Nord Garages v2.0.0 — Documentation

Nord Garages v2.0.0 documentation.

- Getting Started
  - Nord Garages v2.0.0
  - Requirements
  - Installation
  - Quick Start
- Configuration
  - Core Configuration
  - Blips & Vehicle Types
  - Bridges Configuration
  - Vehicle OS Configuration
  - Persistent Parking, Delivery & Valet
- Garage Creator and Management
  - Admin Access & Creator
  - Garage Types & Access
  - Points & Placement
  - Garage Templates
- Player Vehicle Workflows
  - Store & Spawn Flow
  - Vehicle Transfers
  - Showroom

    - Vehicle States

- Impound System
    - Impound Overview
    - Impound Vehicle Type Safety
    - Job Impounds
    - Automatic Recovery

- Vehicle OS
    - VIN & Plate Identity
    - Snapshots & History
    - Advanced Access & Collections
    - Reservations & Physical Slots
    - Service, Insurance, Stolen & EV Data
    - Recovery Center & Analytics

- Integrations and API
    - Framework Bridge
    - Keys, Fuel & Inventory
    - Client Exports
    - Server Exports — Garages & Vehicles
    - Server Exports — Admin, Staff & Sync
    - Server Exports — Vehicle OS
    - Events & Live Sync
    - Nord Phone Integration
    - Nord Staff Integration

- Database and AutoSQL
    - AutoSQL & Schema
    - Core Tables Explained
    - Database Safety

- Troubleshooting and Production
    - Common Problems

    - Diagnostics & Recovery
    - Production Checklist

- Changelog
    - v2.0.0 — Vehicle OS
    - v2.0.0 Stabilization Pass

Getting Started

Nord Garages v2.0.0

Nord Garages v2.0.0

Nord Garages is a complete vehicle storage and lifecycle system for FiveM. Version 2.0.0 combines public/job/fleet/house garages, dedicated impounds, native showrooms, vehicle transfers, real/persistent parking, framework ownership synchronization and the **Vehicle OS** layer.

Main capabilities

- Public, job, fleet and house garage workflows.
- Dedicated impound yards separated from normal garages.
- Land, boat, helicopter and plane type safety.
- In-game Admin Creator and Garage Manager.
- Native GTA showroom interiors without a shell dependency.
- Vehicle transfer system with configurable fees and delivery times.
- Job impound workflow and field impound action.
- VIP Persistent Parking backed by OneSync.
- Permanent VIN identity and plate alias/history support.
- Vehicle snapshots, service, insurance, stolen state and EV metadata.
- Collections, favourites, reservations and physical parking slot backend.
- Recovery Center, analytics and provider registry.
- Live synchronization hooks for Nord Staff and Nord Phone.
- AutoSQL with versioned migrations.

Supported frameworks

`Config.Framework = 'auto'` detects **Qbox/QBX**, **QBCore**, **ESX**, **Nord Core**, then falls back to standalone operation.

Important design rule

Framework vehicle tables remain the ownership source where applicable, while Nord Garages is authoritative for its runtime garage/impound/Vehicle OS state. Avoid editing `nord_garage_vehicles` manually while the server is live unless you fully understand the state model.

Requirements

Requirements

Required

- FiveM server using `cerulean` fx_version.
- Lua 5.4 support.
- **oxmysql** started before `nord_garages`.
- OneSync is strongly recommended and required for the full persistent-world behaviour.

Frameworks

The resource can run with:

- ESX
- QBCore
- QBX / Qbox
- Nord Core
- Standalone

Optional integrations

- `nord_vehiclekeys`
- `qb-vehiclekeys`
- `nord_inventory`
- `ox_inventory`
- `qb-inventory`
- `qs-inventory`
- `ps-inventory`
- `codem-inventory`

- `lj-inventory`
- `tgiann-inventory`
- `ox_fuel`
- `lc_fuel`
- `LegacyFuel`
- `ox_lib` for notifications/TextUI when configured

Recommended start order

```
ensure oxmysql
# framework/core resources
# inventory/fuel/keys resources
ensure nord_vehiclekeys   # when used
ensure nord_garages
# integrations such as nord_staff / nord_phone
```

Starting the key/inventory/fuel providers before Nord Garages allows auto-detection to resolve them immediately.

Getting Started

Installation

Installation

1. Copy the `nord_garages` folder into your server resources directory.
2. Ensure `oxmysql` is installed and connected to the same database used by your framework.
3. Review `shared/config.lua` before first start.
4. Add `ensure nord_garages` to `server.cfg` after its dependencies.
5. Restart the server or start the resource.

Database installation

Nord Garages includes **AutoSQL**. On normal installations you do not need to manually import the SQL file. The resource creates and migrates its own tables on startup.

The included `sql/nord_garages.sql` is a manual fallback for environments where you explicitly prefer manual SQL setup.

First validation

After first start:

- Confirm there are no oxmysql errors.
- Run `/garageadmin` with an account that has the configured ACE permission.
- Create or inspect a garage.
- Publish the garage.
- Store and retrieve one owned test vehicle.
- Verify fuel, keys and vehicle properties are restored.
- Test an impound separately from a normal garage.

