Nord Doorlock

Complete documentation for Nord Doorlock, including installation, administration, configuration, persistence, permissions, developer exports, authorization hooks, security and troubleshooting.

- Getting Started
  - Overview
  - Requirements & Compatibility
  - Installation
  - Quick Start
- Administration
  - Admin Panel & Wizard
  - Door Interaction & Visuals
  - Permissions
  - Lockpicking
- Configuration
  - Full Configuration Reference
  - Bridges & Integrations
  - Storage & Persistence
  - Commands & Diagnostics
- Developer API
  - Server Exports
  - Client Exports
  - Authorization Hooks
  - Door Data Structure
  - Integration Examples
- Operations
  - Duplicate Door Protection

- Security Model
- Troubleshooting
- FAQ

Getting Started

Getting Started documentation for Nord Doorlock.

Overview

Nord Doorlock is a modular FiveM door-locking resource designed for server owners and developers who need in-game door management, persistent state, framework bridges, double-door support, access control and a developer API.

It is standalone at its core and does not require ox_doorlock.

Core Features

- In-game administrator panel opened with `/doors`
- Wizard-based door creation and editing
- Aim-and-capture placement
- Single doors and double doors
- GTA DoorSystem synchronization
- Locked/unlocked state synchronization for all clients
- Persistent doors across resource and server restarts
- Job and minimum-grade access rules
- Identifier access rules
- ACE permission access rules
- Optional lockpicking
- Automatic relocking
- Native world-space locked/unlocked icons
- Custom interaction NUI
- Duplicate physical-door protection
- Automatic duplicate cleanup
- JSON/KVP persistence without a database dependency
- Optional oxmysql persistence
- ESX, QB-Core, QBX, nord-core and standalone framework support
- Notification, Text UI, Target and Inventory bridges
- Server-authoritative access and distance validation
- Client and server exports
- Custom authorization hooks

Design Principles

Door state changes are authoritative on the server. The client may request a lock, unlock or lockpick action, but permissions, distance and persistence are validated server-side.

Door definitions are copied when returned through the developer API. This allows external resources to inspect them without accidentally mutating Nord Doorlock's internal state.

The same physical GTA door object cannot belong to multiple Nord Doorlock entries. This prevents invisible duplicate locks and conflicting DoorSystem registrations.

Requirements & Compatibility

Nord Doorlock can run without a framework or database.

Required

- FiveM server
- GTA V game build supported by your FiveM server
- The resource folder must be named `nord_doorlock` if you use the examples exactly as shown

Optional Integrations

Category	Supported Providers
Framework	Standalone, ESX, QB-Core, QBX, nord-core
Storage	KVP/JSON, oxmysql
Notifications	Standalone fallback, ox_lib, QB-Core, ESX, nord_notify
Text UI	Built-in NUI, ox_lib, nord_textui
Target	ox_target, qb-target, nord_target
Inventory	nord_inventory, ox_inventory, qb-inventory/QB-Core, ESX

Automatic Detection

Most integration settings support:

```
Config.Framework = 'auto'
Config.Notify = 'auto'
```

```
Config.TextUI = 'auto'
Config.Target = 'auto'
Config.Inventory = 'auto'
```

The resource checks running providers and selects the first compatible integration.

For production servers, explicitly setting the provider can make the configuration easier to understand and prevent an unintended provider from being selected after resource changes.

ox_lib

ox_lib is optional. Nord Doorlock only uses it when the relevant global functions exist.

If ox_lib is used for lockpicking, `lib.skillCheck` is used. Without it, the configured fallback success chance is used.

oxmysql

oxmysql is only required when:

```
Config.Storage = 'mysql'
```

If `Config.Storage = 'auto'`, MySQL is selected when oxmysql is started; otherwise Nord Doorlock uses its KVP/JSON persistence mode.

Installation

1. Install the Resource

Place the resource in your FiveM resources directory:

```
resources/[nord]/nord_doorlock
```

Keep the resource folder name as:

```
nord_doorlock
```

2. Start the Resource

Add to `server.cfg`:

```
ensure nord_doorlock
```

When using optional integrations, start them before Nord Doorlock. Example:

```
ensure oxmysql
ensure ox_lib
ensure ox_target
ensure nord_doorlock
```

Only include resources you actually use.

3. Administrator Permission

The default administrator ACE is:

```
Config.AdminAce = 'nord.doorlock.admin'
```

Example:

```
add_ace group.admin nord.doorlock.admin allow
```

Framework administrator groups can also be accepted through:

```
Config.AdminGroups = {
    admin = true,
    superadmin = true,
    god = true
}
```

4. Select Storage

Default:

```
Config.Storage = 'json'
```

This uses FiveM Resource KVP as the authoritative persistence source and mirrors the data to JSON files.

For MySQL:

```
Config.Storage = 'mysql'
```

Start oxmysql before Nord Doorlock. The table is automatically created.

An optional SQL installer is also provided:

```
sql/install.sql
```

5. Start and Verify

Restart the resource:

```
restart nord_doorlock
```

Then check persistence health:

```
doors_storage
```

A healthy result should report that storage is ready and show no active error.

6. Open the Door Manager

In game:

```
/doors
```

If the menu does not open, verify your ACE or framework admin group.

Quick Start

This is the shortest workflow for creating a working door.

Create a Door

Open:

```
/doors
```

Select **New door**.

The wizard contains three steps.

Step 1 — General

Configure:

- Door name
- Interaction distance
- Auto-lock delay
- Default locked state
- Lockpick enabled/disabled

Step 2 — Door Objects

Capture **Door 1**.

