Nord Crafting v6.2.0

Official-style Nord Lab documentation for Nord Crafting v6.2.0, including Nord Inventory-only automatic installation, Admin Studio, blueprints, weapon benches, Nord Staff integration, bridges and developer API.

- Welcome · Nord Crafting v6.2.0
- Quick Start · Nord Inventory Recommended
- Compatibility Matrix
- 1 · Setup & Auto Installation
  - Requirements & Installation
  - Automatic Installation · Nord Inventory Only
  - Manual Inventory Setup · OX, QS, QB & ESX
  - Framework Detection
  - Core Configuration
  - Smart Database Initialization
  - Admin Access
  - Locales & Resource Logs
- 2 · Crafting Core
  - Crafting Flow
  - Recipes · Config + Database
  - Level & XP Progression
  - Level-Based Craft Failure
  - Stashes
  - Crafting Security Model
- 3 · Benches & Access
  - Bench Types
  - Fixed Benches
  - Portable Benches

    - Access Control · Public, Job & Gang
    - Organization Access · Nord Crime / OpCrimes
    - Bench Props & Placement

- 4 · Blueprints
    - Blueprint System Overview
    - Blueprint Item Installation
    - Drafting & Quality
    - Blueprint Metadata & Uses
    - Blueprint-Gated Recipes
    - Blueprint Developer API

- 5 · Weapon Workbench
    - Weapon Workbench Overview
    - Weapon Preview Shell
    - Weapon Component Recipes
    - Compatibility Validation
    - Weapon Craft Security

- 6 · Admin Studio
    - Admin Studio Overview
    - Recipe Manager
    - Bench Manager & Usage Analytics
    - Levels & Runtime Options
    - Prop Manager
    - Blueprint Manager
    - Update Checker & Security

- 7 · Integrations & Bridges
    - Nord Inventory Integration
    - Nord Staff Integration
    - Inventory Bridge
    - Target Bridge
    - Notifications, Text UI & Progress

    - Nord Crime & OpCrimes
    - Nord VIP Integration

- 8 · Developer API & Operations
    - Client Exports
    - Server Exports · Benches
    - Server Exports · Inventory
    - Server Exports · Blueprints & Organizations
    - Server Exports · Admin Runtime
    - Database Tables
    - Discord Logging
    - Troubleshooting
    - Production Checklist
    - Downloadable Reference Files

Welcome · Nord Crafting v6.2.0

Nord Crafting is Nord Lab's advanced crafting, workbench and blueprint system for FiveM. It combines fixed and portable benches, player progression, blueprint-based unlocks, weapon/component workbenches, access control, stashes, database-backed administration and modular bridges for the rest of the server ecosystem.

Highlights

- Fixed and portable crafting benches.
- Craft, blueprint, stash and weapon bench modes.
- In-game **Admin Studio** for benches, recipes, levels, props, blueprints and runtime options.
- Database-backed custom recipes and blueprints.
- Level/XP progression with per-recipe minimum levels.
- Optional level-based craft failure.
- Blueprint drafting, quality, metadata and limited uses.
- Weapon preview environment and component crafting workflow.
- Public, job, gang and organization access rules.
- `nord_crime` and `op-crime` organization bridges.
- Optional `nord_vip` progression rewards.
- Discord logging and bench usage analytics.
- Automatic schema creation and migration through `oxmysql`.
- `nord_staff` Admin Studio integration.
- **Nord Inventory-only automatic installation path** for required inventory setup.

Release identity

The package declares:

```
Resource: nord_crafting
Version: 6.2.0
Build: 610
```

```
Channel: stable
```

Keep the resource folder named exactly:

```
nord_crafting
```

> **Important:** Automatic inventory installation in this release is intended **only for** `nord_inventory`. Other inventory systems remain supported through the Crafting bridge, but their item/image setup is manual.

Quick Start · Nord Inventory Recommended

This is the cleanest setup when you use the Nord ecosystem.

1. Start the required resources

```
ensure oxmysql
ensure ox_lib

# Framework when used
ensure qb-core
# or: ensure es_extended

ensure nord_inventory
ensure nord_crafting

# Optional Nord integration
ensure nord_staff
```

2. Keep automatic database setup enabled

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

Nord Crafting creates and migrates its own database schema on first boot.

3. Use the Nord Inventory auto-installer

When `nord_inventory` is the inventory used by the server, use the built-in automatic installation workflow. It is the supported no-manual-copy path for Nord Crafting inventory requirements.

After the first boot, verify that the Crafting system items required by the modules you use are registered in Nord Inventory. In particular:

- `blueprint` for the blueprint system.
- The configured portable bench item when portable benches are enabled; default: `craft_bench`.

