g., "Recommend proning: P/F 120 and patient eligible." The clinicians then manually execute the proning maneuver (with their team).

- The clinician can then input "Patient is now prone" (or the system detects via sensor).

- The system starts a timer visible on the UI (like a countdown or count-up for how long prone). It likely also issues reminders at certain times (like "16 hours prone reached – consider returning supine").

- If the staff forget or exceed safe duration, an alert would fire. (Prolonged proning beyond 16h continuously might increase pressure injuries, etc., so typically they flip at ~16h).

- The system can also assist by pre-checking before proning: e.g., ensure hemodynamics stable, all infusions secured, etc. It could present a checklist (maybe beyond current scope, but could).

- While prone, TraceLoop might adjust its alarm thresholds (e.g., blood pressure or heart rate changes might be expected; some sensors might not work well in prone like back electrodes). It might silence certain non-critical alerts like "ECG lead off" if known to happen.

- Documentation: Lite mode could auto-record the times in a log for ICU records.

Actuator Integration (Full):
If we have a specialized proning bed:

- That bed likely has internal safety and will only operate with certain confirmations. If it's integrated, TraceLoop would send a command like "Start Prone rotation" and then possibly monitor bed data (like angle) until a "prone achieved" status is returned.

- Actuator conflicts: that bed might have its own conflict rules (e.g., it might pause the ventilator briefly or coordinate with it, etc., but presumably ventilator stays on delivering breaths throughout).

- Another "actuator" concept: turning off certain devices: As mentioned, feeding pumps should be off during turning to avoid aspirationcss.uvchhc.org. TraceLoop in Full could automatically stop the feeding pump via its interface at proning start, then resume it after settled (maybe after 1 hour if tolerated, as protocols say hold feed shortly during turn).

- Similarly, it might ensure mattress adjustments are appropriate (some beds have features like turning pressure relief off during proning).

- If we imagine a future robot-assisted proning, that would be an actuator scenario, but far-future.

Guard-Rails, Conflict-Graph, Fallback:

- **Safety Checks (Pre-Prone):** Typically, proning has contraindications or precautions (unstable spine, open abdomen, etc.). TraceLoop might not fully know those, but it could incorporate some (like if a cervical spine precaution factor is active, it should not allow proning suggestion). We could model that as requires_ok = NO_SPINE_INJURY or something on the proning factor. Also, if high dose vasopressors, proning can cause instability; system might warn but not necessarily block.

- **Monitoring During Prone:** A big risk is hemodynamic collapse when turning. The system should be vigilant: during the turn (perhaps indicated by a transitional status), if severe hypotension or arrhythmia occurs, it should alert/stop (if automated bed, stop rotation). The conflict with hemodynamic loops: While turning, blood pressure reading may be transiently unreliable. The system might suspend certain non-critical alarms to avoid a flood of alerts (like artifact signals). Or it might enforce more frequent blood pressure readings after turn (like instruct

an arterial line flush, etc).

- **Mutual Exclusions:** We already set with HOB and PLR. Also obviously, cannot do CPR or major procedures easily while prone – but if CPR need arises (cardiac arrest in prone), often they supine immediately. The system could detect a code alarm and prompt to supine if prone (maybe beyond automation, but at least to notify).

- **Priority:** In conflict_group BED, proning should likely override other bed adjustments, as it's a complex process that, once started, should finish without interference. So proning factor should have a higher priority than HOB or PLR. We can give PRONING factor a risk profile to ensure that: perhaps moderate severity (for hypoxemia it addresses) and immediate urgency when triggered.

- **Duration Guard:** The system should enforce or at least advise the proper duration. If prone time exceeds, escalate alerts (maybe to higher-level staff if not addressed, per protocol).

- **Fallback:** If using automated bed and something fails mid-turn (mechanical stop, patient instability), the fallback is manual intervention. The system should detect failure (lack of position change, or an abort command from bed) and immediately alert the team to take over. Essentially, in Full mode, if bed cannot complete action, it notifies for manual completion or reversal. If patient becomes unstable, it might recommend pausing proning or going back supine early.

- If communications or sensors for position fail (we can't confirm if patient is prone or supine), the system should assume worst-case (e.g., assume supine or at least stop any timers) and request user confirmation of position.

UI Chip Behavior: The proning integration would have a UI element likely in the respiratory or procedures section. Possibly a chip labeled "Proning".