Existing installations

Do **not** drop old Nord Garages tables when upgrading. AutoSQL migrations are designed to upgrade the current schema in place while preserving data.

Quick Start

Quick Start

1. Give yourself admin access

Default permission:

```
add_ace group.admin nord.garages.admin allow
```

The permission node can be changed using `Config.AdminAce`.

2. Open the creator

Default command:

```
/garageadmin
```

3. Create a garage

Use **New Garage** and choose the correct vehicle categories. Configure at minimum:

- Name and identifier
- Garage type
- Vehicle categories
- Access policy
- Garage point
- Store point(s)
- Spawn point(s)
- Blip visibility

- Published state

4. Test the player flow

Enter the garage interaction area, open the garage, spawn an owned vehicle, then return it through a compatible store point.

5. Create impounds separately

Impound yards have their own management flow. Keep land, boat and air impounds separated unless the configuration deliberately supports the matching vehicle classes.

Configuration

Core Configuration

Core Configuration

The main configuration file is `shared/config.lua`.

```
Config.Locale = 'en'
Config.Framework = 'auto'
Config.Debug = false

Config.AdminCommand = 'garageadmin'
Config.AdminAce = 'nord.garages.admin'
Config.RealParkingCommand = 'parkvehicle'

Config.InteractionKey = 38
Config.InteractionDistance = 4.0
Config.ScanDistance = 70.0
```

Framework

`Config.Framework` accepts:

- `auto`
- `esx`
- `qb`
- `qbx`
- `nord`
- `standalone`

Use `auto` unless you need to force a specific bridge.

Debug

Keep `Config.Debug = false` in production. Enable it temporarily while diagnosing integration or placement issues.

World interaction

```
Config.WorldInteraction = {
    enabled = true,
    refreshSeconds = 10,
}
```

The built-in interaction does not require a target resource. The default flow uses the map blip plus Nord TextUI when the player enters interaction range.

Spawn/return safety

- `Config.SpawnClearRadius`: default clearance around spawn positions.
- `Config.ReturnRadius`: return/store validation radius.
- `Config.ServerSpawnLockSeconds`: short server-side spawn lock to prevent duplicate concurrent spawns.

Blips & Vehicle Types

Blips & Vehicle Types

Blip presets

`Config.Blips.useTypePresets = true` applies deterministic blip presets based on garage category.

Default groups include:

- `land`
- `boat`
- `air`
- `plane`
- `helicopter`
- `mixed`
- `impound`

With `lockTypeSprite = true`, old/custom sprite values stored in SQL are ignored for the sprite itself, while per-location colour/scale may still be used.

GTA class → Nord type

`Config.VehicleTypes` maps GTA vehicle classes to Nord categories such as:

- `car`
- `motorcycle`
- `bicycle`
- `boat`
- `helicopter`
- `plane`
- `service`
- `emergency`
- `industrial`

- `military`
- `special`

Spawn groups

```
Config.SpawnGroups = {
    car = { 'car', 'motorcycle', 'bicycle', 'service', 'emergency', 'industrial', 'special' },
    boat = { 'boat' },
    air = { 'helicopter', 'plane' }
}
```

These groups are important for store/spawn compatibility and strict impound routing.

Bridges Configuration

Bridges Configuration

```
Config.Bridges = {
    notify = 'auto',
    textui = 'custom',
    keys = 'auto',
    fuel = 'auto',
    inventory = 'auto',
}
```

Notify

Supported modes include `auto`, `ox_lib`, `qb`, `esx` and `custom`.

TextUI

Default is Nord's internal NUI TextUI using `custom`. You may switch to supported external providers in the bridge configuration.

Keys

`auto` prefers `nord_vehiclekeys` when available, then compatible alternatives. Use `none` only when no key lifecycle integration is desired.

Fuel

Auto-detection supports `ox_fuel`, `lc_fuel` and `LegacyFuel`.

Inventory

Auto-detection supports Nord Inventory and multiple common FiveM inventories. Inventory code is isolated from the core so integrations can be replaced without rewriting garage logic.

Vehicle OS Configuration

Vehicle OS Configuration

The Vehicle OS layer is controlled by `Config.VehicleOS`.

```
Config.VehicleOS = {
    enabled = true,
    vinPrefix = 'NRD',
    stateEngine = true,
    duplicateGuard = true,
    plateAliases = true,
    snapshots = true,
    analytics = true,
    collections = true,
    reservations = true,
    advancedAccess = true,
    maintenance = true,
    insurance = true,
    stolenState = true,
    evSupport = true,
    providerRegistry = true,
}
```

Maintenance thresholds

Defaults:

- Service: 5000 km
- Oil: 2500 km
- Tyres: 3500 km

These values are advisory metadata thresholds for systems integrating with the maintenance API.

Dynamic transfer defaults

Dynamic transfer calculation uses a base fee/time plus distance-based additions, capped by `maxSeconds`.

