ChatGPT

Indent Feature Configuration and Usage for FLM & IST Trips

Configuration of `send_vehicle_requisition_request` (IDB Key)

- **Organisation Config Flag:** The `send_vehicle_requisition_request` key is an organisation-level flag that enables the *vehicle requisition (indent)* workflow [1]. When set to `true`, the system will initiate vendor vehicle requisition for eligible trips. This flag is often accompanied by related settings under `flm_freight_type_config` to control behavior for First Mile (FLM) shipments based on freight type [2] (e.g. separate toggles for LTL vs FTL). For example, an org config may enable indent globally but only trigger it for FTL trips (`send_vehicle_requisition_request_ftl: true`, `send_vehicle_requisition_request_ltl: false` in config) [2].
- **IDB Storage:** These flags are stored in the organisation's configuration (sometimes in an IDB or settings table). A 2022 DB migration shows the introduction of `send_vehicle_requisition_request` and related keys, indicating how the indent system was rolled out [1]. The config can also include thresholds (like volume utilization) to classify a trip as FTL/LTL and decide indent usage accordingly [2].

Frontend Implications and UI Controls

- **Feature-Gated UI Actions:** When `send_vehicle_requisition_request` is enabled, the front-end (e.g. dashboard) exposes options to utilize the indent flow. For instance, in manual trip creation for FLM, the UI allows selecting a *Vehicle Make* plus a *Vendor* (instead of a specific vehicle/driver) – effectively preparing an indent request. This is gated by both the flag and a sub-config `flm_trip_config.custom_trip_allow_vendor_with_vehiclemake`. The code enforces that *vendor assignment with just a vehicle type is only allowed if indent is enabled* and that sub-setting is true [3]. Otherwise, attempting to assign a vendor to a trip without a concrete vehicle triggers an error *"Vendor assignment not allowed"* [4].
- **Vendor Portal Views:** The system has a "Vendor Portal" mode for trips (used by vendors to respond to indents). UI components (columns, tabs) are adjusted based on the indent flag. For example, in the trip details view, if `send_vehicle_requisition_request` is true, the code may use a special set of vendor portal headers/tabs (showing fields like vehicle number, vendor name, etc.) [5] [6]. The `orgConfig.send_vehicle_requisition_request` value is fetched in UI helper modules [7] [8] to conditionally include vendor-related info (like *Vehicle Vendor Name*, *Freight Type*, etc.) in trip detail columns [9]. Essentially, the front-end will display indent-related data (e.g. **vehicle_vendor_name**, **freight_type** fields in Trip Details) only if the indent feature is on.

Validation Rules Tied to Indent

- **Vendor Contact Requirement:** When indent is enabled, the backend requires that each vehicle vendor has a contact number on record. This is because the system will send SMS notifications for requisitions. In the Vendor model, the creation/update logic throws an error if

`send_vehicle_requisition_request` is true but the vendor's phone number is missing [10] [11]. This ensures no indent request goes out to a vendor without a contact.

- **Trip Assignment Constraints:** If indent is off, assigning a vendor directly to a trip is disallowed. As noted, `vendorAssignmentAllowed` is computed as `orgConfig.send_vehicle_requisition_request && flm_trip_config.custom_trip_allow_vendor_with_vehiclemake` [3]. If this is false, the system will prevent linking a vendor to the trip at planning time. This protects against indent-specific flows being used inadvertently when the feature is disabled.

Indent Flow in Trip Planning (FLM scenario)