- When not active, it might show "Supine" status or "Not Proned".

- If proning is indicated, it could flash a suggestion "Prone patient?" with maybe a one-click to acknowledge that proning has started (Lite) or to command (Full if bed supports).

- When active (patient prone), the chip would turn a distinct color (maybe blue or some icon of a person face-down) indicating proned status. It would also likely show a **timer** like "Proned: 6h 45m so far". Perhaps a small progress bar toward 16h goal.

- It could have an alarm or color change when target time is reached (e.g., turns amber at 16h to say ready to turn supine).

- The user can interact: in Lite, they would mark "Supine now" when they flip back, which stops the timer and flips status. In Full, maybe an "Initiate Supine" button if they want to command

earlier than scheduled.

- The chip also might provide a countdown if prone planned – e.g., when supine, if a next proning session is scheduled, it could show "Next prone due in X hours" (some centers prone daily if needed).

- Additional info on click: maybe a log of all proning sessions, blood gas improvements with proning (TraceLoop could correlate PaO2/FiO2 pre and during prone to show effectiveness), etc.

- Override doesn't exactly apply beyond not doing it – a clinician could always choose not to follow a suggestion. If in Full auto bed scenario, an override would be to abort, which should be easily accessible (like a big "Stop Proning Now" which would command bed to stop rotation and go safe).

- Safety prompts: If the user tries to start proning on UI, maybe it asks for confirmation or to ensure adequate sedation, lines secured, etc. Possibly out of scope for now, but a note.

Figure 13 – Proning Position Module: *This figure depicts the integration of the proning module in TraceLoop. The controller (210) tracks patient position via bed sensors and/or user input (1). When the ARDS oxygenation loop (244) indicates refractory hypoxemia (P/F < threshold), it triggers the Proning factor (illustrated as a switch to PRONE state) (2). If automated, the bed actuator (⓯ conflict_group "BED") is commanded to rotate the patient to prone (3), otherwise a prompt is issued for manual action. The system then starts a timer in the Proning module (4), coordinating with other loops: the enteral feeding pump is paused (via IV Pump channel or feed controller) while proneess.mychlic.org, and ventilator FiO2/PEEP adjustments continue as needed but now often improve due to prone effect (the Ventilation loop registers improved oxygenation, reducing risk scores). After the target duration (e.g., 16 hours) (5), TraceLoop alerts to resume supine positioning; the bed is returned to supine and the Proning factor switches off, re-enabling loops like HOB angle enforcement. All transitions and durations are logged (FIG. 9), and any conflict (e.g., an attempt to raise the head or perform PLR during proning) is prevented by the common BED conflict group arbitration (FIG. 4 ensures only one bed maneuver at a time).*

Implementation Effort: Proning integration is **medium** in complexity. On the one hand, it doesn't involve continuous control or complex algorithms; it's more about scheduling and state tracking. The challenging part is the hardware integration if attempted: controlling a bed to prone a patient is a big leap. If we avoid that and focus on monitoring/support, then device integration is minimal (maybe just a sensor or manual input).

- *Software for scheduling:* straightforward to implement a timer and triggers. The factor table addition is simple: a binary state factor with maybe a column for recommended duration.

- *Cross-links:* We do need to add cross-channel rules (HOB vs PRONING exclusivity, feeding pause, sedation synergy perhaps). These are mostly static config entries.

- *UI and alerts:* Some development to show timers and handle user inputs for start/stop. Minor compared to other UI tasks.

- *Device side:* If implementing orientation detection, say via a wearable accelerometer, that's a small IoT integration (reading BLE data which might just give an orientation quaternion or angle – we have to process that to prone vs supine, which is doable by checking if sensor flipped ~180 degrees). If using bed's own signal, we likely rely on the vendor's interface – maybe through HL7 alarms as we saw "Head of Bed Angle - Below 30 Degrees" messages; maybe beds also have "Rotation mode active" messages or such.

- If controlling a bed (Full automation), device effort skyrockets: the bed's API would need to be thoroughly understood and safety tested. Likely beyond immediate implementation unless working closely with bed manufacturer. Possibly not attempted in initial versions – manual proning with system guidance is more realistic.

- *Testing:* In simulation, one can test proning scheduling with dummy data (e.g., simulate improved oxygenation values during prone to see if system reduces vent FiO2 accordingly – that synergy is emergent rather than direct).

