PAIO User Manual

Manual and Tutorials

2026-11-26

PopcornSAR

PAIO

Download PDF (English) PDF 다운로드 (한국어)

PAIO is an all-in-one AUTOSAR / SDV (Software Defined Vehicle) development IDE with AI built in. A single desktop application covers development for both **Platform.**

Section	What it covers
Overview	What PAIO is, the notation used in this manual, and the base specifications
Installation	Installing PAIO, signing in, and the license states
User Guide	Getting started, projects (one window, one project), and the layout of the application window
Editors	The System editor (formerly IVC), Signal, VFB, Interface, DataType, the SYSTEM panel, ECUC, and Diagnostic
Workflow	Import and export, validation, the AI assistant, code and build, remote development, the AI Workflow
Reference	Keyboard shortcuts, glossary, troubleshooting and FAQ

New to PAIO? Start with the Overview, then Installation.

PAIO Overview

PAIO is an all-in-one AUTOSAR / SDV (Software Defined Vehicle) development IDE with AI built in. A single desktop application covers development for both **Platform**.

Item	Value
Product page	https://autosar.io/products/paio
Vendor	PopcornSAR Co., Ltd.
Applies to	**PAIO 1.0.1**
Document baseline	2026-11-11

This is the product manual for PAIO 1.0.1. It covers what the current version can actually do, and the places where the UI is visible but behaves differently.

Start with the Notation Conventions, then read the Product Overview and the Specifications.

Notation Conventions

This manual uses the following notation conventions throughout.

- `This style` marks file paths, extensions, code identifiers, enumeration values, and shortcut keys.
- **Bold text** marks a label that actually appears in the UI. PAIO's UI strings are in English, so this manual quotes them verbatim instead of translating the
- Labels are written the way the UI paints them. The menu bar, section headers, panel titles and the field labels of side panels appear in upper case (**PI** rows, buttons and dialog fields keep their own case (**New Project...**, **System**, **Create**, **Location**).
- Menu paths are joined with `▸`. For example: `PROJECT ▸ Open Project...`.
- Shortcuts follow Windows and Linux conventions. On macOS, read `Ctrl` as `Cmd`.
- A blockquote (>) marks a fact that needs your attention. It is used in particular for a **label whose behavior differs from what it says** and for an **item th**
- Error messages are quoted exactly as the application prints them in English.

Product Overview

What This Chapter Covers

This chapter explains what PAIO is, what you can do with it, and how data flows through it. It does not explain how to operate each feature; it points you

- Audience: readers new to PAIO
- Prerequisites: none

Figures, versions and lists of supported items are in 02. Specifications. Terms are in 19. Glossary.

1.1 What Is PAIO

PAIO is an **all-in-one AUTOSAR / SDV (Software Defined Vehicle) development IDE with built-in AI**. One Electron-based desktop application covers both the **Platform**.

Network design, the signal database, SWC design, BSW configuration and code used to need separate tools that exchanged ARXML files. PAIO puts the w model. The folder a window opens is the project itself.

You must sign in to use PAIO. Before you sign in you see only the sign-in screen, and until the license is verified the features stay **locked (read-only)**. See

Eight Terms to Know First

Eight terms are enough to read chapters 01 to 05: **ECU**, **HPC**, **Bus / cluster**, **Signal**, **Frame**, **SWC**, **Port** and **ARXML**. Their meanings are in 19. Glossary.

1.2 What You Can Do — Feature Map

Modeling Editors (8)

Pick the editor by the task.

What you want to do	Editor to open
Draw the ECUs of the vehicle and the buses that connect them	**System**
Define the data that travels over those buses	**Signal**
Divide the software into components	**VFB**
Define the format of the data components exchange	**Interface**, **DataType**
Deploy a component to an ECU or HPC	**System** (**SW Mapping**) or **VFB** (**ECU/HPC Mapping**)
Hold the diagnostic data (DEXT)	**Diagnostic**
Create the BSW configuration values	**Module Definition**, **Module Configuration**

1. Click **Design**, the first icon of the icon rail at the far left. The **EDITOR** view opens.
2. Click a row under **DESIGN** (**System**, **VFB**, **Interface**, **DataType**, **Diagnostic**) or **MODULE** (**Definition**, **Configuration**). The editor opens in a tab named `<prc`
3. **Signal** has no row. Open it from the **Signal** icon at the left edge of the System editor canvas.

When Each Model Is Created

A new project's `design` folder holds only `IVC.io.model`, `SYSTEM.io.model` and `VFB.io.model`. The other model files appear the first time you save in tl that content, so a missing file in a new project is normal.

Development Environment

Besides modeling, PAIO has a built-in code IDE, an integrated terminal, document editing, remote development, the vECU view, ECU build and run, and Sear

Conversion · Validation · Automation

ARXML import and export, DBC import, ECU / Machine Extract, PRTE / PARA code generation, PRTE Platform Import / Auto Configuration, Validation, AI chat generate models.

1.3 Design Philosophy — the IO-First Architecture

The most important structural trait of PAIO is that **AUTOSAR is not the original**.

- The **IO Model** is the primary data model, saved as human-readable text that you can put into version control.
- ARXML and DBC are a **converter layer** on top of it.
- The editors, the AI and validation read and write only this model.
- **Import → edit → export → re-import** round trips are supported, but not every ARXML makes the trip without loss.

Diagram Coordinates Do Not Leave in the ARXML

Node positions and sizes stay in the project. **An exported ARXML contains no coordinates.**

- A first import into an empty project **lays the nodes out automatically** on a grid.
- A re-import **keeps the coordinates you arranged**. Moving a node is not a conflict.

1.4 Edit History and One Project per Window

- A **single timeline** holds the edit history. **Undo** in the **EDIT** menu reverses edits from any editor.
- A window holds exactly one project, and its title bar reads `PAIO [<project>]`.
- Creating or opening another project opens it in a window of its own and closes the current one. `PROJECT ▸ New Window` (`Ctrl+Shift+N`) opens an

1.5 AUTOSAR Support Scope

PAIO covers the **Classic Platform** and the **Adaptive Platform**. Output is fixed to AUTOSAR **R25-11** (`AUTOSAR_00054.xsd`). FlexRay is handled in the model an System editor.

1.6 Representative Workflows

Four common tasks.

A. Starting from an Existing ARXML

1. Create a project: **New Project** on the **Welcome** tab, or `PROJECT ▸ New Project...`. Enter **Name**, set **Location**, and click **Create**.
2. Click **Import** on the **DESIGN** header and choose the ARXML (or DBC) files.
3. If the conflict resolution dialog opens, choose which side to keep for each item.
4. Review the result in each editor.
5. Click **Export** on the **DESIGN** header, choose the formats and click **Export**. The files are written to `<project>/export/` unless you pick another folder.

B. Starting from Scratch

1. Create a project as in A.
2. In the **System editor**, place ECU / HPC nodes, connectors, controllers and clusters.
3. In the **Signal editor**, define frames, PDUs and signals.
4. In the **Interface** and **DataType editors**, define port interfaces and data types.
5. In the **VFB editor**, place compositions and SWCs, connect the ports, and map each composition to an ECU or HPC.
6. Open the **VALIDATION** panel with the **V** icon in the status bar and clear the violations.
7. Click **Export** on the **DESIGN** header and export the files.

C. Generating a Draft with AI

1. Open the AI chat and describe the model in natural language.
2. Review and adjust the result in each editor.

See 15. AI Assistant.

D. Delivering Only One ECU's Worth of Data to a Supplier

1. In the **System editor**, right-click the node.
2. Choose **Extract ECU** (ECU node) or **Extract Machine** (HPC node).
3. The ARXML is created under `<project>/export/<node name>/`.

See 13. Import and Export.

Specifications

What This Chapter Covers

This chapter sums up PAIO's fixed numbers, lists and constraints. It does not describe operating procedures.

- Audience: anyone evaluating PAIO, and users narrowing down a problem
- Prerequisites: none

Terms are in 19. Glossary.

2.1 Product Identification

PAIO version 1.0.1, a commercial product under the PopcornSAR PAIO License.

2.2 Supported Platforms and Installation Packages

OS	Architecture	Distribution format
Windows	x64	NSIS installer executable, ZIP
macOS	x64, arm64 (Apple Silicon)	DMG, ZIP
Linux	x64	DEB, tar.gz

Windows Installer (NSIS) Behavior

The installer runs as a wizard, installs per user by default, and lets you change the path. Uninstalling leaves the application data in place.

macOS and Linux

- The macOS build is neither code-signed nor notarized.
- On Linux the program you start is the launcher `paio`. A `.tar.gz` extract runs without the Chromium sandbox until `sudo ./install.sh` has been ru unprivileged user namespaces. The `.deb` sets up the sandbox itself.

2.3 Runtime Environment

One window is one process and one project. Opening a project folder that another PAIO window already holds focuses that window. At most **20** editor tak

2.4 Technology Stack

PAIO is an Electron-based desktop application. Code editing uses Monaco, and the terminal uses xterm.

2.5 AUTOSAR Support Specification

Schema Versions

Every export is written in the AUTOSAR **R25-11** schema (`AUTOSAR_00054.xsd`).

Supported Protocols

CAN / CAN FD, LIN and Ethernet can be added in the System editor. FlexRay has no add menu. The Signal editor has four tabs: **CAN**, **LIN**, **PDU** and **SIGNAL**.

> **FlexRay elements disappear in the System editor.** The System editor does not draw them, and saving rebuilds the IVC model from the screen, so the Fle ARXML that contains FlexRay, do not save from the System editor.

Input and Output Formats

- Import: ARXML, DBC, and ECUC Configuration ARXML.
- Export: ARXML (R25-11) and DBC (one file per CAN cluster), written to `<project>/export/` by default, and ECU / Machine Extract ARXML.

A project export produces at most 12 kinds of file. A file whose condition is not met is not created, so fewer than 12 files is normal.

2.6 Data Model Specification

A project has six fixed folders: `doc`, `design`, `code`, `workflow`, `export` and `import`. A new project's `design` folder holds only `IVC.io.model`, `SYSTEM.io` editors from the **EDITOR** view, not from the Explorer. See also 04. Workspaces and Projects.

2.7 AI Provider Specification

AI chat offers five providers, in this order: **Claude [OPUS]**, **Claude [GLM]**, **Gemini**, **Codex** and **Ollama**. Despite its name, **Claude [GLM]** calls Zhipu AI's GLM API, chat only; the AI Workflow does not use them (2.13).

2.8 Validation Specification

429 rules are registered. Validation runs when the project opens and as soon as you pause after an edit. The status bar badge shows the number of viola

2.9 License and Feature Restrictions

Until sign-in and license validation complete, PAIO is **locked (read-only)**. An expired license allows **read and import only**. Export requires online validation. License.

2.10 Security Specification

Integrity Check Results

On an installation whose signature is not valid, editing is locked read-only for the session; the application does not exit. See section 20.1 of 20. Troublesh Installation, Login and License.

2.11 Update Notification Specification

On Windows, and on Linux installed from the `.deb` package, PAIO downloads and installs an update only when you choose to. On macOS, and on Linux wit download page on request. `HELP ▸ Check for Updates` checks by hand. See also 3.5.

2.12 Known Limitations

Limits and Constraints

Once 20 editor tabs are open, opening a new file **does nothing and gives no message**. Close a tab first.

Items That Do Not Work or Are Not Provided

The following do not work or are not provided in this version: DBC encoding selection, the Signal editor **FlexRay** / **Ethernet** tabs, FlexRay in the System ed

Other Behavior to Know About

Some normal behavior is easy to mistake for a problem: an import first saves unsaved edits, and an exported ARXML does not restore the on-screen layou

2.13 AI Workflow Specification

The AI nodes of the AI Workflow run Claude through the Claude Code runtime bundled with PAIO, separate from the AI chat providers. Each project has on `workflow/WORKFLOW.io.workflow`.

Installation

How to install PAIO on your machine, sign in with an email one-time code, and check or change the license state.

See Installation, Login and License.

Installation, Login and License

What This Chapter Covers

This chapter explains how to install PAIO, sign in, and which features open or close with the license state. It also covers update notifications, the app vers installing PAIO for the first time and for users who need to check a license state. You need the installer package you received and a Google account or en the first sign-in creates the account, and it starts on a trial.

The package names and supported platforms are in 02. Specifications.

3.1 Installation

3.1.1 Windows — Installer (NSIS)

1. Run `PAIO-win-x64.exe`. It installs per-user and does not ask for administrator rights.
2. If PAIO is already installed, **Program Maintenance** appears. **Modify** and **Repair** continue with the normal installation; **Remove** runs the uninstaller and
3. Select **I agree to the License Agreement**.
4. Keep or change the installation folder (default `%LOCALAPPDATA%\Programs\PAIO`).
5. Finish. **Create a desktop icon** is selected by default.

Uninstalling does not delete user data.

3.1.2 Windows — Portable Use (ZIP)

Extract `PAIO-win-x64.zip` anywhere and run `PAIO.exe`. No shortcut or Start menu entry is created.

3.1.3 macOS

1. Open `PAIO-mac-x64.dmg` (Intel) or `PAIO-mac-arm64.dmg` (Apple Silicon) and drag PAIO into the Applications folder.
2. On first launch Gatekeeper warns you. Allow PAIO in System Settings ▸ Privacy & Security. If it says the app is damaged, run `xattr -dr com.apple.` it again.

3.1.4 Linux

- **Debian/Ubuntu**: `sudo apt install ./PAIO-linux-x64.deb`. The package registers the icon and sets up the Chromium sandbox.
- **tar.gz**: extract `PAIO-linux-x64.tar.gz` and run `paio` in the extracted folder. Do not start `.paio-bin` directly.

3.1.5 Installation Integrity Check (Windows Only)

Right after launch, PAIO checks the code signature of `PAIO.exe` in the background. If the executable was modified or re-signed by another signer, editin message appears once:

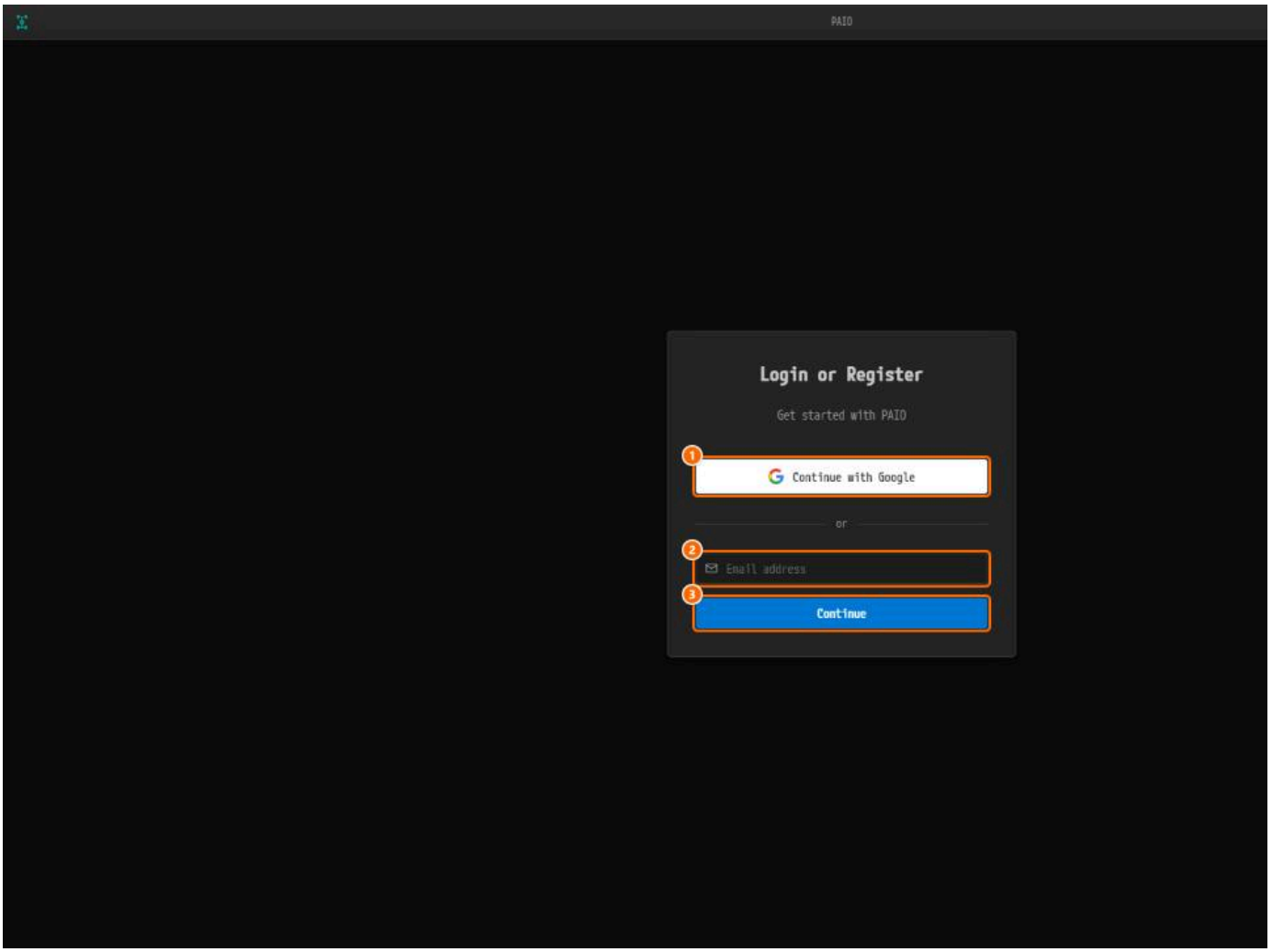
> `PAIO could not verify the integrity of its installation.` `Please reinstall PAIO. Editing is disabled for this session.`

A valid license does not release this lock. Reinstall an official build. The diagnostic file is in 20. Troubleshooting and FAQ 20.1.

3.2 First Launch — Sign In

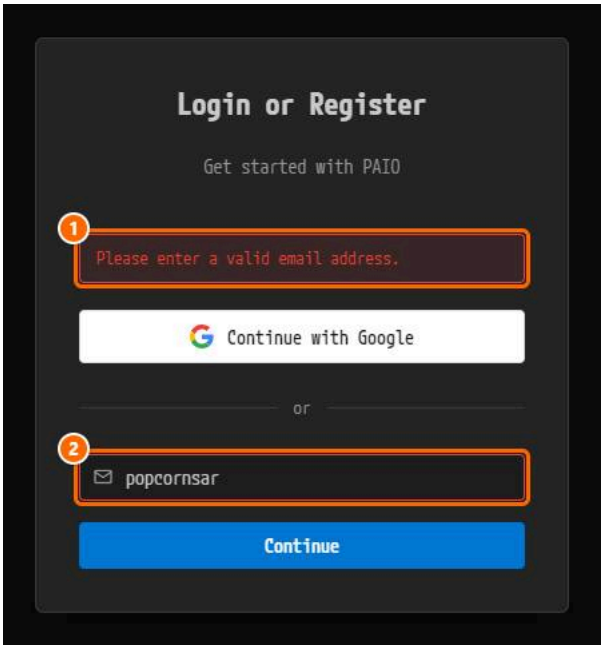
PAIO shows the main screen only after you sign in. There are no passwords: sign in with a Google account or an email one-time code (OTP).

3.2.1 The Login Screen



*The login screen. ① **Continue with Google**, ② the email field, ③ **Continue**.*

If the email is not valid, `Please enter a valid email address.` appears and nothing is sent.



*Clicking **Continue** with `popcornsar`. ① The error message, ② the field with a red border.*

3.2.2 Method 1 — Google Account

1. Click **Continue with Google**. The system browser opens the Google sign-in page.
2. Finish signing in in the browser. PAIO signs you in; there is nothing to paste.

To stop waiting, click **Cancel**. After **5 minutes** PAIO stops waiting; click **Continue with Google** again.

3.2.3 Method 2 — Email One-Time Code (OTP)

1. Enter your email address and click **Continue**.
2. On the **Enter your code** screen, enter the 6-digit code from the email and click **Continue**. The code is valid for **10 minutes** and only once.
3. If the code does not arrive, click **Resend code** (enabled after 60 seconds).

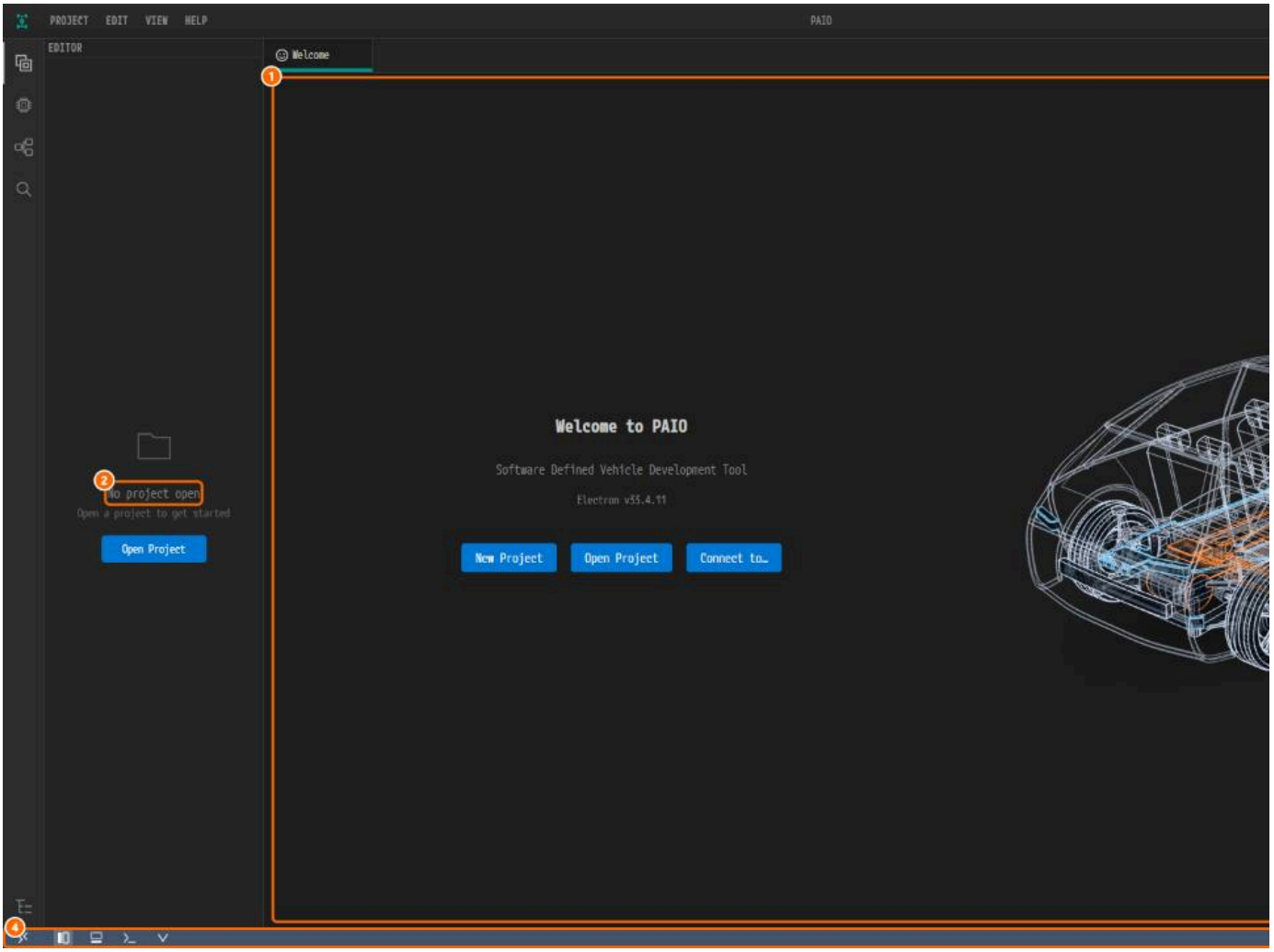


*The **Enter your code** screen (example address). ① The address, ② the code field, ③ **Continue**, ④ the **Resend code** countdown, ⑤ **Back to login**.*

A wrong or expired code shows `Invalid or expired code`.

3.2.4 The First Window After Sign-In

After sign-in PAIO reopens the project you worked on last, if its folder still exists. Otherwise the window has no project open.



*A window with no project open. ① The **Welcome** tab, ② `No project open` in the **EDITOR** view, ③ the top-bar controls, ④ the status bar.*

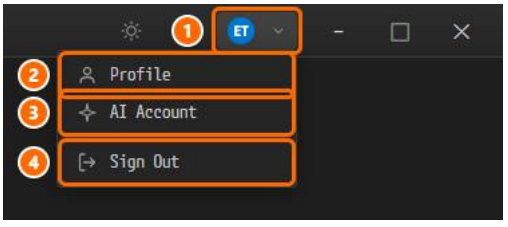
Creating and opening projects is covered in 04. Workspaces and Projects.

3.2.5 When Sign-In Fails

A failed sign-in shows one line on the card, such as `Can't reach the PAIO server (pars.popcornsar.com). Check your network or proxy.` On a c
`pars.popcornsar.com`.

3.3 User Menu and Profile

Click the user avatar on the right of the top bar for **Profile**, **AI Account** and **Sign Out**.



*The user menu. ① The avatar, ② **Profile**, ③ **AI Account**, ④ **Sign Out**.*

- **Profile** shows the account email, the current plan and its status. Nothing on it can be edited. **Subscribe** opens the pricing page.
- **AI Account** connects the Claude account used by the AI nodes of the AI Workflow (21.11). AI chat providers are set up in the chat (15.2).
- **Sign Out** asks first only when tabs have unsaved changes.



*The Profile screen on a trial (example address). ① **Account**, ② **Current Plan**, ③ **Subscribe**, ④ **Sign Out**, ⑤ the close button.*

3.4 License and Feature Restrictions

3.4.1 Plans and States

Permissions follow the license state, not the plan. Only `expired` and `readonly` are restricted. The trial runs for **30 days from the first sign-in in the app**.

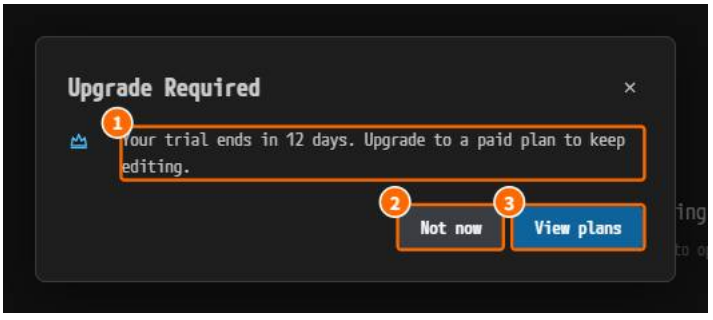
3.4.2 Permission Table

State	Read	Modify	Create	Delete	Export	Import
`trial` / `active`	O	O	O	O	O	O
`expired` / `readonly`	O	X	X	X	X	O

Import is allowed in every state. The AI assistant is unavailable in a restricted state.

3.4.3 What You See Without Permission

An action without permission opens the **Upgrade Required** dialog. **View plans** opens the pricing page; **Not now** closes the dialog.



*The **Upgrade Required** dialog on a 12-day trial. ① The message, ② **Not now**, ③ **View plans**.*

> **Purchase happens on the website.** The app has no payment screen and no license key field. After you subscribe or renew, return to PAIO; it picks up th

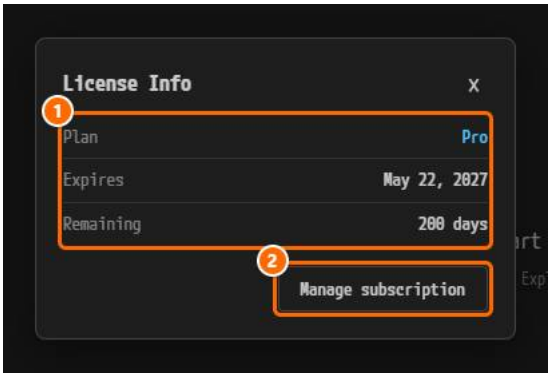
3.4.4 Checking the License State

The license is shown on the right of the status bar, for example `Trial: 12d` or **Pro**.



① The license indicator, here on a Pro license.

Click it. On a trial the **Upgrade Required** dialog opens; on any other plan the **License Info** dialog opens with **Plan**, **Expires** and **Remaining**.



*The **License Info** dialog on a Pro license. ① **Plan, Expires** and **Remaining**, ② **Manage subscription**.*

3.4.5 Offline Behavior and Export

The license is cached for **4 hours**, so you can keep working offline for that long. Export checks the license again every time it runs. It is refused, for exam usable cache exists.

3.4.6 Session Invalidation

One account can be used on **one device** at a time. Signing in on another device signs this device out the next time PAIO re-checks the license, for exampl save unsaved tabs first.

3.5 Update Notifications

PAIO checks for a newer version in the background and shows a **Software Update** toast at the bottom right. To check yourself, choose **HELP** ▸ **Check for U**

- On Windows, click **Update**. PAIO waits for other windows and running tasks to end, downloads and installs the update, and restarts. The window disa
- On Linux, when PAIO was installed from the `.deb` package, **Update** works the same way and asks for an administrator password before it installs. If a moment, PAIO waits up to 2 minutes; if it is still busy, PAIO installs nothing and shows **Update** again, so press it once the other installation has finishe
- On macOS, and on Linux with the `tar.gz`, PAIO leads you to the download page; install the new version as in 3.1.

See also 20.12.

3.6 Checking App Information

Choose **HELP** ▸ **About PAIO** to see the version. The version is also shown at the far right of the status bar as `v<version>`.

3.7 Support and Issue Reports

Choose **HELP** ▸ **Report Issue**. The support Q&A page (`https://autosar.io/support/qna`) opens in the browser.

3.8 Where Account and License Data Are Stored

Authentication tokens, the license cache, the machine identifier and the AI API key are stored once per device; the tokens, the cache and the key are enc
folder. Where OS secure storage is unavailable, you must sign in again at every restart; on Linux, install `libsecret`.

User Guide

How to start working in PAIO. A PAIO window holds one project: the folder it opens is the project, and the title bar shows its name as `PAIO [<project>]`.

Page	What it covers
Getting Started	The first launch, creating a project, and opening the first editor
Workspaces and Projects	Creating and opening a project, and how a window relates to its project folder
UI Guide	Title bar, icon bar, **EDITOR** view, tabs and status bar

Getting Started

This guide walks through the minimal steps from a freshly installed PAIO to an open System editor.

Install and Sign In

Install PAIO and sign in as described in Installation, Login and License.

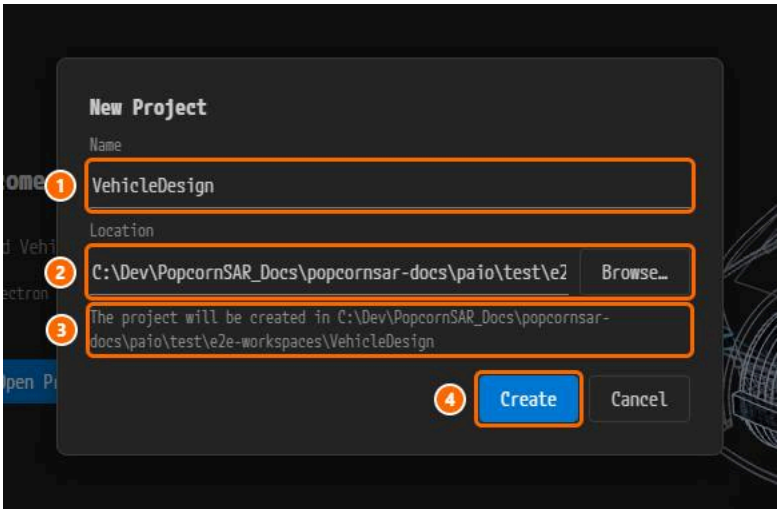
A Window with No Project

When no project is open, the title bar reads `PAIO` and the editor area shows the **Welcome** tab (`Welcome to PAIO`) with the buttons **New Project**, **Open Pr** left reads `No project open` and offers an **Open Project** button.

The same commands are in the **PROJECT** menu: **New Project...**, **Open Project...** (`Ctrl+O`), **Open Recent**, **New Window** (`Ctrl+Shift+N`), **Save** (`Ctrl+S`), **Save** shortcut.

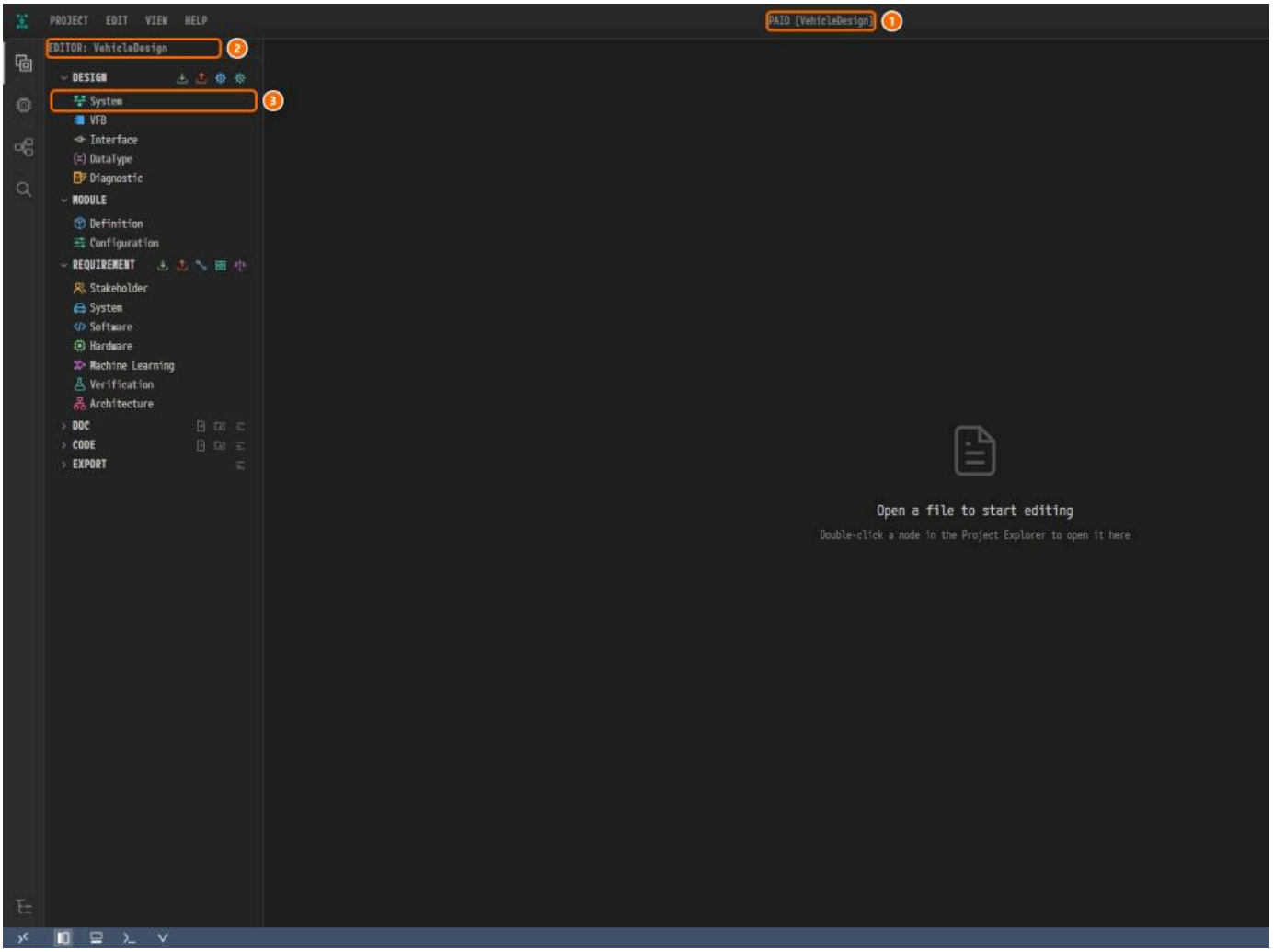
Create a Project

1. On the **Welcome** tab, click **New Project** (or choose `PROJECT ▸ New Project...`).
2. In the **New Project** dialog, type the project name into **Name**.
3. Set **Location** to the folder that will hold the project, or click **Browse...** to pick it.
4. Check the line below the fields, `The project will be created in <Location>\<Name>`, and click **Create**.



*The **New Project** dialog. ① **Name**, ② **Location** and the **Browse...** button, ③ the line that names the new project folder, ④ the **Create** button.*

PAIO creates the folder `<Location>\<Name>` and opens it as the window's project. The title bar changes to `PAIO [<Name>]`, and the **EDITOR** view header expanded.



*The window right after **Create**, for a project named `VehicleDesign`. ① The title bar `PAIO [VehicleDesign]`, ② the view header `EDITOR: VehicleDesign` editor, the editor area shows only `Open a file to start editing`.*

Open the First Editor

In the **EDITOR** view, click **System** under **DESIGN**. The System editor opens in the tab `<Name>:System`. The other design editors open the same way from th

Next Steps

- Workspaces and Projects: creating and opening projects in detail.
- UI Guide: the parts of the application window.

Workspaces and Projects

What This Chapter Covers

This chapter explains how PAIO holds your work: a window opens one folder, and that folder is the project. It covers creating and opening a project, the p saving, and the edit **History**. It is written for users starting their first design in PAIO. You need a login and an active license (03. Installation, Login and Lice delete, rename and save in this chapter is blocked.

4.1 Concepts

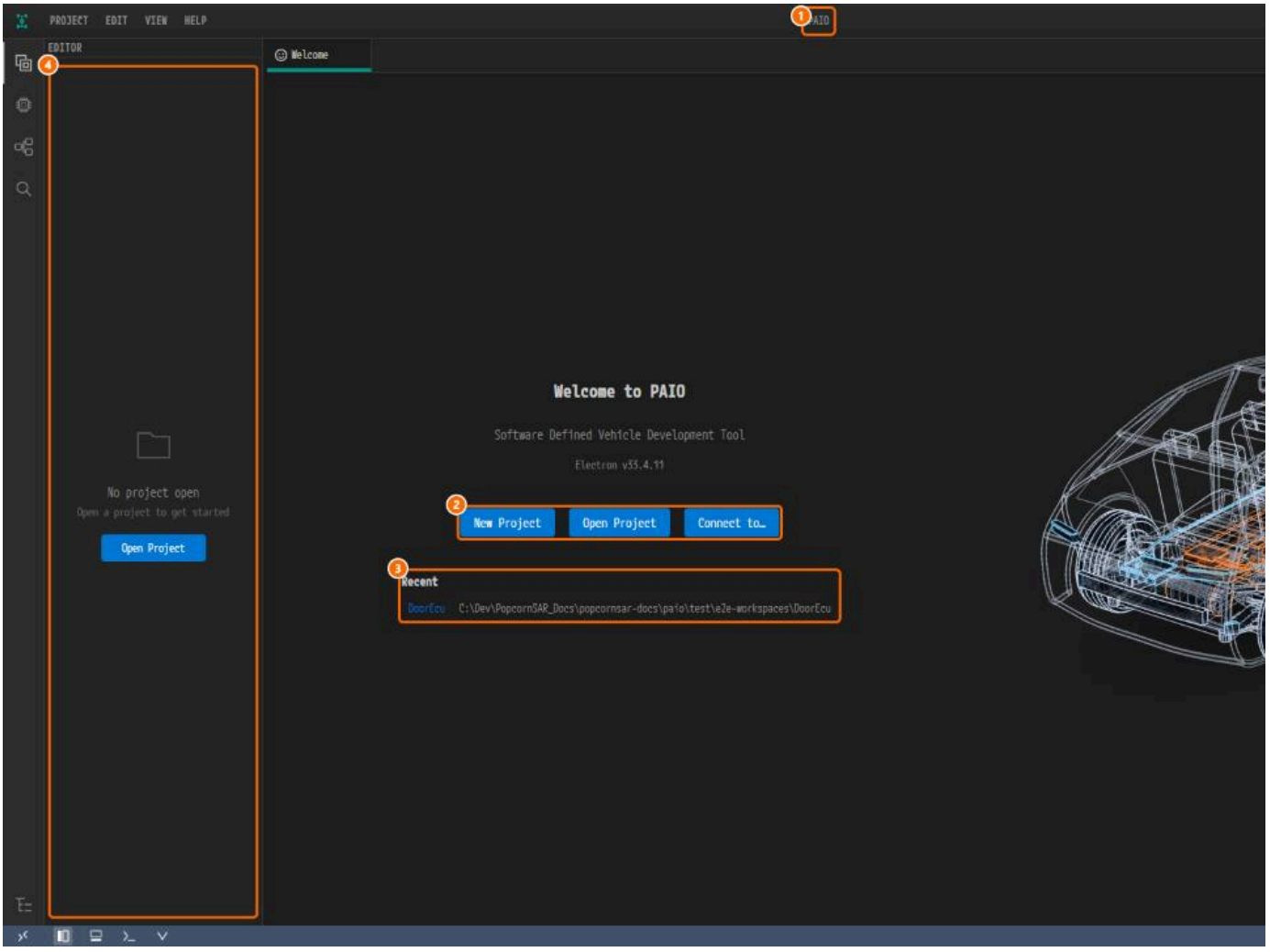
In PAIO the workspace and the project are the same thing. The folder a window opens is the project.

```
DoorEcu/                ← the folder the window opens = the project
 ├── DoorEcu.io.prj     ← project metadata file (hidden in the Explorer)
 ├── code/  design/  doc/  export/  import/  workflow/
 └── ...
```

- **One window, one project.** To work on another project, open it in another window (`PROJECT ▸ New Window`, `Ctrl+Shift+N`).
- **The folder name is the project name.**
- **Opening a folder makes it a project.** If it has no metadata file, PAIO adds the metadata file, the fixed folders and the default models without asking. M

4.2 Opening a Project

A window with no project open has the title `PAIO` and shows the **Welcome** tab.



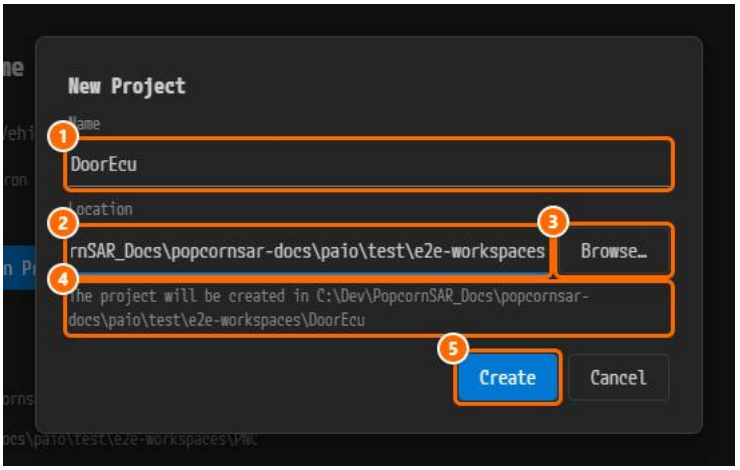
1. Choose **PROJECT ▸ Open Project...** (`Ctrl+O`), or click **Open Project** on the **Welcome** tab. A recent project opens from **Open Recent** or the **Recent** list.
2. Pick the project folder in the folder selection dialog.

The window title changes to `PAIO [<project>]` and the sidebar shows `EDITOR: <project>`. In a window that already has a project, the folder opens in
Remote connections are in 17. Remote Development (SSH, vECU Container).

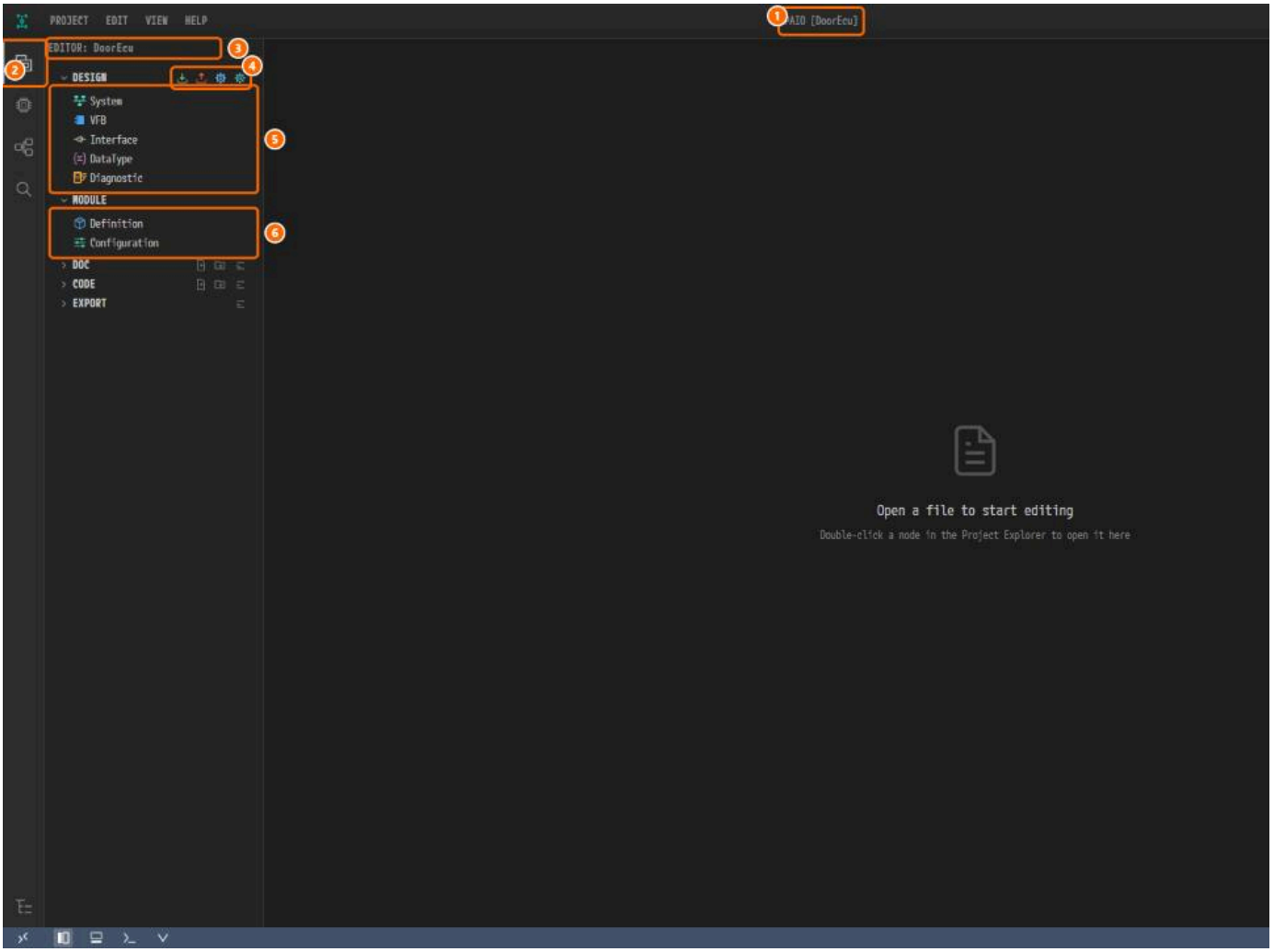
4.3 Creating a Project

4.3.1 Procedure

1. Click **New Project** on the **Welcome** tab, or run `PROJECT ▸ New Project...`.
2. Type the project name into **Name**.
3. Set **Location**, the folder the project is created in. Type the path or click **Browse...**.
4. Check the line below the fields: `The project will be created in <Location>\<Name>`.
5. Click **Create**.



The window title reads `PAIO [<Name>]` and the sidebar `EDITOR: <Name>`, with **DESIGN** and **MODULE** expanded. Click **System** under **DESIGN** to open the Sy



In a window that already has a project, the new project opens in a new window and the current window closes.

4.3.2 Name Rules

Use letters, digits, hyphens and underscores only, for example `DoorEcu`, `body-domain` or `door_ecu_01`. Windows reserved names and a project that alre

4.3.3 What Gets Created

A new project holds the metadata file `<Name>.io.prj`, six fixed folders (`code`, `design`, `doc`, `export`, `import`, `workflow`) and three default models in `de` `IVC.io.model`). Existing folders and models are never overwritten.

4.3.4 Quick Start Projects

The **PROJECT** menu no longer has **Quick Start**. Every project is created with **New Project** (4.3.1).

4.4 Project Folder Structure

Folder	Purpose
`code/`	C/C++/Rust sources and build files
`design/`	Model files (`*.io.model`) and the `SWC/` and `Module/` subfolders
`doc/`	Documents
`export/`	Export results
`import/`	A place for import sources. An ARXML or DBC import does not copy its source here
`workflow/`	Files of the AI Workflow

`code`, `design`, `doc`, `export` and `workflow` cannot be renamed or deleted.

4.5 The design Folder and the EDITOR View

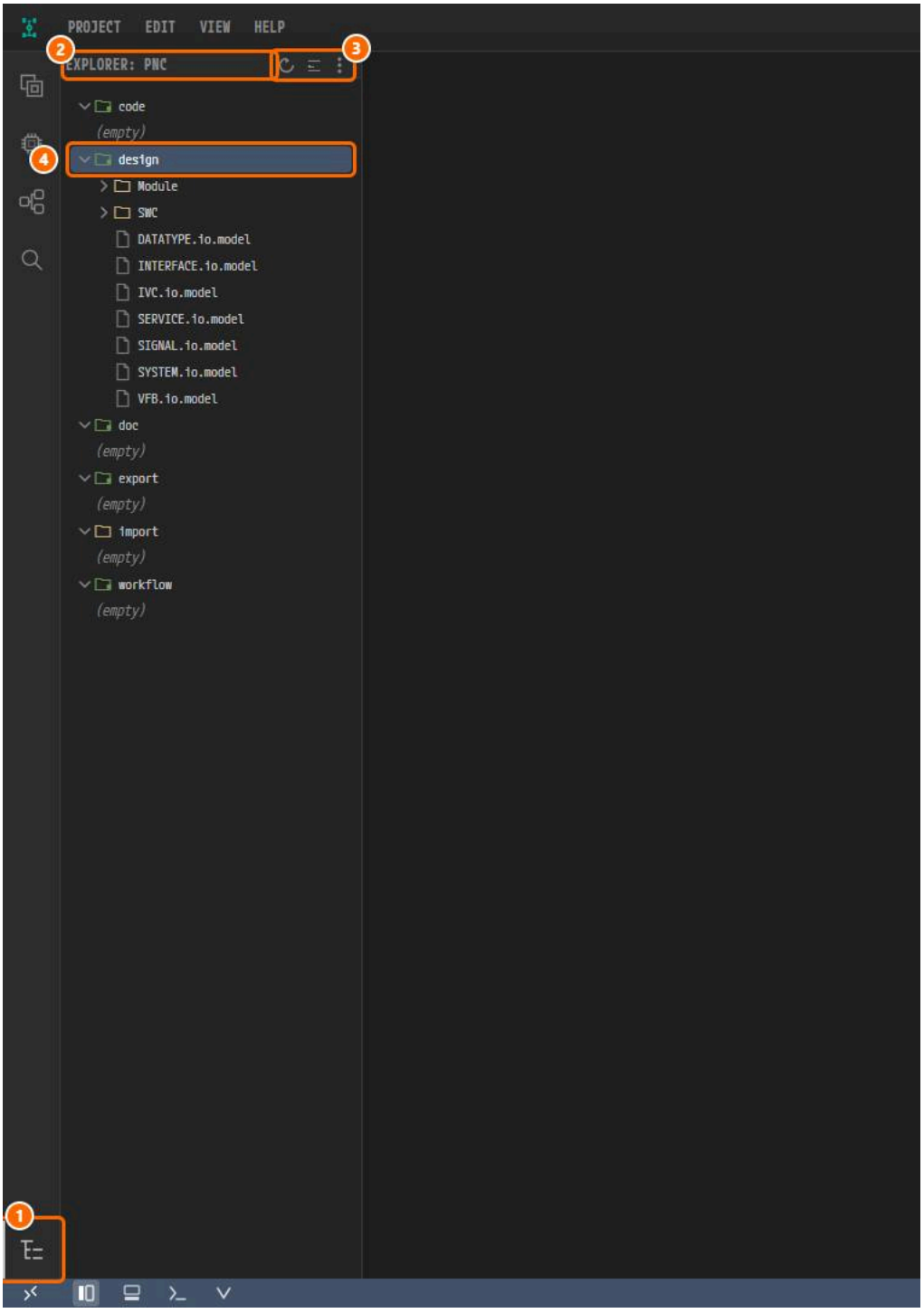
The design editors open from the **EDITOR** view, not from the Explorer.

1. Click a row under **DESIGN** (**System**, **VFB**, **Interface**, **DataType**, **Diagnostic**) or **MODULE** (**Definition**, **Configuration**).
2. The editor opens in a tab named `<project>:<row>`, for example `DoorEcu:System`.

- **Signal has no row.** The Signal editor opens from the **Signal** icon on the System editor canvas (06. System Editor, 07. Signal Editor).
- The **DESIGN** header icons **Import** and **Export** are described in 13. Import and Export. **MODULE** is described in 12. ECUC.
- Double-clicking a `*.io.model` file in the Explorer opens it as read-only JSON, not a design editor.

4.6 Working in the Explorer

Click the **Explorer** icon at the bottom of the rail. The tree lists the project folder directly, with no project row on top.



4.6.1 Selecting

Click to select, `Ctrl` + click or `Shift` + click to select several, and double-click to open a file. At most 20 tabs can be open.

4.6.2 Explorer Keyboard Shortcuts

With the tree focused, `Ctrl+C`, `Ctrl+X` and `Ctrl+V` copy, cut and paste, `Ctrl+A` selects every expanded node, and `Delete` deletes. They do not apply

4.6.3 Sidebar Header Buttons

The header buttons appear while the pointer is over the header: **Refresh Explorer**, **Collapse All** (collapses or expands) and **More Actions...**, which holds **Sh**

4.6.4 Drag and Drop

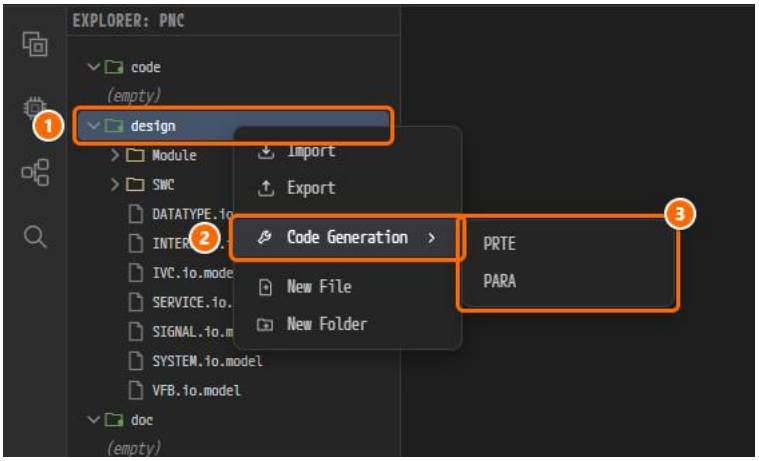
1. Drag a file or folder.
2. Hover over the middle of a folder row. A collapsed folder expands after about 0.6 seconds.

3. Release to move it into that folder.

If the name already exists there, PAIO renames the item instead of overwriting.

4.6.5 Context Menus (Right-Click)

The menu depends on what you right-click: the empty area, a fixed folder, an ordinary folder or a file. The `design` folder has **Import**, **Export** and **Code Ger**



4.6.6 Creating Files and Folders

1. Right-click a folder and choose **New File** or **New Folder**.
2. Enter the name and click **Create**. A new file opens in an editor tab.

`doc` accepts only `.md` and `.markdown`; `code` accepts only C, C++ and Rust source extensions.

4.6.7 Renaming and Deleting

- **Rename** edits only the name; a file keeps its extension. Open tabs and the edit history follow the new path.
- **Delete** (or the `Delete` key) asks for confirmation first. Deletion cannot be undone.
- When the file system refuses a rename, a deletion or a new file or folder (no write permission), the toast `Permission denied` appears.

4.6.8 Cut, Copy, and Paste

Paste puts the items into the selected folder, or into the folder of the selected file. When the name already exists, PAIO renames the copy, for example `m`

4.7 Renaming and Deleting a Project

PAIO has no command to rename or delete a project. The project is its folder, so you rename or delete the folder.

4.7.1 Rename Project

1. Close the PAIO window that has the project open.
2. Rename the project folder with the OS file manager. The new name must follow the rules in 4.3.2.
3. Open the renamed folder in PAIO (`PROJECT ▸ Open Project...`). PAIO renames the metadata file to match.

4.7.2 Delete Project

1. Close the PAIO window that has the project open.
2. Delete the project folder with the OS file manager.

Remove the entry left in the **Recent** list with `PROJECT ▸ Open Recent ▸ Clear Recently Opened`.

4.8 Saving

- **Save** (`PROJECT ▸ Save`, `Ctrl+S`) saves the active editor.
- **Save All** (`PROJECT ▸ Save All`) saves every changed file of the project.

- There is no **Save As...**. Changes in the SYSTEM panel are written as you make them.
- Closing the window with unsaved changes shows the **Unsaved Changes** dialog with **Save**, **Don't Save** and **Cancel**.

While the license is expired or read-only, saving opens an upgrade dialog instead.

4.9 Edit History (History)

The **History** tab of the bottom utility panel shows the project's edit history. Toggle the panel with the **Toggle Panel** button at the left of the status bar.

- The newest entry is at the top. Entries above **Current state** can be redone; entries below can be undone.
- **Double-click** an entry to jump to that point.
- A project keeps up to 25 entries.

4.10 Watching External Changes and Automatic Recovery

PAIO watches the project folder, so files added or deleted outside the app appear in the tree. A fixed folder deleted outside the app is recreated. When a
on disk, PAIO keeps what is on screen, and saving overwrites the disk.

4.11 Editing Files Directly

Models are stored as text JSON, so you can put them under configuration management. Editing them outside PAIO is not recommended: references can b
problems with the validation panel (14. Validation).

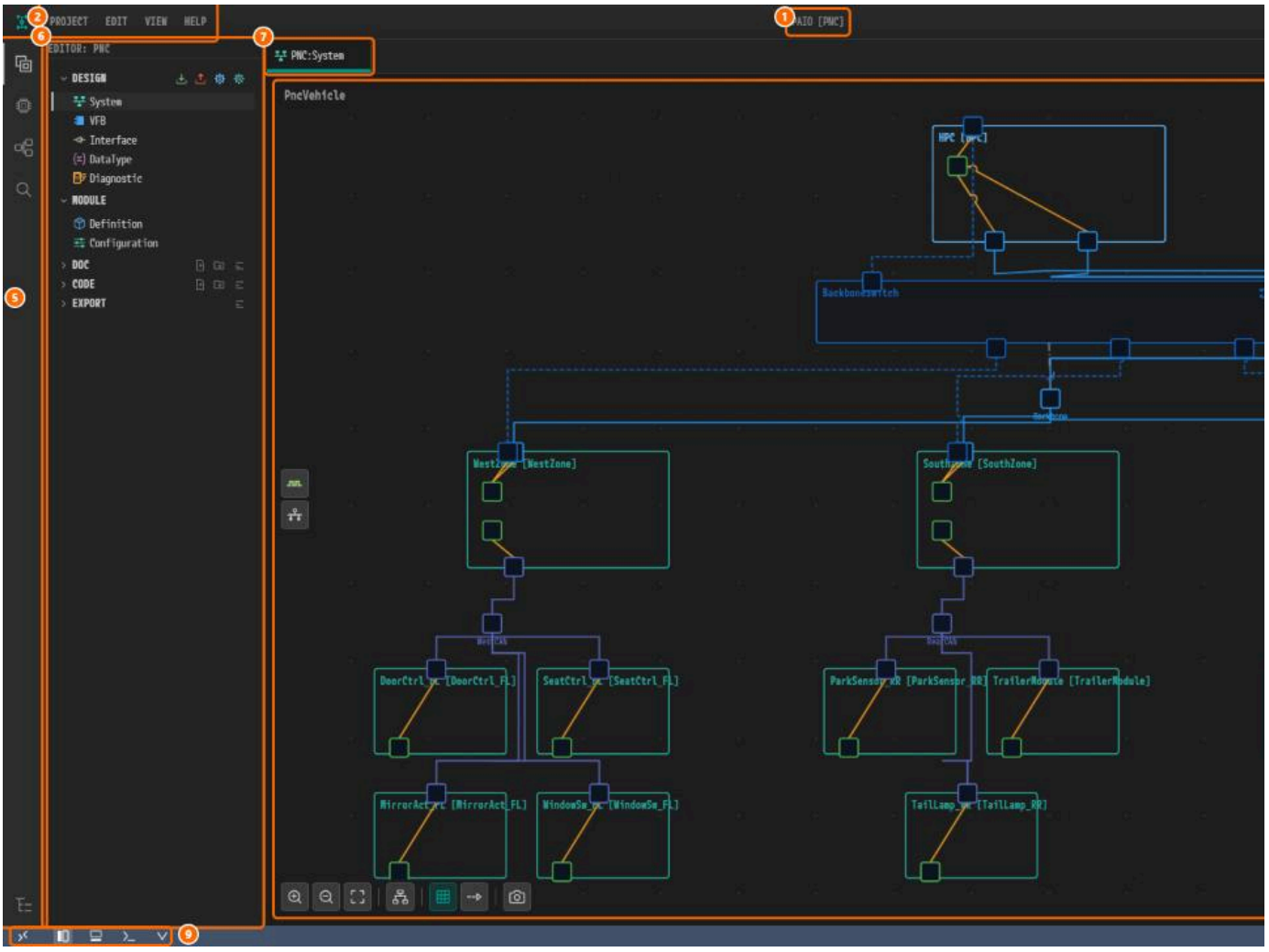
UI Layout and Navigation

What This Chapter Covers

This chapter explains what each region of the PAIO window is for: the header, the icon rail, the sidebar with the **EDITOR** view and the Explorer, the editor a view and the Meta Class search. It is written for users opening PAIO for the first time and for users looking for a feature on screen. There are no prerequisi

- Installation and sign-in: 03. Installation, Login and License
- How a folder becomes the project of a window: 04. Workspaces and Projects
- Shortcuts only: 18. Keyboard Shortcuts

One PAIO window holds one project. The folder a window opens is the project, and its name appears in the window title.



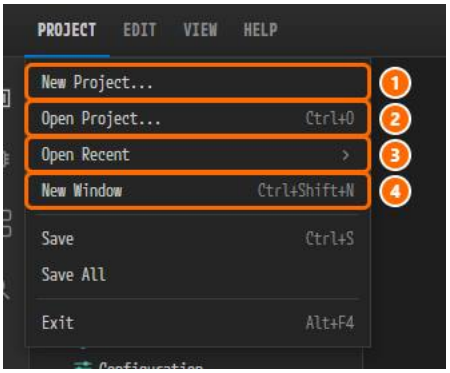
5.1 Header (Top Bar)

5.1.1 The Two Menus

PAIO has two menus. The menu bar drawn by the app (**PROJECT**, **EDIT**, **VIEW**, **HELP**) is on the left of the header. The native application menu is not visible or menu bar at the top of the screen. This chapter describes the menu drawn by the app.

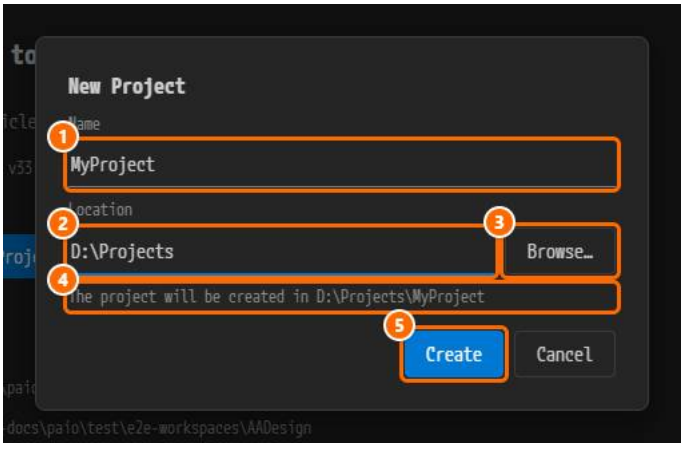
5.1.2 Project Menu

PROJECT creates, opens and saves projects, opens a new window (`Ctrl+Shift+N`), and exits.



To create a project:

1. Choose **New Project...**.
2. Type the project name into **Name**. Only letters, numbers, hyphens and underscores are allowed.
3. Type the parent folder into **Location**, or click **Browse...**.
4. Click **Create**. PAIO makes the folder `<Location>\<Name>` and opens it as the project of the window. If the window already holds a project, the new pr



- **Save** (`Ctrl+S`) saves the active editor; **Save All** saves every modified editor of the project. Both are blocked while the license is expired or read-only.
- When you close the window with unsaved changes, one `Unsaved Changes` dialog asks about all of them.

5.1.3 Edit Menu

EDIT holds **Undo** (`Ctrl+Z`), **Redo** (`Ctrl+Y`), **Cut**, **Copy**, **Paste**, **Select All** and **Go to Meta Class...** (`Ctrl+T`, 5.10). Undo and Redo act on the project history ar **Paste** and **Select All** act on text input only, not on canvas nodes or table rows.

5.1.4 View Menu

VIEW toggles the sidebar (`Ctrl+B`) and the Utility panel, zooms the window (5.7), toggles full screen (`F11`) and switches the **Theme** between **Dark** and **Li**

> The menu lists `Ctrl+J` next to **Toggle Panel**, but the `Ctrl+J` key toggles the **terminal panel**. Open the Utility panel from the menu or the status bar bu

5.1.5 Help Menu

HELP reopens the **Welcome** tab, opens the issue page (**Report Issue**), shows the version (**About PAIO**) and runs **Check for Updates**. Update results appear

5.1.6 The Right Side of the Header

From left to right: the theme toggle, the robot icon that opens the AI Chat dock, the user icon with **Profile**, **AI Account** and **Sign Out** (see 3.3), and the win icon is disabled with the tooltip `Plan expired`.

5.1.7 The Title in the Center of the Header

The title reads `PAIO` with no project open and `PAIO [<project>]` with one (for example `PAIO [PNC]`). The remote connection state is shown on the lef

5.2 Icon Rail and Sidebar (Left)

The icon rail on the left edge chooses what the sidebar shows. The **EDITOR** view opens the design editors; the Explorer shows the files of the project fold

5.2.1 Icon Rail

The rail icons have no text; hover to see the tooltip.

Position	Tooltip	Shows
1 (top)	`Design`	The **EDITOR** view (5.2.2), shown at start
2	`vECU`	The vECU view
3	`Workflow`	Opens the AI Workflow in a tab (21. AI Workflow)
4	`Search`	The Search view (5.8)
Bottom	`Explorer`	The Explorer (5.2.3)

Clicking the icon of the view already shown hides the sidebar; clicking it again shows it.

5.2.2 EDITOR View

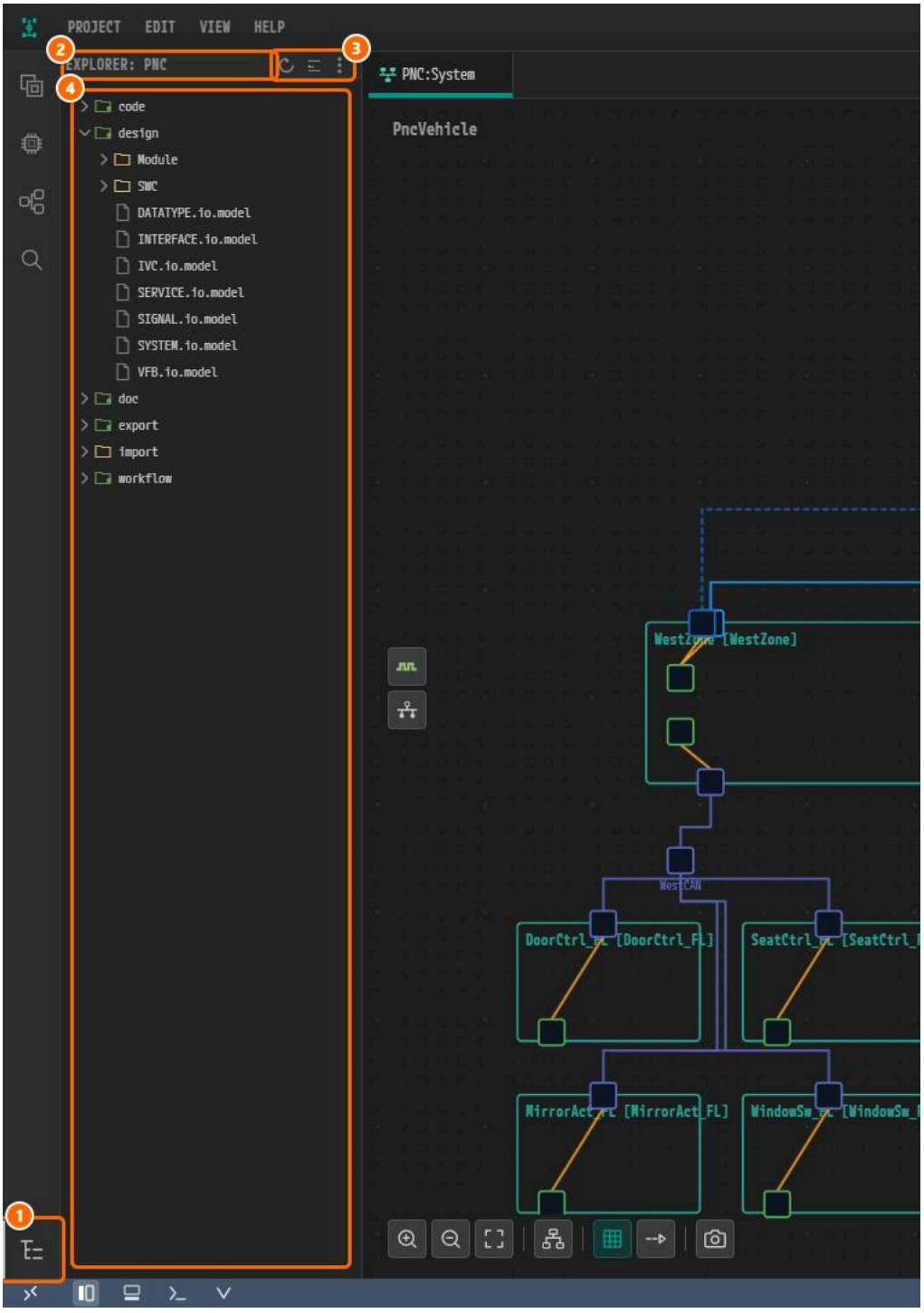
The header reads `EDITOR: <project>`. Below it are five sections: **DESIGN**, **MODULE**, **DOC**, **CODE** and **EXPORT**.

1. Click a row under **DESIGN** or **MODULE**, for example **System**. One click opens its editor, here the tab `PNC:System`.

The Signal editor has no row: it opens from the **Signal** icon on the System editor canvas (see 06. System Editor). The **Import** and **Export** icons on the **DESI**

5.2.3 Explorer

Click the **Explorer** icon at the bottom of the rail. The Explorer lists the project folder directly: `code`, `design`, `doc`, `export`, `import` and `workflow`.



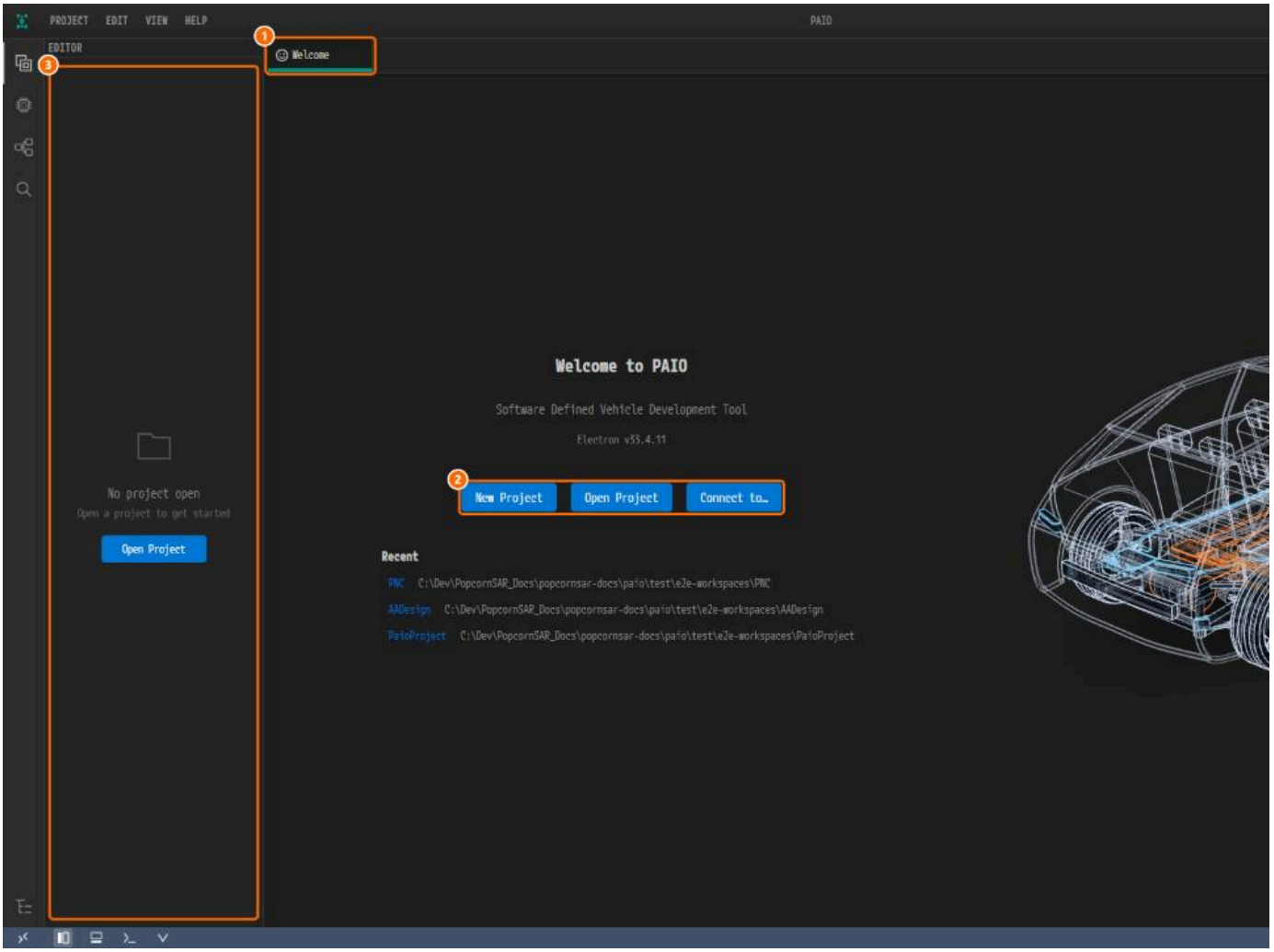
> **The Explorer does not open the design editors.** Double-clicking a `*.io.model` file opens it as read-only JSON (see 5.3.8). Edit the design from the **EDIT**

The header shows the project and the kind of connection (SSH, WSL, container). Folder operations are in 04. Workspaces and Projects, and the keys in 18.

5.3 Editor Area

5.3.1 Welcome Screen

The **Welcome** tab opens each time PAIO starts.



- **New Project** opens the **New Project** dialog (5.1.2).
- **Open Project** opens a folder as the project.
- **Connect to...** starts a remote connection (17. Remote Development (SSH, vECU Container)).
- The **Recent** list below shows recently opened projects.

To open the tab again, choose **HELP** > **Welcome**.

5.3.2 Tabs

Click a tab to activate it. Close it with × or a middle-click, right-click for the context menu, and drag to reorder, move or split. The title has the form `proj` tab carries a dirty marker (●). A tab is as wide as its name. When the tabs do not fit, the two chevron buttons at the right end (**Previous Tabs**, **Next Tabs**)

- At most 20 tabs can be open across all groups. Beyond that, no new tab opens and nothing is shown on screen.
- Opening the same file again moves to the tab already open.

5.3.3 Tab Context Menu

Right-click a tab for **Close**, **Close Others**, **Close Saved** and **Close All**. The last three act on the same group. Closing a tab with unsaved changes asks wheth

5.3.4 Split View

There is no menu item for splitting. You split by dragging a tab.

1. Drag a tab.
2. Drop it on the **right half of the editor body**. The group splits left and right, and the tab moves to the new right group. A rectangular marker with a ver

Dropping on the left half or on the tab bar moves the tab without splitting. Splitting is horizontal only. Drag the divider to change the ratio. A pane made
disappears by itself.

5.3.5 Editor Types

A tab opens in one of nine model editors (System, Signal, VFB, Interface, Data Type, Diagnostic, Module Definition, Module Config, Workflow) or nine file ec
image, HTML, diff, Welcome, Search) depending on the node type. Image files always open in the image viewer.

5.3.6 Save, Undo, and Redo

Each project has one edit history, and save, undo and redo all work through it.

- **EDIT ▸ Undo** reverts an edit made in any editor and moves to the editor that holds it.
- Saving marks a "no changes" point but does not clear the history, so you can keep undoing after a save.
- With several groups, Undo, Redo and Save from the menu run once, in the active group.

5.3.7 External Change Detection

When a file open in an editor changes on disk, PAIO reads it again. **It does not re-read while unsaved edits exist or an edit is in progress**, so your edits are
disk and that saving overwrites the external change.

5.3.8 XML / ARXML Editor

Opening an `.arxml`, `.xml` or `.dbc` file brings up the Monaco-based editor. It opens in **view mode**.

1. Click the eye icon to switch to edit mode. Save, undo, redo and format become enabled.
2. Edit the file. Typing `<` suggests the AUTOSAR child elements allowed at that point.
3. Check the PROBLEMS panel at the bottom. Validation runs when the file opens and after each change.

PAIO documents (`*.io.model` and others) opened from the Explorer are **read-only** here.

5.4 Bottom Panels

Below the editor area are three independent panels, Utility, TERMINAL and VALIDATION, each turned on and off separately. All three, and the AI Chat dock,
kept when you restart.

5.4.1 Utility Panel — History

The **History** tab shows the edit history of the project, newest first. Double-click a row to move to that point. The panel has no Problems tab: diagnostics
the VALIDATION panel.

5.4.2 TERMINAL Panel

The terminal panel adds, splits and kills terminals and changes the font size. Opening it with `Ctrl+J` or `Ctrl+`` creates a terminal if none exists. See 16. (

5.4.3 VALIDATION Panel

This panel lists AUTOSAR constraint violations in a table. Click a violation row to open the editor that holds the element and select it. See 14. Validation.

5.5 Status Bar (Bottom)

5.5.1 Left Side

- The remote indicator (`><`) is always at the far left. Once connected it gets a green background and shows the connection, for example `SSH: <host`
- Four buttons toggle the sidebar, the Utility panel, the terminal and the VALIDATION panel.
- The validation button carries a violation count badge. `!` means validation could not run.
- The **IntelliSense** item shows the state of the C/C++ language server, once one has started.

5.5.2 Right Side

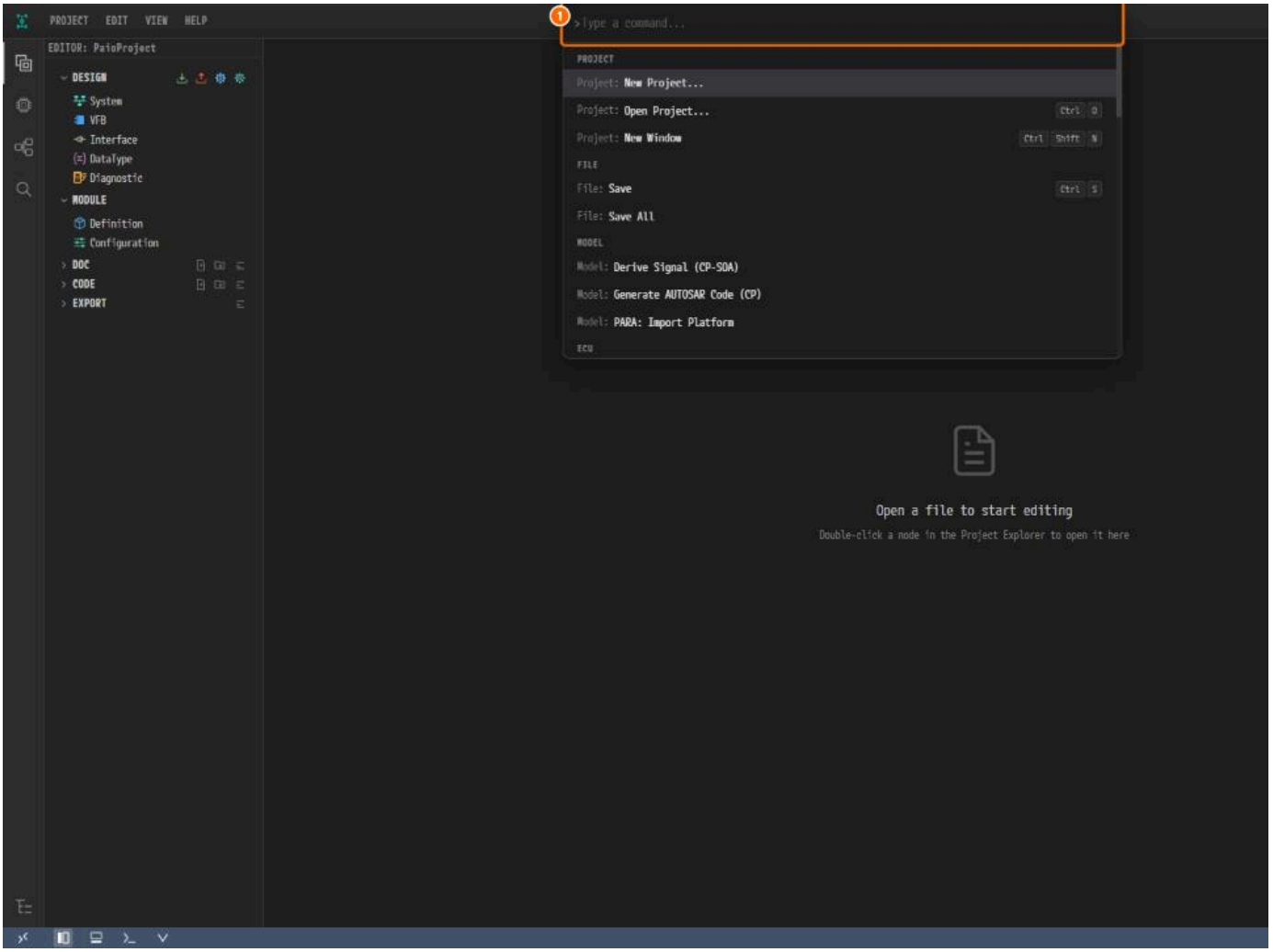
The right side shows the license (for example `Pro`), an update indicator when there is an update to report, and the version `v<version>`. Clicking the lice

5.6 Command Palette

1. Press `Ctrl+Shift+P`.
2. Type part of a command. The search is fuzzy: `tgt` matches `View: Toggle Terminal`.

3. Press `Enter`, or click the command.

Press `Escape` to close.