- **Triggering Indent (Vendor Selection):** In a first-mile trip (tripType **endpoint**), indent flow can be triggered either automatically or manually. When building a trip's plan, if a *Vehicle Make* (type) is chosen rather than a specific vehicle, the system can allocate a vendor. In automatic allocation, the platform's vendor-allocation engine runs *only if indent is enabled for the trip's freight type*. The code filters allocation results such that a trip is offered to vendors only when the corresponding config flag for its freight type is true [12]. For example, if a trip is classified as **FTL** (Full Truck Load) and `send_vehicle_requisition_request_ftl` is true, the vendor will be included in the indent request list; if the trip were LTL with that flag false, the engine would skip indent and no vendor would be assigned [13].
- **Assigning a Vendor to Trip:** Once the system selects a vendor (or an operations user manually picks one), the `EndpointTripHandler.assignVendorToTrip()` method is invoked. This method creates a **VendorTrip** entry linking the trip and vendor, and updates the trip's data. It appends an entry in `trip.extra_details.assigned_vendors` with the vendor ID and timestamp [14]. If an "indent number" is provided (an external reference for the requisition), it's stored to avoid duplicates [15] [16]. After adding the vendor, the trip is marked as having a third-party vendor: if exactly one vendor was assigned and no expiry window (broadcast) is used, the trip's `extra_details` gets `third_party_vendor_id` and name set to that vendor [17]. This identifies the vendor responsible for the trip.
- **Freight Type Determination:** The system also determines the trip's freight type (FTL/LTL) during this process. If not already set, it calculates it using `getTripFreightType()` based on volume/weight vs thresholds from config [18] [19]. The freight type (and utilization metrics like `max_cumulative_product_value`) are stored in trip details for reference [18] [19] – these may influence whether indent was initiated (as above) and are displayed in UI (the **freight_type** column).

Indent Flow in IST (Mid-Mile) scenario

- **Vendor Indent for IST:** For Inter-Stock Transfer (mid-mile) trips, the indent mechanism is very similar, but handled via the IST handler. When creating an IST trip (tripType **midmile/IST**), if indent is enabled the system will not immediately assign a company vehicle. Instead, it can allocate a vendor based on route and vehicle type. The `TmsHandler.allocateVendorToIst()` routine finds serviceable vendors for the given route and vehicle type [20] [21]. It builds a `vendorTripHashMap` of vendor->trip assignments and calls `handleVendorRequisition` (which under the hood uses the same VendorVehicleRequisitionEvent flow) to send out requests [22].
- **IST Trip Updates:** When a vendor is assigned to an IST, the IST record is updated similarly: an indent *vendor priority list* might be attached, and the IST's `extra_details` will include the vendor's details. For example, in the IST vehicle assignment code, if indent is on, they populate `requiredIST.extra_details.vehicle_vendor_details` with the selected vendor's code

and name [23]. This ensures the IST's details page shows which vendor is supposed to provide the vehicle.

- **Mid-mile Execution:** Once a vendor accepts an IST indent, the VendorVehicleRequisitionHandler will assign the provided vehicle and driver to the IST. The method `assignVehicleAndWorkerToIST()` creates or fetches the Vehicle and Worker (similar to FLM flow), then calls `istHandler.assignVehicle(...)` to attach the vehicle/driver to the IST [24]. It also updates the vehicle's current hub and status appropriately (e.g. mark it assigned to the IST) [25] [26]. At this point, the IST status transitions from a pending indent to an active trip with a specific vehicle.

Outbound Requests and Notifications (Downstream Effects)

- **Indent Request Dispatch:** A crucial downstream effect of enabling this flag is the automated communication with vendors. When a vendor is assigned to a trip, the system triggers a **VendorVehicleRequisitionEvent**. This event, upon the transaction commit, sends out an indent **SMS/notification** to the vendor asking for a vehicle. The `VehicleVendorIntegration.sendVendorVehicleRequisitionSMS` API is called with details like the hub, last drop location (city), vendor info, and one or more trip reference numbers [27] [28]. This integration call (to an external service "Raven") is gated by the same config flag – if `send_vehicle_requisition_request` is false, the call is a no-op [29] [30]. Thus, indent requests will only be sent when the feature is active for that org.
- **Batch vs Single Requests:** The event logic can group multiple trips for one vendor into a single request. If multiple trips are assigned at once to the same vendor (e.g. in bulk allocation), the code will send one consolidated SMS containing all trip reference numbers [31] [32]. If it's a single-trip indent, the SMS payload also includes the **vehicle make** required [33] so the vendor knows what type of vehicle to dispatch. After sending, the system records the SMS in a history log on each trip (updating `extra_details.vendor_sms_history`) for auditing [34].
- **VendorTrip Tracking:** The `vendortrip` DB table (introduced with indent system) tracks the vendor's response state. When a request is sent, a `VendorTrip` record is created with `is_active = true` (pending response) [35]. If the indent involves multiple vendors (broadcast scenario), multiple VendorTrip records may exist until one vendor accepts. The code sets a **status** in VendorTrip (e.g. an enum like *REQUESTED*, *ASSIGNED*, *EXPIRED*). On initial assignment, VendorTrip status is likely "REQUESTED", and once a vendor accepts and is confirmed, the system marks it **ASSIGNED** [36] by calling `VendorTrip.setTripStatus(...ASSIGNED...)` for the accepted vendor.
- **Expiration and Timeout:** If the indent is not answered within a configured timeout, the system can handle it in two ways: (1) If **indent expiry** is enabled in org settings (see `indent_settings.is_expiry_module_enabled`), it will allow a broadcast to multiple vendors and auto-expire the request. If a single vendor was assigned and expiry is disabled, the trip is tied to that vendor (third_party_vendor_id) immediately [37]. There is also a mechanism to send an **expiry alert SMS** to vendors if they haven't responded in time. A separate flag `send_vehicle_requisition_expire_alert` controls this. When true, the integration will call `handleVehicleRequisitionExpirationAlert` via Raven to notify vendors of expiry [38] [39]. (This is part of the "indent expiry" feature.)
- **Vendor Acceptance:** When a vendor **accepts** the indent (e.g. via the vendor portal or a mobile app), they provide vehicle and driver details. The backend then invokes `VendorVehicleRequisitionHandler.assignVehicleAndWorker(...)` to finalize the trip. This flow creates a new Vehicle entry if needed (with the vendor as owner) and a Worker record for the driver [40] [41]. It validates that the responding vendor matches the trip's vendor and that the trip is in a pending state for that vendor [42] [43]. Upon successful assignment, the trip's

