6 · Bridges & Compatibility

Keep custom integrations isolated from the core by using the bridge layer.

- Bridge Architecture
- Framework Bridge
- Inventory Bridge
- Notifications & Text UI
- Custom Ownership Bridge

Bridge Architecture

Bridge selection lives in:

```
config/bridges.lua
```

```
Config.Bridges = {
    Framework = 'auto', -- auto | qb | qbx | esx | standalone | custom
    Inventory = 'auto', -- auto | nord | ox | qb | esx | custom
    Notify = 'auto',    -- auto | ox | qb | esx | gta | custom
    TextUI = 'auto',    -- auto | ox | gta | custom
}
```

For a custom resource:

1. Set the relevant bridge to `custom`.
2. Edit only the matching bridge file.
3. Return the values expected by the bridge contract.
4. Leave the core resource logic unchanged.

This keeps updates easier because custom integrations stay isolated.

Framework Bridge

Framework bridge files:

```
client/bridge/framework.lua
server/bridge/framework.lua
```

Auto-detection order:

1. `qbx_core`
2. `qb-core`
3. `es_extended`
4. standalone fallback

The server bridge provides:

- Framework detection.
- Player lookup.
- Stable identifier lookup.
- Job lookup.
- Grade normalization.
- Duty state handling.

Custom integration hooks:

```
NVKServerBridge.CustomGetPlayer(source)
NVKServerBridge.CustomGetIdentifier(source)
NVKServerBridge.CustomGetJob(source)
NVKClientBridge.CustomGetJob()
```

Return your framework's normal player/job objects in those hooks.

Inventory Bridge

Inventory bridge files:

```
client/bridge/inventory.lua
server/bridge/inventory.lua
```

Generic item checks support:

- Nord Inventory
- ox_inventory
- QB/QBX inventory through the framework player object
- ESX inventory
- Custom implementation

Native physical vehicle keys

The full physical `vehicle_key` metadata flow is native to `nord_inventory`.

Custom inventory hooks are provided for complete physical-key support:

```
NVKInventory.CustomHasItem(source, item, count)
NVKInventory.CustomRemoveItem(source, item, count)
NVKInventory.CustomHasVehicleKeyItem(source, plate)
NVKInventory.CustomResolveVehicleKeyMetadata(source, slot)
NVKInventory.CustomGiveVehicleKeyItem(source, plate, keyType, expiresAt, sessionBound)
NVKInventory.CustomRemoveVehicleKeyItem(source, plate)
NVKInventory.CustomCleanupInvalidVehicleKeyItems(source)
```

Client custom count hook:

```
NVKClientInventory.CustomGetItemCount(item, metadata)
```

If you use a custom inventory with strict physical keys, implement the physical-key hooks rather than modifying the core.

Notifications & Text UI

Notification and Text UI are independent bridges.

Files:

```
client/bridge/notify.lua
server/bridge/notify.lua
client/bridge/textui.lua
server/bridge/textui.lua
```

Supported notification modes:

```
auto | ox | qb | esx | gta | custom
```

Supported Text UI modes:

```
auto | ox | gta | custom
```

Custom notification hook:

```
function NVKNotify.Custom(message, kind, duration, title)
    -- show your notification
    return true
end
```

Custom Text UI hooks:

```
function NVKTextUI.CustomShow(id, text, options)
    -- show your UI
    return true
end

function NVKTextUI.CustomHide(id)
    -- hide your UI
    return true
end
```

The recovery interaction uses this bridge instead of hardcoded GTA help text.

Custom Ownership Bridge

Ownership is isolated in:

```
server/bridge/ownership.lua
```

For custom ownership logic:

```
Config.Ownership.Custom = true
```

Implement:

```
function NVKOwnership.CustomGetOwnerIdentifier(plate)
    -- Return the framework identifier that owns this plate.
end
```

For the recovery UI, you can also implement:

```
function NVKOwnership.CustomGetOwnedVehicles(source, identifier)
    return {
        { plate = 'ABC123', model = 'sultan' },
        { plate = 'XYZ987', model = 'adder' },
    }
end
```

Trusted standalone/garage resources can set a runtime owner through the `SetVehicleOwner` server export.