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.