Access roles

Default advanced access roles:

- `coowner`
- `family`
- `employee`
- `temporary`
- `valet`

Temporary access is limited by `Config.VehicleAccess.maxTemporaryHours`.

Configuration

Persistent Parking, Delivery & Valet

Persistent Parking, Delivery & Valet

VIP Persistent Parking

Persistent parking requires both:

1. The feature enabled in `Config.PersistentParkingVIP`.
2. The garage itself allowing real parking where applicable.
3. The player passing a VIP provider, ACE, configured identifier, or admin bypass.

Default ACE:

```
add_ace group.vip nord.garages.vip allow
```

Important defaults:

- Restore on resource start: enabled
- Restore missing vehicles: enabled
- Watchdog: 45 seconds
- Per-player limit: `0` (unlimited)
- Command: `/parkvehicle`

Vehicle Delivery

Default configuration is VIP-only with a 90 second cooldown and configured fee.

Valet

Default configuration is VIP-only with its own fee and cooldown.

External VIP systems should integrate through `RegisterVIPProvider` instead of modifying core logic.

Garage Creator and Management

Admin Access & Creator

Admin Access & Creator

Command

Default:

```
/garageadmin
```

Configured through `Config.AdminCommand`.

Permission

Default ACE node:

```
nord.garages.admin
```

Example:

```
add_ace group.admin nord.garages.admin allow
```

Nord Garages also provides runtime access exports for trusted resources:

```
exports['nord_garages']:AddAdmin(value)
exports['nord_garages']:RemoveAdmin(value)
exports['nord_garages']:IsAdmin(value)
exports['nord_garages']:GetRuntimeAdmins()
```

Compatibility aliases are also available:

```
GiveGaragesAccess
RemoveGaragesAccess
HasGaragesAccess
```

UI structure

- New garages use the guided creation wizard.
- Existing garages use the dedicated Garage Manager modal.
- Placement runs inside the admin workflow and writes points back through the server-authoritative creator API.

Garage Types & Access

Garage Types & Access

Nord Garages supports several logical garage profiles, including templates for:

- Public
- Job
- Marina
- Airport
- House
- Fleet

Each garage stores its own categories, access data, settings, blip data and published state.

Access patterns

Common access modes include public/player access and job-restricted access. Job integrations should always validate access server-side; do not rely on hiding UI elements as a security mechanism.

Capacity

`capacity = 0` means unlimited capacity. Positive values limit the number of vehicles assigned according to the garage capacity rules.

Publishing

An unpublished garage is retained in the database/admin tools but should not act as a normal public gameplay location. Publish only after points and access rules are verified.

Points & Placement

Points & Placement

Garage world configuration is stored in `nord_garage_points`.

Common point types include:

- `entrance` — opens the garage while on foot.
- `store` — validates and stores compatible vehicles.
- `spawn` — candidate location for vehicle retrieval.

Showroom/camera world points from older versions are removed by migration because v2 uses native interior presets.

Placement configuration

The placement editor supports:

- Camera orbit.
- Adjustable placement step sizes.
- Vehicle preview.
- Preview transparency.
- Vehicle-group preview models.
- Realtime telemetry with throttled validation.

Default step sizes:

```
{ 0.01, 0.05, 0.10, 0.25, 0.50, 1.00 }
```

Best practices

- Use separate spawn/store points for air and land vehicles.
- Give helicopters/planes larger `clearRadius` values.

- Do not overlap spawn points with dense traffic/ped areas.
- Confirm the heading and ground/water height in-game before publishing.

Garage Templates

Garage Templates

`Config.GarageTemplates` provides reusable defaults for integrations or creator flows.

Default templates include:

Public

Cars/motorcycles, unlimited capacity, transfers/sharing/real parking/showroom enabled.

Job

Similar to public but intended for job access policies.

Marina

Boat-only template.

Airport

Plane/helicopter template.

House

Small default capacity with showroom disabled.

Fleet

Cars/motorcycles/service/emergency categories with sharing disabled by default.

Templates are configuration defaults only. A registered garage can override its stored categories/settings.

Player Vehicle Workflows

Store & Spawn Flow

Store & Spawn Flow

Storing

The server validates ownership/access, garage compatibility, capacity and current authoritative vehicle state before committing the store operation.

When storing succeeds:

- Vehicle properties/health/fuel are persisted.
- Temporary world tracking is cleared.
- State is changed to `stored`.
- Framework state is synchronized where supported.
- Key lifecycle is reconciled.
- History/live-sync events are emitted.

A failed store validation does **not** intentionally detach a still-active vehicle from runtime tracking.

Spawning

Spawning uses a short server lock to prevent concurrent duplicate requests. The state can temporarily enter `spawning`, then transitions after the client/server confirm flow.

On resource/server restart, stale `spawning` rows are repaired back to `stored` by migration/startup safety logic.

Duplicate guard

Vehicle OS duplicate protection checks authoritative identity/state before restoration or spawn operations. Do not bypass the spawn APIs by creating a second owned entity manually with the

same plate.