- *Risks:* If integrated incorrectly, could cause harm (imagine a bug that triggers proning at wrong time or too frequently). But since actual actuation is manual in likely usage, the main risk is mis-timing or mis-advising. That's lower than direct drug loops. Implementation must ensure correct timekeeping (e.g., handle if someone supines patient early – system should reset timer).

- Another subtlety: multiple proning cycles (system should store when last proned to avoid recommending again too soon).

- Considering all, implementing proning as a guided protocol within TraceLoop is relatively straightforward (like adding a smart alarm/notification feature with logging). So the **code lift** for Lite is low (just logic and UI timers). For *Full* it could be medium to high if controlling devices – but that might be deferred or limited to specialized environment.

- From a system structure perspective, proning doesn't conflict with any existing physical actuators (except bed which we manage via conflict_group). It mostly interacts logically with vent and feeding and sedation loops, which our design addresses with rule edges. This fits well in the relational model without requiring new engine capabilities.

- Overall, proning integration will significantly help adherence to ARDS protocols (which evidence shows improves survival when done properlypmc.ncbi.nlm.nih.gov), thus adding to TraceLoop's value in critical care management.

Spontaneous Breathing Trial (SBT) Readiness

Supported Feed Formats & Sources: Spontaneous Breathing Trial (SBT) readiness is not a single sensor but rather an assessment combining several parameters to determine if a ventilated patient can attempt breathing without full support. To integrate this into TraceLoop, we gather data from:

- **Ventilator settings and patient respiratory metrics:** Key criteria include low ventilator support settings (e.g., FiO2, PEEP) and patient's spontaneous effort. Most modern ventilators can output data like current mode, set PEEP, FiO2, and whether the patient is initiating breaths (e.g., respiratory rate, tidal volumes, pressure support level). These can be obtained via **HL7** messages from ventilator (if connected to a patient monitoring network) or via vendor-specific protocols (like Draeger, GE, etc have data output). If HL7 PCD is used, ventilator data might come in ORU messages or via an interface gateway. If no direct vent data feed, the system could use values charted in the EMR or manually input by clinicians.

- **Oxygenation and PEEP**: For readiness, a common standard is FiO2 ≤ 0.4-0.5 and PEEP ≤ 5-8 cmH2Olitfl.com. These come from ventilator settings (FiO2, PEEP).

- **Hemodynamic stability:** Usually defined as no or low dose vasopressors (maybe dopamine < a certain dose, or no significant hypotension). TraceLoop can get vasopressor infusion status from its infusion loops (it knows if any vasopressor loop is active and at what dose). Alternatively, from monitors (blood pressure trending stable, or an input "hemodynamically stable = yes").

- **Consciousness/sedation:** The patient should be breathing spontaneously and preferably off heavy sedation. This means sedation infusions (propofol, etc.) are turned down or off. TraceLoop knows this from its sedation control loops or infusion data. Also neurological status: a sedation scale like RASS 0 to -2 is often used as a threshold (so patient is awake enough)litfl.com. If integrated with a BIS monitor (brain monitor) or simply the sedation loop target, the system can gauge if sedation is light.

- **Respiratory drive & muscle function:** The patient should be initiating breaths – the ventilator often reports patient-triggered breaths or we can see that controlled rate is low. Also should have adequate neuromuscular function (i.e., not paralyzed, which ties back to TOF integration – need TOF ratio ~1 or basically no paralysis).

- Some protocols include other measures like a rapid shallow breathing index (RSBI = f/VT) threshold (< 105)litfl.com, which requires capturing the patient's spontaneous respiratory rate and tidal volume when on minimal support. Ventilators typically compute RSBI or at least provide the data to compute it. If we have vent data feed: e.g., current spontaneous RR and average tidal volume (in a short period of unsupported or PS 5, PEEP 5 conditions), we could compute RSBI in software.

- **So sources summary:** Ventilator data feed (via HL7 or vendor link) is crucial for FiO2, PEEP, mode, RR, VT, etc. Infusion data (for sedation, pressors) from pumps or internal state. Possibly

ABG results or saturation (SpO2) – we at least want SpO2 and maybe PaO2 to confirm adequate oxygenation on those settings. SpO2 is from patient monitor (HL7 PCD from monitor), and ABG is a lab (HL7 lab feed).