Aim the crosshair at the physical GTA door object. When the object is valid it is highlighted and the capture indicator changes state.

Press:

```
E
```

For a double door, repeat the process for **Door 2**.

Step 3 — Access

Optionally add:

- Jobs and minimum grades
- Identifiers
- ACE permissions

Leave all access fields empty for a public door.

Finish and save the door.

Test It

Walk close to the door.

The default standalone interaction key is:

```
E
```

The default lockpick key is:

```
H
```

Lockpicking only appears when it is globally enabled and enabled for that specific door.

Confirm Persistence

After creating the door:

```
doors_storage
```

Then restart:

```
restart nord_doorlock
```

Open `/doors` again. The saved door should still exist.

Administration

Administration documentation for Nord Doorlock.

Admin Panel & Wizard

The administrator UI is designed to keep the main screen focused on saved door cards. Creation and editing are handled through a modal wizard.

Opening the Panel

Default command:

```
/doors
```

Configured with:

```
Config.AdminCommand = 'doors'
```

Access is validated on the server through the configured ACE and framework administrator groups.

Door Cards

Each saved door is displayed as a card. A card can show information such as:

- ID
- Name
- Locked/unlocked state
- Single or double door
- Object count
- Interaction distance
- Public or restricted access
- Auto-lock
- Lockpick availability

Click a door card to edit it.

Wizard — General

The General step controls the basic door definition.

Name

Friendly identifier used by administrators and developer exports.

Example:

```
Mission Row Front Entrance
```

Interaction Distance

Defines how close a player must be for normal interaction.

Server-side distance validation adds the configured tolerance:

```
Config.ServerDistanceTolerance = 2.0
```

Auto Lock

Number of seconds before an unlocked door automatically locks again.

Use:

```
0
```

to disable automatic relocking.

Default Locked

Determines the default state stored in the door definition.

Lockpick Enabled

Allows that door to participate in the lockpick workflow when global lockpicking is enabled.

Wizard — Door Objects

A door can contain one or two physical objects.

Door 1 is required.

Door 2 is optional and is intended for double doors.

The capture system uses multiple object-detection methods so solid doors, barred gates and some MLO door objects can be captured.

Wizard — Access

Access can be defined by jobs, identifiers and ACE permissions.

The access types are additive alternatives: a player only needs one valid access route.

An administrator is also allowed through the configured admin permission system.

Editing Existing Doors

Open a saved card, change the required values and save.

The server checks whether the new definition overlaps a physical door already owned by another entry. Conflicting definitions are rejected.

Deleting Doors

Deleting a door removes the persistent definition, broadcasts a full synchronization and retires its old DoorSystem hashes on clients.

Door Interaction & Visuals

Nord Doorlock can provide interaction through its built-in NUI, supported target systems or supported Text UI providers.

Interaction Key

Default:

```
Config.ToggleKey = 38 -- E
```

Interaction Distances

```
Config.InteractionDistance = 2.0
Config.DoorUIShowDistance = 6.0
Config.DoorUIInteractDistance = 2.0
```

`DoorUIShowDistance` controls how far away the interaction UI may become visible.

`DoorUIInteractDistance` controls the custom NUI interaction range.

Each door also stores its own `distance`.

Custom Door NUI

Enable:

```
Config.CustomDoorNUI = true
```

The custom NUI displays the current door action and state near the door.

Native Door State Icon

Enable:

```
Config.ShowDoorStateIcon = true
```

Default icon visibility distance:

```
Config.DoorStateIconDistance = 6.0
```

The resource uses GTA streamed textures:

```
'mpsafecracking', 'lock_open'
'mpsafecracking', 'lock_closed'
```

The icon is drawn in world space and disappears outside the configured distance.

Duplicate Icon Protection

```
Config.DoorStateIconMergeDistance = 0.35
```

Nearby icon centers are merged so the same physical location does not display multiple lock icons.

Door Closing Behaviour

```
Config.DoorCloseRate = 10.0
Config.DoorCloseTimeoutMs = 2500
Config.DoorCloseSnapFallback = true
Config.DoorPhysicalFallback = true
```

When locking, Nord Doorlock attempts to close the door before applying the final physical lock fallback. This avoids freezing a door while it is still open.

`DoorCloseSnapFallback` may use the captured heading after the normal close timeout for doors that do not correctly report their DoorSystem close state.

`DoorPhysicalFallback` adds an entity freeze only after the door is considered closed.

Permissions

Door access can be restricted by jobs, identifiers and ACE permissions.

If all access fields are empty, the door is public.

Jobs & Grades

Wizard format:

```
police:0, ambulance:2
```

This means:

Job	Minimum Grade
police	0
ambulance	2

A player with the matching job must meet or exceed the minimum grade.

Door data representation:

```
access = {
    jobs = {
        police = 0,
        ambulance = 2
    }
}
```

Identifiers

Add one identifier per line.

Example:

```
license:xxxxxxxxxxxxxxx
discord:123456789012345678
```

Door data:

```
access = {
    identifiers = {
        'license:xxxxxxxxxxxxxxx',
        'discord:123456789012345678'
    }
}
```

ACE Permissions

Add one ACE string per line.

Example:

```
door.pd
door.staff
```

Then grant the ACE in your server configuration as appropriate.

