# Administration

Administration documentation for Nord Doorlock.


- Admin Panel & Wizard
- Door Interaction & Visuals
- Permissions
- Lockpicking

# 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.