Operations

Operations documentation for Nord Doorlock.

- Duplicate Door Protection
- Security Model
- Troubleshooting
- FAQ

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.

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.