5.6.1 Registered Commands

There are 41 commands in eleven categories: Project, File, Model, ECU, Edit, View, Search, AI, Help, Remote-SSH and Code.

5.6.2 Commands That Do Nothing When Run

> Nine commands appear in the list but do nothing: **Cut**, **Copy**, **Paste**, **Select All**, **Toggle Sidebar**, **Toggle Panel**, **Toggle Full Screen**, **Toggle Terminal** and **Rep** bar buttons instead.

5.7 Screen Zoom

VIEW ▸ Zoom In (`Ctrl+=`), **Zoom Out** (`Ctrl+-`) and **Reset Zoom** (`Ctrl+0`) change the zoom of the whole window, 20% per step, up to eight steps either w kept on restart. Zooming the System and VFB canvases with `Ctrl` + wheel is separate (see 18.2 Graphical Editors (System · VFB)).

5.8 Search View

The Search view searches the saved files of the project's `design`, `doc` and `code` folders.

1. Click the **Search** icon on the rail, or press `Ctrl+Shift+F`.
2. Type the text. The search runs by itself shortly after you stop typing.
3. Click a result to go to it.

> **The search reads saved files only.** If tabs have unsaved changes, PAIO asks whether to save them first.

5.8.1 Search Box

Three toggles sit in the box: `Aa` (**Match Case**), `ab` (**Match Whole Word**) and `.*` (**Use Regular Expression**). `Enter` searches at once; `ArrowUp` / `ArrowDown`

5.8.2 Header Buttons

The header buttons refresh, clear, open a Search Editor (5.9), open the Meta Class search (5.10), switch between list and tree, and collapse or expand the

5.8.3 Results

Results come in three areas, **Design**, **Doc** and **Code**, each with a match count. **Design** groups hits by the editor that opens them; **Doc** and **Code** list files ar chips that narrow the results.

5.8.4 Opening a Result

Click a result row. A Doc or Code line opens the file with the match selected. A Design hit opens the editor that owns the value and marks the field. `F4` an

5.8.5 Find Usages

Right-click an element row in **Design** and choose `Find Usages of '<name>'`. The results list the design references that point to that element. Click `×` or

5.9 Search Editor

A Search Editor is a search of its own in an editor tab. Open one with **Open New Search Editor** in the Search view header (it takes the view's query) or in th be open at once, each with its own query. The tab is read-only.

5.10 Meta Class Search (Ctrl+T)

The Meta Class search answers one question: where in PAIO is this AUTOSAR meta class designed? Type a meta class name such as `ServiceInterface` or element of your project.

1. Press `Ctrl+T`, choose **EDIT ▸ Go to Meta Class...**, click its icon in the Search view header, or run **Go to Meta Class...** from the command palette.
2. Type the meta class name. A space or `-` separates words, so `service interface` and `SERVICE-INTERFACE` both find `ServiceInterface`.
3. Pick a line with `ArrowUp` / `ArrowDown` and press `Enter`, or click it.

Each line is one meta class and one place that designs it, for example `ServiceInterface Designed in Interface › Service`. A class designed in sever opens that editor and goes to the first element of that class the project already has. If the project has none there yet, PAIO marks the section or the but

Below the list, the selected line shows what must exist before the class can be added (`Needs first:`) and what you fill in after adding it (`Fill in afte` tells whether the project already has one.

> **To find the elements of your project, use the Search view (5.8).** The Meta Class search lists meta classes only. In the Markdown editor and on the Doc s new document).

Editors

One page per model editor, in the order you normally use them.

Design editors open from the **EDITOR** view on the left of the window. Click a row under **DESIGN** (**System**, **VFB**, **Interface**, **DataType**, **Diagnostic**) or **MODULE** (**DESIGN** row opens in a tab named after the project and the row, for example `<project>:System`. The Signal editor has no row: open it with **Signal**, the to editor canvas. The SYSTEM panel opens from the **System** button (the PAIO logo) on the right edge of the System and VFB canvases. The Diagnostic editor

Editor	What you define there
System Editor	ECU and HPC nodes, communication clusters, connections. Opens from **DESIGN** ▸ **System** (tab `<project>:System`); formerly the IVC editor
Signal Editor	Signals and PDUs, bit layout, initial values. Opens from the **Signal** icon on the System editor canvas
VFB Editor	Software component compositions, ports, connections
SYSTEM Panel	System **NAME**, **VERSION** and **DESCRIPTION**; **ECU MAPPING** in the System editor; **ROOT COMPOSITION** and **COMPOSITIONS** in the VFB editor
Interface Editor	Port interfaces, data elements, operations
DataType Editor	Application and implementation data types, units
Diagnostic Editor	Diagnostic extract (DEXT): sessions, security, DIDs, routines, events and DTCs, and what each ECU or HPC supports. Opens from **DESIGN** ▸ **Diagnostic** (tab `<`
ECUC	ECU configuration modules, containers, parameters. Opens from **MODULE** ▸ **Definition** and **MODULE** ▸ **Configuration**

System Editor

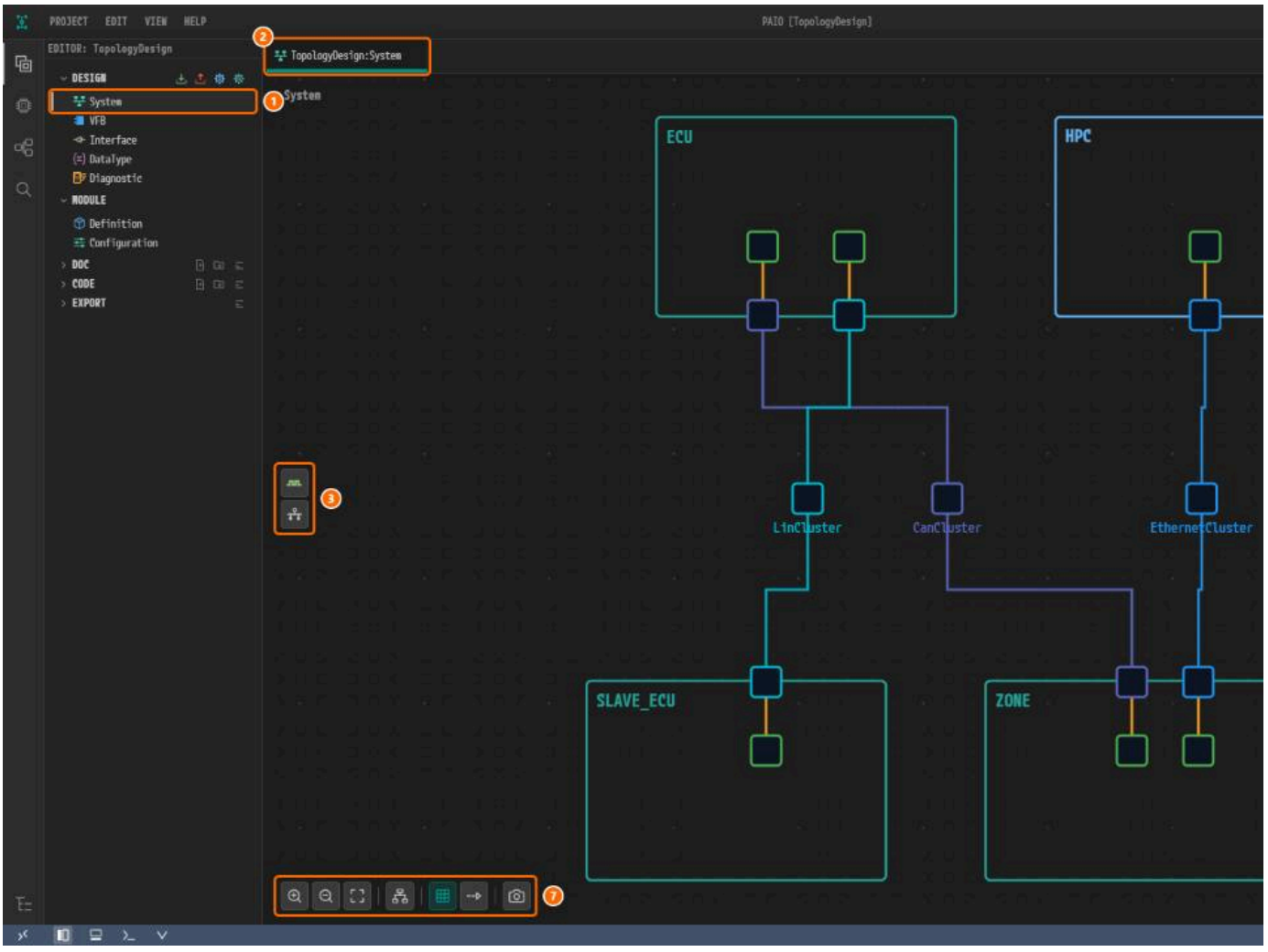
What This Chapter Covers

The System editor is where you design the in-vehicle communication topology as a diagram. You place ECU and HPC nodes with their connectors and cor and physical channels, and cable the ports of an Ethernet switch (Coupling Element) to each other. Term definitions are in 19. Glossary.

- Audience: users who edit AUTOSAR network topology directly
- Prerequisites: the project's window is open (its title bar reads `PAIO [<project>]`)

6.1 Opening the Editor and Save Locations

1. In the **EDITOR** view on the left, click **System** under **DESIGN**. The tab `<project>:System` opens with what you saved before.
2. Save with `Ctrl+S`. The content goes to `design/IVC.io.model`.



*① **System** under **DESIGN**. ② The tab `TopologyDesign:System`. ③ The action strip: **Signal** and **Partial Network (PNC)**. ④ The property panel toggle. ⑤ The l toolbar.*

- The label at the top left of the canvas is the system **NAME**, not the editor name.
- **Signal** in the action strip opens the Signal editor. **Partial Network (PNC)** opens the PNC dialog (6.14).
- **System** at the right edge opens the SYSTEM panel (11. SYSTEM Panel).

Node positions and edge routes are saved too, but they are not included in an ARXML export.

6.2 Elements You Can Work With

The editor places eight kinds of element: **ECU**, **HPC**, **Connector**, **Controller**, **Ethernet Switch** (Coupling Element), **Coupling Port**, **Communication Cluster** and the rectangles inside a node, connectors sit on the node boundary, and a cluster is the line joining connectors with a rectangle in its middle.

> **FlexRay elements do not appear in this editor.** FlexRay elements from an import are filtered out before the canvas is drawn, and saving rebuilds the co editor in a project that contains FlexRay, those elements disappear from the project.

6.3 Canvas Controls

The basic controls for moving the canvas and selecting and moving elements.

6.3.1 Mouse

- Drag on empty space to pan, and use `Ctrl` + wheel to zoom.
- Click a node to select it, and `Ctrl` + click to add to the selection.
- Drag a node to move it. `Shift` + drag connects (6.5.5).
- Double-click a node to open its configuration dialog (6.7).

6.3.2 Keyboard

Shortcuts work only while the editor has focus.

Key	Behavior
`Delete` / `Backspace`	Delete the selection, after a confirmation
`Ctrl+Z` / `Ctrl+Y`	Undo / Redo
`Ctrl+A`	Select all
Arrow keys	Move the selected nodes by 1px (`Shift`: 20px)
`Ctrl+S`	Save

`Ctrl+C` / `Ctrl+V` / `Ctrl+X` have no effect.

6.3.3 Toolbar (Bottom Left of the Canvas)

The toolbar (⑦ in the figure in 6.1) holds **Zoom In**, **Zoom Out**, **Fit to View**, **Auto Layout**, **Toggle Grid**, **Toggle Connector Animation** and **Export as PNG**. **Export** `<project>/doc/capture/` without a file dialog.

6.4 Creating Nodes

You create nodes from the right-click menus. Every menu ends with **Delete**, and every delete asks for confirmation first.

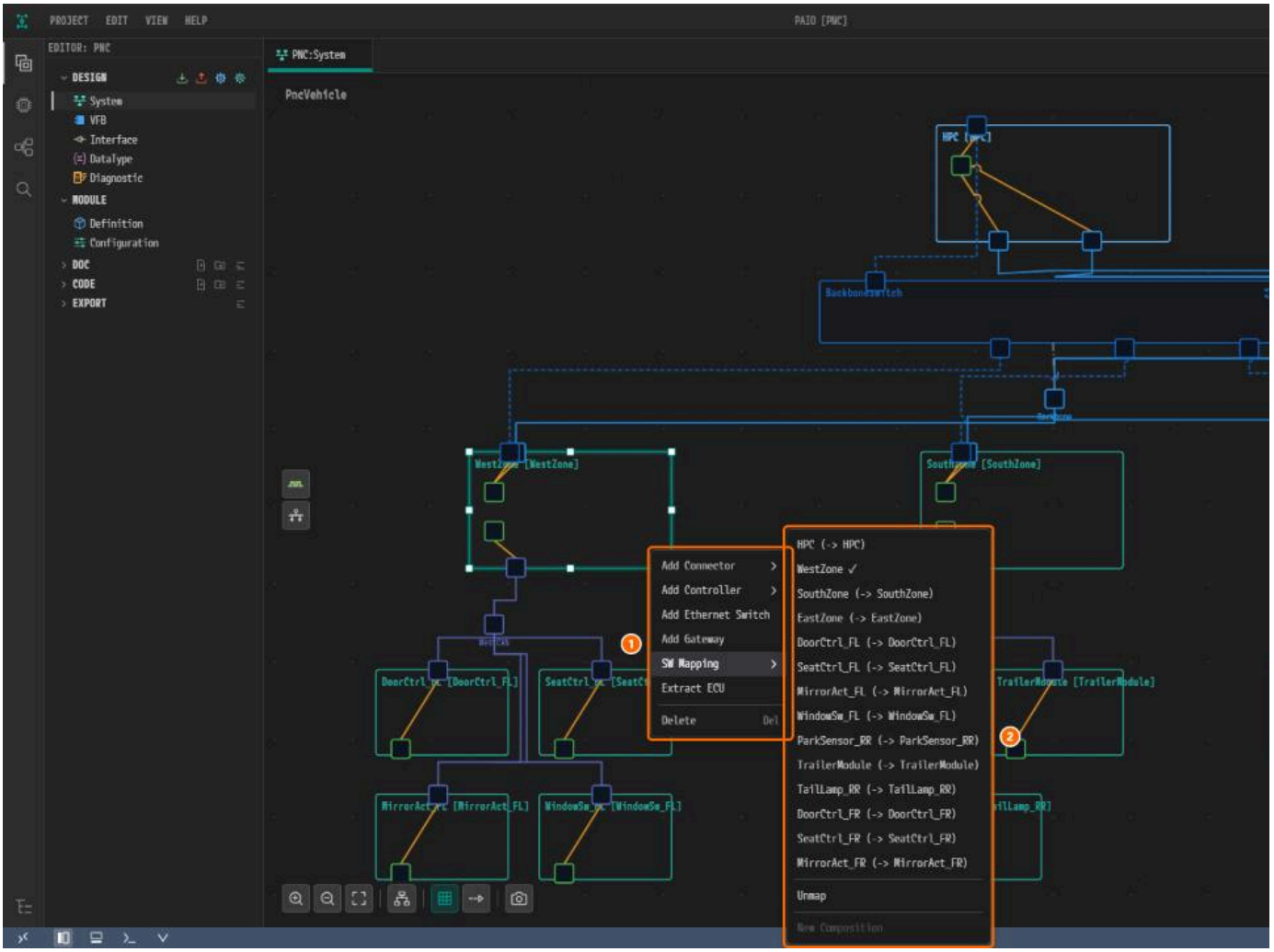
6.4.1 Right-Clicking the Empty Canvas

Right-click the empty canvas and choose **Add ECU** or **Add HPC**. The node is added at the click point. A switch is added from the menu of an ECU or HPC nc

6.4.2 Right-Clicking an ECU or HPC Node

Right-click an ECU or HPC node to add what goes inside it.

- **Add Connector** and **Add Controller**: choose Ethernet, CAN or LIN.
- **Add Ethernet Switch** and **Add Gateway** / **Remove Gateway**.
- **SW Mapping**: map a VFB composition to the node (6.6).
- **Extract ECU** / **Extract Machine**: extract this node alone to ARXML (13. Import and Export).



*① The node menu. ② The **SW Mapping** submenu: `WestZone ✓` is the composition mapped to this node.*

Deleting a node also deletes everything inside it. `Ctrl+Z` restores it.

6.4.3 Right-Clicking an Ethernet Switch (Coupling Element) Node

Add CouplingPort adds a switch port.

6.4.4 Right-Clicking a Controller Node

On an Ethernet controller, **Add Coupling Port** adds a port owned by the controller. An Ethernet controller can own only one port. A CAN or LIN controller sh

6.4.5 Right-Clicking a Coupling Port Node

Connect to Port... picks a peer port and connects it with a physical cable (6.5.4).

6.4.6 Default Names Assigned Automatically

A new node gets a numbered name starting at 0, such as `ECU_0` or `DoorEcu_Can_Connector_0`. Change it in the **NAME** field of the properties panel (6.12)

6.5 Creating Connections

How to connect connectors, clusters, controllers and ports, and how to remove the connections.

6.5.1 From a Connector to a Cluster (Starting on the Connector Side)

A connector not yet connected to a cluster carries a green **[+] badge**.

1. Click the connector's **[+] badge**.
2. Choose **Create CommunicationCluster [...]** to create a new cluster of the same protocol, or pick a cluster ▸ physical channel under **Connect to Chanr**

3. A line is drawn between the connector and the cluster, and the badge disappears.

Right-clicking a connector also shows **Connect to Channel...**, even when it is already connected.

6.5.2 From a Cluster to a Connector (Starting on the Cluster Side)

Right-click a cluster and choose **Add Connector to Channel...** (or **Add Connector...** when there is no channel). Only unconnected connectors of the same physical channel; Ethernet has no limit.

6.5.3 Joining a Controller and a Connector (Internal Wiring)

Internal wiring joins a controller and a connector inside the same ECU, with matching protocols.

- Select the connector and pick a controller in the **Controller** field of the properties panel, or
- `Shift` + drag from the controller to the connector.

6.5.4 Joining Two Ports (Physical Cable)

1. Right-click a Coupling Port node.
2. Pick the peer port under **Connect to Port...**. A line appears between the two ports.

A cable is point-to-point, so ports that are already connected are not listed.

6.5.5 Connecting Directly with `Shift` + Drag

Hold `Shift` and drag from a **Connector**, **Controller**, **Cluster** or **Coupling Port** to the target. The cursor shows whether the target can be connected.

> If you drop on a target that does not meet the condition, **nothing happens and no message appears.** Check the protocol and the owning ECU first.

6.5.6 Removing a Connection

- Connector - cluster: right-click the edge ▸ **Disconnect**, or clear **Connected Channel** in the connector's properties panel.
- Controller - connector: right-click the edge ▸ **Disconnect**.
- Port - port: right-click the edge ▸ **Delete Connection**.

The connector's menus have no **Disconnect** item.

6.5.7 Edge Right-Click Menus and Route Editing

Select an edge and drag its route point handles to adjust the wiring by hand. **To Step Edge** returns to the automatic route, and **To Straight Edge** makes a port edges support route editing.

6.6 Composition Mapping (SWC - ECU)

Place a VFB composition on an ECU or HPC node.

1. Right-click the node and open **SW Mapping** (see the figure in 6.4.2).
2. Choose a composition, or **New Composition** to create one with the node's name.

Mapping is 1:1. Choosing the checked item removes the mapping. The mapping is written to `design/SYSTEM.io.model` and shown in the **ECU MAPPING** list covered in 08. VFB Editor.

6.7 Configuration Dialogs

Double-click a node to open the configuration dialog for its kind.

6.7.1 Which Node Opens Which Dialog

ECU, HPC, cluster and Ethernet switch each open their own dialog. A Coupling Port opens its switch's dialog, or its own dialog when a controller owns it. C them in the properties panel.

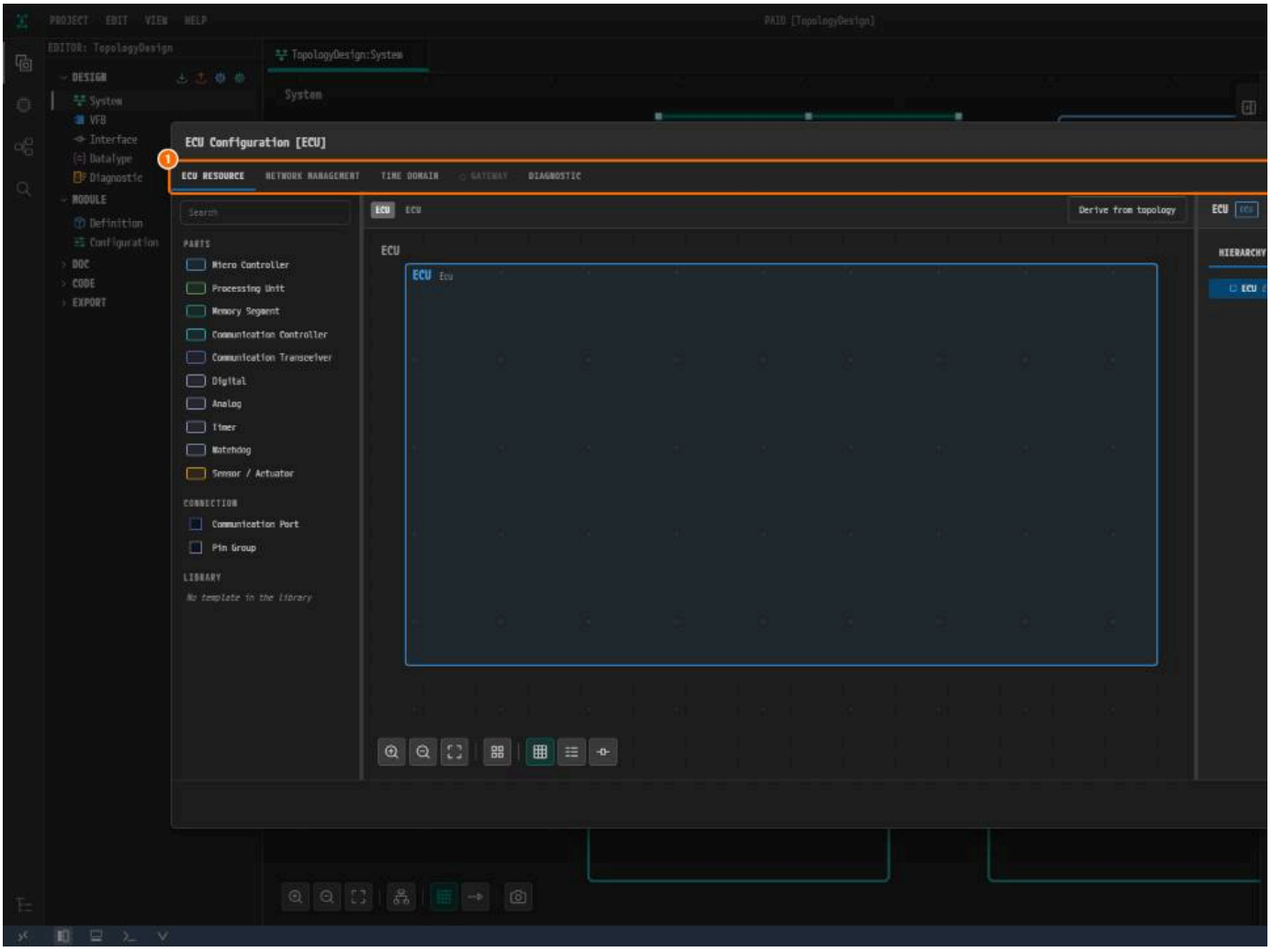
6.7.2 Rules Common to Every Configuration Dialog

- Edits are made on a working copy. Save with `Ctrl+S`; the dialog has only a **Close** button.
- `Ctrl+Z` first rewinds the dialog's own history.

- The **CLUSTER** tab of the cluster dialog and **Route** on the **GATEWAY** tab change the project at once.
- A dot (●) before the title means unsaved changes.

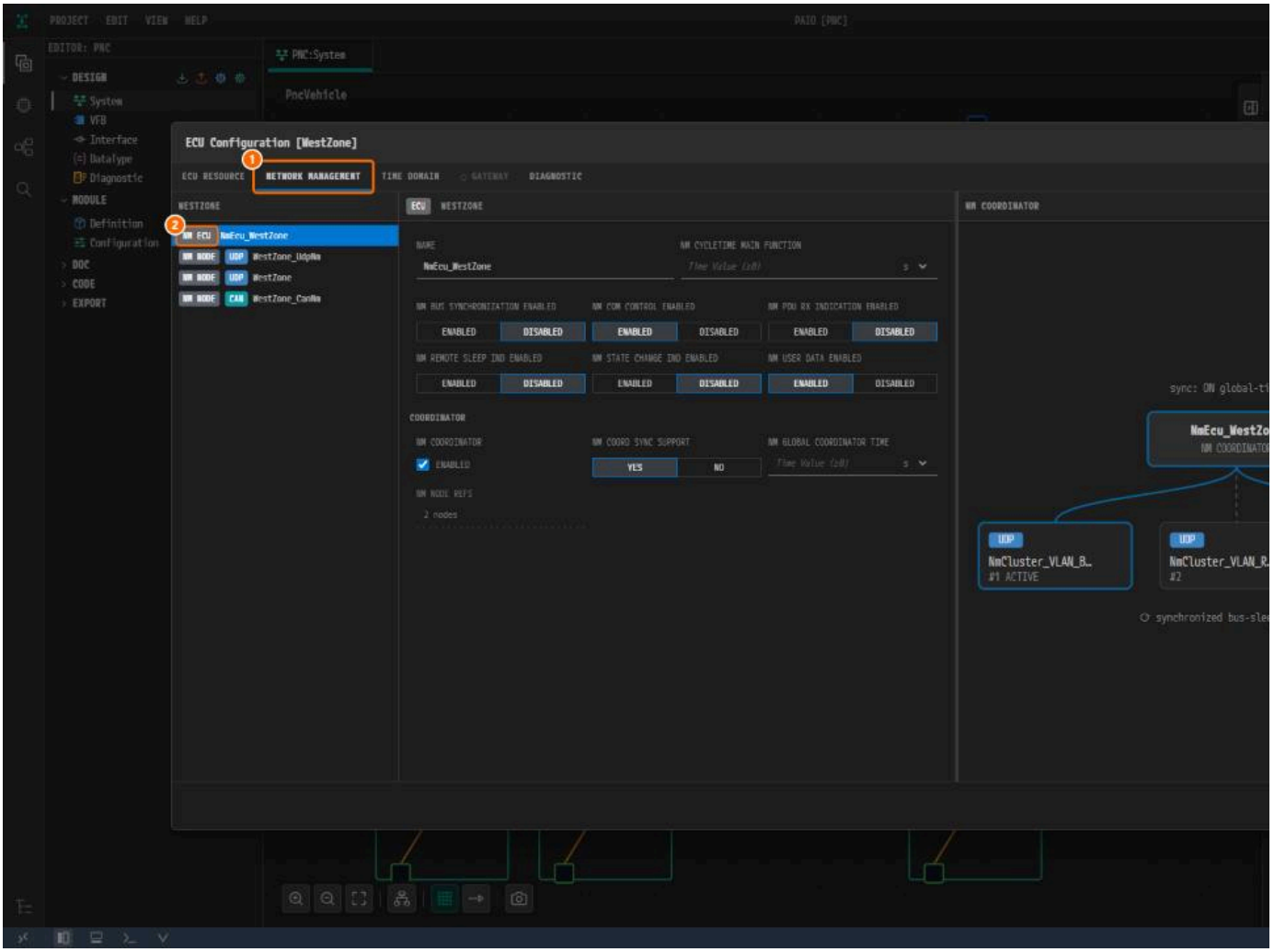
6.7.3 Node Configuration (ECU / HPC)

The ECU dialog has the tabs **ECU RESOURCE**, **NETWORK MANAGEMENT**, **TIME DOMAIN**, **GATEWAY** and **DIAGNOSTIC**. An HPC has no **GATEWAY**.



*① The tab bar. **GATEWAY** is dimmed while the node has no gateway.*

The **NETWORK MANAGEMENT** list is generated from the clusters that have NM enabled; you do not create it yourself.

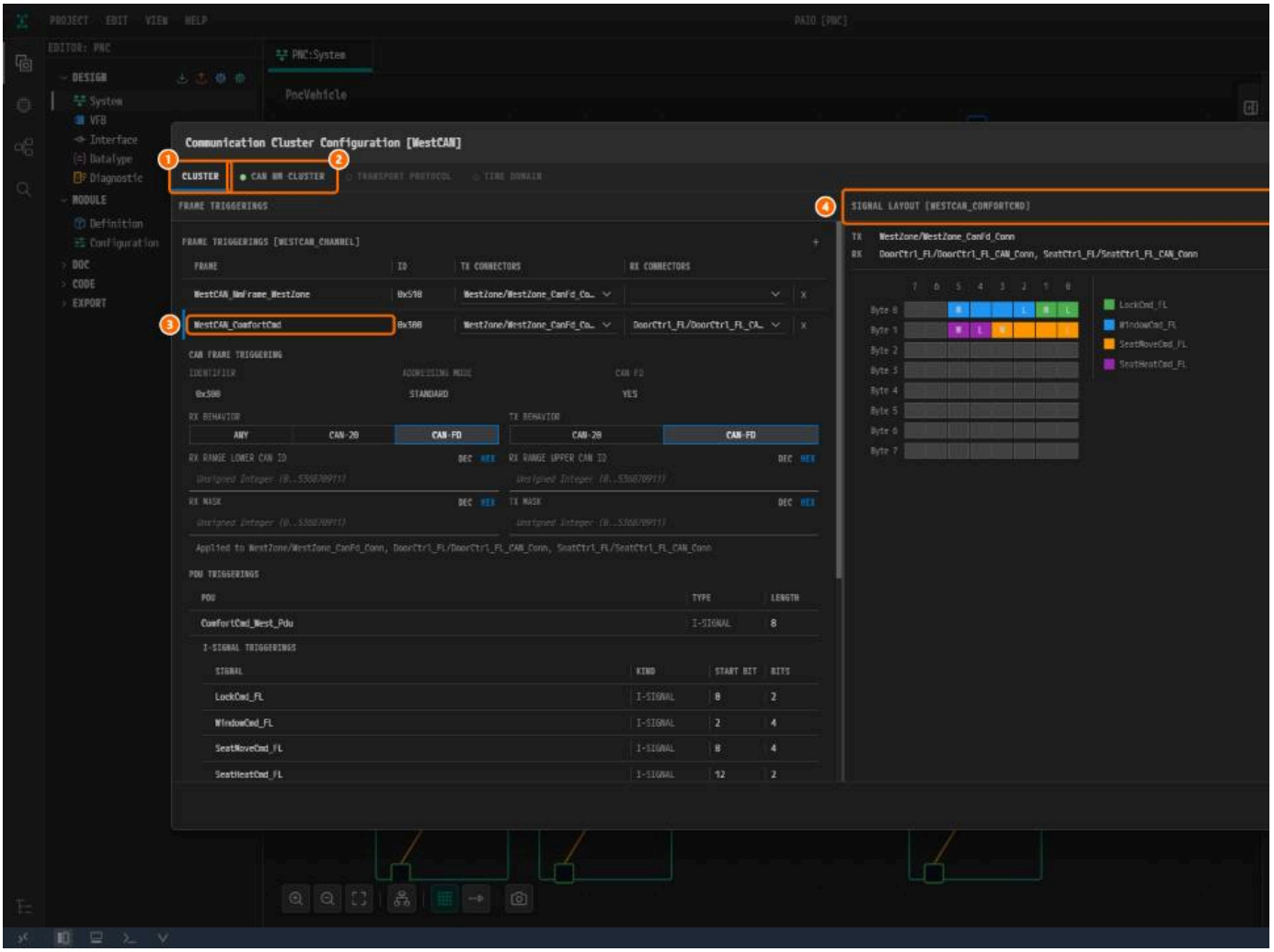


*① The **NETWORK MANAGEMENT** tab. ② The `NM ECU` item `NmEcu_WestZone`, with the `NM NODE` items below it.*

To enable **GATEWAY**, use **Add Gateway** in the node menu, or right-click the tab and choose **Enable**.

6.7.4 Cluster Configuration

The first tab is always **CLUSTER**. The other tabs depend on the protocol: an NM tab (CAN, Ethernet), **TRANSPORT PROTOCOL** (CAN, LIN) and **TIME DOMAIN** (C



*① The **CLUSTER** tab. ② The **CAN NM CLUSTER** tab, enabled. ③ The frame row `WestCAN_ComfortCmd`, expanded. ④ **SIGNAL LAYOUT** for that frame.*

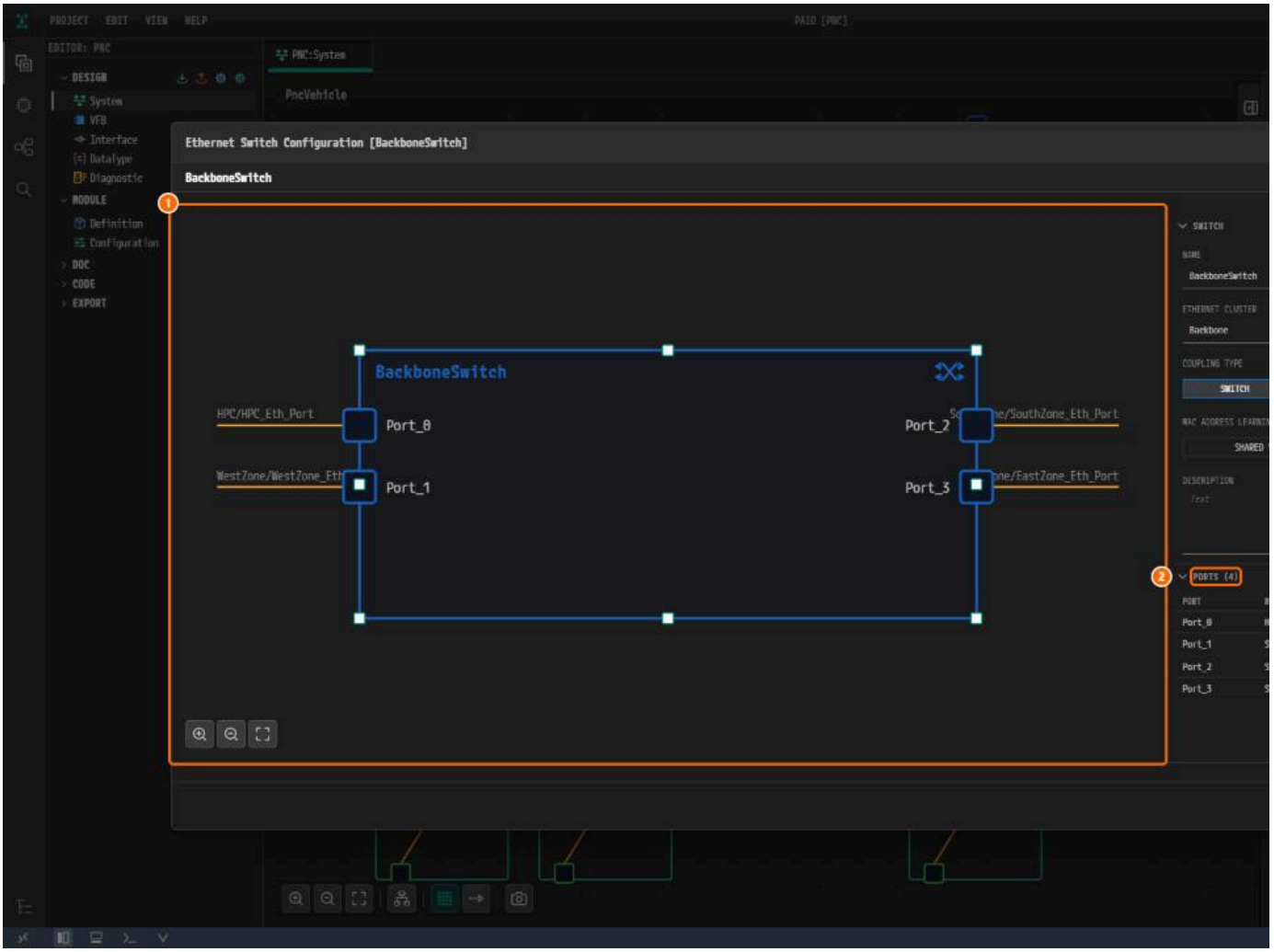
To add a frame to a channel on the **CLUSTER** tab:

1. Click **+** in the header of the channel's table and pick a frame or PDU from the Signal editor.
2. Pick the connectors in the **TX CONNECTORS** and **RX CONNECTORS** cells.

Each edit is applied to the project at once. NM, **TRANSPORT PROTOCOL** and **TIME DOMAIN** are off in a new cluster: right-click the tab and choose **Enable**.

6.7.5 Ethernet Switch Configuration

The dialog has two levels: the switch faceplate, then the pipeline of one port. Double-click a port to go down, and use the breadcrumb (`Switch > Port`



*① The faceplate with `Port_0` to `Port_3`. With no port selected, the right side shows the switch itself. ② **PORTS (4)**.*

- Click a port on the faceplate to see its details on the right. Errors are listed on the bar at the bottom.
- On the port level, click a block (**PORT**, **PRIORITY REGENERATION**, **TRAFFIC CLASS ASSIGNMENT**, **EGRESS**) to edit it.
- In the EGRESS editor, right-click the canvas to add FIFOs, schedulers or a preset.

An ATS shaper cannot be created through the UI.

6.7.6 Node DIAGNOSTIC Tab

The **DIAGNOSTIC** tab holds the diagnostic addressing of the node, in hexadecimal. **ADDRESS** is shown for ECU and HPC, **CONNECTIONS** for an ECU, and **DOIP**

6.7.7 Node GATEWAY Tab

The **GATEWAY** tab of an ECU edits the routes the gateway relays from one bus to another. The left side has the **FRAME**, **PDU** and **SIGNAL** sections, and the

1. Click a received item on the **ROUTING MAP** and then a destination that lights up, or click **+** on a section. The **ROUTE** sheet opens.
2. Check **SOURCE** and **DESTINATION**.
3. Click **Route**. The route is written as one undo step.

6.7.8 ECU RESOURCE Tab

The **ECU RESOURCE** tab describes the hardware of the node.

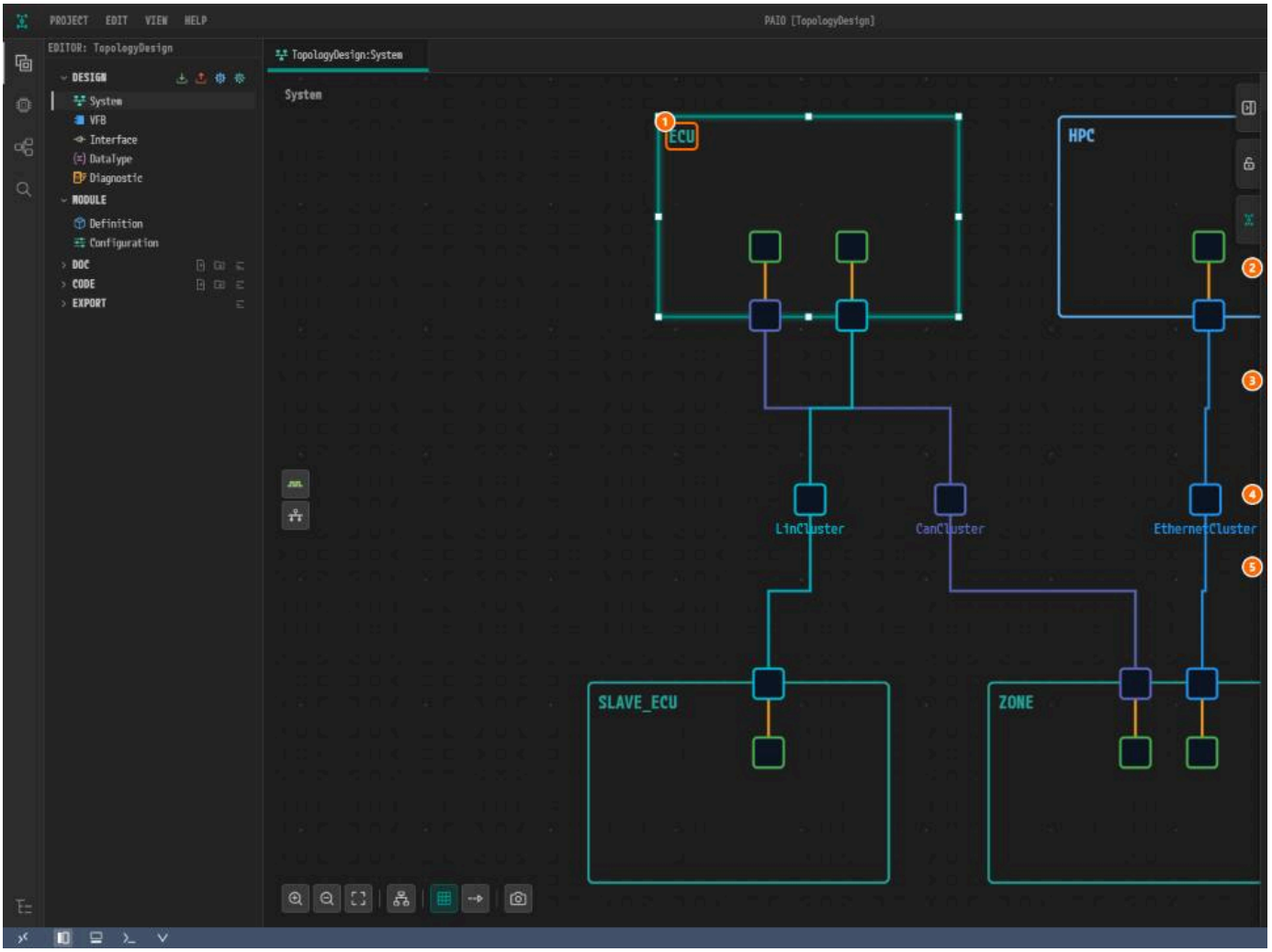
- Drag a part from **PARTS** or **CONNECTION** in the palette into the node box.
- **Derive from topology** fills the box from the System topology in one step.

6.8 Properties Panel (Right)

Select a node or an edge and its items appear as accordions in the properties panel on the right.

6.8.1 Layout per Node Kind

An ECU shows **NAME**, **DESCRIPTION**, **CONNECTORS**, **CONTROLLERS**, **POWER MANAGEMENT**, **PARTIAL NETWORK** and **COM MAIN FUNCTION**. A cluster shows its pro



*① The selected ECU. ② **CONNECTORS**. ③ **CONTROLLERS**. ④ **POWER MANAGEMENT**. ⑤ **PARTIAL NETWORK**.*

6.8.2 When a Connector Is Selected

A connector shows Name, Controller, Connected Channel and Description, plus sections for its protocol (for example **TX Frames** / **RX Frames**). The frame a in the section header to add one.

6.8.3 When a Controller Is Selected

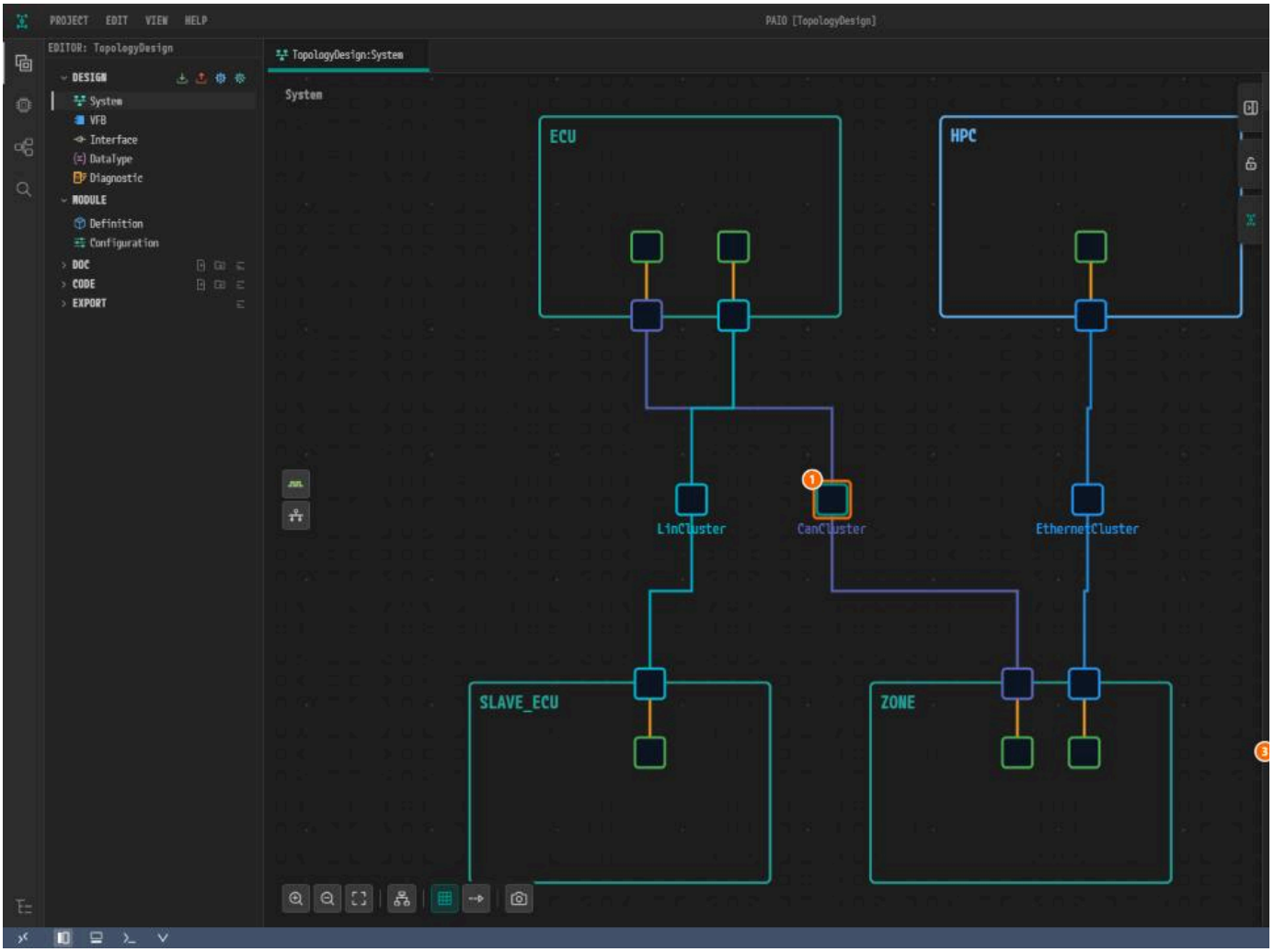
A controller shows Name and Description, then sections for its kind (Ethernet, CAN, LIN Master, LIN Slave).

6.8.4 When an Edge Is Selected

A port-to-port edge shows **First Port** and **Second Port**, read-only. Other edges show nothing.

6.8.5 Editing Physical Channels

Select a cluster and edit its channels in **PHYSICAL CHANNELS**: the channel name, the VLAN ID (Ethernet, 1 to 4094 or empty) and the idle timeout (LIN).



*① The selected cluster. ② **BAUDRATE (ARBITRATION, KBPS)** set to `500`. ③ `Connected Connectors (2)` under the channel.*

To add a connector to a channel:

1. Click **Add Connector** to the right of the `Connected Connectors` line.
2. Pick a connector in the `Select Connector` popup and click **Add**.

6.8.6 LIN Schedule Tables

Select a LIN cluster and edit its schedule tables in **SCHEDULE TABLES**. **+** adds a table, and `Entries (n)` holds its entries.

6.9 ECUC Generation

The System editor has no ECUC section. The BSW module configuration of an ECU is generated from this editor's topology with **PRTE AUTO CONFIGURATIO** **Auto Configuration** icon in the Module Configuration editor (12.2.6). See 12. ECUC.

6.10 Automatic Layout

1. Click **Auto Layout** on the toolbar.
2. If edge routes were adjusted by hand, a confirmation appears. Click **Replace** to continue.
3. When it finishes, the view fits every node.

A single undo reverts the whole layout.

6.11 Interaction with Other Editors

- The composition mapping appears in the SYSTEM panel and the VFB editor.
- The frame and PDU lists of a connector follow the Signal editor.

- Renaming or deleting an ECU or HPC also updates its name in other content, in the same undo step.

6.12 Naming Rules

A name starts with a letter, uses only letters, digits and underscores, and is at most 128 characters. A name that breaks the rule is not saved and returns t

6.13 Items Whose Display and Behavior Differ

If the file changes outside while you have unsaved edits, it is not reloaded and a warning appears. Save to keep your edits, or close the tab and open it a

6.14 Partial Network (PNC) Configuration

The **Partial Network (PNC)** icon in the action strip opens the **Partial Network (PNC) Configuration** dialog. It edits the partial networks of the whole system

1. Click **+** in **PARTIAL NETWORKS** on the left to add a PNC.
2. In the centre pane, check the channels that take part in the PNC under **PHYSICAL CHANNELS**.
3. Save with `Ctrl+S`.

Signal Editor

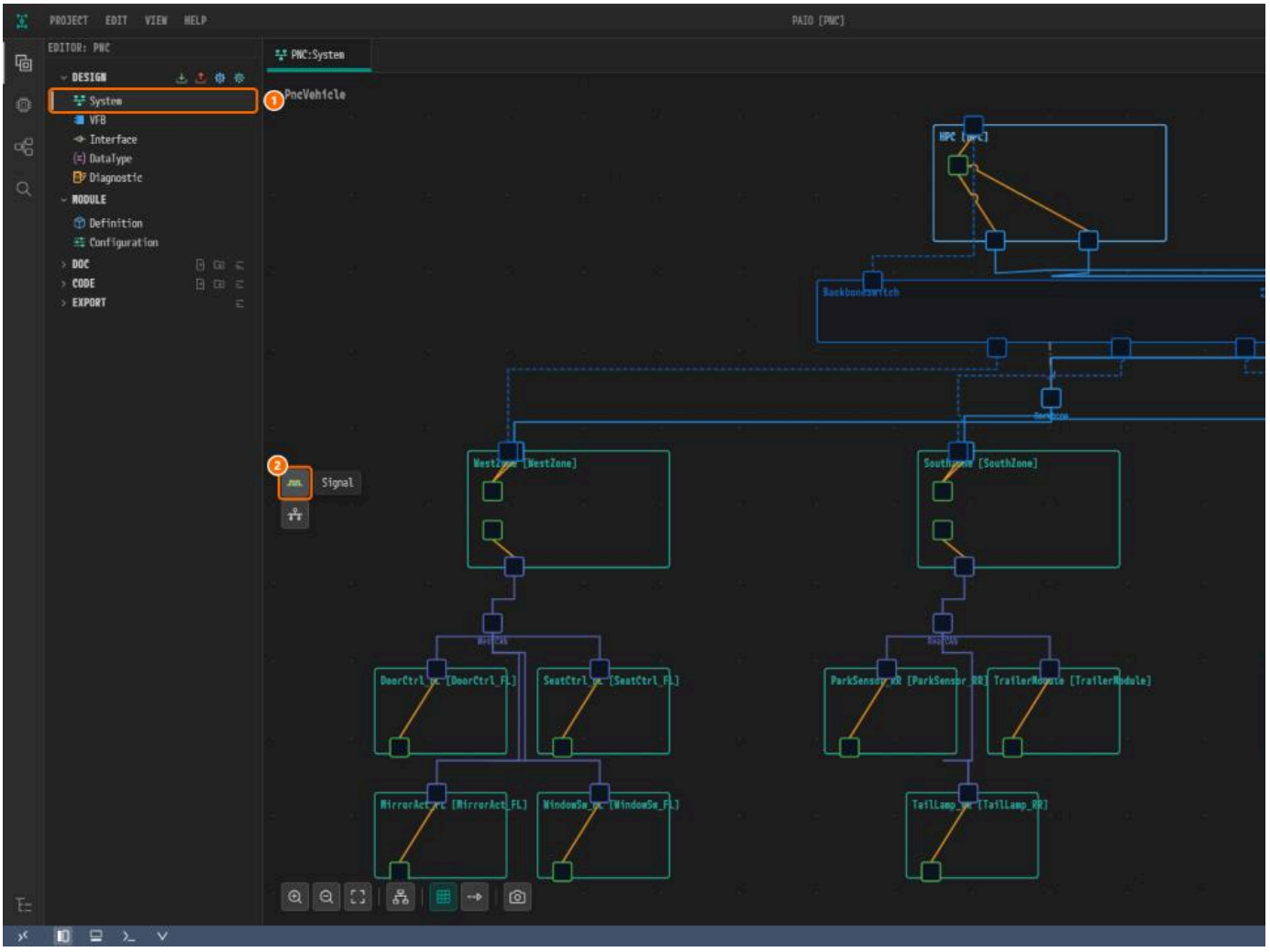
What This Chapter Covers

This chapter explains how to use the Signal editor, which edits the communication data dictionary: CAN and LIN frames, the six PDU kinds, I-Signals and s export procedures are in 13. Import and Export.

- Audience: users who write a bus communication matrix, or review and edit an imported one
- Prerequisites: a project is open in PAIO

The Signal editor has no row under **DESIGN** in the **EDITOR** view. You open it from the System editor.

1. In the **EDITOR** view, click **System** under **DESIGN**. The tab `<project>:System` opens.
2. Click **Signal**, the top icon of the strip at the left edge of the canvas. The tab `<project>:Signal` opens.

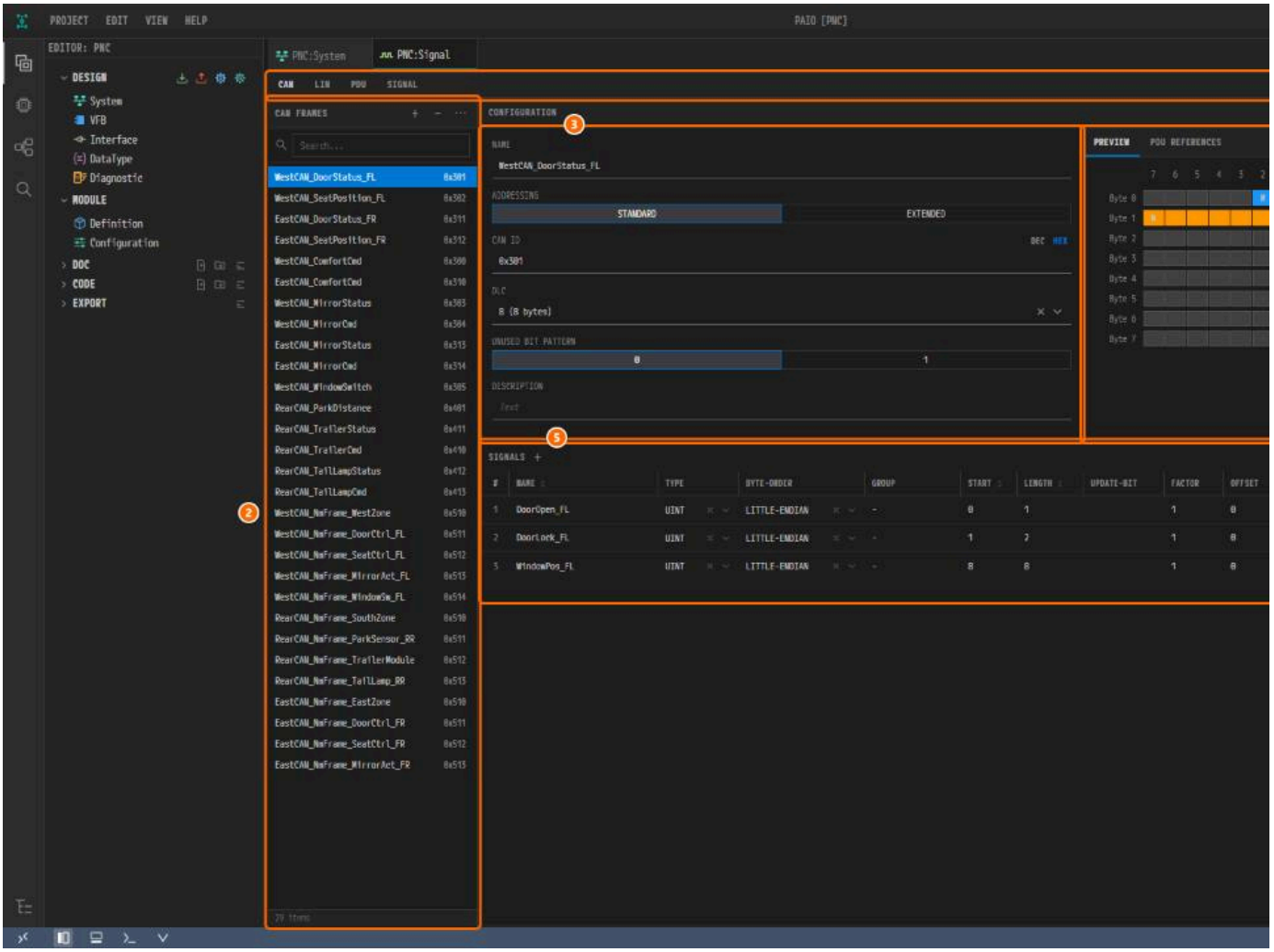


*① **System** under **DESIGN**, ② the **Signal** icon.*

The editor saves to one file, `design/SIGNAL.io.model`. Terms that are easy to confuse, such as I-Signal and System Signal, are in 19. Glossary.

7.1 Screen Layout

The protocol tabs run across the top. Below them are the list panel on the left and the Configuration panel on the rest of the width.



*① Protocol tabs, ② the **CAN FRAMES** list, ③ the Configuration form, ④ the **PREVIEW** and **PDU REFERENCES** tabs, ⑤ the **SIGNALS** table.*

- The bit layout is the **PREVIEW** tab inside the Configuration form. There is no separate Preview panel.
- The **SIGNALS** table appears below the form for frames, I-Signal I-PDUs and NM-PDUs.
- The **SIGNAL GROUPS** panel appears on the right of the SIGNAL tab only.

7.2 Protocol Tabs

Tab	What you edit
CAN	CAN and CAN FD frames
LIN	LIN frames
PDU	All six PDU kinds, including the NM-PDU, in one list
SIGNAL	The signal (I-SIGNAL) dictionary

There is no FlexRay tab and no Ethernet tab. Imported FlexRay and Ethernet data is preserved in the file and included in the export.

7.3 The Left List Panel

The list shows the items of the current tab.

Header Buttons

+ adds an item, **−** deletes the selected items after a confirmation, and **…** opens the filter and sort popover (7.4).

Selecting Several Items

`Ctrl`+click adds an item to the selection. With the list focused, `Ctrl+A` selects every item.

Search and the Status Bar

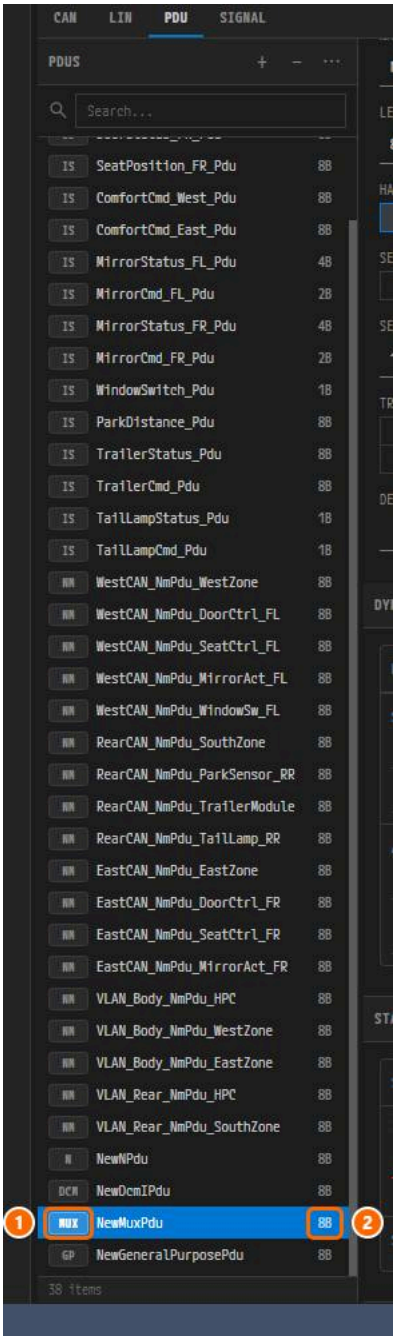
The `Search...` field filters the list by name. `Esc` clears it. The status bar under the list shows the item count.

Empty State and No Search Results

An empty tab shows a hint such as `Add a frame`. A search that matches nothing shows `No results found`.

Badges on List Items

Each item carries a badge: the frame ID on the CAN and LIN tabs, a kind tag and the length on the PDU tab, and the bit length on the SIGNAL tab.

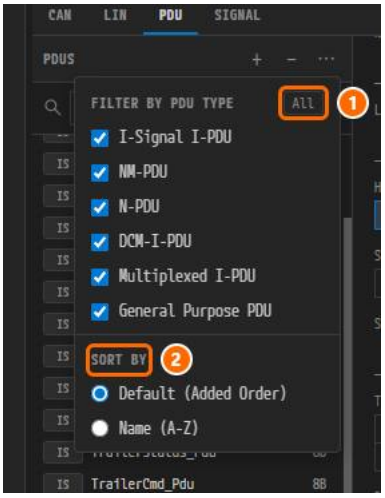


① The `MUX` kind tag, ② the length badge `8B`.

`SOC` and `COM` badges mark items the tool generated. The badge only reports origin; you can still edit and delete the item.

7.4 The More (...) Menu — Filter and Sort

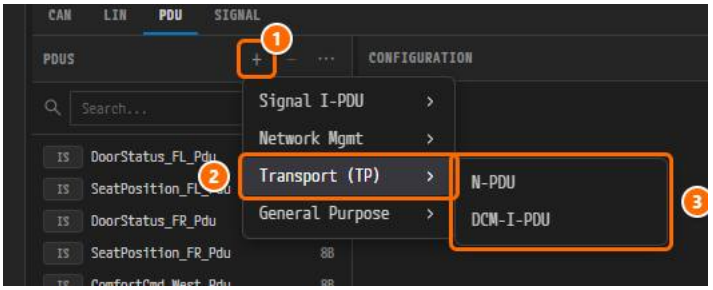
Click **...** to filter by PDU kind (PDU tab only) and to sort the list. Sorting changes only the display order, not the order in the file.



*① **FILTER BY PDU TYPE**, ② **SORT BY**.*

7.5 The Six PDU Kinds

1. On the PDU tab, click **+**. A two-level menu opens.
2. Hover over a group on the left, such as **Transport (TP)**.
3. Click the kind in the submenu. The PDU is created.



*① **+**, ② **Transport (TP)**, ③ **N-PDU / DCM-I-PDU**.*

The kinds are I-Signal I-PDU, Multiplexed I-PDU, NM-PDU, N-PDU, DCM-I-PDU and General Purpose PDU. A name that already exists gets a number, as in `NewF`

7.6 Adding and Deleting Items

Adding a Frame (CAN / LIN Tabs)

1. Select the CAN or LIN tab and click **+**.
2. The new frame is created together with a PDU dedicated to it, and is selected.

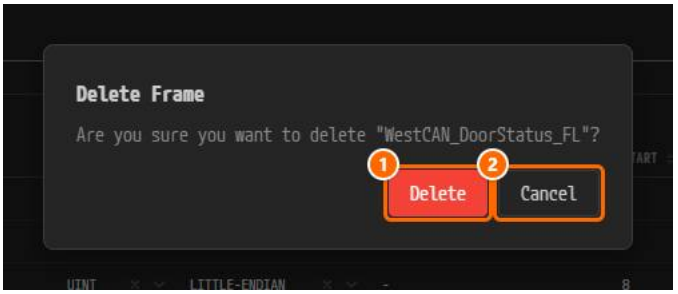
Adding a Signal

- **+ on the SIGNAL tab** adds a signal to the dictionary only. It is not mapped to any PDU.
- **+ in the SIGNALS table header** creates the signal and maps it to the selected frame or PDU. If the PDU is too short, its length grows, and a CAN frame'

When the frame is full, the add is silently ignored. Rearrange the frame or the PDU layout first.

Deleting

1. Select the items in the list.
2. Click **−** or press `Delete`.
3. Click **Delete** in the confirmation dialog.



*① **Delete**, ② **Cancel**.*

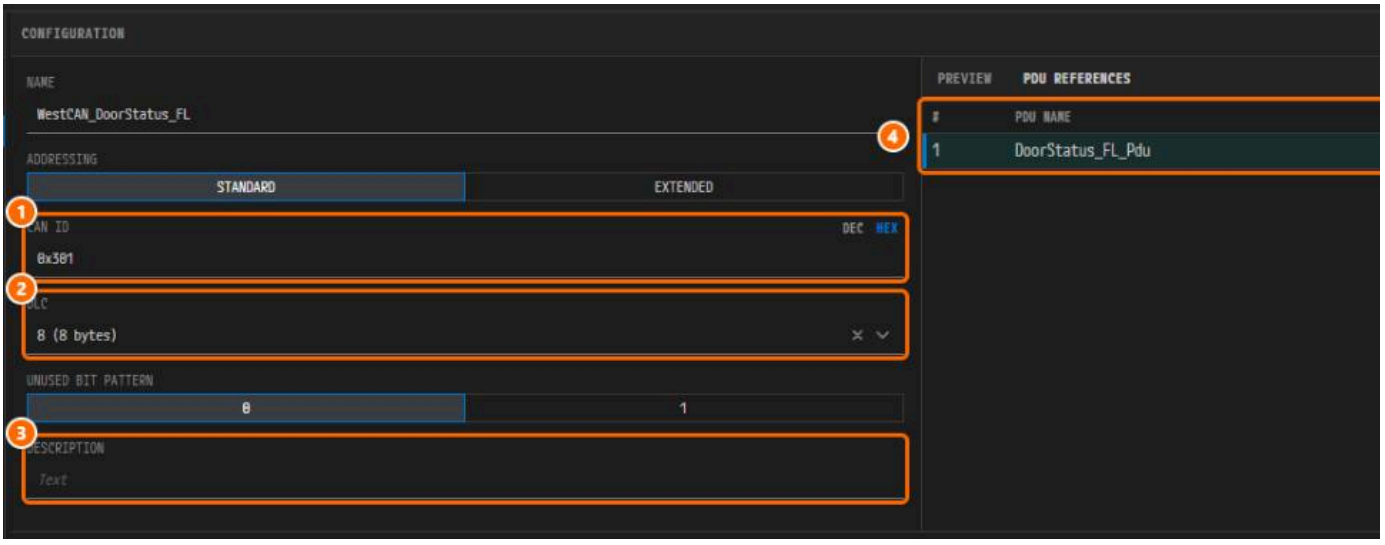
- The × at the end of a **SIGNALS** table row deletes the signal at once, with no dialog, and removes it from the dictionary too. Use **EDIT** > **Undo** to restore
- Delete a signal that has data mappings from the table. Only the table × also cleans up the data mapping references in the System model.

7.7 Configuration Panel — Editable Fields by Item Type

The Configuration panel shows the form of the selected item. Choice fields are drawn as segmented buttons with upper-case labels.

7.7.1 CAN Frame

Set Name, Addressing, CAN ID, DLC, Unused Bit Pattern and Description. The right column has the **PREVIEW** and **PDU REFERENCES** tabs.



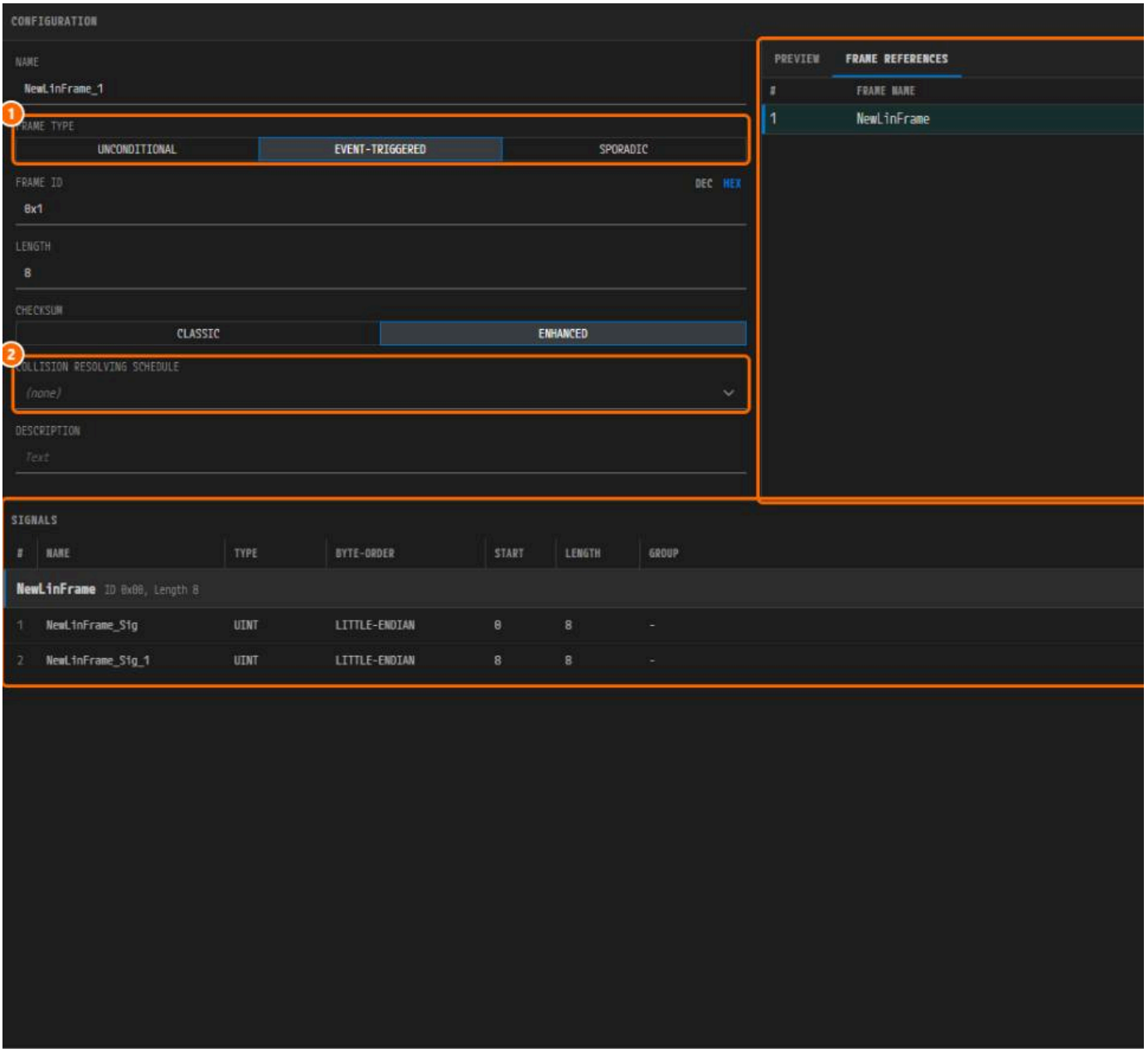
*① **CAN ID** with **DEC / HEX**, ② **DLC**, ③ **Description**, ④ the **PDU REFERENCES** table, ⑤ the PDU picker.*

- The CAN ID range follows Addressing: `standard` allows up to `0x7FF`, `extended` up to `0x1FFFFFFF`.
- DLC values 9 to 15 are the CAN FD lengths. Changing the DLC also changes the length of the PDU, when the frame references a single PDU.

7.7.2 LIN Frame

The fields depend on the Frame Type (`unconditional`, `event-triggered` or `sporadic`).

- An `unconditional` frame has the editable **SIGNALS** table and the **PDU REFERENCES** tab.
- An `event-triggered` or `sporadic` frame has no signals of its own. It has the **FRAME REFERENCES** tab and a read-only list of the referenced frames' s for `event-triggered`.



*① **Frame Type**, ② **Collision Resolving Schedule**, ③ the **FRAME REFERENCES** tab and its picker, ④ the read-only **SIGNALS** list.*

To edit the signals, go to the unconditional frame that owns them.

7.7.3 I-Signal I-PDU

Set Name, Length, Unused Bit Pattern, Cycle Time (ms), Transmission Mode and Description. The form has the **PREVIEW** tab and the **SIGNALS** table. Cycle T
ARXML in seconds.

7.7.4 NM-PDU

Set Name, Length, Unused Bit Pattern, NM Vote Information, NM Data Information and Description. An NM-PDU can carry UserData signals, so it has the sar

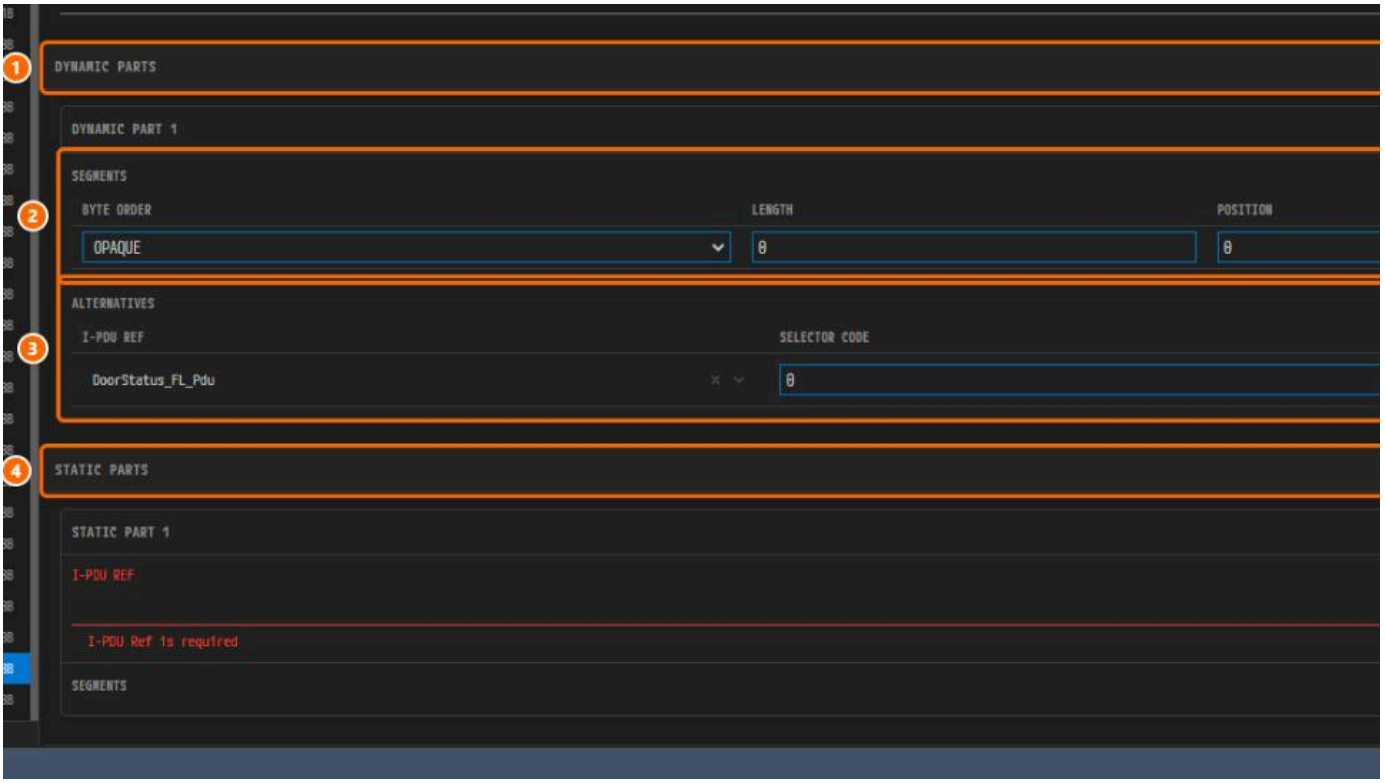
7.7.5 N-PDU · DCM-I-PDU · General Purpose PDU

These three kinds hold no signals. Their form has only Name, Length, Description and one kind-specific field (Diag PDU Type for DCM-I-PDU, Category for (

7.7.6 Multiplexed I-PDU

Set the selector field and the Trigger Mode, then build the parts below the form.

1. Click **+** in `DYNAMIC PARTS` to add a dynamic part. Fill its `SEGMENTS` and `ALTERNATIVES`.
2. Click **+** in `STATIC PARTS` to add a static part. Set its required `I-PDU REF`.



① ***DYNAMIC PARTS***, ② ***SEGMENTS***, ③ ***ALTERNATIVES***, ④ ***STATIC PARTS***, *with* `Static Part 1` *missing its I-PDU Ref.*

An empty `I-PDU REF` shows `I-PDU Ref is required` in red.

7.7.7 Signal (SIGNAL Tab)

Set Name, Bit Length, Type, Unit, Factor, Offset, Min, Max, Init Value (7.11) and Description.

- **Signal Groups** is display only. Change the group in the `GROUP` column of the **SIGNALS** table (7.9).
- `array` and `struct` can be set only here; the table offers `UINT`, `SINT` and `FLOAT`.

7.8 The PDU REFERENCES and FRAME REFERENCES Tabs

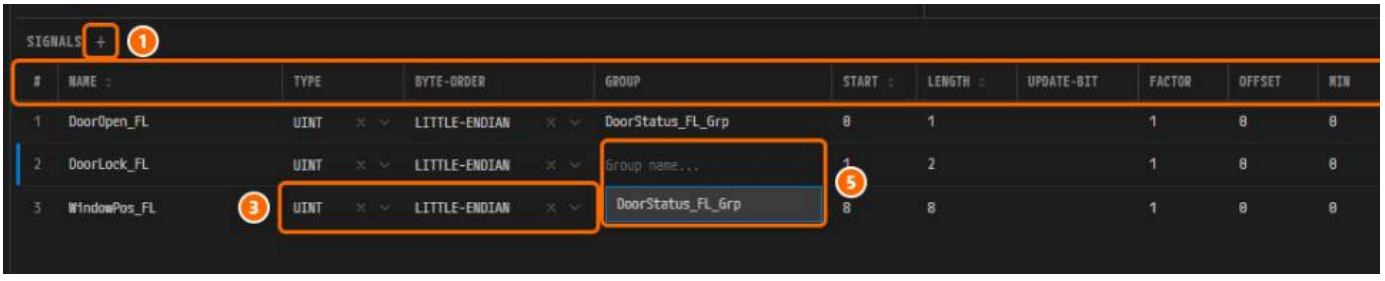
A frame's references are set on the second tab of the form's right column.

1. Click **PDU REFERENCES** (CAN frame, LIN unconditional frame) or **FRAME REFERENCES** (LIN event-triggered or sporadic frame). A picker appears at the ri
2. Click the picker and check the items you need.
3. Close the dropdown. The items are applied and listed in the table. With two or more, drag a row to reorder.

Both references are required. An empty one turns the tab label red. The **PDU REFERENCES** label also turns red when the PDUs exceed the frame size or sig

7.9 The SIGNALS Table

The table below the form lists the signals mapped to the selected frame, I-Signal I-PDU or NM-PDU. Click **+** after the title to add a signal.



① **+** (`Add Signal`), ② *the column headers,* ③ `TYPE` *and* `BYTE-ORDER`, ④ *a* `PDU` *cell,* ⑤ *a* `GROUP` *cell being edited.*

Columns

The main columns are `NAME`, `TYPE`, `BYTE-ORDER`, `GROUP`, `START`, `LENGTH`, `INIT VALUE` and `PDU`. `NAME`, `START` and `LENGTH` can be sorted. Rows whose bit

Editing a Cell and Rejected Input

1. Click the cell.
2. Change the value and press `Enter`, or move the focus away.
3. Press `Esc` to cancel.

An empty, duplicate or rule-breaking `NAME` is not saved. A wrong number restores the original value.