> **Auto-install limitation:** Do not expect the auto-installer to modify OX, QS, QB or ESX inventory files. Those inventories require manual setup.

4. Configure admin access

Add your FiveM license to `Config.AdminLicenses`:

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

Then use:

```
/craftbench
```

5. Create the first bench

Open Admin Studio, create or import recipes, configure levels if required, then create a bench and assign the recipes to it.

6. Production check

Before opening the server to players, test:

1. Crafting one normal recipe.
2. Missing ingredient rejection.
3. Job/gang/org access where used.
4. Blueprint creation and consumption.
5. Stash opening.
6. Level XP and level-locked recipes.
7. Weapon workbench if enabled.

Compatibility Matrix

Core dependencies

System	Status
FiveM / OneSync	Required
`oxmysql`	Required
`ox_lib`	Required
QBCore	Supported
ESX	Supported

Inventory

Inventory	Crafting bridge	Automatic installation
`nord_inventory`	Recommended Nord ecosystem path	**Yes**
`ox_inventory`	Supported	No — manual
`qs-inventory` / `qs_inventory`	Supported	No — manual
`qb-inventory`	Supported	No — manual
ESX inventory	Limited/basic bridge	No — manual

The important distinction is that **runtime compatibility and automatic installation are not the same feature**. The auto-installer is intentionally Nord Inventory-only.

Target

- `ox_target`
- `qb-target`
- `qtarget`
- automatic detection

Notifications / Text UI / Progress

- `nord_textui`
- CodeM Supreme Notification
- `ox_lib`
- QBCore notifications
- ESX notifications
- `qb-progressbar`
- built-in/fallback progress behavior

Nord ecosystem

- `nord_staff` — Admin Studio integration.
- `nord_crime` — organization access integration.
- `nord_vip` — optional crafting mission/pass XP integration.

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.

2 · Crafting Core

Understand recipes, progression, failure behavior, stashes and the server-authoritative crafting flow.

Crafting Flow

A standard craft is validated server-side.

Flow

1. Player opens a bench.
2. Server resolves the bench instance and access policy.
3. The selected recipe must be assigned to that bench.
4. Distance is validated.
5. Required level is validated when enabled.
6. Required blueprint is validated when enabled.
7. Ingredient counts are checked.
8. Ingredients are removed.
9. Optional level-based failure is rolled.
10. Output item is awarded on success.
11. Blueprint uses are reduced when applicable.
12. XP, logs, analytics and optional Nord VIP progress are updated.

This design prevents the NUI from deciding whether a player should receive an item.

Distance

Player distance is checked against the bench coordinates, with the configured craft distance used as the baseline.

```
Config.UI.CraftDistance = 2.0
```

Recipes · Config + Database

Recipes can come from two sources.

Default recipes

Defined in:

```
config/config_recipes.lua
```

Example:

```
lockpick = {
    label = 'Lockpick',
    item = 'lockpick',
    amount = 1,
    time = 6000,
    use_blueprint = 0,
    blueprint_type = '',
    ingredients = {
        iron = 10,
        plastic = 5
    }
}
```

Custom recipes

Custom Admin Studio recipes are stored in:

```
nord_crafting_recipes
```

Custom recipes can be created, updated, deleted and imported from the Admin Studio.

Common fields

Field	Purpose
`id` / recipe key	Stable recipe identifier
`label`	Display name
`item`	Output item
`amount`	Output quantity
`time`	Craft time in milliseconds
`ingredients`	Item → required amount map
`recipe_type`	General/weapon/component classification
`use_blueprint`	Require blueprint
`blueprint_type`	Required blueprint ID
`use_level`	Enable level gate
`min_level`	Minimum crafting level

Level & XP Progression

```
Config.LevelSystem = {
    Enabled = true,
    DefaultCraftXp = 25,
    DefaultLevels = {
        { level = 1, xp_required = 0,    label = 'Iniciante' },
        { level = 2, xp_required = 150,  label = 'Aprendiz' },
        { level = 3, xp_required = 450,  label = 'Experiente' },
        { level = 4, xp_required = 900,  label = 'Especialista' },
        { level = 5, xp_required = 1600, label = 'Mestre' }
    }
}
```

Persistence

Player progression is stored in:

```
nord_crafting_player_levels
```

The identifier uses the framework citizen/identifier when available, with license fallback where the Crafting logic requires it.

Per-recipe gates

A recipe can require a minimum level. If the player's current level is lower than `min_level`, the server rejects the craft before removing ingredients.

Runtime management

Admin Studio can create, update and remove level tiers. Level 1 is protected as the base level.

