Developer Ecosystem SDK

Developer Ecosystem SDK

Provider-aware runtime items, actions, schemas, images, hooks and health reporting.

Applies to: `Nord Inventory v2.0.0`

Developer Ecosystem

Nord Inventory exposes a provider-aware server SDK so another resource can own its items and behavior without editing `shared/items.lua` or the Nord core.

Core rules

- Runtime registrations are transient and should be repeated when the provider resource starts.
- The calling FiveM resource is detected with `GetInvokingResource()`; integrations do not need to hard-code their own provider name for bare export handlers.
- A runtime definition cannot replace a database-authoritative item. `RegisterItem` / `RegisterIntegration` return `database_authoritative` for that definition.
- A provider may still attach an external use handler to an already-authoritative item.
- Runtime ownership, use handlers, item actions, metadata schemas and hooks are cleaned up by provider ownership when the resource stops or calls `UnregisterIntegration()`.
- One provider cannot silently steal a runtime item, external use handler, metadata schema or an action ID owned by another provider.

Capability detection

```
if not exports.nord_inventory:HasFeature('integration_manifest') then
    return
```

```
end


local capabilities = exports.nord_inventory:GetCapabilities()
```

Useful capability names include `runtime_items`, `external_use`, `provider_registry`, `integration_manifest`, `item_actions`, `metadata_schema_api`, `durability_api`, `containers`, `image_registration`, `health_check` and `backups`.

RegisterIntegration

Use one server-side manifest when your resource owns several pieces of an integration:

```
local ok, result = exports.nord_inventory:RegisterIntegration({
    name = 'Example Medical',
    version = '1.0.0',
    author = 'Example Studio',
    description = 'Native Nord Inventory integration',
    dependencies = { 'nord_inventory' },

    items = {
        medical_scanner = {
            label = 'Medical Scanner',
            description = 'Portable diagnostic device',
            weight = 1200,
            stack = false,
            close = true,

            -- Optional asset copied from this provider resource.
            imageSource = 'install/nord_inventory/medical_scanner.png',

            client = {
                export = 'UseMedicalScanner',
                remove = 0
            },

            metadataDefaults = {
                serial = '',
                owner = ''
            },
```

```
            metadataSchema = {
                serial = { type = 'text', label = 'Serial' },
                owner = { type = 'text', label = 'Owner' }
            },

            actions = {
                {
                    id = 'inspect',
                    label = 'Inspect Scanner',
                    icon = 'fa-magnifying-glass',
                    side = 'client',
                    export = 'InspectMedicalScanner',
                    close = false
                }
            }
        }
    },

    hooks = {
        itemUsed = function(payload)
            -- Provider-scoped hook; removed automatically with the integration.
        end
    }
})
```

Client exports in the provider resource:

```
exports('UseMedicalScanner', function(item, slot)
    StartMedicalScanner(item, slot)
    return true
end)

exports('InspectMedicalScanner', function(item, slot, actionId)
    InspectScanner(item, slot)
    return true
end)
```

RegisterItem / RegisterItems

```
local ok, def = exports.nord_inventory:RegisterItem('service_token', {
    label = 'Service Token',
    weight = 5,
    stack = true,
    close = false,
    metadataDefaults = { issuer = '', issuedAt = 0 },
    client = { export = 'UseServiceToken', remove = 0 }
})
```

Batch form:

```
exports.nord_inventory:RegisterItems({
    item_one = { label = 'Item One', weight = 100 },
    item_two = { label = 'Item Two', weight = 200 }
})
```

Use handlers

Attach a callback/funcref:

```
exports.nord_inventory:RegisterUsableItem('legacy_item', function(source, item, definition,
slot)
    return true
end, 'server')
```

Or attach an external export:

```
exports.nord_inventory:RegisterUseExport('carplay', 'carplay', 'client')
```

Full explicit form:

```
exports.nord_inventory:RegisterUseExport('service_token', {
    resource = 'my_resource',
    export = 'UseServiceToken',
    side = 'server'
})
```

Cross-resource FiveM funcrefs are accepted and wrapped safely; they are not indexed as ordinary Lua tables.