GROUP Column Autocomplete

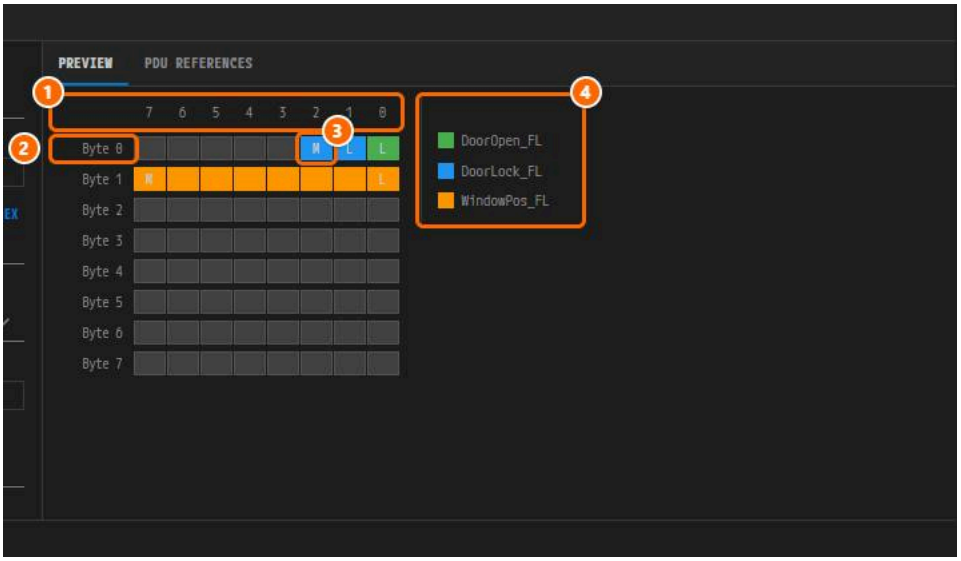
1. Click a `GROUP` cell.
2. Type a group name. Pick a candidate, or choose `Create "<name>"` to make a new group.
3. Clear the value, or enter `-`, to remove the signal from its group.

A signal belongs to one group at a time.

7.10 The PREVIEW Tab and the SIGNAL GROUPS Panel

Bit Layout (PREVIEW Tab)

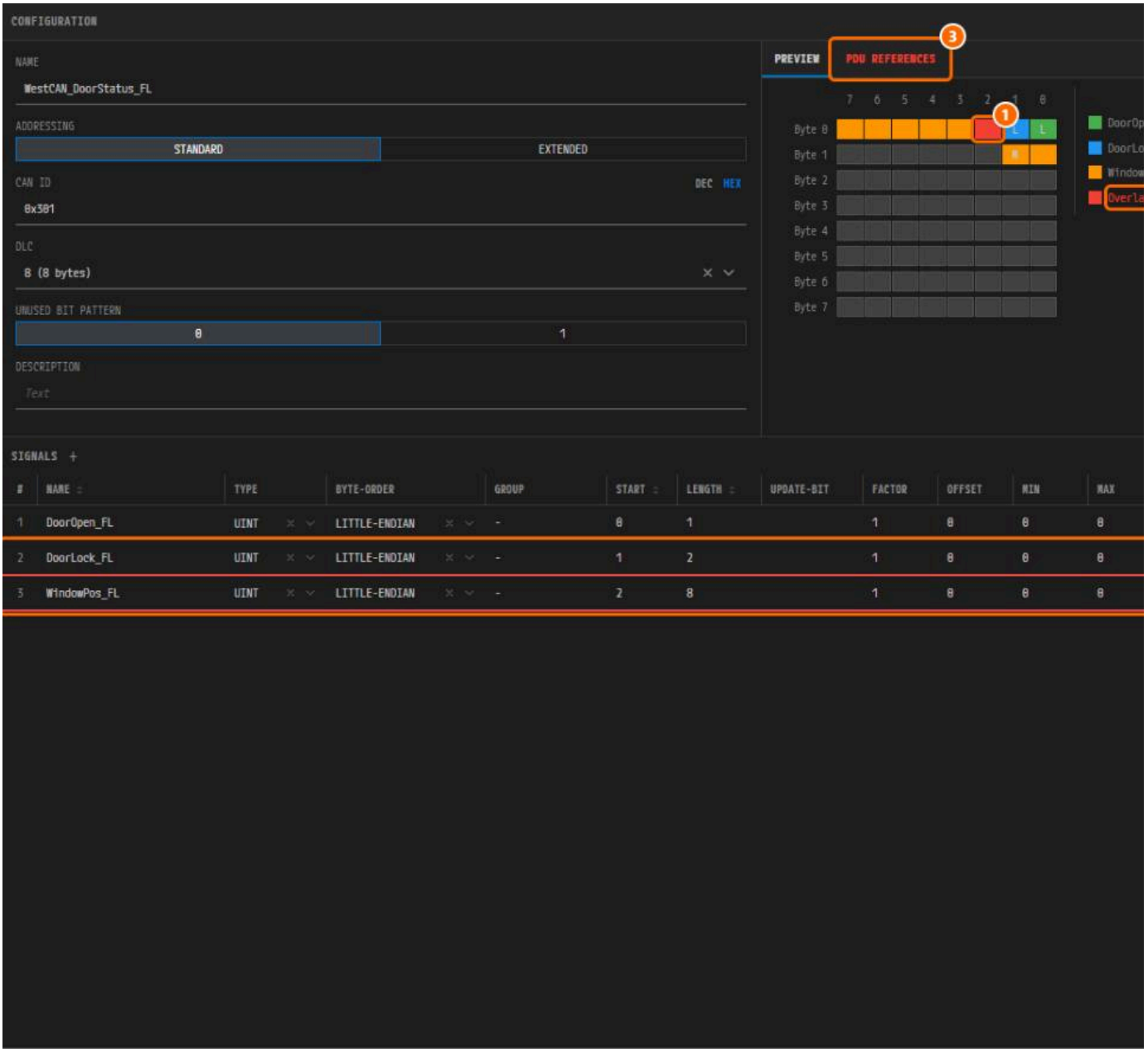
PREVIEW draws the bits of each byte, with one color per signal. `L` marks the LSB, `M` the MSB, and `U` the update bit.



① The bit number row, ② a byte row label, ③ an MSB (`M`) marker, ④ the legend.

An overlap is an error. To fix it:

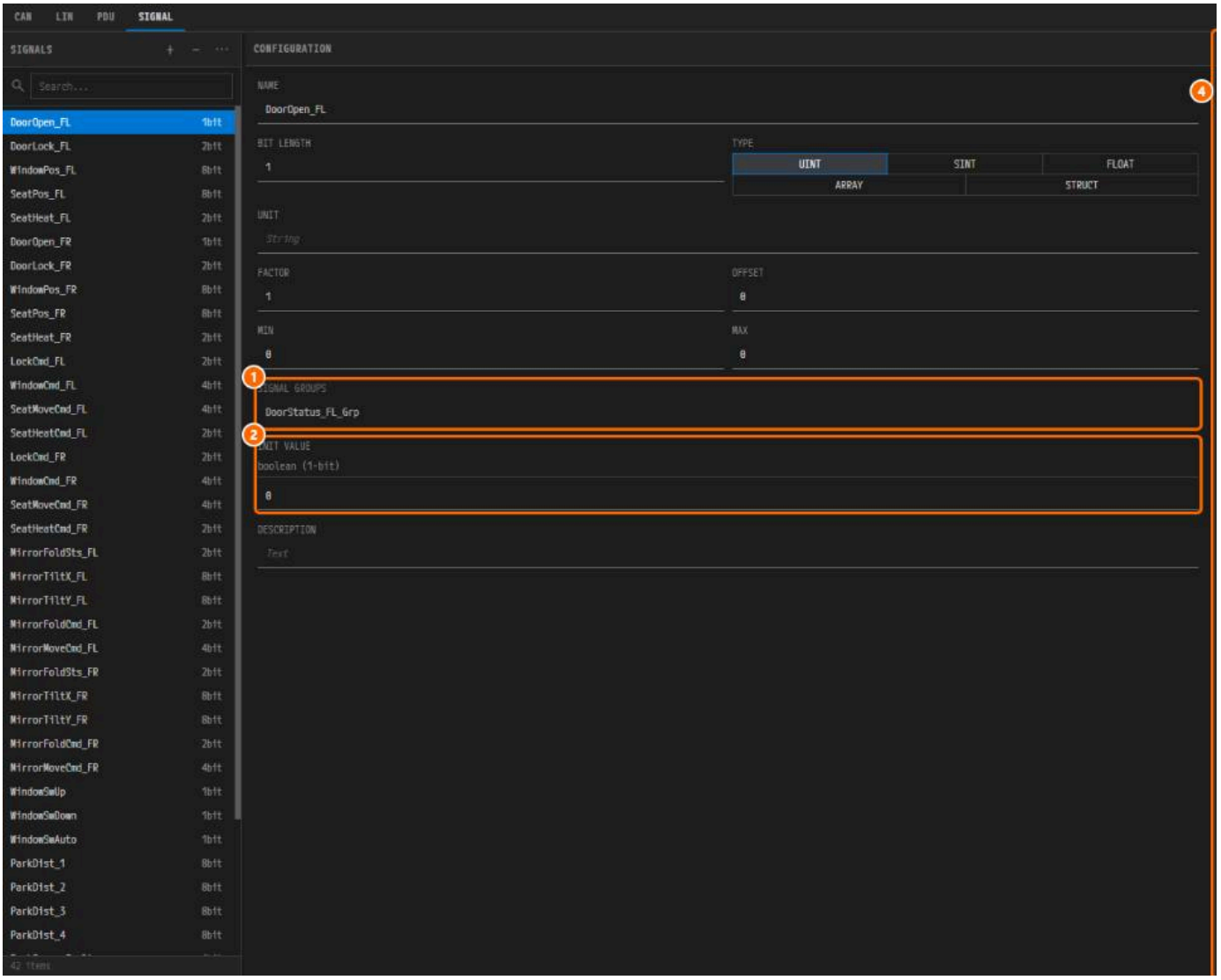
1. Find the overlapping bit in **PREVIEW**. It is painted in the error color.
2. Change the **START** of one of the two signals (7.9).
3. If there is still no room, check that the last bit does not exceed the frame's **DLC** (**Length** for LIN).



*① The overlapping bit, ② `Overlap (Error)`, ③ the red **PDU REFERENCES** label, ④ the highlighted rows.*

Signal Group Tree (SIGNAL GROUPS Panel)

A read-only tree on the right of the SIGNAL tab. Click a group to expand it, and click a signal to open it in the Configuration panel.



*① The read-only **SIGNAL GROUPS** field, ② **INIT VALUE**, ③ the **SIGNAL GROUPS** panel, ④ an expanded group row.*

7.11 Initial Values (Init Value)

The input form is derived from the signal's own properties: a combo box for Text Table entries, 0 or 1 for a 1-bit signal, and an integer range for `uint` and

- The model always stores a number.
- A value outside the range is still accepted and only marked as an error.

The INIT VALUE Column in the Table

A plain number can be edited in the cell. A structured value shows a badge such as `STRUCT` and cannot be edited.

7.12 Text Table (Enumeration)

A signal's enumeration entries are listed in the `Text Table (N entries)` section of the Configuration panel. They are read-only and are created only by

7.13 Saving and Undo

Action	Command
Save	`Ctrl+S`
Undo	**EDIT** > **Undo** (`Ctrl+Z`)
Redo	**EDIT** > **Redo** (`Ctrl+Y`)

Shortcuts work while the focus is inside the editor. Otherwise use the **EDIT** menu. Undo switches to the tab of the reverted item and selects it.

7.14 External Changes and Import

- When the file changes on disk, the editor reloads it.
- With unsaved changes, it keeps your edits and shows the warning `Changed on disk` with the second line `Unsaved edits kept`.
- An ARXML or DBC import saves every open editor first and reloads this editor when it finishes.

7.15 Integration with Other Editors and Features

- **System editor**: its IVC model references the frames and PDUs defined here.
- **System model data mappings**: renaming a signal also updates them.
- **ECUC**: Com and PduR configurations are not generated from this model (12. ECUC).
- **Validation**: results are collected in 14. Validation.
- **DBC**: import with **Import** in the **DESIGN** header. Export with **Export** in the **DESIGN** header, or with **Export to DBC** on a CAN cluster in the System editor (

7.16 Editing Rules Summary

- Names follow the AUTOSAR SHORT-NAME rules: a letter first, then letters, digits and underscores. A name that breaks them is not applied.
- Empty required references and an empty `TYPE` or `BYTE-ORDER` are marked in red.
- Sort, filter, panel widths and column widths are screen state only and are not saved.

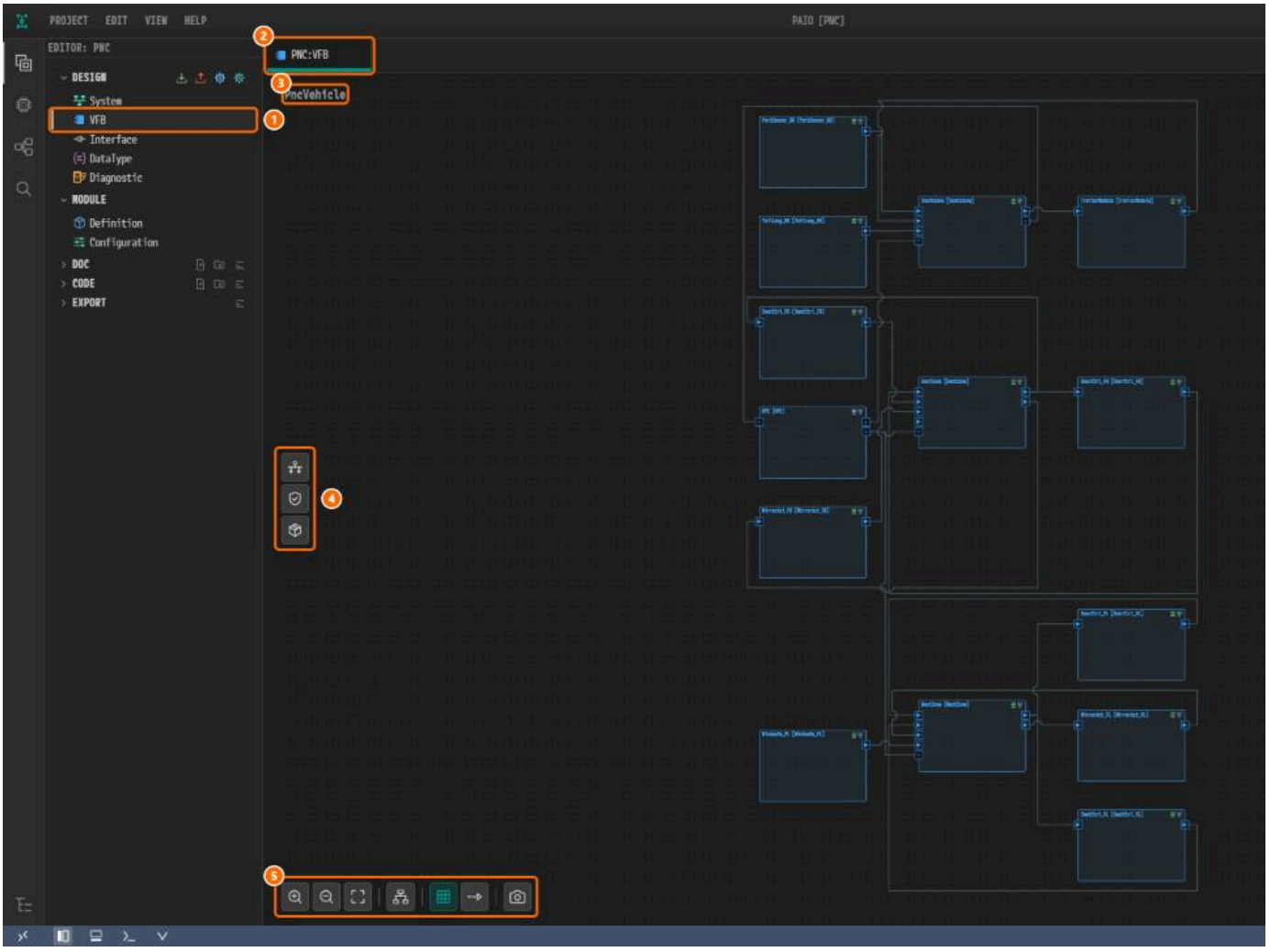
VFB Editor

What This Chapter Covers

This chapter explains how to use the VFB editor: from the Virtual Functional Bus point of view you place software components (SWC) and compositions,

- Audience: users who design the software architecture of the AUTOSAR Classic or Adaptive platform
- Prerequisites: a project is open in the window. It helps to define the port interfaces first in 09. Interface Editor. The ECU and HPC nodes used by ECU/

To open the editor, click **VFB** under **DESIGN** in the **EDITOR** view on the left. The tab `<project>:VFB` opens, for example `PNC:VFB`. Do not open `design/VFB` read-only JSON view, not as the editor.



*① **VFB** under **DESIGN**. ② The tab `PNC:VFB`. ③ The system label `PncVehicle`. ④ The action strip at the left edge of the canvas. ⑤ The canvas toolbar.*

Control	Contents
Action strip (left edge of the canvas)	**VFC (PNC Overlay)** (8.3), **Intrusion Detection System** (8.19), **Software Cluster** (8.20)
Canvas toolbar (bottom left)	**Zoom In**, **Zoom Out**, **Fit to View**, **Auto Layout**, **Toggle Grid**, **Toggle Connector Animation**, **Export as PNG** (8.3)
Right-edge buttons	Property panel toggle, lock, and **System** (the PAIO logo), which shows the SYSTEM panel (8.7)

8.1 Files Written on Save

One save writes several files together: the VFB model, the SWC definition of each SWC, the canvas layout of each composition (`design/SWC/Layout/<Com`) configuration.

- Connectors are stored in the composition's SWC definition, not in the VFB model.

- The `.vfb.layout` file holds only the screen arrangement. It is not written to ARXML.
- The old file of a renamed SWC is deleted only when you save. If the file list looks wrong after a rename, save with `Ctrl+S` first.

8.2 SWC Kinds

The **Add SWC** submenu offers seven kinds: **Composition**, **Adaptive Application**, **Application**, **Complex Device Driver**, **Ecu Abstraction**, **Sensor Actuator** and **S**

An InterfaceMapping node is not an SWC kind. You create it with **Add Interface Mapping** on the empty canvas at the root level, and its only role is to link a

8.3 Canvas Operations

The operations you use most are below.

Mouse

- Drag on empty space with the left button, the middle button, or `Space` + left button to pan the view. `Ctrl` + wheel zooms.
- Drag a node to move it. `Shift` + drag from a port starts a connection (8.6).
- Double-click a composition to drill into it. Double-click an Adaptive Application to open **AA Internal Behavior** (8.9), and any other atomic SWC to oper
- Right-click opens the context menu (8.5).

Keyboard

- `Ctrl+S` saves, `Ctrl+Z` undoes, and `Ctrl+Y` redoes.
- `Delete` / `Backspace` deletes the selected boxes after a confirmation. It does not delete a connection: use **Disconnect** in the connection's right-clic
- The arrow keys and `Ctrl+C` / `Ctrl+V` / `Ctrl+X` do nothing on the VFB canvas.

Toolbar Buttons (Bottom Left of the Canvas)

- **Fit to View** fits the whole content into view.
- **Auto Layout** runs the automatic layout. It replaces every connection route you adjusted by hand, so it asks first when such a route exists.
- **Export as PNG** saves the current diagram into the `doc/capture/` folder of the project.

VFC (PNC Overlay)

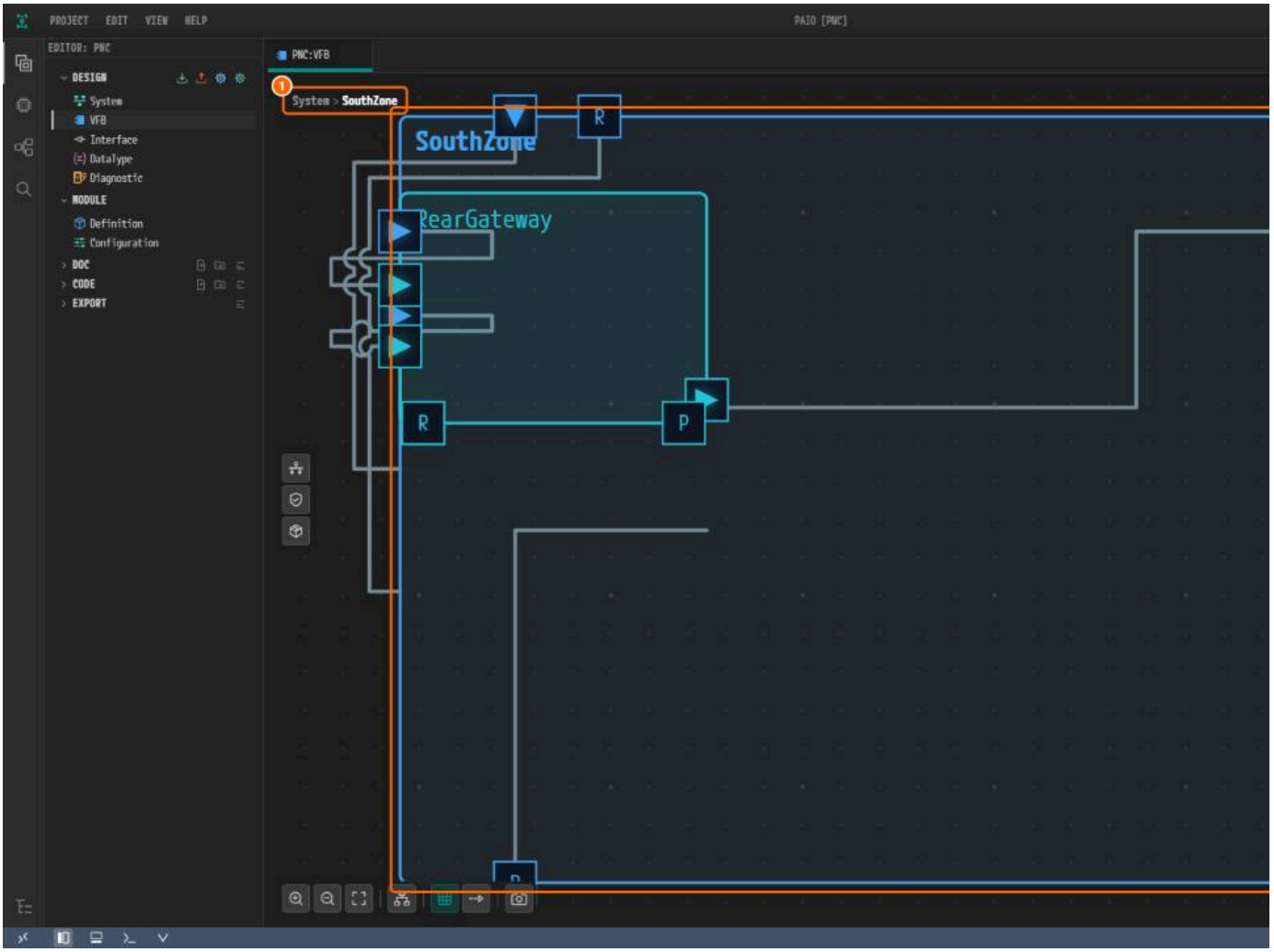
The first icon of the action strip turns an overlay on and off. Pick a VFC (Virtual Function Cluster, a port group) in the **PNC OVERLAY** panel: its nodes and po dimmed. **BOUND PNCS** lists the partial networks the VFC is mapped onto.

The overlay only shows; it never changes the model. You declare VFCs in the **Partial Network (PNC) Configuration** dialog of 06. System Editor.

8.4 The Root Level and Drill-Down

- **Root level**: you have not entered any composition. Top-level compositions and InterfaceMapping nodes are placed here.
- **Drilled-down state**: you have entered one composition. It is drawn as a frame, with its child nodes inside.

1. Double-click a composition. A breadcrumb such as `System > SouthZone` appears at the top left of the canvas.
2. Click an earlier entry of the breadcrumb to return to that level.



① The breadcrumb `System > SouthZone`. ② The composition frame, with the child SWC `RearGateway`, the boundary ports and the connectors.

To create a node while drilled down, right-click the frame (the composition). The empty canvas outside the frame shows no menu.

8.5 Context Menus

The menu depends on what you right-click. With several boxes selected, right-clicking one of them shows only **Delete N items**.

8.5.1 Empty Canvas (Root Level Only)

The menu has **Add Composition**, **Attach Composition** (attach an already defined composition as a prototype) and **Add Interface Mapping**.

8.5.2 Composition Frame (the Composition You Are Currently Drilled Into)

The menu has **Add SWC** (one of the seven kinds of 8.2), **Attach SWC**, **Add Port** (**PPort**, **RPort** or **PRPort**), **Add Composition** and **Attach Composition**.

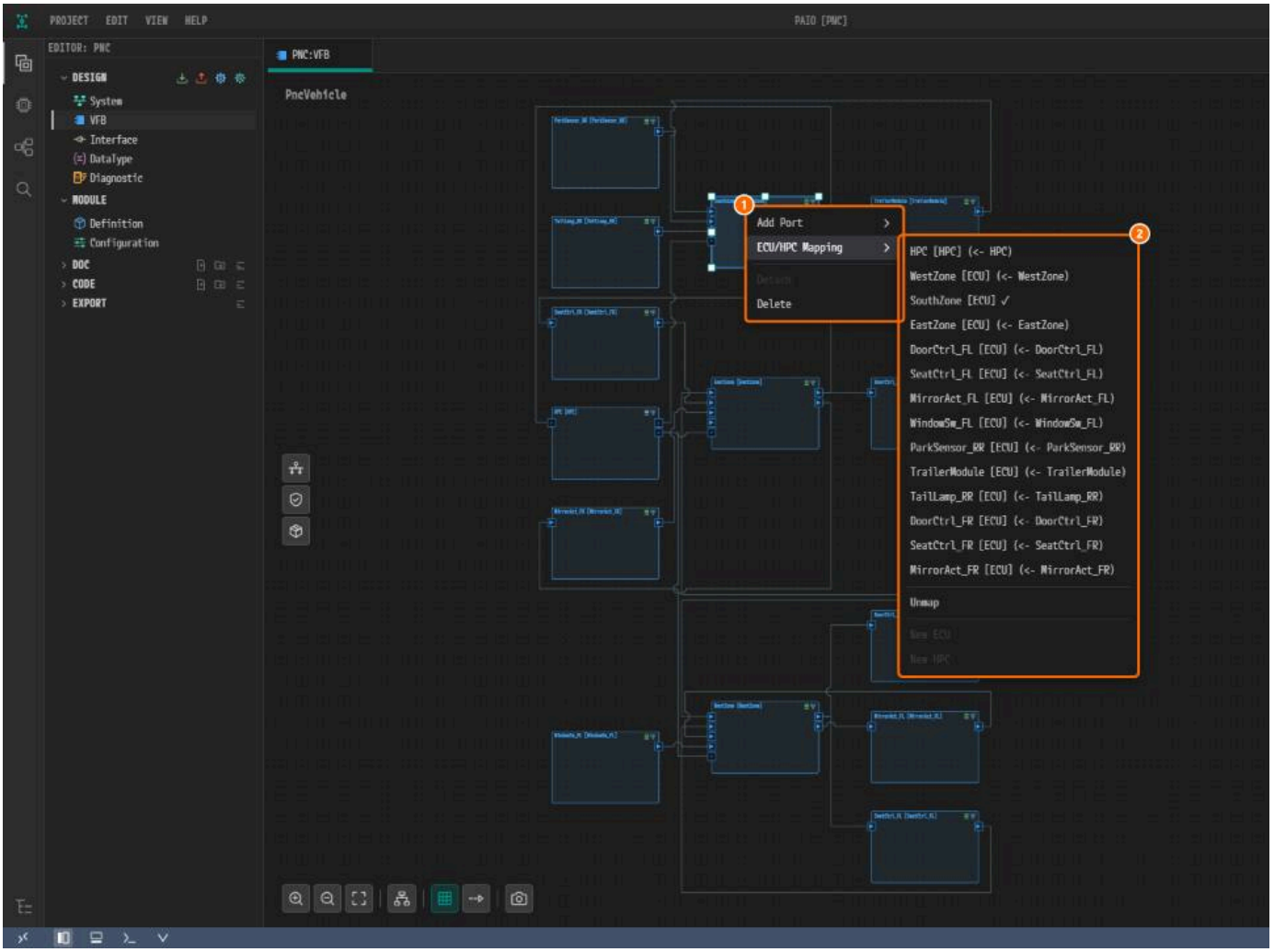
8.5.3 Composition Node (a Composition You Are Not Drilled Into)

The menu has **Add Port**, **ECU/HPC Mapping** (top-level compositions only), **Detach** and **Delete**. To manage child nodes, drill into the composition.

- **Detach** removes only that prototype from the parent composition. The SWC type remains.
- **Delete** asks in the **Delete SWC** dialog, then removes the SWC from every composition and deletes its type file. The file deletion takes effect when you

To map a composition to an ECU or HPC:

1. Right-click a top-level composition and open **ECU/HPC Mapping**.
2. Pick a node of the System editor, or click **New ECU** / **New HPC** to create a node of the same name and map it. **Unmap** clears the mapping.



*① The node menu of `SouthZone`. ② The **ECU/HPC Mapping** submenu: `SouthZone [ECU] ✓` is the node mapped to this composition.*

A mapped composition shows the node name in its header, such as `SouthZone [SouthZone]`, and an ECU or HPC icon. Click the icon to open the ECU Con (8.10) dialog.

8.5.4 Atomic SWC Node

The menu has **Add Port**, **Detach SWC** and **Delete**. **Delete** goes through the same **Delete SWC** dialog as 8.5.3 and deletes the type file as well.

8.5.5 Port Node

The menu has **Connect to**, **Connect to Mapping**, **Derive SR/CS Interface** and **Delete**. **Derive SR/CS Interface** appears only at the root level, on a port whose

8.5.6 InterfaceMapping Node

Delete is the only menu item.

8.5.7 Connections

The menu has **To Step Edge** (the automatic step router draws it again), **To Straight Edge** and **Disconnect**. Right-clicking a waypoint handle opens the sam

8.6 Connecting Ports

1. Hold `Shift` and drag from a port. Valid drop targets are highlighted in green, invalid ones in red.
2. Release the button over the target.
   - Over a compatible port: the two ports are connected.
   - Over the body of an SWC: a matching port is created and connected.
   - Over an InterfaceMapping node: the port is connected to that node.
3. Click the connection. If a **Basic Info** section appears in the properties panel, the connection exists.

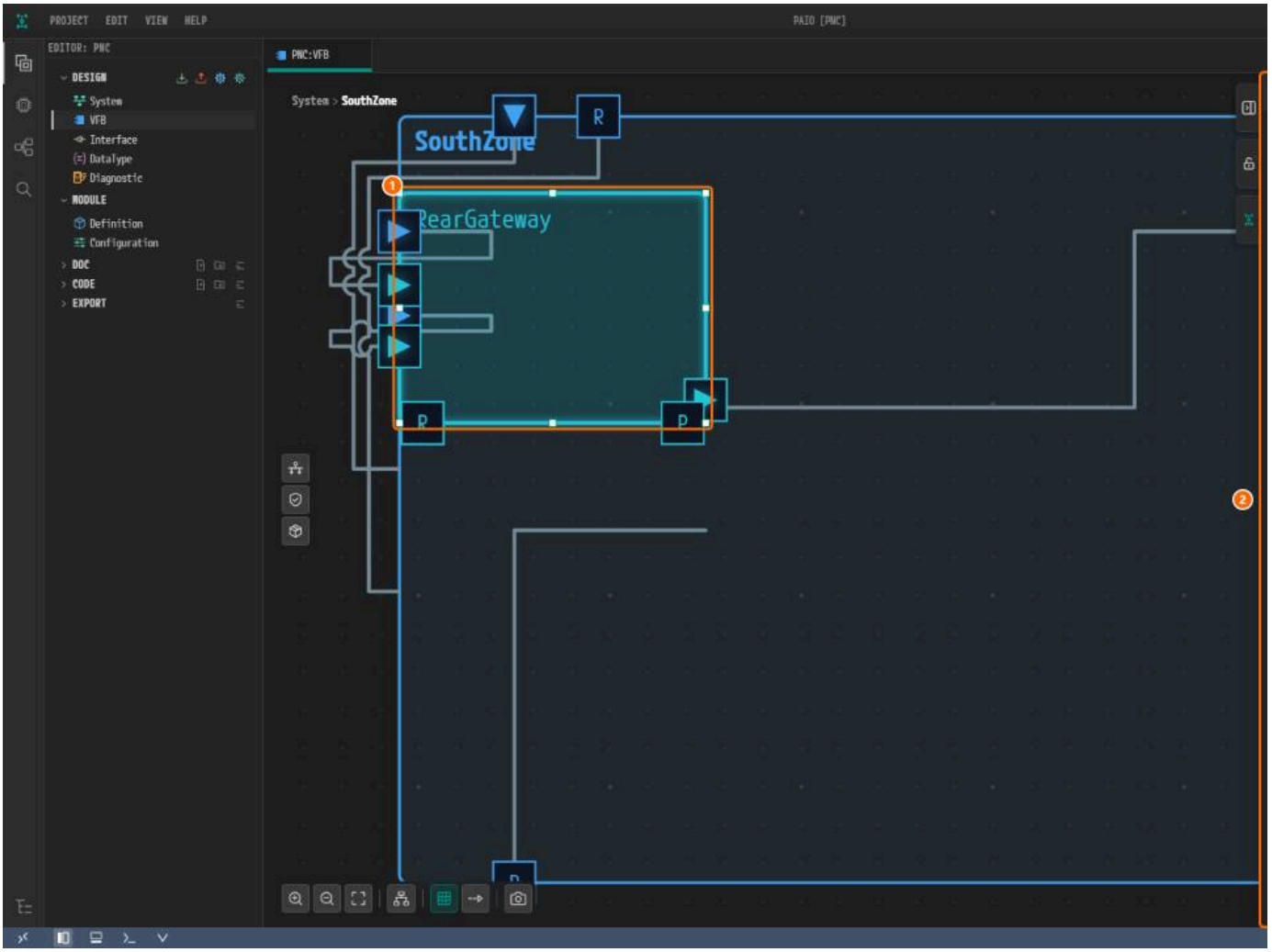
Dragging without `Shift` moves the item. You can also right-click the port and pick the counterpart from **Connect to / Connect to Mapping** (8.5.5). If **Con** you expect to connect, use `Shift` + drag.

8.7 Properties Panel (Right)

Shows the fields of what you select. Three buttons on the right edge of the canvas toggle the panel, lock it open, and show the SYSTEM panel (**System**, t

SWC / Composition Node

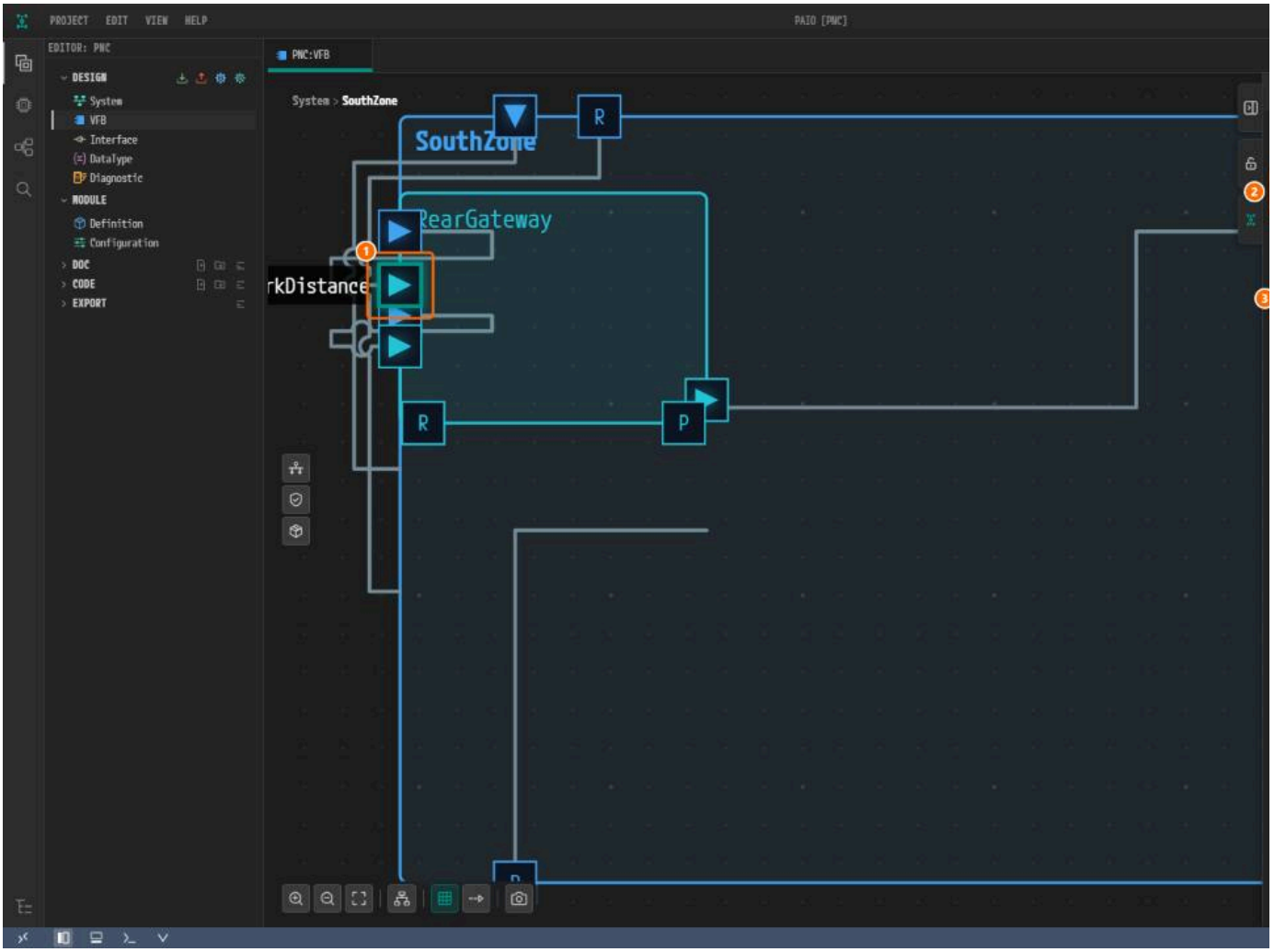
The panel shows **Name**, **AS Prototype**, **Description** and **Ports**. A composition also has **Port Groups**. The port groups of an Atomic SWC are edited in the SWC



*① The selected SWC `RearGateway`. ② The properties panel, titled **APPLICATION**, with **PORTS** listing each port and its PortInterface.*

Port Node

1. Click a port.
2. Pick its interface in **Port Interface**, or create a new one with `[+]`. The candidates depend on the parent SWC kind.
3. Edit the ComSpec section that matches the interface category, for example **Data Element** for Sender-Receiver.



*① The selected port. ② **PORT INTERFACE** `SenderReceiver: SRI_ParkDistance`. ③ The **DATA ELEMENT** section.*

Connections

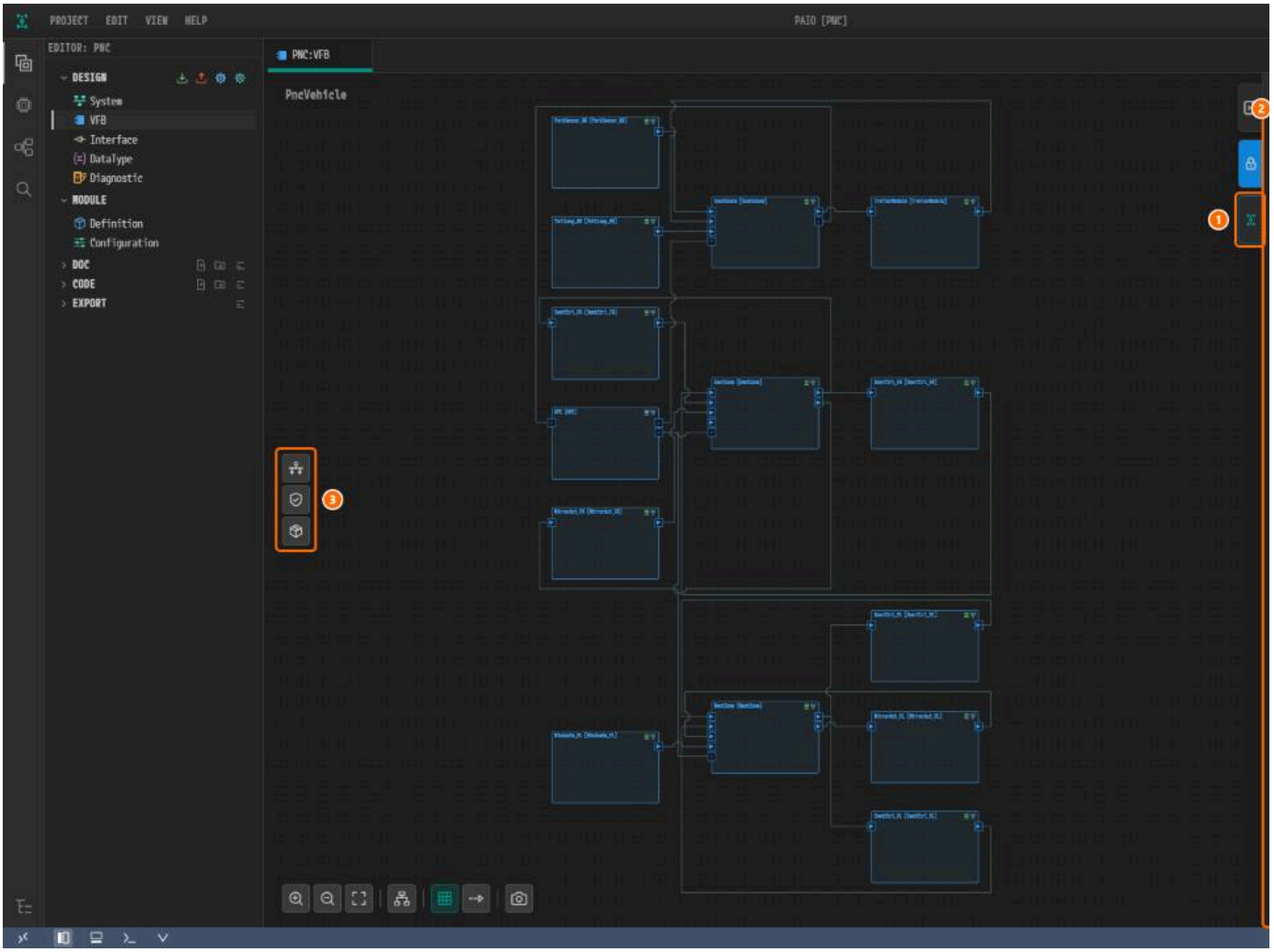
The **Basic Info** section has a single **Name** field.

InterfaceMapping Node

Maps the elements of a service interface to SR/CS interfaces in **Event Mappings**, **Method Mappings** and **Field Mappings**. **Connected Ports** lists the ports c
connected to a ServiceInterface port, the mapping sections do not appear.

The SYSTEM Panel

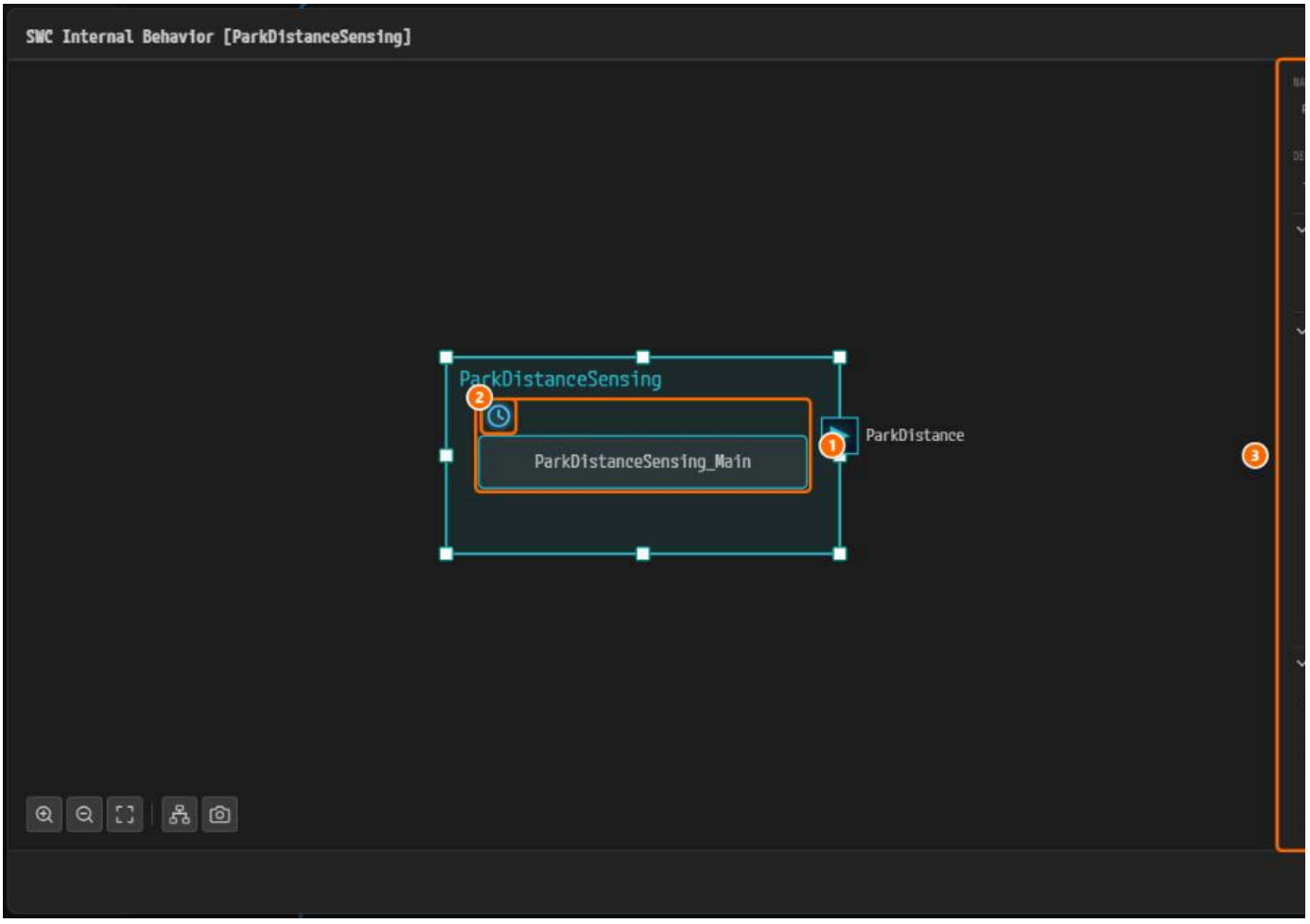
Click **System** on the right edge to show **NAME**, **VERSION**, **DESCRIPTION**, **ROOT COMPOSITION** and the **COMPOSITIONS** tree. Changes are saved to the project a
NAME, **VERSION** and **DESCRIPTION** are shared with the System editor (11. SYSTEM Panel).



*① **System** (the PAIO logo). ② The SYSTEM panel. ③ The action strip.*

8.8 SWC Internal Behavior Dialog

Double-click an Application, Complex Device Driver, Service, Ecu Abstraction or Sensor Actuator node. The canvas is on the left and the properties panel o
Implementation.



*① The Runnable `ParkDistanceSensing_Main`. ② Its RTE event icon (a TimingEvent clock). ③ The right panel with **PORTS**, **PORT GROUPS** and **IMPLEMENTAT***

While edits are unsaved, the title starts with `•`, and **Close** asks whether to save them.

Right-Clicking the Canvas

1. Right-click the SWC body and choose **Add Port**, **Add Runnable** or **Add IRV**.
2. Right-click a Runnable and choose one of the RTE events under **Add RTEEvent**. There are 13, including DataReceiveErrorEvent, DataSendCompletedEv ModeSwitchedAckEvent and ExternalTriggerOccurredEvent.

Each element also has a delete item in its menu.

Runnable Properties

Click a Runnable to edit **NAME**, **SYMBOL** (the C function name in the generated code), **CAN BE INVOKED** and **MIN START INTERVAL**. The `+` in the **Access Point** or an RTE event.

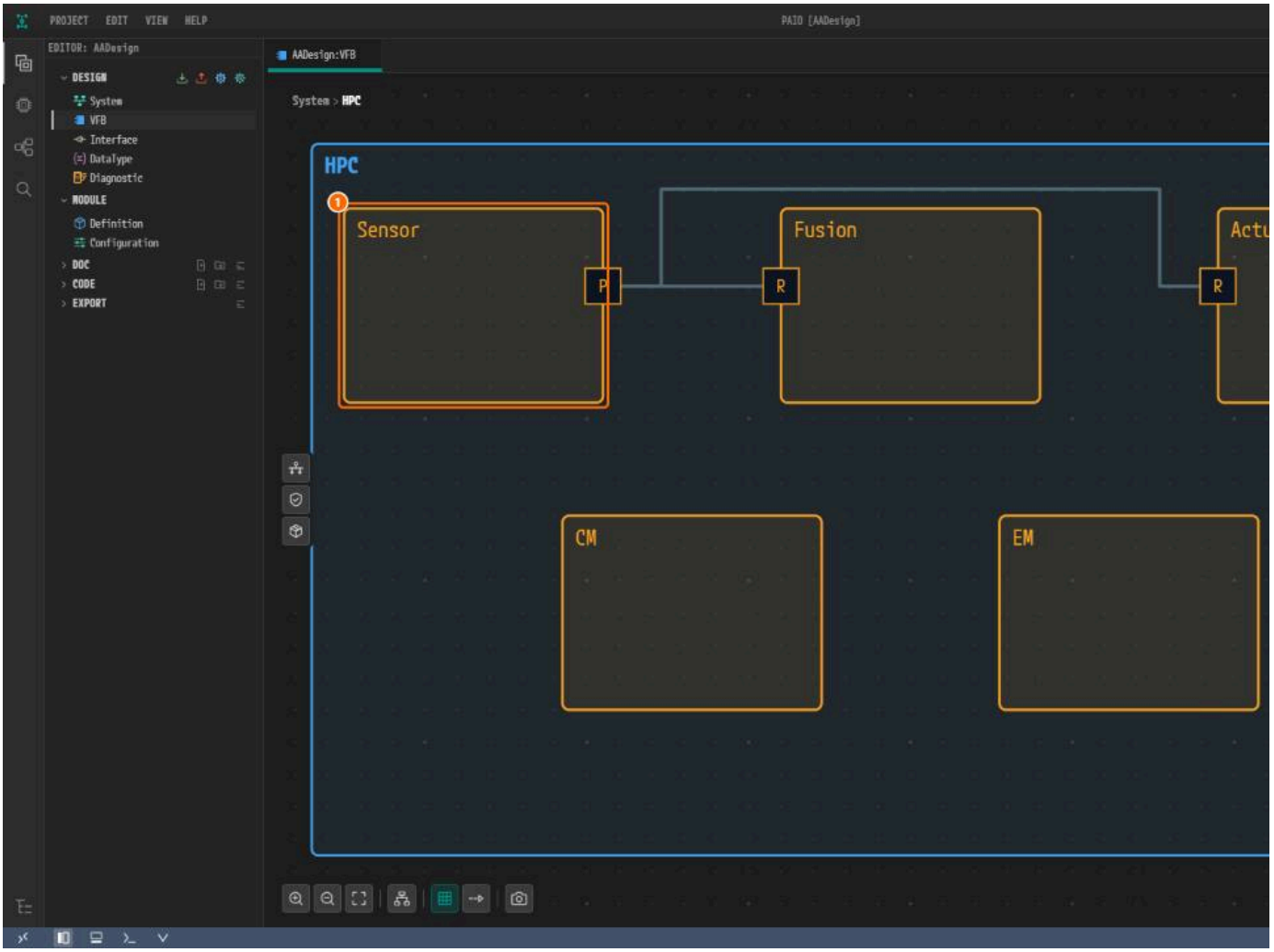
`Shift` + drag from a port onto a Runnable shows only the items that match the port's direction and interface category. For example, a Sender-Receiver DataWriteAccess. A port typed by a trigger interface (9.3.8) suggests ExternalTriggeringPoint on a P-Port and ExternalTriggerOccurredEvent on an R-Por AsynchronousServerCallResultPoint.

Port API Option

Clicking a port in this dialog adds a **Port API Option** section with **Enable Take Address**, **Indirect API**, **Error Handling** and **Port Arg Values**. The section exists c

8.9 AA Internal Behavior Dialog (Adaptive Application)

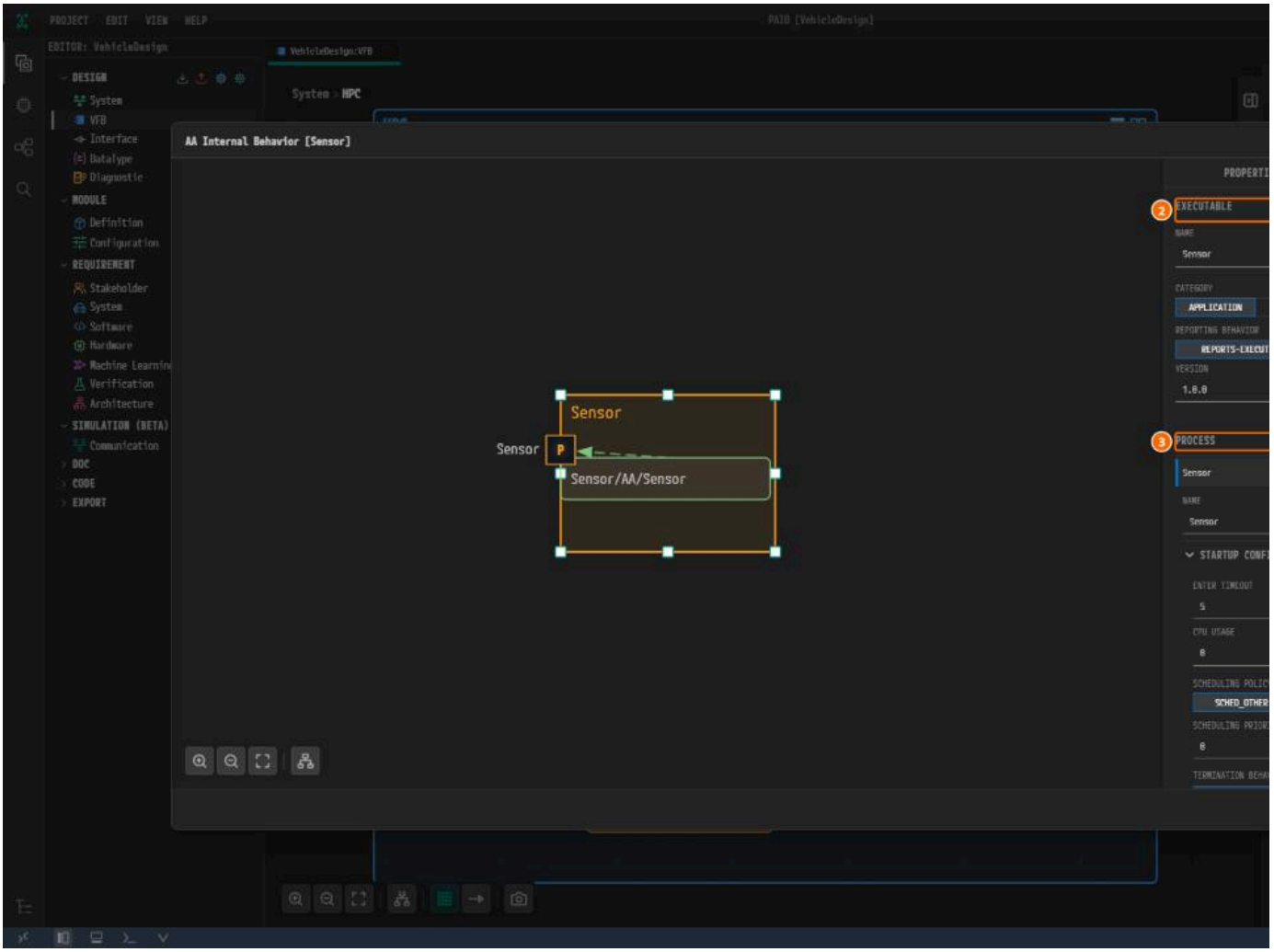
Double-click an Adaptive Application node. Adaptive Application nodes are drawn in orange.



① The node `Sensor` inside the composition `HPC`.

The right panel has two tabs.

- **PROPERTIES**: the node's **Name**, **AS Prototype**, **Description** and **Ports**.
- **EXECUTION**: the **EXECUTABLE**, and one **PROCESS** card per process with **STARTUP CONFIG**, **FUNCTION GROUP STATE**, **EXECUTION DEPENDENCY**, **PROGRAM A CONFIG**. Click the process name to open or close its card. The **+** of **FUNCTION GROUP STATE** is red while the list is empty.



*① The **EXECUTION** tab. ② **EXECUTABLE**. ③ The **PROCESS** card* `Sensor`.

PARA writes the program arguments and environment variables into the process's execution manifest.

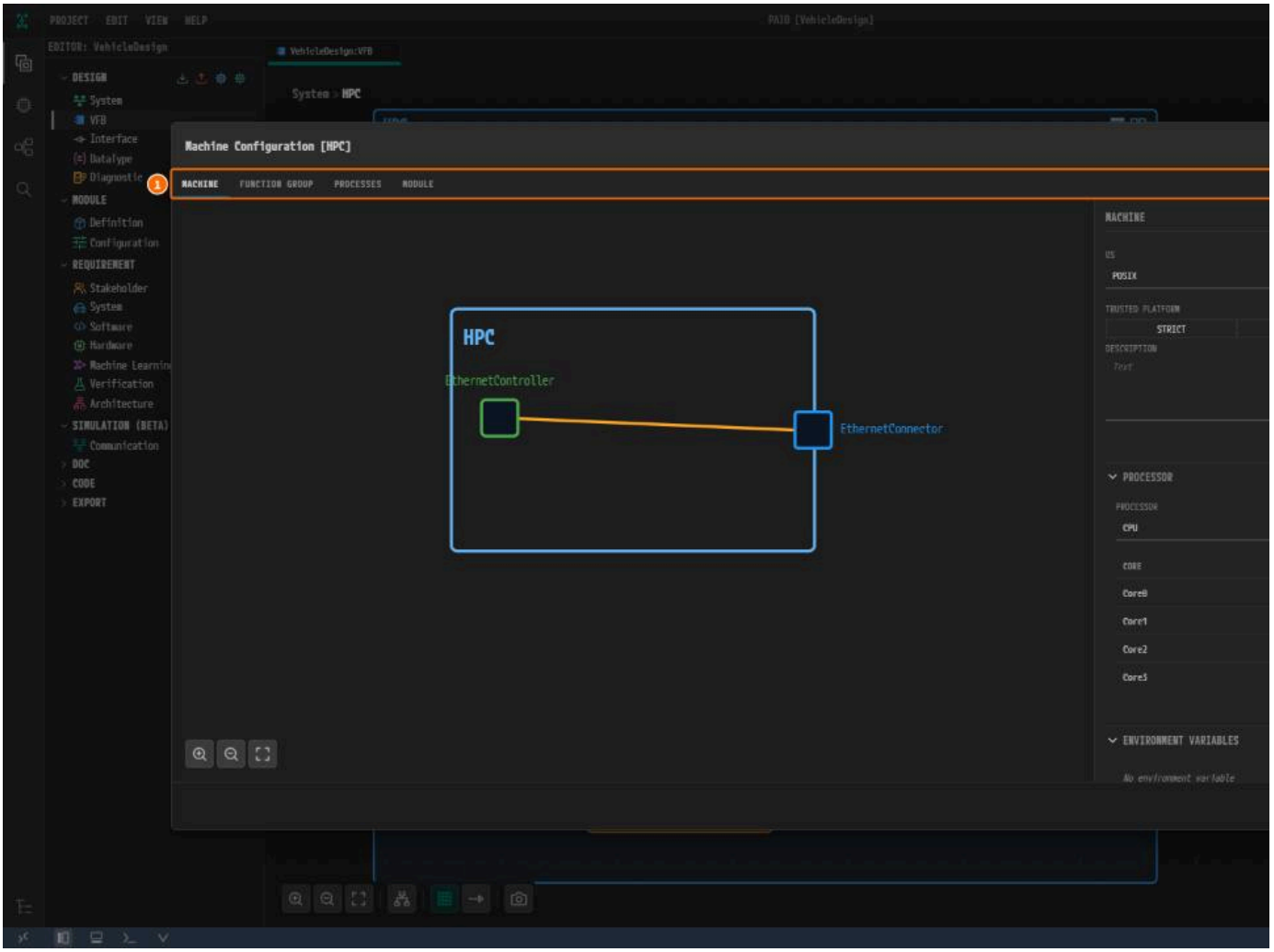
Right-Clicking the Canvas (the AA Body)

The menu has **Add Port** and **Add Instance**. **Add Instance** adds an AP mapping box, grouped by function cluster (**ara::com**, **ara::log**, **ara::sm** and others). Sh
mapping box.

8.10 Machine Configuration Dialog (Adaptive)

Click the HPC icon on a composition node mapped to an HPC. The dialog has four tabs.

- **MACHINE**: OS, trusted platform, processor and cores, the Machine's environment variables, and a read-only network summary of the HPC.
- **FUNCTION GROUP**: the function groups and their state machine.
- **PROCESSES**: the Adaptive Application processes of this composition (8.10.2).
- **MODULE**: the AP function cluster modules (8.10.1).



*① The tabs **MACHINE**, **FUNCTION GROUP**, **PROCESSES** and **MODULE**. ② **PARA Platform Import**.*

When the dialog opens, missing elements are filled in: the **MachineFG** function group, the eleven module instances, and a single `CPU` / `Core0`. At first you the rest alone.

8.10.1 The Eleven Modules on the MODULE Tab

Click a module in the **MODULES** list: its editing view appears in the middle and an overview on the right. **Cryptography** and **Persistency** have no right pane A disabled module keeps its configuration, but it is neither exported nor checked by Validation.

8.10.2 The PROCESSES Tab

Pick a process in the **Processes** list. **CORE AFFINITY** is the only editable value; the rest is read-only.

To make a process the State Management or Platform Health Management process:

1. Select the process.
2. Click the **SM** or **PHM** icon at the right of the header (tooltips `Mark as State Management` and `Mark as Platform Health Management`). A grey icor affiliation can have only one process; marking another process asks whether to reassign it.

Marking a process as SM also creates a StateClient R-Port and a FunctionGroupPortMapping on the SM SWC for every function group. Clicking the lit SM i them again. Clicking the lit PHM icon clears its mark at once; PHM creates no port or mapping. Each assignment is one undo entry. To undo only the assig `Ctrl+Z`.

8.11 PARA Base Configuration

Injects the PopcornSAR AUTOSAR Adaptive Platform base configuration into the project.

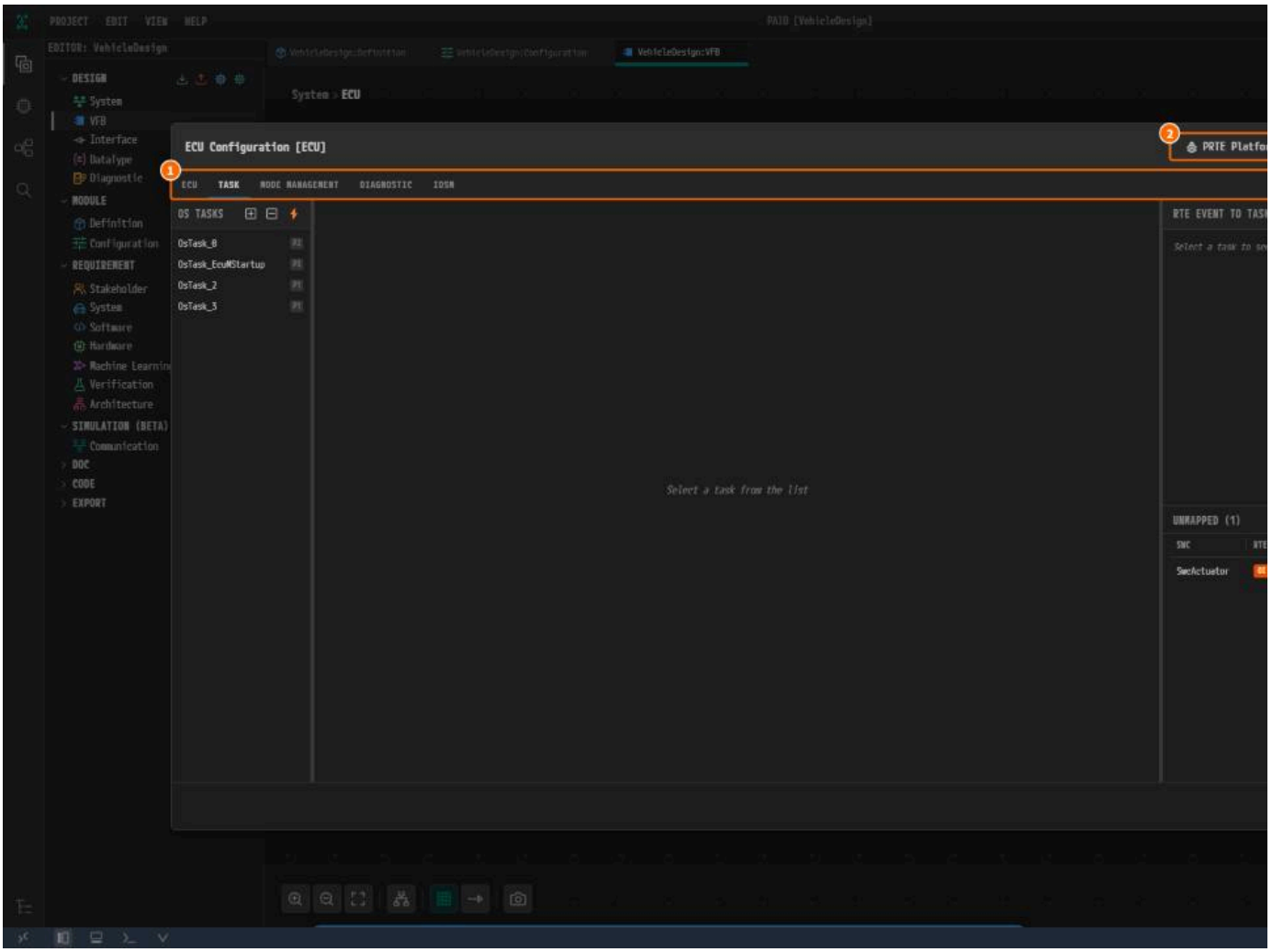
1. Click **PARA Platform Import** in the header of the Machine Configuration dialog, or run **PARA: Import Platform** from the command palette.
2. Click **Import** in the confirmation dialog.

3. If there is a conflict, the conflict resolution dialog opens. On success the Machine Configuration dialog reopens with the merged MachineFG and mo

The whole operation is one undo step. The machine name, the processor and the OS are left untouched.

8.12 ECU Configuration Dialog (Classic)

Click the ECU icon on a composition node mapped to an ECU. The dialog has five tabs, **ECU**, **TASK**, **MODE MANAGEMENT**, **DIAGNOSTIC** and **IDSM**, and two hea **AUTO CONFIGURATION** (8.16). Save its edits with `Ctrl+S`.



*① The tab bar. ② **PRTE Platform Import**. ③ **PRTE AUTO CONFIGURATION**.*

To set up the tasks on the **TASK** tab:

1. In **OS TASKS**, click ⚡ (Auto-Generate) to build the tasks and alarms from the RTE events, or `+` to add a task by hand.
2. Select a task and edit it in **TASK CONFIGURATION** in the middle.
3. Assign the RTE events in **RTE EVENT TO TASK MAPPING** on the right. Drag a mapped row up or down to set the execution order within the task.

Auto-Generate puts the BSW main functions on one task per period (`OsTask_Bsw_10ms` and so on) and sets every task's priority: background lowest, the period higher. An event the RTE starts by a direct call needs no task; it is listed under **DIRECT CALL** and counts as neither mapped nor unmapped. **Autosta**

The TASK tab is synchronized both ways with the `Os` and `Rte` configurations, so the tasks also appear in 12. ECUC.

8.12.1 The ECU Tab

Edits the ECU's **PARTITIONS** (`+` adds, `×` removes) and shows a read-only summary of its **NETWORKS** and **ECU-WIDE** facts. Saving writes the partitions to th partitions; one `Ctrl+Z` covers both. You edit the networks in 06. System Editor.

8.12.2 The MODE MANAGEMENT Tab

Edits the ECU's BswM configuration in three panels: the **BSWM** element tree, the parameters of the selected element, and **SWC SIDE (BswM.swc)** with a C Discovery rules derived from the ECU's SOME/IP service instances. The tab needs PRTE Platform Import (8.15) first.

8.12.3 The DIAGNOSTIC Tab

Shows counts from the ECU's diagnostic specification and holds the switch **Derive Dcm/Dem ECUC from DEXT on Apply**. When the switch is on, saving the from the specification. You edit the specification itself in the Diagnostic editor.

8.12.4 The IDSM Tab

Edits the ECU's IdsM instance with the same editor as the Intrusion Detection System dialog (8.19). **ADD IDSM INSTANCE** creates the instance and **REMOVE**

8.13 Saving, Undo, and External Changes

- `Ctrl+S` writes the files of 8.1 in one go.
- Undo and redo use one timeline shared by every editor of a project. Node positions, sizes and connection routes can be undone; panning and zoomi
- If the VFB file changes on disk while you have unsaved edits, PAIO keeps your edits and shows a warning. Saving overwrites the external change.

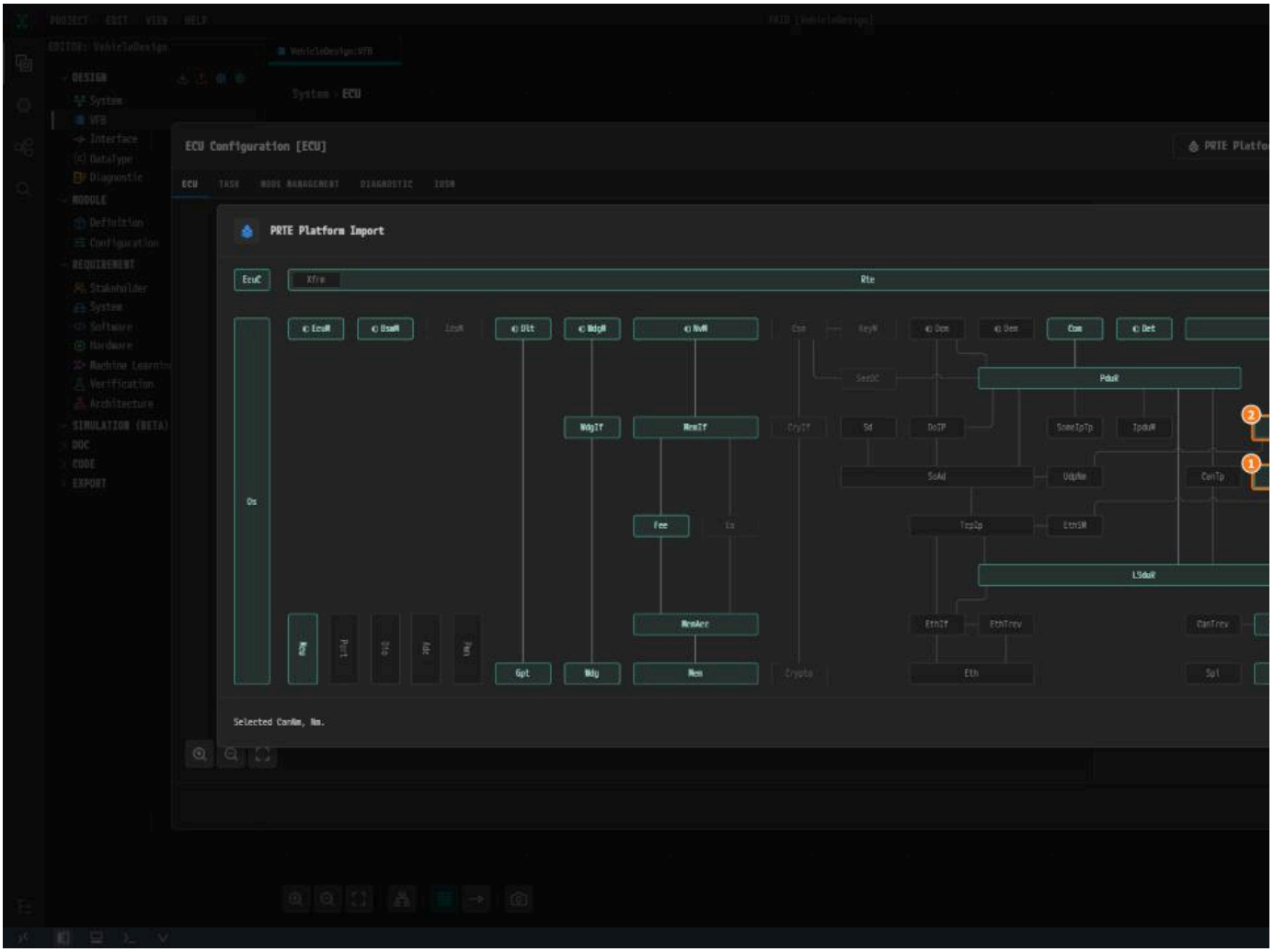
8.14 Integration with Other Editors

- The composition to ECU/HPC mapping is shared with 06. System Editor.
- Port interfaces come from 09. Interface Editor. Renaming an interface updates the VFB.
- An InterfaceMapping is the same data as the **Mapping** tab of the Interface editor.
- ARXML import and export are in 13. Import and Export, and the shortcuts in 18. Keyboard Shortcuts.

8.15 PRTE Platform Import

Brings the PopcornSAR AUTOSAR Classic Platform base set into the project for one ECU: the PRTE module definitions (VSMD), the bundled service interfac of the BSW modules you select. It requires write permission.

1. In the ECU Configuration dialog (8.12), click **PRTE Platform Import** in the header.
2. A map of the BSW modules opens. The modules the design needs are selected, and Os, Rte, EcuM, BswM and Mcu are always selected. Click a module that need it, follow, and the line at the bottom names them. Dashed modules come in a later PRTE release. **Reset to suggestion** restores the suggest
3. Click **Import**. If there is a conflict, the conflict resolution dialog opens. When it succeeds, the ECU Configuration dialog opens again.



① `CanNm`, ② `Nm`, ③ ***Import.***

Opened again, the window shows the imported modules as selected. Turn one off and click **Apply** to remove it.

The whole import is one undo step. The service components go under the ECU: `Composition/PRTE_<ECU>` and `Service/<ECU>/<Module>`. A module defir

8.16 PRTE Auto Configuration

Writes the BSW module configurations of one ECU from its design. A value the design cannot decide is left to you or to the AI. It requires write permission

1. In the ECU Configuration dialog (8.12), click **PRTE AUTO CONFIGURATION** in the header.
2. Read the preview; nothing is written yet. It lists the files by area and the **NOTES** of the run. Click a file to see what changes (12.2.6). If modules are mi **PRTE PLATFORM IMPORT** opens 8.15.
3. Click **Configure** or **Apply**. PAIO writes the files directly to disk, so no `Ctrl+S` is needed.

The whole run is one undo step. Running it again on an unchanged design writes nothing.

8.17 PRTE Code Generation

Exports the ECU Extract of one ECU and generates the PRTE BSW configuration code from it. It requires write permission.

1. Click **PRTE Code Generation** on the **DESIGN** header of the **EDITOR** view.
2. Choose the **Target ECU**.
3. Click **Generate**. A validation error stops the generation (14. Validation).
4. The ECU Extract goes to `export/<ECU>/` and the code to `<project>/code/<ECU>/`.

It runs on the host; in an SSH window PAIO copies the project to the PC, generates there and writes the result back to the server.

8.18 PARA Code Generation

Exports the Machine Extract of one HPC and generates the PARA (AUTOSAR Adaptive Platform) code from it. It requires write permission.

1. Click **PARA Code Generation** on the **DESIGN** header of the **EDITOR** view.
2. Choose the **Target HPC**.
3. Click **Generate**. A validation error stops the generation.
4. The Machine Extract goes to `export/<HPC>/` and the code to `<project>/code/<Machine>/`, one folder per Machine.

A rerun overwrites the files the generator owns and keeps the files you added. You build and run the result in the PARA vECU (17. Remote Development (SS
SSH window and writes the result back to the server.

8.19 Intrusion Detection System Dialog

The second icon of the action strip opens this dialog. It edits `design/IDS.io.model`, which has no editor of its own.

- **IDSM INSTANCES**: one IdsM instance per ECU or HPC node, with its limitation, filter chain, signature and context mapping sections.
- **SECURITY EVENTS**: the security events of the whole system. **HANDLED BY** lists each IdsM instance that reports the event.

Save with `Ctrl+S`; the save is one undo step.

8.20 Software Cluster Dialog

The third icon of the action strip opens this dialog. It edits `design/SOFTWARE_CLUSTER.io.model`: the software clusters, software packages and vehicle
the **VEHICLE PACKAGE** tab.

- **SOFTWARE CLUSTER**: `+` adds a cluster to a node. The sub-tabs are **CLUSTER**, **PACKAGE**, **LICENSE** and **RELEASE NOTES**, and **CONTENTS** shows read-only v
- **VEHICLE PACKAGE**: the sub-tabs are **CONFIGURATION**, **VEHICLE DESCRIPTION** and **ROLLOUT STEP**.

Save with `Ctrl+S`; the save is one undo step.

SYSTEM Panel

What This Chapter Covers

This chapter explains the SYSTEM panel, where you read and edit the information of the SYSTEM element. The panel has no tab of its own. It opens in the

- In the System editor (tab `<project>:System`) it shows **NAME**, **VERSION**, **DESCRIPTION** and the read-only **ECU MAPPING** list of the SWC ↔ ECU mapping
- In the VFB editor (tab `<project>:VFB`) it shows **NAME**, **VERSION**, **DESCRIPTION**, **ROOT COMPOSITION** and the read-only **COMPOSITIONS** tree of the comp

ECU and HPC nodes are created in 06. System Editor, and compositions in 08. VFB Editor.

11.1 Opening the Panel and Save Location

1. In the **EDITOR** view, click **System** (or **VFB**) under **DESIGN**.
2. Click the **System** button on the right edge of the canvas. It is the third button, below the panel toggle and the lock.
3. The property panel opens with the title **SYSTEM**.

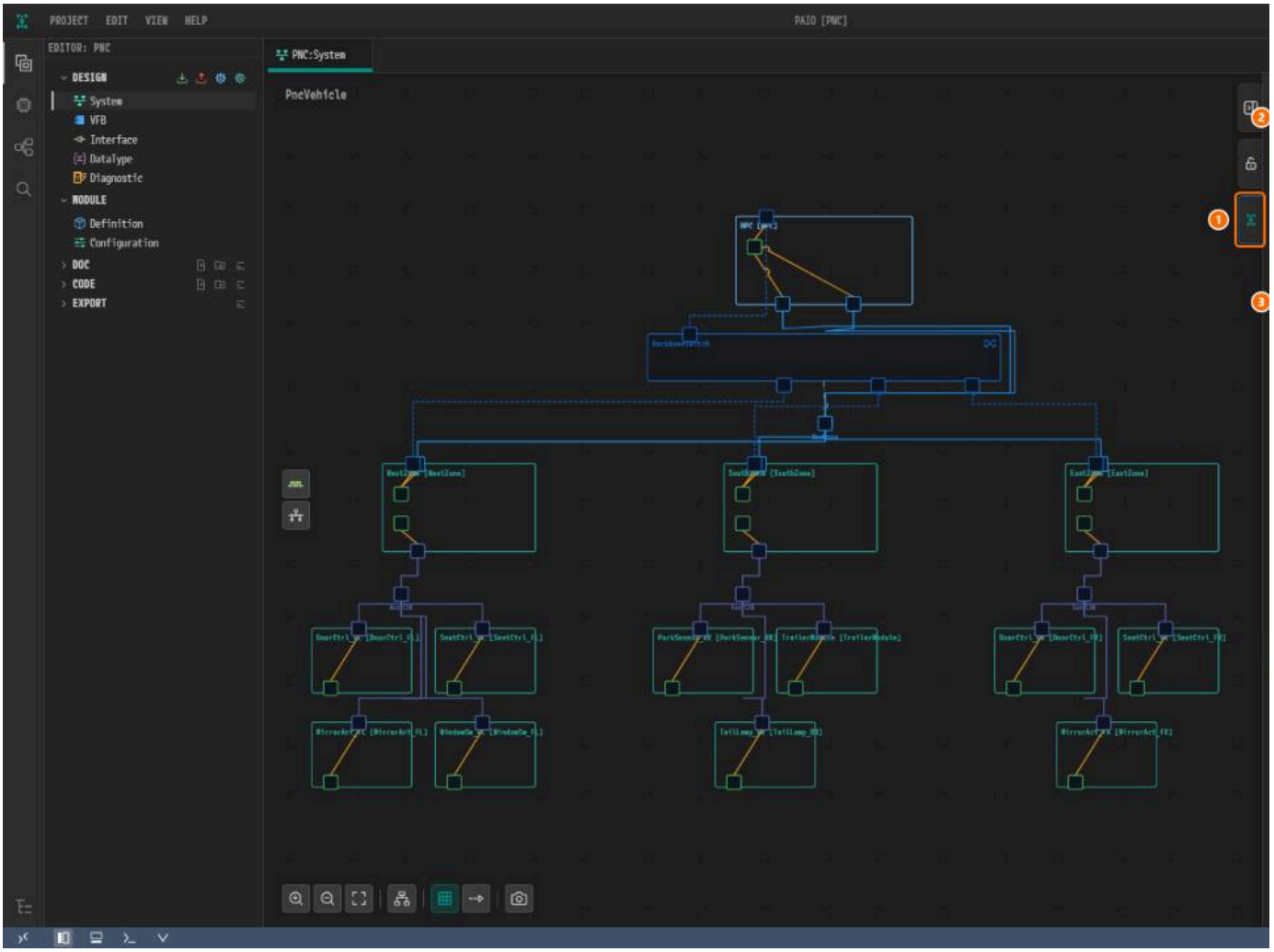


*① The panel toggle, ② the lock, ③ the **System** button.*

There is no save step. Each change is written to `design/SYSTEM.io.model` at once.