- **Manual input** can supplement if automation is incomplete (like a doctor can check off readiness criteria and input "start SBT").

Payload Schema: There isn't a single "SBT readiness sensor"; rather we create a composite indicator. But we can treat it as a factor that evaluates to TRUE (ready) or FALSE (not ready) based on multiple conditions. The "fields" that factor uses include:

- Vent settings: FiO2, PEEP, PressureSupport (if any), VentMode.
- Patient metrics: SpO2 (should be ≥ 92% on those settingsahrq.gov), RespRate, Vt (tidal volume), maybe ABG values like PaO2 or PaCO2 (some protocols ensure PaCO2 manageable).
- Sedation: SedationLevel or infusion status (e.g., propofol off or RASS).
- Cardiovascular: MAP or PressorDose to ensure no active high support (like no high-dose vasopressor).
- Neuromuscular: TOF_ratio or simply ensure paralytic infusion is off and recovered.
- We might compute an internal boolean for each criterion and then combine (all must be true to declare ready).
- Could also have an RSBI field computed if an SBT trial was done (though RSBI is more used to decide success of SBT, not readiness, but some do use it for readiness to extubate).

For the factor table, define a factor SBT_READY. It could be a **gate** factor that becomes true when all conditions are met. It doesn't directly actuate something (the actual SBT trial is an action, often just switching vent mode to CPAP and seeing how patient does for 30-120 min). But we might design TraceLoop to automatically conduct SBT in Full mode, which means:

- Changing ventilator mode to a low support mode (like Pressure Support 5 or T-piece).
- Running for a period (30 min, etc.), monitoring vitals for failure criteria, then deciding pass/fail. That gets complex (that's the SBT trial itself).
- But the question specifically asks for integration of SBT readiness, not the trial execution or extubation per se. Likely focusing on the readiness feed.

So the SBT_READY factor will typically serve as a **requires_ok gate** for an extubation or ventilator liberation loop. I.e., the "Extubation" factor or "Vent Wean" factor would have requires_ok = SBT_READY. Also, SBT readiness being true might trigger an action: e.g., in Full mode, the ventilator loop might automatically start an SBT trial when ready.

Conflict_group: If we consider an automated SBT trial initiation as part of this, it would involve the ventilator (adjusting settings). The ventilator is an actuator conflict_group (likely VENTILATOR). So if SBT_READY factor actually commands a ventilator mode change, it would be conflict_group = VENTILATOR. However, the ventilator is already controlled by presumably a Vent_Control factor (maintaining certain blood gas targets). We then have two loops wanting to control the vent: the normal mode vs SBT mode. That could be handled by conflict_group arbitration: only one can have control at a time. Indeed, in the TraceLoop architecture, each physical actuator (like ventilator) has one loop chosen by priority at a time. We might design SBT trial as a separate factor that has priority when conditions are right. But focusing on readiness (the decision to allow SBT), which is more of a gating condition, not a continuous control (it's like a one-time check that triggers a mode switch).
Given complexity, let's do simpler:

- SBT_READY factor with no direct actuation (just a boolean). Conflict_group could be a dummy or associated with Ventilator for formality. Maybe better to associate with Ventilator group, since it conceptually deals with vent, and to ensure it is considered in vent arbitration logic (though if it doesn't send commands, it might not matter).

- Alternatively, consider SBT readiness as just a computed variable used by the vent control factor. But since the question calls it a feed integration, we treat it as a factor.

Control Logic Path:

In **Full mode** (semi-automated ventilator weaning):

- TraceLoop continually evaluates SBT readiness criteria. This could be done each control cycle as part of the factor's condition check (most values change slowly so maybe it effectively checks every few seconds or on relevant changes).

- When criteria are met, the SBT_READY factor flips true. This can trigger a sequence:

    1. It signals the ventilator control loop to initiate an SBT. Possibly done by another factor SBT_TRIAL that requires SBT_READY. That factor would take over the ventilator conflict_group with a command to switch to CPAP or low support mode for trial. It might also adjust sedation target (often sedation is minimized or paused at trial).

    2. The SBT trial then runs (the ventilator is essentially letting patient breathe on their own).

    3. During the trial, the system monitors for failure signs (some listed in the LITFL referencelitfl.com, like RR > 38, SpO2 < 88-92, HR > 140, etc.). This could be another

factor or internal logic that would abort the trial if triggered. If aborted, the vent goes back to full support (the main vent loop regains control), and SBT_READY might be set false again until conditions improve.