Runtime options can also enable/disable the level system and change default XP per craft.

Level-Based Craft Failure

Nord Crafting can apply a failure chance based on the player's crafting level.

```
Config.CraftFailure = {
    EnabledByDefault = true,
    MinChance = 0.02,
    MaxChance = 0.20
}
```

Lower levels receive the higher side of the configured failure range, while higher levels approach the minimum chance.

Important behavior

For standard crafting, ingredient removal happens **before** the failure roll. Therefore:

- Success → output item is added.
- Failure → no output is added and the consumed materials remain consumed.

This makes failed crafting an actual material risk rather than a free retry.

Admin Studio

The Options panel can control:

- Level system enabled/disabled.
- Craft failure enabled/disabled.
- Minimum failure percentage.
- Maximum failure percentage.
- Default craft XP.

These runtime settings are stored in `nord_crafting_settings`.

Stashes

Benches can include a storage area.

Typical fixed-bench defaults:

```
Config.AdminFixedDefault = {
    stash = {
        enabled = false,
        slots = 30,
        weight = 40000
    }
}
```

Portable default:

```
stash = {
    enabled = true,
    slots = 20,
    weight = 20000
}
```

Inventory behavior

The inventory bridge provides:

```
Inventory.RegisterStash(...)
Inventory.OpenStash(...)
```

OX uses registered stash support directly. Other supported inventory bridges use the Crafting client stash bridge where implemented.

The server checks bench existence/access before opening its storage.

Discord stash logging can be enabled through the dedicated Stash webhook.

Crafting Security Model

Nord Crafting keeps the important decisions server-side.

Validations

The server checks:

- Valid request payload.
- Existing bench instance.
- Player distance from bench.
- Bench access policy.
- Recipe assignment to the bench.
- Recipe existence.
- Required crafting level.
- Required blueprint and remaining uses.
- Ingredient counts.
- Weapon/component compatibility in weapon workflows.

Server authority

Inventory removal and output addition are performed through the server inventory bridge. The client/NUI is used for presentation and input, not for authoritative item grants.

Anti-race/craft locking

Crafting maintains per-player craft lock state to reduce spam and overlapping craft operations.

Recommended production rule

Do not expose custom events that bypass `GetBenchOpenPayload`, access validation or the normal crafting handlers. If another script needs to open a bench, use the public exports documented in

the Developer API chapter.

3 · Benches & Access

Create fixed and portable benches, control access, manage props and integrate organization ownership.

Bench Types

The runtime normalizes benches into four main modes:

Type	Purpose
`craft`	Standard item crafting
`blueprint`	Blueprint drafting/creation
`stash`	Storage-focused bench
`weapon`	Weapon preview/component workflow

Older labels such as `general`, `illegal` or `military` can still be used as recipe organization concepts, but the current bench runtime normalizes the operational mode to the types above.

The bench type controls which interface and server validation path is used.

Fixed Benches

Fixed benches are persistent world benches stored in:

```
nord_crafting_fixed
```

They contain:

- Label.
- Prop/model.
- Coordinates and heading.
- Bench type.
- Access JSON.
- Assigned recipe/blueprint IDs.
- Stash configuration.
- Creator and creation timestamp.

Admin Studio provides placement mode for creating fixed benches in-world.

After changes, the bench cache is reloaded and synchronized to clients.

Portable Benches

Portable benches are enabled in:

```
Config.Portable = {
    Enabled = true,
    ItemName = 'craft_bench',
    Default = {
        label = 'Bancada Portátil',
        model = `prop_tool_bench02`,
        access = { type = 'public' },
        recipes = { ... },
        stash = { enabled = true, slots = 20, weight = 20000 },
        expiresMinutes = nil
    }
}
```

Placed portable benches are stored in:

```
nord_crafting_placed
```

Inventory requirement

If portable benches are enabled, the active inventory must contain the item named by `Config.Portable.ItemName`.

- With Nord Inventory, use the supported auto-install workflow and verify the item exists after setup.
- With other inventories, create/register the item manually.

Expiration

`expiresMinutes` can be used to create temporary placed benches. `nil` means no configured expiration from this default.

Access Control · Public, Job & Gang

Public

```
access = { type = 'public' }
```

Any player can use the bench, subject to normal distance and recipe validation.

Job

```
access = {
    type = 'job',
    jobs = { 'police', 'mechanic' },
    minGrade = 2
}
```

Supported on QBCore and ESX.

Gang

```
access = {
    type = 'gang',
    gangs = { 'ballas', 'lostmc' },
    minGrade = 1
}
```

Gang access is built into the QBCore bridge.