11.2 Screen Layout

The fields sit at the top of the panel. The list (System editor) or the tree (VFB editor) sits below them.



*① The **System** button, ② **NAME**, **VERSION** and **DESCRIPTION**, ③ **ECU MAPPING**, one row per ECU or HPC node.*

- Select a node and the panel switches to that node. Click **System** again to return.
- When the panel is not locked, clicking empty canvas collapses it.

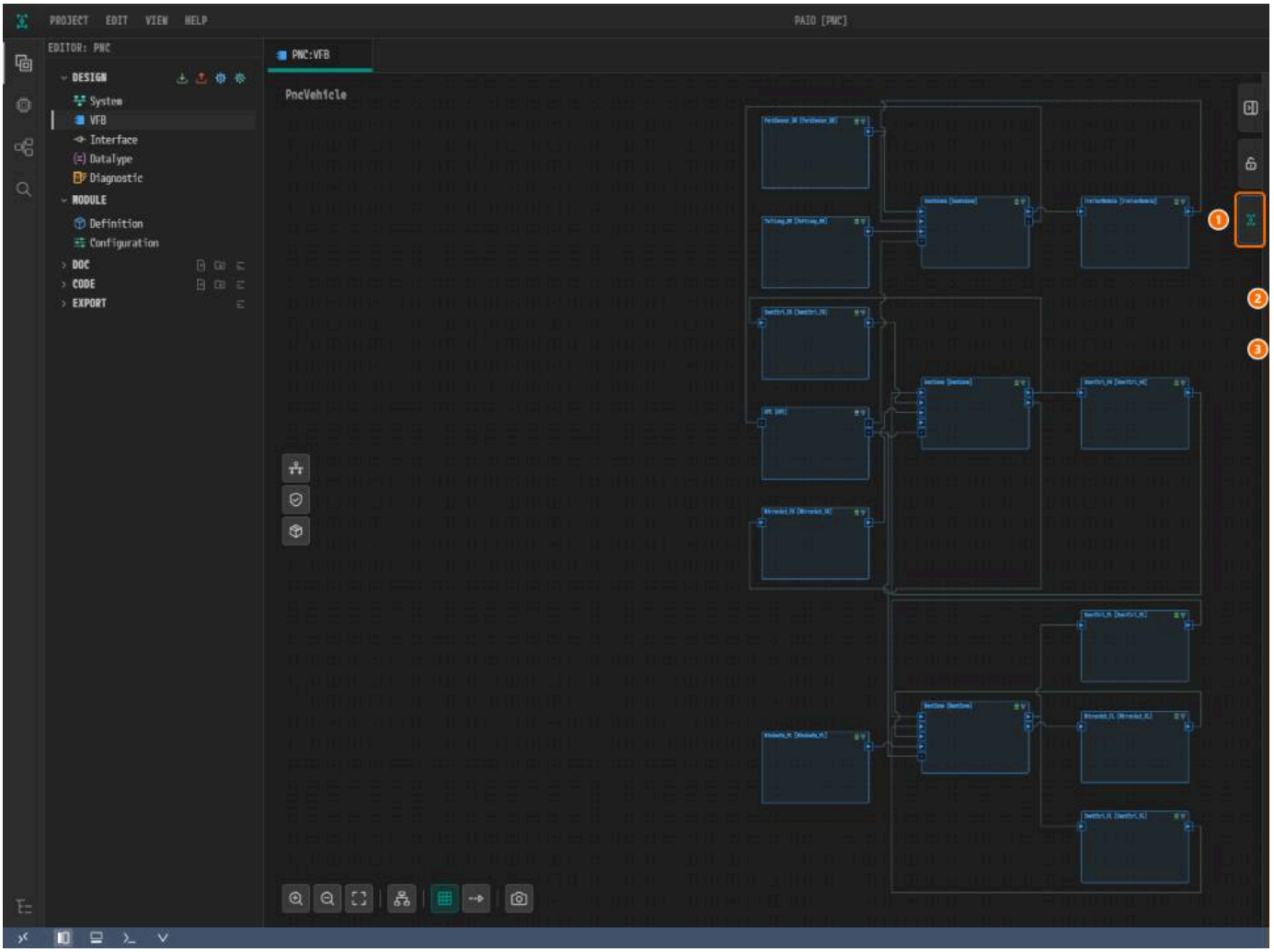
11.3 System Information

1. Enter **NAME**, the SHORT-NAME of the SYSTEM element. It commits on `Enter` or when the field loses focus.
2. Enter **VERSION** in `major.minor.patch` form, such as `1.0.0`.
3. Enter **DESCRIPTION** as free text.

- An empty or rule-breaking **NAME** is not written. The field returns to the stored name and shows an error.
- A wrong **VERSION** shows a warning, but the value is still written.

11.4 Compositions Tree (VFB Editor)

The **COMPOSITIONS** tree in the VFB editor's panel shows the composition hierarchy. It is read-only. Add, delete and rename compositions in 08. VFB Editor



*① The **System** button, ② **ROOT COMPOSITION**, ③ the **COMPOSITIONS** tree.*

11.4.1 Row Contents

Each row shows an icon for the SWC kind, the prototype name, and the SWC type name in italics when it differs.

11.4.2 Row Kinds

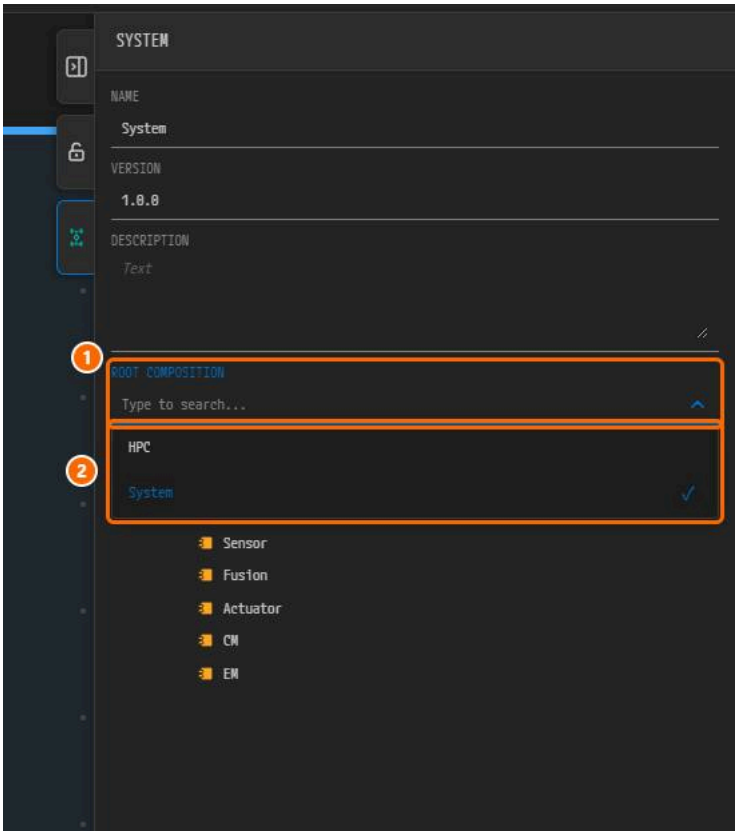
The tree lists compositions and their prototypes. Hover over a row's icon to read its kind, such as `COMPOSITION`.

11.4.3 Navigating from the Tree

Click a composition row to drill the VFB canvas into that composition. Names cannot be edited in the tree.

11.4.4 Setting the Root Composition

1. Click **ROOT COMPOSITION**. The candidate list opens, with the current root checked.
2. Select a composition. The VFB model is restructured around the new root.



*① **ROOT COMPOSITION** with its search field, ② the candidates, with the current root checked.*

- Save the VFB editor first. With unsaved edits, the pick is refused.
- One `Ctrl+Z` restores the previous root.

11.4.5 Deploying to an ECU or HPC

Deployment is not edited in this panel. Use one of these menus. The result appears in **ECU MAPPING** (11.5).

- **System editor**: right-click an ECU or HPC node ▸ **SW Mapping**.
- **VFB editor**: right-click a top-level composition ▸ **ECU/HPC Mapping**.

11.5 ECU Mapping

ECU MAPPING in the System editor's panel lists the ECU and HPC nodes with the composition mapped to each. It is read-only. Click a row to select that nc
`No ECU or HPC in the diagram`.

11.6 Statistics (Removed)

The statistics panel of the former System dashboard has been removed. Rule checks are in 14. Validation.

11.6.1 System Completeness

Removed. Read **ECU MAPPING** (11.5) instead.

11.6.2 Architecture Quality

Removed. Use 14. Validation instead.

11.7 Saving, Undo, and Shortcuts

- There is nothing to save. Each change is already written.
- Press `Ctrl+Z`, or use the **EDIT** menu, to undo a change.
- **VERSION** is written on every keystroke, so undo steps back one keystroke at a time.

11.7.1 External Changes and Conflicts

The panel has no unsaved state, so there is no conflict warning. Each write changes only its own field, so the two SYSTEM panels do not overwrite each o

11.8 Integration with Other Editors

- Both SYSTEM panels edit the same system information.
- **ECU MAPPING** comes from the System editor canvas, and the **COMPOSITIONS** tree from the VFB diagram.
- Data mapping (port data element ↔ signal) is recorded in the System model but edited in 08. VFB Editor.

11.9 Common Situations

Symptom	What to do
The SYSTEM panel disappeared	Click **System** again, or lock the panel
A **NAME** returns to the old name	Read the error below the field
A **ROOT COMPOSITION** pick is refused	Save the VFB editor, then pick again

Interface Editor

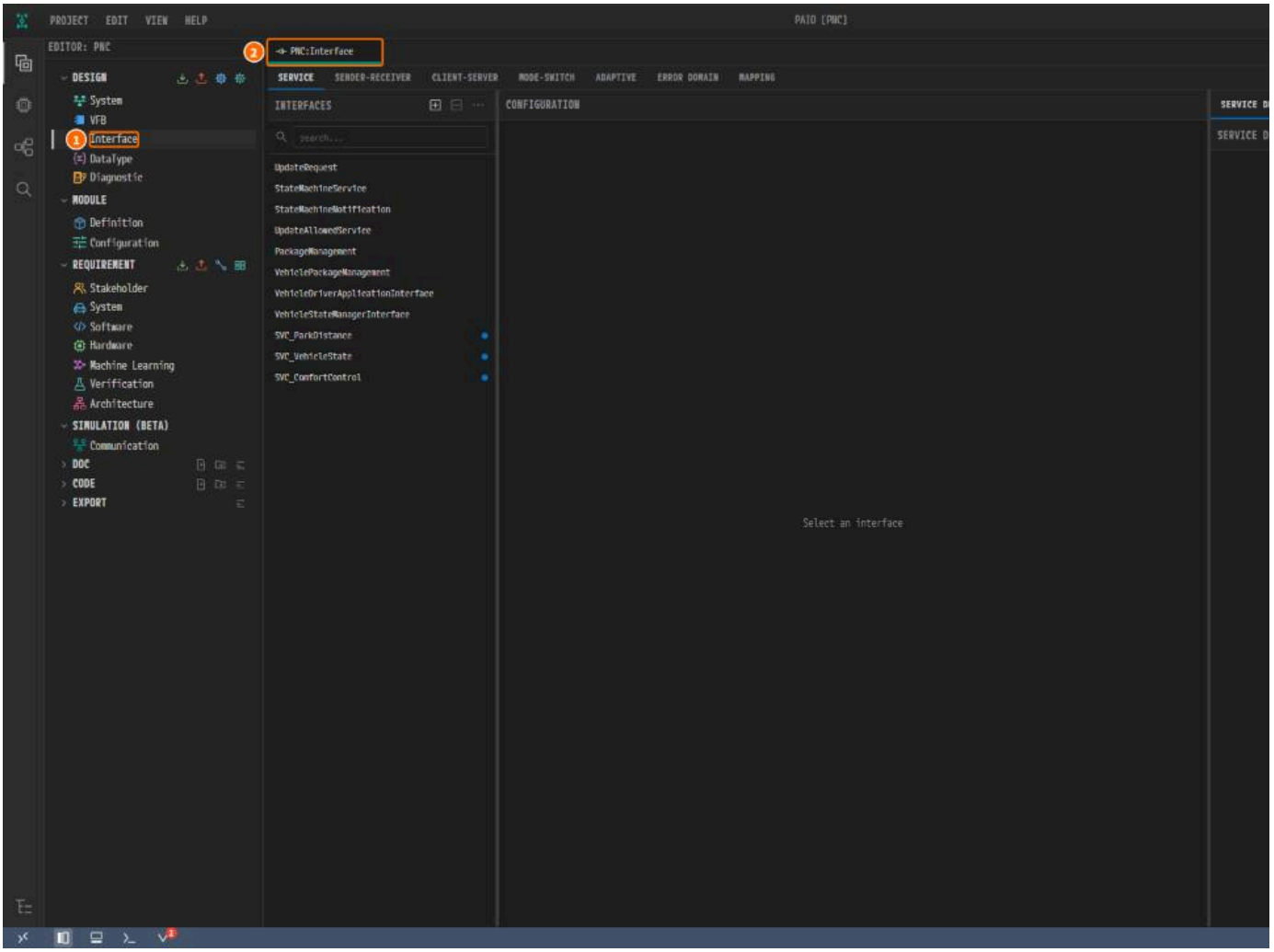
What This Chapter Covers

This chapter explains how to use the Interface editor. It defines the port interfaces that SWC ports reference, and the deployments (SOME/IP, DDS, IPC) a

- Audience: users who define AUTOSAR interfaces and service deployments
- Prerequisites: a project is open; derivation also needs an active license

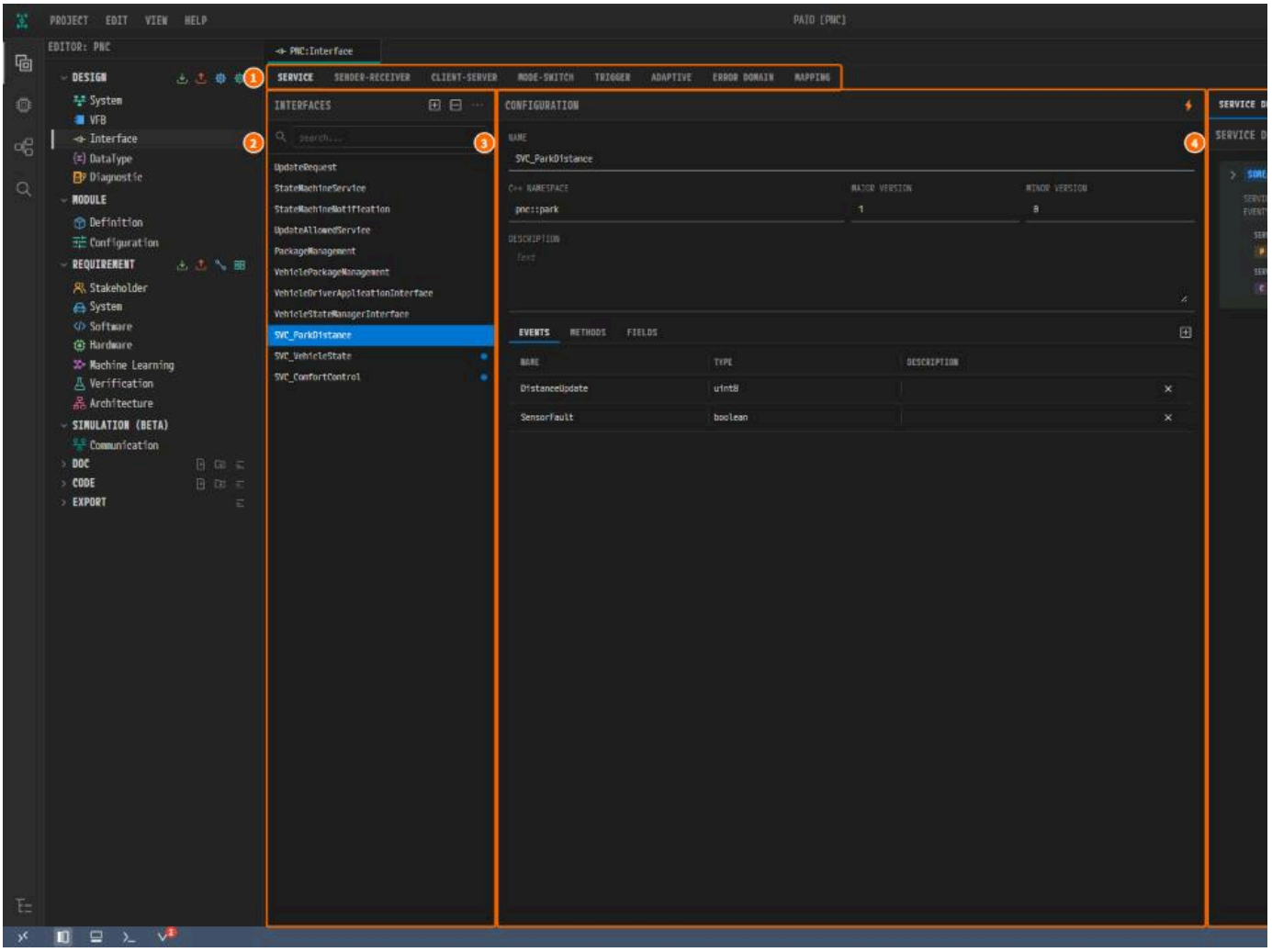
You can define data types inside this screen, so you do not need to open 10. DataType Editor first.

To open the editor, click **Interface** under **DESIGN** in the **EDITOR** view on the left. The tab `<project>:Interface` opens. Do not open `design/INTERFACE.i` Explorer: there they open as a read-only JSON view.



*① **Interface** under **DESIGN**. ② The tab `PNC:Interface`.*

9.1 Screen Layout

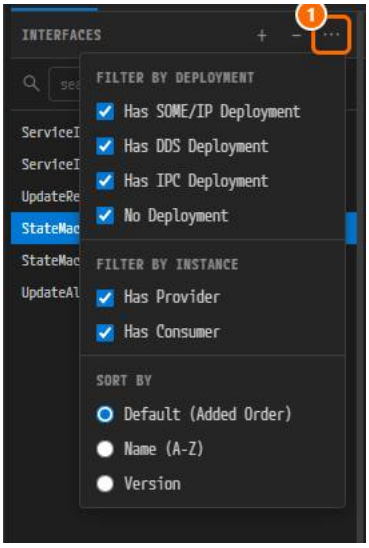


*① The eight category tabs. ② The **INTERFACES** panel. ③ The **CONFIGURATION** panel. ④ The right panel.*

Drag the borders between the three panels to adjust their widths.

The INTERFACES Panel on the Left

- `+` adds an item that suits the current tab, and `-` deletes the selected items after a confirmation dialog.
- `···` opens the sort and filter popover, and the search box filters by name.
- `Ctrl`+click adds a row to the selection, and `Ctrl+A` selects every row.



① *The* `⋯` *(**Sort & Filter**) button on the Service tab.*

The CONFIGURATION Panel in the Middle

Shows the fields of the selected interface. The ⚡ button in its header performs derivation (9.5).

Interfaces are stored in the Interface model, and service deployments and service instances in the Service model. One `Ctrl+S` writes both together with exception: it goes to the VFB model as soon as you edit it (9.3.7).

9.2 The Eight Category Tabs



*The eight category tabs, with **SERVICE** active.*

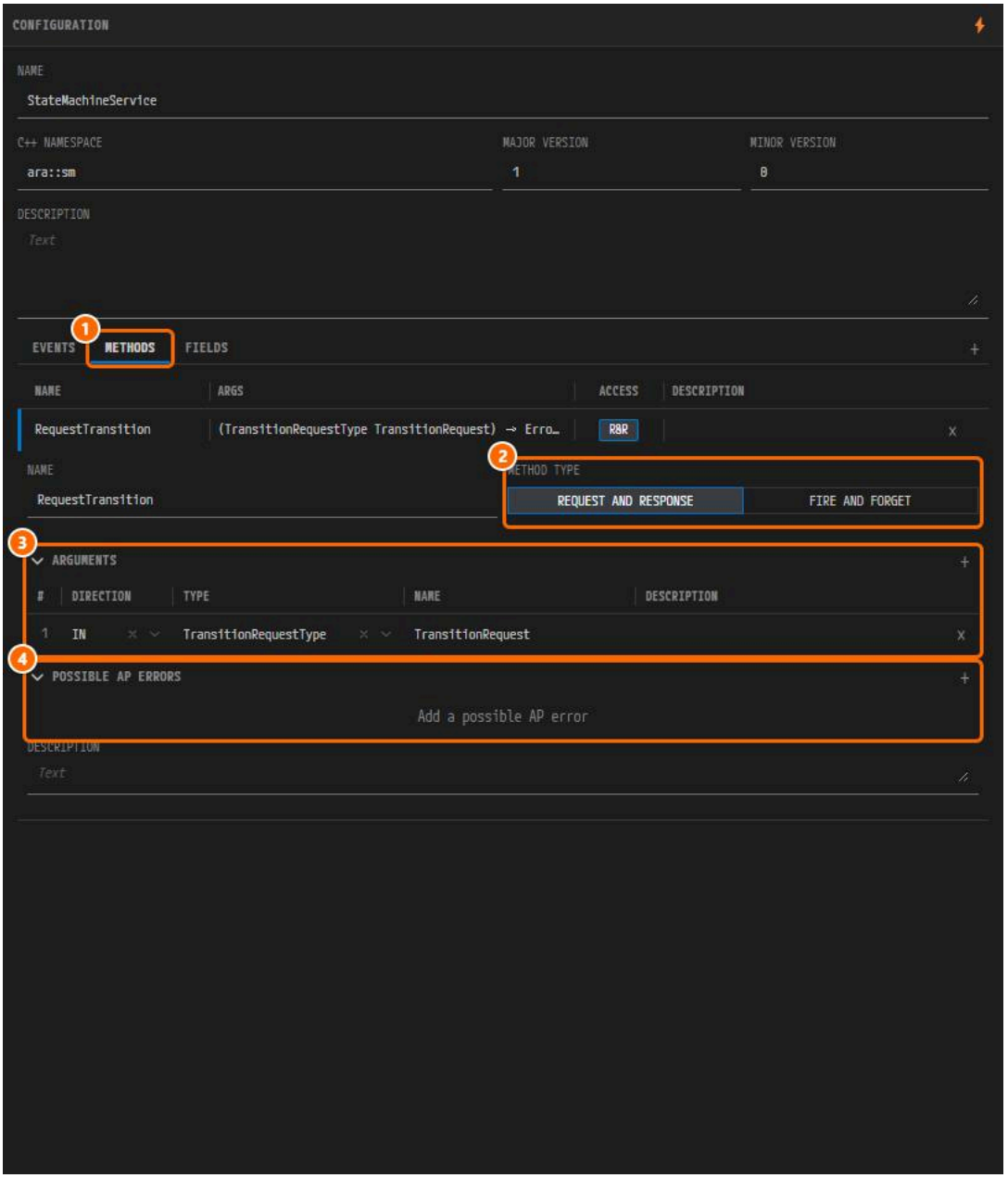
Tab label	What it covers
SERVICE	AP ServiceInterface + service deployment + service instances
SENDER-RECEIVER	CP SenderReceiverInterface
CLIENT-SERVER	CP ClientServerInterface
MODE-SWITCH	CP ModeSwitchInterface
TRIGGER	CP TriggerInterface
ADAPTIVE	Dedicated port interfaces grouped by AP functional cluster
ERROR DOMAIN	AP error domains
MAPPING	The InterfaceMapping that links an AP service interface to a CP interface

9.3 What You Edit on Each Category

Every tab has **NAME** and a free-text **DESCRIPTION**. The sections below cover what differs. Naming follows 9.8.

9.3.1 Service

1. On the **SERVICE** tab, click `+` and select the new interface.
2. Set **NAME**, **C++ NAMESPACE**, **MAJOR VERSION** and **MINOR VERSION**.
3. Open the **EVENTS**, **METHODS** or **FIELDS** sub-tab and click `+` in the header to add an element.
4. Expand a method row to set its **METHOD TYPE**, **ARGUMENTS** and **POSSIBLE AP ERRORS**.

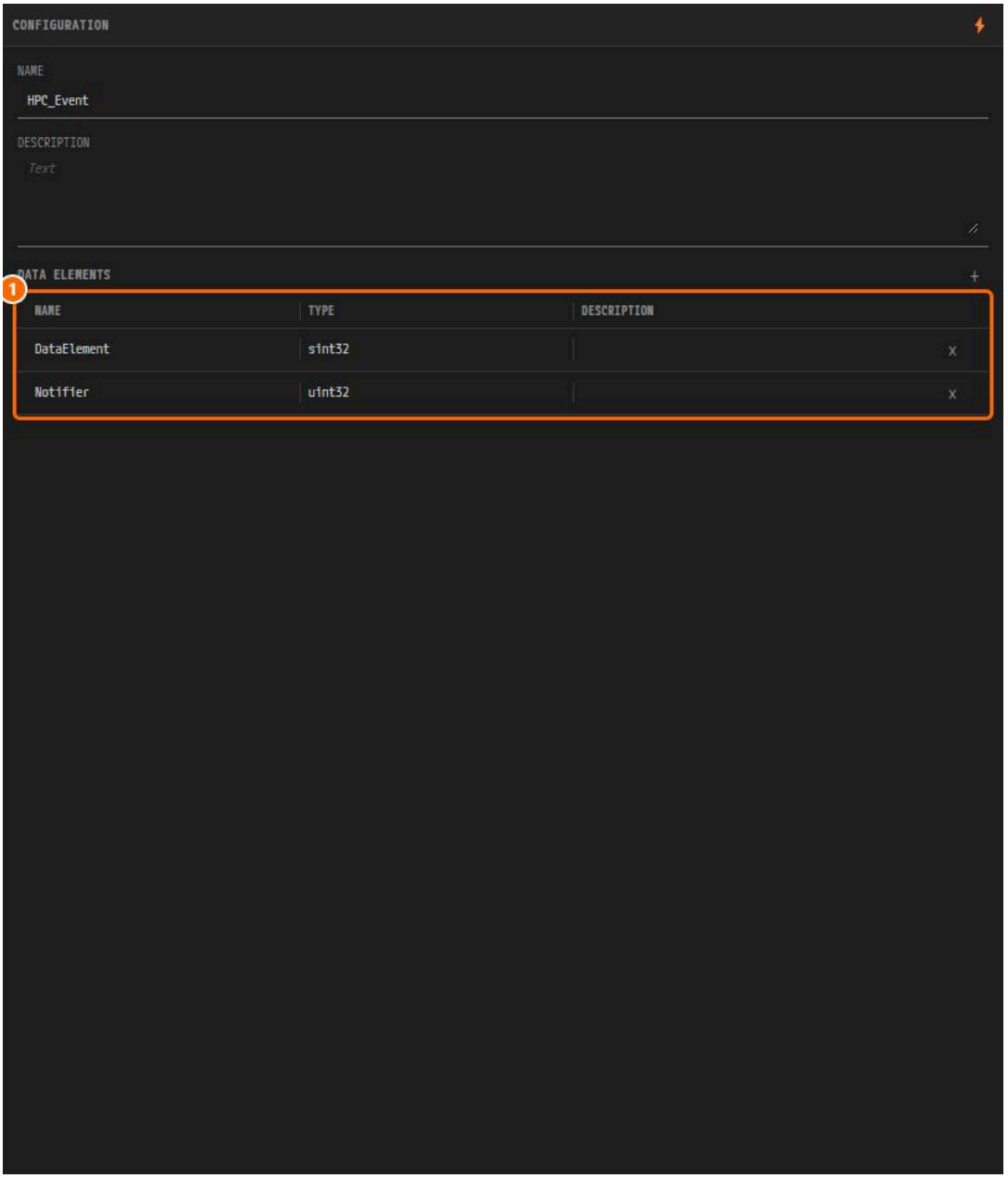


*① The **METHODS** sub-tab. ② **METHOD TYPE**. ③ **ARGUMENTS**. ④ **POSSIBLE AP ERRORS**.*

- **POSSIBLE AP ERRORS** references errors from the Error Domain tab, so create the domain first.
- Drag the `#` cell of an argument row to reorder the arguments.

9.3.2 Sender-Receiver

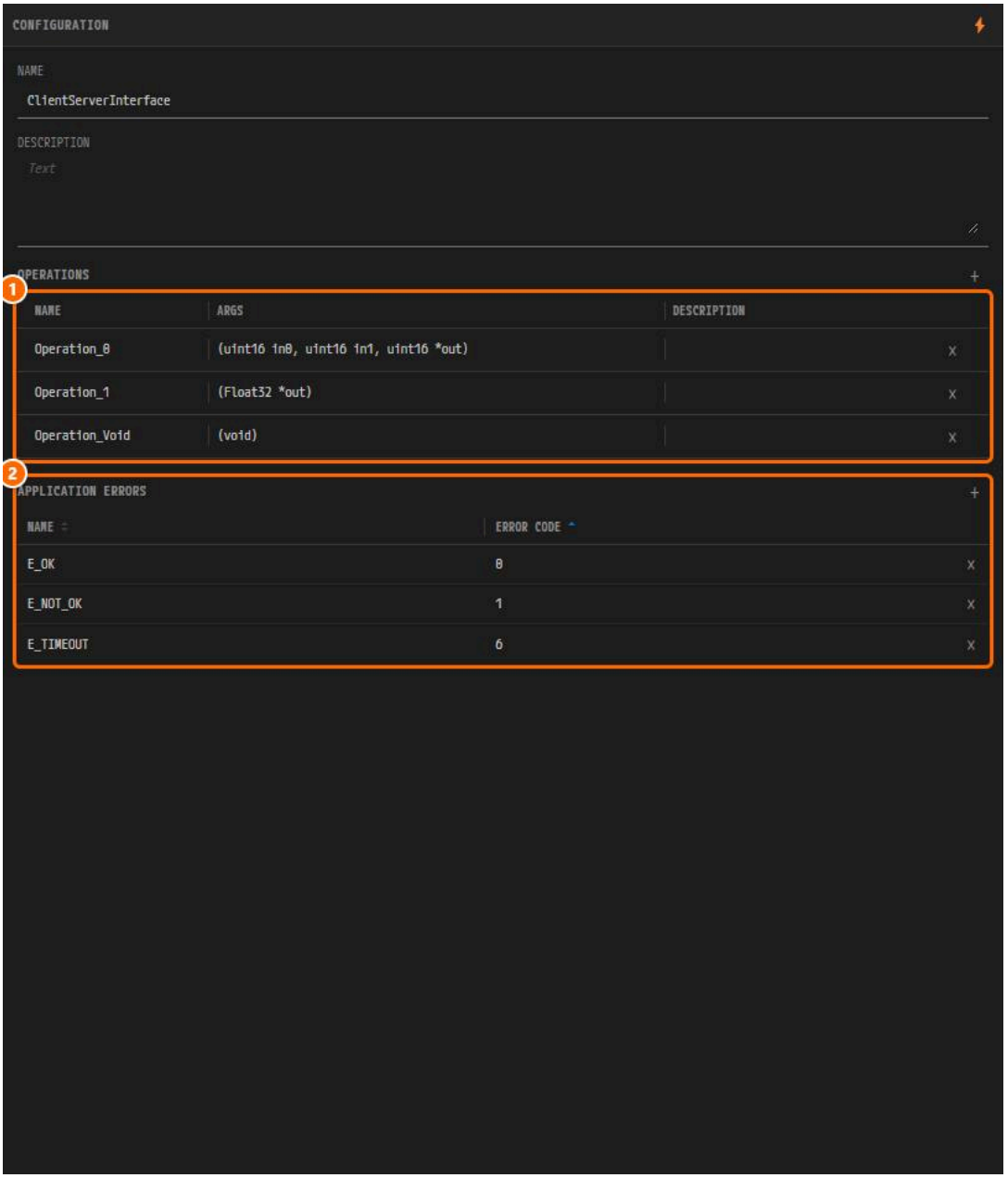
Add data elements to the **DATA ELEMENTS** table and give each a **NAME** and a **TYPE**.



*① The **DATA ELEMENTS** table.*

9.3.3 Client-Server

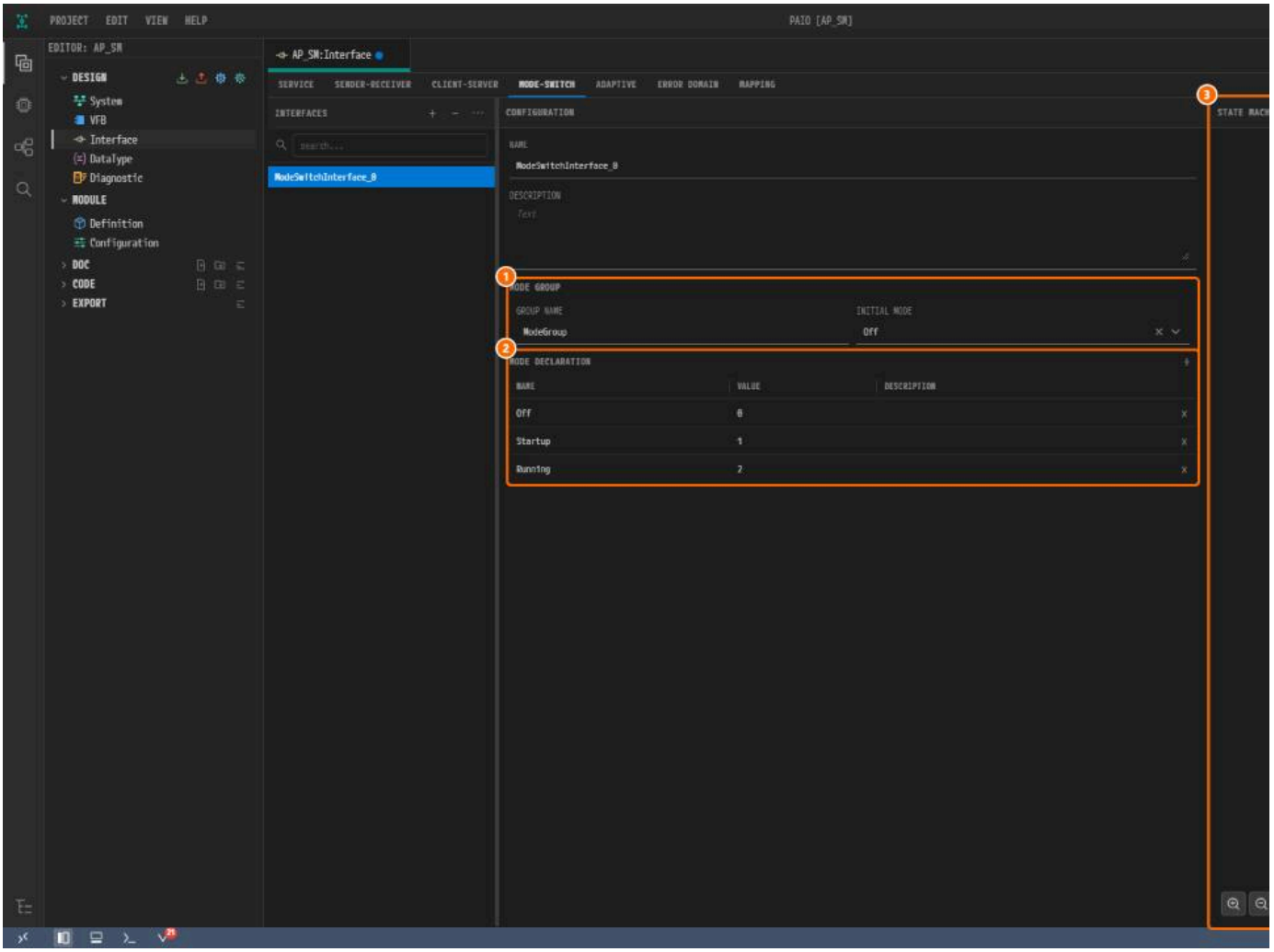
Add operations to **OPERATIONS** and expand one to set its **ARGUMENTS** (**IN** / **OUT** / **INOUT**). **APPLICATION ERRORS** below applies to the whole interface.



*① **OPERATIONS**, with each operation as a C parameter list in **ARGS**. ② **APPLICATION ERRORS**.*

9.3.4 Mode-Switch

1. Add mode declarations with `+` in **MODE DECLARATION**.
2. Pick the **INITIAL MODE** in **MODE GROUP**.
3. Draw the transitions in the **STATE MACHINE** diagram on the right: `Shift` + drag one node onto another. Transitions are not created automatically.

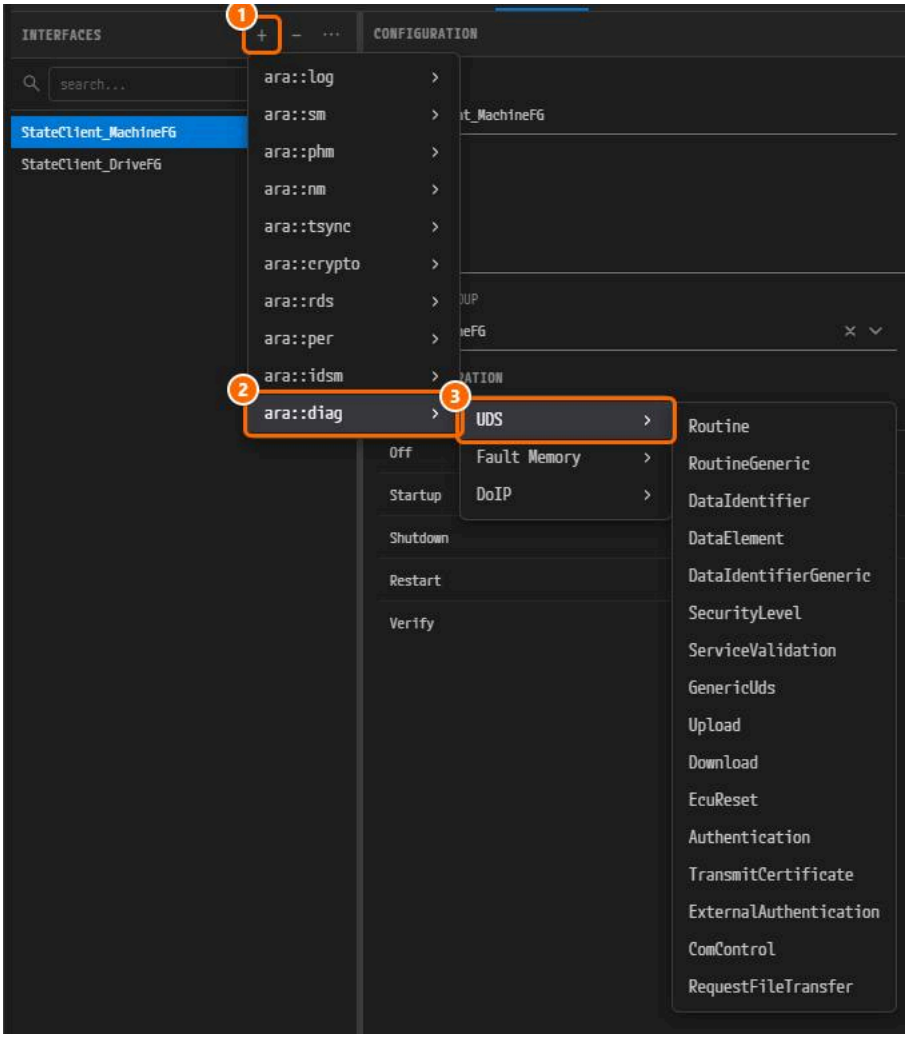


*① **MODE GROUP** with **INITIAL MODE** `Off`. ② **MODE DECLARATION**. ③ The **STATE MACHINE** diagram.*

9.3.5 Adaptive

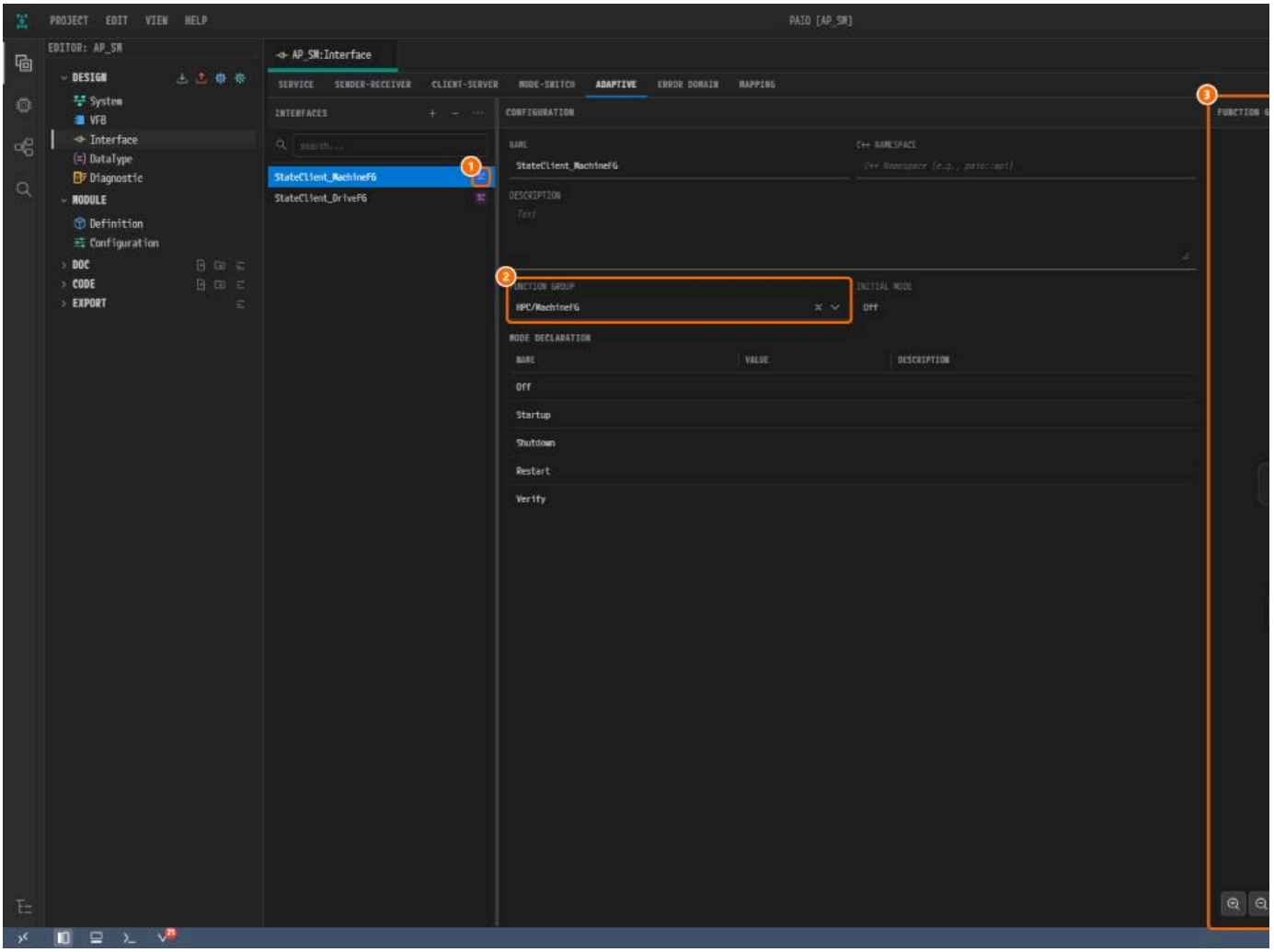
Creates the dedicated interfaces of the AP functional clusters.

1. Click `+`. The list of ten clusters opens.
2. Hover over a cluster and click an interface kind. **ara::diag** has one more level: **UDS**, **Fault Memory**, **DoIP**.



*① The + button. ② **ara::diag**. ③ The **UDS** submenu.*

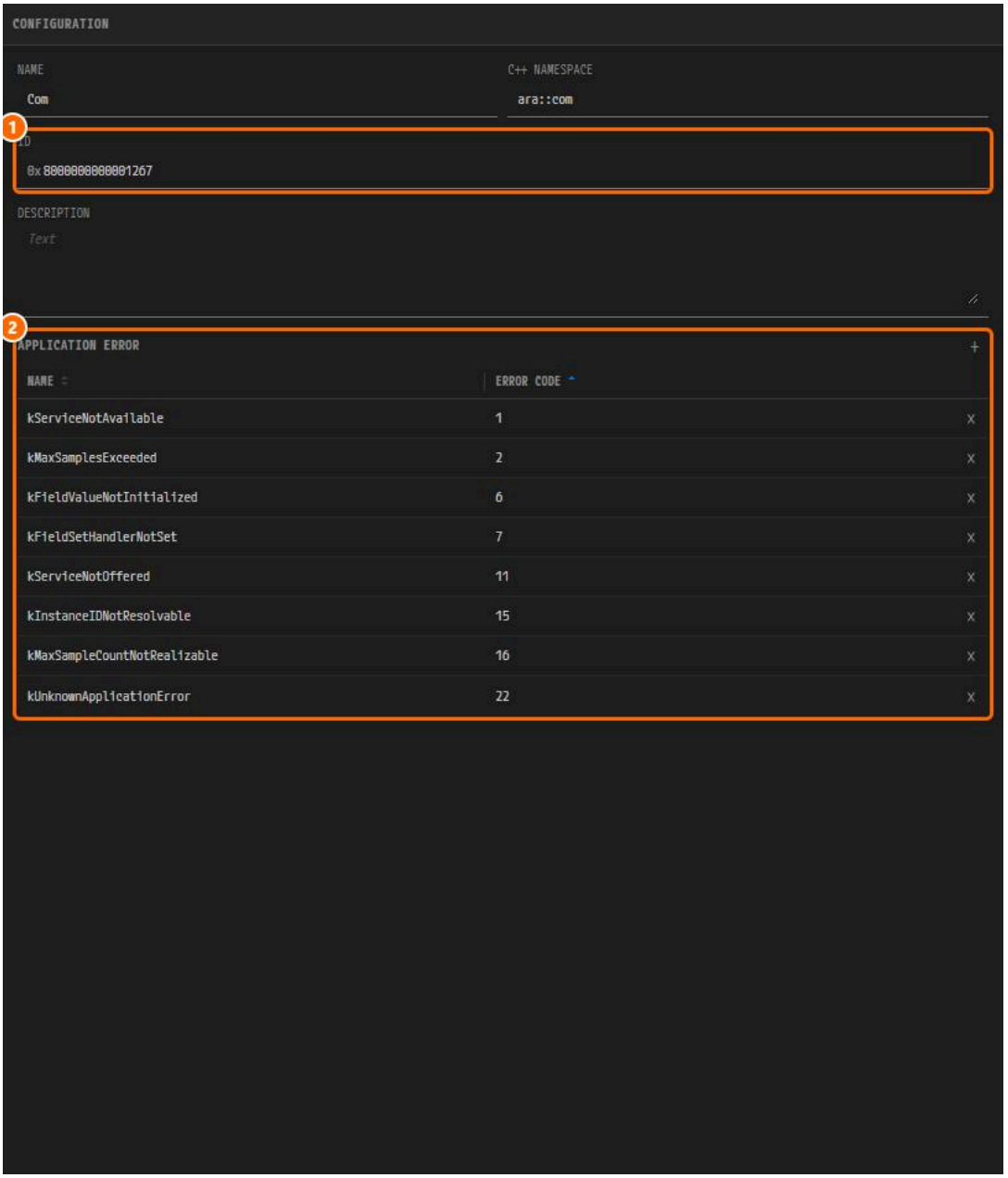
A badge after each name shows the kind. For a **StateClient**, pick the **FUNCTION GROUP** defined in the Machine Configuration dialog; the right panel then sl



*① The `SC` badge. ② **FUNCTION GROUP** set to `HPC/MachineFG`. ③ The **FUNCTION GROUP STATE MACHINE** viewer.*

9.3.6 Error Domain

Set the domain **ID** in hexadecimal (the `0x` prefix is fixed) and add errors to **APPLICATION ERROR**, each with a `NAME` and an `ERROR CODE`.

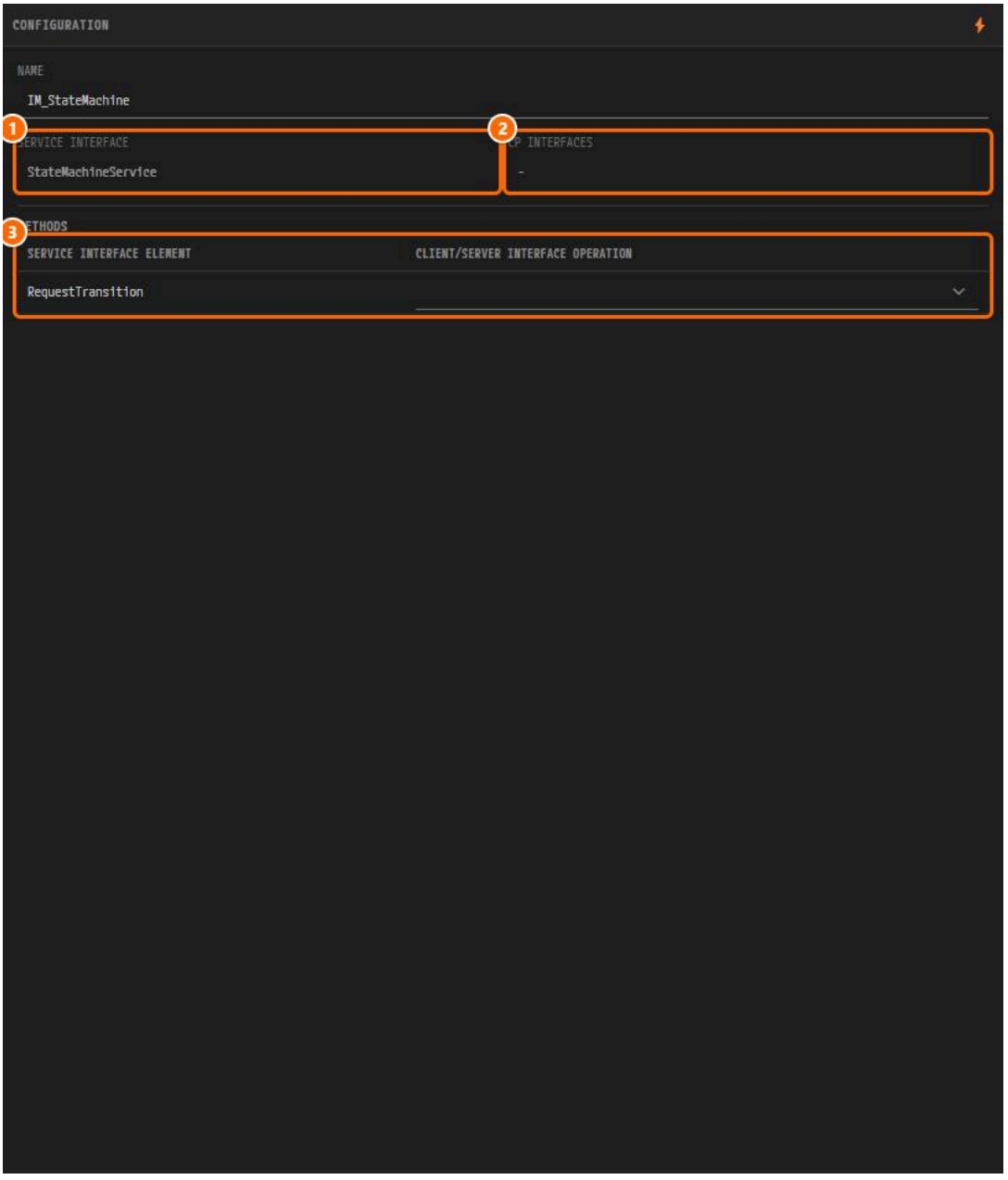


*① **ID**. ② **APPLICATION ERROR**.*

9.3.7 Mapping

Defines the InterfaceMapping that links an AP service interface to a CP interface.

1. Click `+`. In the first **Add Interface Mapping** dialog enter a name; in the second pick a service interface or leave it empty.
2. Select the mapping. **SERVICE INTERFACE** and **CP INTERFACES** are read-only.
3. In the **EVENTS**, **METHODS** and **FIELDS** sections, pick the CP element for each service element in the right column.



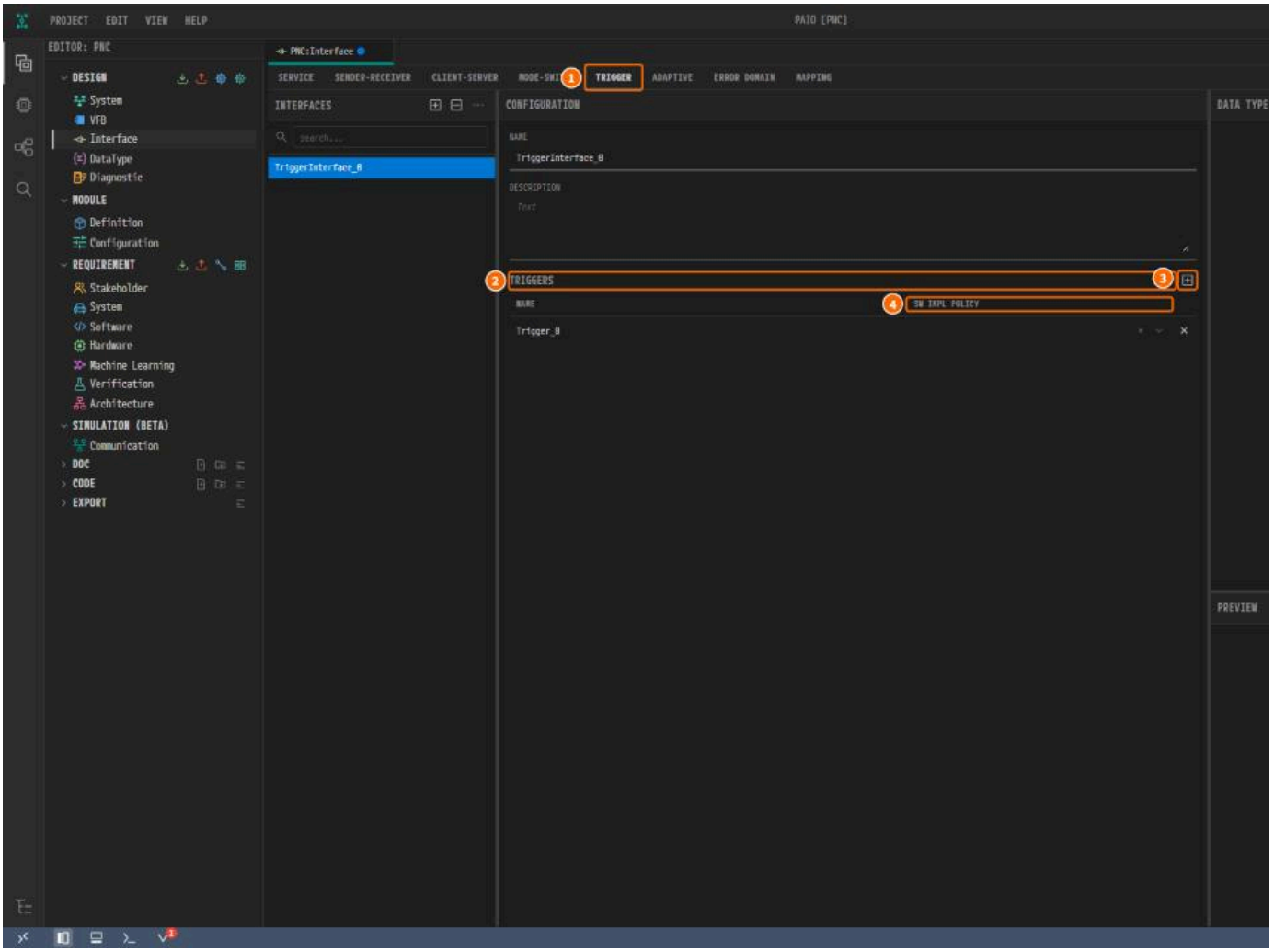
*① **SERVICE INTERFACE**. ② **CP INTERFACES**. ③ The **METHODS** section.*

InterfaceMapping is saved to the VFB model, not the Interface model, and each edit takes effect at once.

9.3.8 Trigger

Defines a trigger interface, which a Classic SW-C port can use to fire or receive triggers.

1. On the **TRIGGER** tab, click `+` and select the new interface.
2. In **TRIGGERS**, click `+` (**Add Trigger**) to add a trigger.
3. Give the trigger a **Name** and, if needed, a **Sw Impl Policy** (`STANDARD` or `QUEUED`).



*① The **TRIGGER** tab. ② **TRIGGERS**. ③ `+` (**Add Trigger**). ④ **Sw Impl Policy**.*

Trigger names must be unique within the interface. In the VFB editor a port typed by a trigger interface shows a trigger icon (see 8.8).

9.4 The Right Panel

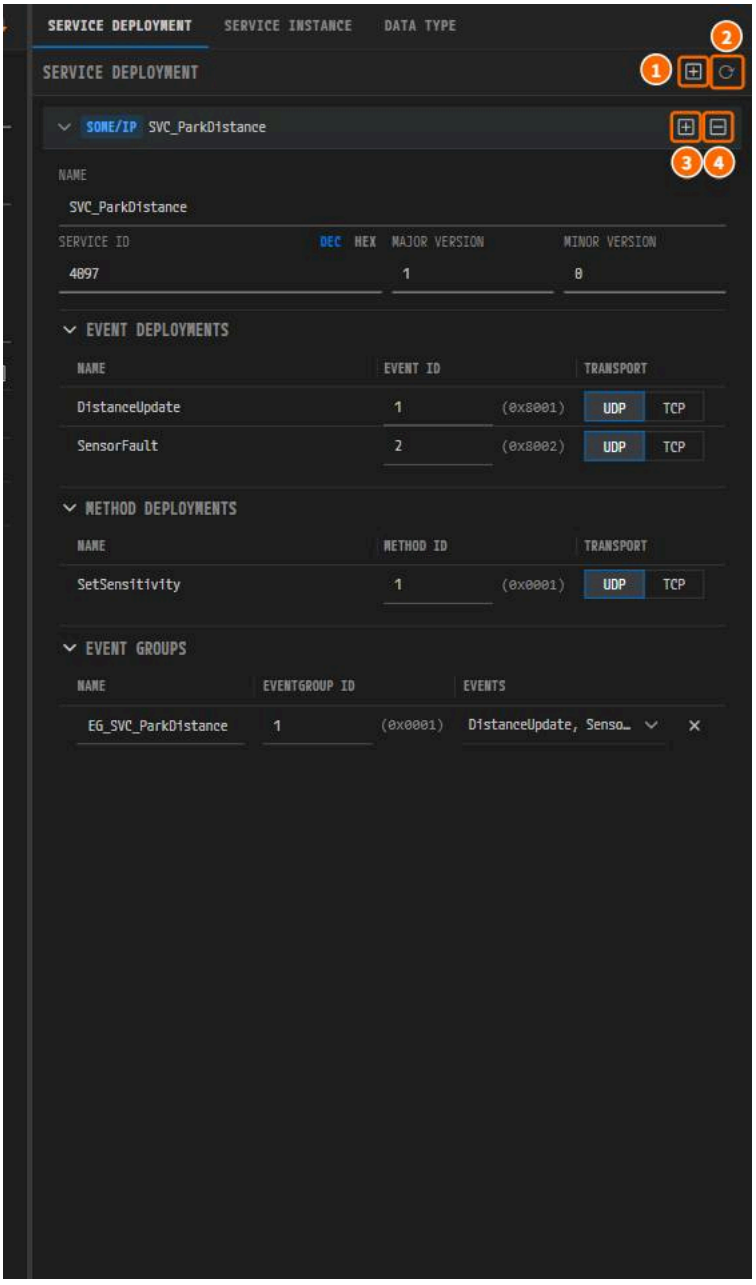
The right panel changes with the tab and the selection.

Situation	Right panel
The **SERVICE** / **SENDER-RECEIVER** / **CLIENT-SERVER** tabs	**SERVICE DEPLOYMENT** / **SERVICE INSTANCE** / **DATA TYPE** sub-tabs
The **MODE-SWITCH** tab	The **STATE MACHINE** diagram
The **ADAPTIVE** tab with a StateClient selected	The **FUNCTION GROUP STATE MACHINE** viewer (read-only)
Every other case	The embedded DataType editing panel

On the Sender-Receiver and Client-Server tabs, the sub-tabs work on the ServiceInterface mapped to the selected CP interface.

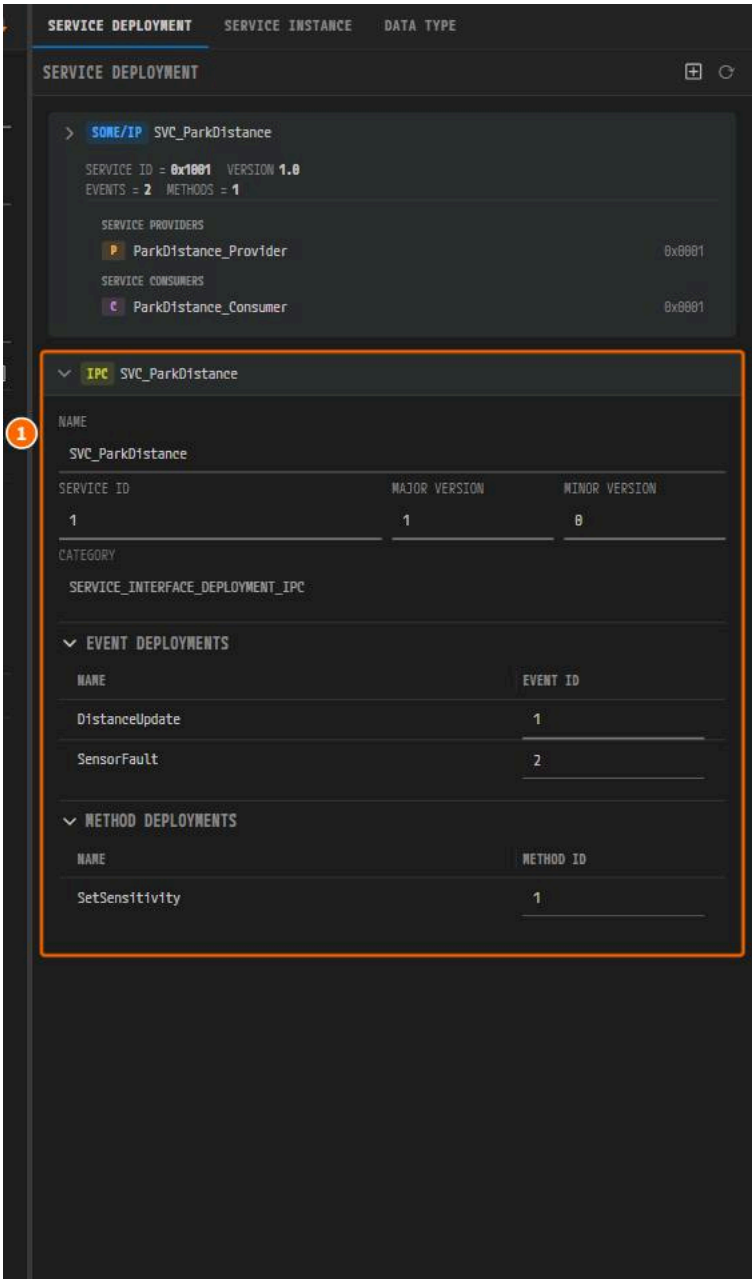
9.4.1 SERVICE DEPLOYMENT

1. Select a service interface and open **SERVICE DEPLOYMENT**.
2. Click `+` (**Add Deployment**) and pick **SOME/IP**, **DDS** or **IPC**. A deployment card appears.
3. Fill in the card, such as **SERVICE ID** and the versions.



*① `+` **(Add Deployment)**. ② `⟳` **(Derive Signal)**. ③ ④ The card header's `+` **(Add service instance)** and `-` **(Remove deployment)**.*

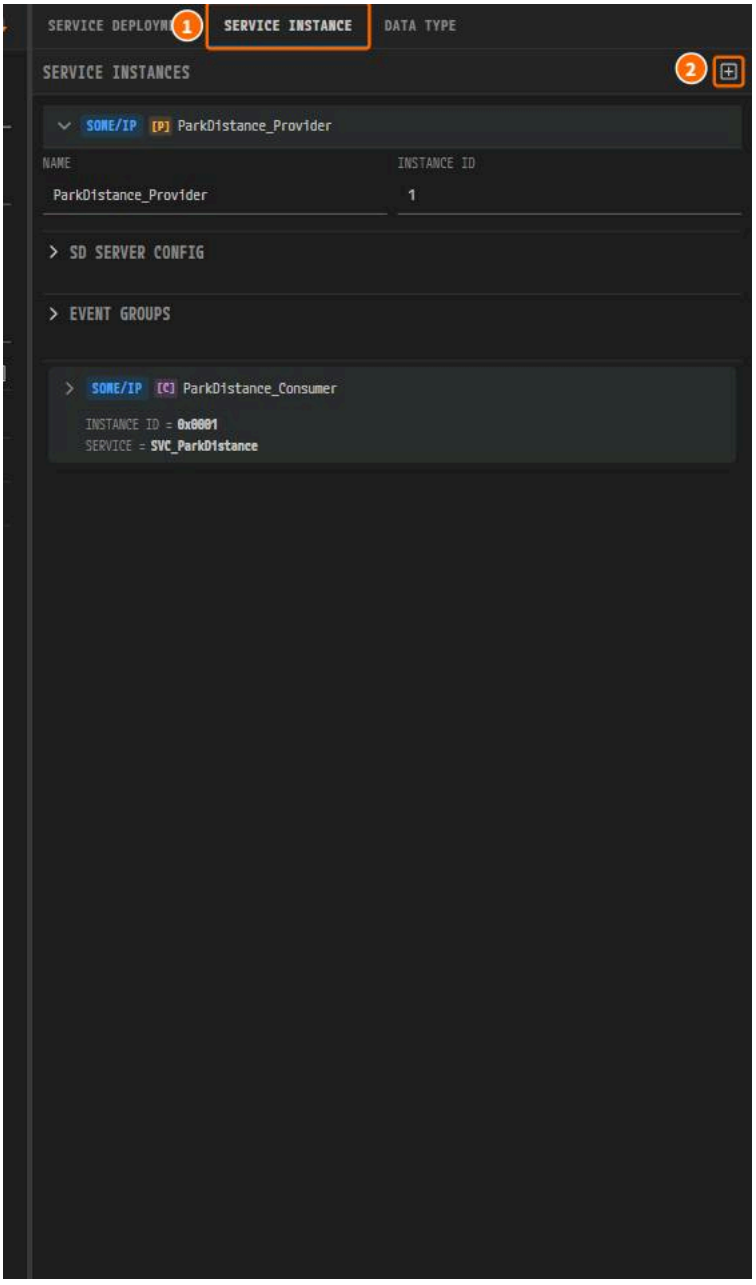
- `⟳` runs CP-SOA derivation again by hand. It normally runs automatically.
- Removing a deployment also removes the service instances that use it.



① The new IPC card.

9.4.2 SERVICE INSTANCE

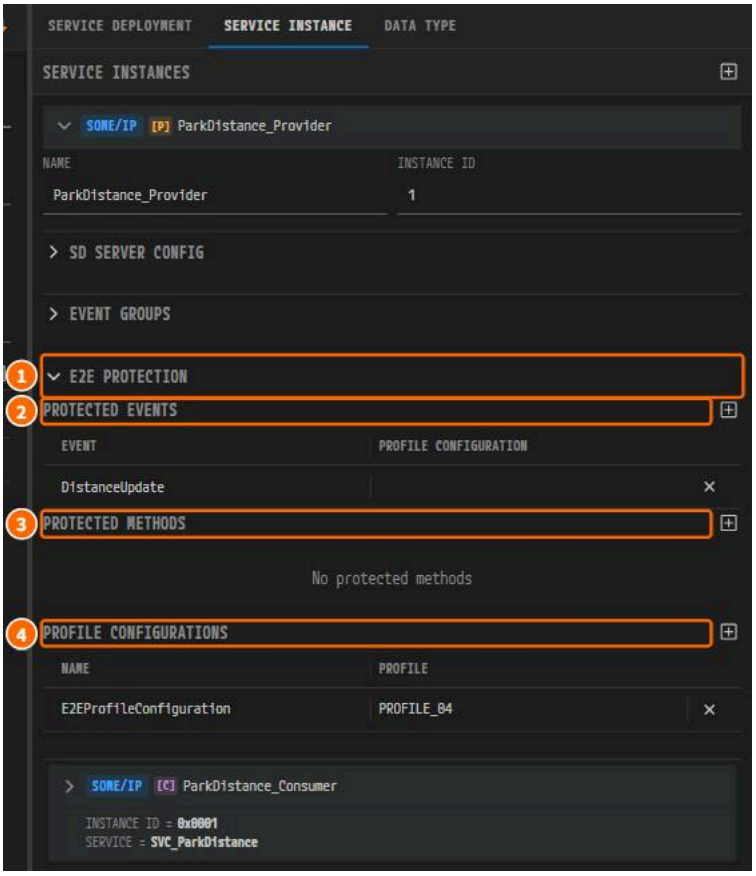
Click `+` and pick a provider or consumer of a deployment, for example **SOME/IP Provider**. A provider carries `[P]` and a consumer `[C]`.



*① The **SERVICE INSTANCE** sub-tab. ② The `+` that adds an instance.*

A SOME/IP provider or consumer card also has an **E2E PROTECTION** section for end-to-end protection:

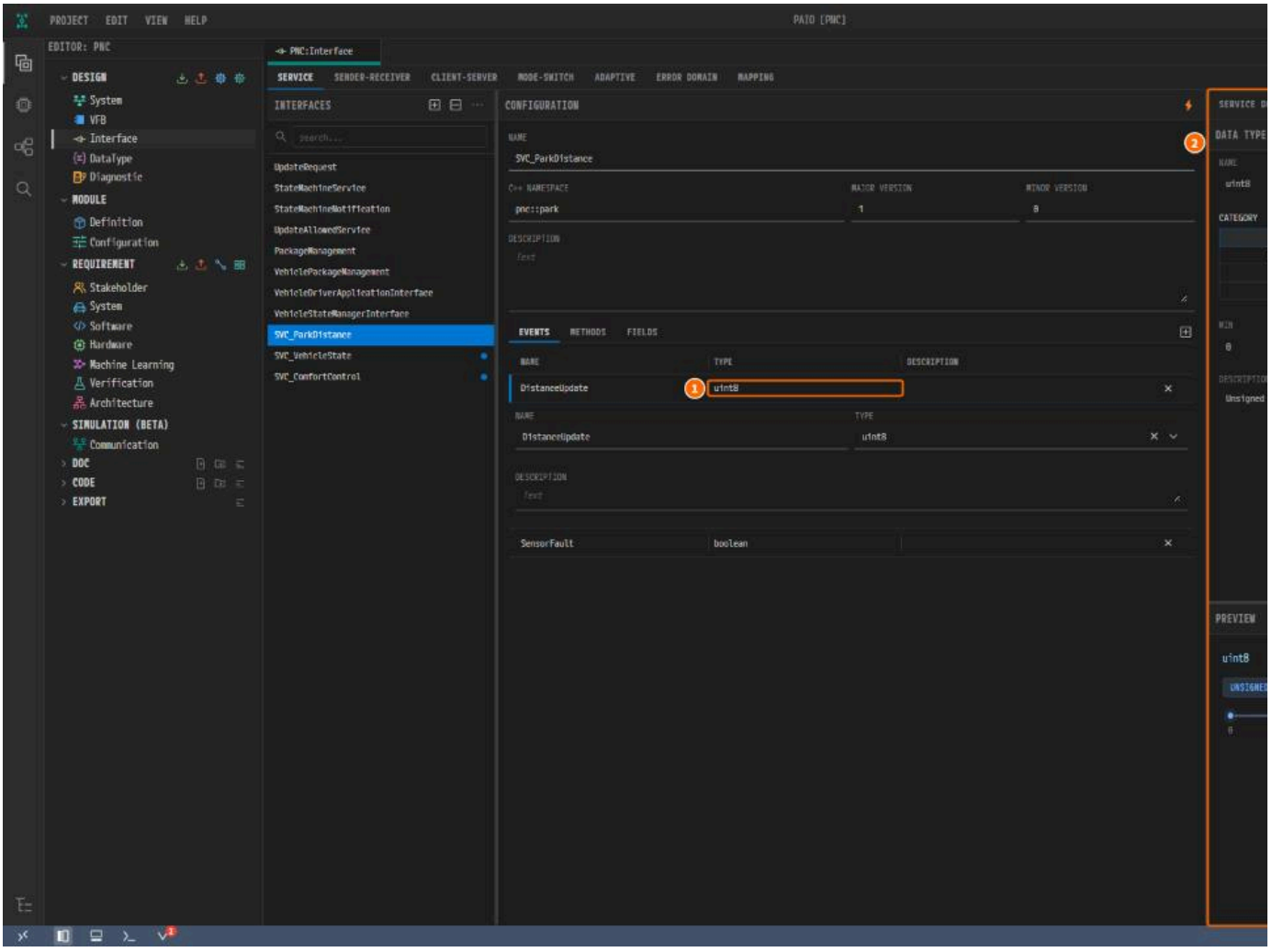
- **PROFILE CONFIGURATIONS**: `+` adds a profile configuration. Expand its row to set its **Name**, **Profile** (`PROFILE_04`, `PROFILE_05` and so on), **Data ID Mode** configurations belong to the project, so every instance card shows the same list.
- **PROTECTED EVENTS** and **PROTECTED METHODS**: `+` lists the events or methods of the service that can be protected; pick one. Expand the new row to the other values.



*① **E2E PROTECTION**. ② **PROTECTED EVENTS**, with the event `DistanceUpdate` protected. ③ **PROTECTED METHODS**. ④ **PROFILE CONFIGURATIONS**.*

9.4.3 DATA TYPE

Click the **TYPE** of an interface element and the right panel switches to **DATA TYPE** and shows that type. Edits go to the same DataType model as 10. DataT

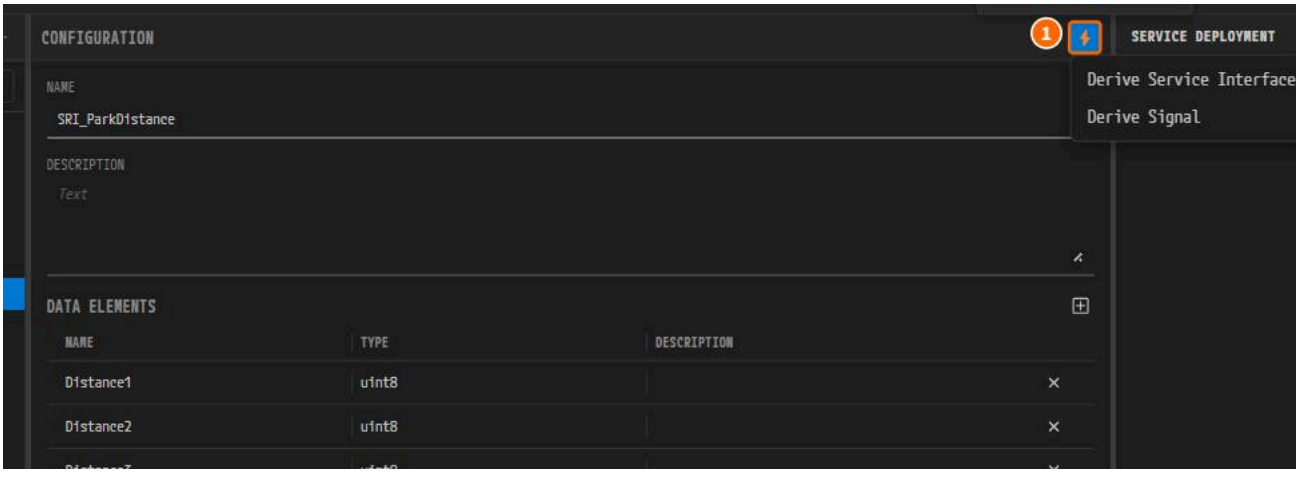


① *Click the type* `uint8` *of an event.* ② *The right panel shows* `uint8`.

9.5 Interface Derivation (Derive)

The ⚡ button in the CONFIGURATION header creates matching interfaces from the selected one. It is hidden when the conditions are not met.

- On the **SERVICE** or **MAPPING** tab it creates SR/CS interfaces named `{name}_SR` and `{name}_CS` at once.
- On the **SENDER-RECEIVER** or **CLIENT-SERVER** tab, **Derive Service Interface** asks for a service interface name. An existing name adds the elements to th
- On the **SENDER-RECEIVER** tab, **Derive Signal** creates signals and PDUs in the Signal editor.



① ⚡ *in the CONFIGURATION header.*

Derivation also fills the InterfaceMapping in the VFB model, and one `Ctrl+Z` reverts all of it.

9.6 Shortcuts

The shortcuts are the common ones (see 18.4). One `Ctrl+S` writes INTERFACE, SERVICE and the embedded DataType together. `Delete` deletes the rows s

9.7 Saving, Undo, and External Changes

- One `Ctrl+S` saves the Interface, Service and DataType models.
- The unit of undo is one edit. An edit that spans two files reverts in one `Ctrl+Z`.
- When a file changes on disk while you have unsaved edits, PAIO keeps your edits and shows a warning.

9.8 Naming Rules

Names start with a letter, then letters, digits, and underscores. A name that breaks the rule shows an orange warning. Some places reject it and others ac
warned name remains before you export.

9.9 Working with the Other Editors

- The VFB editor offers the interfaces defined here when you assign one to a port (see 08. VFB Editor).
- Renaming an interface also updates the SWC ports and ECUC references that use it.
- ARXML import and export are covered in 13. Import and Export.

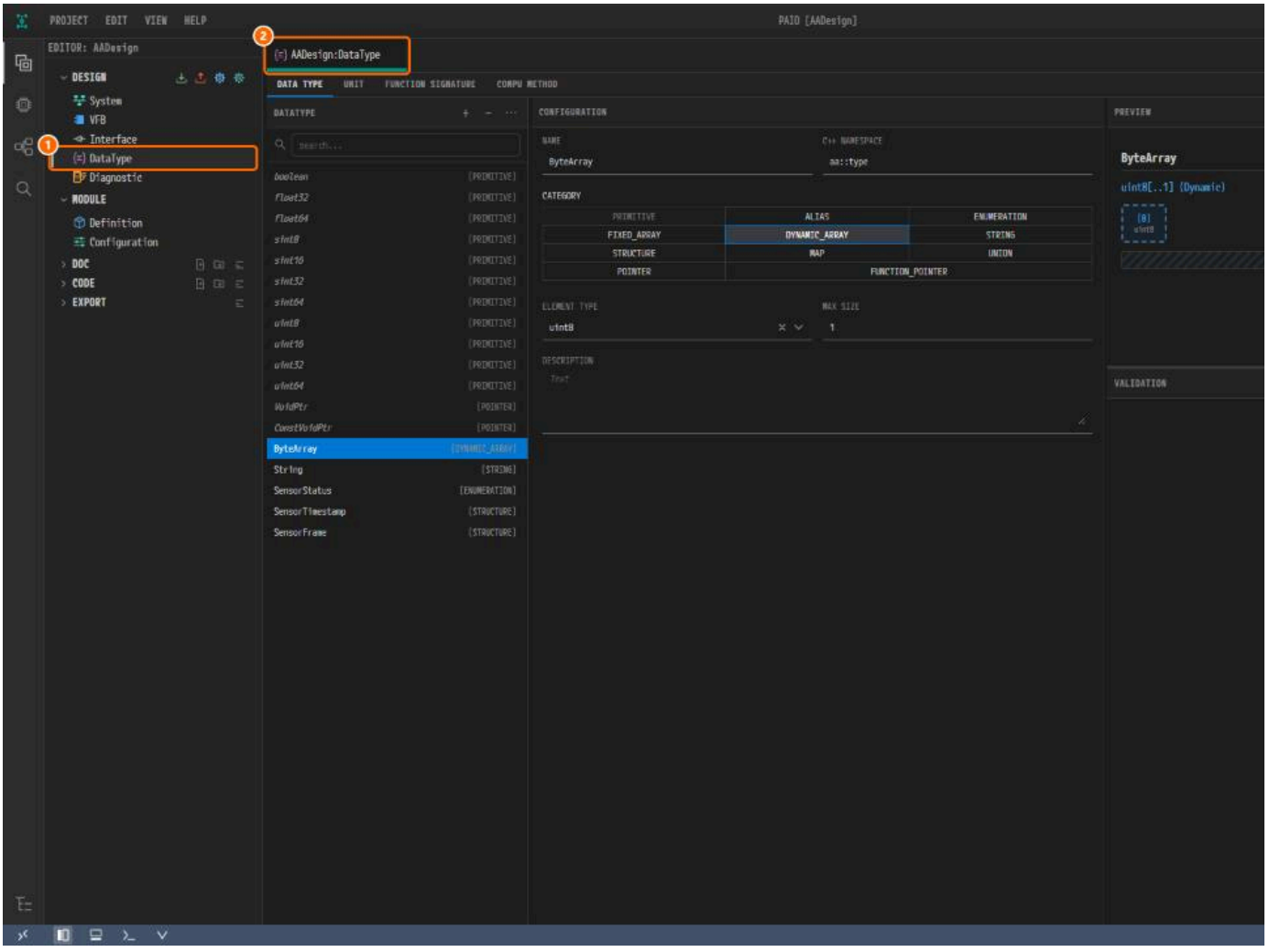
DataType Editor

What This Chapter Covers

This chapter explains how to use the DataType editor. It defines application and implementation data types, physical units, function signatures, and com
import and export are covered in 13. Import and Export.