4. If the trial duration is completed successfully (often 30-120 min, e.g., 1 hour), then the system considers the patient for extubation. That could raise an event or set another factor EXTUBATE_READY true, which might prompt a human to extubate or an automated extubation if that were allowed (likely not automatically pulling the tube, at least a prompt).

- The SBT readiness factor thus mainly gates the transition. It might also incorporate a safety margin (like require these conditions to be stable for a certain time before declaring ready).

- If the patient fails an SBT, SBT_READY would go false (since perhaps higher support or sedation resumed, etc.), and the system would wait some hours before trying again (common practice is daily SBT attempts).

- The factor could enforce that wait by requiring e.g. 24h since last trial or specific triggers to reset it (like clinician command).

- The normal ventilator control loop would have lower priority than the SBT trial loop when trial is running, or might be effectively paused.

In **Lite mode**:

- TraceLoop will compute readiness and display it to clinicians. It might have a checklist style display: e.g.,

    - "Lung disease resolving: ☑" (could be a note if last X-ray improved or static, possibly omitted for automation).

    - "FiO2 40%, PEEP 5: ☑ meets criteria"litfl.com,

    - "Hemodynamic stable: ☑ minimal vasopressors",

    - "Patient initiating breaths: ☑",

    - "Sedation: ☑ off or light",

    - "No paralysis: ☑",
      etc.
      If all are checks, it would highlight "SBT Ready".

- It would then recommend to the clinician: "Patient appears ready for SBT. Suggest trial on CPAP for 30 min." Possibly an action button "Start SBT" on the UI. If integrated with vent, pressing that could automatically change mode (semi-automation with user confirmation).

- During the trial, the system would monitor vitals and provide decision support: e.g., a timer counting trial length and warnings like "RR high, consider terminating trial" if criteria degrade.

- After trial, it might declare "SBT passed" or "failed" based on collected data. E.g., if for the past 30 min none of the failure criteria triggered, mark pass, else fail. It can log RSBI at the start or end for reference.

- The clinician then decides to extubate or not based on that result, and TraceLoop can assist by highlighting extubation readiness factors (like ensure cough/gag etc, which is outside sensor scope except maybe requiring suction frequency etc).

- If SBT fails, TraceLoop suggests waiting and maybe adjusting settings or sedation before next attempt (maybe prints "SBT failed, recommend next attempt in 24h unless condition changes").

Actuator Integration: For Full automation, actuators involved:

- **Ventilator mode switch:** Changing vent settings (like dropping pressure support to 5, switching to CPAP mode or T-piece equivalent on vent). If TraceLoop has direct vent control (some future connectivity via something like an HL7 control or ventilator API), it would send those commands. If not, maybe instruct a connected ventilator automation sub-module, if any, to do it (less likely, as vent control historically not open, but let's assume some integration since this is a hypothetical advanced system).

- **Sedation pump adjustments:** Often, before an SBT, sedation infusion is significantly reduced or paused (to allow patient to breathe). TraceLoop's sedation loop could be commanded to lighten sedation when SBT trial starts. For instance, sedation factor might have priority_over sedation if an SBT factor is on – or simpler, the sedation target is changed to RASS ~0 by the system. If Full, it can directly slow the infusion (actuator: infusion pump for sedation).

- Possibly **Oxygen adjustments:** If the patient tolerates, some protocols try reducing FiO2 during trial to more room-air like, but usually it's kept same. So probably no direct change.

- The system might also ensure alarms are appropriately set (like backup ventilation alarms in case patient apneic).

- If trial fails, actuators: vent goes back to prior mode (increase support again), sedation might be re-increased to maintain comfort if needed.

Guard-Rails, Conflict Interactions & Fallback:

- **Criteria enforcement:** The SBT_READY factor's logic itself is a guard to prevent unsafe trials. It ensures the patient is not put on trial unless conditions are satisfied (akin to a safety interlock that matches standard ICU protocolslitfl.com).

- **Mutual Exclusion:** SBT trials should not happen concurrently with certain other loops:

    - If the patient is **paralyzed** or on NMBA, obviously an SBT is not possible. So SBT_READY should require no paralysis (which we ensure by sedation/NMBA off in criteria). And if by some reason NMBA loop tries to run during an SBT, either SBT wouldn't start, or that should abort it. We could put mutually_exclusive between SBT_TRIAL and PARALYSIS_CONTROL (meaning they cannot both be active).