Example:

```
add_ace group.police door.pd allow
```

Administrator Access

Administrators can be recognized through:

```
Config.AdminAce = 'nord.doorlock.admin'
```

and/or:

```
Config.AdminGroups = {
    admin = true,
```

```
    superadmin = true,
    god = true
}
```

Custom Authorization

External scripts can extend or override authorization with the `doorAuthorization` hook.

See the **Authorization Hooks** page.

Server Authority

Do not trust a client-side permission check for security-sensitive logic.

`HasAccess` is available as a server export and should be preferred when another server resource needs to determine whether a player may use a Nord Doorlock door.

Lockpicking

Nord Doorlock includes an optional lockpick workflow.

Enable Globally

```
Config.EnableLockpick = true
```

Enable Per Door

Enable **Lockpick** in the door wizard.

Door structure:

```
lockpick = {
    enabled = true
}
```

Default Key

```
Config.LockpickKey = 74 -- H
```

Lockpick Configuration

```
Config.Lockpick = {
    item = 'lockpick',
    consumeOnSuccess = false,
    consumeOnFail = true,
```

```
    fallbackSuccessChance = 45,
    oxDifficulty = { 'easy', 'easy', 'medium' },
    oxInputs = { 'w', 'a', 's', 'd' }
}
```

item

Inventory item required before the attempt starts.

consumeOnSuccess

When `true`, one configured lockpick item is removed after a successful attempt.

consumeOnFail

When `true`, one configured lockpick item is removed after a failed attempt.

fallbackSuccessChance

Percentage used when an ox_lib skill check is unavailable.

oxDifficulty / oxInputs

Passed to `lib.skillCheck` when supported.

Server Validation

Before starting a lockpick attempt, the server checks:

- The door exists
- The player is near the door
- Lockpicking is globally enabled
- Lockpicking is enabled for the door
- The inventory bridge reports the required item

The server stores a short-lived pending attempt and validates it again when the client returns the result.

Standalone Inventory Warning

The default standalone inventory fallback cannot verify a real inventory item and therefore returns success for inventory checks.

If your standalone server uses a custom inventory, integrate it in:

```
server/bridge/inventory.lua
```

This is strongly recommended when lockpick items must be enforced.

Configuration

Configuration documentation for Nord Doorlock.

Full Configuration Reference

Main configuration file:

```
config/config.lua
```

Locale

```
Config.Locale = 'en'
```

Included locales:

```
en
pt
```

Storage

```
Config.Storage = 'json'
```

Values:

```
json
mysql
auto
```

`json` uses KVP as the authoritative store and JSON files as mirrors.

`mysql` requires oxmysql.

`auto` uses MySQL when oxmysql is running, otherwise KVP/JSON.

Base Distances & Defaults

```
Config.InteractionDistance = 2.0
Config.DrawDistance = 12.0
Config.DefaultDoorDistance = 2.0
Config.DefaultLocked = true
Config.DefaultAutoLock = 0
```

Door Closing

```
Config.DoorCloseRate = 10.0
Config.DoorCloseTimeoutMs = 2500
Config.DoorCloseSnapFallback = true
Config.DoorPhysicalFallback = true
```

Controls

```
Config.ToggleKey = 38
Config.LockpickKey = 74
```

Defaults:

```
E = toggle
H = lockpick
```

Feature Toggles

```
Config.EnableTarget = true
Config.EnableTextUI = true
Config.EnableLockpick = true
Config.CustomDoorNUI = true
Config.ShowDoorStateIcon = true
```

Native State Icon

```
Config.DoorStateIconDistance = 6.0
Config.DoorStateIconOffsetZ = 0.0
Config.DoorStateIconMergeDistance = 0.35
```

Sprite configuration:

```
Config.DoorStateSprites = {
    [0] = {
        'mpsafecracking',
        'lock_open',
        0.0, 0.0,
        0.018, 0.018,
        0.0,
        255, 255, 255, 180
    },

    [1] = {
        'mpsafecracking',
        'lock_closed',
        0.0, 0.0,
        0.018, 0.018,
        0.0,
        255, 255, 255, 180
    }
}
```

State keys:

```
0 = unlocked
1 = locked
```

Custom Door UI

```
Config.DoorUIShowDistance = 6.0
Config.DoorUIInteractDistance = 2.0
Config.DoorUIOffsetZ = 0.15
```

Administrator Settings

```
Config.AdminCommand = 'doors'
Config.AdminAce = 'nord.doorlock.admin'

Config.AdminGroups = {
    admin = true,
    superadmin = true,
    god = true
}
```

Integration Providers

```
Config.Framework = 'auto'
Config.Notify = 'auto'
Config.TextUI = 'auto'
Config.Target = 'auto'
Config.Inventory = 'auto'
```

Framework

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

Notify

```
auto
ox
qb
esx
```

```
nord
standalone
```

Text UI

```
auto
ox
nord
standalone
```

Target

```
auto
ox
qb
nord
standalone
```

Inventory

```
auto
ox
qb
esx
nord
standalone
```

Lockpick

```
Config.Lockpick = {
    item = 'lockpick',
    consumeOnSuccess = false,
    consumeOnFail = true,
    fallbackSuccessChance = 45,
```

```
    oxDifficulty = { 'easy', 'easy', 'medium' },
    oxInputs = { 'w', 'a', 's', 'd' }
}
```

Server Security

```
Config.RateLimitMs = 350
Config.ServerDistanceCheck = true
Config.ServerDistanceTolerance = 2.0
```

`RateLimitMs` reduces event spam.

`ServerDistanceCheck` enables authoritative proximity validation for player interactions.