- Audience: users who define the data types that interfaces or SWCs will use
- Prerequisites: a project is open in the window

To open the editor, click **DataType** under **DESIGN** in the **EDITOR** view on the left. The tab `<project>:DataType` opens. Data types, units, function signatur
DataType model.

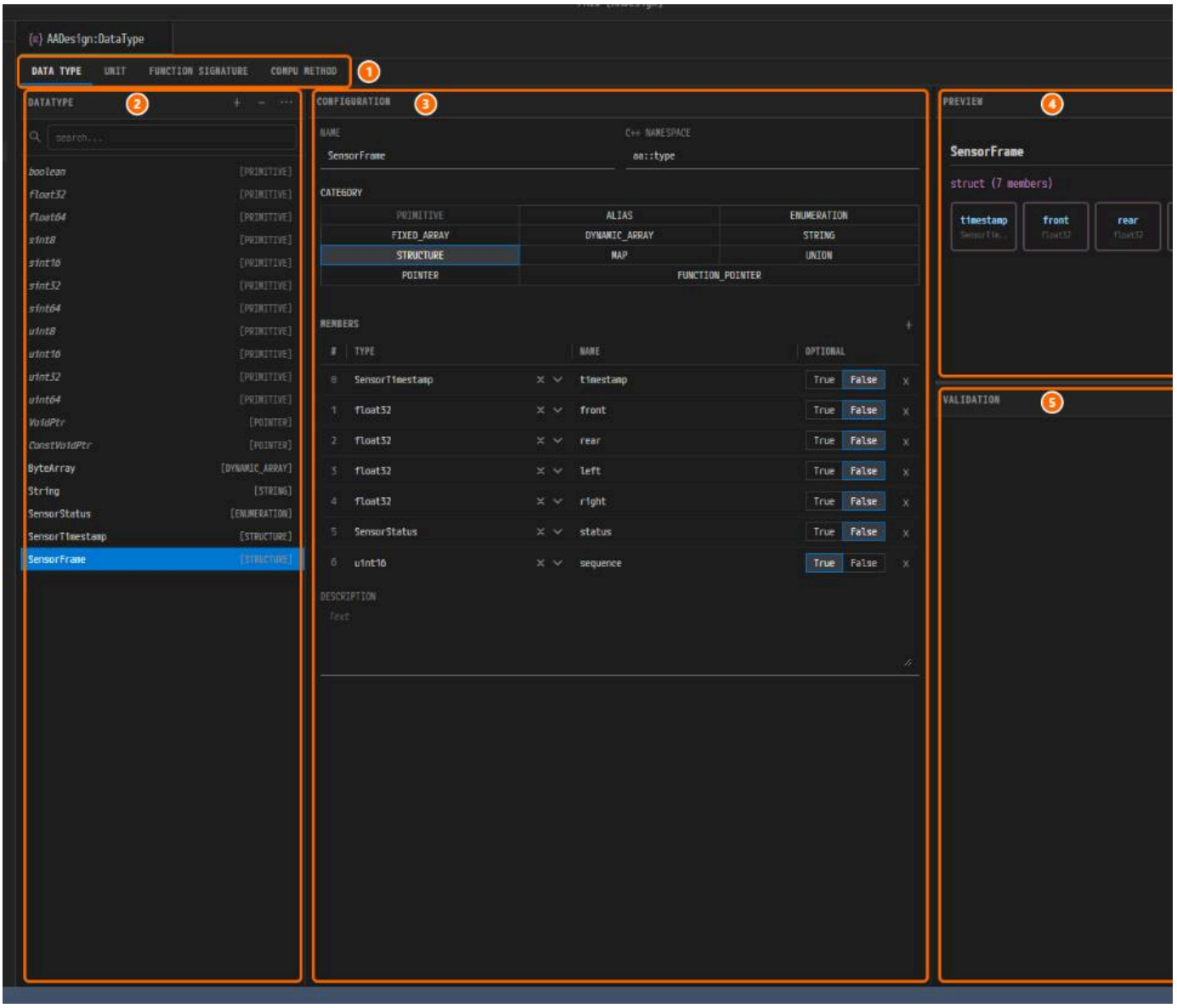


*① **DataType** under **DESIGN**. ② The tab `AADesign:DataType` it opens.*

10.1 Screen Layout

Four tab buttons sit at the top of the editor: **DATA TYPE**, **UNIT**, **FUNCTION SIGNATURE** (10.12) and **COMPU METHOD** (10.13). Each shows a list and its detail forr

DataType Tab Layout

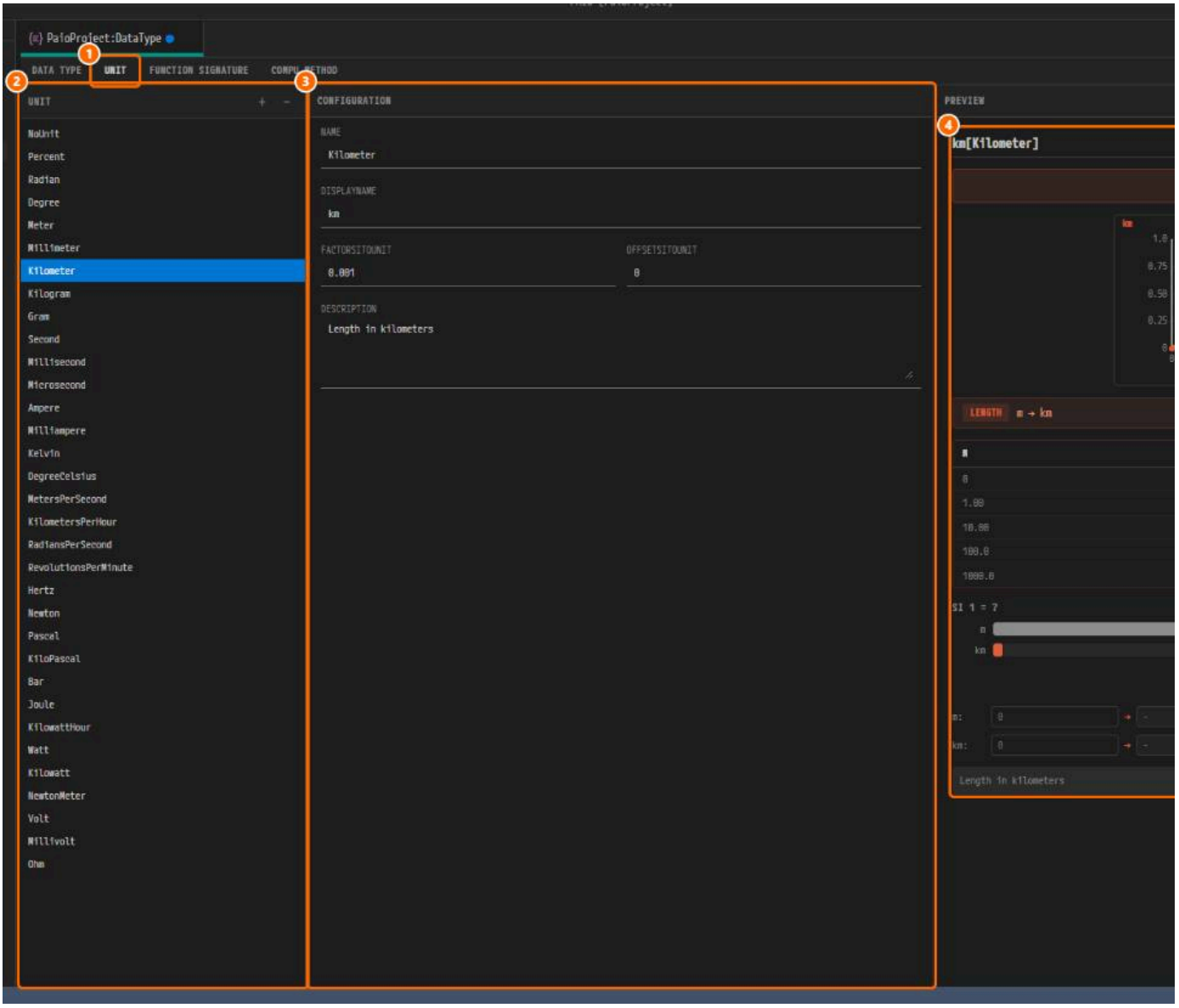


① The four tab buttons. ② The DATATYPE list. ③ CONFIGURATION with the CATEGORY segment and the MEMBERS table. ④ PREVIEW. ⑤ VALIDATION.

Drag the borders between the panels to resize them.

Unit Tab Layout

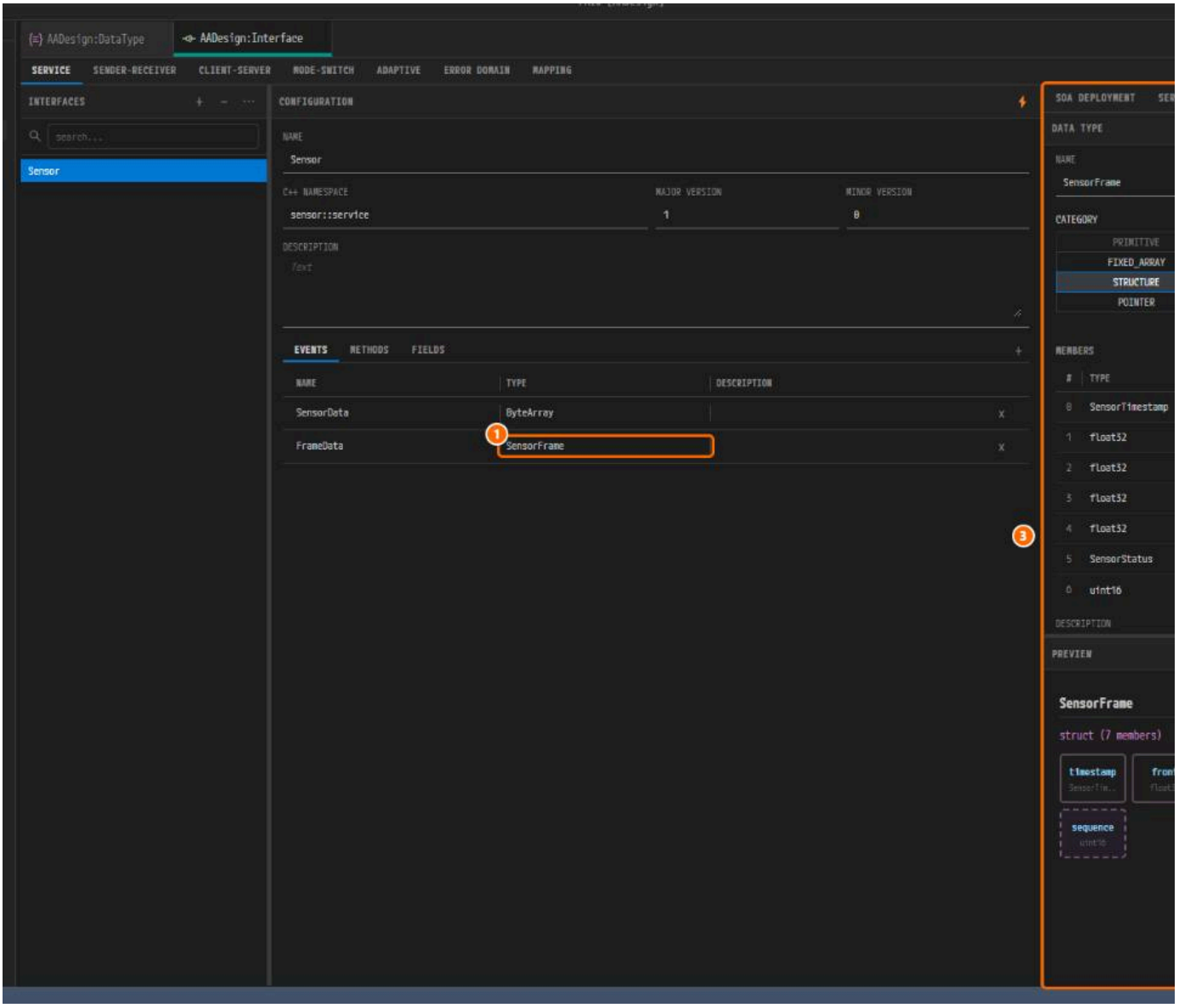
The unit list sits on the left, and the CONFIGURATION form and PREVIEW sit side by side on the right. The Unit tab has no search, filter, or Validation area.



① The UNIT tab button. ② The UNIT list. ③ CONFIGURATION. ④ Preview.

Compact Mode Embedded in the Interface Editor

The 09. Interface Editor shows the same editor in compact mode: click the type of an event and its **DATA TYPE** sub-tab shows only the properties form ar and the Interface editor handles saving and undo.



① The type SensorFrame of the event FrameData. ② The DATA TYPE sub-tab. ③ The compact editor.

10.2 Data Type Categories (11 Kinds)

To define a type:

1. Click **+** in the list. A new type appears with an empty Category.
2. Pick the Category in the segmented control. The form shows that category's properties.
3. Fill in the properties. Reference fields such as BaseType and Element Type are searchable selection boxes.



① PRIMITIVE is dimmed and cannot be selected.

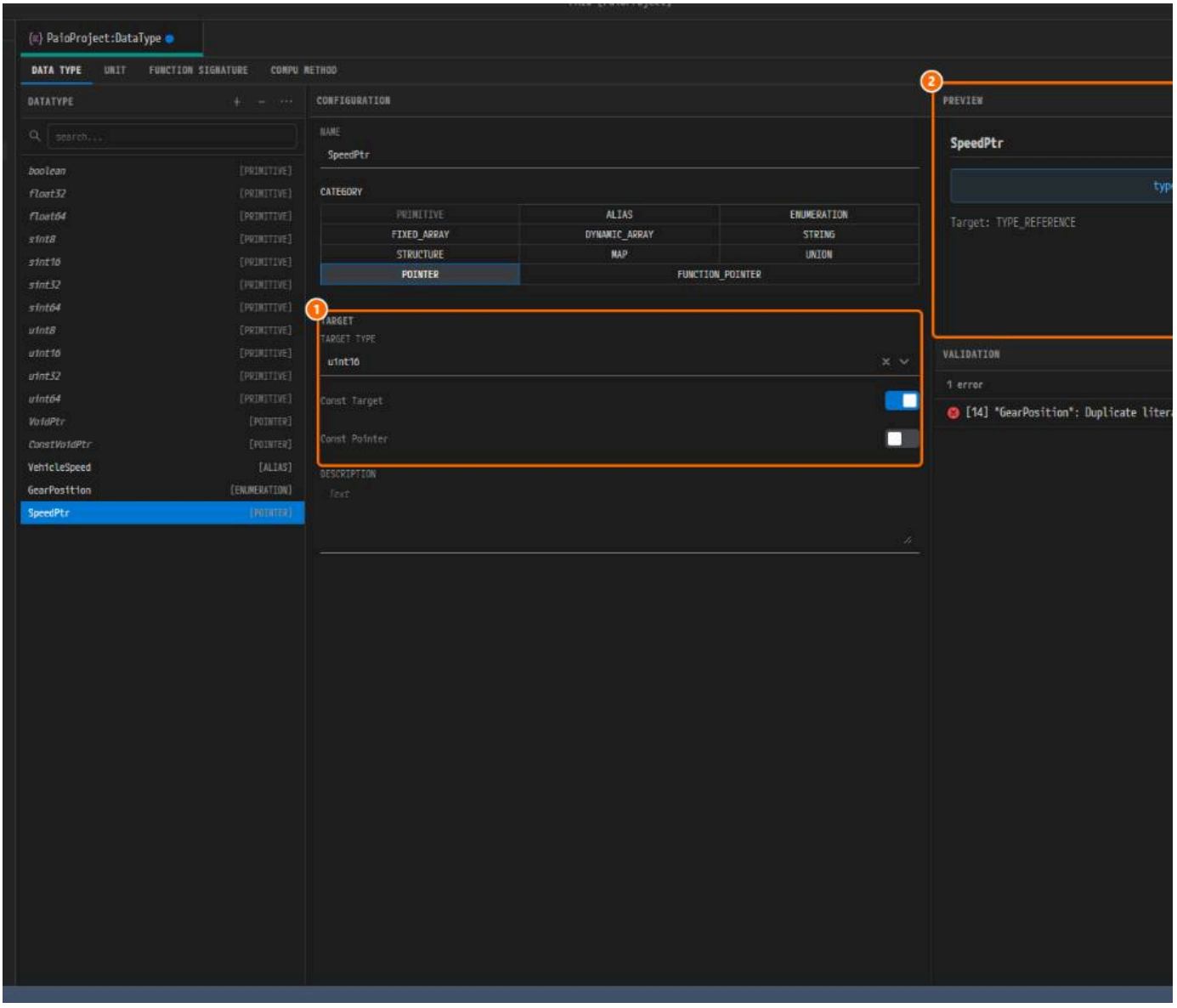
- Every category has `NAME`, `Category` and `Description`.
- The BaseType of ALIAS and ENUMERATION accepts **only the 11 built-in primitives**.
- STRING and MAP are AP only; POINTER is CP only.



① The BaseType list of an ALIAS holds only the 11 built-in primitives.

POINTER and FUNCTION_POINTER

A POINTER sets **Target Type**, the type it points at, and the toggles **Const Target** and **Const Pointer**. A FUNCTION_POINTER names a function signature in **Sig** FUNCTION SIGNATURE tab (10.12).

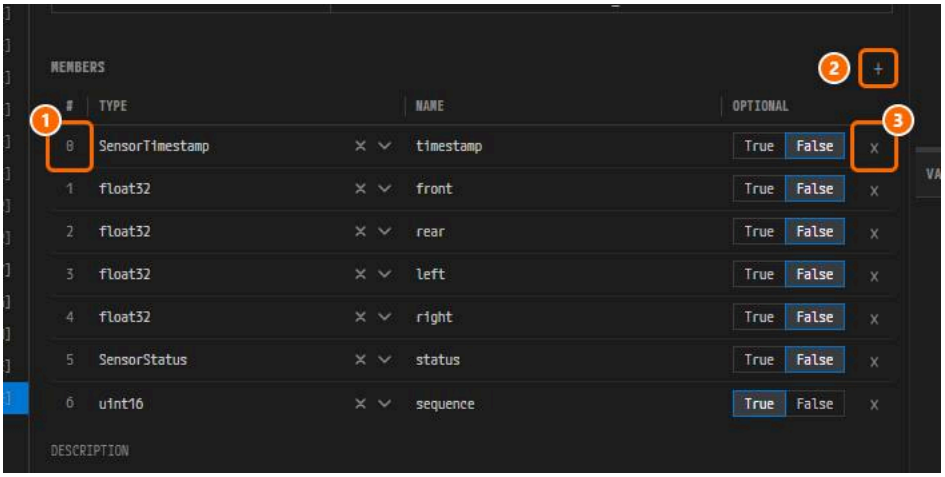


① The Target section with Target Type `uint16` and Const Target on. ② Preview with the typedef line.

10.3 The MEMBERS / ENUMERATION Sub-Table

STRUCTURE, UNION and ENUMERATION add a sub-table below the form: `MEMBERS` for the first two, `ENUMERATION` for the last.

1. Click **+** (`Add Member` or `Add Enumeration`) to add a row.
2. Set the type and name (or the name and value for ENUMERATION).
3. Drag the `#` cell to reorder rows, and click **×** to delete one.



*① The `#` drag handle. ② **+** (Add Member). ③ **×** (Remove Member).*

- A member or literal name that breaks the AUTOSAR rules is not saved; the previous name is kept.
- An ENUMERATION value is clamped to the range of a built-in BaseType. With `uint8`, 300 becomes 255.

10.3.1 Pointer Members

A STRUCTURE or UNION member can be a pointer; such members usually come from an imported ARXML file. Its `Type` cell shows the C type as text, and yc **Target** fields under the row.

10.4 Working with the List (DataType Tab)

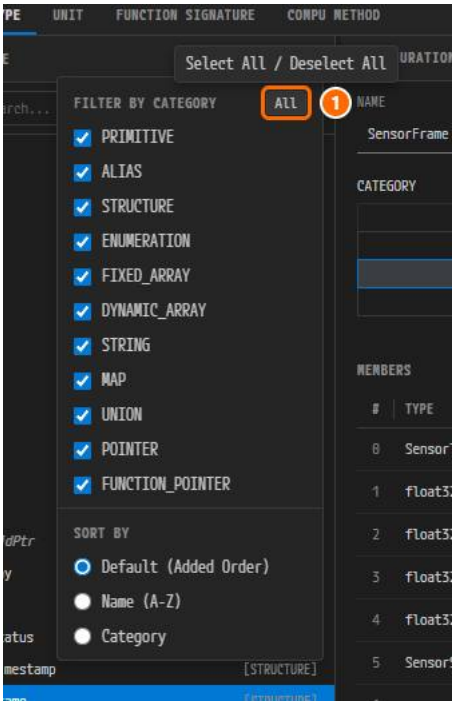
- **+** creates a type and **−** deletes the selected ones after confirmation. `Ctrl`+click selects several, and the `Delete` key deletes like **−**.
- Deleting a type clears the references to it in other types.
- Built-in types cannot be deleted.
- The search box filters names.



① With only ENUMERATION selected, the title reads `DATATYPE [ENUMERATION]`. ② The `[ENUMERATION]` badge.

The … Menu

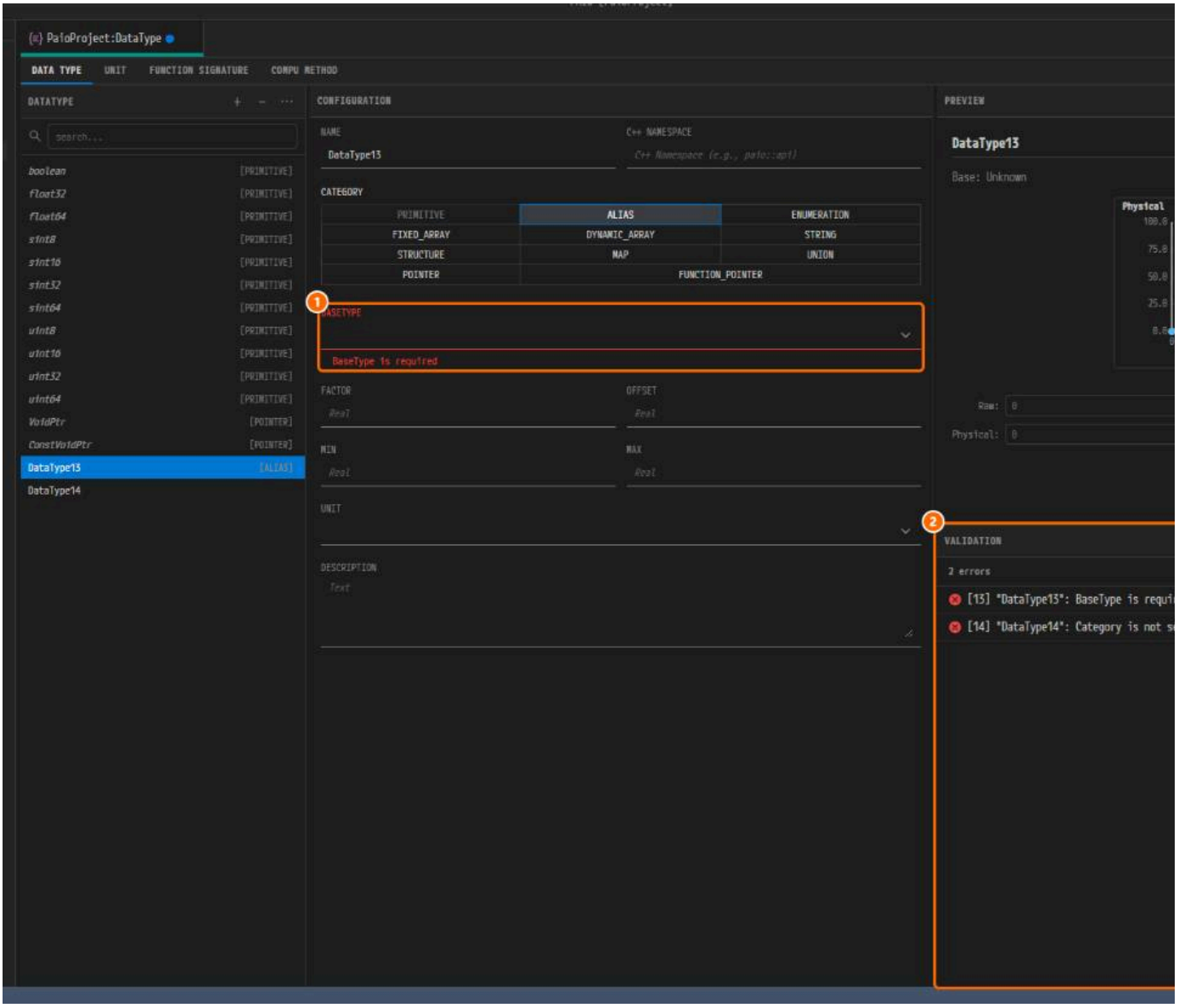
… opens a popover that filters the list by category and sorts it. Sorting changes only the display order, not the file.



*① The **All** button. Below it, the FILTER BY CATEGORY and SORT BY entries.*

10.5 Validation Panel

The `VALIDATION` area at the bottom right checks your edits in real time. Click an entry to select that type in the list.



① The BASETYPE field with `BaseType is required`*. ② VALIDATION with the summary* `2 errors`*.*

A message has the form `[index] "type name": text`, for example `[13] "DataType13": BaseType is required`.

Errors (error)

Missing names, missing required references, self-references and circular references are reported as errors.

Structure and Enumeration Detail Errors

Empty or duplicate member and literal names, and literal values outside the BaseType range, are reported too.

Warnings (warning)

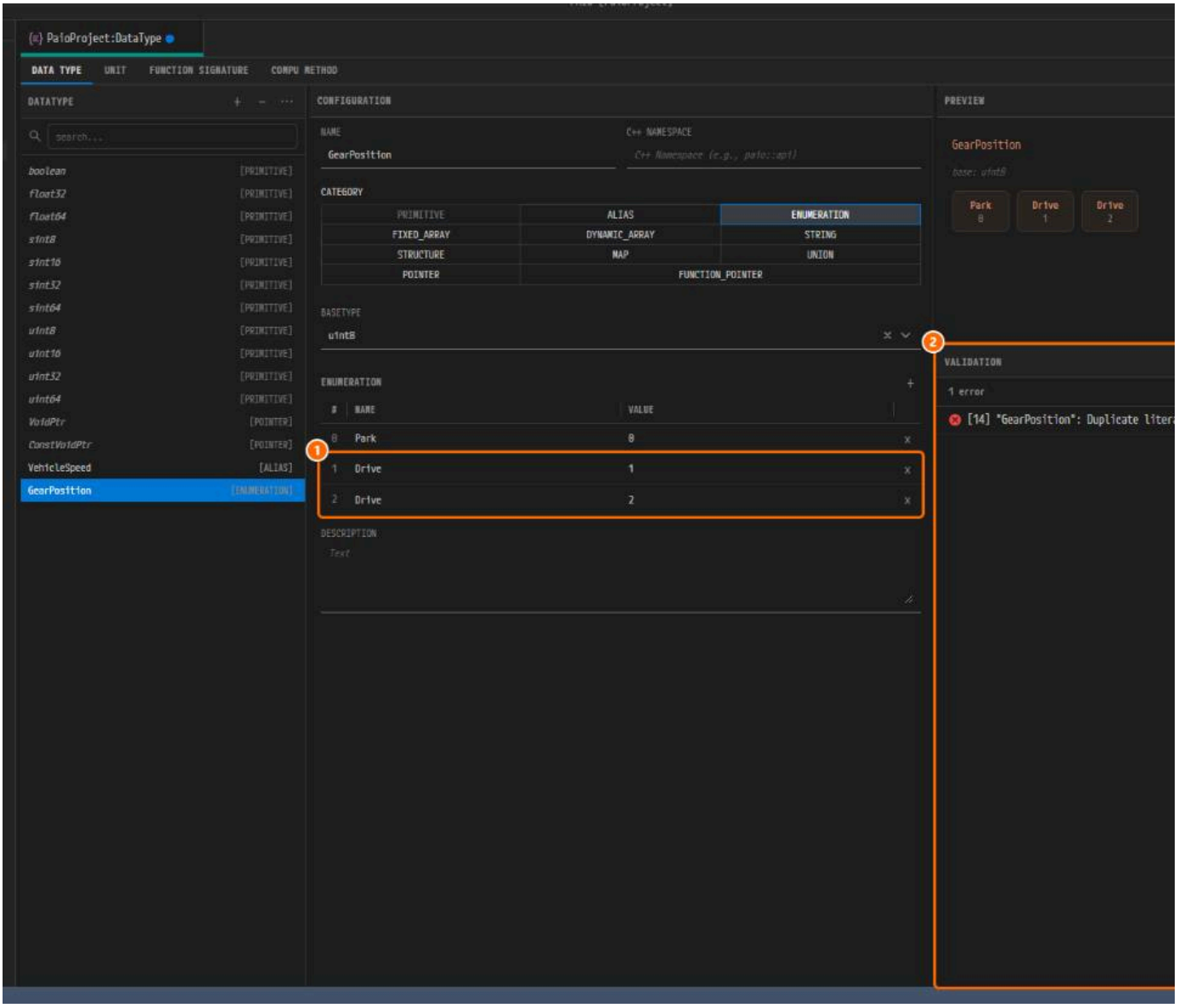
A STRING without Max Size gives a warning, and the default 255 is applied.

Field Errors That Appear as You Type

Some problems, such as Factor 0 or Min > Max on an ALIAS, also appear directly below the field.

Problems in MEMBERS / ENUMERATION Rows

Problems in sub-table rows appear in the Validation panel, for example `[14] "GearPosition": Duplicate literal name: "drive"`.

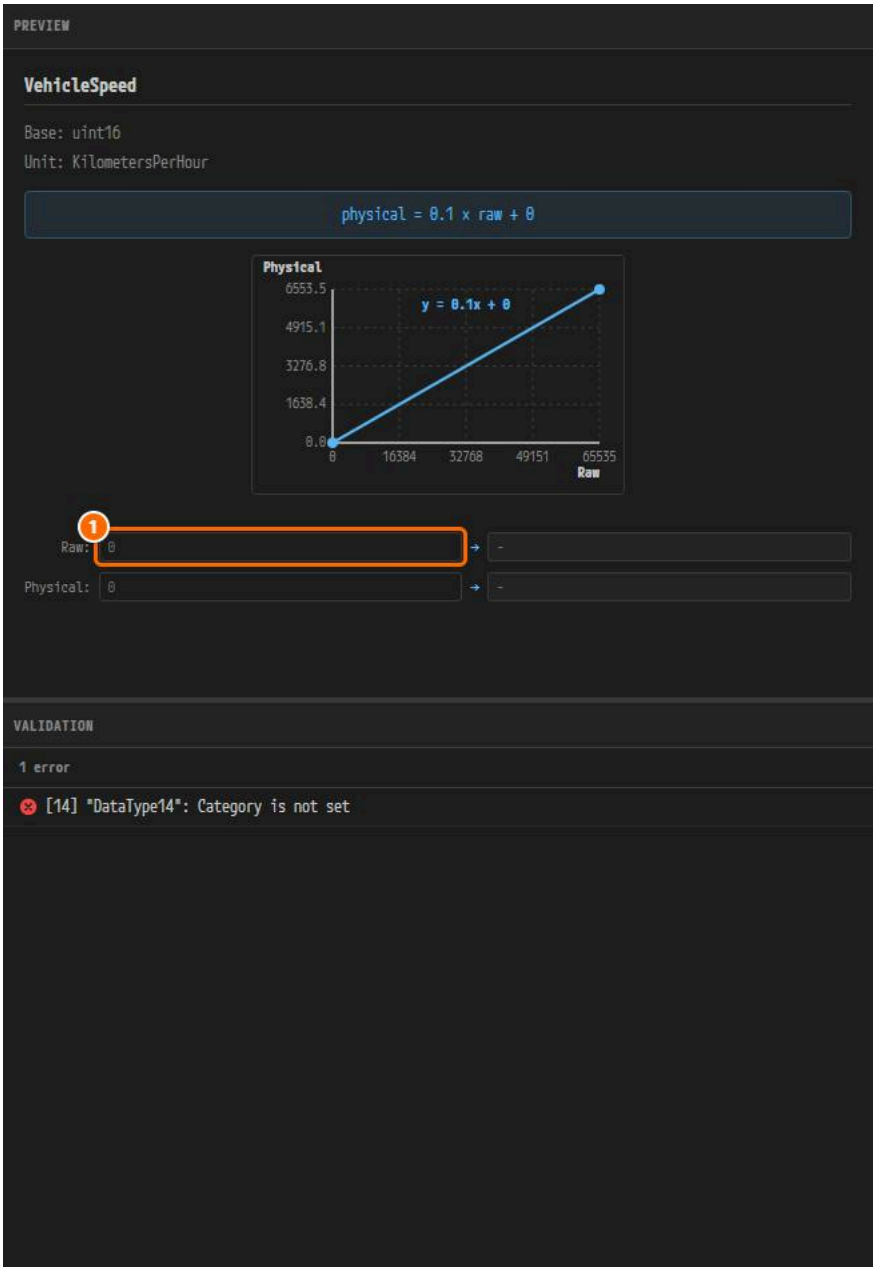


① The two Drive rows. ② The Duplicate literal name entry in VALIDATION.

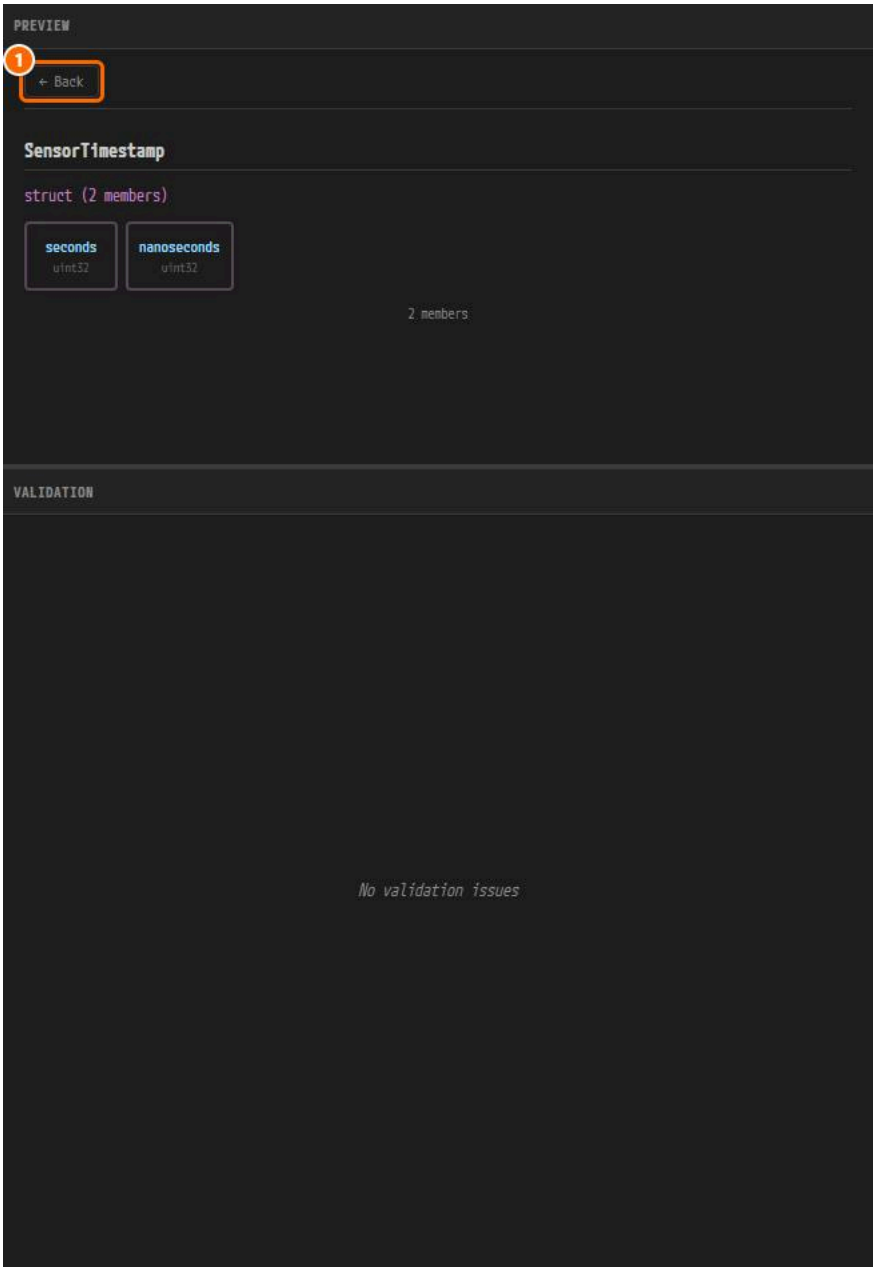
10.6 Preview Panel

Preview draws the structure of the selected type: a range bar, a struct grid, literal chips, a C `typedef`, and so on, depending on the category.

- The ALIAS Preview has a **value converter** that turns Raw into Physical and back. It is a calculator and does not change the model.
- Click a member box whose type has its own structure to drill into it. ← **Back** returns.



① The Raw field of the value converter.



① After clicking the timestamp box of SensorFrame, Preview shows SensorTimestamp and ***← Back****.*

10.7 Built-In Types (11 Primitives and 2 Platform Pointers)

The list always starts with 13 built-in types in italics: 11 primitives (`boolean`, `float32`, `float64`, `sint8` to `sint64`, `uint8` to `uint64`) and the pointer ty

- You cannot rename, edit or delete them, and their names are reserved.
- They are not saved to the model file. This is normal, not a loss.
- Inside the model, use the AUTOSAR notation (`uint8`). C++ notation such as `uint8_t` appears only in C++ ARXML export.

The Structure Created on Export

On ARXML export, one primitive becomes four AUTOSAR representations, and two DATA-TYPE-MAPPING-SETs are added.

10.8 Unit Tab

Units are referenced by the `Unit` property of an ALIAS type.

List

+ creates a unit and **−** deletes the selected one after confirmation.

Editable Properties

1. Click **+** or pick a unit.
2. Enter `NAME`, `DisplayName` (the symbol on screen, such as `km/h`), `FactorSiToUnit` and `OffsetSiToUnit`.

Renaming a unit updates the data types that use it; deleting one clears their `Unit` value.

Preview

Preview shows the conversion formula, a graph, a conversion table, comparison bars and a value converter. A domain badge such as `LENGTH` appears only conversion.

The 33 Default Units

A project starts with 33 common units, from `NoUnit` to `Ohm`. A unit left at its default values is not saved to the file, so a short unit list in the file is normal

10.9 Saving and Undo

- `Ctrl+S` saves all four tabs to the same DataType model.
- `Ctrl+Z` undoes and `Ctrl+Y` redoes. Undo uses one timeline shared by the whole project.

10.10 Integration with Other Screens

- The Interface editor embeds this editor in compact mode.
- The initial value (ValueSpec) editor builds its input form from the types defined here.
- Renaming a type updates the type references in the Interface model, the ECUC configuration and the function signatures. SWC definitions are not u
- Project-wide checks are in 14. Validation.

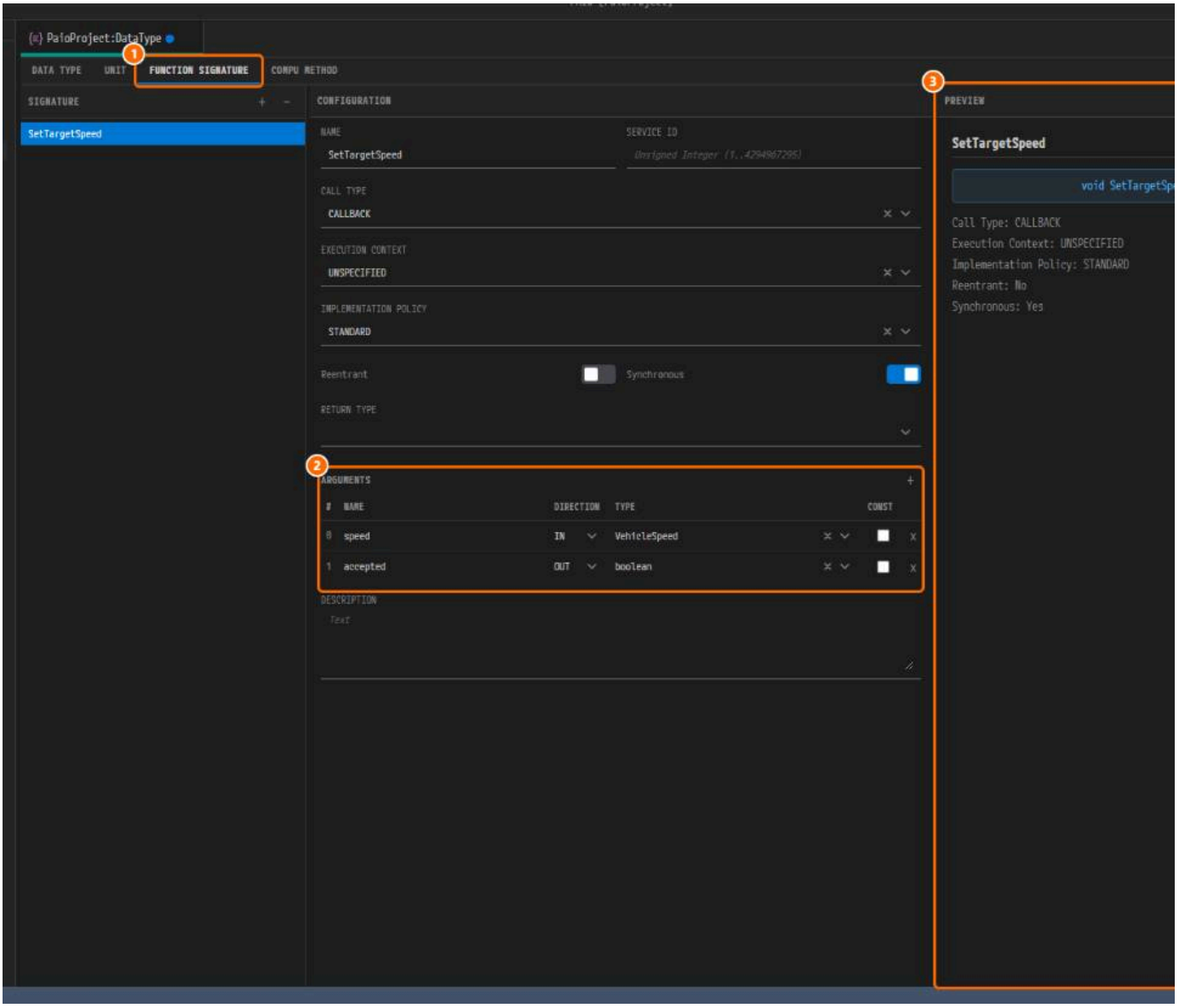
10.11 Editing Rules Summary

- Names start with a letter, then letters, digits, and underscores.
- A type name that is empty or reserved is not saved. Other rule breaks only show a warning, so fix them before you export.
- Every delete asks for confirmation first.

10.12 Function Signature Tab

Defines the function prototypes that FUNCTION_POINTER types name.

1. Click **+** in the `SIGNATURE` list. The entry `FunctionSignature0` appears.
2. Set `RETURN TYPE` and the other properties.
3. Click **+** at the right of `ARGUMENTS` to add an argument, then set its Direction (`IN`, `OUT`, `INOUT`) and Type.

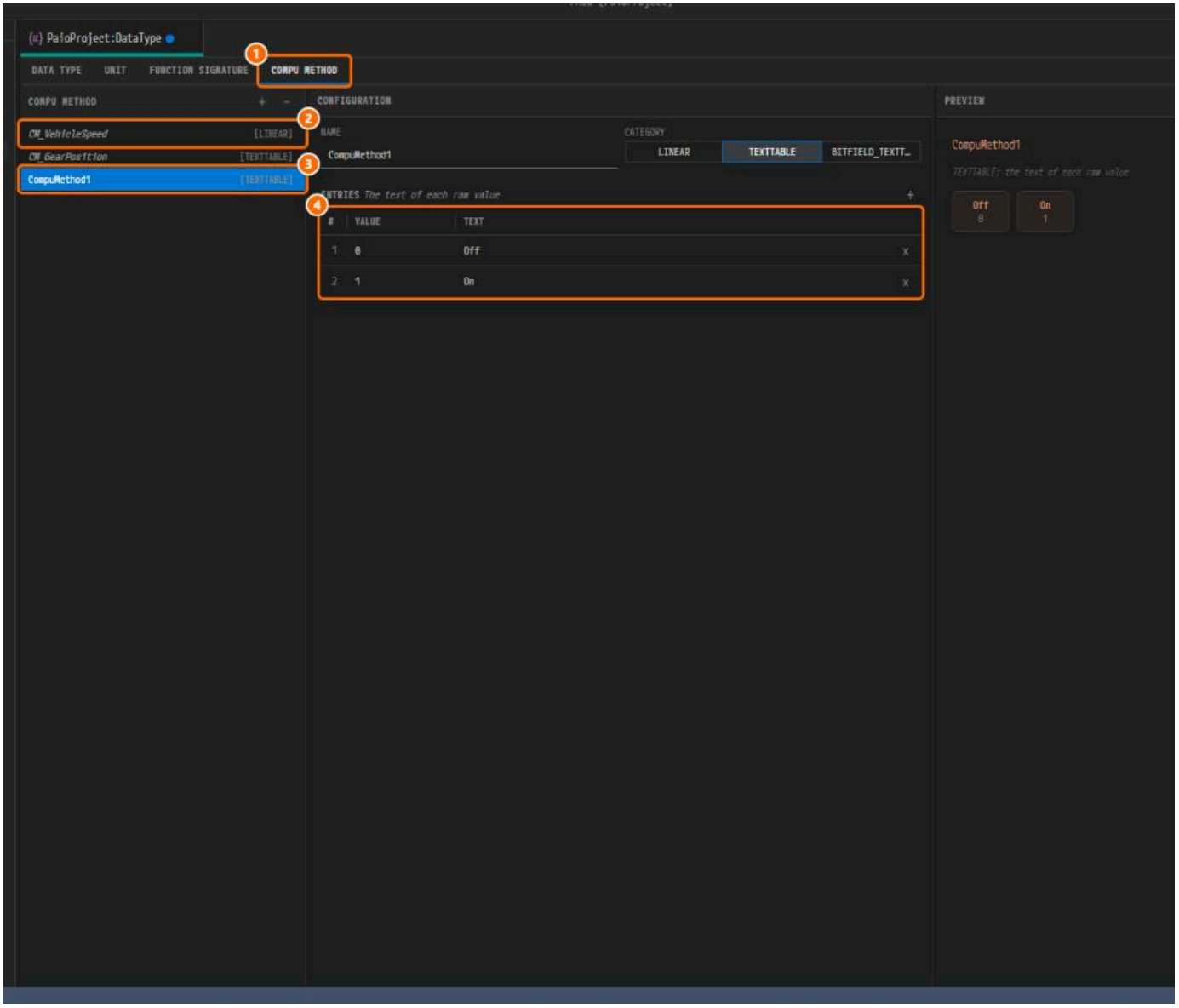


① SetTargetSpeed selected. ② The ARGUMENTS table. ③ Preview with the C prototype.

10.13 Compu Method Tab

Lists the computation methods of the project.

- **Derived entries** come from data types (an ALIAS with a Factor, an ENUMERATION). They are read-only; edit them on the DataType tab with **Open Data**
- **Standalone entries** are the ones you add with **+**. Pick `LINEAR`, `TEXTTABLE` or `BITFIELD_TEXTTABLE` in `CATEGORY`.



① The COMPU METHOD tab. ② The derived entry CM_ VehicleSpeed. ③ The standalone entry CompuMethod1. ④ Its ENTRIES table.

Diagnostic Editor

What This Chapter Covers

This chapter explains the Diagnostic editor, where you edit the diagnostic extract (DEXT) of the project. The editor has two levels:

- **SYSTEM COMMON** holds the catalogues every node shares: sessions, security levels, access permissions, authentication roles, data identifiers (DIDs)
- **DIAGNOSTIC ECUs** lists the ECU and HPC nodes of the System editor. For each node you choose which catalogue entries it supports, which UDS servi

The nodes themselves are created in 06. System Editor. A node's diagnostic addressing (physical and functional addresses, connections, DoIP) is edited i editor.

22.1 Opening the Editor and Save Location

1. In the **EDITOR** view on the left, click **Diagnostic** under **DESIGN**. The tab `<project>:Diagnostic` opens.
2. Edit, then press `Ctrl+S` to save.

The editor reads and writes `design/DIAGNOSTIC.io.model`, which is created on the first save. If the file cannot be read, editing is disabled until you fix c

22.2 Screen Layout

The editor has a tree on the left and the editing area on the right.

- **SYSTEM COMMON** has two rows: **SERVICE** (tabs **ACCESS**, **DATA IDENTIFIER**, **ROUTINE**) and **EVENT**.
- **DIAGNOSTIC ECUs** has one row per ECU and HPC, with an **ECU** or **HPC** badge. The badge is highlighted when the node has a diagnostic specification.

A specification whose node was deleted or renamed in the System editor stays in the list as `<name> (missing)`.

22.3 Enabling Diagnostics for a Node

A node has no diagnostic specification until you enable it. Until then its tabs are read-only.

1. Select the node under **DIAGNOSTIC ECUs**. The banner `Diagnostics are not enabled for <node>.` appears.
2. Click **ENABLE DIAGNOSTICS**. The banner disappears and the node's badge is highlighted.

A newly enabled node supports every session and security level of the catalogue and no service, DID, routine or event.

22.4 Working with Tables

Most lists of the editor are row tables that work the same way.

- **+** in the section header adds a row or opens a menu of choices. Click a row to expand its fields.
- **×** at the end of a row deletes it. An entry that other entries still reference opens **Delete with References**, which deletes the entry and removes those
- A name is checked when the field loses focus. An empty name or one that breaks the AUTOSAR naming rule is refused. A rename is carried into every

22.5 SYSTEM COMMON ▸ SERVICE

The three tabs of **SERVICE** are the only place where the shared service catalogues are created and edited. The node tabs of the same names (22.7) only c

22.5.1 ACCESS

ACCESS has four sections: **SESSIONS**, **SECURITY LEVELS**, **ACCESS PERMISSIONS** and **AUTHENTICATION (0x29)**.

1. Click **+** on **SESSIONS** and pick an ISO 14229-1 session, or **Custom session**.
2. Add security levels with **+** on **SECURITY LEVELS**.
3. Add an access permission and choose the sessions and security levels it allows, and whether **AUTHENTICATION** is required.

22.5.2 DATA IDENTIFIER

The table lists the DIDs of the project.

1. Click **+** and choose **Custom DID** or a DID of ISO 14229-1 Table C.1.

2. Expand the DID and set **NAME**, **ID** and the other fields.
3. In the **PARAMETERS** table, click **+** to add parameters. **Insert DID** appends the parameters of another DID. Drag the **#** cell to reorder rows.

COMPU METHOD offers the CompuMethods of the DataType editor.

22.5.3 ROUTINE

1. Click **+**. A routine named `Routine1` with the START sub-function is added.
2. Expand it and switch **START**, **STOP** and **RESULT** under **SUB-FUNCTIONS** on or off.
3. Fill in the request and response parameter tables of each sub-function that is on. They work like the DID **PARAMETERS** table.

22.6 SYSTEM COMMON ▸ EVENT

EVENT edits the diagnostic events, the DTCs, and the definitions they use. A sub-tab bar at the top holds the **EVENT** sub-tab and the definition sub-tabs.

22.6.1 The EVENT Sub-Tab

The sub-tab has three panels: **EVENTS** on the left, the selected **EVENT** in the center, and its **DTC** on the right.

1. Click **+** in **EVENTS**. `Event1` is added.
2. Set the event fields in the center. For **OPERATION CYCLE** and **DEBOUNCE**, pick a definition from the chevron list, or click **+** to create and bind a new on
3. In the right panel, click **+** in the **DTC** header. A DTC with the next free 3-byte value from `0x500000` is created and bound to the event.

Deleting an event also deletes its DTC.

22.6.2 The Definition Sub-Tabs

The other sub-tabs (**OPERATION CYCLE**, **DEBOUNCE**, **AGING**, **FREEZE FRAME**, **EXTENDED DATA**, **MEMORY**, **DTC GROUP**) list the definitions the events and DTCs u

22.7 Node Tabs

Select a node under **DIAGNOSTIC ECUs** to edit what that node supports. The node tabs never create catalogue entries; they choose from SYSTEM COMMC

22.7.1 OVERVIEW

Sets the protocol, timing, response, DTC status and security values of the node, such as **DEFAULT ENDIANNESS** and **S3 SERVER TIMEOUT (S)** (unset means 5

22.7.2 ACCESS

Owns the session control (`0x10`), security access (`0x27`) and authentication (`0x29`) services of the node.

1. Check **SUPPORTED** on a session or security level. It is added to the node with its service instance.
2. Expand the row and pick its **ACCESS PERMISSION** (**Unrestricted** or an access permission of the catalogue).
3. Check **FUNC** if the instance is reachable by functional addressing.

FUNC is disabled unless one of the node's diagnostic connections has a functional receive PDU (System editor, **DIAGNOSTIC** tab, 6.7.6).

22.7.3 SERVICES

Lists the ISO 14229-1 services of the node, grouped by SID, with one row per sub-function.

1. Check **SUPPORTED** on each row the node offers.
2. Pick its **PERMISSION** and, if needed, check **FUNC**.

Type in the filter field (`Filter SID / name`) to narrow the list.

22.7.4 DATA IDENTIFIER

Check **USED** to assign a DID to the node with unrestricted read access. Expand the row to set the read, write and IO control access to **Unsupported**, **Unre**

22.7.5 ROUTINE

Check **USED** to give every present sub-function of a routine unrestricted access. Expand the row to pick a permission per sub-function.

22.7.6 EVENT

Check **USED** to apply an event to the node; when the event has a DTC, the DTC and its records follow. The event and DTC are read-only here; edit them un

22.8 Saving, Undo, and External Changes

`Ctrl+S` saves, and `Ctrl+Z` / `Ctrl+Y` undo and redo. When the file changes on disk, the editor reloads it, but keeps your unsaved edits if you have any.

22.9 Validation and Limits

14. Validation also checks the diagnostic model. Clicking a diagnostic entry opens this editor on the node, tab and row concerned. A reference that name

22.10 Integration with Other Editors

- The node list comes from the System editor.
- The ECU Configuration dialog of the VFB editor has a **DIAGNOSTIC** tab (8.12.3) with the switch **Derive Dcm/Dem ECUC from DEXT on Apply**, which write
- CompuMethods of DID and routine parameters come from the DataType editor.

22.11 Common Situations

The most common one: a node's tabs are read-only because the node has no diagnostic specification. Click **ENABLE DIAGNOSTICS** (22.3).

ECUC

What This Chapter Covers

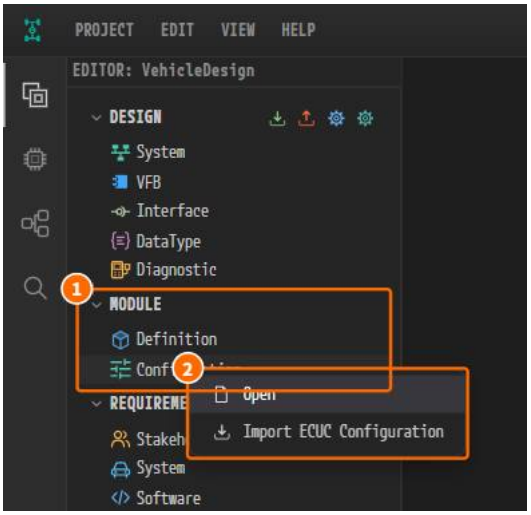
This chapter explains how to use the two editors for the BSW module configuration (ECUC) of an AUTOSAR Classic Platform ECU. The examples use the EC

- Audience: users who configure the BSW modules of an ECU (Os, Rte, EcuM, BswM and others)
- Prerequisites: a project is open and the System editor has an ECU node

Both editors open from **MODULE** in the **EDITOR** view on the left.

Editor	What it holds	Where it opens
Module Definition	Which parameters a module has (the definition, `.def`)	**MODULE ▸ Definition**
Module Configuration	The values of those parameters (the configuration, `.conf`)	**MODULE ▸ Configuration**

A configuration always follows a definition. Most work happens in the Module Configuration editor. **PRTE Platform Import** brings in the definitions of the F yourself.



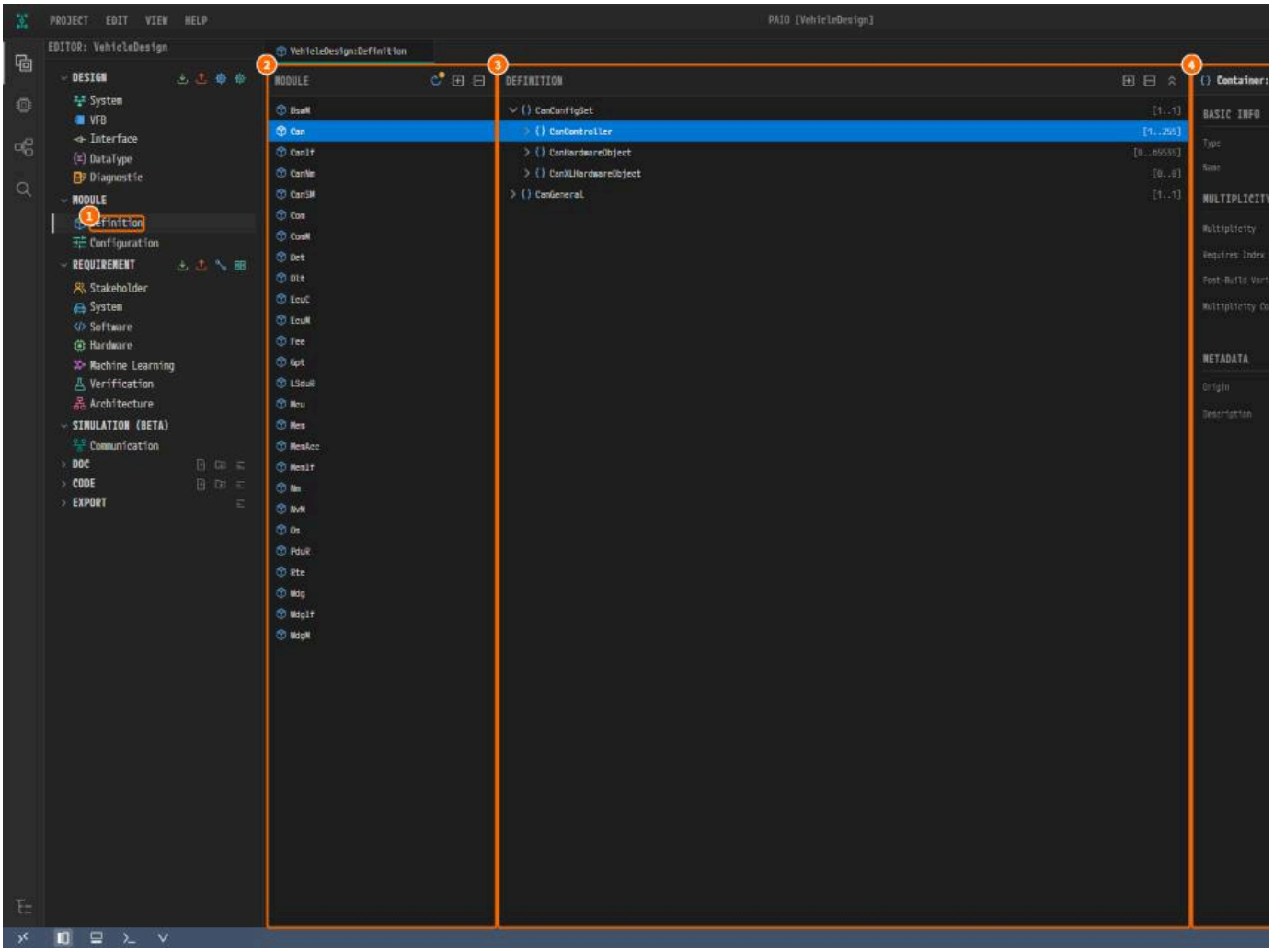
*① **MODULE** in the **EDITOR** view, with **Definition** and **Configuration** below it. ② Right-clicking **Configuration** shows **Open** and **Import ECUC Configuration**.*

12.1 Module Definition Editor

Defines the structure of a module: its containers, parameters and references. You use it only when you build a vendor module yourself.

12.1.1 Opening the Editor and Screen Layout

1. Click **Definition** under **MODULE** in the **EDITOR** view. The tab `<project>:Definition` opens.
2. Pick a module in the ② **MODULE** panel on the left. The ③ **DEFINITION** tree in the middle and the ④ properties panel on the right fill in.



*① **Definition**, ② the **MODULE** panel with `Can` picked, ③ the **DEFINITION** tree, ④ the properties of the selected `CanController`.*

12.1.2 MODULE Panel

- **+** creates a module and **−** deletes the selected one.
- The refresh icon with a dot (**Update to current PRTE VSMD**) appears when a PRTE definition of the project is older than the one PAIO ships. Click it, ch click **Update**. A definition you modified is replaced only if you choose it. The same icon is in the **ECU** list of the Module Configuration editor.
- A new module gets its file only when you save with `Ctrl+S`.

12.1.3 DEFINITION Tree

1. Right-click a container row.
2. Under **Add Container**, **Add Parameter** or **Add Reference**, choose the kind to create.
3. Click the new item and set its name and values in the properties panel on the right.

12.1.4 Properties Panel

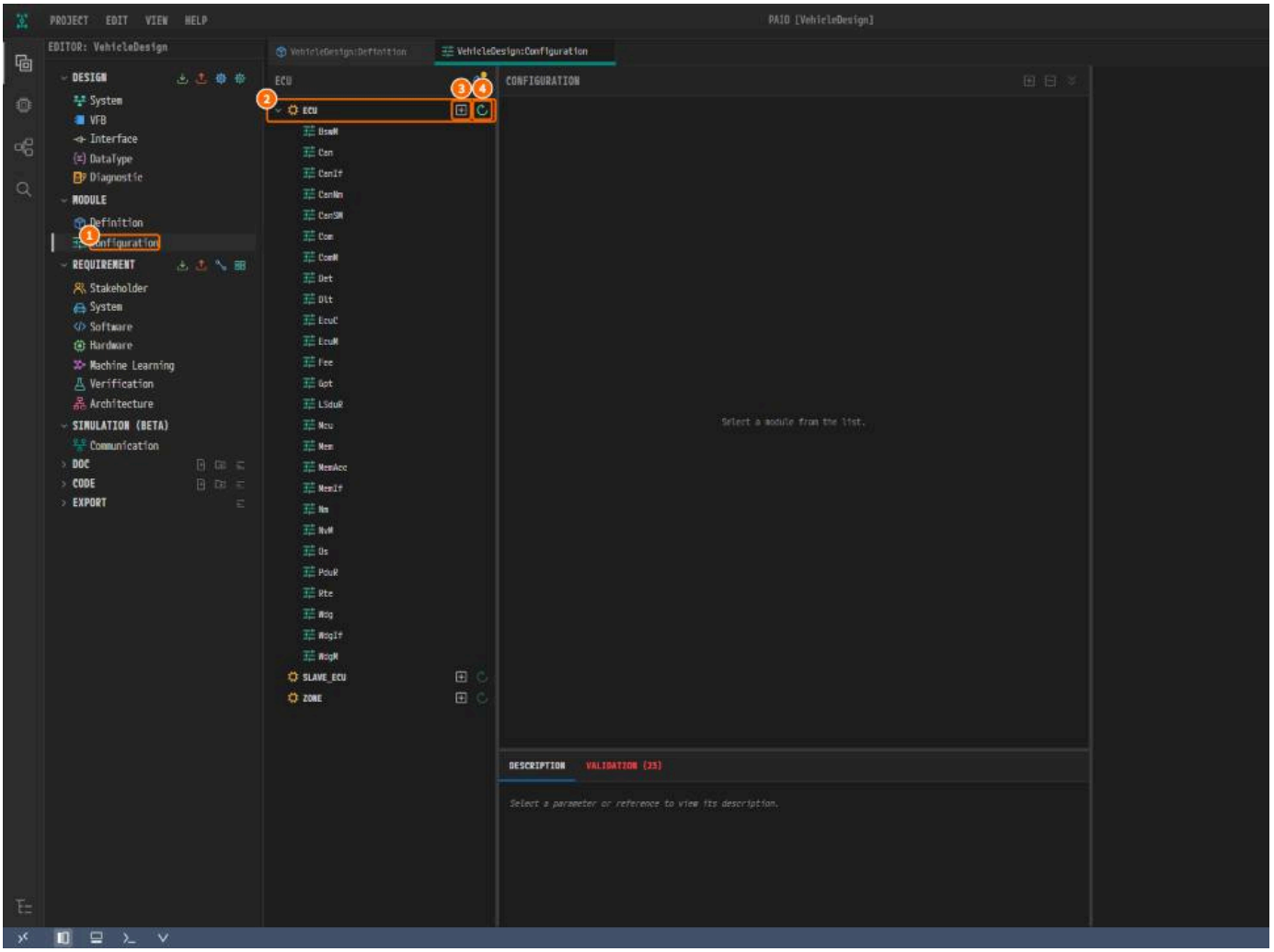
Edits the name, the **Multiplicity** and the options of each type (range, default, literals, reference destination) of the selected item.

12.2 Module Configuration Editor

The editor where you enter the values of an ECU's BSW modules.

12.2.1 Opening the Editor and Screen Layout

1. Click **Configuration** under **MODULE** in the **EDITOR** view. The tab `<project>:Configuration` opens.
2. The **ECU** list on the left shows the ECUs of the design, with the module configurations of each ECU below it.



*① **Configuration**, ② the ECU `ECU` and its modules, ③ **Add Module**, ④ **PRTE Auto Configuration**.*

12.2.2 ECU List

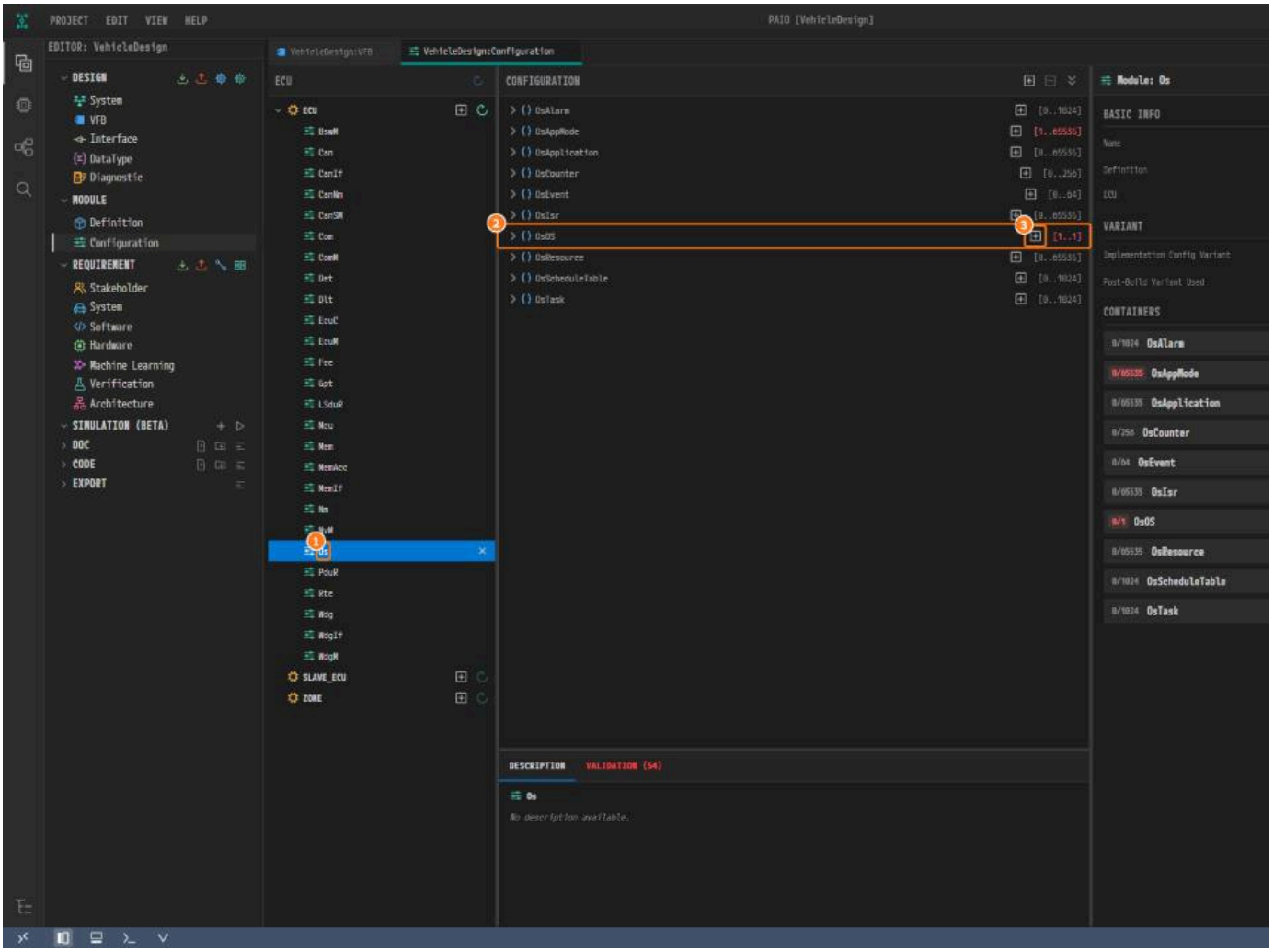
- **+** (**Add Module**) on an ECU row lists the modules. Pick one to create its configuration in that ECU. The name is the module's name; a second configura
- **×** on a configuration row deletes the configuration. Double-click its name to rename it.
- **Add Module** is off while the project holds no module definition, and **PRTE Auto Configuration** is off while the ECU holds no configuration (`Import PRT`
- A new configuration gets its file only when you save with `Ctrl+S`.

12.2.3 CONFIGURATION Tree

The tree follows the definition. The definition's containers appear as group rows, with the actual instances under them. A group that does not yet reach i red.

To add a container, using `OsOS` of `Os` as the example:

1. Click ① **Os** under the ECU in the list.
2. Click ③ **+** on the red `OsOS` group row ②. The instance `OsOS` appears.

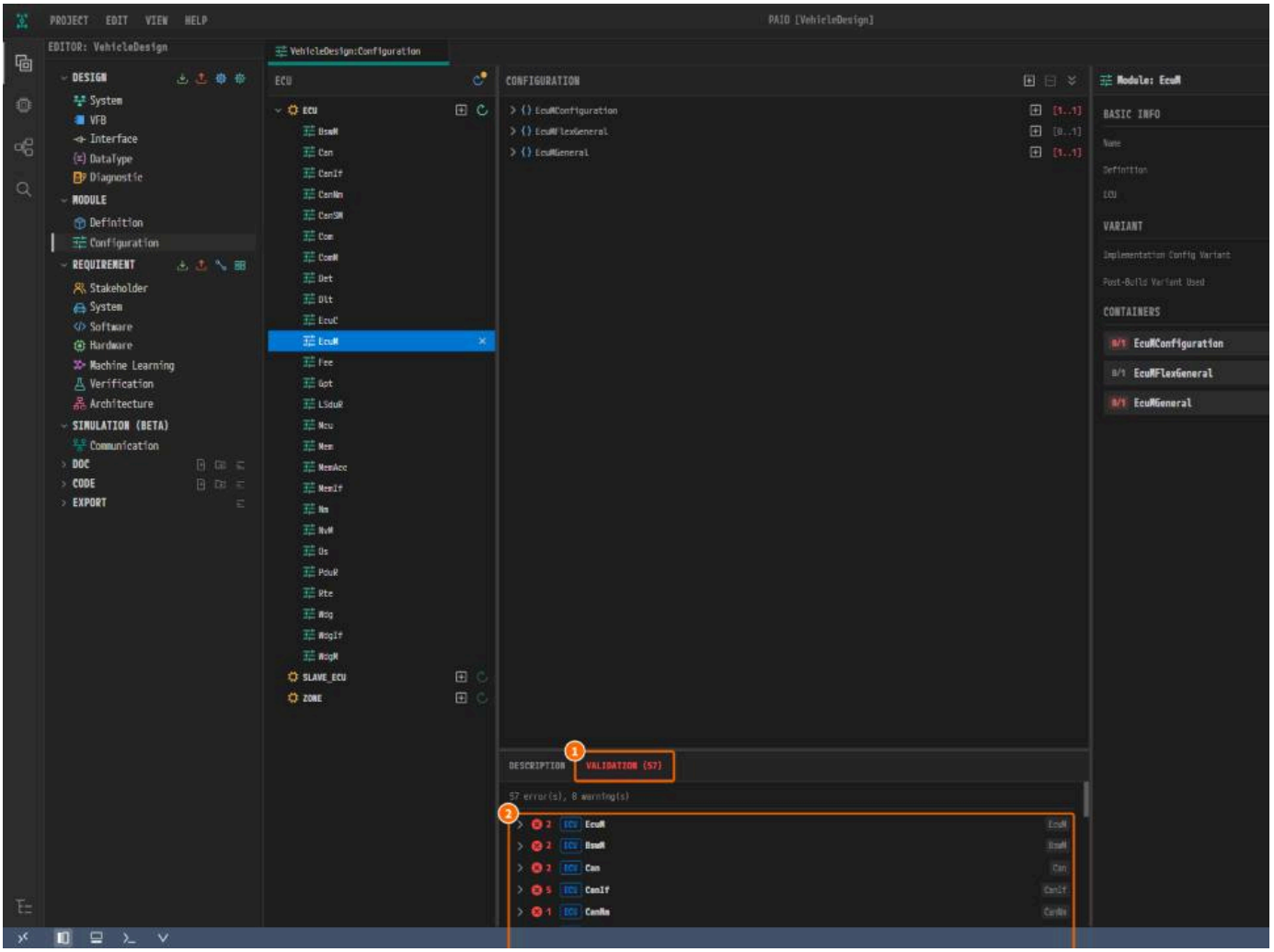


*① **Os**, ② the `OsOS` row, ③ **+**. The lower bound of `OsOS` is 1, so its multiplicity is red.*

12.2.4 Description Tab and Validation Tab

Two tabs sit at the bottom of the middle panel.

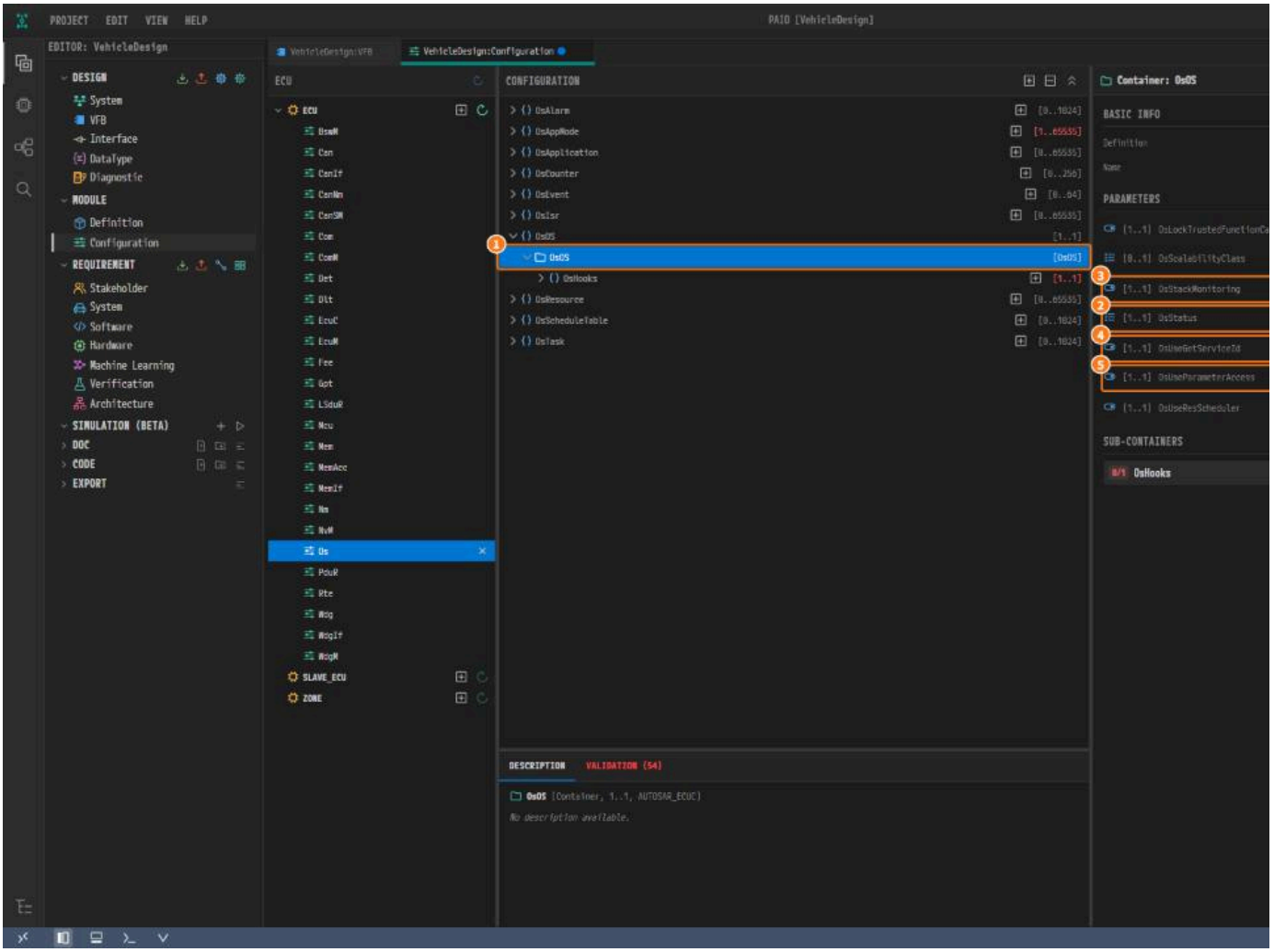
- **DESCRIPTION** shows the definition's description of the parameter you just clicked.
- **VALIDATION** collects problems such as a required parameter with no value or a value out of range. Click an entry to go to it. The number in the tab lab



① The tab label `VALIDATION (57)`. ② Errors and warnings grouped by module.

12.2.5 Value Editing Panel

1. Click an instance in the tree, such as `OsOS`.
2. Enter the values under **PARAMETERS** in the panel on the right. Numbers and text use an input box, enumerations a combo box, and booleans the **TRU**
3. Save with `Ctrl+S`.