Normalization

The access resolver accepts common forms such as `list`, plural keys (`jobs`, `gangs`), singular keys, names and comma/space-separated strings.

Organization Access · Nord Crime / OpCrimes

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

In automatic mode, Crafting prefers `nord_crime` when available and can fall back to `op-crime`.

Organization access

```
access = {
    type = 'org',
    orgId = 12
}
```

The player's organization state is checked server-side.

Nord Crime-specific modes

The bridge supports Nord Crime organization access patterns including:

- Direct organization ID.
- `zone_owner` mode.
- `org_type` mode.

Bench creation exports

Nord Crime or another compatible server resource can create organization benches using:

```
exports['nord_crafting']:CreateNordCrimeOrgBench(orgId, data)
exports['nord_crafting']:CreateOrgBench(orgId, data)
```

```
exports['nord_crafting']:AddOrgCraftingBench(orgId, data)
```

These aliases call the same organization bench creation implementation.

Bench Props & Placement

Nord Crafting loads bench prop catalogs from:

```
models/default_props.json
models/custom_props.json
```

The Admin Studio can also manage custom prop entries stored in:

```
nord_crafting_props
```

Placement

Admin placement spawns the selected model in-world and stores coordinates/heading when confirmed.

Target interaction

Spawned bench entities are registered through the target bridge when a supported target is active.

Restart behavior

On Crafting restart, the client clears spawned bench entities/cache and requests a fresh synchronization from the server, reducing stale world props after resource restarts.

4 · Blueprints

Install blueprint items, create technical drawings, manage quality/uses and gate recipes through metadata.

Blueprint System Overview

Blueprints are metadata-rich inventory items used to unlock or authorize recipes.

A blueprint definition is stored in:

```
nord_crafting_blueprints
```

Each definition can include:

- Stable blueprint ID.
- Label.
- Item/material requirements for drafting.
- Maximum uses.
- Image or image URL.

The physical inventory item is named:

```
blueprint
```

Each crafted blueprint instance receives its own metadata, allowing different quality and remaining-use values.

Blueprint Item Installation

The blueprint system requires the inventory item:

```
blueprint
```

Nord Inventory

Use Nord Crafting's **automatic installation workflow**. The auto-installer is intended only for `nord_inventory`.

After first boot, verify `blueprint` is present in the Nord Inventory item registry before testing drafting.

Other inventories

Manual files are included:

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

If the item is missing, the blueprint startup check reports that `blueprint` was not found in the active inventory and points to the manual install files.

> For metadata-based uses/quality to work correctly, use a unique/non-stacking blueprint definition where your inventory requires that behavior.

Drafting & Quality

Blueprint benches can open the drafting interface. The client sends the detected blueprint type and drawing quality to the server.

Server validation

The server:

1. Normalizes the detected blueprint ID.
2. Confirms the definition exists in the database.
3. Clamps quality to **1–100**.
4. Reads drafting ingredients from the database.
5. Verifies and removes materials.
6. Creates the `blueprint` item with metadata.
7. Refunds materials when adding the final blueprint item fails.

Quality metadata

The blueprint item label includes its quality percentage, for example:

```
Pistol Blueprint (85%)
```

Quality is stored in:

```
metadata.quality
```

Blueprint Metadata & Uses

A generated blueprint contains metadata similar to:

```
{
    label = 'Pistol Blueprint (85%)',
    description = 'Projeto técnico avançado\nUtilizações: 3/3',
    blueprint = true,
    blueprint_id = 'pistol',
    quality = 85,
    uses = 3,
    max_uses = 3,
    ingredients = { ... },
    createdAt = os.time()
}
```

An optional local image is stored as `metadata.image`; HTTP/HTTPS images are stored as `metadata.imageurl`.

Use consumption

When a blueprint-required recipe succeeds:

- `uses` decreases by 1.
- The description is updated.
- When uses reach zero, the blueprint item is removed.

If the inventory cannot update metadata directly, Crafting can fall back to removing/re-adding the item with updated metadata where the bridge allows it.

Blueprint-Gated Recipes

A recipe can require a blueprint type.

Database-backed recipes use:

```
use_blueprint
blueprint_type
```

The server searches the player's inventory for an item named `blueprint` whose metadata contains the matching `blueprint_id`.

Example concept:

```
use_blueprint = 1
blueprint_type = 'pistol'
```

The recipe is rejected if:

- The blueprint type is empty.
- No matching blueprint exists.
- The matching blueprint has no remaining uses.

Blueprint validation occurs before ingredient removal.