    - If the patient is in **prone position**, typically SBT is not done (proning is for severe ARDS, those patients usually not ready to wean). So possibly SBT readiness requires patient supine. We can incorporate requires_ok = not PRONING or similar. This is a soft rule clinically but could be added.

    - **HOB angle** – Actually, for SBT trial, one would want HOB raised (semi-recumbent) to reduce aspiration risk if they do spontaneously breathe. Our HOB loop anyway keeps >30°, so that's fine. But ensure bed is not flat; if it were, maybe not an absolute stop but the HOB loop should raise it anyway.

    - **Other loops**: If the patient still requires significant support from e.g. vasopressors or dialysis for severe metabolic issues, sometimes they postpone extubation attempts. But not absolute – some criteria say no myocardial ischemia, etc. Hard to automate fully. Maybe we stick to vital signs/pressors as main conditions.

- **Priority & Arbitration:** If SBT trial is considered a separate factor controlling ventilator, we should assign it an appropriate risk priority. Perhaps extubation readiness is considered high priority (to get patient off vent as soon as safely possible, because prolonged intubation has harm). But during the trial itself, if something like a lethal arrhythmia or other emergency arises, those loops would override.

    - Actually, more relevant: if mid-SBT the patient's condition deteriorates, the system should abort. This can be implemented either by the trial factor constantly checking criteria or by separate factors raising alarm (e.g., a "Respiratory Distress" factor that triggers if RR>35, etc., which then preempts SBT factor).

    - The fallback if SBT aborts is the ventilation loop resumes full support. Since vent is single conflict_group, how to implement abort elegantly? Possibly by giving the main vent loop a *higher priority in case of distress.* That could happen automatically if risk (like a high harm score for impending respiratory failure) is computed; the risk engine would then choose the main vent loop over continuing SBT. We might do that by raising a harm_severity when failure criteria met, effectively pushing normal ventilation to

override (like an automatic fail-safe).

    - Also the failsafe override concept: a clinician could at any time hit "abort SBT" on UI if they see the patient failing. That triggers an immediate reversion to prior settings.

- **Guard-rails** within SBT:

    - The known failure criteria (tachypnea, desat, tachycardia, hypertension or hypotension, agitation)litfl.com are guardrails for stopping trial. The system should enforce them as automated guardrails. For example, if SpO2 < 88% for 30s or HR up by >20% or absolute >140, etc., then it declares failure and stops.

    - Each of these can be separate thresholds in the factor definitions or implemented as sub-rules. Possibly we encode them into the risk formula: if these happen, risk skyrockets and aborts SBT (because then some other loop with safety profile takes over).

    - Alternatively, model a sentinel factor: e.g., SBT_FAIL_CRITERIA factor that monitors those and if true, it has priority_over SBT trial. That might be simpler logically: SBT_FAIL factor could be triggered to high risk if any condition hits threshold, thereby kicking the vent back.

- **Fallback:** If communications or sensors fail during SBT (e.g., if ventilator data feed is lost mid-trial, the system might auto-end trial because it can't monitor safely, or alert staff).

    - If something outside algorithm triggers (like clinician decides patient is not tolerating even if criteria not fully tripped, maybe they see patient anxiety), they can manually end it; the system should allow immediate override.

    - If sedation infusion was off and trial fails, system might automatically resume some sedation to calm the patient once back on vent (under the assumption if they got agitated).

    - Ensure that after trial, regardless of pass/fail, the system resets appropriately: e.g., if fail, schedule next attempt ~24h later (the SBT_READY factor might enforce a delay or require manual reset by clinician).

    - If pass, system might now wait for clinician command to extubate (which is not an actuator TraceLoop would do, it's a manual procedure). The system can however prepare things (like ensure the tube suction is done, alarm limits changed after extubation etc., but those are beyond current telemetry integration).

UI Chip Behavior: The SBT readiness/trial integration would appear on UI perhaps as a combined chip (like "Weaning Trial" or "Extubation readiness" chip).

- Before an SBT, it might display readiness status: perhaps a checklist icon or simply "SBT Ready: Yes/No". If not ready, perhaps grey or with an explanation which criteria not met (like "Not Ready: FiO2 0.6 too high" or "Not Ready: on vasopressors").