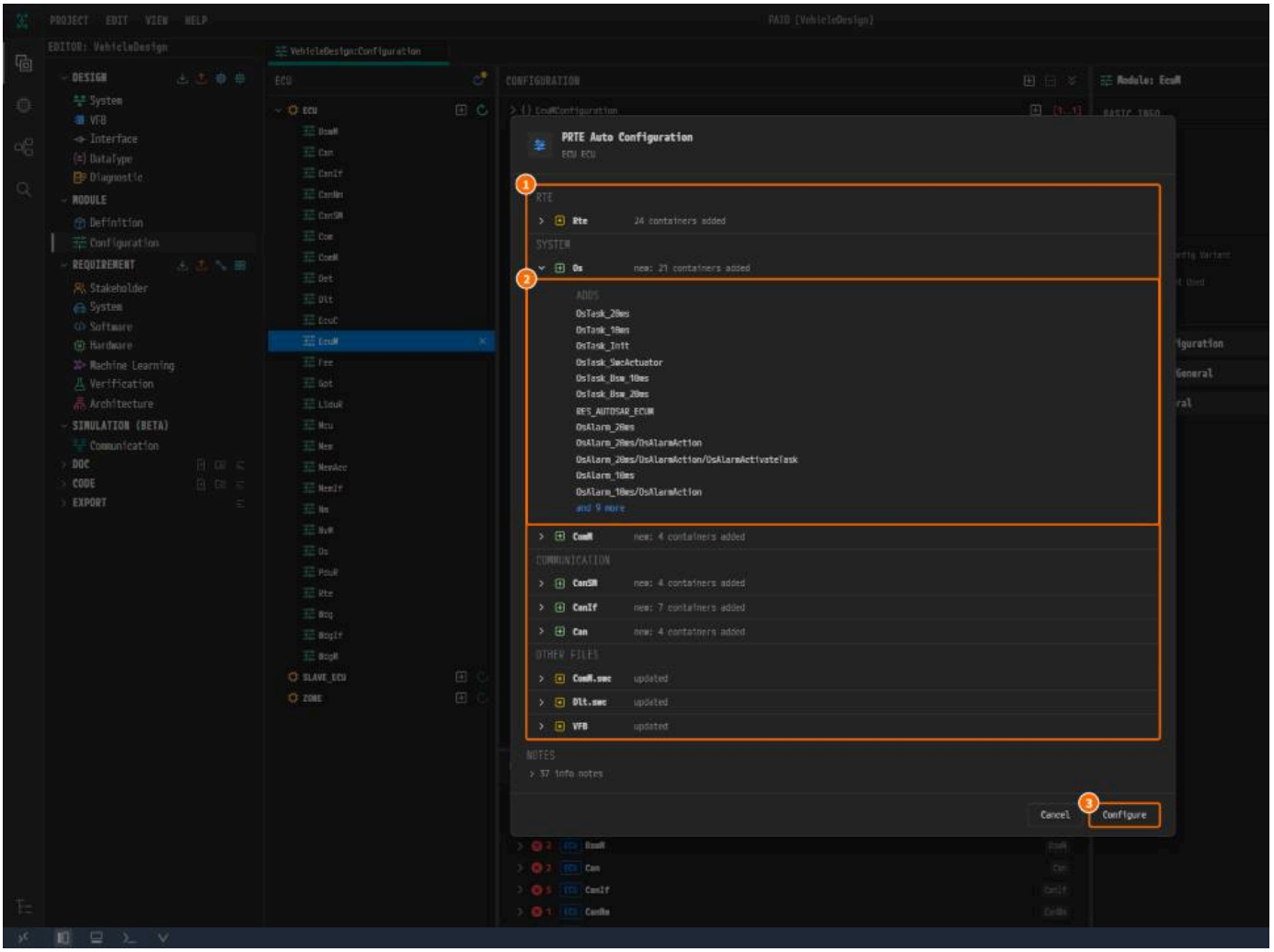
① `OsOS`, ② to ⑤ *the parameters with values entered.* `[1..1]` *at the start of a row means the parameter must have a value.*

- An empty required parameter is underlined in red.
- The revert icon at the right of a row resets the value to the definition's default.
- A grey value in an input box may be only the default, not a value saved in the file.

12.2.6 Configuring an ECU from the Design

The values the design decides (tasks, RTE mappings, the communication stack and so on) are filled in from the design, not by hand.

1. Click the **PRTE Auto Configuration** icon on the ECU row of the **ECU** list.
2. The **PRTE Auto Configuration** window previews the run; nothing is written yet. The files are grouped by area. Click a file to see what it **Adds**, the **Value** and what is **Removed**. **NOTES** lists the remarks of the run, and `Not imported:` names modules you still have to import.
3. Click **Configure**. On a later run the button reads **Apply**.



*① The files the run writes, ② the opened file `Os`, ③ **Configure**.*

> **Caution.** The run also changes values you entered that the design decides. Read the preview before you click **Apply**.

- The whole run is one undo step.
- After a design change, the icon carries a dot, and its tooltip lists the files the run would change. Click the icon to bring the configuration back in line
- The design does not decide every value. The **VALIDATION** tab shows the values that are still missing.

12.3 How References Are Resolved

You pick a reference value from a list that holds only the destination containers the definition names. A value whose target does not exist is highlighted .
When you rename an element in another editor, the reference values in the configurations follow.

12.4 Automatic ECUC Generation

Two features build ECUC from the design. Both also open from the ECU Configuration window of the VFB editor.

Feature	What it does
PRTE Platform Import	Brings in the PRTE definitions and empty configurations of the chosen BSW modules
PRTE AUTO CONFIGURATION (the **PRTE Auto Configuration** icon)	Writes the values the design decides and leaves the rest. It adds no module

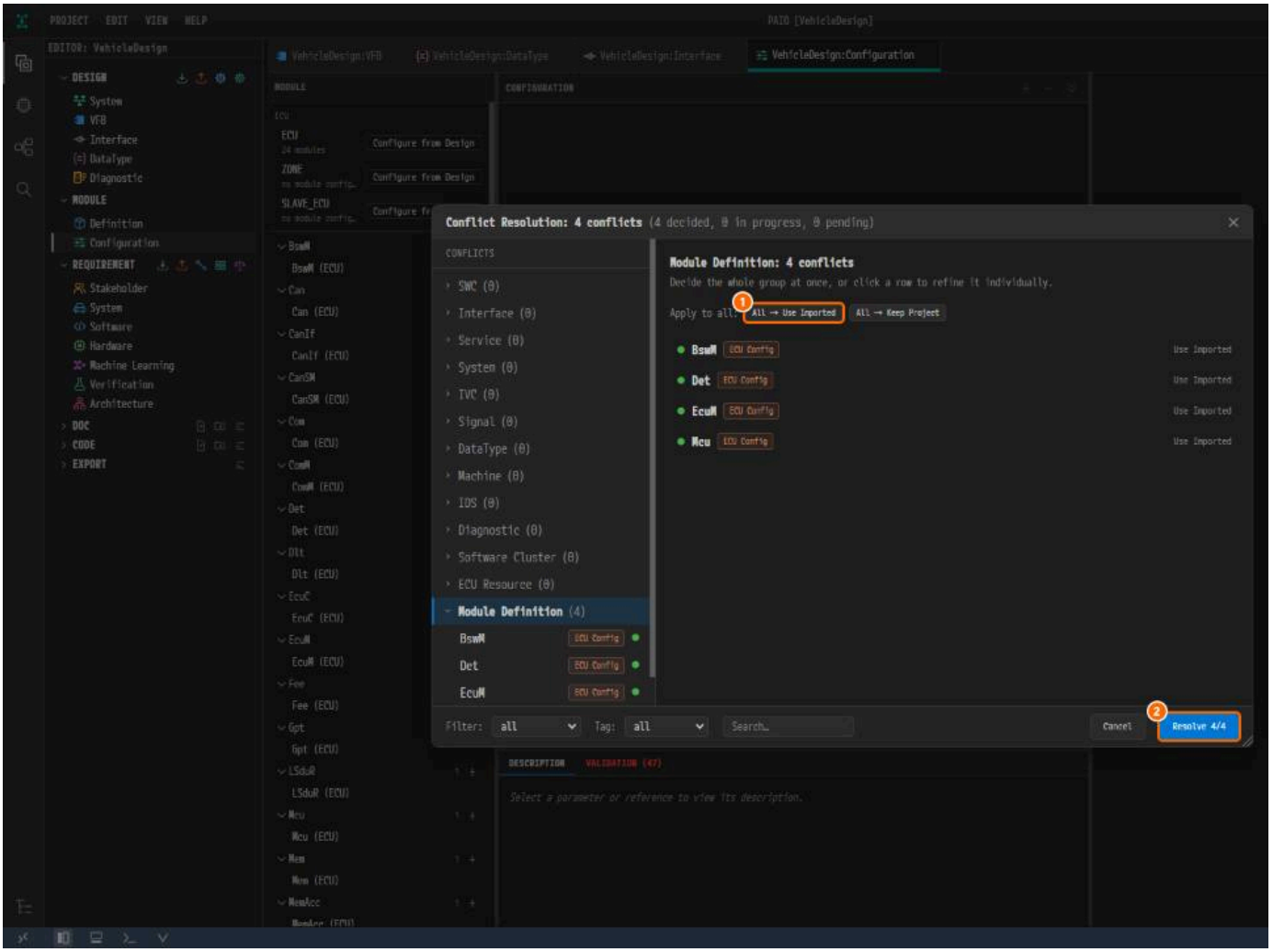
PRTE needs the configurations of the five modules `Os`, `Rte`, `EcuM`, `BswM` and `Mcu` to generate code. For the actual order, follow chapters 6 to 10 of the A
an ECU on a CAN bus, network management included, by hand, without PRTE AUTO CONFIGURATION.

12.5 ECUC Import and Export

12.5.1 Import

Brings in ECUC values made by another tool from an ARXML file.

1. Right-click **Configuration** under **MODULE** and click **Import ECUC Configuration**.
2. Choose an `.arxml` file.
3. If it overlaps existing configurations, the **Conflict Resolution** window opens. Click **Module Definition**, choose **All → Use Imported**, then click **Resolve**.



*① **All → Use Imported**, ② **Resolve 4/4**.*

When it finishes, `ECUC Configuration Import Complete: imported <n> module configuration(s).` appears. A module whose definition is not found

12.5.2 Export

Export in the **DESIGN** header of the **EDITOR** view exports ECUC together with the rest. Only configurations with an **ECU** set are exported, so check the **ECU**

12.6 Editing Rules and Shortcuts

12.6.1 Naming Rules

A name starts with a letter, uses only letters, digits and underscores, and is at most 128 characters. A name that breaks the rule is not saved.

12.6.2 Shortcuts

Action	Shortcut
Save	`Ctrl+S`
Undo / Redo	`Ctrl+Z` / `Ctrl+Y`
Delete the selection	`Delete`
Duplicate a container instance (Configuration editor)	`Ctrl+D`

12.6.3 External Changes

When another tool changes an open file and you have unsaved edits, the editor keeps your edits and shows a warning. Without edits it reloads quietly.

12.6.4 License State and Editing Restrictions

A read-only session blocks editing. See 03. Installation, Login and License.

Workflow

What you do with a model once it exists - moving it in and out, checking it, and building from it.

- Import and Export
- Validation
- AI Assistant
- Code Editing, Build, Terminal and Documents
- Remote Development
- AI Workflow

Import and Export

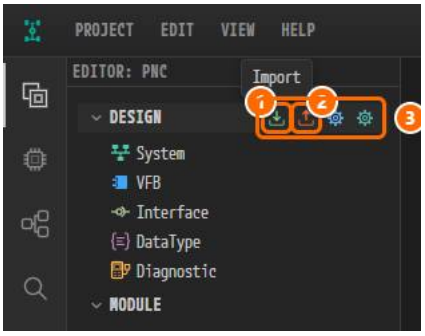
What This Chapter Covers

This chapter explains how to bring external files (ARXML, DBC) into a PAIO project and how to export the project model back out as AUTOSAR ARXML or DI of a single ECU, and the separate ECUC import.

- Who this is for: users moving ARXML or DBC files from another tool into PAIO, or handing a PAIO model over to another tool.
- Prerequisites: a project is open (04. Workspaces and Projects).

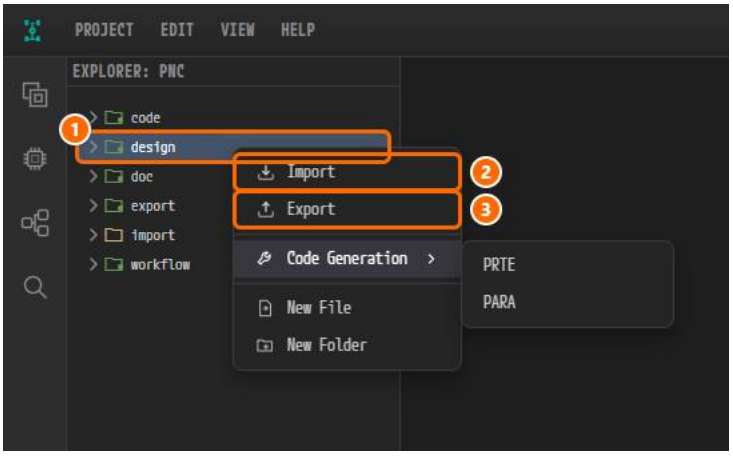
13.1 At a Glance

What you want to do	Where to start
Import an ARXML or DBC file into a project	The **Import** icon in the **DESIGN** header of the **EDITOR** view
Export the whole project as ARXML, DBC or both	The **Export** icon in the **DESIGN** header of the **EDITOR** view
Export one CAN cluster as DBC	Right-click the CAN cluster in the System editor ▸ **Export to DBC**
Export a single ECU or HPC	Right-click the node in the System editor ▸ **Extract ECU** / **Extract Machine**
Import ECUC configuration values only	Right-click **Configuration** under **MODULE** ▸ **Import ECUC Configuration**
Add the PopcornSAR AP base configuration	The **PARA Platform Import** button in the Machine Configuration dialog



*① **Import**, ② **Export**, ③ the four header icons: **Import**, **Export**, **PRTE Code Generation**, **PARA Code Generation**.*

The same two commands are on the context menu of the `design` folder in the Explorer.



*① The `design` folder, ② **Import**, ③ **Export**.*

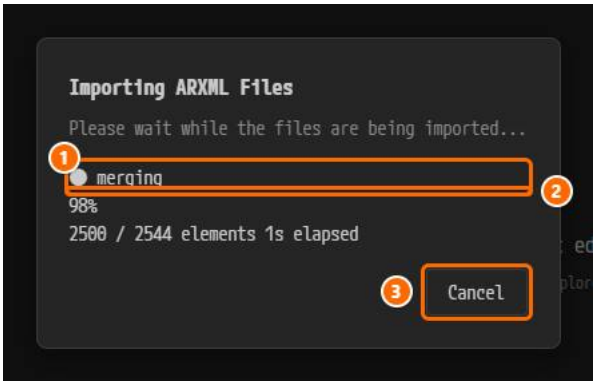
License Conditions

Export (ARXML and DBC) and Extract need **both an active license and online server validation**. Import works even when the license has expired.

13.2 Importing ARXML and DBC

13.2.1 Procedure

1. Click the **Import** icon in the **DESIGN** header. The dialog `Select files to import` opens.
2. Select the files. You can select several files at once and mix ARXML and DBC.
3. Confirm. All unsaved edits in the project are written to disk first.
4. The progress dialog `Importing ARXML Files` appears. **Cancel** stops the import and restores every file it already wrote.
5. Depending on the result, the import continues along one of three paths (13.2.3 to 13.2.5).



*① The current stage name, ② the stage row with the percentage and element counter, ③ **Cancel**.*

Source files are read in place; they are not copied into the project's `import/` folder.

13.2.2 When an Import Is Rejected

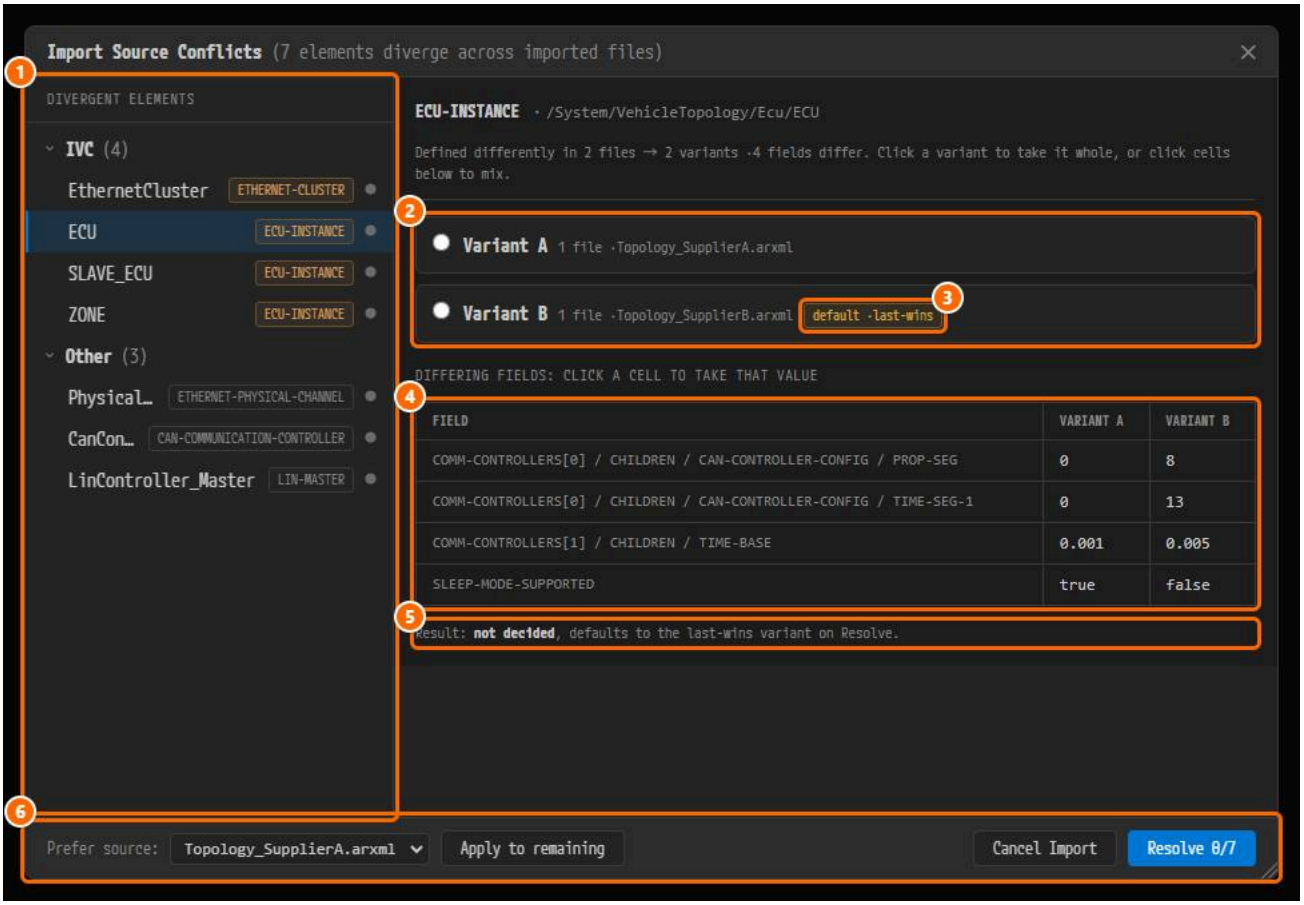
Only one import can run at a time. While a previous import is unfinished, a new one is rejected with `Import rejected: <reason>. Finish or cancel it`

13.2.3 Result 1 — Success

`Import complete: N elements imported from M file(s)` appears and the dialog closes on its own. The open editors reload the new content.

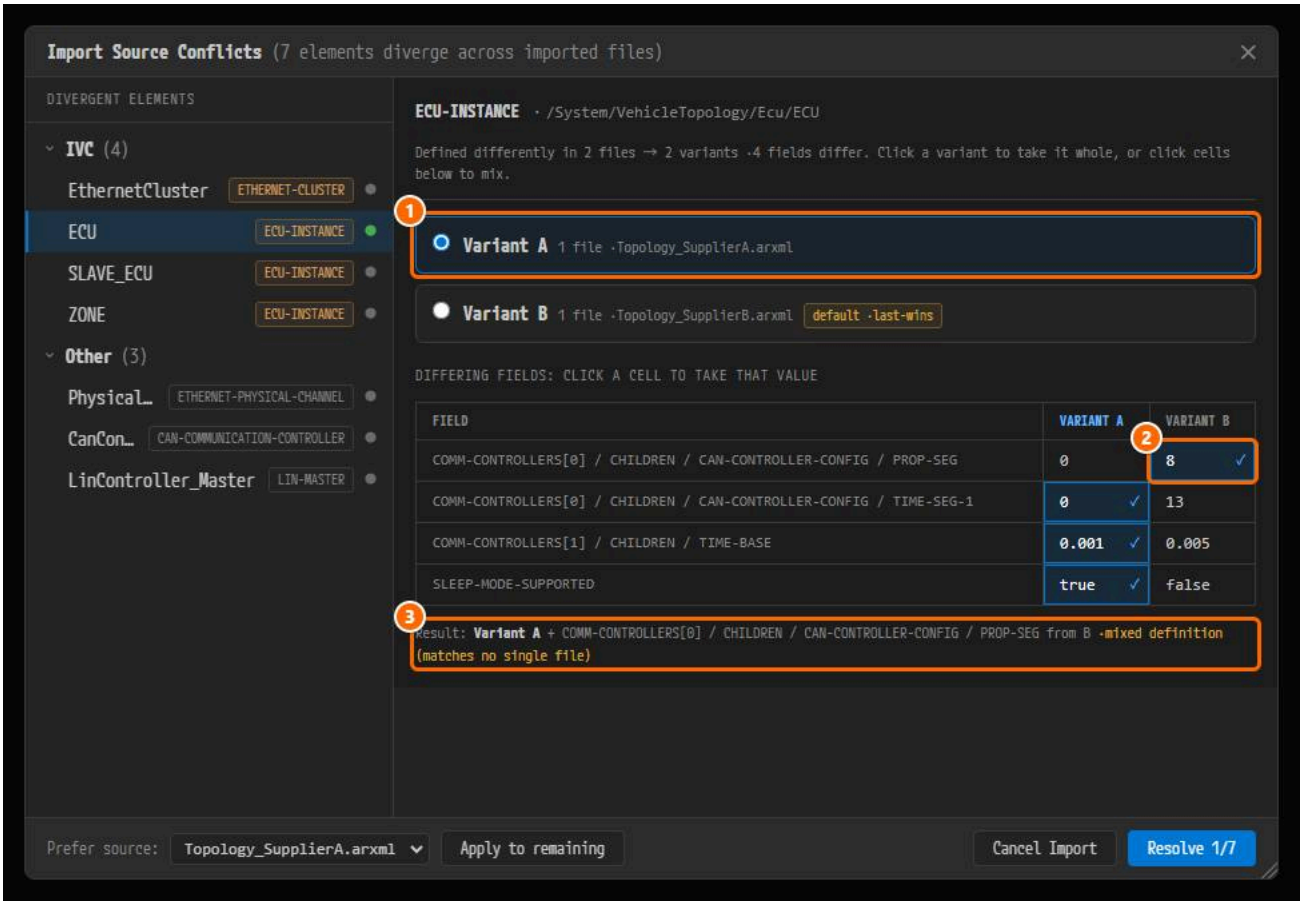
13.2.4 Result 2 — Import Source Conflicts

If several files selected in one go define the same element differently, the **Import Source Conflicts** dialog appears before anything is written to disk.

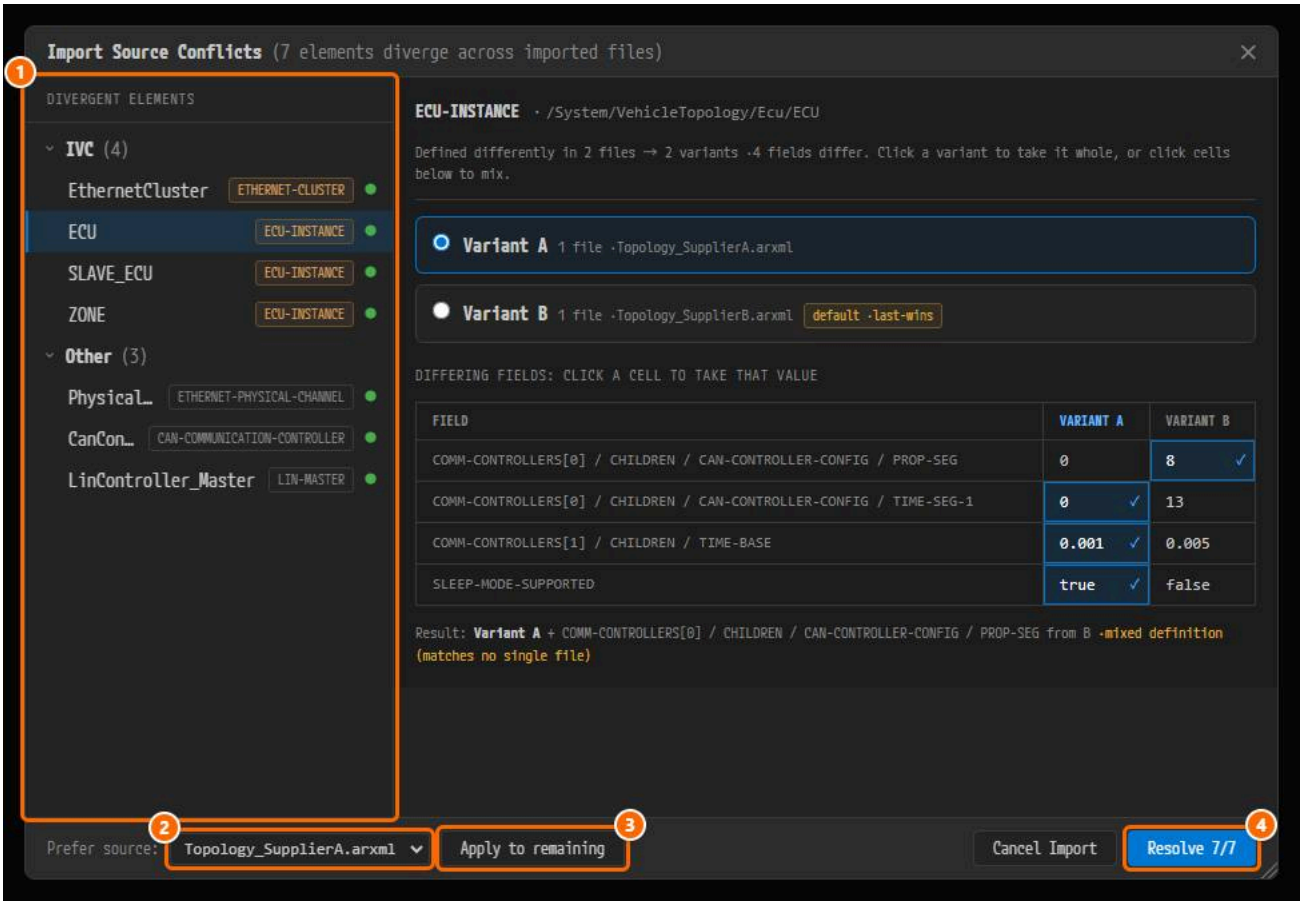


*① **Divergent Elements**, ② the variant cards, ③ the default badge, ④ the field table, ⑤ the `Result:` line, ⑥ the footer.*

1. Select an element in the **Divergent Elements** list on the left.
2. Click a variant card to take that variant whole, or click cells in the field table to take single fields.
3. To decide every remaining element at once, pick a file in **Prefer source:** and click **Apply to remaining**.
4. Click **Resolve d/N**. Undecided elements take the variant with the default badge.



① `Variant A` taken whole, ② one cell taken from `Variant B`, ③ the `Result:` line shows `· mixed definition (matches no single file)`.

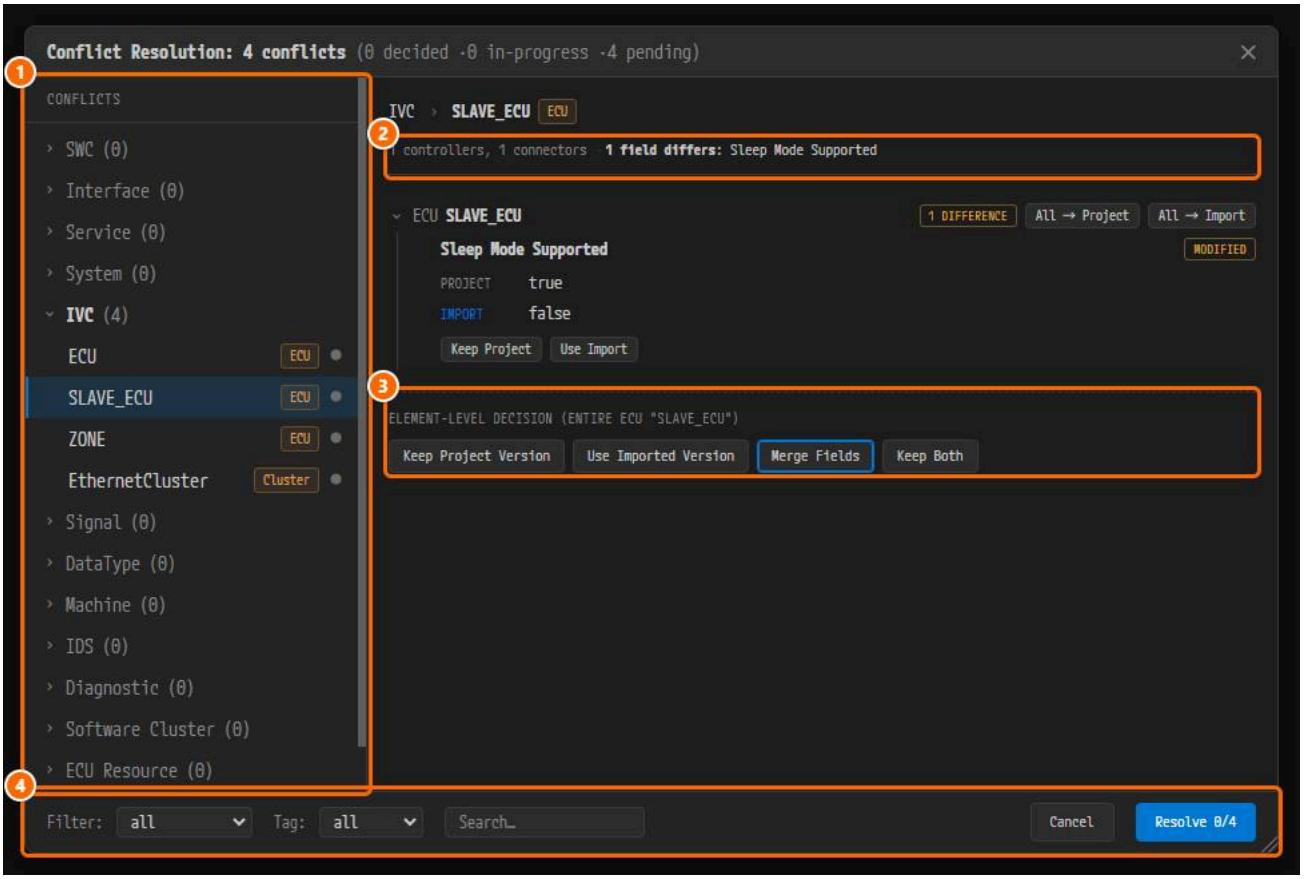


① *Every dot is lit*, ② *the file chosen in* ***Prefer source:***, ③ ***Apply to remaining***, ④ ***Resolve 7/7***.

Cancel Import stops the import; nothing has been written yet.

13.2.5 Result 3 — Conflict Resolution

When an element already in the project and an incoming element hold different values, the **Conflict Resolution** dialog appears.



*① The **Conflicts** tree, ② the summary line, ③ the element-level decision buttons, ④ the footer.*

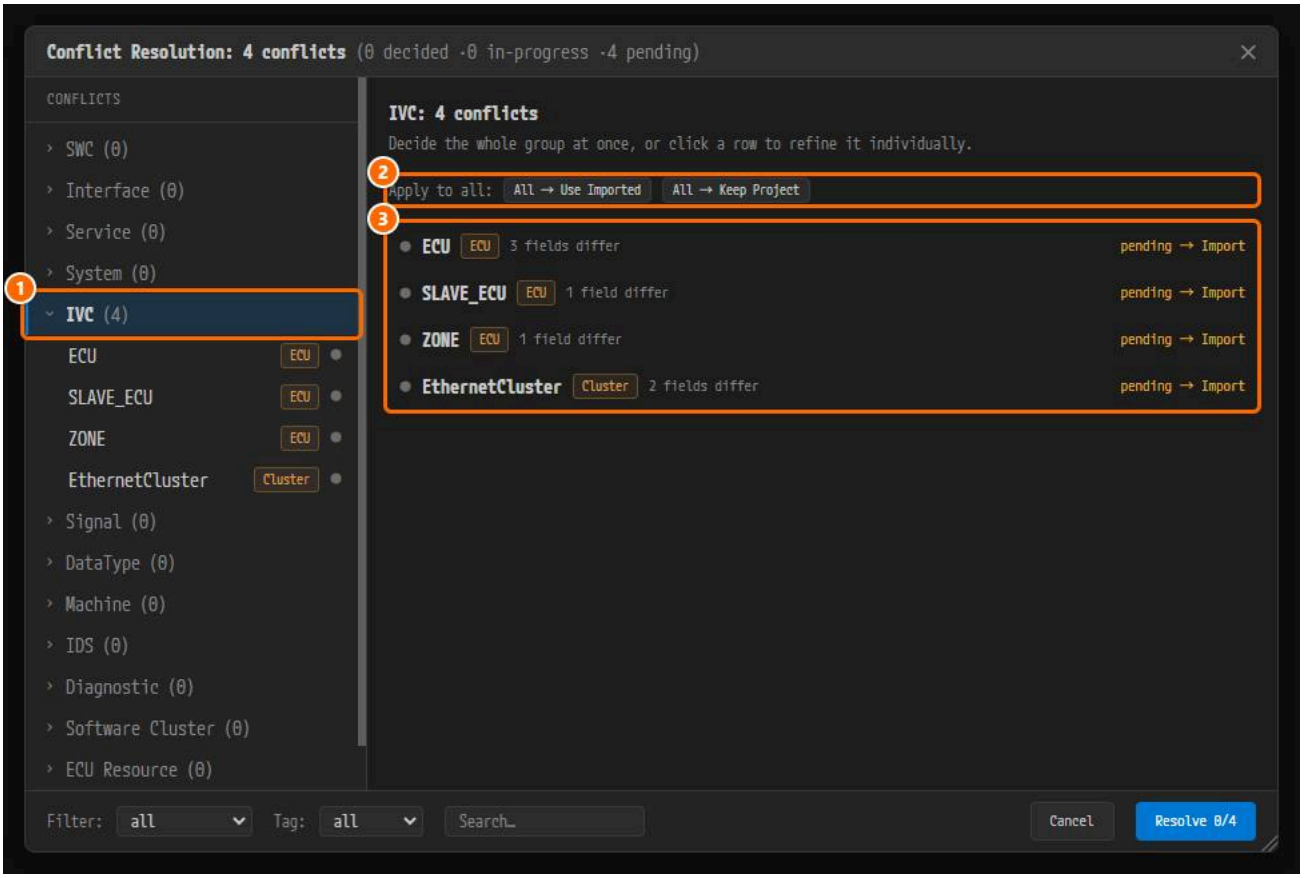
1. Select an item in the **Conflicts** tree on the left and compare the project value with the imported value.
2. Choose one of the four buttons under `Element-level decision`.

Button	Behavior
Keep Project Version	Leaves the value already in the project as it is
Use Imported Version	Replaces it with the imported value
Merge Fields	Lets you choose which side's value to use, field by field
Keep Both	Keeps both by giving the imported element a new name

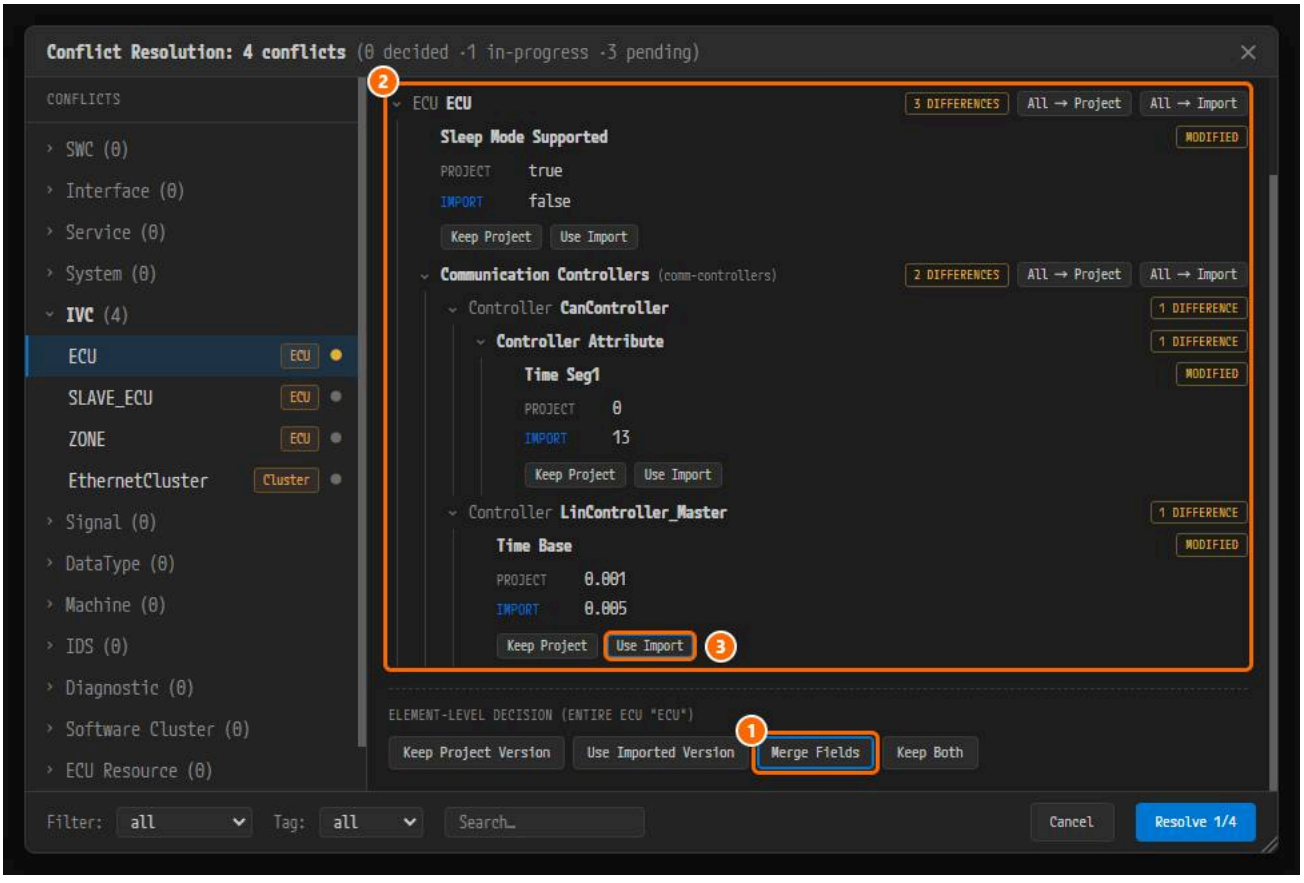
3. Click **Resolve n/N** to commit.

> **Items you have not decided are overwritten with the imported value**, not kept. If undecided items remain, the `Unresolved Conflicts` dialog asks first

To handle a whole area at once, click its header and use **All → Use Imported** or **All → Keep Project**.



*① The area header (**IVC**), ② the bulk buttons, ③ the rows with their current decision.*

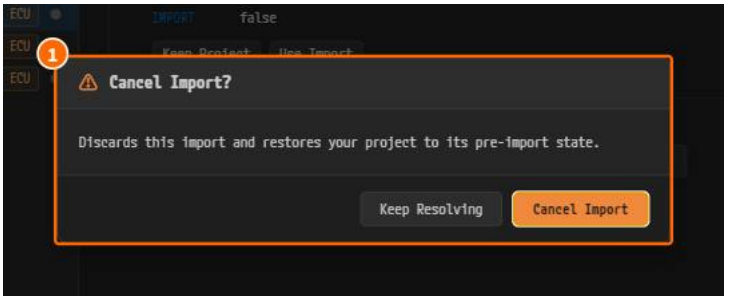


*① **Merge Fields**, ② the field tree of the item, ③ **Use Import** taken for one field.*



① *The `Unresolved Conflicts` dialog (**Cancel**, **Overwrite & Continue**).*

Cancel stops the import and restores the project to its pre-import state.



① *The `Cancel Import?` dialog (**Keep Resolving**, **Cancel Import**).*

13.2.6 What an Import Handles Together

One ARXML import handles all of these together: network topology, signals, SWC, interfaces, data types, services, system, security events, diagnostics, s configuration, and ECUC module definitions and configuration values. Definitions are read before configuration values, so both may be in the same file.

The whole import is **one undo step**: one `Ctrl+Z` returns the project to its pre-import state.

13.2.7 Re-importing Merges Rather Than Overwrites

An element already in the project is combined field by field, not replaced.

- If only one side has a value, that value is used.
- A conflict is raised only when both sides have a value and the two differ.
- Diagram coordinates and sizes are not content, so the existing layout is kept.

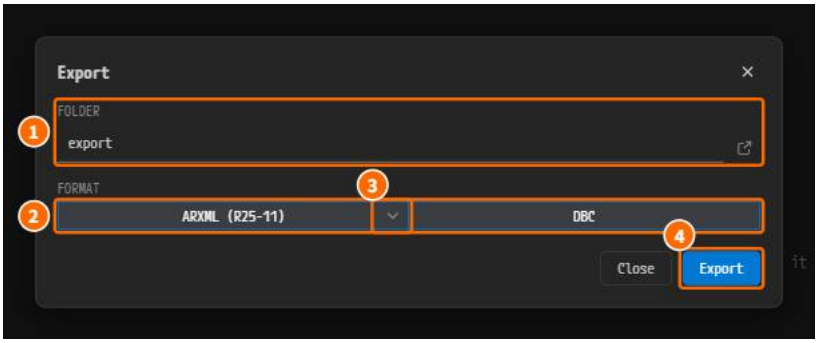
13.2.8 DBC Encoding Caveat

A DBC file is read as UTF-8 **only when it has a UTF-8 BOM**, and as ISO-8859-1 otherwise. If characters come in broken, add a BOM to the file or save it again

13.3 Exporting a Project (ARXML, DBC)

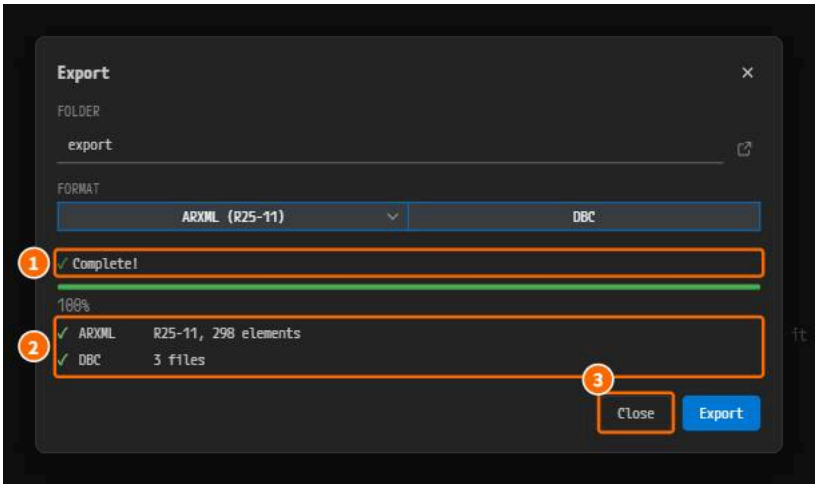
13.3.1 Procedure

1. Save your edits (**PROJECT ▸ Save All**).
2. Click the **Export** icon in the **DESIGN** header. The **Export** dialog opens.
3. Check **FOLDER**. `export` is the project's `export` folder. Type another project folder or a full path, or pick one with the icon at the right end. The icon i
4. Choose one or both of **ARXML** and **DBC** in **FORMAT**. At least one stays chosen. The arrow on the **ARXML** button lists the AUTOSAR versions; today that



*① **FOLDER**, ② **FORMAT** with both formats chosen, ③ the AUTOSAR version, ④ **Export**.*

5. Click **Export**. The formats are written one after the other and overwrite files of the same name. **Cancel** stops the run.
6. `✓ Complete!` appears, with one result line per format. Click **Close**.

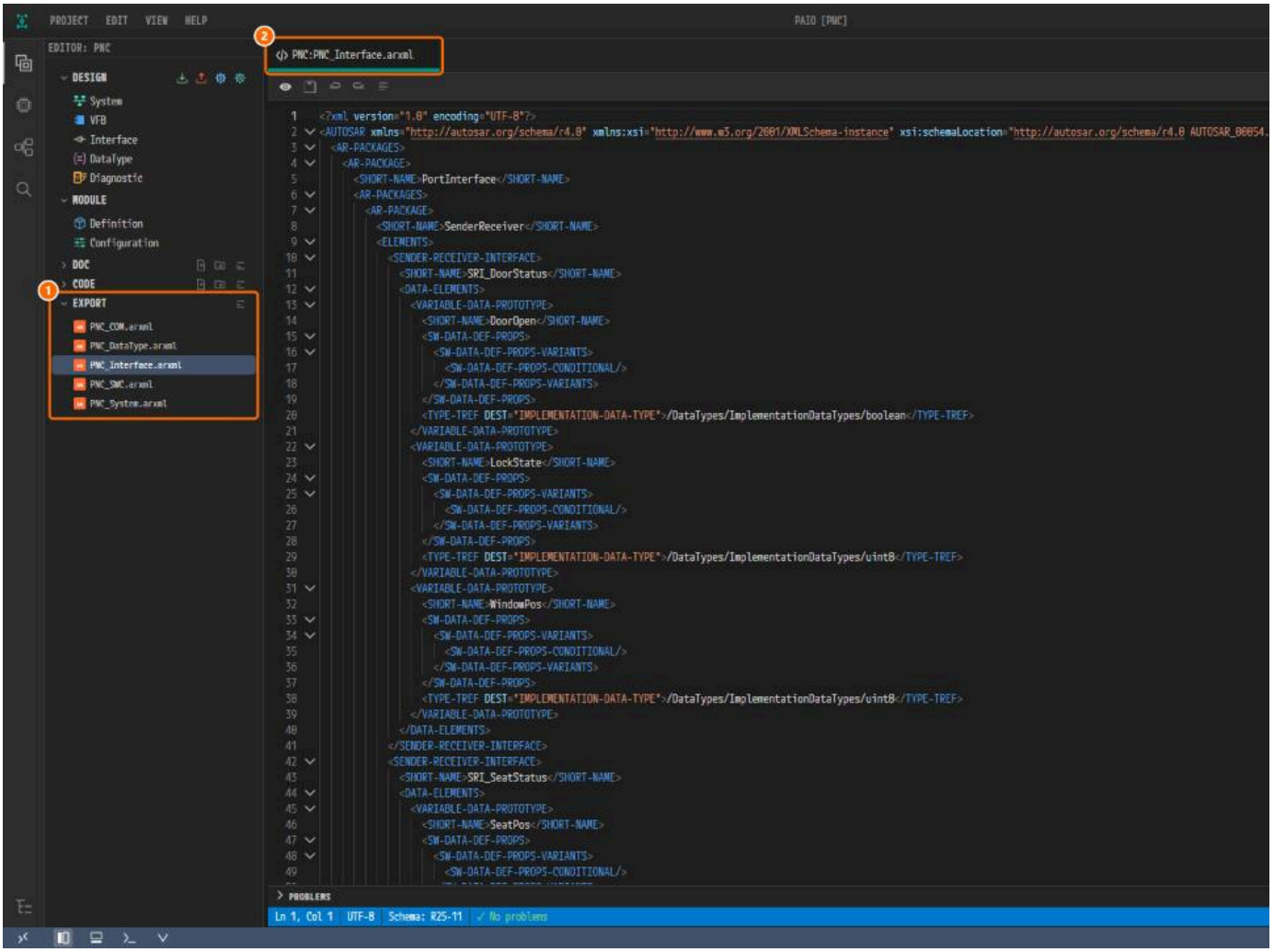


*① `✓ Complete!`, ② the result of each format, ③ **Close**.*

If a format fails, `X Finished with errors` appears and the line of that format shows the reason. The other format is still written.

13.3.2 Files That Are Created

The ARXML export writes several files named `<project>_<suffix>.arxml` (for example `MyProject_System.arxml`, `MyProject_COM.arxml`). A file is crea `_DataType` are always created. The DBC export writes one `<cluster>.dbc` file per CAN cluster (13.6).



*① The exported files in the **EXPORT** section, ② a file opened by double-click.*

13.3.3 The Data an Export Reads

The ARXML export reads **only the saved files in the `design/` folder**. Unsaved edits are not exported, even while their editor is open. Save before exportin

13.3.4 Output Specification

The AUTOSAR namespace is fixed at **R25-11**. A reference with no value is not exported.

13.3.5 Package Layout of the Result

The AR-PACKAGE structure of the result is fixed by rule.

13.3.6 Information That Does Not Go Out to AUTOSAR

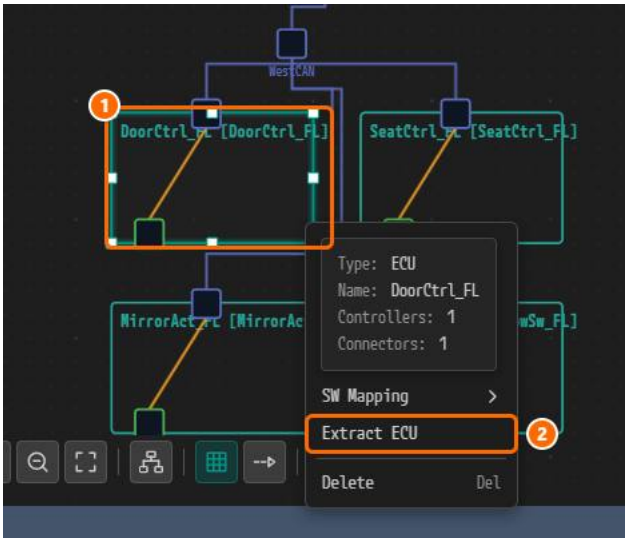
- Ownership relations, such as which ECU an ECUC configuration belongs to, travel in the ARXML ADMIN-DATA SDG and survive a re-import.
- **Diagram coordinates and screen layout do not go out to ARXML.** Re-imported into an empty project, the nodes are laid out automatically. Re-import kept (13.2.7).

13.4 ECU / Machine Extract

Extract cuts out one node (an ECU or an HPC) with only what it references and writes it to a separate ARXML. Use it to hand a specific ECU owner only th

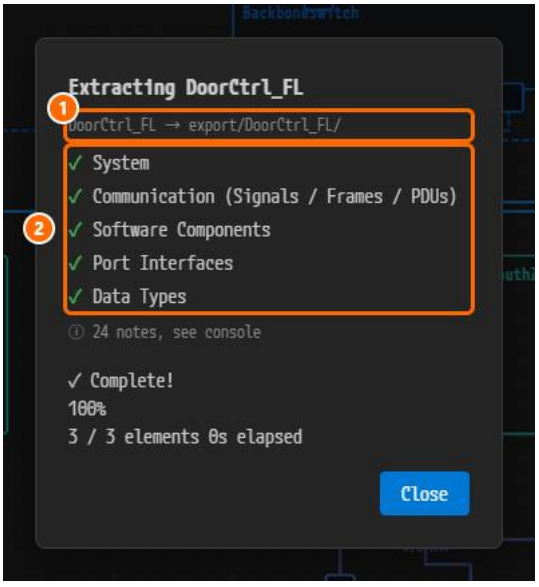
13.4.1 Procedure

1. Save your edits, then open the System editor (**System** in the **DESIGN** list of the **EDITOR** view).
2. Right-click the target node and select **Extract ECU** (Classic ECU) or **Extract Machine** (Adaptive HPC).



① *The ECU node* `DoorCtrl_FL`, ② ***Extract ECU.***

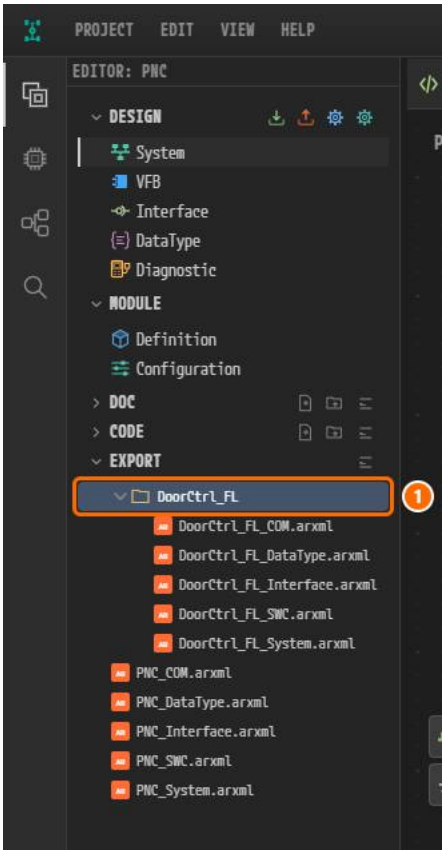
3. On success, the files that were created are listed by area. Close the dialog with **Close**.



① *The subtitle* `DoorCtrl_FL → export/DoorCtrl_FL/`, ② *the checklist of areas that were written.*

13.4.2 Result

The files go to `<project>/export/<node name>/` and use the **node name** as the prefix (`DoorEcu_System.arxml`). Only the saved model is read.



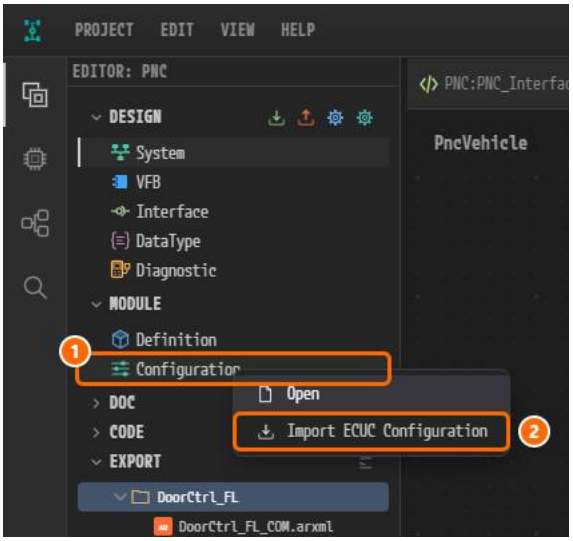
① The `DoorCtrl_FL` folder, expanded with the chevron at the left of the row.

13.4.3 What Is Cut Out

Extract keeps the target node, the frames, PDUs and signals it uses, and its ECUC configuration values. All ECUC module definitions are exported. The rem no dangling references.

13.5 Importing an ECUC Configuration on Its Own

Right-click **Configuration** under **MODULE** in the **EDITOR** view and choose **Import ECUC Configuration**.



*① The **Configuration** row under **MODULE**, ② **Import ECUC Configuration**.*

13.5.1 Procedure

1. Select **one** ARXML file in `Select ARXML File to Import ECUC Configuration`.
2. If the file contains an `ECUC-MODULE-DEF`, the module definition is imported first.

3. If a configuration already exists at the same path, the conflict dialog of 13.2.5 appears.
4. The result is reported. A skipped or rejected module is listed with its reason.

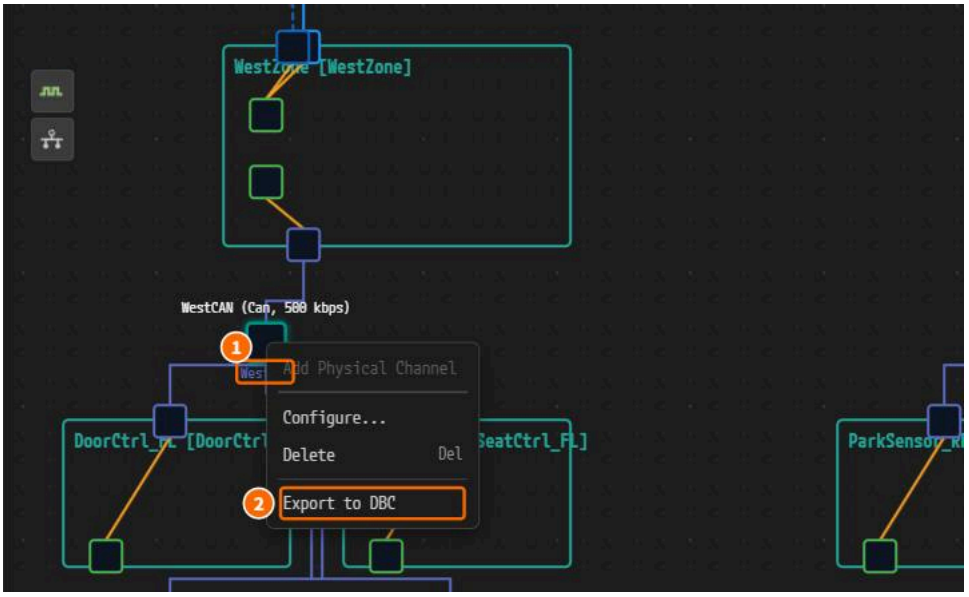
The whole import is one undo step.

13.5.2 Reasons for Rejection

If a container, parameter or reference of a configuration value is not in the definition, the whole module is rejected.

13.6 DBC

- **Import**: select a `.dbc` file in the same **Import** dialog as in 13.2. A multiplexed message comes in as a MULTIPLEXED-I-PDU, and whether a bus is CAN or
- **Export of every CAN cluster**: choose **DBC** in the **Export** dialog of 13.3. One `<cluster>.dbc` is written per CAN cluster. A project without a CAN cluster
- **Export of one CAN cluster**: in the System editor, right-click the CAN cluster and choose **Export to DBC**. The **Exporting to DBC** dialog shows `DBC expor` file is `export/<cluster>.dbc`.



*① The CAN cluster `WestCAN`, ② **Export to DBC**.*

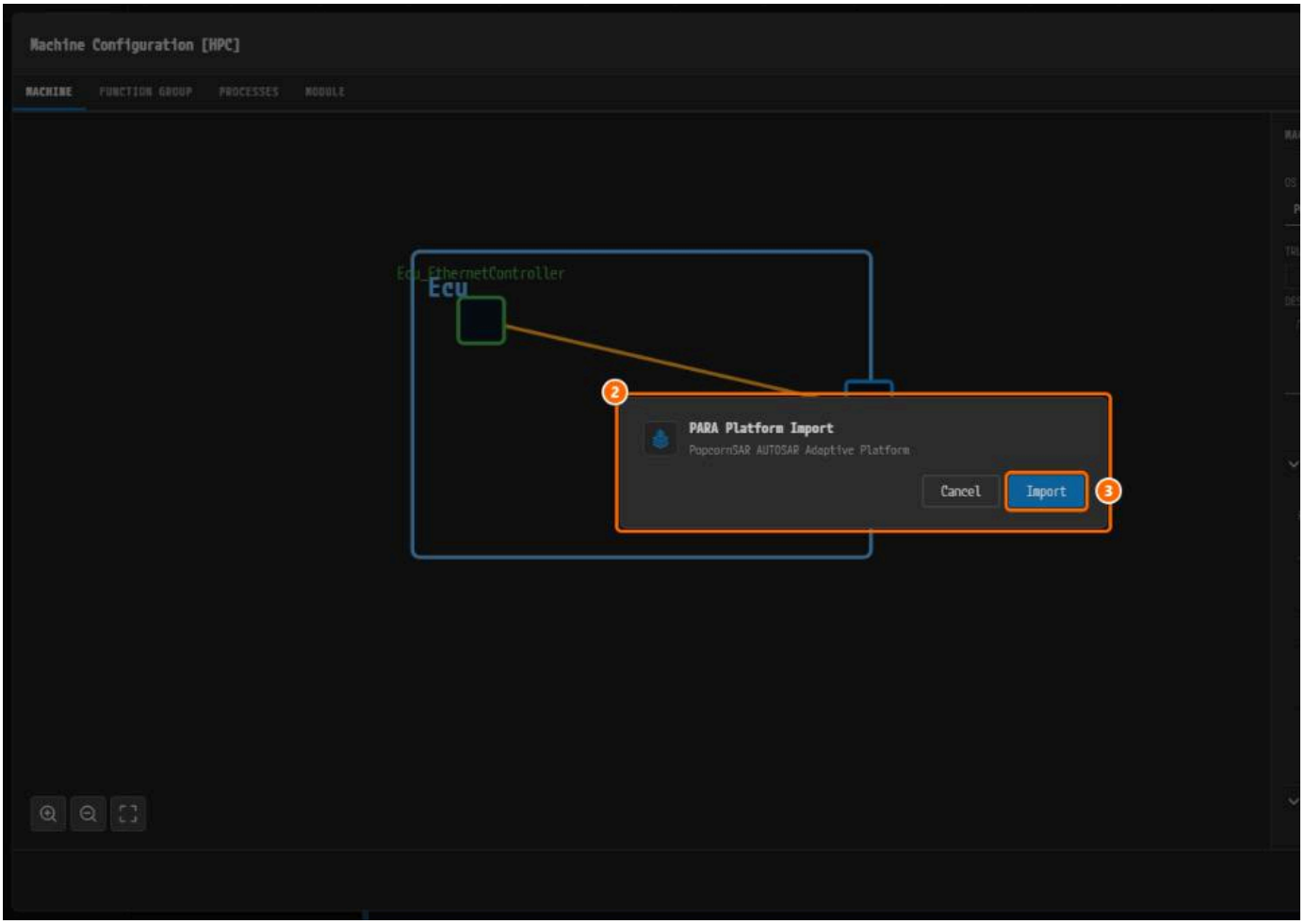
A DBC file holds only the nodes with a CAN connector. When the project has CAN clusters, a frame that none of them carries is in no DBC file.

13.7 Adding the PARA Platform Base Configuration

This adds the base configuration of the PopcornSAR AUTOSAR Adaptive Platform to the project. The set shipped with the product is used; you choose nc

13.7.1 Procedure

1. In the VFB editor, open the **Machine Configuration** dialog of a composition mapped to an HPC.
2. Click **PARA Platform Import** in the header.
3. Click **Import** in the confirmation dialog. From here it behaves like an ARXML import.



*① **PARA Platform Import**, ② the confirmation dialog, ③ **Import**.*

13.7.2 Limits and Constraints

The operation is rejected while another import is running, and when the license has expired.

13.8 Situations You Will Often Meet

- Export shows only an upgrade notice: the license has expired. Import remains available.
- An existing value changed after **Resolve**: undecided items are overwritten with the imported value (13.2.5).
- The Extract or export is missing your latest edits: save first.

13.9 Importing in an SSH Window

In a window connected over SSH, an import writes to the SSH server.

- The files are picked on the server first: the picker opens at the top of the window on the project folder. Click a file and click **Open**. In **Import**, `Ctrl`+cl file dialog instead. **Import** and **Import ECUC Configuration** both work this way.
- If the connection closes during the import, the import fails with `Import failed: The remote connection closed during the import. Part of import again.` Nothing is written to your PC. Reconnect and import again.
- Closing the connection yourself while an import runs asks first; **Close Connection** cancels the import. See 17.7.

Validation

What This Chapter Covers

This chapter explains real-time validation in PAIO: how to open the panel, when validation runs, how to read the list, and how to jump to a problem and fix
constraints, and some checks without a standard number were added by PAIO. They span network configuration and communication, software structure
configuration and service deployment, diagnostics, Software Clusters and Vehicle Packages, the Intrusion Detection System, and ECU configuration. Seve
rest **warning**.

- Intended readers: users who edit the model and want to clear errors before export
- Prerequisites: a project is open. See 04. Workspaces and Projects.

Validation has no run button. It runs as soon as the project opens and again whenever you edit the model.

14.1 Opening and Closing the Validation Panel

1. Click the **V** button in the status bar at the bottom of the window (tooltip `Toggle Validation`). The panel opens below the editor area.
2. To close it, click **V** again or **Close Panel** in the panel header.

V is the only entry point; the menus, the command palette and the shortcuts have no validation item.

Reading the Status Bar Badge

- No badge: no violations.
- A number: errors and warnings together. It is red when at least one error is present.
- `!`: validation could not run. This does **not** mean "no problems".

The badge next to the `VALIDATION` title in the panel header counts errors only.

14.2 Which Project Is Validated

A PAIO window holds one project, and validation always targets that project. To validate another project, open it in its own window. With no project open,
`Explorer to run validation.`

14.3 When Validation Runs and What It Reads

Validation runs automatically:

- when the project opens
- whenever you edit the model
- after you import ARXML
- when a design file of the project changes on disk

It reads the entire design data of the project, unsaved edits included. Parts you have not built yet are not reported as errors.

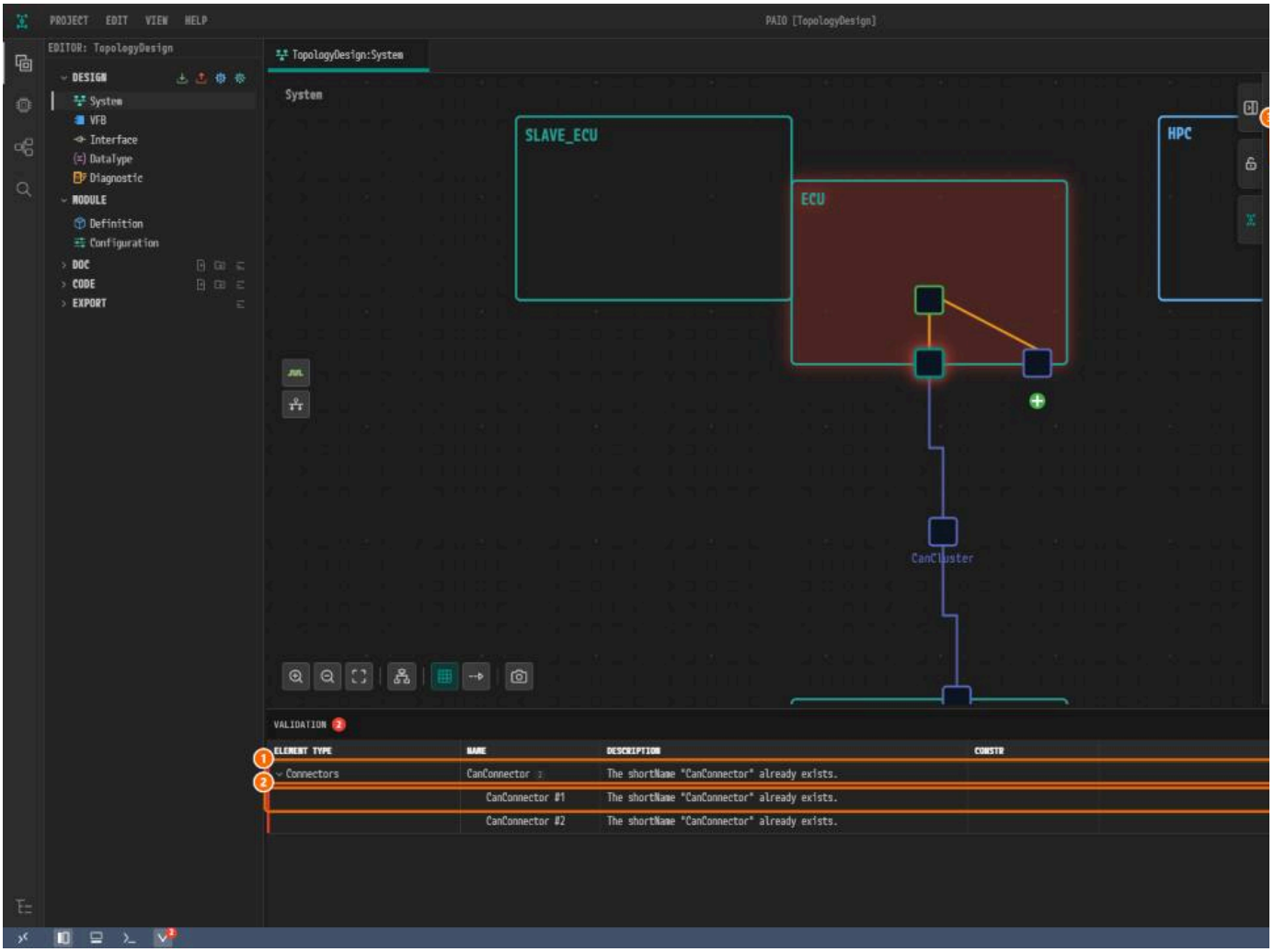
14.4 Reading the List

The list has four columns: **Element Type**, **Name**, **Description** and **Constr** (the AUTOSAR constraint number). A red bar on a row marks an error, an orange bar
checks that have no standard constraint number.

Group Rows

Sibling elements under the same parent that share a SHORT-NAME are collected into one group row with a count badge.

1. Click the group row. The child rows `#1`, `#2` and so on appear.
2. Click a child row to mark that element.



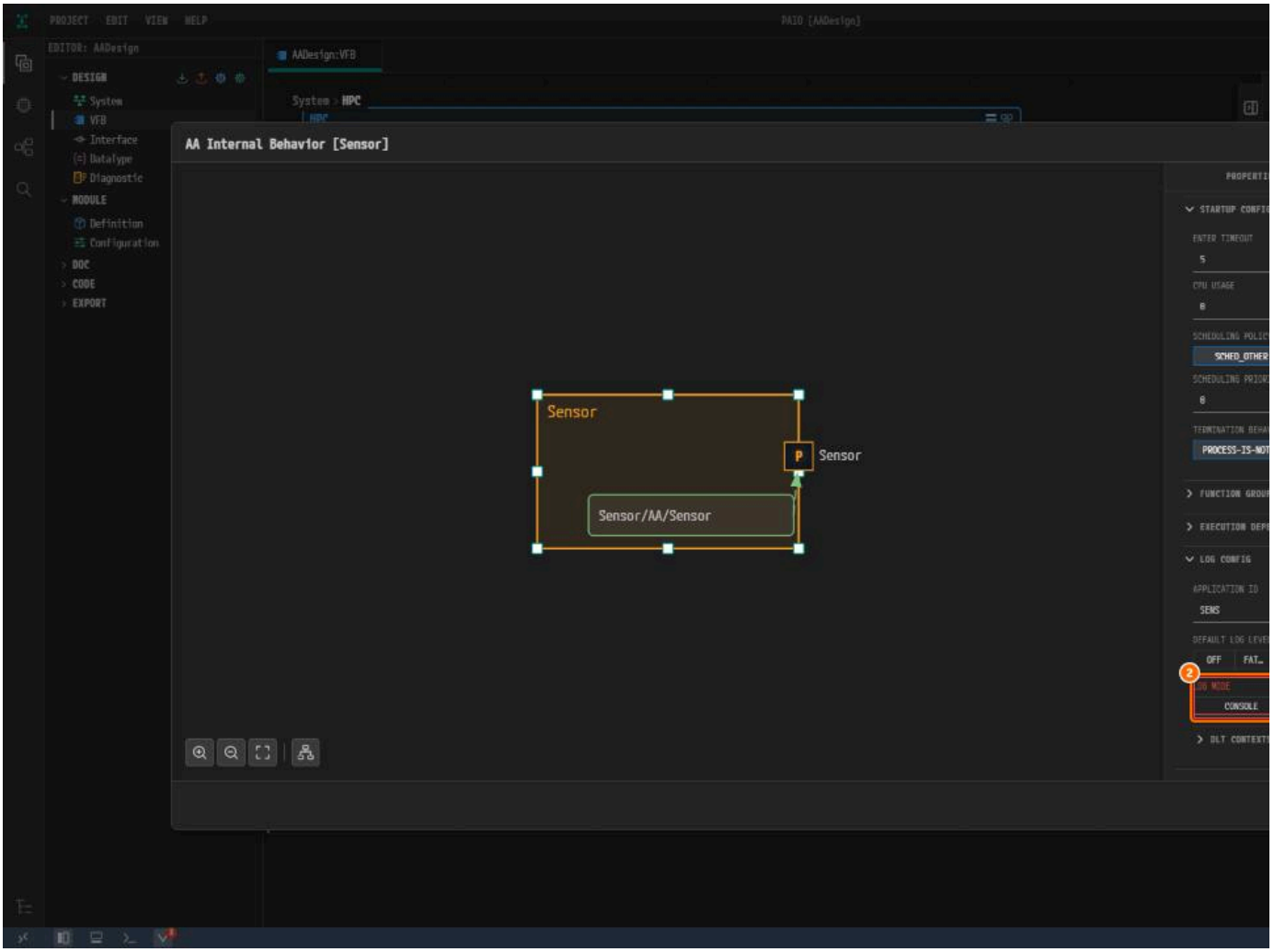
① The group row `Connectors` / `CanConnector` with its count badge `2`, expanded. ② The clicked child row `CanConnector #1`. ③ The NAME field of that c

When the List Is Empty

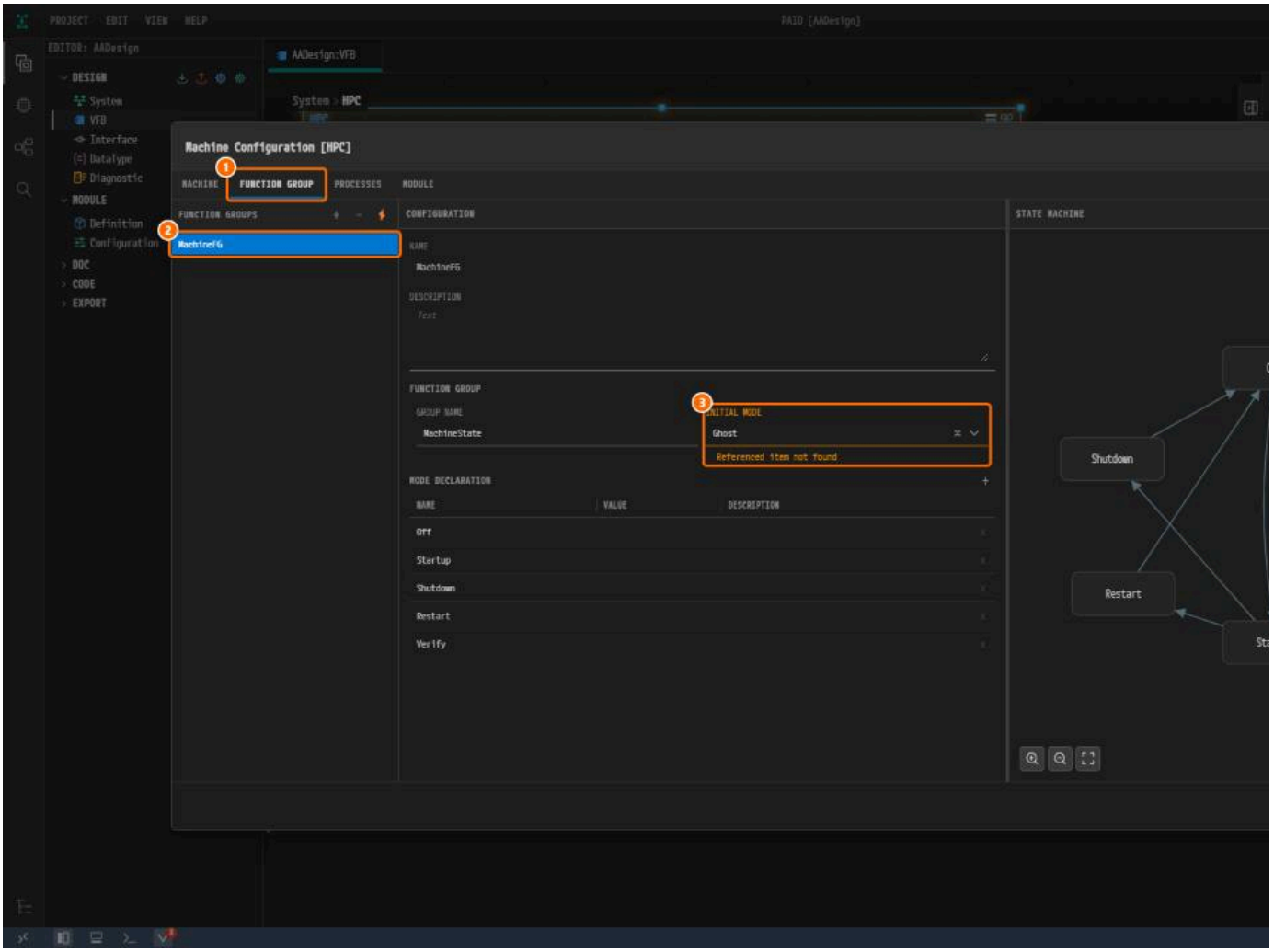
`No validation issues.` means there are no violations. `Validation could not run. See the developer console for details.` means validation fa

14.5 Jumping to a Violation

1. Click a row. The editor tab that owns the element opens, and the element is selected.
2. A violation marker stays on the element in the row's severity color: red for an error, orange for a warning. If the target is an input field, the field recei
3. Elements edited inside a dialog (**Machine Configuration**, **AA Internal Behavior** and so on) open all the way down to that dialog.