Blueprint Developer API

Server exports

```
exports['nord_crafting']:GetPlayerBlueprintRecipes(source)
exports['nord_crafting']:GetBlueprintDefinition('pistol')
exports['nord_crafting']:BuildBlueprintMetadata(blueprintRow, 85)
exports['nord_crafting']:CreateBlueprintItem(source, 'pistol', 85)
```

Custom uses and metadata can be supplied:

```
local ok, metadata = exports['nord_crafting']:CreateBlueprintItem(
    source,
    'pistol',
    85,
    3,
    {
        recipes = { 'weapon_pistol' }
    }
)
```

`GetPlayerBlueprintRecipes` reads `metadata.recipes` from blueprint items and returns the unlocked recipe set.

5 · Weapon Workbench

Configure the 3D preview, component recipes, compatibility rules and server-side weapon crafting validations.

Weapon Workbench Overview

A bench with type:

```
weapon
```

uses the dedicated weapon crafting interface and validation path.

The workflow can:

- Inspect weapon items in the player's inventory.
- Preview a weapon in a configured 3D location.
- Preview/attach/detach selected components in the interface.
- Validate component recipes assigned to the bench.
- Validate player level.
- Validate ingredients.
- Process multiple selected component recipes as one selection.

The server rejects weapon-component requests sent to a non-weapon bench.

Weapon Preview Shell

The preview environment is configured in `config/config.lua`.

```
Config.WeaponPreviewShell = {
    coords = vector3(1048.34, -3107.5, -39.0),
    heading = 0.0,
    mode = 'wall', -- wall | center
    wallOffset = 0.06,
    heightOffset = 1.10,
    cameraDistance = 2.65,
    cameraHeight = 0.08,
    spin = false
}
```

Important

Nord Crafting does **not** create the preview shell/interior. The configured coordinates must already be a suitable location on your server.

Wall mode

Place the coordinates at the desired wall position. `heading` should point from the wall toward the camera. If the weapon faces the wrong direction, adjust the heading by 180 degrees.

Center mode

Use a free central preview point where the weapon can be displayed without a wall offset.

Weapon Component Recipes

Component recipes use the same database/config recipe model but are classified as component recipes.

The weapon callback validates:

- Bench type is `weapon`.
- Recipe is assigned to that bench.
- Recipe exists.
- Recipe type resolves to `component`.
- Player meets required level.
- Player has required ingredients.

For multi-selection, ingredient totals are combined before removal so the full selected set must be affordable.

Compatibility Validation

The weapon system groups weapons into families such as:

- pistol
- smg
- rifle
- shotgun
- sniper
- other

Component compatibility is validated against the weapon/component profile data used by the workbench.

When a selected component is not compatible with the target weapon, the server rejects the selection before consuming the combined materials.

This prevents a client from forcing an arbitrary component recipe onto an incompatible weapon through NUI manipulation.

Weapon Craft Security

Weapon callbacks perform the same trust-boundary checks expected from standard crafting, plus weapon-specific checks.

Server checks

- Valid request structure.
- Existing bench ID.
- Player distance.
- Access policy.
- Weapon bench type.
- Allowed recipe list.
- Recipe type.
- Weapon item ownership when required by the selected workflow.
- Component compatibility.
- Level requirements.
- Total ingredient requirements.

XP is applied after a valid weapon/component craft, and bench usage analytics are updated.

6 · Admin Studio

Operate the complete in-game management interface for recipes, benches, analytics, levels, props, blueprints and updates.

Admin Studio Overview

Open the Crafting Admin Studio with:

```
/craftbench
```

or through the `nord_staff` integration.

The current interface covers:

- Dashboard/system overview.
- Recipes.
- Benches.
- Bench Manager and usage analytics.
- Props.
- Levels.
- Blueprints.
- Runtime Options.
- Update status.

Admin actions are validated server-side through the Crafting admin manager.

Recipe Manager

The recipe manager can:

- Create custom recipes.
- Edit database-backed custom recipes.
- Delete custom recipes.
- Import recipe data.
- Search and filter recipe sources.
- Configure output item/amount/time.
- Configure ingredients.
- Enable blueprint requirements.
- Enable level requirements.
- Configure minimum level.

Default config recipes are presented as defaults, while database recipes are treated as custom entries.

The server protects the edit/delete path so Admin Studio does not silently rewrite the static Lua config recipe file.

Bench Manager & Usage Analytics

Admin Studio can request an overview of registered benches and usage data.

The interface supports:

- Search by bench/model/ID.
- Edit bench configuration.
- Teleport an admin to a bench location.
- Copy/inspect coordinates.
- Review top benches by usage.
- Review usage series/history where data is available.

Craft and stash activity is tracked by bench ID so administrators can understand which benches are actually being used.