`ServerDistanceTolerance` is added to the door's configured distance when validating the player server-side.

Capture

```
Config.AdminCaptureDistance = 15.0
Config.AdminCaptureRadius = 0.42
Config.AdminCaptureFallbackRadius = 1.35
Config.AdminCaptureScreenRadius = 0.075
```

The capture system can use a normal object ray, a capsule test and a camera-aligned object fallback.

Duplicate Door Detection

```
Config.DuplicateDoorTolerance = 0.15
```

The same physical door object cannot belong to two different door definitions.

Debug

```
Config.Debug = false
```

Enable temporarily when diagnosing persistence or synchronization problems.

Configuration

Bridges & Integrations

Nord Doorlock keeps integrations outside the core logic.

Client Bridges

```
client/bridge/framework.lua
client/bridge/notify.lua
client/bridge/textui.lua
client/bridge/target.lua
```

Server Bridges

```
server/bridge/framework.lua
server/bridge/inventory.lua
```

This structure allows a server owner or developer to add a custom provider without editing the main door synchronization logic.

Framework Bridge

Responsible for:

- Player job and grade
- Administrator group detection
- Framework player data

Supported:

```
ESX
QB-Core
```

```
QBX
nord-core
standalone
```

Notification Bridge

Supported automatic providers include:

```
ox_lib
nord_notify
QB-Core
ESX
standalone chat fallback
```

The notification type values used internally include:

```
inform
success
error
```

Text UI Bridge

Supported:

```
ox_lib
nord_textui
built-in NUI fallback
```

The standalone fallback is pure NUI and does not rely on GTA text natives.

Target Bridge

Supported:

```
ox_target
qb-target
```

```
nord_target
```

When no supported target is active, normal key/NUI interaction remains available.

The current nord_target bridge registers the standard toggle interaction. If your nord_target implementation should expose a separate lockpick option, extend `client/bridge/target.lua`.

Inventory Bridge

Supported:

```
nord_inventory
ox_inventory
QB inventory
ESX inventory
standalone fallback
```

Used primarily for the lockpick item.

Adding a Custom Provider

Prefer editing or extending the corresponding bridge file rather than placing provider-specific code in:

```
client/main.lua
server/main.lua
```

This keeps future updates easier to merge.

Configuration

Storage & Persistence

Nord Doorlock supports persistence without requiring a database.

JSON Mode

Default:

```
Config.Storage = 'json'
```

Despite the name, the authoritative store is FiveM Resource KVP.

KVP key:

```
nord_doorlock:doors:v2
```

The resource also mirrors data to:

```
data/doors.json
data/doors.backup.json
```

This design allows doors to survive resource/server restarts even when the resource directory is not writable.

Source Selection on Load

When using JSON mode, Nord Doorlock can read:

- KVP
- `data/doors.json`
- `data/doors.backup.json`

The newest valid source is selected.

Legacy JSON arrays are accepted and migrated to the newer persisted wrapper.

JSON Payload Shape

Current mirror structure:

```
{
  "version": 2,
  "savedAt": 1700000000,
  "doors": []
}
```

MySQL Mode

Configure:

```
Config.Storage = 'mysql'
```

oxmysql must be started.

The table is auto-created.

Optional SQL:

```
CREATE TABLE IF NOT EXISTS `nord_doorlock` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `data` LONGTEXT NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Each database row stores the door payload in the `data` column.

Auto Mode

```
Config.Storage = 'auto'
```

If oxmysql is running, MySQL is used.

Otherwise Nord Doorlock falls back to KVP/JSON.

Persistence Safety

For authoritative changes, the server saves before broadcasting success.

If persistence fails, state-changing API operations attempt to roll back the in-memory change rather than pretending the operation succeeded.

Storage Diagnostics

Console or administrator command:

```
doors_storage
```

Typical fields:

```
mode
ready
doors
error
```

Developer export:

```
local health = exports.nord_doorlock:GetStorageHealth()
```

Returned data includes values such as:

```
{
    mode = 'json',
    configuredMode = 'json',
    lastError = nil,
    kvpKey = 'nord_doorlock:doors:v2',
    ready = true,
    serverError = nil,
    doorCount = 10
}
```

Reloading From Persistence

Server export:

```
local ok, count, removedDuplicates =
    exports.nord_doorlock:ReloadDoors()
```

This reloads definitions, performs duplicate cleanup and broadcasts a full synchronization.

Commands & Diagnostics

/doors

Opens the in-game administrator panel.

Command name:

```
Config.AdminCommand = 'doors'
```

Default:

```
/doors
```

The server validates administrator permission before opening the panel.

doors_storage

Displays storage health.

Server console:

```
doors_storage
```

Authorized administrators can also run it in game.

Example information:

```
mode=json ready=true doors=12 error=none
```

Use this command when:

- Doors disappear after restart
- A save reports an error
- You are unsure which storage provider is active

- You have changed `Config.Storage`

doors_dedupe

Scans the currently loaded door definitions and removes overlapping physical duplicates.