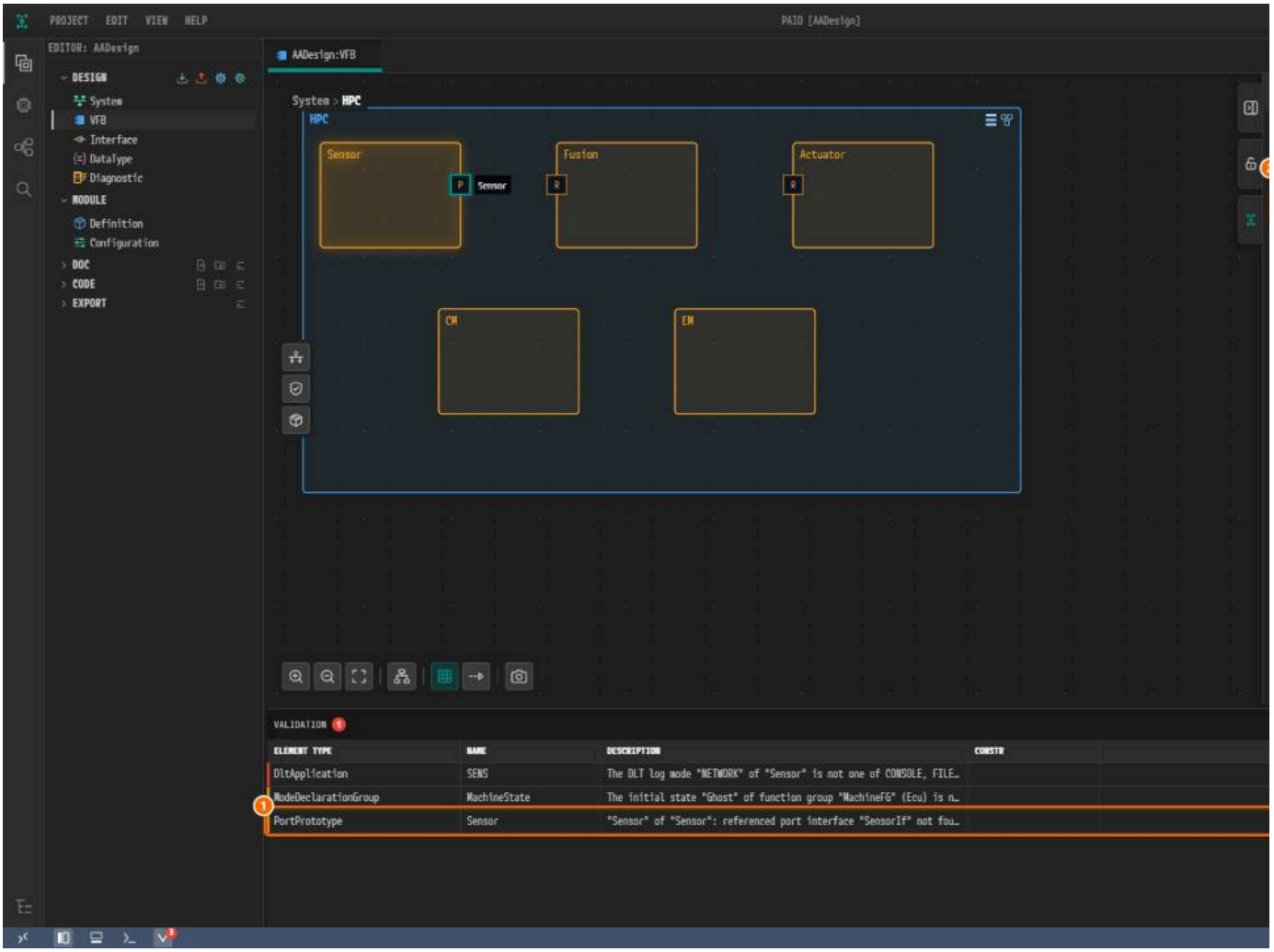


*A field inside a dialog. ① The **EXECUTION** tab with **LOG CONFIG** expanded. ② `LOG MODE`, which holds a value that is not allowed, marked in red.*

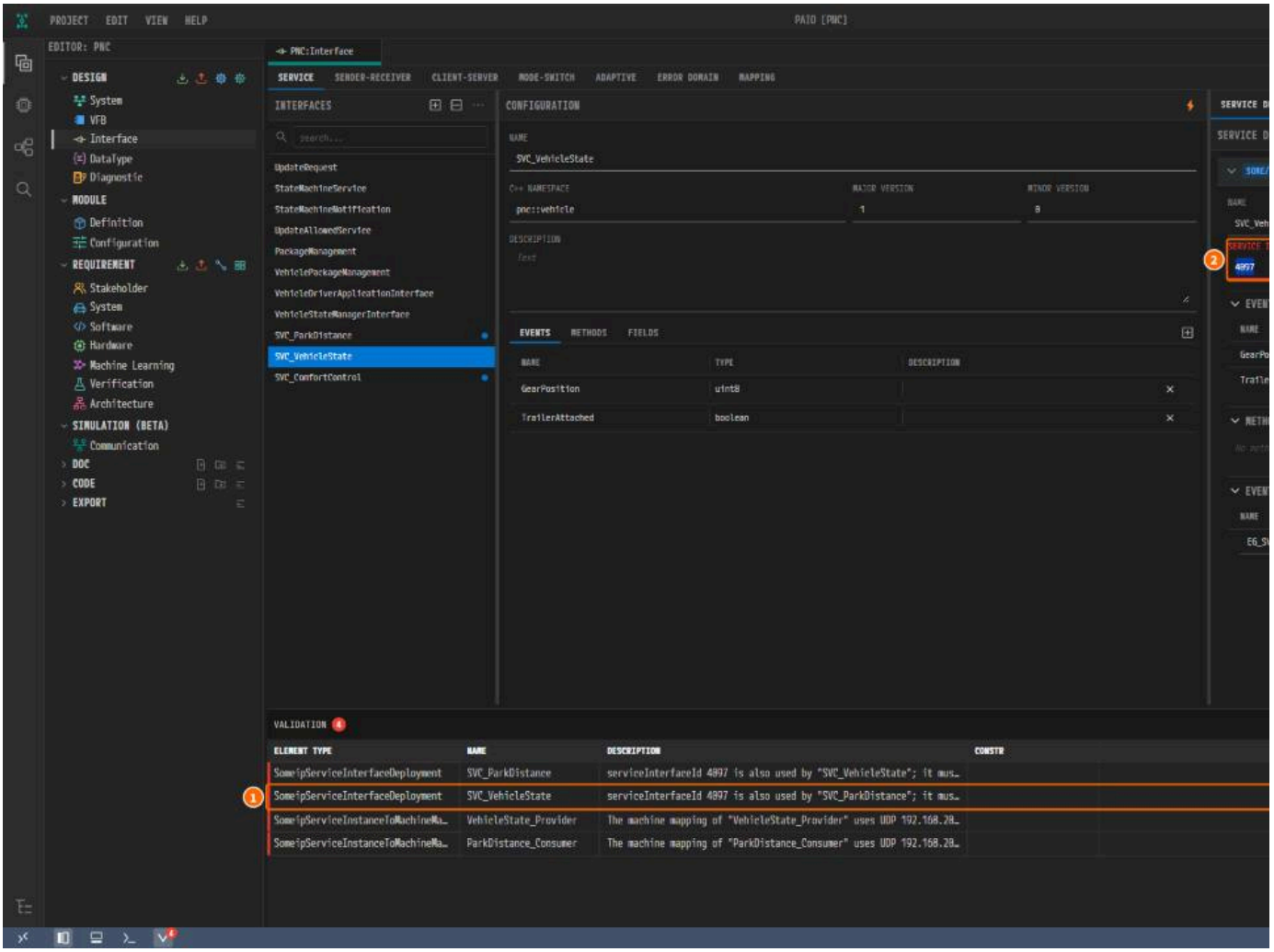


*A row can open a dialog without marking a field. ① The **FUNCTION GROUP** tab. ② Select `MachineFG` yourself. ③ `INITIAL MODE` shows `Ghost` with `Refere` check.*

4. Correct the value. Validation runs again, and the fixed row leaves the list.

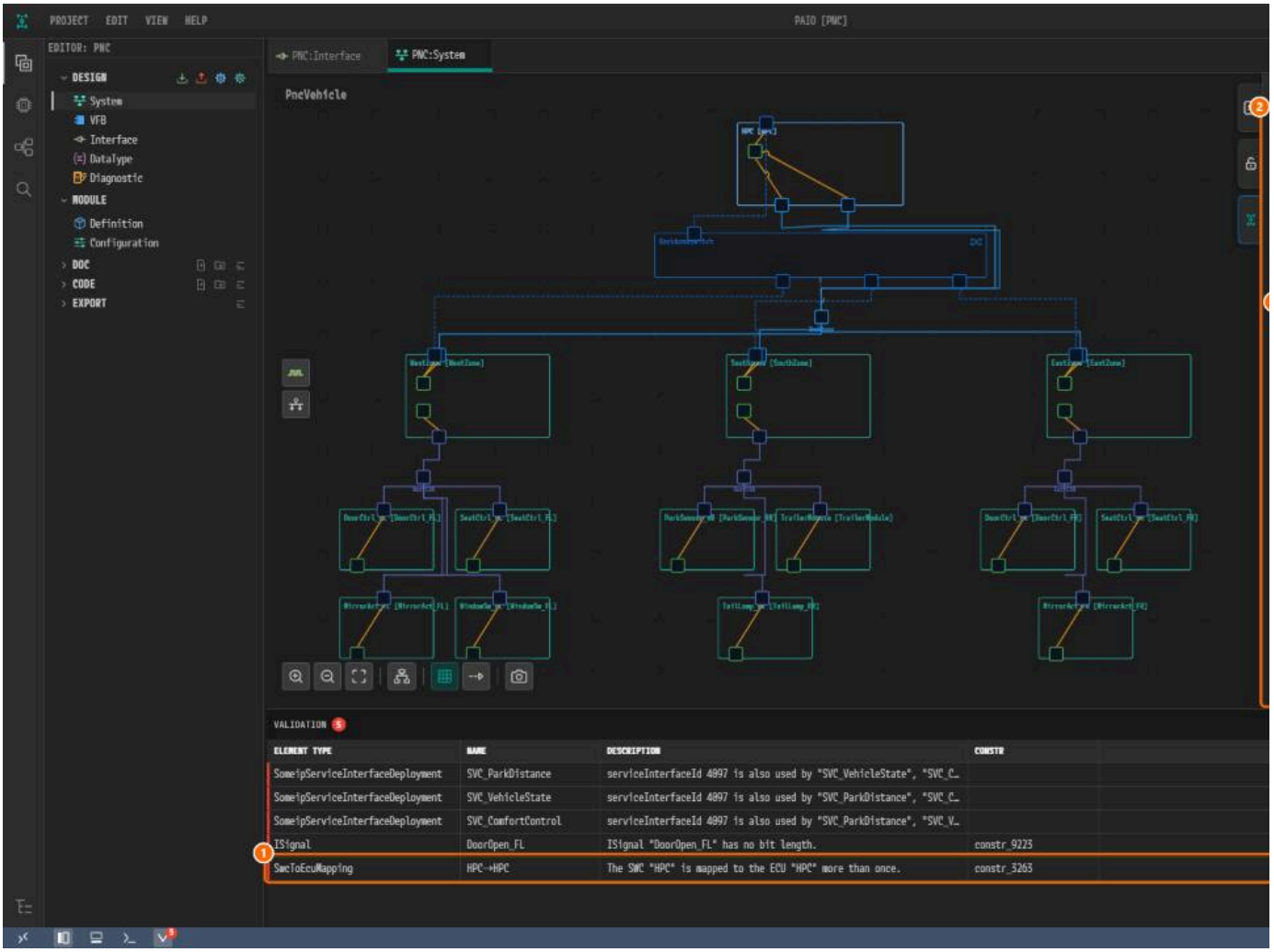


① The VFB editor selects the port `Sensor`. ② The `PORT INTERFACE` field, marked in orange because its target does not exist.



The same works outside the diagram editors. ① Click the row: the `PNC:Interface` tab opens with the interface selected. ② Only the `SERVICE ID` field is

Only one element is marked at a time. Some rows have no single field to mark; the editor then only selects the element. A row about a SWC↔ECU mapping System editor.



① The `SwcToEcuMapping` row. ② The SYSTEM panel. ③ ***ECU MAPPING****. No field is marked. See 11. SYSTEM Panel.*

Editors You Can Jump To

The editor that opens depends on the area of the element: for example the DataType editor for data types, the Interface editor for port interfaces, the \ editor for nodes and connectors, and the Diagnostic editor for DEXT elements.

Rows That Do Not Navigate on Click

ECUC rows (such as `EcucParameterValue`), `RuleFailure` rows and group header rows do not open an editor.

Passing validation does not mean the exported ARXML is schema-valid. See 13. Import and Export.

14.6 Required and Broken References

Reference fields in every editor and dialog mark two states, and the validation panel reports the same two:

- A reference the design requires is empty: the field is red, and the panel reports an **error**.
- The value names an element that does not exist: the field is orange with `Referenced item not found`, and the panel reports a **warning**.

The clear button (×) of a reference field empties the value in the model, not only on screen.

AI Assistant

What This Chapter Covers

PAIO's AI chat creates and modifies the model from natural language. This chapter explains what it does, how to write a request, and how to check and re

- Audience: users who want to create or modify the model from natural language
- Prerequisites: an active license and at least one LLM provider (15.2). To modify a model, the window must have a project open.

> **The AI drafts, the engineer decides.** What the AI produces is always a draft. PAIO blocks malformed results and gives you a way to revert, but it does nc check the result in the editors and in the 14. Validation panel.

15.1 What the AI Does for You

PAIO has two AI features: **AI chat** (this chapter), which uses the provider you connect and choose, and the AI nodes of the **AI Workflow**, which always use for AI chat does not change what the AI Workflow uses.

AI chat changes only the part of the model you ask about. It can touch:

- network topology: ECUs, HPCs, communication clusters, connectors
- signals, PDUs, frames
- data types and units
- port interfaces and service deployment
- SWCs and compositions, SWC-to-ECU mappings
- ECUC module definitions and configuration values
- the AI Workflow graph: its nodes, connections, notes and settings (21. AI Workflow)

You can also ask questions that do not modify the model.

15.2 Preparation — Connecting a Provider

1. Click the gear button in the chat header (tooltip `Manage AI accounts`). The **AI Providers** dialog lists five providers.
2. Prepare a provider outside PAIO. For example, sign in to the Claude CLI in a terminal with `claude auth login`. PAIO itself has no log-in button.
3. Click **Use** on an available provider to make it active. The active provider carries an **ACTIVE** badge.



*① The status dot: green is available, gray unavailable. ② The active provider with **ACTIVE**. ③ The status line. ④ **Use** on an available provider. ⑤ **Set API key**. The caret that expands a row.*

15.2.1 Claude Model and Effort

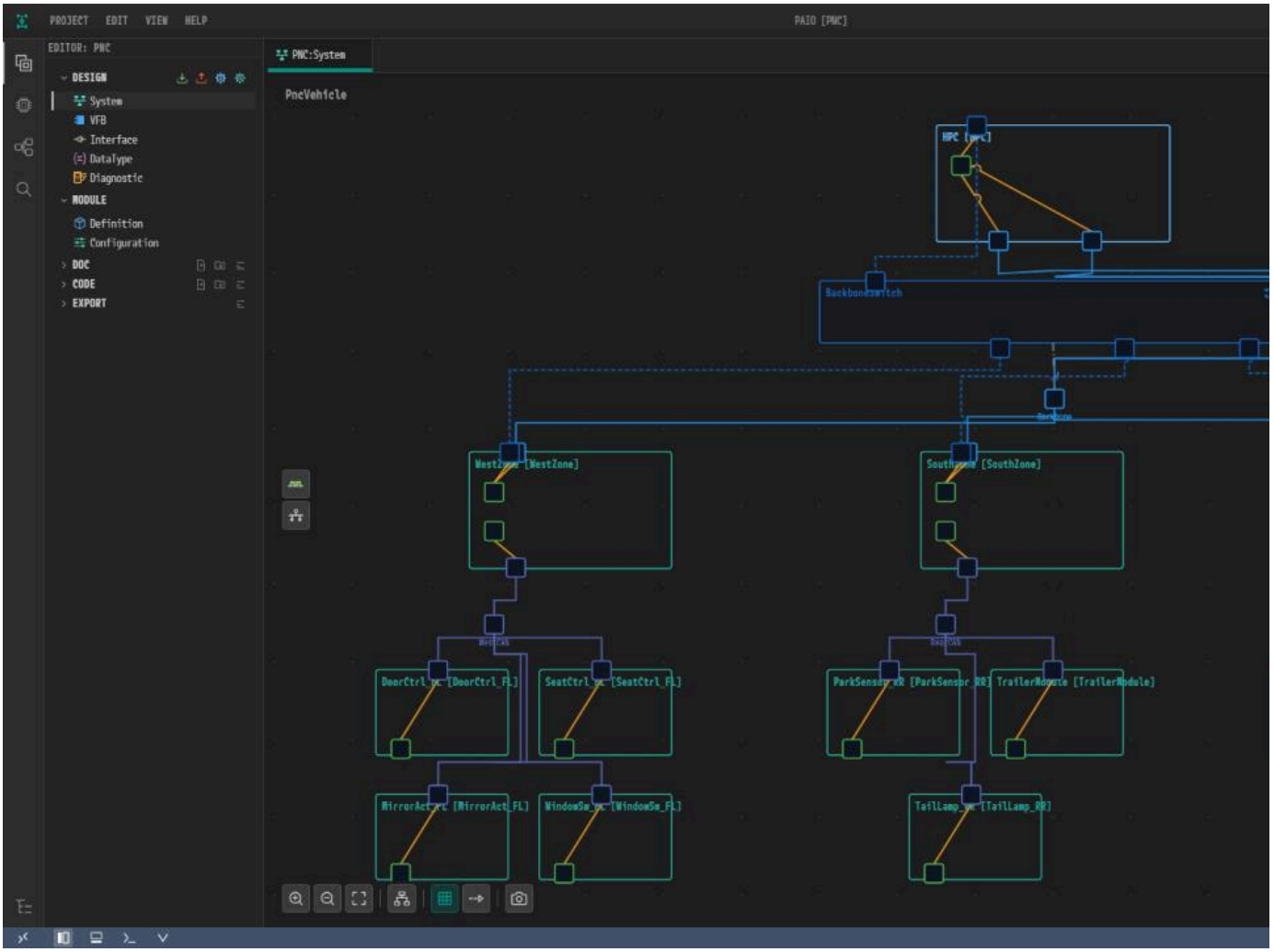
Expand the **Claude [OPUS]** row with its caret and choose **MODEL** and **EFFORT**. The defaults are **Opus** and **Medium**. A change applies from the next message



*The run settings with **MODEL** set to **Opus** and **EFFORT** to **Medium**.*

15.3 Using AI Chat

1. Click the robot button at the right of the top bar (tooltip `AI Chat Panel`). The chat panel **PAIO Robo** opens on the right.
2. Type a request and press `Enter`. `Shift+Enter` starts a new line.
3. To interrupt an answer, click **Stop response**.

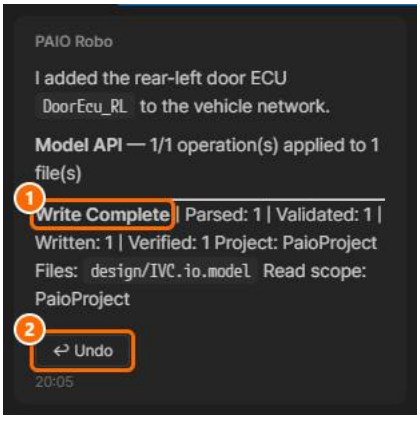


*① The robot button. ② The title **PAIO Robo**. ③ The answer language. ④ The provider in use. ⑤ The input field. ⑥ The send button.*

The language selector offers **KO**, **EN**, **JA** and **ZH**, and the assistant answers in the chosen language.

15.3.1 The Basic Flow

1. Write your request with the project open.
2. Read the answer. If the request modified the model, the result is written straight to the files, and a summary (**Model API**, **Write Complete**, `Project:`,
3. Open each editor and check the result. Click **V** in the status bar to check the validation panel as well.
4. If you are not satisfied, revert (15.3.3).



*① **Write Complete** with the counts and the file written. ② The **↩ Undo** button.*

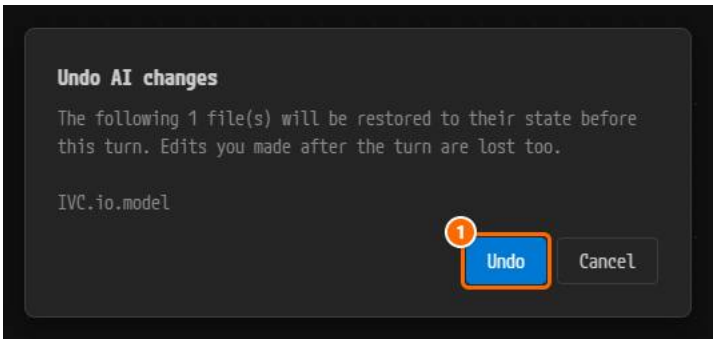
PAIO rejects a document with an empty or invalid name or a wrong structure, and re-sends the request automatically up to twice.

15.3.2 Which Project Gets Written

The AI writes into the project open in that window, whichever editor tab is open. The `Project:` line of the summary names the project that was written. W

15.3.3 Reverting Changes

1. Click ↩ **Undo** below the answer.
2. The confirmation **Undo AI changes** lists the files to be restored. Edits you made after that answer are lost too.
3. Click **Undo**.



① ***Undo*** *restores the file;* ***Cancel*** *leaves it as it is.*

You can also revert with `Ctrl+Z`; a change written by the AI is one undo step. The backup that ↩ **Undo** uses is kept for 30 minutes only, and not across a

15.4 How to Write a Good Request

The wording of your request mostly decides the result. Four rules are enough.

Name the Target Explicitly

The AI cannot see what you selected on screen. Say what to create and where, for example: Add a door ECU named `DoorEcu_RL` to the CAN cluster `CAN_I`

Ask for One Thing at a Time

Split a request that carries several concerns, and check each result before you move on.

State the Conditions

Write down any fixed value up front: a count, a bit length, a cycle time, a platform. If you leave it out, the AI picks a value on its own.

Ask First When You Are Unsure

If you only ask instead of telling the AI to modify the model, the files stay untouched.

15.5 The QuickStart Workflow

QuickStart is not offered in PAIO 1.0.1. To start from a blank slate, create a project with **New Project** (4.3.1) and build the model in the editors, or with AI cha

15.6 Limits to Know About

- An active license is required (see 20.8).
- The conversation and the model content are sent to the provider you selected. Ollama communicates only with the local server.
- A large project is not sent in full, so name the target specifically.
- What is generated is a draft and must be reviewed by a person.
- The AI Workflow's AI nodes always send to Anthropic Claude (21).

15.7 Long Requests, Stop and Links

15.7.1 A Request That Runs Several Rounds

A large request, such as designing a whole VFB, is written over several rounds. PAIO asks for the rest by itself, up to 10 rounds per request, and each rounc remains after that, reply `continue`. When what the AI wrote adds new validation findings, PAIO also asks it to fix them, and the answer lists any that rema

15.7.2 Stopping a Request

Stop response stops writing from that moment. Rounds already written stay in the files. ↩ **Undo** below the answer reverts the whole request, every rounc

15.7.3 Links in an Answer

A link to a project file opens the file as a double-click in the Explorer does. An `http` or `https` address opens in the system browser.

15.8 AI Chat in an SSH Window

In a window connected over SSH (17. Remote Development), AI chat writes to the SSH server, not to your PC.

1. Send the request as usual. While PAIO copies the project from the server, the answer area reads `Reading the project from the SSH server…`.
2. Every file the AI writes goes to the server first. ↩ **Undo** and `Ctrl+Z` restore the server files.
3. The conversations are stored in the server project.

Code Editing, Build, Terminal and Documents

What This Chapter Covers

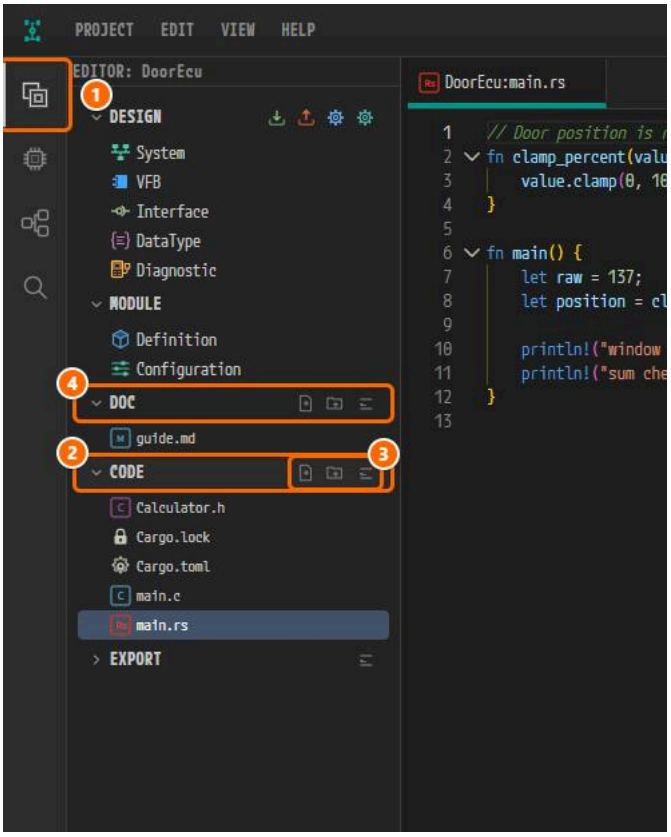
This chapter covers the general development tools: how to edit, build and run the C, C++ and Rust sources in the project's `code` folder, the integrated te the `doc` folder.

- Audience: developers who write and build application code inside PAIO
- Prerequisite: a project is open
- On a remote target (an SSH host or a vECU container), read 17. Remote Development (SSH, vECU Container) alongside this chapter.

16.1 The Code Editor (C / C++ / Rust)

16.1.1 Which Files Open Here

1. Click the top icon of the left rail (tooltip **Design**) to open the **EDITOR** view.
2. Expand the **CODE** section (files of `code/`) or the **DOC** section (files of `doc/`).
3. Double-click a file to open it.

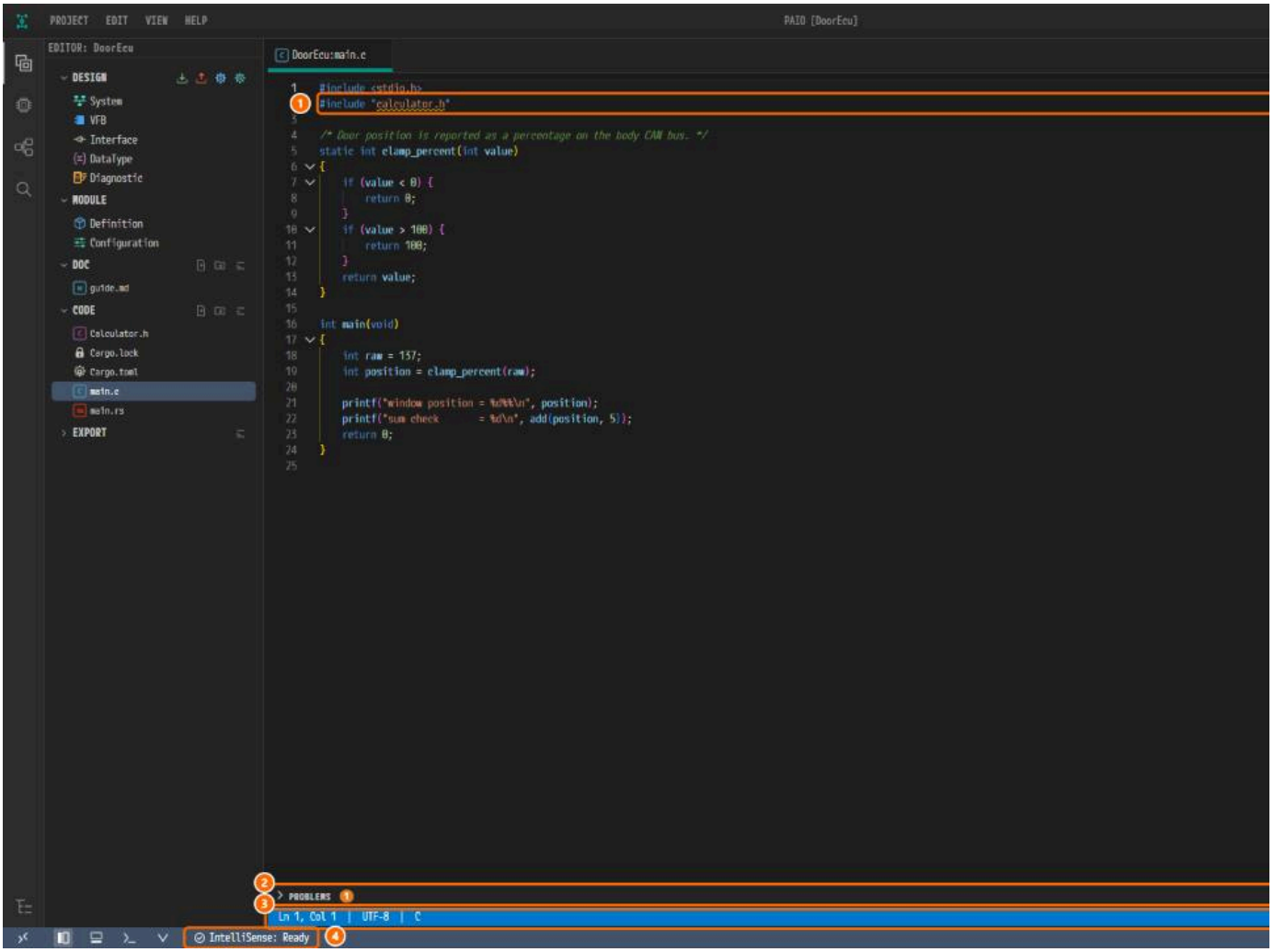


*① The rail icon, ② the **CODE** header, ③ **New File**, **New Folder**, **Collapse All**, ④ the **DOC** header.*

The extension alone decides which editor opens, in any folder.

16.1.2 Layout and Shortcuts

The **PROBLEMS** panel sits below the edit area. When a build starts, an **Output** panel appears below that.



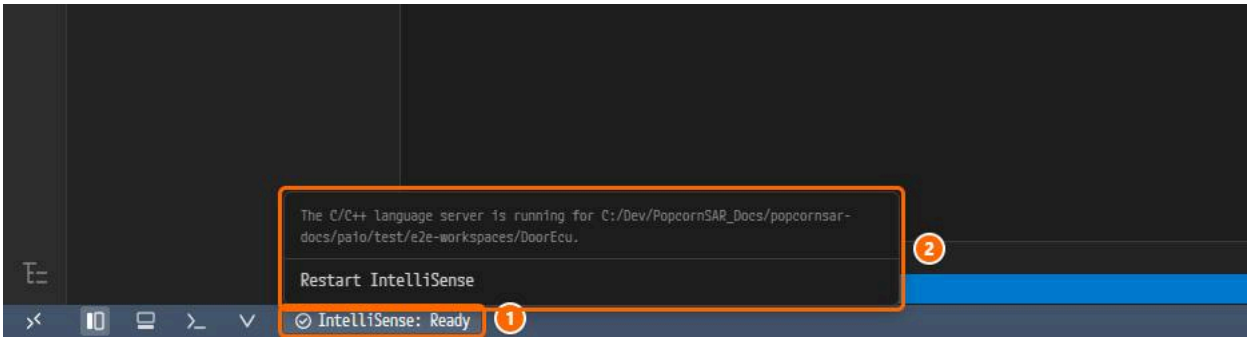
*① A warning squiggle on line 2, ② **PROBLEMS**, ③ the editor bar, ④ `IntelliSense: Ready`.*

Shortcut	Action
`Ctrl+B`	**Build**: compiles the current file only
`Ctrl+F5`	**Build & Run**: detects the project type, then builds and runs
`Ctrl+S`	Save
`F12`	Go to definition

16.1.3 Language Servers (LSP)

Completion, hover, go to definition and formatting come from language servers: clangd for C and C++, rust-analyzer for Rust. Both ship with the app. Ren

Once a C or C++ file is open, the status bar shows the item **IntelliSense**. When completion and diagnostics quietly stop working, click it and choose **Resta**

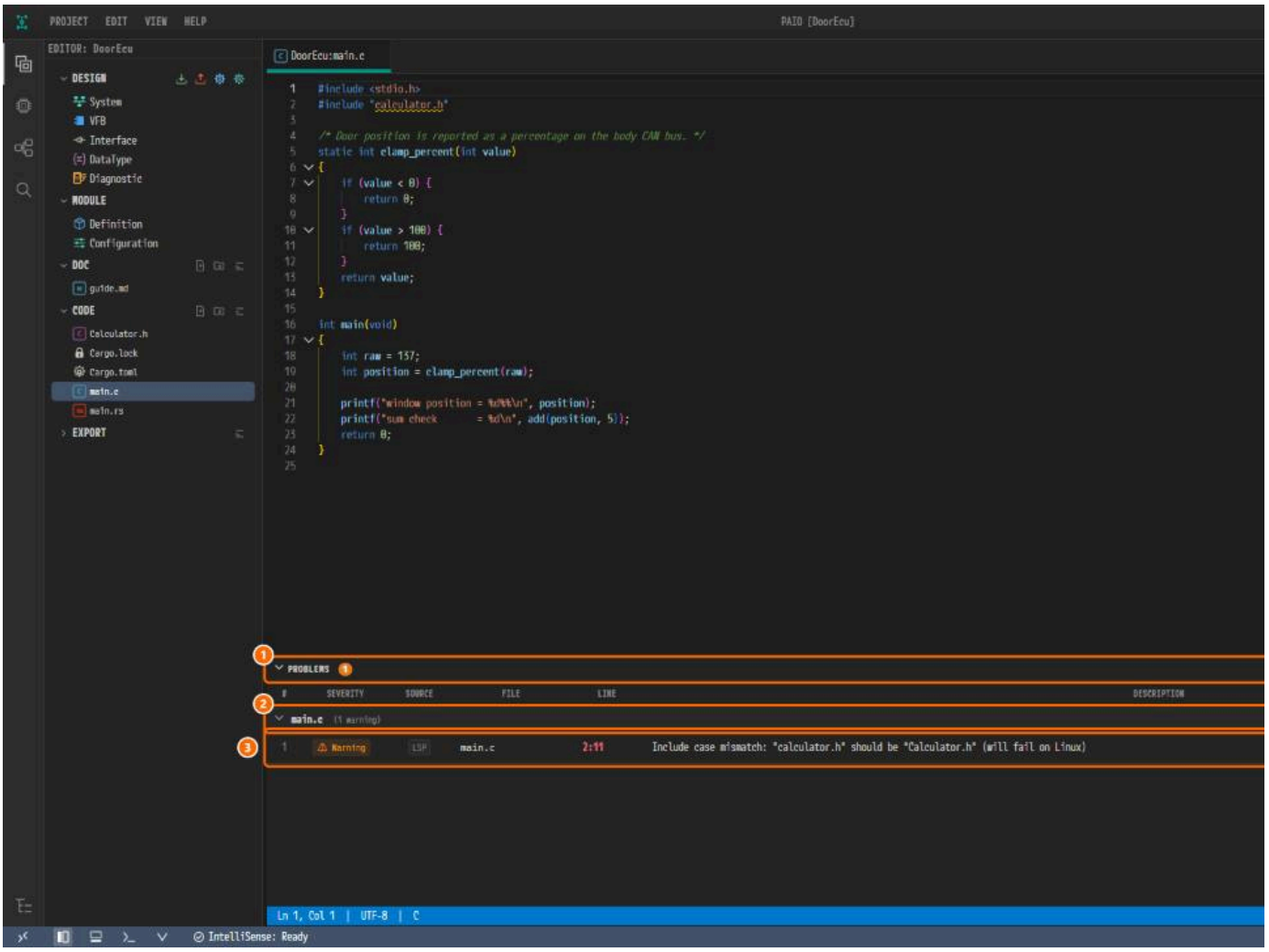


① `IntelliSense: Ready`, ② *its menu with* ***Restart IntelliSense****.*

`code/Cargo.toml` is generated automatically and overwritten, so keep a hand-managed Cargo project in a subfolder.

16.1.4 The PROBLEMS Panel

The diagnostics list, grouped by file. Click an entry to jump to that location.



① The header with its count, ② the file group, ③ an entry.

Errors from the code editor's **Build** and **Build & Run** do not arrive here; they appear in the **Output** panel. Compiler diagnostics from **Build ECU** do arrive, wit

16.1.5 Saving and External Changes

Press `Ctrl+S` to save. When an open file with unsaved edits changes on disk, the editor keeps your edits and shows a warning; saving then writes them

16.2 Build and Run

Build and **Build & Run** build the open file, or its CMake or Cargo project, in the environment of the window. An ECU is built in its vECU with **Build ECU**: see 17

16.2.1 Build and Build & Run Are Different

- `Ctrl+B` (**Build**) compiles only the file that is currently open, with gcc or g++.
- `Ctrl+F5` (**Build & Run**) is the project-level build.

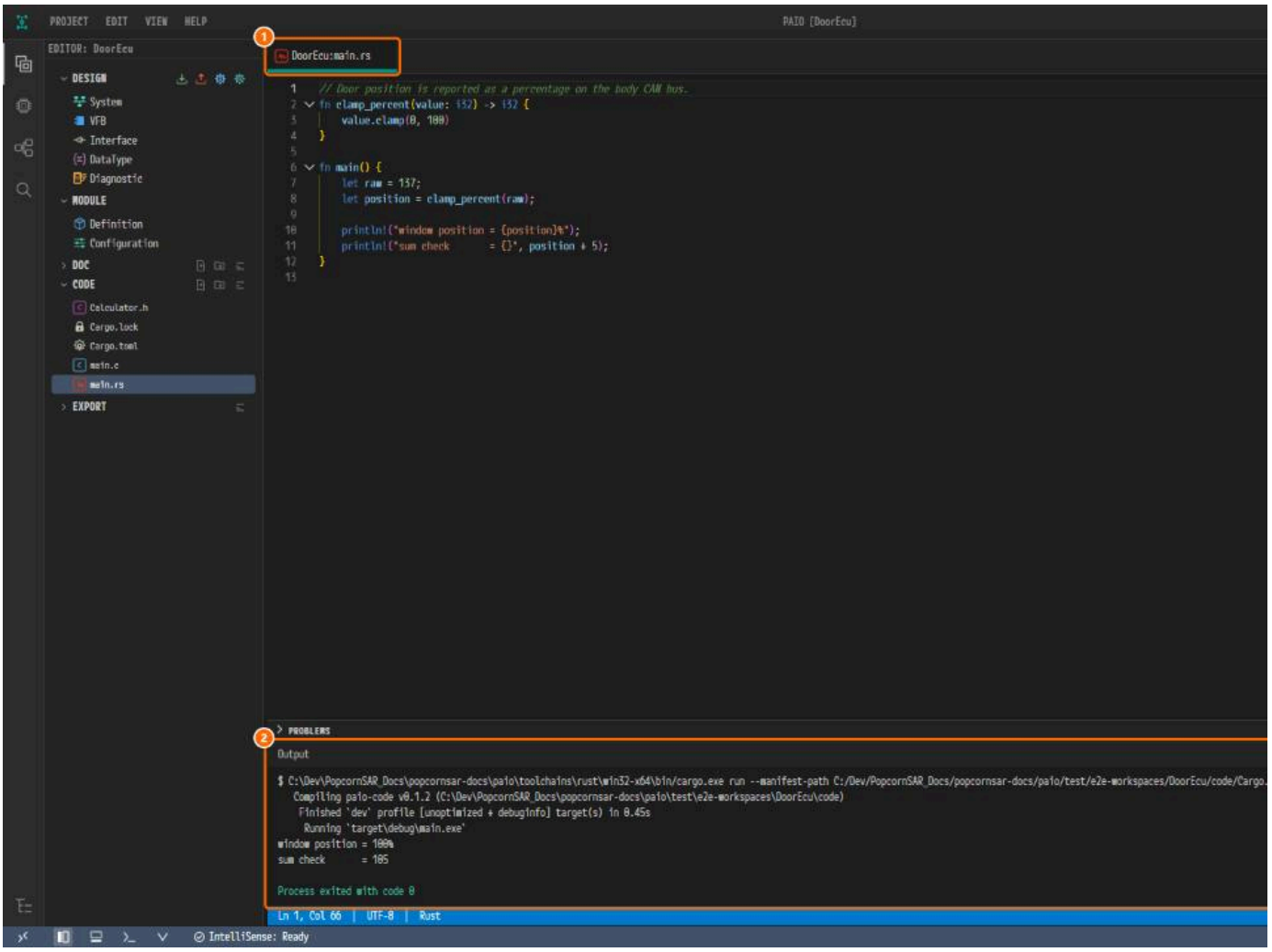
Both save unsaved edits first.

16.2.2 The Order Build & Run Uses to Detect a Project

From the current file's folder upward, PAIO uses the first match: a `CMakeLists.txt` (configure and build, no run), a `Cargo.toml` (`cargo run`), or a single

16.2.3 Commands Executed and Their Output

The build type is fixed to `Debug`. The **Output** panel shows the command, the result in real time, and `Process exited with code <number>`.



*① The tab where `Ctrl+F5` was pressed, ② the **Output** panel.*

There is no button to cancel a build in progress.

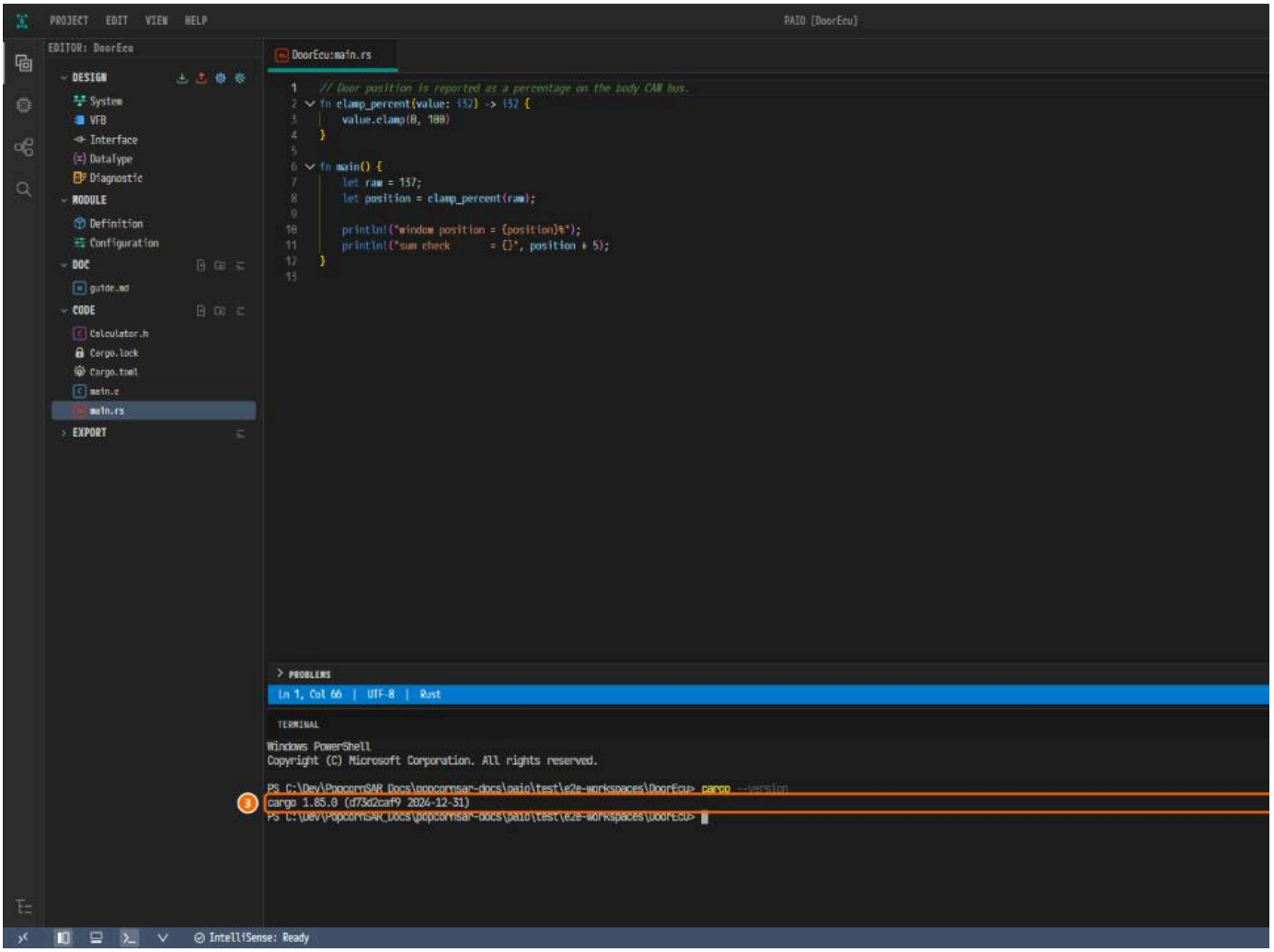
16.2.4 The Toolchain Used

The Rust toolchain ships with the app. PAIO uses a bundled tool first and falls back to the tool on the system `PATH`.

16.3 The Integrated Terminal

16.3.1 Opening and Operating It

1. Press `Ctrl+`` to show or hide the terminal panel. A new terminal starts in the project folder.
2. To start in a particular folder, right-click it in the **Explorer** and choose **Open in Integrated Terminal**.



① ***Decrease, Increase, New Terminal, Split Terminal, Kill Terminal, Close Panel.*** ② *The terminal list.* ③ `cargo --version` *answered in the terminal.*

16.3.2 Which Shell Opens

A vECU container window opens the shell inside the container, and an SSH window the remote SSH shell. The default local shell is PowerShell on Windows.

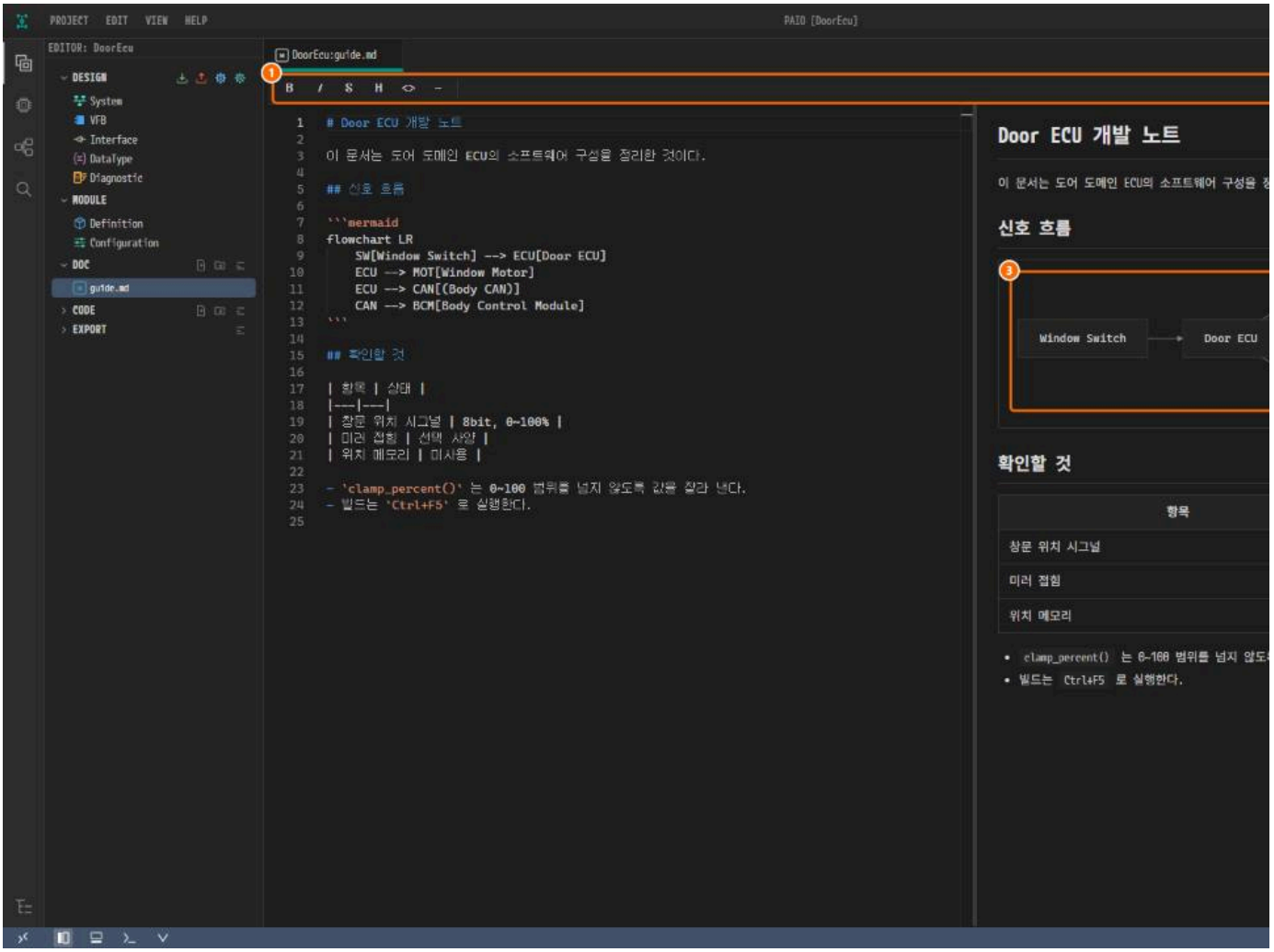
16.3.3 The Bundled Toolchain and PATH

A local terminal has the bundled toolchain paths on `PATH`, so you can type `cargo` directly. This does not apply to SSH shells; build tools for remote targe

16.4 The Markdown Document Editor

16.4.1 Editing and Preview

Any `.md` file opens in the Markdown editor, with a live preview beside it.



① The formatting toolbar, ② the three view buttons, ③ a `mermaid` block drawn in the preview.

Apply bold and italic with the toolbar buttons: `Ctrl+B` and `Ctrl+I` do not format text here.

16.4.2 Diagrams

Set a code block's language to `mermaid`, `drawio`, `zenuml` or `plantuml` and the preview draws it. Only PlantUML needs an internet connection.

16.5 Other Viewers and Editors

The extension decides the editor: HTML preview, image viewer, XML editor, read-only JSON view for PAIO's own documents, and a text editor for other tex and **MODULE** sections of the **EDITOR** view, not from model files in the **Explorer**.

Remote Development (SSH, vECU Container)

What This Chapter Covers

This chapter explains how to work with a project that is not on your local file system. Two targets are offered: an **SSH host**, and the project's **vECU contai**
vECU is the container for the AUTOSAR Classic Platform, and **PARA vECU** is the container for the AUTOSAR Adaptive Platform.

- One window holds one project. The remote folder you open becomes the window's project.
- A window with no project connects in place. A window that already has a project keeps it, and the connection opens in a window of its own.
- SSH needs a reachable server. The vECU needs Docker installed and running locally, and the PRTE vECU or PARA vECU image on your machine.
- For the code editor, the build commands and the terminal themselves, see 16. Code Editing, Build, Terminal and Documents.

17.1 Entry Points

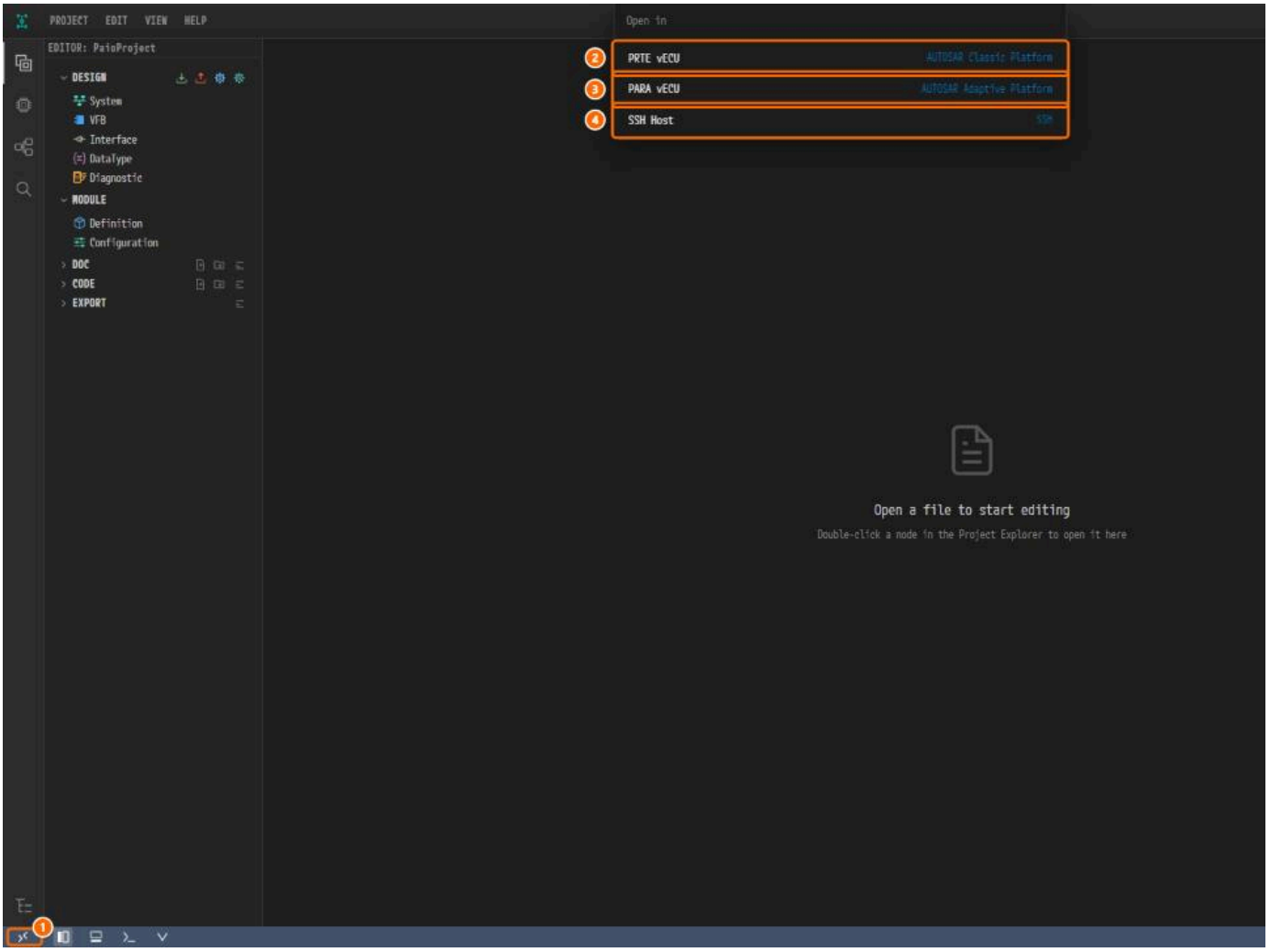
A remote connection starts from the `><` indicator at the left end of the status bar, the **Connect to...** button on the **Welcome** tab, the command palette (S
item on the left rail.

17.1.1 The Remote Indicator

The leftmost item in the status bar is the `><` indicator. It shows the connection state: `SSH: <user>@<host>` or `Container: <name> (<image tag>)` on
a menu headed `Open in` drops down from the top of the window.

17.1.2 The Menu While Not Connected

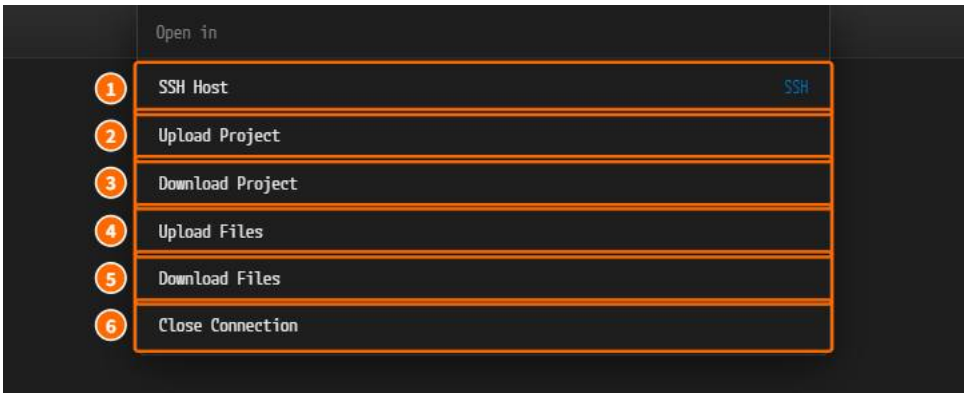
The menu has three rows: **PRTE vECU**, **PARA vECU** and **SSH Host**. The two vECU rows open the project's container (17.3.2). **SSH Host** opens the SSH quick co



*① The `><` indicator. ② **PRTE vECU**, ③ **PARA vECU**, ④ **SSH Host**.*

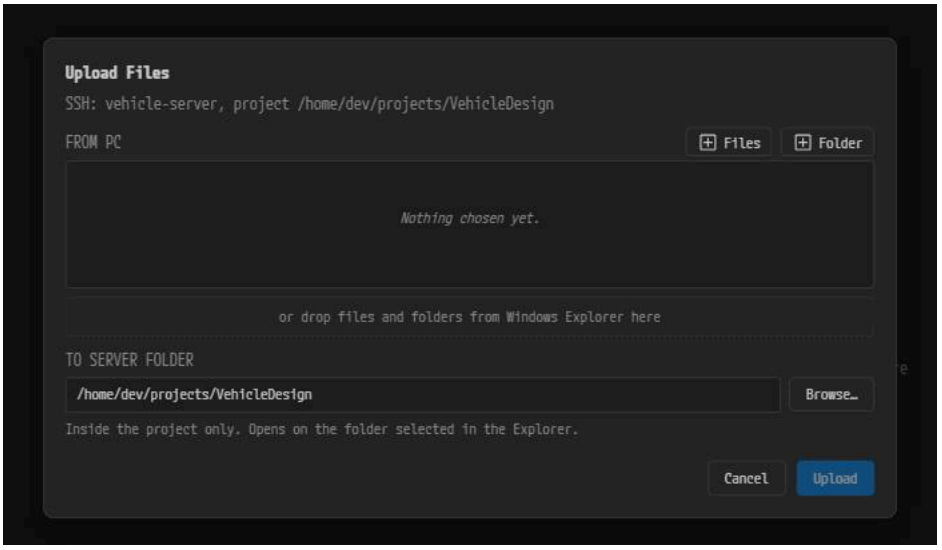
17.1.3 The Menu While Connected

In a window connected over SSH, the menu lists **SSH Host**, **Upload Project**, **Download Project**, **Upload Files**, **Download Files** and **Close Connection**. In a vECU **Connection** only.

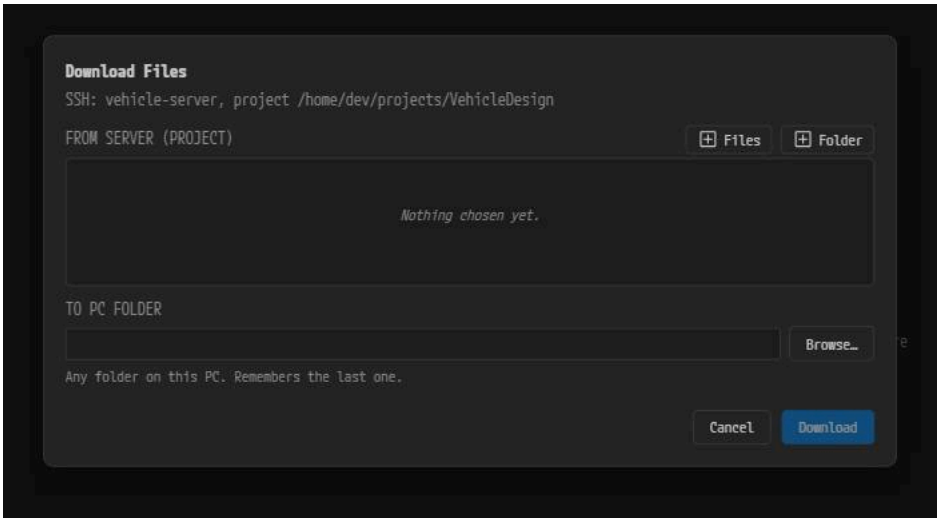


*① **SSH Host**, ② **Upload Project**, ③ **Download Project**, ④ **Upload Files**, ⑤ **Download Files**, ⑥ **Close Connection**.*

- **Upload Project** replaces the server folder of the window with a project from your PC, and **Download Project** copies the server project to a PC folder. E confirm.
- **Upload Files** and **Download Files** copy chosen files and folders. Choose them with **+ Files** and **+ Folder**, choose the target folder with **Browse…**, and cl drop files on the **Upload Files** dialog.
- If files of the same name exist, PAIO asks whether to replace or skip them.
- The server side is limited to the project folder.



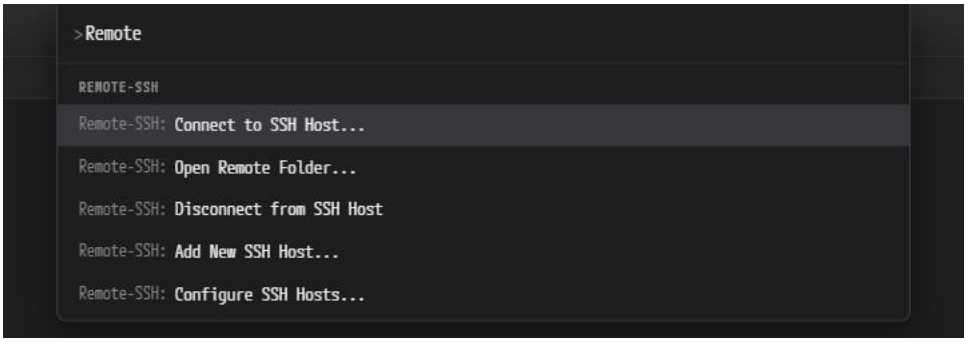
*The **Upload Files** dialog: **FROM PC** lists the files to copy, **TO SERVER FOLDER** is the target.*



*The **Download Files** dialog: **FROM SERVER (PROJECT)** lists the files to copy, **TO PC FOLDER** is the target.*

17.1.4 The Command Palette

The command palette carries SSH items only, under `Remote-SSH`. Open a vECU container from the **vECU** view or from the indicator menu.

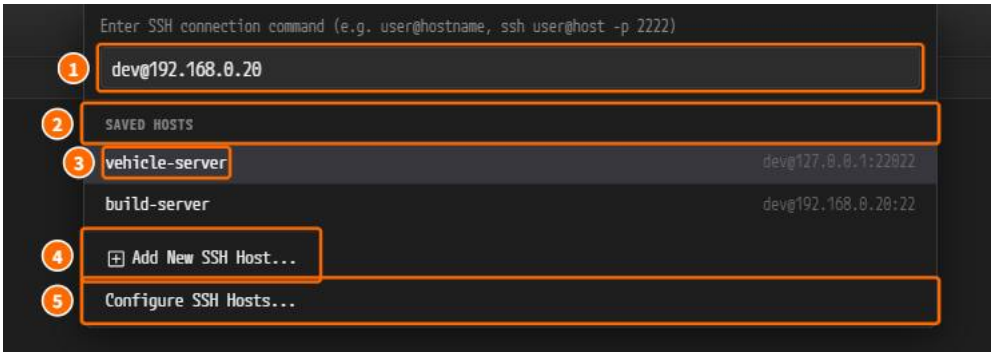


Press `Ctrl+Shift+P` and type `Remote`. The five items are grouped under `REMOTE-SSH`.

17.2 SSH

17.2.1 Quick Connect

1. Select **SSH Host** in the indicator menu. An input opens at the top of the window.
2. Type the connection details, such as `user@host` or `user@host:2222`, and press `Enter`. You can also pick a host under `SAVED HOSTS`.
3. In a window with no project, the window reloads and the remote folder browser opens (17.5). Pick a folder and click **OK**.



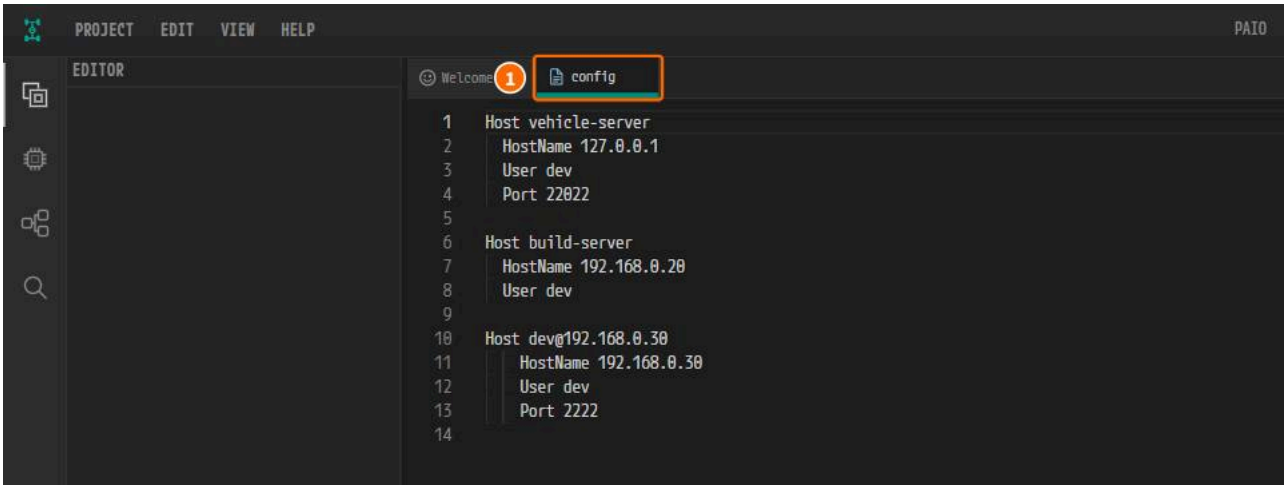
*① The input. ② `SAVED HOSTS`, ③ a saved host. ④ **Add New SSH Host...** (17.2.3), ⑤ **Configure SSH Hosts...** (17.2.2).*

Details you type are saved under a `user@host` label in your own `~/.ssh/config`. A host already saved for the same target is used as it is and never rewri

17.2.2 Configuring Hosts

The hosts are the ones in your `~/.ssh/config`. To edit or delete a host, edit that file.

1. Choose **Configure SSH Hosts...** in the quick connect input or the command palette.
2. Pick the file under `Select SSH configuration file to update`. It opens in an editor tab.
3. Edit the file and press `Ctrl+S`.



① The tab `config`, with one `Host` block per host.

The file does not open in a window connected to a remote target. Use a local window.

17.2.3 Adding a Host

1. Choose **Add New SSH Host...** in the quick connect input or the command palette.
2. Type the connection command, such as `ssh dev@192.168.0.30 -p 2222`, and press `Enter`. `user@host` and `user@host:port` work too, and the po
3. Pick the file under `Select SSH configuration file to update`. The toast `Host added` appears, and the host is saved under the label `user@host`



① The connection command.

A label that already exists is kept as it is, and the toast `Host already added` appears.

17.2.4 The Order Authentication Is Attempted

PAIO first connects with no credentials, so the keys in `~/.ssh` and ssh-agent are tried. If that fails, it asks for a password.

17.2.5 Host Key Verification

PAIO keeps host key fingerprints in its own store, not in `~/.ssh/known_hosts`. A new host is accepted and recorded. If the fingerprint changes later, PAIC `Changed` dialog.

17.2.6 Keep-Alive and Automatic Reconnection

When the connection drops, PAIO reconnects automatically up to three times, and the indicator reads `Connecting...` meanwhile.

17.2.7 Connection Failure Messages

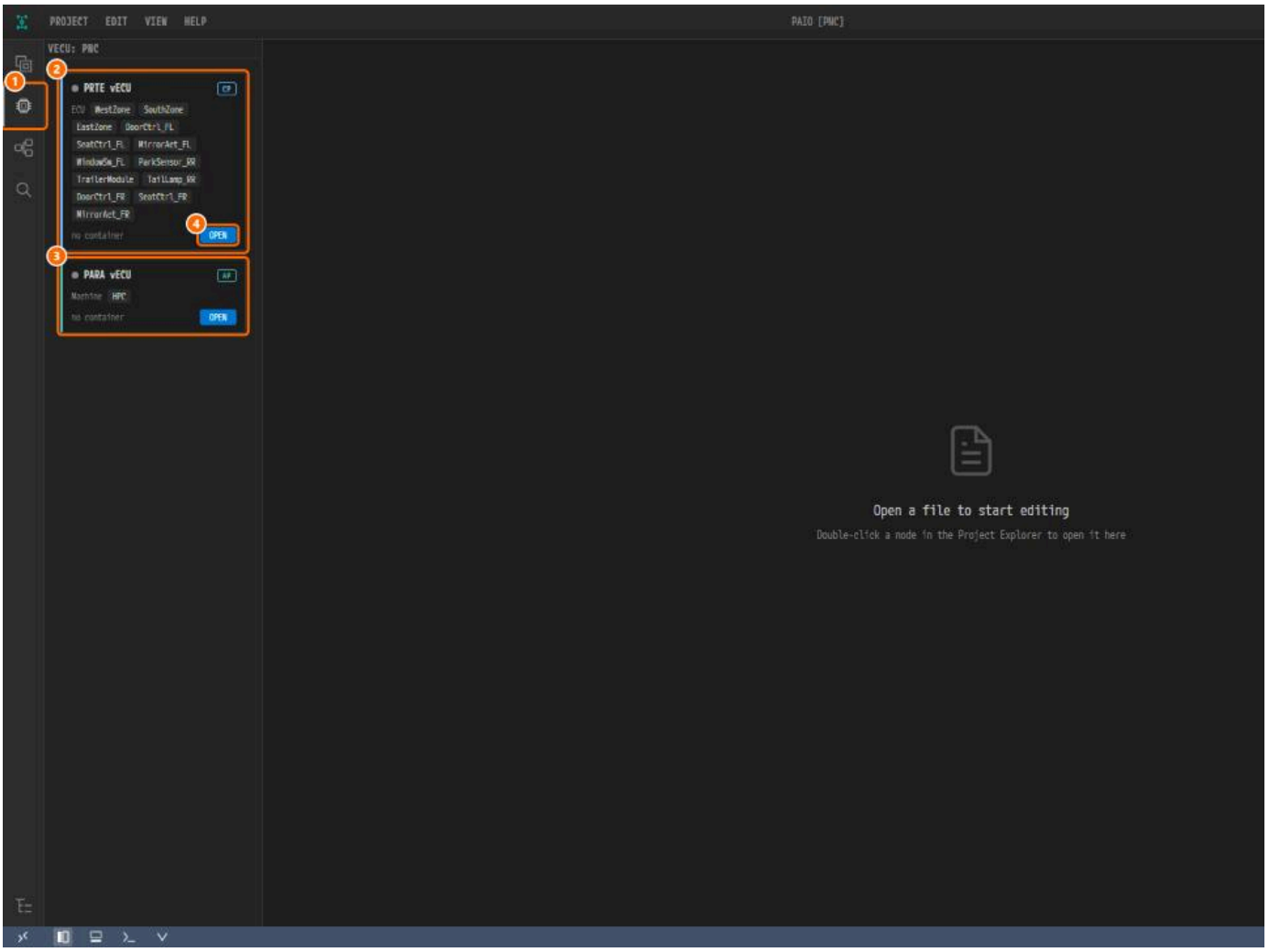
PAIO rewrites a connection failure into a message that says what to check.

17.3 vECU Containers (Docker)

A vECU is the project's execution environment: a Docker container built from the PRTE vECU or PARA vECU image, in which the project's ECUs are built and view.

- One **PRTE vECU** card (badge `CP`) with the project's Classic Platform ECUs.
- One **PARA vECU** card (badge `AP`) per HPC, or a single one when the project has no HPC yet.

Each card shows the state of its container and one action, **OPEN**.



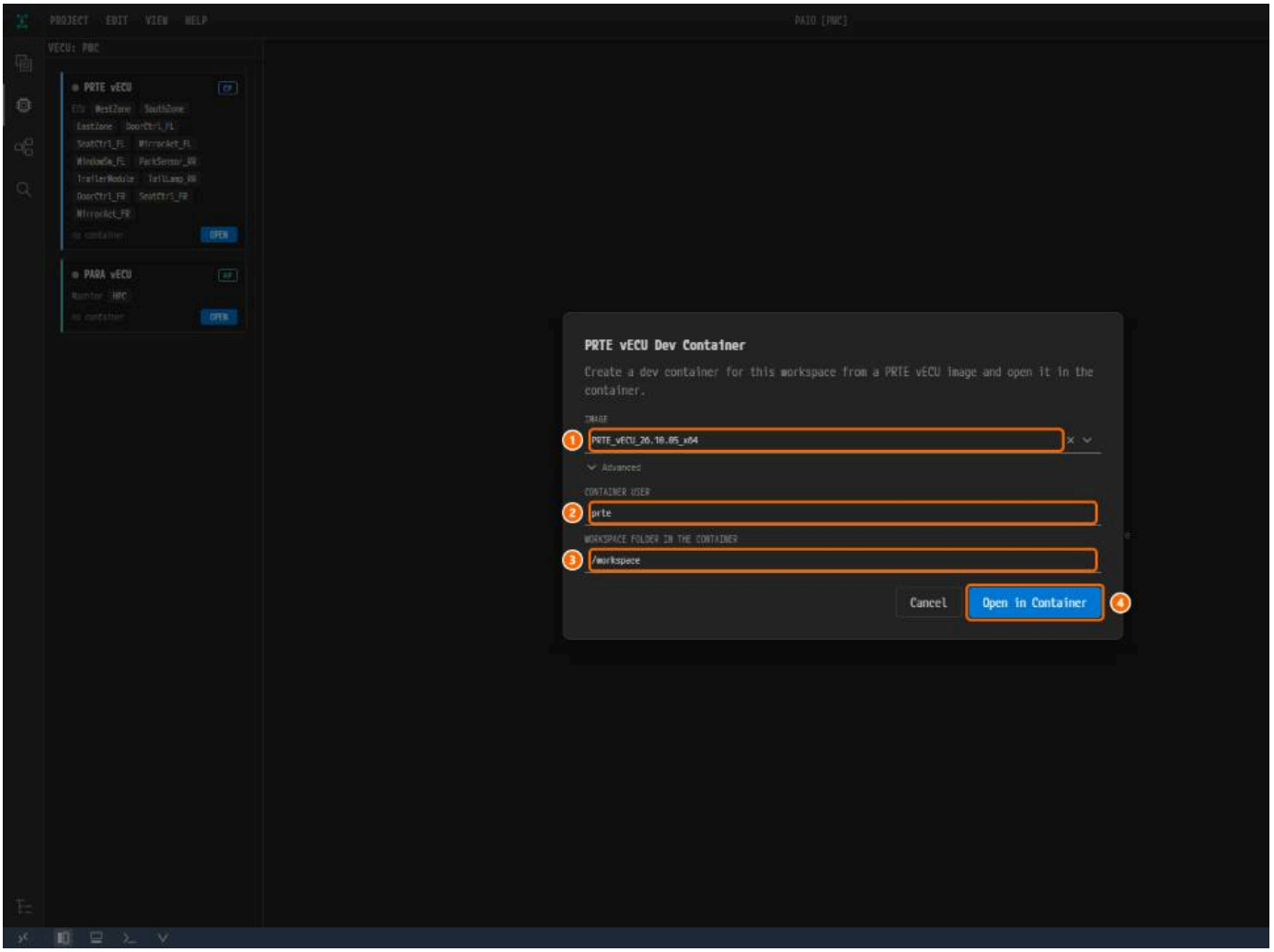
*① The **vECU** item on the rail. ② The **PRTE vECU** card, ③ the **PARA vECU** card, ④ **OPEN**.*

17.3.1 Prerequisites

Docker must be installed and running locally, and the PRTE vECU or PARA vECU image must be on your machine. To get an image, sign in at `autosar.io/su` `pull` commands the page shows.

17.3.2 Opening a vECU Container

1. Click **OPEN** on a card.
2. When the slot has no container yet, the `PRTE vECU Dev Container` dialog (or its PARA vECU counterpart) opens. Choose the image under **Image**.
3. Click **Open in Container**.
4. PAIO builds the container and opens the project in it, in a window of its own beside the host window.



*① **Image**, ② **Container user**, ③ **Workspace folder in the container** (under **Advanced**), ④ **Open in Container**.*

When the slot already has a container, **OPEN** enters it directly.

17.3.3 The Progress Toast

Progress is shown in a toast in the lower right corner, and you can keep working meanwhile. **Show log** shows the log and **Cancel** stops the work.

17.3.4 Entering an Existing Container

Once a slot has a container, **OPEN** on that card enters it again without the dialog. Attaching to an arbitrary running container is not offered.

17.3.5 vECU Configuration Files

The vECU dialog writes one configuration file per slot inside the project, under `.devcontainer/`.

17.3.6 Starting Over with a Fresh Container

There is no rebuild command. Remove the slot's container instead; the workspace volume and the host project are kept.

1. In the **vECU** view, click the three-dot button at the top right of the card (tooltip `More`).
2. Choose **REMOVE CONTAINER**, then click **Remove**.
3. The next **OPEN** builds a fresh container.

17.3.7 Leaving the Container Window

Close Connection copies the container's work back to the host project folder, asking about each conflicting file (17.10), and returns to the host project's copies the work back the same way.

17.3.8 When the Container Stops from Outside

If the container stops outside PAIO (for example `docker stop`), the connection state is cleared and a toast appears. To use the container again, open it

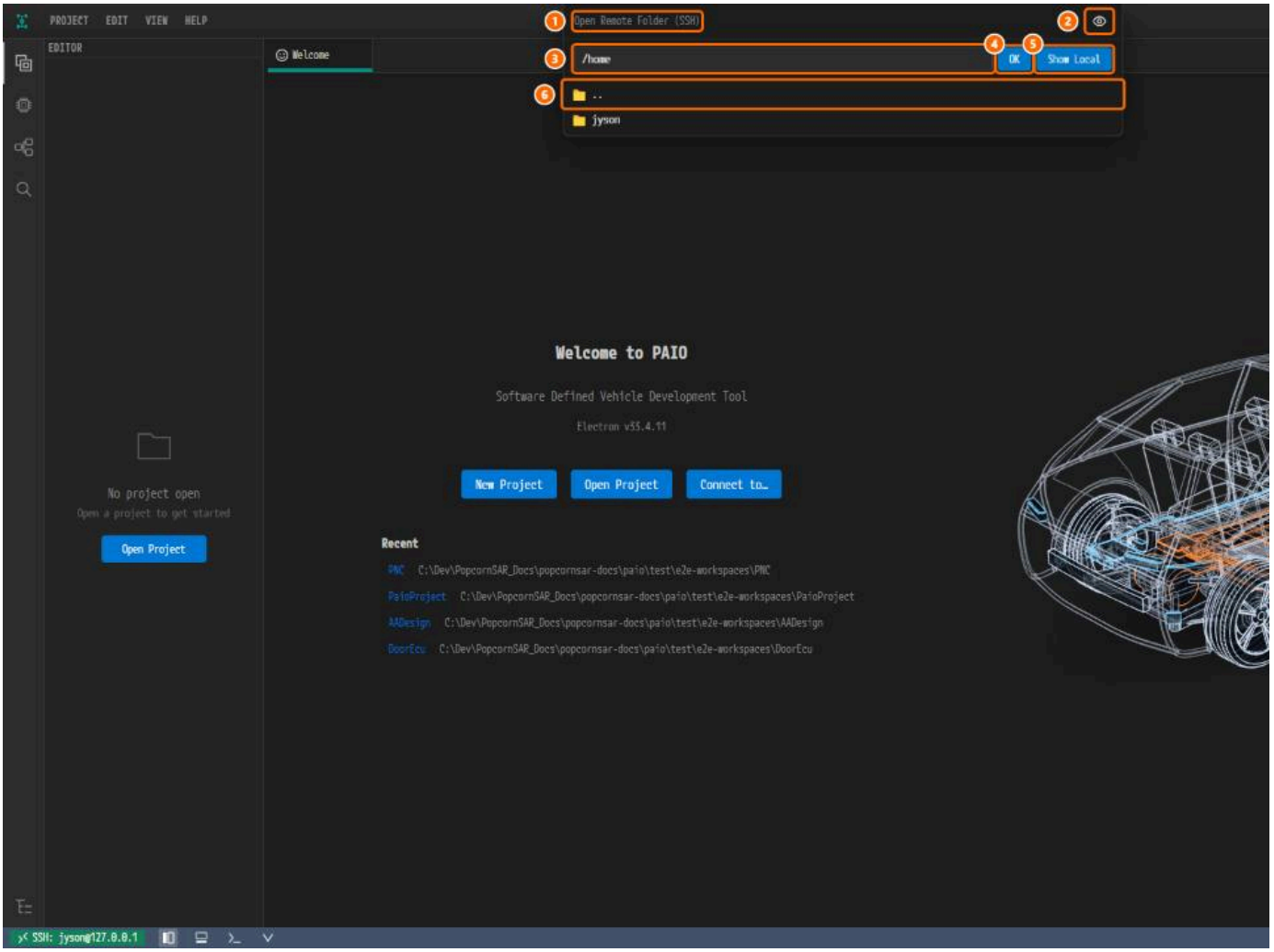
17.4 WSL (Windows)

WSL is not a connection target. A folder under `\\wsl$\<distro>` opens as an ordinary local window. Build and run the project's ECUs in the vECU (17.3).

17.5 The Remote Folder Browser (SSH)

The remote folder browser picks the folder to open on the SSH host. It opens by itself after connecting from a window with no project, and from **PROJEC** SSH.

1. Click a folder to enter it, or `..` to go up. You can also type a path: the list follows what you type. The arrow keys move through the list.
2. Click **OK** to open the folder shown in the path field as the window's project.
3. To make a folder, click the **+** row (tooltip `New Folder`) under the list, type the name and press `Enter`.



*① The header `Open Remote Folder (SSH)`, ② eye icon (dot files), ③ path field, ④ **OK**, ⑤ **Show Local**, ⑥ `..`.*

The list shows folders only.

17.6 Remote Language Servers and Build Tools

17.6.1 Automatic Deployment

When you open or build a C/C++/Rust file in an SSH or container window, PAIO uploads the Linux x64 binaries it ships with (clangd, rust-analyzer, CMake, g runs them there. You do not install anything yourself.

17.6.2 Supported Environments

The bundled binaries are built for glibc-based Linux x86_64. arm64, macOS and musl-based distributions such as Alpine are not supported.

17.7 Closing the Connection

Select **Close Connection** in the indicator menu.

- **SSH**: the connection is dropped and the window reloads as a window with no project.
- **vECU container**: the container's work is copied back to the host and you return to the host project's window.

In both cases the next launch does not reconnect automatically.

While an ARXML import runs, Close Connection first asks `An ARXML import is running. Closing the connection cancels it.` In an SSH window it al

17.8 Limits and Constraints

- Connecting from a window that has a project opens a new window. Save unsaved edits before connecting or disconnecting.
- Port forwarding is not provided.
- ECU build and run take place in the vECU. They are refused in a local Windows window and in an SSH window.
- A remote build is killed after 120 seconds.
- One window holds one remote mode at a time.

17.9 Building and Running ECUs in the vECU

The project's ECUs are built and run with four commands: **Build ECU**, **Run ECU**, **Stop ECU** and **ECU Doctor**. They are not the code editor's **Build** and **Build & F**

17.9.1 Where the Commands Are

1. In a vECU container window, right-click a folder directly under `code/` (for example `code/SwcDemo`) in the **CODE** section or the **Explorer**.
2. Choose **Build ECU**, then **Run ECU**.

The command palette also carries the four commands under `ECU`.

17.9.2 Where They Run

The commands run in the vECU container of the window. A local Windows window and an SSH window refuse them.

17.9.3 What Runs

PAIO decides the platform from the folder (PRTE ECU or PARA Machine) and runs CMake with the SDK of the vECU. When the output warns about a version SDK of the vECU.

17.9.4 Output and Result

Each **Build ECU** or **Run ECU** opens a read-only **Output** channel in the terminal panel. The last line is the result, for example `Build ECU SwcDemo: succeede` for example `Build ECU succeeded`. Compiler diagnostics that name a file also appear in **PROBLEMS** with the tag `BUILD`.

17.9.5 The ECU RUNS List and Stop

The **ECU RUNS** section above the terminal list shows the builds and runs in progress and the recent finished ones. **Stop** on a row, or **Stop ECU**, ends the ru

17.9.6 ECU Doctor

ECU Doctor checks the tools and the SDK the build needs, and changes nothing. When **Build ECU** fails, run **ECU Doctor** first.

17.9.7 ECU Settings

ECU Settings... sets the SDK root, the run environment and the run arguments of one ECU. Set them in the vECU window that runs the ECU, because the se

17.10 Keeping the Host and the Container in Sync

A vECU container works on its own copy of the project. PAIO copies files between that copy and the host project folder when you enter, when you save, a container window.

17.10.1 Progress and Result

A toast in the lower right corner shows the progress of a whole-project pass and then the result.

17.10.2 Which Side Owns What

- `design/`, `doc/`, `workflow/`, `export/` and `.devcontainer/` are owned by the host.
- Your code under `code/<node>/` is copied to the host as soon as you save it in the vECU window.
- Edit the code of an ECU in its vECU window. A save of the same file in the host window does not reach a running container until you enter it again.

17.10.3 Conflict Questions

When a file needs a decision, PAIO asks one dialog per file.

17.10.4 Two vECUs at Once

The PRTE vECU and a PARA vECU of the same project can run at the same time, each in its own window.

17.11 Code Intelligence in a vECU Container

In a vECU container window, C/C++ code intelligence works against the SDK inside the image, so the SDK headers that stay unresolved in a local window r project loads, and **PROBLEMS** covers the whole project.