Levels & Runtime Options

The Levels tab manages progression tiers stored in `nord_crafting_levels`.

The Options panel persists global runtime values in `nord_crafting_settings`.

Current runtime options

- Level System.
- Base XP per Craft.
- Level-Based Failure.
- Minimum failure percentage.
- Maximum failure percentage.
- Debug Logs.
- Disable updater logs.
- Disable all resource logs.
- OpCrimes toggle.
- Discord Logs.
- Nord VIP integration.

The server applies runtime options back to the active `Config` table, so the change can affect live behavior without rewriting the original configuration file.

Prop Manager

The Prop Manager combines default and custom bench prop catalogs.

Custom prop records are stored in:

```
nord_crafting_props
```

Admin actions include:

- Create prop entry.
- Update prop entry.
- Delete prop entry.
- Load merged prop list into the bench creation UI.

Static prop catalogs are also shipped in:

```
models/default_props.json
models/custom_props.json
```

Blueprint Manager

Admin Studio can create and delete blueprint definitions stored in the database.

A blueprint definition can include:

- ID.
- Label.
- Drafting materials.
- Output/reference item fields.
- Maximum uses.
- Image.

The blueprint list is synchronized to the NUI and is also used by blueprint bench drafting flows.

Before publishing a new blueprint-gated recipe, verify:

1. Blueprint definition exists.
2. `blueprint` inventory item exists.
3. Draft materials are valid items.
4. Required recipe `blueprint_type` matches the definition ID exactly after normalization.

Update Checker & Security

Nord Crafting contains a GitHub release checker for private/public repository workflows.

Console command:

```
nordcrafting:update
```

The Admin Studio can also request update status.

Recommended token handling

The updater supports providing the GitHub token through a server convar:

```
set nord_crafting_github_token "YOUR_TOKEN"
```

Do not publish a live GitHub access token inside documentation, screenshots, public ZIPs or shared config examples.

If a token has previously been distributed in plaintext, rotate/revoke it in GitHub and move the replacement to a private server-side secret/convar.

Download behavior

The updater can download the latest release ZIP into the resource directory, but the current flow still expects the server owner to extract/apply it and restart the resource/server manually.

7 · Integrations & Bridges

Connect Nord Crafting to Nord Inventory, Nord Staff, targets, notifications, crime organizations and Nord VIP.

Nord Inventory Integration

Nord Inventory is the recommended inventory for the Nord Crafting ecosystem because it is the target of the new **automatic installation workflow**.

Recommended startup

```
ensure nord_inventory
ensure nord_crafting
```

Automatic installation

Use the Nord Inventory auto-installer for Crafting's required inventory setup. This is the only inventory integration for which the documentation should promise automatic installation.

Other inventories

Crafting includes inventory bridge logic for OX/QS/QB/ESX workflows, but those integrations require manual item/image setup.

Support wording

A useful rule for customer support is:

> Nord Inventory: automatic installation supported. Third-party inventory: runtime bridge supported where listed, item installation is manual.

Nord Staff Integration

Nord Crafting exposes its Admin Studio to `nord_staff`.

The manifest declares:

```
nord_staff_admin_export 'OpenAdminInterface'
```

and the client provides:

```
exports('OpenAdminInterface', function(options)
    if type(options) == 'table' and options.probe == true then
        return true
    end

    TriggerServerEvent('nord_crafting:server:adminOpenUI')
    return true
end)
```

What the integration provides

`nord_staff` can discover/probe the Crafting admin interface and open the same Admin Studio used by `/craftbench`.

This means staff do not need a separate duplicated Crafting admin UI.

Recommended startup order

```
ensure nord_crafting
ensure nord_staff
```

When Nord Inventory is also used:

```
ensure nord_inventory
ensure nord_crafting
ensure nord_staff
```

Dependency model

`nord_staff` is **optional**. Nord Crafting remains usable without it through its own admin command and license permission system.

Inventory Bridge

Crafting centralizes inventory operations in `server/bridge/inventory.lua`.

Public operations include:

- Get item count.
- Add item.
- Remove item.
- Search slots/items.
- Read inventory.
- Read item registry.
- Set metadata where supported.
- Register/open stashes where supported.

Active bridge types in the package

- `ox`
- `qs`
- `qb`
- `esx`

The exact feature depth varies by inventory. OX has the most direct stash and metadata support in the current generic bridge.

> Automatic installation remains a separate Nord Inventory-only feature; do not confuse it with these runtime bridge operations.