```
doors_dedupe
```

This is useful after migrating from an older build where one physical door may have been registered multiple times.

The resource also performs duplicate cleanup while loading persisted doors.

Debug Mode

Enable:

```
Config.Debug = true
```

Use it temporarily to get extra storage/persistence logging.

Disable it again after troubleshooting on production servers unless the extra logging is required.

Developer API

Developer API documentation for Nord Doorlock.

Server Exports

Server exports provide the authoritative developer API.

API version:

```
local version = exports.nord_doorlock:GetApiVersion()
```

Current API version returned by the resource:

```
1
```

GetApiVersion

```
local version = exports.nord_doorlock:GetApiVersion()
```

Returns a number.

IsReady

```
local ready, errorMessage = exports.nord_doorlock:IsReady()
```

Returns:

```
boolean ready
string|nil errorMessage
```

Use this when another resource starts at the same time as Nord Doorlock.

GetDoor

```
local door = exports.nord_doorlock:GetDoor(id)
```

Returns a copy of the door table or `nil`.

GetDoors

```
local doors = exports.nord_doorlock:GetDoors()
```

Returns an array of copied door definitions sorted by ID.

GetDoorCount

```
local count = exports.nord_doorlock:GetDoorCount()
```

Returns the number of loaded doors.

GetDoorState

```
local locked = exports.nord_doorlock:GetDoorState(id)
```

Returns:

```
true  = locked
false = unlocked
nil   = door not found
```

GetDoorByName

Case-insensitive exact match:

```
local door = exports.nord_doorlock:GetDoorByName('Mission Row Front')
```

Returns the first matching door or `nil`.

FindDoorsByName

Case-insensitive partial search:

```
local doors = exports.nord_doorlock:FindDoorsByName('mission')
```

Returns an array.

FindDoorByCoords

```
local door, distance = exports.nord_doorlock:FindDoorByCoords(
    vector3(440.0, -998.0, 30.0),
    5.0
)
```

Coordinates can be a `vector3` or a table:

```
{ x = 440.0, y = -998.0, z = 30.0 }
```

Returns the nearest door within the radius and its distance.

GetNearestDoor

Server-side nearest door for a player:

```
local door, distance =
    exports.nord_doorlock:GetNearestDoor(source, 10.0)
```

Requires a valid player source.

HasAccess

```
local allowed = exports.nord_doorlock:HasAccess(source, doorId)
```

Returns `true` or `false`.

This includes the normal Nord Doorlock access rules and registered authorization hooks.

SetDoorState

```
local ok, stateOrError =
    exports.nord_doorlock:SetDoorState(1, true)
```

`true` means locked.

On success:

```
ok = true
stateOrError = true/false final state
```

On failure:

```
ok = false
stateOrError = error code
```

LockDoor

```
local ok, result = exports.nord_doorlock:LockDoor(1)
```

UnlockDoor

```
local ok, result = exports.nord_doorlock:UnlockDoor(1)
```

When the door has `autoLock > 0`, unlocking through the API schedules automatic relocking.

ToggleDoorState

```
local ok, result = exports.nord_doorlock:ToggleDoorState(1)
```

CreateDoor

```
local ok, doorOrError, duplicateId =
    exports.nord_doorlock:CreateDoor({
        name = 'Police Front Door',
        distance = 2.0,
        autoLock = 0,
        defaultLocked = true,
        locked = true,

        parts = {
            {
                model = 123456789,
                coords = {
                    x = 440.0,
                    y = -998.0,
                    z = 30.0
                },
                heading = 90.0
            }
        },

        access = {
            jobs = {
                police = 0
            },
            identifiers = {},
            ace = {}
        },

        lockpick = {
            enabled = false
        }
    })
```

On success:

```
ok = true
doorOrError = created door
```

Potential errors include:

```
storage_not_ready
invalid_payload
invalid_parts
invalid_part
invalid_part_values
physical_door_already_registered
persistence_failed
```

When the physical door is already registered, the third return value can contain the conflicting door ID.

UpdateDoor

Patch an existing door:

```
local ok, doorOrError, duplicateId =
    exports.nord_doorlock:UpdateDoor(1, {
        autoLock = 30,

        access = {
            jobs = {
                police = 2
            },
            identifiers = {},
            ace = {}
        }
    })
```

Fields not included in the patch are copied from the existing definition.

The `id` is supplied separately and is not overwritten by the patch.

DeleteDoor

```
local ok, errorMessage =
    exports.nord_doorlock:DeleteDoor(1)
```

On success, the door is removed from persistence and clients receive a full synchronization.

ReloadDoors

```
local ok, countOrError, removedDuplicates =
    exports.nord_doorlock:ReloadDoors()
```

On success:

```
ok = true
countOrError = current door count
removedDuplicates = number removed during reload
```

GetStorageHealth

```
local health = exports.nord_doorlock:GetStorageHealth()
```

See the **Storage & Persistence** page.

RegisterHook

```
local hookId, errorMessage =
    exports.nord_doorlock:RegisterHook(
        'doorAuthorization',
        function(payload)
            return nil
        end,
        {}
    )
```

See the **Authorization Hooks** page.

RemoveHooks

Remove one hook owned by the calling resource:

```
local ok = exports.nord_doorlock:RemoveHooks(hookId)
```

Remove all hooks owned by the calling resource:

```
local ok = exports.nord_doorlock:RemoveHooks()
```

Compatibility Aliases

Lowercase compatibility aliases also exist:

```
exports.nord_doorlock:registerHook(...)
exports.nord_doorlock:removeHooks(...)
```

For new integrations, prefer:

```
RegisterHook
RemoveHooks
```

Developer API

Client Exports

Client exports are intended for local door lookup, UI integrations and server-authoritative action requests.

Do not use client exports as a security boundary.

GetApiVersion

```
local version = exports.nord_doorlock:GetApiVersion()
```

GetDoor

```
local door = exports.nord_doorlock:GetDoor(id)
```

Returns a copied door table or `nil`.

GetDoors