`vehicle_id` and `worker_id` are set, and the trip status moves from "created" to "assigned/ongoing".

- **State Updates on Accept:** Several updates happen once a vendor's vehicle is assigned:
- The trip's **TripEvent log** records an *Indent Accepted/Assigned* event. The code explicitly creates an `IndentAssignedEvent` for the trip when the vendor's vehicle is assigned [44], marking in the timeline that a vendor responded. (It skips creating duplicate events if one already exists [45].)
- The VendorTrip record for that vendor is updated to reflect the acceptance (is_active may remain true, but an internal status enum likely flips to ASSIGNED as noted).
- The system sets up the worker and vehicle to start the trip. For FLM, this means linking the worker to the vehicle (via a Medium record) and assigning them to the trip/medium [46] [47]. For IST, it calls ISTHandler to attach the vehicle/driver to the IST as described above [24].
- The trip's `extra_details` may be updated to remove any `vendor_priority_list` or indent number now that it's fulfilled, and normal trip execution info (like start times, etc.) will proceed.
- **Notification of Acceptance:** Once the vendor is assigned, the system sends out a **"Vehicle Request Accepted"** notification. In code, after assigning the vehicle and worker, it invokes `VehicleVendorIntegration.sendVendorVehicleRequestAcceptSMS` [48]. This uses Raven to send an SMS (or email) indicating the vendor has provided a vehicle. The integration again checks the main flag (it only sends if `send_vehicle_requisition_request` is true) [49]. This acceptance notification could be directed to internal ops teams or the end-customer, as a confirmation that a vehicle is on the way.
- **Trip Execution & Completion:** With a vendor's vehicle on the trip, from here the trip proceeds like a normal one. During or after execution, if any changes happen (like vehicle breakdown or manual reassignment), there are handlers in place but those are beyond the initial indent scope. On trip completion or cancellation, if needed, the system can send a **derequisition** message. For instance, if a trip is canceled after indent, `sendVendorVehicleDerequisitionSMS` will be called to inform the vendor not to send the vehicle [50] [51] (this also respects the main flag and will no-op if false [52]).