Unregister ownership

```
exports.nord_inventory:UnregisterItem('service_token')
exports.nord_inventory:UnregisterItems({ 'item_one', 'item_two' })
exports.nord_inventory:UnregisterUseHandler('legacy_item')
exports.nord_inventory:UnregisterIntegration()
```

These calls are provider-scoped. A resource cannot unregister another provider's owned registration.

Provider inspection

```
local provider = exports.nord_inventory:GetRegisteredProvider('service_token')
local providers = exports.nord_inventory:GetProviderRegistry()
local debug = exports.nord_inventory:DebugItem('service_token')
```

`DebugItem` reports the definition source, provider, registered actions, metadata extension and use-handler presence.

Item Actions API

Add an action to an existing item context menu:

```
exports.nord_inventory:RegisterItemAction('id_card', {
    id = 'verify_card',
    label = 'Verify Card',
    icon = 'fa-id-card',
    side = 'server',
    export = 'VerifyCard',
    close = false
})

exports('VerifyCard', function(source, item, slot, definition, actionId)
```

```
    -- Server-side validation here.
    return true
end)
```

Remove only your own action:

```
exports.nord_inventory:UnregisterItemAction('id_card', 'verify_card')
```

Read public actions:

```
local actions = exports.nord_inventory:GetItemActions('id_card')
```

Metadata Schema API

```
exports.nord_inventory:RegisterMetadataSchema('employee_card', {
    employeeName = { type = 'text', label = 'Employee' },
    company = { type = 'text', label = 'Company' },
    employeeId = { type = 'text', label = 'Employee ID' }
}, {
    employeeName = true,
    company = true,
    employeeId = true
})
```

The extension is merged into the item metadata presentation without rewriting the authoritative item definition.

```
local schema = exports.nord_inventory:GetMetadataSchema('employee_card')
exports.nord_inventory:UnregisterMetadataSchema('employee_card')
```

Item Image Registration

Trusted server resources can copy a PNG/WebP shipped inside their own resource:

```
local ok, imageName = exports.nord_inventory:RegisterItemImage(
    'install/nord_inventory/service_token.png',
    'service_token.png'
```

```
)
```

The API rejects traversal paths, unsupported formats, oversized images and target-name ownership conflicts.

For `RegisterIntegration`, use `imageSource` directly inside an item to copy the asset and assign the resulting image name to the definition.

Durability API

```
local value, maximum = exports.nord_inventory:GetDurability(source, slot)
exports.nord_inventory:SetDurability(source, slot, 50)
exports.nord_inventory:AddDurability(source, slot, 20)
exports.nord_inventory:RemoveDurability(source, slot, 10)
exports.nord_inventory:RepairItem(source, slot)      -- full repair
exports.nord_inventory:RepairItem(source, slot, 25) -- add 25
```

All mutations run through the normal Nord inventory metadata path and sync the affected inventory.

Containers

```
exports.nord_inventory:OpenItemContainer(source, slot)
local container = exports.nord_inventory:GetContainerInventory(source, slot)
```

The slot must contain an item whose Nord definition has a `container` configuration.

Hooks

```
exports.nord_inventory:RegisterHook('itemUsed', function(payload)
    print(payload.source, payload.item and payload.item.name)
end)
```

Provider-owned hooks are removed on `UnregisterIntegration()` or resource stop.

Hook names emitted by the current core include `beforeUseItem`, `itemUsed`, `itemAdded`, `itemRemoved`, `itemMoved`, `metadataChanged`, `itemRepaired`, `itemBroken`, `itemAction`, `inventoryOpened` and `inventoryClosed`.