```
local doors = exports.nord_doorlock:GetDoors()
```

Returns all locally synchronized doors sorted by ID.

GetDoorState

```
local locked = exports.nord_doorlock:GetDoorState(id)
```

Returns:

```
true
false
nil
```

IsDoorLocked

Alias-style state helper:

```
local locked = exports.nord_doorlock:IsDoorLocked(id)
```

GetDoorByName

```
local door = exports.nord_doorlock:GetDoorByName('Police Front Door')
```

Case-insensitive exact match.

FindDoorsByName

```
local doors = exports.nord_doorlock:FindDoorsByName('police')
```

Case-insensitive partial match.

GetDoorCenter

```
local coords = exports.nord_doorlock:GetDoorCenter(id)
```

Returns:

```
{
    x = 440.0,
    y = -998.0,
    z = 30.0
}
```

or `nil`.

For double doors, the logical center is calculated from both parts.

FindDoorByCoords

```
local door, distance =
    exports.nord_doorlock:FindDoorByCoords(
        GetEntityCoords(PlayerPedId()),
        5.0
    )
```

GetNearestDoor

Returns the nearest door currently tracked by Nord Doorlock's interaction loop:

```
local door, distance = exports.nord_doorlock:GetNearestDoor()
```

The result may be `nil` when no relevant nearby door is being tracked.

GetDoorEntities

```
local entities = exports.nord_doorlock:GetDoorEntities(id)
```

Returns an array indexed by door part.

Example:

```
local entities = exports.nord_doorlock:GetDoorEntities(12)

for index, entity in pairs(entities) do
    print(index, entity)
end
```

Only currently resolvable entities are returned.

RequestToggle

```
local sent = exports.nord_doorlock:RequestToggle(id)
```

This does **not** directly change the door state.

It sends the standard server request, where distance and access are validated.

Returns `false` if the local door ID does not exist.

RequestLockpick

```
local sent = exports.nord_doorlock:RequestLockpick(id)
```

Returns `false` if the local door does not exist or lockpicking is unavailable for that door.

The real validation still occurs server-side.

RequestSync

```
exports.nord_doorlock:RequestSync()
```

Requests a fresh full door synchronization from the server.

OpenAdmin

```
exports.nord_doorlock:OpenAdmin()
```

Requests the administrator panel.

The server still validates administrator permissions.

Authorization Hooks

Authorization hooks allow another server resource to extend or override door access without modifying Nord Doorlock.

Supported hook event:

```
doorAuthorization
```

Basic Example

```
local hookId = exports.nord_doorlock:RegisterHook(
    'doorAuthorization',
    function(payload)
        if payload.door.name == 'VIP Entrance' then
            return exports.my_vip:IsVip(payload.source)
        end
    end
)
```

Hook Payload

The callback receives:

```
{
    source = 12,
    door = {
        -- copied door definition
    },
    lockpick = false,
    authorised = true
}
```

source

Player server ID.

door

A copy of the current door definition.

lockpick

Boolean indicating the authorization context supplied by Nord Doorlock.

authorised

Authorization result calculated before this hook.

Return Values

Return:

```
true
```

to authorize.

Return:

```
false
```

to deny.

Return nothing / `nil` to leave the current result unchanged.

Multiple hooks can run. A boolean returned by a later matching hook becomes the current result for subsequent hooks.

Name Filter

Hook options may contain a Lua pattern:

```
local hookId = exports.nord_doorlock:RegisterHook(
    'doorAuthorization',
    function(payload)
        return exports.my_access:CanUseDoor(
            payload.source,
            payload.door.id
        )
    end,
    {
        nameFilter = '^Police'
    }
)
```

Only matching door names are evaluated by that hook.

`nameFilter` is used with Lua's `string.match`, so treat it as a Lua pattern rather than a plain substring.

Remove One Hook

```
exports.nord_doorlock:RemoveHooks(hookId)
```

Remove All Hooks Owned by Your Resource

```
exports.nord_doorlock:RemoveHooks()
```

A resource can only remove hooks registered under its own invoking resource identity.

Recommended Resource Cleanup

```lua
local hookId

CreateThread(function()
    hookId = exports.nord_doorlock:RegisterHook(
        'doorAuthorization',
        function(payload)
            -- custom access
        end
    )
end)

AddEventHandler('onResourceStop', function(resource)
    if resource ~= GetCurrentResourceName() then return end

    if hookId then
        exports.nord_doorlock:RemoveHooks(hookId)
    end
end)
```

Developer API

Door Data Structure

A Nord Doorlock door is represented as a Lua table.

Example — Single Door

```
{
    id = 1,
    name = 'Police Front Door',
    distance = 2.0,
    autoLock = 0,
    locked = true,
    defaultLocked = true,

    parts = {
        {
            model = 123456789,
            coords = {
                x = 440.1234,
                y = -998.5678,
                z = 30.1234
            },
            heading = 90.0
        }
    },

    access = {
        jobs = {
            police = 0
        },

        identifiers = {},
```

```
        ace = {}
    },

    lockpick = {
        enabled = false
    }
}
```

Example — Double Door

```
{
    name = 'Main Double Door',

    parts = {
        {
            model = 111111111,
            coords = {
                x = 100.0,
                y = 200.0,
                z = 30.0
            },
            heading = 90.0
        },