Vehicle Transfers

Vehicle Transfers

Transfers can be enabled globally and per garage.

Legacy/default transfer values:

```
Config.Transfer = {
    enabled = true,
    defaultFee = 500,
    defaultSeconds = 300,
}
```

Vehicle OS can also use dynamic distance-based pricing/timing:

```
Config.DynamicTransfer = {
    enabled = true,
    baseFee = 150,
    feePerKm = 45,
    baseSeconds = 30,
    secondsPerKm = 18,
    maxSeconds = 1800,
}
```

During transfer, the vehicle enters a non-stored transit state and should not be spawned from another garage until the transfer resolves.

For integrations, always request the current state from Nord Garages rather than assuming the framework garage column alone describes live transfer status.

Showroom

Showroom

Nord Garages v2 uses native GTA interior presets. No shell resource is required for the included showroom system.

Default group presets:

- Land: `vehicle_warehouse`
- Boat: `yacht_marina`
- Air: `smuggler_hangar`

Isolation

Each showroom session is isolated using routing buckets based on `Config.Showroom.routingBucketBase`.

Vehicle behaviour

Showroom vehicles are preview/display entities. They must not be treated as normal spawned owned vehicles by inventory, target or garage logic.

Useful client exports:

```
exports['nord_garages']:IsInShowroom()
exports['nord_garages']:IsShowroomVehicle(vehicle)
exports['nord_garages']:GetShowroomVehicleData(vehicle)
```

Temporary keys

When Nord Vehicle Keys is active, showroom access can grant session-only temporary key access without creating a permanent physical key item. Existing owner/shared keys are preserved.

Vehicle States

Vehicle States

Core integrations should understand the Nord state machine.

Common states include:

- `stored`
- `spawning`
- `active`
- `parked`
- `transit`
- `impounded`
- `seized`
- `service`
- `detached`

Detached

`detached` is intentionally used when an owned vehicle is removed from all garages without a valid live entity. It prevents the row from appearing as a phantom active/stored vehicle.

State safety

External scripts should prefer `SetVehicleState` or dedicated exports instead of direct SQL updates. The resource records normal state changes in `nord_garage_history` and notifies live integrations.

Impound System

Impound Overview

Impound Overview

Impounds are deliberately separate from normal garage locations.

`GetGarages()` does not include impound yards. Use `GetImpounds()` for impound-specific integrations.

Core features

- Public recovery yards.
- Job-managed impounds.
- Field impound action for authorised jobs.
- Configurable fines.
- Optional release delays.
- Automatic missing-entity recovery.
- Strict vehicle-type routing.
- History/audit entries.

Default configuration

Important values under `Config.Impound` include default fine, recovery fee, release time, missing-entity grace periods and optional preferred default impound.

Impound Vehicle Type Safety

Impound Vehicle Type Safety

Nord Garages validates vehicle type server-side before assigning an impound.

Expected routing:

- Land vehicle → compatible land impound.
- Boat → boat impound.
- Plane → compatible air/plane impound.
- Helicopter → compatible air/helicopter impound.

The resolver prioritizes reliable runtime/model information and framework metadata, then repairs known legacy type values where possible.

Existing incorrect impound assignments can be reconciled automatically through the impound reconciliation system.

Useful exports:

```
exports['nord_garages']:ResolveVehicleType(model, fallback, netId, secondaryModel)
exports['nord_garages']:AuditVehicleTypes()
exports['nord_garages']:ReconcileImpounds(options)
```

Job Impounds

Job Impounds

Job impound **locations and permissions are created in-game** from the admin interface.

Authorised employees can use the configured field command/key while near a vehicle.

Defaults:

```
jobFieldCommand = 'impoundvehicle'
jobFieldKey = 'F7'
jobFieldDistance = 6.0
jobFieldServerDistance = 10.0
```

The player may rebind the FiveM keybind through game settings.

Server exports

```
exports['nord_garages']:GetJobImpounds(source)
exports['nord_garages']:CanJobManageImpound(source, identifier)
exports['nord_garages']:JobImpoundVehicle(source, plate, impoundIdentifier, reason, fine,
releaseSeconds)
exports['nord_garages']:JobReleaseImpound(source, plate)
```

Job/grade/category validation remains server-side.

Automatic Recovery

Automatic Recovery

When enabled, vehicles that were active/parked but whose OneSync entity disappears may be routed to recovery/impound instead of silently returning for free.

Important grace values distinguish normal missing entities from disconnect/migration races.

Defaults include:

- Entity removed grace: 3 seconds
- Disconnect grace: 20 seconds
- Fallback scan interval: 10 seconds
- Missing grace: 20 seconds
- Startup grace: 15 seconds

Why the delays exist

OneSync ownership migration and player disconnects can briefly remove or move an entity. Immediate impound on every removal can create false positives, so Nord Garages uses short context-specific grace windows plus a slower fallback scan.

Explicit deletion

Server-side entity removal is monitored. Trusted staff integrations should use the dedicated staff exports so the desired garage state is explicit rather than relying only on entity disappearance detection.