- If ready, it could highlight green and maybe prompt "Ready for SBT".

- Possibly a distinct button appears: "Start SBT" which the clinician can press.

- In Full mode, if conditions trigger automatically (some protocols say do daily SBT at a set time each day if ready), the system might just announce it's going to start ("Commencing SBT trial" with countdown) to notify staff.

- During SBT trial, the chip likely shows a timer (like "SBT: 10:00 elapsed of 30:00"), and key vitals like RR, SpO2, HR in real-time, possibly with some gauge against thresholds. Maybe it highlights any borderline metrics (like if RR climbing).

- If criteria cross threshold, the chip could flash red "SBT Failed – returning to support!" and perhaps a countdown for how long it will let them try if it's borderline.

- If the trial completes, the chip could show "SBT Passed! Extubate?" and maybe turn green or give a checkmark. It might then have an "Extubate" button just for logging or for instructing to do so (the system won't pull tube but it marks that extubation is now the next step and perhaps flips ventilation factor to a "pre-extubation" state).

- If in Full mode, sedation chip might also show changes concurrently (like sedation went off – sedation chip would indicate paused or off).

- Overrides: at any time a clinician might hit "Terminate SBT Now" if they are uncomfortable; that should be available on the UI, perhaps by tapping the chip and selecting abort (or in Full perhaps turning the physical override knob as per overall system which then might have an option to stop SBT).

- After extubation, the chip might disappear or show "Extubated" and no trial needed obviously.

- The UI can also provide historical info: if SBT failed, it could show why (e.g., "Failed: RR 45, RSBI 120" etc).

- In summary, the chip guides the clinician through the readiness and trial, ensuring they know when the patient is likely to succeed and when not, thus preventing premature trials or delays.

Implementation Effort: SBT readiness integration is **moderate to high** on logic, but much of it is combining existing data:

- *Data integration:* We need ventilator data feed reliably. If the system already integrates ventilator parameters for other loops (like an oxygenation or ventilation control loop, presumably TraceLoop does have a ventilation management component), then those values are accessible. Pump data for sedation/pressors is likely integrated from infusion loops. SpO2 is from patient monitor (should be integrated as a basic vital).

- So technically, the data is likely in the ecosystem. It's a matter of writing the logic to evaluate readiness.

- *Factor logic:* possibly the first multi-variable composite condition factor in the system. Implementation could either hard-code in code ("if X and Y and Z then ready") or treat it as a rule table. Could be represented in config as well ("requires: FiO2<=0.5, PEEP<=5, etc."). Might be easier to code due to complexity, but either way, it's not too difficult logically.

- *Trial control:* If we implement automated trial (Full mode), controlling the ventilator is a big step. Many current systems do not allow external write to ventilator settings (for safety reasons). If we assume future connectivity, that needs careful integration and testing. Without direct control, the system can still function by prompting clinicians to change mode (semi-automated).

- *Trial monitoring logic:* This is like a little subroutine that monitors multiple inputs for threshold exceedance over time – a bit of state management (like counting 5 min intervals for sustained tachypnea etc.). Not heavy but needs reliability (maybe have a small buffer or average).

- *Integration with sedation off:* This might require adjusting sedation loop logic (like adding a state for "weaning sedation for SBT"). Possibly simpler: when SBT trial factor is active, sedation factor gets an override to minimal rate (like it's forced to off unless something triggers).

- *Testing:* Will require simulation of various patient scenarios (like one where SBT is ready and passes, one where fails due to each criterion, etc.). The logic might need tuning (for example, how sensitive the failure detection should be – maybe allow transient tachycardia vs sustained).

- *UI and user experience:* This portion is quite interactive (timers, status updates), requiring UI development efforts to present clearly the status and allow control. But it's a guided process, which can be done akin to how override sequences are handled in UI (which the doc describes in override flow).

- *Risk management:* Automated SBT is somewhat cutting-edge; many closed-loop systems avoid controlling ventilator modes fully due to safety. But including it in TraceLoop would be a big highlight if done right. The risk is if the system mistakenly deems someone ready who isn't (leading to a failed trial or potential harm). However, since a failed trial means just going back on vent, the risk is discomfort and short-term instability, which is manageable. The system should have fail-safes to quickly revert which we've included. Regulatory-wise, it's like an advisement turned partial automation – likely acceptable if it can be overridden quickly.