Health diagnostics

```
local report = exports.nord_inventory:GetHealthReport()
```

The report is also visible in Admin Studio and checks provider/resource state, dependencies, export descriptors, missing images, durability configuration, registration conflicts and orphan extensions.

Metadata privacy from `RegisterItem`

External resources can decide whether an item's metadata detail cards are visible to players directly when registering the item.

Use the canonical option:

```
exports['nord_inventory']:RegisterItem('nr_food', {
    label = 'Restaurant Food',
    weight = 250,
    stack = true,

    -- Player can still see metadata.label / metadata.image as presentation,
    -- but the remaining metadata values are hidden from the item-info modal.
    hideMetadataDetails = true
})
```

Then an instance can still use normal presentation metadata:

```
exports['nord_inventory']:AddItem(source, 'nr_food', 1, {
    label = 'Cute Cafe Burger',
    image = 'cute_cafe_burger.png',

    recipeId = 14,
    restaurant = 'cute_cafe',
```

```
    hunger = 25,
    basePrice = 20
})
```

The player sees `Cute Cafe Burger` and `cute_cafe_burger.png`, but does not see `recipeId`, `restaurant`, `hunger`, `basePrice`, or other metadata detail cards.

The metadata is **not deleted or filtered from server exports**. Scripts still receive the complete metadata.

For compatibility, these registration aliases are also accepted:

```
hideMetadata = true
metadataPrivate = true
metadataPlayerVisible = false
metadataVisibility = false
metadataVisibility = 'hidden'
```

Nord normalizes all of them internally to:

```
hideMetadataDetails = true
```

The same option works with `RegisterItems` and with each item inside `RegisterIntegration`.

Metadata details are hidden by default

All registered items now use a privacy-first default.

This is enough:

```
exports['nord_inventory']:RegisterItem('nr_food', {
    label = 'Restaurant Food',
    weight = 250
})
```

Metadata detail cards are hidden automatically.

Instance presentation still works:

```
exports['nord_inventory']:AddItem(source, 'nr_food', 1, {
    label = 'Cute Cafe Burger',
    image = 'cute_cafe_burger.png',
    recipeId = 14,
    restaurant = 'cute_cafe'
})
```

The player sees `Cute Cafe Burger` and its metadata image, but does not see `recipeId`, `restaurant`, or other metadata cards.

To deliberately show metadata details for one item:

```
exports['nord_inventory']:RegisterItem('identity_card', {
    label = 'Identity Card',
    weight = 10,
    hideMetadataDetails = false
})
```

The equivalent opt-in alias is:

```
showMetadataDetails = true
```

You can still use per-field `metadataDisplay` / schema `show` settings once the item has opted into visible metadata details.

Register items together with their category

For integrations such as mechanics, restaurants or jobs, you can register the category and all items in one call. Every item automatically receives the category; you do not need to repeat `category = 'mechanic'` on each definition.

```
local ok, result = exports['nord_inventory']:RegisterCategoryItems({
    name = 'mechanic_materials',
    label = 'Mechanic Materials',
    icon = 'fa-screwdriver-wrench',
    order = 80
}, {
```

```
    mech_sandpaper = {
        label = 'Sandpaper',
        weight = 150,
        stack = true
    },

    mech_primer = {
        label = 'Paint Primer',
        weight = 500,
        stack = true
    },

    mech_clearcoat = {
        label = 'Clear Coat',
        weight = 500,
        stack = true
    }
})
```

The result includes the category state plus registered/failed items:

```
{
    category = 'mechanic_materials',
    categoryState = 'created', -- or existing
    registered = { 'mech_sandpaper', 'mech_primer', 'mech_clearcoat' },
    failed = {}
}
```

The equivalent alias is:

```
exports['nord_inventory']:RegisterItemsWithCategory(category, items)
```

A string category name is also supported with the category definition as the third argument:

```
exports['nord_inventory']:RegisterCategoryItems(
    'mechanic_materials',
    items,
    { label = 'Mechanic Materials', icon = 'fa-screwdriver-wrench' }
)
```