Vehicle OS

VIN & Plate Identity

VIN & Plate Identity

Vehicle OS adds a permanent VIN identity that is independent of the current plate.

Why VIN exists

Plates can be changed. A VIN provides a stable identity for history, snapshots, access and service metadata even after a plate replacement.

Plate aliases

Old plates can be retained as aliases/history so integrations can resolve previous identities.

Useful exports:

```
exports['nord_garages']:GetVehicleIdentity(plateOrVin)
exports['nord_garages']:ChangeVehiclePlate(plateOrVin, newPlate, actor)
exports['nord_garages']:CompareVehicles({ valueA, valueB })
```

Do not change a framework plate directly without also reconciling Nord Garages; use the supported export for integrated workflows.

Snapshots & History

Snapshots & History

Vehicle snapshots provide recoverable copies of vehicle state such as properties, health, fuel and position.

```
local snapshots = exports['nord_garages']:GetVehicleSnapshots(plateOrVin, 20)
local history = exports['nord_garages']:GetVehicleHistory(plateOrVin, 50)

exports['nord_garages']:RestoreVehicleSnapshot(plateOrVin, snapshotId, actor)
```

Use cases

- Recover after a bad customization/integration write.
- Compare prior vehicle state.
- Admin investigation.
- Restore a known state without rewriting the ownership row manually.

Restores should be treated as administrative/integration operations and should be auditable via the actor value.

Advanced Access & Collections

Advanced Access & Collections

Vehicle access

Vehicle OS supports additional access roles such as co-owner, family, employee, temporary and valet.

```
exports['nord_garages']:GrantVehicleAccess(vehicle, identifier, role, expiresAt, actor)
exports['nord_garages']:RevokeVehicleAccess(vehicle, identifier, actor)
```

Temporary access may be constrained by the configured maximum duration.

Collections / favourites

```
exports['nord_garages']:SetVehicleCollection(source, vehicle, 'Daily', true)
```

Collections organize vehicles without changing ownership or garage state.

Reservations & Physical Slots

Reservations & Physical Slots

Reservations

Garage reservations can reserve capacity for a vehicle/user for a limited period.

```
exports['nord_garages']:CreateGarageReservation(source, garageIdentifier, vehicleId, minutes)
exports['nord_garages']:GetGarageReservations(garageIdentifier)
```

Expired/invalid reservations are ignored or cleaned by the database/runtime safety logic.

Physical slots

Vehicle OS includes a backend for named physical parking slots/floors.

```
exports['nord_garages']:GetGarageSlots(identifier)
exports['nord_garages']:UpsertGarageSlot(identifier, slot)
```

Slot data can include label, floor, position, allowed vehicle types, enabled state and sort order.

Service, Insurance, Stolen & EV Data

Service, Insurance, Stolen & EV Data

Vehicle OS exposes provider-neutral metadata APIs.

```
exports['nord_garages']:SetVehicleServiceData(vehicle, data, actor)
exports['nord_garages']:SetVehicleInsurance(vehicle, data, actor)
exports['nord_garages']:SetVehicleStolen(vehicle, true, actor, details)
exports['nord_garages']:SetVehicleEVData(vehicle, data, actor)
exports['nord_garages']:GetMaintenanceStatus(vehicle)
```

These APIs allow mechanic, insurance, police and EV resources to integrate without directly coupling to Nord Garages internal SQL schema.

The exact metadata object structure can be defined by the integrating provider; keep payloads JSON-serializable and avoid storing huge binary data.

Recovery Center & Analytics

Recovery Center & Analytics

Recovery Center

```
local issues = exports['nord_garages']:GetRecoveryCenter()
exports['nord_garages']:RepairVehicleState(vehicle, action, garageIdentifier, actor)
```

The recovery system can repair stale spawning/transit states, missing garage assignments and incompatible impound assignments using normal synchronization paths.

Intentional `detached` records are not treated as recovery errors.

Analytics

```
local analytics = exports['nord_garages']:GetGarageAnalytics(days)
```

Analytics data is stored separately from ownership state and is intended for operational dashboards/diagnostics.

Integrations and API

Framework Bridge

Framework Bridge

Framework-specific code is isolated under `client/bridge/` and `server/bridge/`.

Auto-detection order is:

1. Qbox / QBX
2. QBCore
3. ESX
4. Nord Core
5. Standalone

Framework ownership/economy is bridged while Nord Garages keeps its own authoritative runtime state.

Framework table mapping

The configuration contains default table mappings for ESX and QB/QBX. If your server uses a heavily customized ownership schema, adjust only the bridge/table mapping instead of rewriting the garage core.

Nord Core

The included bridge expects a `GetPlayer(source)` style export. If your local Nord Core API differs, adapt the Nord bridge file while preserving the core API contract.

Keys, Fuel & Inventory

Keys, Fuel & Inventory

Nord Vehicle Keys

When `nord_vehiclekeys` is used, Nord Garages integrates server-authoritatively.

Typical API usage includes registering the owned network entity, checking keys/physical key items and granting keys only when needed.