        {
            model = 222222222,
            coords = {
                x = 102.0,
                y = 200.0,
                z = 30.0
            },
            heading = 270.0
        }
    }
}
```

Field Reference

Field	Type	Description
`id`	number	Persistent internal ID
`name`	string	Friendly door name
`distance`	number	Interaction distance
`autoLock`	number	Seconds before relocking, `0` disables
`locked`	boolean	Current persistent state
`defaultLocked`	boolean	Default configured state
`parts`	array	One or two physical door parts
`access`	table	Jobs, identifiers and ACE rules
`lockpick`	table	Lockpick settings

Part Structure

```
{
    model = 123456789,

    coords = {
        x = 0.0,
        y = 0.0,
        z = 0.0
    },

    heading = 0.0
}
```

The captured heading is important for physical fallback behavior on doors that do not correctly close through GTA DoorSystem alone.

Access Structure

```
access = {
    jobs = {
        police = 0
    },

    identifiers = {
        'license:...'
    },

    ace = {
        'door.police'
    }
}
```

Lockpick Structure

```
lockpick = {
    enabled = true
}
```

Developer API Copies

`GetDoor`, `GetDoors` and similar read exports return copies.

Changing the returned table does not edit the actual Nord Doorlock definition.

Use `UpdateDoor` for persistent server-side edits.

Integration Examples

Lock a Door From Another Server Resource

```
local success, result =
    exports.nord_doorlock:LockDoor(12)

if not success then
    print(('Failed to lock door: %s'):format(result))
end
```

Unlock a Door Temporarily

If the door has an auto-lock delay, using `UnlockDoor` schedules its normal relock behavior.

```
exports.nord_doorlock:UnlockDoor(12)
```

Toggle by Door Name

```
local door =
    exports.nord_doorlock:GetDoorByName('Bank Main Entrance')

if door then
    exports.nord_doorlock:ToggleDoorState(door.id)
end
```

Check Player Access Before Running Another Action

```
RegisterNetEvent('my_script:server:secureAction', function(doorId)
    local src = source

    if not exports.nord_doorlock:HasAccess(src, doorId) then
        return
    end

    -- Run protected server-side action.
end)
```

Find a Door Near Coordinates

```
local door, distance =
    exports.nord_doorlock:FindDoorByCoords(
        vector3(440.0, -998.0, 30.0),
        10.0
    )

if door then
    print(('Found %s at %.2fm'):format(
        door.name,
        distance
    ))
end
```

Create a Job Door Programmatically

```
local ok, doorOrError, duplicateId =
    exports.nord_doorlock:CreateDoor({
        name = 'EMS Secure Room',
        distance = 2.0,
        locked = true,
        defaultLocked = true,

        parts = {
            {
                model = 123456789,
                coords = {
                    x = 300.0,
                    y = -600.0,
                    z = 43.0
                },
                heading = 180.0
            }
        },

        access = {
            jobs = {
                ambulance = 2
            },
            identifiers = {},
            ace = {}
        },

        lockpick = {
            enabled = false
        }
    })

if not ok then
    print('CreateDoor failed:', doorOrError, duplicateId)
end
```

Add Temporary Custom Access With a Hook

```
local hook = exports.nord_doorlock:RegisterHook(
    'doorAuthorization',
    function(payload)
        if payload.door.name == 'Event Room' then
            return GlobalState.eventRoomOpen == true
        end
    end
)
```

Client HUD Integration

```
CreateThread(function()
    while true do
        local door, distance =
            exports.nord_doorlock:GetNearestDoor()

        if door and distance then
            -- Update your HUD.
        end

        Wait(250)
    end
end)
```

Client Interaction Button

Do not set the local state yourself.

Use:

```
exports.nord_doorlock:RequestToggle(doorId)
```

This preserves server-side permissions and distance validation.

Operations

Operations documentation for Nord Doorlock.

Duplicate Door Protection

A physical GTA door object must only be owned by one Nord Doorlock definition.

Duplicate registrations can create conflicting DoorSystem states where one definition unlocks the object while another keeps it locked.

Nord Doorlock includes protections for this problem.

Duplicate Tolerance

```
Config.DuplicateDoorTolerance = 0.15
```

Door parts with the same model and coordinates inside the tolerance are considered the same physical object.

New Door Validation

When creating or editing a door, the server checks whether any captured part is already owned by another saved entry.

Conflicting definitions are rejected.

Developer API error:

```
physical_door_already_registered
```

The conflicting door ID can also be returned.

Startup Cleanup

Persisted definitions are scanned when they are loaded.

When an older single-door entry conflicts with a more complete double-door entry, the more complete definition is preferred.

Removed IDs are retired and sent to clients so stale GTA DoorSystem hashes can be unlocked and removed.

Manual Cleanup

Run:

```
doors_dedupe
```

This scans the current door list and removes duplicate physical registrations.

Duplicate Icons

World icon rendering also has a visual merge distance:

```
Config.DoorStateIconMergeDistance = 0.35
```

This is only a visual safety layer.

The server-side physical duplicate protection is the important protection against conflicting lock states.

Operations

Security Model

Nord Doorlock treats the server as authoritative for security-sensitive door actions.

Server-Side Validation

Normal player toggle requests are validated for:

- Door existence
- Rate limiting
- Player proximity
- Door access rules
- Authorization hooks

Lockpick requests add inventory and lockpick-enabled validation.

Distance Validation

```
Config.ServerDistanceCheck = true
Config.ServerDistanceTolerance = 2.0
```

Keep `ServerDistanceCheck` enabled on production servers unless you have a specific architecture that replaces it.

Rate Limiting

```
Config.RateLimitMs = 350
```

This limits repeated interaction requests from a source.

Client Exports

Functions such as:

