1 · Setup & Auto Installation

Install Nord Crafting, configure its dependencies and understand the Nord Inventory-only automatic installation workflow.

Requirements & Installation

Required resources

Nord Crafting declares these dependencies:

```
dependencies {
    'ox_lib',
    'oxmysql'
}
```

Install the resource at:

```
resources/[nord]/nord_crafting
```

Add it to `server.cfg` after its dependencies and framework/inventory resources.

```
ensure oxmysql
ensure ox_lib
ensure nord_inventory
ensure nord_crafting
```

If you use QBCore or ESX, start the framework before Crafting so `Config.Framework = 'auto'` can detect it.

First boot

With the default settings:

```
Config.AutoCreateTables = true
Config.AutoMigration = true
```

Nord Crafting waits for `oxmysql`, creates the schema when needed, runs migrations and marks the database ready before the crafting system boots.

Version source

Use `fxmanifest.lua` and `version.json` as the release identity for this package. Both report **v6.2.0**.

Automatic Installation · Nord Inventory Only

Nord Crafting now includes an automatic installation path designed specifically for `nord_inventory`.

What this means

When your server uses Nord Inventory, the Crafting setup can use the automatic installer instead of asking the server owner to manually edit third-party inventory item files.

This is the recommended Nord Lab setup:

```
ensure nord_inventory
ensure nord_crafting
```

Scope

The automatic installer is **not a universal inventory patcher**.

It does **not** automatically install Crafting items into:

- `ox_inventory`
- `qs-inventory`
- `qb-inventory`
- ESX inventory item definitions

Those systems use the manual workflow described on the next page.

Verification after install

After first boot, verify the items required by your enabled modules exist in the active inventory registry.

The blueprint system requires:

```
blueprint
```

Portable benches use the configurable item:

```
Config.Portable.ItemName = 'craft_bench'
```

If you change `ItemName`, the inventory must contain the new name.

Why the limitation exists

Nord Inventory exposes a controlled Nord ecosystem integration path. Third-party inventories store and load item definitions differently, so Nord Crafting avoids rewriting their resource files automatically.

> **Support rule:** If the server is not using `nord_inventory`, document the inventory setup as manual. This avoids telling customers that OX/QS/QB files will be modified automatically when they will not be.

Manual Inventory Setup · OX, QS, QB & ESX

Use this workflow whenever the active inventory is **not** Nord Inventory.

Blueprint item

The package includes manual definitions under:

```
install/itens/item.lua
install/itens/item_qs.lua
install/itens/blueprint.png
```

OX-style definition

```
['blueprint'] = {
    label = 'Blueprint',
    weight = 50,
    stack = false,
    close = true,
    client = {
        image = 'blueprint.png'
    }
},
```

Copy the image into the image directory used by your inventory and place the definition in its item registry.

QS

Use `install/itens/item_qs.lua` as the base definition, then place `blueprint.png` in the configured QS image directory.

QB

Create a `blueprint` shared item using your QBCore/QB Inventory format. Keep it unique/non-stackable if you want each blueprint instance to preserve its own metadata and remaining uses.

ESX

Create the `blueprint` item through the item registry/database used by your ESX inventory setup.

Portable bench item

Portable benches default to:

```
Config.Portable.ItemName = 'craft_bench'
```

The package's `install/itens/` folder provides the blueprint definitions, so if you enable portable benches on a non-Nord inventory you should also create a compatible `craft_bench` item manually.

Restart

Restart the inventory resource after changing its item registry, then restart `nord_crafting`.

Framework Detection

```
Config.Framework = 'auto' -- auto | qb | esx
```

In automatic mode, Crafting checks:

1. `qb-core`
2. `es_extended`

The active framework is used for player data, citizen/identifier resolution, jobs and QBCore gangs.

QBCore

- Player identity: `citizenid` where required by Crafting.
- Job name and grade are supported.
- Gang name and grade are supported.

ESX

- Player identity: ESX player identifier.
- Job name and grade are supported.
- Gang access is not available through the built-in ESX bridge.

Force a framework

```
Config.Framework = 'qb'
```

or:

```
Config.Framework = 'esx'
```

Use a forced value when both framework resources are present but only one should be used.

Core Configuration

The primary configuration is `config/config.lua`.

Core providers

```
Config.Framework = 'auto'
Config.Locale = 'pt'
Config.Inventory = 'auto'
Config.Target = 'auto'
Config.Progress = 'auto'
```

Notifications

```
Config.Notify = {
    type = 'auto',
    title = 'Nord Crafting',
}
```

Crime provider

```
Config.CrimeBridge = 'auto'
```

Values:

- `auto`
- `nord_crime`
- `op-crime`
- `false`

Database

```
Config.Version = '6.2.0'
Config.AutoMigration = true
Config.AutoCreateTables = true
Config.DebugMigration = false
```

Debug/console

```
Config.DebugLogs = false

Config.TxConsole = {
    DisableUpdateLogs = true,
    DisableAllLogs = false
}
```

Discord logs

```
Config.DiscordLogs.Enabled = false
```

Separate webhook slots exist for Craft, Admin, Blueprint, Recipe and Stash logging.

Smart Database Initialization

Nord Crafting includes a database initializer and migration system.

First start

When automatic creation is enabled, the resource:

1. Waits for `oxmysql`.
2. Ensures the settings table exists.
3. Creates required Crafting tables.
4. Inserts default data where required.
5. Runs migrations.
6. Stores the schema version.
7. Writes local state to `server/db_state.json`.

Smart restart behavior

`server/db_state.json` stores the current schema version. When the local state and database are already current, Crafting can skip unnecessary heavy CREATE/ALTER work on normal restarts.

Manual SQL

A full SQL file remains available at:

```
install/sql.sql
```

Use it when your deployment policy requires manual schema management.

Readiness export

```
local ready = exports['nord_crafting']:IsNordDBReady()
```

This is useful for another server resource that must wait for Crafting's database layer before calling database-backed APIs.

Admin Access

Administrator access is based on FiveM license identifiers.

```
Config.AdminLicenses = {
    ['license:YOUR_LICENSE'] = true
}
```

The Admin Manager also supports runtime administrators, which can be added or removed without editing the base config.

Open Admin Studio

Default client command:

```
/craftbench
```

The server validates admin access before sending the Admin Studio payload.

Runtime admin exports

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

`value` can be a supported license/source form handled by the Admin Manager.

Runtime admins are persisted to:

```
server/data/runtime_admins.json
```

This API is useful for staff-management resources such as `nord_staff`.

Locales & Resource Logs

The package ships with:

```
locales/en.lua
locales/pt.lua
```

Select the locale:

```
Config.Locale = 'pt'
```

Runtime debug

```
Config.DebugLogs = false
```

Console/txAdmin can also toggle blueprint/crafting debug at runtime:

```
bpdebug on
bpdebug off
```

The command is console-only.

TX console policy

```
Config.TxConsole = {
    DisableUpdateLogs = true,
    DisableAllLogs = false
}
```

Use these options to keep production console output clean without changing the Crafting logic.