Developer API

Developer API documentation for Nord Doorlock.

- Server Exports
- Client Exports
- Authorization Hooks
- Door Data Structure
- Integration Examples

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
```

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)
```

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.