```
RequestToggle
RequestLockpick
OpenAdmin
```

are requests, not privileged direct state changes.

The server validates them.

Server Exports

Server exports can perform authoritative state changes because server resources are considered trusted code.

Do not expose unrestricted server events in your own resource that directly forward arbitrary player input into:

```
CreateDoor
UpdateDoor
DeleteDoor
LockDoor
UnlockDoor
```

without your own authorization.

Admin Panel

Opening the panel is validated server-side using the configured administrator ACE and framework groups.

Persistence

Authoritative state changes are persisted before clients are told the change succeeded.

When persistence fails, Nord Doorlock attempts to restore the previous in-memory state.

Hooks

Authorization hooks execute server-side.

Only register trusted server callbacks.

Inventory Bridge

If lockpick item ownership matters, make sure your selected inventory bridge performs a real server-side item check.

The standalone fallback should be customized for servers with a custom inventory.

Troubleshooting

`/doors` Does Not Open

Check:

```
add_ace group.admin nord.doorlock.admin allow
```

Confirm:

```
Config.AdminAce = 'nord.doorlock.admin'
```

If using framework groups, verify `Config.AdminGroups`.

Door Disappears After Restart

Run:

```
doors_storage
```

Check that:

```
ready=true
error=none
```

In JSON mode, Nord Doorlock uses KVP as the authoritative store and JSON as a mirror.

Do not assume an empty `data/doors.json` means KVP is empty.

Developer check:

```
local health = exports.nord_doorlock:GetStorageHealth()
print(json.encode(health))
```

Same Door Is Locked Twice / Feels Locked From Behind

Run:

```
doors_dedupe
```

Also verify that the same physical object was not captured into two separate definitions.

Nord Doorlock normally blocks this automatically.

Two-Leaf Door Only Controls One Side

Edit the definition and verify that **Door 1** and **Door 2** are both captured.

Both physical objects must be stored in the same double-door definition.

Do not create each leaf as a separate overlapping Nord Doorlock entry.

Door Locks While Still Open

Relevant settings:

```
Config.DoorCloseRate = 10.0
Config.DoorCloseTimeoutMs = 2500
Config.DoorCloseSnapFallback = true
Config.DoorPhysicalFallback = true
```

For unusual MLO doors, increase the timeout before disabling physical fallback.

Cannot Capture a Barred Gate

The capture system uses:

```
Config.AdminCaptureRadius = 0.42
Config.AdminCaptureFallbackRadius = 1.35
Config.AdminCaptureScreenRadius = 0.075
```

Aim near the visual center of the door or gate.

A valid capture candidate should be highlighted.

World Lock Icon Appears Too Far Away

Reduce:

```
Config.DoorStateIconDistance = 6.0
```

Example:

```
Config.DoorStateIconDistance = 4.0
```

Duplicate World Lock Icon

Verify:

```
Config.DoorStateIconMergeDistance = 0.35
```

Then run:

```
doors_dedupe
```

ox_lib `lib` Is Nil

Nord Doorlock only uses ox_lib functions when the corresponding global exists.

If you explicitly configure an ox provider, make sure your ox_lib setup exposes the required functions before Nord Doorlock starts.

Otherwise use:

```
Config.Notify = 'auto'
Config.TextUI = 'auto'
```

Lockpick Always Works on a Standalone Custom Inventory

Integrate your inventory in:

```
server/bridge/inventory.lua
```

The default standalone inventory fallback cannot know whether the player really owns an item.

MySQL Storage Fails

Verify:

```
ensure oxmysql
ensure nord_doorlock
```

and:

```
Config.Storage = 'mysql'
```

Run:

```
doors_storage
```

Check server console errors from:

```
[nord_doorlock:storage]
```

Door State Does Not Update in Another Script

Read the state from the server:

```
local locked =
    exports.nord_doorlock:GetDoorState(id)
```

or request the synchronized client value on the client.

Do not cache the state forever in another resource unless you also handle state changes.

FAQ

Does Nord Doorlock require ox_doorlock?

No.

Does it require a database?

No. Default persistence uses FiveM KVP with JSON mirrors.

Can I use oxmysql?

Yes.

```
Config.Storage = 'mysql'
```

Can one entry contain two physical doors?

Yes. Capture Door 1 and Door 2 in the wizard.

Can I give access by job grade?

Yes.

Example:

```
police:2
```

Can I restrict access by Discord identifier or license?

Yes. Add identifiers in the access step.

Can another script lock or unlock a door?

Yes. Use server exports such as:

```
exports.nord_doorlock:LockDoor(id)
exports.nord_doorlock:UnlockDoor(id)
```

Can another script add a custom permission system?

Yes. Use the `doorAuthorization` hook.

Can I create doors programmatically?

Yes.

Use:

```
exports.nord_doorlock:CreateDoor(payload)
```

Can a client script directly force a door unlocked?

The public client API intentionally exposes request functions rather than an authoritative `SetDoorState`.

Use a server export for trusted state changes.

Why does Nord Doorlock reject my new door as a duplicate?

One of the physical door parts is already owned by another entry inside the configured duplicate tolerance.

Find the existing door or run:

```
doors_dedupe
```

How do I check whether Nord Doorlock is ready before calling exports?

```
local ready, err = exports.nord_doorlock:IsReady()

if not ready then
    print(err)
    return
```

```
end
```

Where should I add support for my custom notification/target/inventory?

Use the bridge files instead of editing the core.