With `Config.Keys.keepOwnerKeysOnStore = false`, storing/impounding removes the physical vehicle key lifecycle where appropriate, and spawning recreates valid owner access without duplicating physical key items.

Fuel

The fuel bridge restores the saved fuel value during entity initialization, then stops writing so the active fuel resource controls normal consumption.

`lc_fuel` receives extra initialization safety to avoid persisting an uninitialized decorator as a false `0%` tank.

Inventory bridge API

Internally normalized operations include:

```
NordInventory.GetCount(source, item, metadata)
NordInventory.HasItem(source, item, amount, metadata)
NordInventory.GetItem(source, item, metadata)
NordInventory.AddItem(source, item, amount, metadata, slot, reason)
```

```
NordInventory.RemoveItem(source, item, amount, metadata, slot, reason)
NordInventory.RegisterStash(id, label, slots, weight, owner, groups, coords)
NordInventory.OpenStash(source, id, owner)
```

Client Exports

Client Exports

Garage UI

```
exports['nord_garages']:OpenGarage(identifier)
exports['nord_garages']:CloseGarage()
local garages = exports['nord_garages']:GetGarages()
```

Impounds

```
exports['nord_garages']:OpenImpound(identifier)
local impounds = exports['nord_garages']:GetImpounds()
exports['nord_garages']:OpenImpoundManager()
exports['nord_garages']:OpenJobImpoundManager()
```

Admin permission for management remains server-side.

Showroom helpers

```
exports['nord_garages']:IsInShowroom()
exports['nord_garages']:IsShowroomVehicle(vehicle)
exports['nord_garages']:GetShowroomVehicleData(vehicle)
```

Use these to ensure target/inventory resources ignore preview entities where appropriate.

Server Exports — Garages & Vehicles

Server Exports — Garages & Vehicles

Garage lookup/registration

```
exports['nord_garages']:GetGarage(identifier)
exports['nord_garages']:GetGarages(true)
exports['nord_garages']:GetImpounds(true)
exports['nord_garages']:GetGarageTemplates()
exports['nord_garages']:RegisterGarage(data)
```

Fleet

```
exports['nord_garages']:RegisterFleetVehicle(job, garageIdentifier, vehicleData)
```

Vehicle state

```
exports['nord_garages']:SetVehicleState(plate, state, garageIdentifier, details)
exports['nord_garages']:ImpoundVehicle(plate, garageIdentifier, reason, releaseAt, fine)
exports['nord_garages']:ReleaseVehicle(plate, garageIdentifier)
```

Staff-safe garage mutations

```
exports['nord_garages']:AddVehicleToGarage(plate, garageIdentifier, options)
exports['nord_garages']:ForceStoreVehicle(plate, garageIdentifier, options)
exports['nord_garages']:RemoveVehicleFromGarage(plate, options)
exports['nord_garages']:DetachVehicleFromGarages(plate, options)
exports['nord_garages']:StaffDeleteWorldVehicle(plate, garageIdentifier, options)
exports['nord_garages']:GetVehicleGarageState(plate)
```

All trusted mutation integrations should check the returned `ok`/`error` fields.

Server Exports — Admin, Staff & Sync

Server Exports — Admin, Staff & Sync

Admin access

```
AddAdmin(value)
RemoveAdmin(value)
IsAdmin(value)
GetRuntimeAdmins()
```

Compatibility aliases:

```
GiveGaragesAccess(value)
RemoveGaragesAccess(value)
HasGaragesAccess(value)
```

Staff snapshots

```
local rows = exports['nord_garages']:GetStaffVehicleSnapshot(limit)
```

The helper is designed for bulk authoritative reads instead of one export call per vehicle.

Framework reconciliation

```
exports['nord_garages']:ReconcileOwnedVehicleByPlate(plate, options)
exports['nord_garages']:SyncAllFrameworkVehicles(limit, force)
exports['nord_garages']:ReconcileImpounds(options)
```

Use these after trusted staff/ownership operations when another resource has changed the framework ownership table.

Ownership deletion

`DeleteOwnedVehicleRecord` is intended for explicit staff ownership deletion. Do not confuse this with `StaffDeleteWorldVehicle`, which is **world-only** and does not delete framework ownership.

Server Exports — Vehicle OS

Server Exports — Vehicle OS

Main exports include:

```
RegisterVIPProvider(name, handler)
RegisterProvider(kind, name, handler)
GetProviders()
IsPersistentParkingVIP(source)
GetVehicleIdentity(value)
ChangeVehiclePlate(value, newPlate, actor)
CompareVehicles(values)
SearchVehicles(source, query, limit)
GetVehicleHistory(value, limit)
GetVehicleSnapshots(value, limit)
RestoreVehicleSnapshot(value, snapshotId, actor)
GrantVehicleAccess(value, identifier, role, expiresAt, actor)
RevokeVehicleAccess(value, identifier, actor)
SetVehicleCollection(source, value, collection, favorite)
CreateGarageReservation(source, garage, vehicleId, minutes)
GetGarageReservations(garage)
GetGarageSlots(identifier)
UpsertGarageSlot(identifier, slot)
SetVehicleServiceData(value, patch, actor)
SetVehicleInsurance(value, patch, actor)
SetVehicleStolen(value, stolen, actor, details)
SetVehicleEVData(value, patch, actor)
GetMaintenanceStatus(value)
TowVehicleToImpound(value, garage, reason, actor, fine)
GetRecoveryCenter()
RepairVehicleState(value, action, garage, actor)
GetGarageAnalytics(days)
RestorePersistentParking()
```

```
RequestVehicleDelivery(source, plate, mode)
```

External resources should use these contracts rather than querying auxiliary Vehicle OS tables directly.