Summary of Impact

Enabling `send_vehicle_requisition_request` fundamentally changes the trip workflow for FLM and IST trip types: - **In Planning:** Trips are not immediately tied to an in-house vehicle/driver. Instead, the planning includes selecting a vehicle *type* and optionally a vendor, deferring actual vehicle assignment to the vendor. Internal queries for available vehicles switch to consider vendor fleet availability (joining on `vehiclemake.vehicle_vendor_id` instead of a concrete vehicle's vendor) [53], so the pool of "available vehicles" becomes the pool of vendor options when indent is on. - **Data & State:** Additional data is stored with the trip – the list of invited vendors, chosen third-party vendor, indent number/reference, freight type classification, etc. – to manage the indent. New DB relations (VendorTrip) keep track of vendor-trip linkage and status. - **User Interface:** Operations sees trips in a "Requested" state waiting for a vendor. The UI can show **Vendor Name** and **Freight Type** on the trip card (when enabled) [9], and provide actions like "Add New Rider" (to manually input vendor's driver details) which are only active if indent is on. The vendor gets a portal view to accept or reject requests. - **Notifications & Integration:** Downstream, the system communicates through Raven (or similar) to vendors via SMS/email at each stage: request sent, request accepted, expiration, etc., all guarded by this flag [29] [54]. If the flag is off, none of these indent communications or vendor allocations occur – trips default to standard fleet assignment. - **Overall Flow:** For **FLM trips**, indent is typically used for *FTL shipments* (if configured) – the trip goes into a holding pattern until a transporter confirms a truck. For **IST (MM) trips**, indent allows outsourcing inter-hub movements to 3PL vendors via an automated request/response system. All validations (like requiring vendor contact and mapping vehicle types to vendors) and state transitions (trip events, status changes) revolve around the

`send_vehicle_requisition_request` setting to ensure the indent process runs only when intended and with all necessary information.

Sources:

- Configuration flags introduction (DB migration) [55] and freight-type sub-keys [2]
- Vendor model – require contact if indent on [10] [11]
- Trip handler – allow vendor+vehicleType assignment only if indent on [56]
- FLM allocation – indent only for allowed freight types (FTL/LTL) [13]
- IST assign – add vendor details to IST when indent on [23]
- AssignVendorToTrip – create VendorTrip, mark third_party_vendor, etc. [14] [17]
- Send indent request SMS via integration (Raven) [27] [33] (executes only if flag true [29])
- Accept flow – assignVehicleAndWorker (vendor provides vehicle/driver) [57] [58] and send "request accept" notif [49].
- VendorTrip status update on accept [36].
- "Add New Rider" API flow tying into indent (manual accept) [59] [60].
- Derequisition alert SMS (on cancellation) also tied to the flag [50] [51].

---

[1] [2] [35] [55] 21-04-2022-indent-management-system.txt
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/db_migrations/21-04-2022-indent-management-system.txt

[3] [4] [56] endpoint-trip-handler.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/trip-manager/endpoint-trip-handler.js

[5] [6] ist-trip-details.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/ist-handler/ist-utils/ist-trip-details.js

[7] [8] [9] ops-trip-details.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/models/crm-dashboard-parts/crm-utils/ops-trip-details.js

[10] [11] vehicle-vendor.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/models/vehicle-vendor.js

[12] [13] endpoint-trip-handler-vendor-allocation.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/trip-manager/endpoint-trip-handler-vendor-allocation.js

[14] [15] [16] [17] [37] trip-handler.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/trip-manager/trip-handler.js

[18] [19] [53] trip-manager-utils.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/trip-manager/trip-manager-utils.js

[20] [21] [22] tms-indent-handler.js
https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/tms-handler/tms-handler-class/tms-handler-parts/tms-indent-handler.js

23 vehicle-assign-parts.js

https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/ist-handler/ist-handler-parts/vehicle-assign-parts.js

24 25 26 vendor-vehicle-requisiton-handler-ist-parts.js

https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/vendor-vehicle-requisition-handler/vendor-vehicle-requisition-handler-class/vendor-vehicle-requisiton-handler-ist-parts.js

27 28 31 32 33 34 vendor-vehicle-requisition-event.js

https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/vendor-vehicle-requisition-handler/vendor-vehicle-requisition-event/vendor-vehicle-requisition-event.js

29 30 38 39 49 50 51 52 54 vehicle-vendor-integration.js

https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/vendor-vehicle-requisition-handler/integration-handler/vehicle-vendor-integration.js

36 40 41 42 43 44 45 46 47 48 57 58 vendor-vehicle-requisition-handler-class.js

https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/domain-models/vendor-vehicle-requisition-handler/vendor-vehicle-requisition-handler-class/vendor-vehicle-requisition-handler-class.js

59 60 trip-manager-api.js

https://github.com/shipsy/projectx/blob/3f337d8e1e2e04f91b4157e542f23dd9ceac6332/common/models/retail-dashboard-parts/trip-manager-api.js