Target Bridge

```
Config.Target = 'auto'
```

Automatic detection order:

1. `ox_target`
2. `qb-target`
3. `qtarget`
4. no target provider

The bridge exposes internal helpers to add/remove local entity targets and a safe-add routine that waits briefly for the entity to exist.

Interaction distance uses:

```
Config.UI.CraftDistance
```

when the target provider requires a distance value.

Notifications, Text UI & Progress

Notification auto-detection

When `Config.Notify.type = 'auto'`, the current bridge checks:

1. `nord_textui`
2. `codem-supreme-notification`
3. `ox_lib`
4. QBCore
5. ESX
6. console print fallback

Text UI

Text UI prefers:

1. `nord_textui`
2. `ox_lib`

Progress

Craft progress supports:

- Nord Text UI progress.
- `ox_lib` progress circle.
- `qb-progressbar`.
- Fallback timer that temporarily freezes the player.

This bridge design allows the core crafting logic to stay independent from the visual notification/progress provider.

Nord Crime & OpCrimes

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

Automatic provider choice

The intended automatic behavior is to use `nord_crime` when started and fall back to `op-crime` when appropriate.

Organization-aware benches read organization state from the player and validate access server-side.

Nord Crime integration also provides exports that can create fixed organization benches directly from another resource.

Set:

```
Config.CrimeBridge = false
```

to disable organization access integration entirely.

Nord VIP Integration

Nord VIP support is optional.

```
Config.NordVIP = {
    Enabled = false,
    MissionKey = 'crafts',
    MissionProgressPerCraft = 1,
    PassXpPerCraft = 50,
}
```

When enabled and `nord_vip` is started, a successful craft can call:

```
exports['nord_vip']:AddPassXpAndMission(
    source,
    passXp,
    missionKey,
    missionProgress
)
```

If the resource is not running, Crafting skips the reward and can log the issue when debug is enabled.

8 · Developer API & Operations

Use public exports safely, understand the database, diagnose common issues and prepare the resource for production.

Client Exports

Admin Studio

```
exports['nord_crafting']:OpenAdminInterface()
```

Probe support used by integrations:

```
local supported = exports['nord_crafting']:OpenAdminInterface({ probe = true })
```

Bench UI

```
exports['nord_crafting']:OpenBench('fixed_1')
exports['nord_crafting']:CloseBench()

local open, current = exports['nord_crafting']:IsBenchOpen()
local bench = exports['nord_crafting']:GetBench('fixed_1')
local cache = exports['nord_crafting']:GetBenchCache()
```

Use these exports instead of manually sending internal NUI messages from another client resource.

Server Exports · Benches

```
exports['nord_crafting']:GetFixedBenches()
exports['nord_crafting']:GetPlacedBenches()
exports['nord_crafting']:GetAllBenches()
exports['nord_crafting']:GetBench('fixed_1')
exports['nord_crafting']:ReloadBenches()
```

Add a placed bench:

```
exports['nord_crafting']:AddPlacedBench(ownerCid, payload, def)
```

Access/opening helpers:

```
local payload, err = exports['nord_crafting']:GetBenchOpenPayload(source, 'fixed_1')
local allowed, reason = exports['nord_crafting']:CanPlayerAccessBench(source, 'fixed_1')
local ok, reason = exports['nord_crafting']:OpenBenchForPlayer(source, 'fixed_1')
```

`OpenBenchForPlayer` performs the normal payload/access path and then opens the bench on the target client.

Server Exports · Inventory

These exports expose Crafting's active inventory bridge to other server resources.

```
local invType = exports['nord_crafting']:GetInventoryType()
local count = exports['nord_crafting']:GetItemCount(source, 'iron')
local ok = exports['nord_crafting']:AddItem(source, 'iron', 5, { quality = 100 })
local ok = exports['nord_crafting']:RemoveItem(source, 'iron', 2)
local slots = exports['nord_crafting']:SearchInventory(source, 'slots', 'blueprint')
local inventory = exports['nord_crafting']:GetInventory(source)
local items = exports['nord_crafting']:GetItemList()
```

The bridge normalizes basic operations across supported inventory providers, but metadata/stash capabilities can vary by provider.

Server Exports · Blueprints & Organizations

Blueprint API

```
exports['nord_crafting']:GetPlayerBlueprintRecipes(source)
exports['nord_crafting']:GetBlueprintDefinition('pistol')
exports['nord_crafting']:BuildBlueprintMetadata(row, 85)
exports['nord_crafting']:CreateBlueprintItem(source, 'pistol', 85)
```

Organization bench API

```
exports['nord_crafting']:CreateNordCrimeOrgBench(orgId, data)
exports['nord_crafting']:CreateOrgBench(orgId, data)
exports['nord_crafting']:AddOrgCraftingBench(orgId, data)
```

Database readiness

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

Server Exports · Admin Runtime

```
local ok, identifier = exports['nord_crafting']:AddAdmin(value)
local ok, identifier = exports['nord_crafting']:RemoveAdmin(value)
local isAdmin = exports['nord_crafting']:IsAdmin(value)
local runtimeAdmins = exports['nord_crafting']:GetRuntimeAdmins()
```

These are useful for a central staff resource that manages access dynamically.