Events & Live Sync

Events & Live Sync

Nord Garages emits events so Staff, Phone and other integrations can refresh without polling the full database continuously.

Vehicle changed

```
AddEventHandler('nord_garages:server:vehicleChanged', function(payload)
    -- refresh local integration cache
end)
```

Vehicle OS event bus

```
AddEventHandler('nord_garages:server:event', function(eventName, payload)
end)

AddEventHandler('nord_garages:server:vehicleChangedV2', function(payload)
end)
```

Garage changes

Garage create/update/delete operations emit the garage change flow used by live admin/staff integrations.

Integration recommendation

Prefer event-driven cache invalidation plus occasional reconciliation. Avoid high-frequency SQL polling for every player/vehicle.

Nord Phone Integration

Nord Phone Integration

Server-side phone exports provide UI-safe normalized vehicle data.

```
local vehicles = exports['nord_garages']:GetPhoneVehicles(source)
local vehicle = exports['nord_garages']:GetPhoneVehicle(source, plate)
local garages = exports['nord_garages']:GetPhoneGarages()
```

The normalized vehicle snapshot can include plate, model/label, state, garage or impound identity, location, fuel and impound metadata.

A notification helper is also available:

```
exports['nord_garages']:NotifyPhoneVehicleChanged(ownerIdentifier, plate, action,
garageIdentifier)
```

Vehicle history/state changes invalidate the live phone cache so store, spawn, transfer, impound, release and reconciliation operations can refresh without a resource restart.

Nord Staff Integration

Nord Staff Integration

Nord Staff should use the trusted server exports instead of directly manipulating Nord Garages rows.

Give/store vehicle

After creating framework ownership, call `ReconcileOwnedVehicleByPlate` or `AddVehicleToGarage` so the new vehicle appears immediately without restart.

DV / remove from world

Use:

```
exports['nord_garages']:StaffDeleteWorldVehicle(plate, nil, {
    source = source,
    netId = netId,
})
```

This reconciles world tracking/garage state **without deleting ownership**.

Delete ownership

Use the explicit ownership deletion export only when the staff action genuinely removes the vehicle from the player's ownership database.

Global list

Use `GetStaffVehicleSnapshot(limit)` for bulk state and `SyncAllFrameworkVehicles(limit, force)` when a full framework sync is required.

Database and AutoSQL

AutoSQL & Schema

AutoSQL & Schema

Nord Garages automatically creates and migrates its schema during resource startup.

The stabilization line includes a schema migration that also verifies the Vehicle OS auxiliary tables/columns/indexes, which helps repair older or partially upgraded installations.

Core tables

- `nord_garages`
- `nord_garage_points`
- `nord_garage_vehicles`
- `nord_garage_history`
- `nord_garage_schema`

Vehicle OS tables

- `nord_garage_plate_aliases`
- `nord_garage_vehicle_snapshots`
- `nord_garage_vehicle_access`
- `nord_garage_vehicle_collections`
- `nord_garage_reservations`
- `nord_garage_slots`
- `nord_garage_analytics`

Normal users should let AutoSQL manage these tables.

Core Tables Explained

Core Tables Explained

nord_garages

Stores garage definition data: identifier, name, garage type, categories, access, settings, blip/NPC data and published state.

nord_garage_points

Stores world points linked to a garage identifier, including position, point type and metadata.

nord_garage_vehicles

Stores Nord's normalized vehicle state including owner, plate, model/type, garage assignment, state, properties, health, fuel, mileage, last position, sharing/metadata and current tracking identifiers.

Vehicle OS extends this table with VIN, slot, service, insurance, EV and stolen-state fields.

nord_garage_history

Append-style operational history for vehicle state/actions.

nord_garage_schema

Tracks migration/schema version. Do not manually reduce the version number on a live database.

Database Safety

Database Safety

Do not edit active state casually

Directly changing a vehicle row while the resource is running can desynchronize:

- Framework ownership state
- OneSync entity tracking
- Keys
- Phone/Staff caches
- Impound rules
- Vehicle OS history

Use server exports wherever possible.

Backups

Before a major upgrade, back up both:

1. Your framework ownership table (`owned_vehicles` or `player_vehicles`).
2. All `nord_garage_*` tables.

Stale spawning state

Startup SQL repairs stale `spawning` rows back to `stored` so a crash/restart does not permanently lock a vehicle.