Register a complete item catalog

Use `RegisterItemCatalog` when one resource owns several categories:

```
exports['nord_inventory']:RegisterItemCatalog({
    mechanic_paint = {
        label = 'Paint Materials',
        icon = 'fa-spray-can-sparkles',
        order = 80,
        items = {
            mech_basecoat = { label = 'Base Coat', weight = 500 },
            mech_clearcoat = { label = 'Clear Coat', weight = 500 }
        }
    },

    mechanic_tools = {
        label = 'Mechanic Tools',
        icon = 'fa-screwdriver-wrench',
        order = 90,
        items = {
            mech_polisher = { label = 'Polisher', weight = 1800, stack = false },
            mech_wrench = { label = 'Workshop Wrench', weight = 900, stack = false }
        }
    }
})
```

Categories are created/reused first, then their items are registered. Runtime items still follow the normal provider ownership rules and should be registered again when the provider resource starts. Custom category folders are persistent.

Dynamic category registration

External resources can install their own item categories before registering items. Category registration is idempotent: if the category already exists, Nord Inventory reuses it instead of creating a duplicate.

Register one category

```
local ok, category, state = exports['nord_inventory']:RegisterCategory(
    'restaurant_food',
    {
        label = 'Restaurant Food',
        icon = 'fa-utensils',
        order = 100
    }
)
```

`state` is `created` or `existing`.

Register several categories

```
exports['nord_inventory']:RegisterCategories({
    restaurant_food = {
        label = 'Restaurant Food',
        icon = 'fa-utensils',
        order = 100
    },

    restaurant_drinks = {
        label = 'Restaurant Drinks',
        icon = 'fa-glass-water',
        order = 110
    }
})
```

Create a category directly from `RegisterItem`

An item can carry an inline category definition:

```
exports['nord_inventory']:RegisterItem('nr_burger', {
    label = 'Restaurant Burger',
    weight = 250,

    category = {
        name = 'restaurant_food',
```

```
        label = 'Restaurant Food',
        icon = 'fa-utensils',
        order = 100
    }
})
```

Nord creates/reuses `restaurant_food` first, then registers the item with:

```
category = 'restaurant_food'
```

`RegisterIntegration` categories

This is the recommended form for complete script integrations:

```
exports['nord_inventory']:RegisterIntegration({
    name = 'Nord Restaurants',

    categories = {
        restaurant_food = {
            label = 'Restaurant Food',
            icon = 'fa-utensils',
            order = 100
        },

        restaurant_drinks = {
            label = 'Restaurant Drinks',
            icon = 'fa-glass-water',
            order = 110
        }
    },

    items = {
        nr_burger = {
            label = 'Burger',
            category = 'restaurant_food',
            weight = 250
        },

        nr_coffee = {
```

```
            label = 'Coffee',
            category = 'restaurant_drinks',
            weight = 300
        }
    }
})
```

Categories are registered before items in the same integration call.

Existing categories

If a category already exists, it is reused and is not overwritten by another resource. A resource that originally created a custom category may explicitly update it:

```
exports['nord_inventory']:RegisterCategory('restaurant_food', {
    label = 'Restaurant Meals',
    icon = 'fa-utensils',
    order = 90,
    updateExisting = true
})
```

Only the owning resource can use `updateExisting = true`.

Read categories

```
local category = exports['nord_inventory']:GetCategory('restaurant_food')
local categories = exports['nord_inventory']:GetCategories()
```

Remove an owned category

```
local ok, reason = exports['nord_inventory']:UnregisterCategory('restaurant_food')
```

For safety, a category can only be removed by the resource that created it and only while it contains no active items. `UnregisterIntegration` does not delete persistent category folders automatically.

Revision #1
Created 2026-09-30 01:04:38 UTC by Riguita
Updated 2026-09-30 01:04:38 UTC by Riguita