`nord_staff` additionally integrates through the manifest-declared client Admin Studio export, so the two systems can share a single Crafting administration surface.

Database Tables

Nord Crafting creates/manages the following tables in this package:

Table	Purpose
`nord_crafting_settings`	Schema version and runtime options
`nord_crafting_meta`	Crafting metadata/version compatibility
`nord_crafting_blueprints`	Blueprint definitions
`nord_crafting_fixed`	Fixed benches
`nord_crafting_placed`	Portable/placed benches
`nord_crafting_logs`	Crafting log records
`nord_crafting_props`	Custom prop catalog
`nord_crafting_recipes`	Custom recipes
`nord_crafting_levels`	Level tiers
`nord_crafting_player_levels`	Player XP/level
`nord_craft_benches`	Compatibility/legacy bench schema
`nord_craft_recipes`	Compatibility/legacy recipe schema

The automatic initializer uses the configured Crafting version as the database schema version.

Discord Logging

```
Config.DiscordLogs = {
    Enabled = false,
    Webhooks = {
        Craft = '',
        Admin = '',
        Blueprint = '',
        Recipe = '',
        Stash = ''
    }
}
```

Enable only the destinations you actually use and keep webhook URLs private.

Typical logged actions include:

- Successful/failed blueprint operations.
- Admin actions.
- Recipe management.
- Crafting activity.
- Stash access.

Use separate webhooks when you want different Discord channels for player activity and administrative changes.

Troubleshooting

`blueprint` item not found

Nord Inventory: run/verify the Nord Inventory-only automatic installation flow, then confirm the item exists.

Other inventories: install the item definition/image manually from `install/itens/`.

Portable bench cannot be used

Verify the active inventory contains the item configured by:

```
Config.Portable.ItemName
```

Default: `craft_bench`.

No inventory detected

Check `Config.Inventory` and make sure the inventory resource starts before Crafting. For the generic bridge, valid configured modes include OX, QS, QB and ESX.

Bench appears but cannot be opened

Check:

- Player distance.
- Access type and minimum grade.

- Job/gang/org state.
- Bench cache synchronization.
- Assigned recipes.

Craft recipe missing from UI

Confirm the recipe ID is assigned to the selected bench and, when blueprint gating is enabled, that the player has the matching blueprint.

Database does not initialize

Verify `oxmysql` is started, database credentials are valid and the DB user can create/alter tables when automatic migration is enabled.

Admin Studio denied

Add the correct FiveM `license:` identifier to `Config.AdminLicenses` or use the runtime Admin Manager API.

Weapon preview is empty/wrong position

The resource does not create the shell. Correct `Config.WeaponPreviewShell.coords`, `heading`, mode and offsets.

Production Checklist

Before release:

- [ ] `oxmysql` and `ox_lib` start before Crafting.
- [ ] Framework auto-detection reports the expected framework.
- [ ] Inventory integration is tested.
- [ ] **Nord Inventory auto-installer** is verified when using `nord_inventory`.
- [ ] Third-party inventory item setup is completed manually when applicable.
- [ ] `blueprint` item exists.
- [ ] `craft_bench` (or renamed portable item) exists when portable benches are enabled.
- [ ] Admin licenses are correct.
- [ ] Database tables initialize without errors.
- [ ] Standard recipe craft tested.
- [ ] Failed craft behavior is accepted/balanced.
- [ ] Blueprint drafting and use consumption tested.
- [ ] Level progression tested.
- [ ] Job/gang/org bench permissions tested.
- [ ] Bench stash tested.
- [ ] Weapon bench tested if used.
- [ ] Discord webhook secrets are private.
- [ ] GitHub updater token is stored privately and not shipped publicly.
- [ ] `nord_staff` can open Crafting Admin Studio when the integration is enabled.
- [ ] Debug logging is disabled or intentionally configured for production.

Downloadable Reference Files

This page contains the key editable/reference files from the supplied **Nord Crafting v6.2.0** package, plus sanitized setup notes generated for this documentation.

Use the live resource files as the source of truth when you change configuration.

> The private GitHub updater configuration is intentionally **not attached** here because secret tokens should not be redistributed inside documentation exports.