Legacy showroom points

Older `showroom` and `camera` point types are removed because v2 uses the native showroom implementation.

Troubleshooting and Production

Common Problems

Common Problems

Admin command does nothing

Check:

- `Config.AdminCommand`
- `Config.AdminAce`
- ACE assignment in `server.cfg`
- Server console errors

Garage is not visible

Check that the garage is published, has valid points, correct access rules and compatible categories.

Vehicle does not appear

Check framework ownership first, then run the trusted reconciliation path. For staff-created vehicles, ensure Nord Garages is notified after the ownership row is created.

Vehicle appears in wrong impound

Use the impound reconciliation/type audit exports and verify the vehicle model/class is correct. Separate plane and helicopter yards must match their configured categories.

Fuel restores as 0

Confirm the configured fuel provider is actually running before Nord Garages and that no second resource overwrites fuel immediately after spawn.

Keys duplicate or disappear

Start your key resource before Nord Garages. With Nord Vehicle Keys, avoid granting/removing permanent physical keys client-side from a second integration.

UI opens but data is stale

Check live sync events between Staff/Phone/custom integrations rather than forcing constant UI reopen/restart behaviour.

Diagnostics & Recovery

Diagnostics & Recovery

Useful server-side diagnostic exports:

```
AuditVehicleTypes()
ReconcileImpounds(options)
GetRecoveryCenter()
RepairVehicleState(value, action, garage, actor)
GetVehicleGarageState(plate)
GetStaffVehicleSnapshot(limit)
SyncAllFrameworkVehicles(limit, force)
GetProviders()
```

Recovery Center examples

Recovery can address stale transient states and missing/incompatible garage assignments through the normal state synchronization flow.

Production logging

Keep `Config.Debug = false` normally. Enable it only while reproducing a problem, collect the relevant console output, then disable it again.

What runtime testing must cover

Static validation cannot reproduce every combination of framework build, MLO geometry, OneSync ownership migration, fuel/key version or existing database contents. Always perform a staging smoke test before replacing a busy live resource.

Production Checklist

Production Checklist

Before release to players:

- [ ] `oxmysql` starts before Nord Garages.
- [ ] Framework auto-detection resolves the intended core.
- [ ] Correct key, fuel and inventory providers resolve.
- [ ] `/garageadmin` is restricted to the intended ACE/admin source.
- [ ] At least one land garage stores/spawns correctly.
- [ ] Boat/air garages are tested if used.
- [ ] Impounds are tested separately from normal garages.
- [ ] Job impound grades/categories are verified server-side.
- [ ] Capacity `0` behaves as unlimited where intended.
- [ ] Vehicle transfer completes and restores state.
- [ ] Physical keys do not duplicate across store/spawn cycles.
- [ ] Fuel persists across store/spawn.
- [ ] Showroom preview vehicles do not leak into normal target/inventory logic.
- [ ] Staff DV and ownership deletion use different integration paths.
- [ ] Phone/Staff lists refresh without resource restart.
- [ ] VIP persistent parking is restricted as intended.
- [ ] Server restart restores/repairs vehicle state correctly.
- [ ] Database backup exists before rollout.

Changelog

Changelog

v2.0.0 — Vehicle OS

v2.0.0 — Vehicle OS

Nord Garages 2.0 preserves ESX/QB/QBX ownership compatibility while adding the Vehicle OS layer.

Added

- Permanent VIN identity.
- Plate alias/history support.
- Authoritative state engine and duplicate guard.
- VIP Persistent Parking with OneSync restoration watchdog.
- Vehicle Delivery and Valet configuration.
- Vehicle snapshots and restore API.
- Advanced access roles.
- Collections/favourites.
- Garage reservations.
- Physical parking slot/floor backend.
- Dynamic distance-based transfers.
- Service/mileage, insurance, stolen-state and EV metadata APIs.
- Recovery Center.
- Analytics.
- Provider registry.
- Smart vehicle search.
- Tow-to-impound API.
- Expanded Staff and Phone live synchronization.

v2.0.0 Stabilization Pass

v2.0.0 Stabilization Pass

The stabilization pass keeps existing gameplay logic while hardening failure paths.

Hardened

- Store flow keeps entity tracking until validation and authoritative SQL state change succeed.
- Store mutation rejects conflicting concurrent impound/seizure/transit/service/detached changes instead of overwriting them.
- Impound payment/release uses an atomic transition and can refund the fine if another operation changed state first.
- AutoSQL migrations cover Vehicle OS auxiliary tables and missing legacy columns/indexes.
- Invalid legacy reservations without expiry are cleaned.
- Persistent vehicle fallback scanning is less aggressive while state-bag updates remain immediate.
- NUI POST handling rejects malformed/non-object JSON responses before downstream UI code uses them.

Validation performed

- NUI JavaScript syntax validation.
- Local NUI asset reference checks.
- Internal network event registration/trigger cross-check.
- Merge/conflict marker checks.
- Resource/fxmanifest layout review.

Runtime-specific framework/MLO behaviour still requires staging-server validation.