17.12 Working in an SSH Window

In an SSH window, what you produce is stored on the SSH server:

- **Code Generation** (PRTE and PARA) runs on your PC over a copy of the server project and writes `export/<node>/` and `code/<node>/` back to the ser
- **AI chat** works on a copy of the server project and writes every file to the server first. Its conversations are stored in the server project (15.8).
- **AI Workflow** runs work on a copy of the server project, and each node's files are written to the server when the node finishes (21.13).
- **Build ECU** and **Run ECU** stay refused (17.9.2).

AI Workflow

What This Chapter Covers

This chapter explains the AI Workflow: a graph of nodes that takes a project from its input documents through design, AUTOSAR export and code genera and a verification report. You place nodes, connect them, set their parameters and run the graph. Each AI node runs a Claude session that writes files in t

- Readers: users who want PAIO to carry a project through several steps at once.
- Prerequisites: an open project, an active license, a Claude account for the AI nodes (21.11), and Docker with a PRTE vECU image for the nodes that bui

> **What a workflow writes is a draft.** PAIO checks what a node writes and sends defects back to the node that caused them, but it does not decide whet results in the editors and in the 14. Validation panel before you rely on them.

The AI chat is a separate feature: the provider you choose in 15. AI Assistant does not apply to the AI Workflow.

21.1 What the AI Workflow Does

A workflow is a directed graph. Each node does one step and hands its result to the nodes connected to its outputs. The usual chain is:

1. **Input Analysis** reads the project's documents, model files and code and builds a knowledge base under `workflow/knowledge/`.
2. The design nodes write the design under `design/`.
3. **AUTOSAR Export** writes the ECU Extract, and **AUTOSAR Code Generation** runs the PRTE (CP) or PARA (AP) generator.
4. **User Code Implementation** and **Test Code Implementation** write the user code and the tests.
5. **Compile/Build**, **Test Execution** and **Runtime Check** build and run the code in the vECU, and **Verification Report** writes the requirement traceability tab

A node that finds defects can send them back through its **Rejected** output to an earlier node, which runs again (21.8.4). Every project has exactly one wo `workflow/WORKFLOW.io.workflow`.

21.2 Prerequisites

You need an open project, an active license, a Claude account for every AI node (21.11), and Docker with a local PRTE vECU image for the build and run nod Containers (Docker).

21.3 Opening the Editor and Its Layout

1. Click the **Workflow** icon, the third icon of the rail. The editor tab `Workflow` opens.
2. In a new project the canvas is empty; the first `Ctrl+S` creates `workflow/WORKFLOW.io.workflow`.

The editor has the node palette on the left, the canvas in the center with the run log under it, and the inspector on the right. Double-clicking `WORKFLOW.` JSON viewer, not the editor.

21.4 Node Palette and Node Types

21.4.1 The Palette

The palette lists the node types by category. Type in `Search nodes` to filter, and drag an item onto the canvas to add a node. Clicking an item adds noth

21.4.2 The Node Card

A card shows the node name and its type label. `(OFF)` marks a disabled node, and the markers `START`, `END` and `PINNED` show the settings of 21.5.3. Durir is running, finished or failed.

21.4.3 Node Types

The nodes cover requirements, design, generation, implementation, build and verification, and control (**Approval**, **Condition Branch**, **Sticky Note**). Every inp

21.5 Building the Workflow

21.5.1 Connecting Nodes

Drag from an output port (right edge of a node) to an input port (left edge of another node). One output may feed several nodes. A loop is allowed only a **Rejected** port back to an earlier node, or back into the START node.

21.5.2 Selecting and Moving

Click a node to select it and show its settings; click the empty canvas to show the workflow settings. Drag the empty canvas to pan and use `Ctrl`+whee

21.5.3 Context Menus

Right-click a node to disable it, run from it, pin its output, or make it the START or END node of the run.

21.5.4 Sticky Notes

Right-click the empty canvas and choose **Add Sticky Note**. Select the note and type its text in the **NOTE** field of the inspector.

21.5.5 The Floating Toolbar

The toolbar at the bottom-left of the canvas has `Run Workflow`, `Cancel Run`, `Auto Layout`, `Templates` and `Show/Hide Run Log`.

21.5.6 Templates

A template is a whole workflow graph kept in a library on this computer, shared by every project. Use **Save as Template** to save the current graph, and `Loa` with a template.

21.5.7 Saving and Undo

`Ctrl+S` saves `workflow/WORKFLOW.io.workflow`. `Ctrl+Z` undoes and `Ctrl+Y` redoes.

21.6 The Inspector

21.6.1 Workflow Settings

Click the empty canvas to show the **Workflow** settings.

- **RUN MODE** > **NEXT RUN** decides what the next run does with the existing results: `UPDATE` (default), `REBUILD` or `RESET`. It applies to one run only.
- **AI DEFAULTS** sets the Claude model and effort of every AI node.
- **PER-ECU NODES** > **PARALLEL ECUS** sets how many ECUs some nodes handle at the same time.

> `RESET` first deletes `design/` (every design document, hand-made ones included) and other folders. The deletion cannot be undone.

21.6.2 Node Settings

Select a node to show its settings, such as **ENABLED**, **NAME**, **AI SETTINGS**, **PARAMETERS** (21.7) and **LAST RUN** (21.9.1).

21.6.3 Settings Shared by Many Nodes

Several fields appear on many nodes: **ADDITIONAL INSTRUCTIONS**, **INSUFFICIENT INFORMATION**, **FEASIBILITY CHECK**, **CONNECT TO VECU** and **TARGET ECU / TARG**

21.6.4 The PRTE vECU Image

One PRTE vECU image applies to the whole workflow. The list is read from the local `docker images`, and the default is the newest local image at run time.

21.7 Node Settings Reference

Each node has its own fields under **PARAMETERS**. The subsections below say what each node's settings decide.

21.7.1 Input Analysis

Chooses the sources to analyze (documents, model files, code) and whether to run the Feasibility Check, build the knowledge base and write the operati

21.7.2 System Design

Sets the system name, the communication scope, and what happens to each ECU / HPC node.

21.7.3 Diagnostic Design

TARGET ECU / HPC names the nodes to design; empty means every ECU and HPC.

21.7.4 SWC Design

Sets the target ECUs, what happens to existing SW-Cs, the design mode, and whether existing interfaces may change.

21.7.5 BSW Config

Sets the target ECUs, the Base Set (`PRTE` or `Other Vendor`) and how the modules are chosen.

21.7.6 SWC + BSW Integration and Functional Cluster Config

Both have a target list (**TARGET ECU** / **TARGET HPC**).

21.7.7 AA Design

TARGET COMPOSITION names the composition. The node does not run in this version (21.12).

21.7.8 Design Review

REVIEWER is `AI` (default) or `HUMAN`; `HUMAN` pauses the run until a person approves or rejects (21.8.3).

21.7.9 AUTOSAR Code Generation

Sets the target ECUs and how generator findings are fixed.

21.7.10 User Code Implementation

Sets the platform, the targets, the code scope, the MISRA C:2025 check and how existing code is handled.

21.7.11 Test Code Implementation

TEST LEVEL is `Unit` (default) or `Integration`.

21.7.12 Compile/Build

Sets warnings as errors, the AI auto fix and the parallel jobs.

21.7.13 Runtime Check

Sets the run times, the bus, the fix mode and the expected behaviour to check.

21.7.14 Test Execution and Verification Report

Test Execution has a timeout and an AI failure analysis; **Verification Report** has an AI summary.

21.7.15 Approval and Condition Branch

Approval has **APPROVAL MESSAGE**, the text shown while the run waits. **Condition Branch** has **CONDITION** but does not run in this version (21.12).

21.8 Running a Workflow

21.8.1 Starting a Run

1. Click `Run Workflow` in the toolbar. The document runs as it is shown, unsaved edits included.
2. To start part way, right-click a node and choose **Run from Here**. Upstream values come from the newest recorded run.

Only one run per project at a time.

21.8.2 Execution Order

Nodes whose inputs are ready run in parallel. A node with several inputs waits until all of them arrive. A disabled node passes its inputs through.

21.8.3 Nodes That Wait for You

A **Design Review** with **REVIEWER** `HUMAN` and an **Approval** node pause the run.

1. Select the node. **LAST RUN** reads `Waiting for response…`.

2. Type an optional comment in `Comment (optional)`.
3. Click the check (`Approve`) or the cross (`Reject`).

A node that needs information it cannot get from the project asks `Waiting for user input: <question>`; do what it asks, then press Approve.

21.8.4 Rejected Loops

When a node reports defects on its **Rejected** port and that port is wired back to an earlier node, the run runs that node again with the findings, then the stops there.

21.8.5 Following a Run

The run log under the canvas shows one line per step. A run continues in the background when you close the editor tab, and a toast reports when it ends

21.8.6 Cancelling and Closing

`Cancel Run` in the toolbar stops the run and rolls back every file it wrote. Closing the tab or the window during a run behaves differently.

21.8.7 Resuming an Interrupted Run

If PAIO quit during a run, reopening the editor asks **Interrupted Workflow Run. Resume** reuses the completed nodes and runs the rest; **Discard** drops the r

21.9 Run Results and Where They Go

21.9.1 Last Run

Select a node to see **LAST RUN**: its status, then rows such as the designed elements, the findings and the changed files. It shows the runs of the current

21.9.2 Comparing Before and After

In **LAST RUN**, `Compare before/after of this run` opens a read-only side-by-side comparison of a file before the run and now.

21.9.3 Where the Results Are Written

The nodes write under `workflow/`, `doc/`, `design/`, `export/` and `code/`. The reports go to `doc/report/`.

21.9.4 The Run Folder

Every run is recorded in `<project>/.paio/workflow-runs/<runId>/`, with the backup used for rollback and the diff. The `.paio` folder is hidden in the E

21.10 The Knowledge View

Input Analysis with **BUILD KNOWLEDGE (LLM WIKI)** builds pages under `workflow/knowledge/`, and every later node works from them. Click **KNOWLEDGE** ab **WORKFLOW** to return. Click a page to preview it; on a requirement page you can set how the requirement is verified.

21.11 Connecting the Claude Account (AI Account)

Workflow AI nodes always use Claude through the Claude Code runtime bundled with PAIO. By default they use this computer's Claude Code login.

1. Open the user avatar on the right of the top bar > **AI Account** (3.3).
2. Check the status line. `Not connected` means AI nodes cannot run.
3. Click **Connect** to sign in through the browser, or **Paste Token** to paste a token.

21.12 Limits and Nodes That Are Not Implemented

- **AA Design** and **Condition Branch** can be placed but do not run in this version. Disable them or leave them out.
- **Runtime Check** > **BUS** = `REAL BUS` and **User Code Implementation** > **PLATFORM** = `AP` are not available yet.
- Only one run per project at a time.

21.13 Running a Workflow in an SSH Window

In a window connected over SSH (17. Remote Development), a run works on a copy of the server project on your PC, and each node's files are written to th

- **User Code Implementation**, **Test Code Implementation**, **Compile/Build**, **Test Execution** and **Runtime Check** do not run there: they build and run in a vEC
- **Close Connection** during a run asks first; closing cancels the run without rolling back its files.
- The run records stay on your PC.

You can also build or change the workflow graph from AI chat (15. AI Assistant).

Reference

Page	What it covers
Keyboard Shortcuts	Shortcut keys, grouped by where they work
Glossary	Terms used in PAIO and in this manual
Troubleshooting and FAQ	Common problems, their causes, and what to do

Keyboard Shortcuts

What This Chapter Covers

This chapter lists the keyboard and mouse actions that work in PAIO, by screen area, including the keys that behave differently by situation. Area names
and Navigation.

- On macOS, read `Ctrl` as `Cmd`.
- Keys not in these tables are not registered. Text areas follow the browser defaults.
- Some keys are accelerators of the hidden native menu (`Ctrl+O`, `Ctrl+Shift+N`, `Ctrl+S`, `F11`). The rest, including the zoom keys, are handled by the
  See 05.1.1.

18.1 Global

These keys work on any screen in the application.

Shortcut	Action	Notes
`Ctrl+O`	Open Project...	An accelerator registered by the native menu. The folder you pick becomes the window's project. It behaves diff Markdown editor (see 18.11)
`Ctrl+Shift+N`	New Window	An accelerator registered by the native menu. Opens a second PAIO window with no project open
`Ctrl+S`	Save the active editor	Each editable editor handles this itself. A `Ctrl+S` that no editor handles saves through the same path as the me
`Ctrl+Shift+S`	Save the active editor	Works the same as `Ctrl+S`. There is no Save As
`Ctrl+Z`	Undo	Each editable editor handles this itself and reverts the single edit history that the project has. On a non-editable
`Ctrl+Y`	Redo	The System and VFB canvases, the configuration dialogs, and the DataType editor also accept `Ctrl+Shift+Z` fo
`Ctrl+X` / `Ctrl+C` / `Ctrl+V`	Cut / Copy / Paste	These target text input. The **EDIT** menu items run the browser clipboard commands
`Ctrl+A`	Select all	This targets text input. The Explorer, the System editor canvas, and the VFB canvas have their own behavior
`Ctrl+B`	Toggle the sidebar	Shows or hides the sidebar view chosen in the left icon bar (the **EDITOR** view, the Explorer, and so on). The code e
`Ctrl+``	Toggle the terminal panel	If no terminal exists, one is created
`Ctrl+J`	**Toggle the terminal panel**	The menu label reads `Toggle Panel`, but the actual target is the terminal
`Ctrl+Shift+P`	Open or close the command palette	
`Ctrl+Shift+F`	Open the Search view	Works even while the terminal has focus. Does nothing while a modal dialog covers the window
`F4` / `Shift+F4`	Go to the next / previous search result	Works only while the Search view holds results and no editor or dialog has taken the key
`Ctrl+T`	Open the Meta Class search	Finds where an AUTOSAR meta class is designed (05.10). Not while a modal dialog is open. In the Markdown editor instead (see 18.7, 18.11)
`Ctrl+=` / `Ctrl+-` / `Ctrl+0`	Zoom in / Zoom out / Reset zoom	`Ctrl++` and the numeric keypad `+`, `-`, and `0` with `Ctrl` work as well. The keys, the menu items, and the comma the level is kept for the next launch on the same machine
`F11`	Toggle full screen	An accelerator registered by the native menu
`Alt+F4`	Close the window	The OS handles this window close; it is not a key the application registers. If there are unsaved changes, the save

New Project... has no shortcut. Start it from the **New Project** button on the **Welcome** tab or from the **PROJECT** menu.

18.1.1 Key Whose Menu Label Differs From Its Behavior

The following key is written in the menu, but pressing it does something other than the menu item.

Shortcut	Menu label	Actual behavior
`Ctrl+J`	Toggle Panel	Clicking the menu item toggles the Utility panel, but pressing the key toggles the terminal panel. There is no shortcut that opens the Utility panel

18.2 Graphical Editors (System · VFB)

The System editor (tab `<project>:System`) and the VFB editor receive the same keys, but **the range in which they actually work differs.** In the table bel has no effect at all. In that case the browser's default behavior is blocked as well.

18.2.1 Keyboard

Shortcut	System	
`Ctrl+S` / `Ctrl+Shift+S`	Save	Save
`Ctrl+Z`	Undo	Undo
`Ctrl+Y` / `Ctrl+Shift+Z`	Redo	Redo
`Esc`	Clear the selection	Clear the selection
`Delete` / `Backspace`	Delete the selected nodes after a confirmation. A connection is not deleted with this key; remove it from its own context menu	Delete the selection, the same as Del deleting its connector
`Ctrl+A`	Select all	Select all nodes at the current drill-d
Arrow keys	Move the selected items by 1 px	Does nothing
`Shift` + arrow keys	Move the selected items by 20 px (one grid cell)	Does nothing
`Ctrl+O`	Open a file selection dialog and load the chosen IVC model file into the current System editor	Does nothing
`Ctrl+C` / `Ctrl+V` / `Ctrl+X`	Does nothing (copying and pasting nodes is not provided)	Same

The canvas handles `Ctrl+S`, `Ctrl+Shift+S`, `Ctrl+O`, `Ctrl+Z`, and `Ctrl+Y` even when a field in the properties panel has focus. The undo and redo keys r were typing is committed as one operation and then reverted. The other keys (`Delete`, `Ctrl+A`, the arrow keys, and so on) are ignored while a field has f

Even when you move several nodes at once with the arrow keys, undo reverts them in **one** step.

If you press `Esc` while dragging a waypoint, the drag is canceled and the path returns to its original shape instead of the selection being cleared. No ope

> `Ctrl+O` **in the System editor does not open a project.** When the System editor canvas has focus, a dialog for selecting an IVC model file opens, and th **System editor.**

18.2.2 Mouse

Mouse operations on the canvas are as follows.

Operation	Action
Left-drag on empty space	Pan the view
`Ctrl` + left-drag on empty space	Rectangle selection. The nodes the rectangle fully encloses become the selection, replacing the previous one
Middle-button drag	Pan the view
`Space` + left-drag	Pan the view
Wheel (no modifier)	Pan vertically (behaves like scrolling a document)
`Shift` + wheel	Pan horizontally
`Ctrl` + wheel	Zoom (0.1x to 4.0x, centered on the cursor)
`Shift` + drag	Connection mode
`Ctrl` + click	Multi-select (toggle)
Double-click	Per-node configuration dialog / composition drill-down
Right-click	Context menu

18.3 Configuration Dialogs

These keys apply to the node, cluster, and switch configuration dialogs and the **Partial Network (PNC) Configuration** dialog of the System editor, and to t
Configuration, **SWC Internal Behavior**, **AA Internal Behavior**, **Intrusion Detection System**, and **Software Cluster** dialogs.

Shortcut	Action
`Ctrl+S` / `Ctrl+Shift+S`	Save
`Ctrl+Z`	Undo inside the dialog
`Ctrl+Y` / `Ctrl+Shift+Z`	Redo inside the dialog
`Esc`	Closes the innermost open UI, or the dialog itself if nothing else is open

`Esc` first closes a transient UI such as a dropdown, a popup menu or an inline editor; press it again to close the dialog. With unsaved changes, the `Do yo`
appears first, and you choose among **Save**, **Don't Save**, and **Cancel**.

> **The scope of `Ctrl+Z` differs from dialog to dialog.**
>
> - The VFB **Machine Configuration**, **ECU Configuration**, **SWC Internal Behavior**, **AA Internal Behavior**, **Intrusion Detection System**, and **Software Cluster**
> **Network (PNC) Configuration** dialog **fall through to the project edit history** once the internal history is empty. `Ctrl+Z` works even right after a sav
> - The System editor's cluster configuration dialog falls through only on its cluster tab, whose edits go straight into the project edit history.
> - The System editor's node and switch configuration dialogs do not fall through. When the internal history is empty, `Ctrl+Z` does nothing. A key th
> either, so to revert an earlier edit, close the dialog first and then undo.

The internal history of a configuration dialog keeps up to 50 steps, in both the System editor and VFB.

18.4 Table-Based Model Editors

The six Signal, Interface, DataType, Diagnostic, Module Definition, and Module Configuration editors are built from lists and a properties panel rather than
open from **DESIGN** in the **EDITOR** view, Module Definition and Module Configuration from **MODULE** > **Definition** and **Configuration**, and Signal only from the S
canvas. All six share the save, undo, and redo keys.

Shortcut	Action
`Ctrl+S` / `Ctrl+Shift+S`	Save
`Ctrl+Z`	Undo (project edit history)
`Ctrl+Y`	Redo. The DataType editor also accepts `Ctrl+Shift+Z`

The undo and redo keys commit the value you were typing as one operation and then revert it. As a result the whole field edit is reverted, and the browse interfere.

Each editor adds the following keys.

Editor	Shortcut	Action
Interface	`Delete` / `Backspace`	Deletes the selected interface after a confirmation
Module Definition	`Delete` / `Backspace`	Deletes the selected definition node after a confirmation (module nodes excluded)
Module Configuration	`Delete` / `Backspace`	Deletes every selected container instance after a confirmation
Module Configuration	`Ctrl+D`	Duplicates the selected container instance
Module Configuration	`F2`	Moves focus to the first field (Name) in the properties panel and selects its value

The `Delete` keys are ignored while a field, a text area, or a combo box has focus.

In a list you can move between rows with `ArrowUp` and `ArrowDown`. The type list and unit list in DataType, the interface list in Interface, and the frame lis

The reach of these keys splits into two cases. DataType receives keys from the whole window while that editor is visible, and the other five require focus focus is outside the editor area still saves the active editor through the menu Save path.

18.5 Code Editor (C / C++ / Rust)

The keys used in the code editor are as follows.

Shortcut	Action
`Ctrl+S`	Save
`Ctrl+B`	Build
`Ctrl+F5`	Build & Run
`Shift+Alt+F`	Format Document
`Alt+O`	Switch Header/Source (`.c` ↔ `.h`, `.cpp` ↔ `.hpp`, and so on)
`F12`	Go to definition, the same as `Ctrl` + click
`Ctrl` + click	Go to definition. Without a language server, only a quoted `#include "..."` opens, and only when the header exists at that path relative to the current file's f

Other editing shortcuts follow the Monaco defaults. For the build flow in detail, see 16. Code Editing, Build, Terminal and Documents.

18.6 XML / ARXML Editor

The keys used in the XML / ARXML editor are as follows.

Shortcut	Action
`Ctrl+S`	Save
`Ctrl+Z` / `Ctrl+Y`	Undo / Redo (Monaco's own history)
`Shift+Alt+F`	Format XML

`Ctrl+S` and the undo and redo keys work **only in edit mode**. A file opens in view mode by default, so click the eye icon at the left of the toolbar first to s
redo, and format buttons on the toolbar are disabled as well.

PAIO's own documents (`*.io.model`, `*.io.workflow`, `*.io.prj`, `*.vfb.layout`), such as the design files you open from the Explorer, are shown read-o
mode, so these keys do nothing there. Edit a design in its design editor instead.

For a `.dbc` file the format button is hidden, but the `Shift+Alt+F` key still works: in edit mode it runs the XML formatting function.

In edit mode, typing `<` suggests the child elements that the AUTOSAR schema allows, through autocompletion.

18.7 Markdown Editor

The keys used in the Markdown editor are as follows.

Shortcut	Action
`Ctrl+S`	Saves the document. If it has never been saved, the save location dialog appears
`Ctrl+T`	Opens a new document (if you are already editing one, it opens in a separate tab)
`Ctrl+O`	Opens a Markdown file selection dialog and opens the file you choose in a separate tab

These three keys work only while the Markdown editor is visible. `Ctrl+T` opens a new document only while the focus is in the Markdown editor or on the
Class search (18.1).

The tooltips on the bold and italic buttons read `Bold (Ctrl+B)` and `Italic (Ctrl+I)`, but **those keys do not work in this editor.** Apply formatting with
`B`, `I`, `S` (strikethrough), `H` (heading), `<>` (inline code), and `-` (bullet list).

18.8 Explorer

The Explorer opens from the **Explorer** icon at the bottom of the icon bar on the far left. It lists the project folder's own contents, with no project row abo
follows.

Operation	Action
Click	Select
`Ctrl` + click	Multi-select (toggle)
`Shift` + click	Range select
Double-click	Opens the file in the editor that fits its file type. A design file (`*.io.model`) opens read-only (see 18.6); open the design editors from the **EDITOR** vi
Right-click	Context menu
Click the arrow (chevron)	Expand / collapse
Drag	Move. **Only file and folder nodes** can be dragged

The keys used while a node has focus are as follows.

Shortcut	Action
`Enter` / `Space`	Expands or collapses the node if it has children, otherwise opens it in an editor
`ArrowRight`	Expands a collapsed node
`ArrowLeft`	Collapses an expanded node
`Delete`	Deletes the selected **file or folder** (a confirmation dialog appears)
`Ctrl+C`	Copy
`Ctrl+X`	Cut
`Ctrl+V`	Paste
`Ctrl+A`	Selects the focused folder and its expanded contents (for a file, the expanded contents of the folder that holds it). Works only inside the `doc`, `code`, and `e` change

`Delete` does not work on the fixed folders (`code`, `design`, `doc`, `export`, `workflow`), so that a fixed folder is not removed by mistake.

18.9 Tabs

The operations available on editor tabs are as follows.

Operation	Action
Click / `Enter` / `Space`	Activate the tab
Middle-button click	Close the tab
Right-click	Close / Close Others / Close Saved / Close All
Drag onto the tab bar	Reorder (within the same group) or move between groups
Drag onto the right half of the body	Split left and right, then move to the new group
Drag onto the left half of the body	Move to that group without splitting

While the tab context menu is open, move between items with `ArrowUp`, `ArrowDown`, `Home`, and `End`, run an item with `Enter` or `Space`, and close the me

Splitting is supported left and right only. There is no operation that splits top and bottom. **Close Others**, **Close Saved**, and **Close All** in the context menu cl **Close**, so a tab with unsaved changes asks whether to save first (see 05.3.3).

18.10 Command Palette

The keys used in the command palette are as follows.

Shortcut	Action
`Ctrl+Shift+P`	Open / close
`ArrowDown` / `ArrowUp`	Move between items
`Tab` / `Shift+Tab`	Move between items
`Enter`	Run the selected command
`Esc`	Close

The commands registered in the palette are listed in 05.6 Command Palette.

18.11 Conflicts and Pitfalls to Know

This section collects the keys that behave differently depending on the screen area.

Key	Behavior by situation
`Ctrl+B`	In the code editor it is registered as **Build**. Elsewhere it toggles the sidebar
`Ctrl+O`	Default → Open Project... / System editor canvas → **load an IVC model file** / VFB canvas → **nothing** / Markdown editor → open a Markdown file
`Ctrl+J`	The menu item toggles the Utility panel, but the key actually toggles the **terminal panel**
`Ctrl+T`	Default → open the Meta Class search / focus in the Markdown editor or on the Doc start page → a new document / a modal dialog open → nothing
`Ctrl+A`	Text input → select all / Explorer → the expanded contents of a folder inside `doc`, `code`, or `export` / System editor canvas → select all / VFB canvas → select all at
`Delete`	Explorer → delete files and folders / System editor canvas → delete the selected nodes (connections excluded) / VFB canvas → delete the selection / Interface, Mo selected items. Every case asks for confirmation first
`Esc`	An open dropdown or menu → close it / nothing open → close the dialog / on a canvas → clear the selection (cancel the drag if a waypoint is being dragged)
`Ctrl+Z`	Editable editor → project edit history / configuration dialog → the dialog's internal history first (see 18.3) / code and XML editors → Monaco's own history / image ar

Model edits collect into one project history, but **the physical `Ctrl+Z` is handled by each editor**. On a tab with no edit history, such as the image viewer, t the command palette instead: it reverts the most recent model edit from any tab and moves the view to that edit's editor. The **EDIT** menu shows the nam tab of the Utility panel shows the full list (see 05.4.1).

18.12 Search View

These keys work in the Search view and in a Search Editor tab (05.8, 05.9). `Ctrl+Shift+F` and `F4` / `Shift+F4` are listed in 18.1.

In the search box:

Shortcut	Action	Notes
`Enter`	Search now	Also records the query in the search history
`Escape`	Clear the search box	Search view only, and only when the box holds text
`Ctrl+Down`	Move into the results	Search view only
`ArrowUp` / `ArrowDown`	Walk through earlier queries of this project	While the suggestion list is open, the arrows move through the list and `Enter` searches the picked qu

In the results of the Search view:

Shortcut	Action	Notes
`ArrowUp` / `ArrowDown`	Move to the previous / next row	`ArrowUp` on the first row returns to the search box
`Home` / `End`	Move to the first / last row	
`ArrowRight` / `ArrowLeft`	Expand / collapse a group	`ArrowRight` on an expanded group moves to its first row; `ArrowLeft` on a row inside a group moves to th
`Enter` / `Space`	Open the result, the same as a click	A group row that opens nothing expands or collapses. Result rows of a Search Editor tab take these keys

18.13 AI Workflow

Keys of the AI Workflow (21).

Shortcut	Action	Notes
`Ctrl+S`	Save the workflow document	Creates `workflow/WORKFLOW.io.workflow` on the first save
`Ctrl+Z`	Undo	Not while the focus is in a text field (the field undoes the typing)
`Ctrl+Y` / `Ctrl+Shift+Z`	Redo	As `Ctrl+Z`
`Delete` / `Backspace`	Delete the selected nodes, notes or connection	The canvas must have focus
`Shift`+click	Add a node to or remove it from the selection	
`Ctrl`+wheel	Zoom at the pointer	
Wheel / `Shift`+wheel	Pan vertically / horizontally	Dragging the empty canvas also pans

18.14 Meta Class Search

Keys of the Meta Class search box (05.10). `Ctrl+T` itself is listed in 18.1.

Shortcut	Action	Notes
`ArrowDown` / `ArrowUp`	Move the selection	The pointer only moves the highlight
`Enter`	Open the selected line	The same as a click
`Escape`	Close the box	A click outside the box closes it too
`Ctrl+T`	Put the caret back in the search box	While the box is open

Glossary

This chapter collects the AUTOSAR and PAIO-specific terms used in the PAIO documentation, with **the PAIO screen and file each term corresponds to**.

How to Read This Glossary

- Each table has three columns: **term, meaning, and where you see it in PAIO**. The link in the third column leads to the document that covers the term.
- Bold in the first column is the headword; bold elsewhere is a UI label, never translated. `Backticks` mark file paths, extensions, key names, enumerati
- AUTOSAR element names use the upper-case spelling that appears in ARXML (`ECU-INSTANCE`).
- Items that appear in the UI but do not actually work are marked as such.
- On a first read, start with 19.1. Look up 19.2 through 19.7 as you need them.

19.1 PAIO-Specific Terms

Concepts that PAIO defines on its own. They do not exist in the AUTOSAR standard.

Term	Meaning	
IO Model	PAIO's primary data model. It is format-neutral, human-readable JSON. ARXML and DBC are conversion layers on top of it, not the original	Every model file under `design/`
IO-First	PAIO's design principle that the IO Model, not AUTOSAR, is the source of truth. As a result, information that AUTOSAR cannot express still survives in the model	01. Product Overview
Workspace	The folder that a PAIO window has open. In PAIO the workspace and the project are the same thing: one window holds one workspace, and that workspace is one project. The window takes a single-instance lock on the folder it opens	**PROJECT** > **Open Project...** (`Ctr` **Open Project** buttons of the **We** Workspaces and Projects
Project	One set of design deliverables, held in one folder. The folder name is the project name. It consists of a project metadata file (`<project>.io.prj`) and six fixed folders. To work on another project, open it in another window (**PROJECT** > **New Window**)	The **New Project** dialog, the win `<project>` · 04. Workspaces an
Fixed Folder	One of the six folders that are always created with a project: `doc`, `design`, `code`, `workflow`, `export`, `import`. All but `import` can be neither renamed nor deleted; `import` has the full folder menu, and PAIO recreates it if it is deleted	The top-level folders in the **Exp** and `export` also appear as the 04. Workspaces and Projects
EDITOR view (formerly the design tree)	The sidebar view that opens the design editors. Its **DESIGN** section lists **System**, **VFB**, **Interface**, **DataType**, and **Diagnostic**, and its **MODULE** section lists **Definition** and **Configuration**; clicking a row opens that editor. The **DOC**, **CODE**, and **EXPORT** sections below it list files. **Signal** has no row: the Signal editor opens from the System editor canvas	The **Design** icon at the top of th and Navigation
edit history (History panel)	A single timeline that manages undo, redo, modified state, and saving for the project open in the window. Editors do not keep separate undo stacks, so **EDIT** > **Undo** reverts an edit made in any editor	The **History** tab in the bottom u
editor tab	The unit that opens one model node or one file. The title has the form `<project>:<node>` — for example `<project>:System`, `<project>:VFB`, or `<project>:Signal` — and at most **20** tabs can be open at once	The top of the editor area · 05.
editor group / split view	A split pane created by dragging a tab to the edge of the editor area	The editor area · 05. UI Layout a
dirty marker (●)	The dot attached to a tab that has unsaved changes. It is derived from entries in the edit history, so a change that leaves no entry is not marked	Next to the tab title · 05. UI Lay
license lock	The gate that turns writing, creating, deleting, and exporting on and off according to license state. Read and import are allowed in every state. The default is **locked (read-only)**, and it opens after sign-in and license validation	The upgrade dialog that appea License
ECU Extract / Machine Extract	The feature that produces a partial ARXML containing one node (an ECU or an HPC) and only what that node references. It cleans up the result so that no remaining element holds a dangling reference	Right-click an ECU node on the **Extract Machine** · 13. Import and
QuickStart workflow	The feature that generates a whole project model step by step with AI from a domain template (24 templates across 7 domains) and its variables. **In v1.0.1 no menu opens it**: the **PROJECT** menu no longer has **Quick Start**	Not reachable from the UI in v1.
PARA Base Set	The bundle that injects the PopcornSAR AP base configuration (service interfaces, data types, platform Adaptive Applications, the `PARA` composition, and machine content) into a project. It runs through the same merge pipeline as an ARXML import	The **PARA Platform Import** butto palette **PARA: Import Platform** ·
PRTE Base Set	The bundle that PRTE Platform Import puts into a project for one ECU: the PRTE module definitions (VSMD), the PRTE service interfaces and data types, and the service component of each selected BSW module, placed under `Composition/PRTE_<ECU>` and `Service/<ECU>/<Module>`. It runs through the same merge pipeline as an ARXML import	The **PRTE Platform Import** butto
PRTE Auto Configuration	The engine that writes one ECU's BSW module configurations from the design, only where the design gives one answer. It previews every change and runs as one undo step	**PRTE AUTO CONFIGURATION** in th **PRTE Auto Configuration** icon in
CP-SOA	The computation that derives signals, PDUs, and interfaces from service definitions so that SOME/IP service communication can be used on the Classic Platform. The derived result is stored in the model files, and only an origin marker (the `SOC` badge) is attached separately. Because the badge is only a marker, derived items can also be renamed, edited, and deleted	Command palette **Derive Signal** in **SERVICE DEPLOYMENT** of the · 07.3
DEST catalog	The correspondence table that records which IO Model element the `DESTINATION-TYPE` of an ECUC Foreign or Instance Reference (an AUTOSAR meta-type string) points to. It sorts 62 meta-types into three grades: candidates provided, name synthesized, and opaque	Entering a reference value in th
Conflict Resolution	The procedure for choosing, item by item, between **Keep Project Version**, **Use Imported Version**, **Merge Fields**, and **Keep Both** when an imported element holds a different value from an existing one. Items you leave undecided are	The **Conflict Resolution** dialog t

Term	Meaning	
	overwritten with the imported value	
Import Source Conflicts	A separate screen that appears when **several import files** selected at once define the same element differently. You pick a reference file and overwrite field by field	The **Import Source Conflicts** dia
`paio_*` rules	PAIO's own validation rules, which catch practical errors that the AUTOSAR standard constraints do not cover (for example `paio_ref_dangling`, `paio_someip_service_id_unique`)	The `Constr` column of the **VALI**
`.vfb.layout`	A file that holds only the layout information of the VFB canvas (positions, sizes, viewport). It is kept separate from the model data	`design/SWC/Layout/<composi`
IVC	In-Vehicle Communication. PAIO's name for the model of the in-vehicle communication topology, stored in `design/IVC.io.model`. It is edited in the System editor; the editor itself is no longer called IVC	`design/IVC.io.model` · 06. Sy
System editor (formerly the IVC editor)	The editor for the network topology: ECU and HPC nodes, controllers, connectors, clusters, and switches. Its canvas has the **Signal** and **Partial Network (PNC)** icons on the left edge and the **System** button on the right edge	**EDITOR** view ▸ **DESIGN** ▸ **Systen**
SYSTEM panel	The panel that shows the information of the SYSTEM element: **NAME**, **VERSION**, **DESCRIPTION**, plus **ECU MAPPING** in the System editor, or **ROOT COMPOSITION** and **COMPOSITIONS** in the VFB editor. It replaces the former System dashboard	The **System** button (PAIO logo) canvas · 11. SYSTEM Panel
rail (Activity Bar)	The column of icons at the far left of the window. From the top: **Design** (the **EDITOR** view), **vECU**, **Workflow**, **Search**; **Explorer** sits at the bottom	The far left of the window · 05.
Diagnostic editor	The design editor for diagnostics. It opens from the **DESIGN** section like the other design editors	**EDITOR** view ▸ **DESIGN** ▸ **Diagnc**
New Project dialog	The dialog that creates a project. **Name** is the project and folder name, **Location** is the folder the project is created in (**Browse…** picks it), and **Create** creates it	**New Project** on the **Welcome** ta Projects
Search View	The sidebar view that searches the saved files of the project's `design`, `doc`, and `code` folders; Design hits open the editor field that shows the value	The rail's **Search** icon, `Ctrl+Sh`
Search Editor	A search of its own in an editor tab, titled `Search: <query>`; read-only	**Open New Search Editor** · 05.9
Find Usages	Lists the design references that point to one element	Right-click a Design result row
Meta Class search	A box that takes an AUTOSAR meta class name and lists, one line per place, where PAIO designs that class (`Designed in <editor> › …`), with what must exist before it can be added. It lists no element of the project	`Ctrl+T`, command palette **Go t**
IntelliSense item	The status bar item that shows the state of the window's C/C++ language server (`Starting`, `Ready`, `Failed`, `Stopped`); its menu offers **Restart IntelliSense**	The left of the status bar · 05.5
AI Workflow	A graph of AI and tool nodes that carries a project from its input documents through design, code generation, build and verification. One document per project, `workflow/WORKFLOW.io.workflow`	The **Workflow** rail icon · 21. AI Wo
Knowledge base (LLM Wiki)	Typed, source-grounded pages that the Input Analysis node builds from the project documents, models and code; every later workflow node works from them	`workflow/knowledge/`, the **KN**
Rejected loop	A workflow edge from a node's **Rejected** output back to an earlier node; the run re-runs the owning node with the findings	Dashed edges on the Workflow

19.2 AUTOSAR Common

Terms that appear throughout ARXML, regardless of platform.

Term	Meaning	
ARXML	AUTOSAR's XML exchange format. PAIO supports it in both directions, import and export	Export: the **Export** icon the files go to the proje Import and Export
AR-PACKAGE	The package that holds elements inside an ARXML (the equivalent of a folder). An element's absolute path is built from it. The IO Model has no AR-PACKAGE concept, so PAIO strips packages on import and rebuilds them by rule on export	The exported ARXML · 1
SHORT-NAME	The identifier of an AUTOSAR element. The first character must be a letter, the rest may be letters, digits, or underscores, and the maximum length is 128 characters. PAIO **does not apply a name that breaks the rule to the model** (the field only shows an error and keeps the previous value)	The Name field in every
DESC	The descriptive text of an element. PAIO's Description field maps to it	The Description field in
ADMIN-DATA	The administrative information block attached to an element. PAIO carries an SDG here to record tool-specific information	The exported ARXML
SDG (Special Data Group)	The extension slot that AUTOSAR allows. PAIO records ownership relationships that no standard element can express — for example which ECU an ECUC configuration belongs to (`IO-ECUC-CONFIG-TO-ECU-MAPPING`), and which Machine a composition is mapped to (`IO-SWC-TO-MACHINE-MAPPING`)	The exported ARXML · 1
ECU-INSTANCE	The electronic control unit element of the Classic Platform. PAIO's **ECU** node is exported as this	The ECU node on the Sy
MACHINE-DESIGN	The element that holds the network side of an Adaptive Platform machine (connectors, addresses, and so on). PAIO's **HPC** node is exported as this	The HPC node on the Sy
SYSTEM	The top-level element that ties the whole system together. It holds FIBEX-ELEMENTS (the list of constituent elements), MAPPINGS, and ROOT-SOFTWARE-COMPOSITIONS	The SYSTEM panel (the canvas) · 11. SYSTEM Par
FibexElement	The collective name for the communication-related elements that the FIBEX-ELEMENTS list of SYSTEM must reference. Clusters, frames, PDUs, signals, ECU-INSTANCEs, and TP configurations all qualify	The exported `<Projec`
CATEGORY	A free-form string that states the kind of an element. For SYSTEM, PAIO writes one of `SYSTEM_DESIGN_DESCRIPTION` (the whole system), `ECU_EXTRACT` (an ECU extract), or `MACHINE_DESIGN_EXTRACT` (a Machine extract)	The ARXML produced b
DEST	An attribute on a reference (`*-REF`) tag that states the element kind of the target	The exported ARXML, a
XSD (schema)	The XML schema that defines the structure of ARXML. PAIO bundles R25-11 (`AUTOSAR_00054.xsd`) and R20-11 (`AUTOSAR_00049.xsd`)	The exported ARXML · 1
R25-11	The AUTOSAR schema release that PAIO **always outputs**. Every export uses this namespace	The header of the expo
AUTOSAR feature version	A separate axis handled by the conversion layer, ranging from `4.3` to `4.7` (default `4.7`). It is independent of the schema release above	Not exposed in the UI ·
Classic Platform (CP)	The traditional AUTOSAR platform based on ECUs, BSW, and the RTE	The **ECU** node on the Sy
Adaptive Platform (AP)	The POSIX-based AUTOSAR platform for high-performance computing	The **HPC** node on the Sy 01.5
constr_xxxx	A constraint number assigned by the AUTOSAR specification. Many PAIO validation rules use the number as-is	The `Constr` column of

19.3 Communication

The area covered by the System editor and the Signal editor.

Topology

Term	Meaning	Wh
Communication Cluster	The logical bus that nodes using the same protocol attach to. There are three kinds: CAN, Ethernet, and LIN	The cluster node on the System editor
Physical Channel	A physical channel inside a cluster. It holds which connectors are attached, and an Ethernet channel carries a VLAN ID. You edit it in the properties panel, not in the Cluster Configuration dialog	The **Physical Channels** section of the p
Connector	The attachment point where a node (an ECU or an HPC) joins a channel. There is one kind per protocol — Ethernet, CAN, and LIN — and each connector references one controller	Right-click an ECU or HPC node on the
Communication Controller	The element that corresponds to a node's communication hardware. There are four kinds: `ethernet`, `can`, `lin-master`, and `lin-slave`	Right-click an ECU or HPC node on the
Coupling Element	The element that represents an Ethernet switch, hub, or router. The kinds are `switch`, `hub`, and `router`	Right-click an ECU or HPC node on the
Coupling Port	A port on a switch, or a port owned by an Ethernet controller. Joining two ports corresponds to a physical cable	Right-click a switch ▸ **Add CouplingPo**
Network Endpoint	The IP-layer configuration of an Ethernet connector (IPv4/IPv6 address, netmask, gateway, DNS, TTL, and so on)	The **Network Endpoint** section of the p
Application Endpoint	The definition of a single TCP or UDP port. Service instances communicate through this port	The **Application Endpoints** section of t **Ethernet connector** is selected (it doe
Service Instance	The mapping that ties together which service this node provides (`provider`) or consumes (`consumer`) and which port it uses. It references an instance defined in the Interface editor	The **Service Instances** section of the p selected · 06.8.2, 09.4.2
PLCA	The physical-layer collision avoidance configuration of 10BASE-T1S Ethernet. It only displays imported values and offers no way to edit them	The **PLCA (10BASE-T1S)** section of the selected · 06.8.4

Frames, PDUs, and Signals

Term	Meaning	
Frame	The physical transmission unit that travels on the bus. PAIO models four kinds — CAN, LIN, FlexRay, and Ethernet — but provides only two editing tabs, **CAN** and **LIN**	The **CAN** and **LIN** tabs the left edge of the S
DLC	The data length code of a CAN frame. There are 16 selectable values, from `0 (0 bytes)` to `15 (64 bytes)`; 9 through 15 denote the CAN FD extended lengths (12/16/20/24/32/48/64 bytes)	The **DLC** field of a CAN
CAN Addressing	`standard` (an 11-bit ID) and `extended` (a 29-bit ID)	The **Addressing** field
LIN Frame Type	`unconditional`, `event-triggered`, or `sporadic`. The latter two reference another unconditional frame	The **Frame Type** field
LIN Checksum	`classic` or `enhanced`	The **Checksum** field o
PDU (Protocol Data Unit)	The protocol data unit carried inside a frame. PAIO distinguishes six kinds	The **PDU** tab of the Si
I-Signal I-PDU	The most common PDU, which carries signals and is sent periodically. It has a length, a cycle time, and signal mappings	**PDU** tab ▸ **+** ▸ **I-Signa**
Multiplexed I-PDU	A PDU whose internal layout changes according to a selector value. It is split into a STATIC part that is always present and a DYNAMIC part that switches	**PDU** tab ▸ **+** ▸ **Multip**
NM-PDU	A PDU dedicated to network management. NM nodes reference it as a transmit or receive target	**PDU** tab ▸ **+** ▸ **NM-PD**
N-PDU	The segment-carrying PDU used by the transport protocol layer. A CanTp or LinTp connection references it as a data PDU or a flow control PDU	**PDU** tab ▸ **+** ▸ **N-PDU**
DCM-I-PDU	The PDU that carries diagnostic (Dcm) data. The transport protocol layer segments and carries it	**PDU** tab ▸ **+** ▸ **DCM-I-**
General Purpose PDU	A general-purpose PDU whose use is distinguished by CATEGORY. The SOME/IP service discovery PDU is one of these	**PDU** tab ▸ **+** ▸ **Genera**
I-Signal	The communication-layer identity of a signal placed inside a PDU. The **Name** in the Signal editor is this name	The **SIGNAL** tab of the
System Signal	The system-layer identity of the same signal. One logical signal is exported with both names, the I-Signal name and the System Signal name. Data mappings reference the System Signal	`systemSignalName` i
Signal Group	A logical bundle of signals that must be handled together. Members are recorded by System Signal name	The `GROUP` column of
Text Table	The enumeration table that ties a signal value to a display label. When this table exists, the initial value field becomes a combo box. **In the current version it is read-only and cannot be changed in the editor** — entries are created only by an ARXML or DBC import	The `Text Table (N`
Factor / Offset	The coefficients that convert a raw value into a physical value: `physical value = raw value × Factor + Offset`	The `FACTOR` and `OFF`
Byte Order	The byte order: `big-endian` (most significant byte first) or `little-endian`	The `BYTE-ORDER` colu
Update Bit	The position of a separate bit that signals that the signal has been updated	The `UPDATE-BIT` colu
bit layout (Preview)	The picture that shows the bit arrangement inside a PDU or frame in color. Overlapping bits are shown in the error color	The **PREVIEW** tab to t editor · 07.10

Network Services

Term	Meaning	
NM (Network Management)	The network management that coordinates sleep and wakeup on the bus. PAIO covers CAN NM and UDP NM	Node Configuration ▸ **NETWORK CLUSTER** / **UDP NM CLUSTER** tab
NM Cluster	The set of nodes that share the same NM rules. It holds the cluster-level NM parameters	The NM tabs of Cluster Configu
NM Node	The element that attaches one ECU to one NM Cluster	Node Configuration ▸ **NETWORK**
NM Coordinator	The role that coordinates sleep across several NM clusters. Each channel carries `ACTIVE`, `PASSIVE`, or `NONE`	The NM coordinator diagram in
Transport Protocol (TP)	The layer that splits data too large for a single frame and sends it in pieces. PAIO covers **CanTp** and **LinTp**	Cluster Configuration ▸ **TRANSF**
Addressing Format	The ISO 15765-2 addressing mode of a CanTp connection — `STANDARD` / `EXTENDED` / `MIXED` / `MIXED29BIT` / `NORMALFIXED`	The **TRANSPORT PROTOCOL** tab ·
Target Address Type	`PHYSICAL` (one to one) or `FUNCTIONAL` (one to many)	The **TRANSPORT PROTOCOL** tab ·
Global Time	Time synchronization between nodes. Ethernet uses IEEE 802.1AS (gPTP); CAN uses an AUTOSAR-specific method	Node Configuration / Cluster Co
Time Domain	One synchronization domain. It carries master, slave, and gateway roles and a CRC policy	The **TIME DOMAIN** tab · 06.7.4
SOME/IP	The service-oriented communication protocol of the Adaptive Platform. Its deployment information holds the service ID and an identifier for each event, method, and field	A **SOME/IP Deployment** in the **SE**
DDS	Another service-oriented communication middleware. Its deployment information holds the topic name, the service ID, and so on	A **DDS Deployment** in the **SERVIC**

TSN Ethernet Switch

Term	Meaning	
VLAN	The virtual LAN that divides one physical Ethernet logically. Each channel carries a VLAN ID, and each port sets its transmission mode (`TAGGED` / `UNTAGGED` / `NOT SENT`)	The **VLAN MEMBER** **Ethernet Switch C**
PCP (Priority Code Point)	The priority value from 0 to 7 carried in the VLAN tag	The **PRIORITY REGE**
Priority Regeneration	The mapping that rewrites the PCP of a received frame to a different value	The **PRIORITY REGE**
Traffic Class	The grade of the transmit queue that a group of PCP values is assigned to	The **TRAFFIC CLASS**
FIFO	The transmit queue responsible for one traffic class. You can attach a shaper to it	The FIFO nodes in
CBS (Credit-Based Shaper)	The credit-based shaper of IEEE 802.1Qav. It carries `idleSlope` (the credit increase per second)	Right-click a FIFO
ETS (Enhanced Transmission Selection)	The bandwidth-allocating shaper and scheduler of IEEE 802.1Qaz. Weights are given as a percentage or as a weight value	Right-click a FIFO
ATS (Asynchronous Traffic Shaper)	The shaper of IEEE 802.1Qcr. The model and import support it, but **you cannot create one from the UI** (switch-level TSN settings replace it). Imported values are displayed and preserved	Shown as `ATS` in t
Frame Preemption	The feature that cuts a long frame short so that an urgent frame goes first. Each traffic class is set to `EXPRESS` or `PREEMPTABLE`	The FIFO settings i
Scheduler	The element that decides which of several FIFOs is sent first. It is either `STRICT PRIORITY` or `ETS`	The scheduler nod

---

19.4 Software Structure

The area covered by the VFB editor and the Interface editor, plus the mapping of compositions onto ECU and HPC nodes.

Term	Meaning	W
VFB (Virtual Functional Bus)	The abstract layer that deals only with the connections between software components, without regard to hardware placement	**EDITOR** view ▸ **DESIGN** ▸ **VFB**, tab <p
SWC (Software Component)	A software component. It communicates with the outside only through ports	The nodes on the VFB canvas · 08.2
Composition	A container-type SWC that holds other components. It has ports of its own as well	Right-click empty canvas ▸ **Add Co**
Atomic SWC	An SWC that is an indivisible unit of execution. The kinds are **Application**, **Complex Device Driver**, **Ecu Abstraction**, **Sensor Actuator**, **Service** (a BSW service), and **Adaptive Application**. The **Add SWC** menu adds **Composition** to these and so offers seven entries	Right-click inside a composition ▸ **A**
Interface Mapping node	A dedicated node that ties AP service interfaces to the SR and CS elements of CP. It is not an SWC kind but a separate node kind	Right-click empty canvas at root le
Prototype	An instance of an SWC **type** actually placed inside a composition. You can place the same type several times as several prototypes	Right-click inside a composition ▸ **A** editor's SYSTEM panel (the prototy SYSTEM Panel
drill-down	Entering a composition to edit its contents. The breadcrumb at the top of the screen shows your current location	Double-click a composition · 08.4
Port	The point at which an SWC exchanges data with the outside. There are three directions: **PPort** (provided), **RPort** (required), and **PRPort** (both)	Right-click a node ▸ **Add Port** · 08.5
Port Interface	The contract for what a port exchanges. A port references one interface	The Interface editor · 09. Interface I
Sender-Receiver (SR)	The interface that sends and receives data elements in one direction	The **Sender-Receiver** tab of the Inte
Client-Server (CS)	The interface that calls operations and receives results. Arguments carry an **IN**, **OUT**, or **INOUT** direction	The **Client-Server** tab of the Interfa
Mode-Switch	The interface that announces a mode change. It holds a ModeDeclarationGroup name, an initial mode, and a list of mode declarations	The **Mode-Switch** tab of the Interfa
Service Interface	The service interface of AP. It holds events, methods, and fields, together with a C++ namespace and a version	The **Service** tab of the Interface edi
Data Element	One piece of data carried by a Sender-Receiver interface. It has a name and a data type	The **Sender-Receiver** tab · 09.3.2
Operation	One call offered by a Client-Server interface	The **Client-Server** tab · 09.3.3
ComSpec	The communication properties of a port. The kind depends on direction and on whether a queue is used, and it holds an initial value, a timeout, a transmission mode, and so on	The ComSpec section of the proper
Internal Behavior	The definition of an Atomic SWC's internal behavior. It holds Runnables, RTE events, and Inter-Runnable Variables	The **SWC Internal Behavior** dialog · C
Runnable (Runnable Entity)	The unit of execution that the RTE calls. It holds `symbol` (the C function name in the generated code), a minimum start interval, and so on	The canvas nodes of the **SWC Interr**
RTE Event	The trigger that wakes a Runnable. The menu offers seven kinds — **TimingEvent**, **BackgroundEvent**, **DataReceivedEvent**, **OperationInvokedEvent**, **SwcModeSwitchEvent**, **InitEvent**, and **InternalTriggeredEvent** (the AUTOSAR name is `InternalTriggerOccurredEvent`)	The **RTE Events** add menu of the **SW**
Access Point	The point at which a Runnable accesses a port or a variable. There are nine kinds — **DataSendPoint**, **DataReceivedPoint**, **SyncServerCallPoint**, **DataWriteAccess**, **DataReadAccess**, **WrittenLocalVariable**, **ReadLocalVariable**, **ModeSwitchPoint**, **ModeAccessPoint**	The **Access Points** add menu of the
InternalTriggeringPoint	The origin of a trigger that wakes another Runnable from inside an SWC. It belongs to the owning Runnable. It sits in the **Access Points** menu, but it is stored as a separate element rather than as a VARIABLE-ACCESS	The tenth entry of the **Access Point**
Inter-Runnable Variable (IRV)	A variable shared by the Runnables inside one SWC	**SWC Internal Behavior** dialog ▸ righ

Term	Meaning	W
SWC-to-ECU mapping	The relationship that decides which composition is placed on which ECU or HPC. You can edit it from either side, and both stay synchronized	System editor: right-click a node ▸ 06.6; VFB editor: right-click a comp in **ECU MAPPING** of the System edit
Data Mapping	The mapping that ties a port's data element to a communication signal (or signal group). It is recorded in the System model	The **Data Mapping** section that app **composition boundary port** in VFB ·
InterfaceMapping	The definition that maps the events, methods, and fields of an AP service interface onto the data elements and operations of CP. **It is stored in the VFB model**, not in the Interface model	The **Mapping** tab of the Interface e
Root Composition	The top-level composition of the system. The SYSTEM element references it	The **ROOT COMPOSITION** field of the

19.5 Data Types

The area covered by the DataType editor.

Term	Meaning	
Application Data Type	A data type defined in terms of physical meaning (unit and range). On export it becomes `APPLICATION-PRIMITIVE-DATA-TYPE` and the like	The DataType edit
Implementation Data Type	A data type defined in terms of the implementation (the C language). It is exported as `IMPLEMENTATION-DATA-TYPE`	The exported `<Pr`
StdCpp Implementation Data Type	An implementation data type defined in terms of C++. It is exported as `STD-CPP-IMPLEMENTATION-DATA-TYPE`, and primitives are rewritten into C++ spelling (`uint8_t` and so on) only here	The exported `<Pr`
SW-BASE-TYPE	The base type that holds the bit size, the encoding, and the native C declaration	The `BaseTypes` p 13.3.5
CompuMethod	The conversion rule that ties a raw value to a physical value (or to text). PAIO **generates it automatically** — as `TEXTTABLE` for an ENUMERATION and as `LINEAR` for an ALIAS — and names it `CM_<type name>`	The exported `<Pr`
Unit	A physical unit. It holds a display name and an SI conversion factor and offset. 33 default units are included	The **Unit** tab of th
DataConstr	A value range constraint. It is generated automatically as `DC_<type name>` only when an ALIAS type has a minimum or a maximum	The exported `<Pr`
DataTypeMappingSet	The set that pairs Application types with implementation types. PAIO creates two automatically, `DataTypeMappingsCP` (for C) and `DataTypeMappingsAP` (for C++)	The exported `<Pr`
Category	The kind of a data type. There are nine — `PRIMITIVE`, `ALIAS`, `STRUCTURE`, `UNION`, `ENUMERATION`, `FIXED_ARRAY`, `DYNAMIC_ARRAY`, `STRING`, `MAP`	The **Category** field
built-in primitives	The 11 base types PAIO reserves — `boolean`, `float32`, `float64`, `sint8/16/32/64`, `uint8/16/32/64`. They are injected into memory automatically when a project opens and **are not stored in the model files**	The top of the Da
ALIAS	An alias for another type. Factor, Offset, Min, Max, and Unit are specified only here	The ALIAS details
Init Value	The initial value of an element. PAIO uses one shared widget on every initial value screen, and the input form is determined automatically from the structure of the target type	The `INIT VALUE` 07.11, the port Con
ValueSpecification	The AUTOSAR structure that expresses initial values in structure, array, and map form as well as single values. A form PAIO does not recognize is **preserved verbatim** and shown in a read-only viewer	The initial value ec

19.6 Adaptive Platform

The area covered by the **AA Internal Behavior** and **Machine Configuration** dialogs of the VFB editor and by the **Adaptive** tab of the Interface editor.

Basic Concepts

Term	Meaning	
HPC	High-Performance Computer. In PAIO it denotes an Adaptive Platform node, and it is exported as `MACHINE-DESIGN`	Right-click empty S
Machine	One AP execution environment. It holds processors and cores, the OS, Function Groups, and platform module instances	Click the HPC icon o (tabs **MACHINE**, **FUN**
Function Group	The group that bundles and defines the operating states of a machine. Each group has a state machine (mode declarations plus transitions)	Machine Configurati
MachineFG	The default Function Group that every machine must have. The default state machine is named `MachineState`, the initial mode is `Off`, and the declared states are `Off` / `Startup` / `Shutdown` / `Restart` / `Verify`	Machine Configurati
Mode Declaration	One state of a state machine	The state diagram o
Adaptive Application (AA)	The application SWC of AP. It holds both Executable and Process settings	Double-click on the
Executable	One build artifact. It holds a version, a category, a build type (`BUILD-TYPE-DEBUG` / `BUILD-TYPE-RELEASE`), the root SWC prototype, and the minimum timer resolution	**AA Internal Behavior**
Process	The process that actually runs an Executable. It holds start conditions (Function Group states), execution dependencies, log settings, and core affinity	**AA Internal Behavior** tab · 08.9
Execution Dependency	The condition that starts a process only while another process is in a given state (`Idle` / `Starting` / `Running` / `Terminated`)	**AA Internal Behavior**
Module Instantiation	A platform module instance placed on a machine. Each module kind has its own configuration screen	Machine Configurati
Error Domain	The domain that groups AP application error codes	The **Error Domain** ta
Software Cluster	An Adaptive Platform deployment unit: the Adaptive Applications shipped together, delivered by a software package and rolled out through vehicle packages. Stored in `design/SOFTWARE_CLUSTER.io.model`	The **Software Cluste**
IdsM instance	One node's instance of the Intrusion Detection System Manager, with its filters, limitation and the security events it reports. Stored in `design/IDS.io.model`	The **Intrusion Detec** **IDSM** tab of the ECU

Functional Clusters

Port interfaces per functional cluster are created on the **Adaptive** tab of the Interface editor, and the machine side is configured on the **MODULE** tab of t tab has **9** clusters (CM and UCM have no dedicated port interface); the **MODULE** tab has **10** modules (RDS has no corresponding module). The table lists b

Cluster	Meaning	
ara::log (LOG, Log and Trace)	Logging and tracing. It covers the DLT application ID, the log mode (`CONSOLE` / `FILE` / `NETWORK`), and the log level (`OFF` through `VERBOSE`)	Interface ▸ **Ad** **LOG** · 09.3.5, 08
CM (Communication Management)	Service-oriented communication management	Machine ▸ **MO**
ara::sm (SM, State Management)	State management. There are four interface kinds: StateClient, DiagTrigger, EmError, and PhmError	Interface ▸ **Ad** machines, acti · 09.3.5, 08.10
ara::phm (PHM, Platform Health Management)	Platform health monitoring. There are two interfaces: PhmSupervisedEntity and PhmRecoveryNotification	Interface ▸ **Ad** (checkpoints, 08.10
ara::nm (NM, Network Management)	The network management of AP. There is one interface, NetworkManagement, and the machine side deals with NetworkHandle	Interface ▸ **Ad** 08.10
ara::tsync (TS, Time Synchronization)	Time synchronization. There are two interfaces: SynchronizedTimeBaseConsumer and SynchronizedTimeBaseProvider	Interface ▸ **Ad** 08.10
ara::crypto (CRY, Cryptography)	Cryptography. There are five interfaces: CryptoKeySlot, CryptoKeySlotClient, CryptoCertificate, CryptoCertificateGroup, and CryptoProvider	Interface ▸ **Ad** 08.10
ara::per (PER, Persistency)	Persistent storage. There are two interfaces, PersistencyKeyValueStorage and PersistencyFileStorage, and the machine owns the actual storage	Interface ▸ **Ad** 08.10
ara::rds (Raw Data Stream)	Raw data streams. There are four interfaces: RawDataStreamClient, RawDataStreamServer, Ieee1722RawDataStreamConsumer, and Ieee1722RawDataStreamProducer. The **MODULE** tab has no corresponding module, so this is handled through mappings on the AA side	Interface ▸ **Ad** 09.3.5, 08.9
ara::idsm (IDS, Intrusion Detection)	Intrusion detection. There are five interfaces: SecurityEventReport, IdsmTimestampProvider, IdsmContextProvider, IdsmReportingModeProvider, and IdsmQualifiedEventReceiver	Interface ▸ **Ad** exported `<Pro`
ara::ucm (UCM, Update and Configuration Management)	Update and configuration management. There is no dedicated port interface; it is handled only through machine module settings (Master / Subordinate)	Machine ▸ **MO**
DLT	Diagnostic Log and Trace. A log transport specification that classifies logs by application ID and context ID	The DLT applic **Behavior** ▸ **EXI**

Adaptive Interface Badges

These are the badges that follow an interface name in the **Adaptive** tab list of the Interface editor (09.3.5).

Badge	Interface
`SC` / `DT` / `EM` / `PHM`	StateClient / DiagTrigger / EmError / PhmError
`PHM/SE` / `PHM/RN`	PhmSupervisedEntity / PhmRecoveryNotification
`NM`	NetworkManagement
`TS/C` / `TS/P`	SynchronizedTimeBaseConsumer / Provider
`CR/KS` / `CR/KSC` / `CR/CERT` / `CR/CG` / `CR/PRV`	CryptoKeySlot / KeySlotClient / Certificate / CertificateGroup / Provider
`RDS/C` / `RDS/S` / `RDS/1722C` / `RDS/1722P`	The four RawDataStream interfaces
`PER/KV` / `PER/FS`	PersistencyKeyValueStorage / FileStorage
`IDSM/SEV` / `IDSM/TS` / `IDSM/CTX` / `IDSM/RM` / `IDSM/QEV`	The five IDSM interfaces
`LnT`	LogAndTrace

19.7 ECUC

The area that covers Classic Platform BSW module configuration. Two editors work as a pair.

Term	Meaning	
ECUC (ECU Configuration)	The configuration system for BSW modules. It separates "which parameters exist" (the definition) from "what their values are" (the configuration)	**EDITOR** view ▸ **MODULE** ▸ **De**
Module Definition (`.def`)	The document that declares the structure of containers, parameters, and references a module can have. It holds no values	The Module Definition edito
Module Configuration (`.conf`)	The document that fills in actual values according to a definition. It always references a definition and is kept per ECU	The Module Configuration e
Container	The bundle that holds parameters and references. A definition has two kinds, `ParamDef` (ordinary) and `Choice` (either-or)	Right-click the DEFINITION tr
Container Instance	One actual instance of a defined container, created in a configuration. You can name it and duplicate it	The container tree ▸ **+** (Add
Parameter	An item that holds a value. A definition offers nine kinds — Integer, Float, Boolean, Enumeration, String, Function Name, Linker Symbol, Multiline String, Add Info	Right-click the DEFINITION tr
Reference	An item that points to another container instance. The definition screen can create four kinds (the four rows below)	Right-click the DEFINITION tr
Reference (simple)	Points to one instance that matches the container definition path written in `destination`	The candidate dropdown of
Choice Reference	Holds several `destination` entries. PAIO resolves each one separately and shows the combined candidates	The combined dropdown of
Foreign Reference	Points to an AUTOSAR element outside ECUC by `DESTINATION-TYPE` (a meta-type name)	A candidate dropdown if the not · 12.3
Instance Reference	Points to an outside element **together with a context chain**. `DESTINATION-CONTEXT` lists the element kinds it passes through, separated by spaces	Editing the two fields DEST
Symbolic Name Reference	An old reference kind removed in AUTOSAR 4.4.0. **Only import is supported**, and it does not appear in the **Add Reference** menu of the definition editor. On export it becomes an ordinary Reference	Only inside an imported mod
destination	The path of the target a reference points to. Inside PAIO it has the form `<module>/<container>[/<child>...]`, without an AR-PACKAGE	The destination field of the
DESTINATION-TYPE	The AUTOSAR meta-type name a Foreign or Instance Reference points to (for example `RUNNABLE-ENTITY`). PAIO connects it to an IO Model element through the DEST catalog	The PROPERTIES panel and t
DESTINATION-CONTEXT	The chain of element kinds an Instance Reference passes through on the way to its target	The PROPERTIES panel · 12.1
Multiplicity	How many of an item may exist. It is written as a lower and an upper bound, and an unlimited upper bound is `inf`	The multiplicity fields of the
Origin	A string that records who created the definition item (for example a standard name or a vendor name)	The Origin field of the PROPE
Config Class	The grade that states when a value must be fixed — `PRE-COMPILE`, `LINK`, `POST-BUILD`, `PUBLISHED-INFORMATION`. It is recorded paired with a variant	The Value and Multiplicity C badges (display only) of the
Config Variant	The configuration approach that determines when a setting is fixed — `VARIANT-PRE-COMPILE`, `VARIANT-LINK-TIME`, `VARIANT-POST-BUILD`, `PRECONFIGURED-CONFIGURATION`, `RECOMMENDED-CONFIGURATION`	Definition: **Supported Config Variant** · 12.1, 12.2
Post-Build Variant Used	Whether this configuration uses a post-build variant. It has three states — unset, `true`, and `false` — and unset is never changed to `false` on its own	The module-level Variant are
`EcucDefinitionCollection`	The element that bundles module definitions on export	The exported `<Project>_E`

Troubleshooting and FAQ

What This Chapter Covers

This chapter lists the symptoms you are most likely to meet in PAIO, with the cause, the message on screen, and the fix. Read it when something behaves use of each feature is in its own chapter. Places where the screen differs from the real behavior are called out in a blockquote. Terms are explained in 19. (

> The quoted messages are the English strings the app prints. Search this chapter for the wording you saw. For permission symptoms, also read the lice conflict resolution entry.

20.0 Find It by Symptom

Look up the symptom below and go to the section it points to. Other symptoms are grouped by area from 20.1 on. To search by an English message, use 2

Symptom	Short answer
Right-clicking the empty canvas inside a composition shows no menu	This is normal. Right-click the **composition frame**, not the empty area
I cannot find the button that runs validation	There is no button. Validation runs automatically on every edit
I pressed Save but the dot (●) on the tab is still there	The license has expired, or the write failed
I cannot rename or delete the `design`, `workflow`, or `export` folder	This is normal. The fixed folders of a project have no Rename or Delete entry
I edited an ARXML in the `export` folder but nothing changed	`export` is an output folder. Edit in the editor and export again
I cannot find the IVC editor	It is now the **System editor**. Click **System** in the **DESIGN** list; its tab is `<project>:System`
Signal is missing from the **DESIGN** list	The Signal editor opens from the **Signal** icon on the left edge of the System editor canvas
A signal is shown in red as `Overlap`	Two signals use the same bits. Adjust **START**
I tried to draw a connector and nothing happened	The protocol or the owning ECU does not match. No message appears
I cannot create a FlexRay frame	The current version has no FlexRay or Ethernet tab
The name I type keeps reverting to the previous one	It violates the AUTOSAR naming rule
No more tabs will open	You can have 20 tabs open at once
I want to split the screen but there is no menu for it	There is no menu. Drag the tab onto the right half of the editor area
I am not sure whether I can use an AI-generated model as it is	It is a draft. Always check it in each editor and in the Validation panel
The ECU properties panel has no **ECUC GENERATION** section	BSW configurations are generated with **PRTE AUTO CONFIGURATION** or the **PRTE Auto Configurat**
Search does not find a value I just typed	Search reads saved files only. Save, or choose **Save All & Search**
Clicking a search result opened a read-only `*.io.model` instead of an editor	No editor field shows that value; this is intended
A workflow run does not start, or a workflow node fails	Read the toast or the node's result; the common messages are listed
Build ECU is refused in a local Windows window	ECUs are built and run in the vECU. Open the vECU first

20.1 Startup, Login, and License

The app starts but shows only the login screen instead of the workspace

This is normal. PAIO builds the main layout only after you sign in. Use either option:

1. Click **Continue with Google**. Google sign-in opens in your default browser.
2. Type your address in `Email address` and click **Continue**. A six-digit OTP screen opens.

When login finishes, the overlay disappears and the header, the left sidebar, and the editor area appear.

I am logged in but cannot edit, and only an upgrade dialog appears

The license is expired (`expired`) or read-only (`readonly`). Both states allow only reading and importing.

License state	Read	Import	Edit, generate, delete	Export
`active` (Pro / Enterprise)	O	O	O	O
`trial` (within the trial period)	O	O	O	O
`expired`	O	O	X	X
`readonly` (integrity check failed)	O	O	X	X

The license indicator at the right of the status bar shows the state:

Display	Meaning
`License Expired` (warning icon)	Expired or read-only
`Trial: 12d` (or `Trial` alone)	Trial, days remaining
`Pro - 5 days` (warning icon)	Expires within 7 days
`Pro` / `Enterprise` (crown icon)	Normal

The indicator is clickable (tooltip `Click to manage license`). See 03. Installation, Login and License.

The title of the refusal dialog depends on the license:

License	Title
Within a trial, or a single refused operation	**Upgrade Required**
The trial has ended	**Trial Ended**
No plan	**Subscription Required**
A paid plan has expired	**Subscription Expired**

Close it with **Not now** or the close button (×). **View plans** opens the plan page.

> **When the dialog keeps coming back**: in the expired state it appears on every edit attempt. Only one opens at a time, so they do not pile up.

I left the app offline for a long time and now export is refused

The license cache is valid for **4 hours**. If the app cannot reach the server within that time, export is refused with `OFFLINE_EXPORT_DENIED: A server co` authenticated online can still edit and save. If you started offline and the cache has also expired, editing and saving are blocked too. Restore the networ and export works again.

If the license turns expired while you work, a different dialog appears: titled **Trial Ended**, **Subscription Required**, or **Subscription Expired**, without × or **Not out** button sits beside **View plans**. Once the license is valid again, the dialog closes by itself; no restart is needed.

I was signed out without warning

Someone signed in to the same account on another device. The dialog is titled **Signed in on Another Device**, with the body `You've been signed out be device.`

- With no unsaved tabs, a single **OK** button signs you out.
- With unsaved tabs, their names are listed and you choose **Save & Sign Out** or **Sign Out**. If the save fails, a **Save Failed** dialog asks whether to retry.

Clicking the icon does not start the app (Windows)

Part of the installation is damaged or quarantined by antivirus software. When the program bundle inside the executable differs from what was installed, **no log.** This differs from the integrity-check lock below, where a window opens.

Uninstall PAIO and install an official release again. Your user data is kept.

The integrity check locks features (Windows)

Right after startup, PAIO checks `PAIO.exe` against its own Authenticode signature in the background, **on Windows installations only**.

Check result	Behavior
Signature valid and signer matches	Normal. Nothing is locked
No signature	Nothing is locked
Cannot be determined	Nothing is locked. Expired certificate, revocation server unreachable, and so on
Hash mismatch (`hash-mismatch`)	Locked
Signer mismatch (`wrong-signer`)	Locked

When locked, this dialog appears once:

```
PAIO could not verify the integrity of its installation.
Please reinstall PAIO. Editing is disabled for this session.
```

The app does not exit. You can open and read projects; only editing is blocked. Diagnostics go to `%APPDATA%\PAIO\tamper-report.log` (in that parent f holds only the executable path and the signature state, so you can send it to support as it is. The fix is to install an official release again.

Sign-in fails with Can't reach the PAIO server

The card shows `Can't reach the PAIO server (pars.popcornsar.com). Check your network or proxy.` Check the network; on a company networ the proxy or firewall. Other sign-in failures are in 3.2.5.

Linux: PAIO runs without the Chromium sandbox

Started from a `.tar.gz` extract, the terminal shows `[paio] This system refuses unprivileged user namespaces and <folder>/chrome-sandbox : the Chromium sandbox. ...`. Run `sudo ./install.sh` once in the extracted folder (again for every new extract), or install the `.deb` package. See 3.1.4

20.2 Workspaces and Projects

A PAIO window holds one project: the folder it opened. The title shows `PAIO [<project>]`. Open a second project in another window (**PROJECT ▸ New Wi**

My project name is rejected

New Project on the **Welcome** tab and **PROJECT ▸ New Project...** open the same dialog with **Name** and **Location** (with **Browse...**). Once both are filled in, `The <Name>` appears and **Create** becomes available.

Condition	What you see
Empty	**Create** stays disabled
Contains anything other than letters, digits, hyphens, and underscores	`Project name can only contain letters, numbers, hyphens, and underscores`
A reserved Windows device name	`"<name>" is a reserved system name and cannot be used`
A folder of that name already exists in the location	`Project "<name>" already exists` (after you click **Create**)

The reserved names are `CON`, `PRN`, `AUX`, `NUL`, `COM0` to `COM9`, and `LPT0` to `LPT9`, in any case. For example, `DoorEcu` and `Door_Ecu-01` pass; `Door Ecu` an the name or location and click **Create** again.

There is no Rename Project entry

The project name is the folder name. The Explorer lists the folder's contents without a project row, so there is nothing to rename.

A file does not appear in the Explorer

The Explorer lists every file and folder on disk except these, hidden at every level by default:

Hidden entry	What it is
`.paio`	PAIO's own working data
`.devcontainer`	The Dev Container configuration PAIO writes
`.git`	The Git repository store
`<project>.io.prj`	The project marker file

To show them, choose **More Actions...** ▸ **Show Hidden Entries** in the Explorer header. The **DOC**, **CODE**, and **EXPORT** sections of the **EDITOR** view show fewer **Refresh Explorer** at the top of the Explorer.

I am looking for the `import` folder

A project has six folders on disk: `code`, `design`, `doc`, `export`, `import` and `workflow`. The Explorer shows all six, sorted by name.

Expanding the `design` folder shows model files, not editors

This is intended. In the Explorer, `design` is an ordinary folder of model files (`IVC.io.model`, `SYSTEM.io.model`, `VFB.io.model`, the `SWC` and `Module` fold opens a read-only view, not the editor.

Open the editors from the **EDITOR** view: **System**, **VFB**, **Interface**, **DataType**, and **Diagnostic** under **DESIGN**, **Definition** and **Configuration** under **MODULE**. To op right-click the empty area of the Explorer and choose **Reveal in File Explorer**.

I cannot rename a fixed folder

`code`, `design`, `doc`, `export`, and `workflow` have no rename or delete entries; `import` has the full folder menu (4.6.5).

Folder	Context menu
`design`	**Import**, **Export**, **Code Generation** (**PRTE** / **PARA**), **New File**, **New Folder**
`doc`, `code`	**New File**, **New Folder**
`workflow`, `export`	**New File**, **New Folder**
`import`	The full folder menu, including **Rename** and **Delete**

I deleted a fixed folder in the OS explorer and it came back

This is intended. When the file watcher sees a fixed folder deleted (`unlinkDir`), it recreates it, except while the project itself is being deleted.

Permission denied appears when I open a folder or work in the Explorer

The file system refused PAIO (for example an SSH folder your user may not write, or a folder you may not read). PAIO shows the toast `Permission denied`

Where	What you see
Opening a folder you may not read, or in which the project folders cannot be made	The toast, and `Permission denied` with the folder path and an **Open Project** button in th
A **DOC**, **CODE**, or **EXPORT** section whose folder cannot be read	The toast with the folder name, and `Permission denied` with the folder path in the sect
New File, **New Folder**, **Rename**, or **Delete** in the Explorer	The toast
A save	The toast with the name of the file that was not saved

Fix the permissions of the folder, or open another one with **Open Project**. **Refresh** in the **EDITOR** view header reads a refused section folder again.

20.3 Editing and Saving

The dot (●) on the tab does not go away

The tab has unsaved changes. Save in one of three ways:

1. `Ctrl+S` saves the active editor, through the same path as the **Save** menu even when focus is outside the editor.
2. **PROJECT** ▸ **Save** (`Ctrl+S`) does the same.
3. **PROJECT** ▸ **Save All** saves every unsaved tab of the project.

A successful save removes the dot. If it stays, either:

1. The license has expired. The upgrade dialog appears instead (see 20.1).
2. The write failed. When the file system refuses it (no write permission), the toast `Permission denied` names the file that was not saved (see 20.2). C a disk error) may show no message.

> **There is no Save As.** The **PROJECT** menu has only **Save** and **Save All**, and `Ctrl+Shift+S` works like `Ctrl+S`. Model files have fixed names, so a copy und Keyboard Shortcuts.

The name I typed is not applied and only an error marker appears

The name breaks the AUTOSAR SHORT-NAME rule: a letter first, then letters, digits, and underscores only. Where a name is rejected, a 128-character limit al

Condition	Message shown
Empty or whitespace only	`Name is required`
Contains characters the rule does not allow	`Must start with a letter, then letters/digits/underscores only`

An empty name is shown in red and never written: it would overwrite references in other files with an empty string, which cannot be undone.

Disallowed characters raise only an orange warning, and whether the value is blocked depends on the screen. The Adaptive tab and the Persistency inter Sender-Receiver, Client-Server, and Mode-Switch tabs warn but still write the value. Check that no field still has a warning before you export.

For example, renaming to `1stPort` on the Adaptive tab is rejected; `Port1` is applied at once.

Undo covers only this window's project

Undo, Redo, and **Save All** act on the window's one project. A project in a second window keeps its own history and unsaved changes. The tab dot (●) is tr

I edited a file outside PAIO and the editor does not pick it up

If the file has unsaved edits or an edit is in progress, the reload is skipped to protect your edits, and a warning toast titled `Changed on disk` appears. In grey line reads `Unsaved edits kept`; in the code editor it reads `<file name>: unsaved edits kept`.

The warning does not repeat for the same file within 5 seconds, so a tool that writes repeatedly does not flood you. Any of these counts as an edit:

- Any one of the files that editor handles is unsaved
- A value being typed is not applied yet, such as during typing or while editing a grid cell
- That editor's configuration dialog is open and holds a change

To resolve it, save to overwrite the file on disk, or close and reopen the tab to discard your changes.

The dialog that asks about saving when a window or a tab closes

Situation	Title	Body
Closing one tab	**Unsaved Changes**	`Do you want to save the changes to <tab name>?`
Closing the window	**Unsaved Changes**	`Do you want to save the changes to <tab names>?`

Close Others, **Close Saved**, and **Close All** close tabs one at a time and ask about each unsaved tab like **Close**. If work such as a workflow run is in progress confirmation asks first whether to stop it.

`Ctrl+B` runs a build instead of toggling the sidebar

In the code editor (Monaco), `Ctrl+B` is **Build**. Click outside it and the key toggles the sidebar. There, `Ctrl+F5` is **Build & Run**, `Shift+Alt+F` is **Format Docu** 18. Keyboard Shortcuts.

No more tabs will open

You can have **20** tabs open at once. Past that, a new tab silently fails to open.

> **Where display and behavior differ**: no message is shown. Only the developer console records `Maximum number of tabs (20) reached`. If clicking a fil

To clean up, use the tab context menu entry above.

20.4 Import

Start an import with **Import**, the first icon at the right of the **DESIGN** header in the **EDITOR** view, or right-click `design` in the Explorer ▸ **Import**. The picker `Supported Files` (`.arxml` + `.dbc`), `ARXML Files`, and `DBC Files`, and takes several files at once. See 13. Import and Export.

I am worried that the edits I made before importing will be lost

They will not. Before the work starts, the import **saves the unsaved edits of every open editor**, because it rewrites model files and the editors re-read the

The whole import is one undo step, so one `Ctrl+Z` reverts it. **EDIT** ▸ **Undo** reads `Undo Import <file name>` with the **first** selected file, but the undo cov

I ran the import again and it was rejected

A second import does not start while the previous one runs or waits at a conflict dialog. Finish or cancel that one first. The progress dialog shows one of:

```
Import rejected: an import is already running. Finish or cancel it first.
Import rejected: an import is waiting on the source-conflict dialog. Finish or cancel it first.
Import rejected: an import is waiting on conflict resolution. Finish or cancel it first.
```

This keeps the second import from erasing the first one's files and undo information. **PARA Platform Import** in the Machine Configuration dialog uses the `Base configuration rejected: ...`. Adding the PRTE base set is refused with `PRTE Base Set rejected: ...`.

An import in an SSH window failed because the connection closed

While an ARXML import runs in a remote window, closing the connection asks first (**Close Connection**: `An ARXML import is running. Closing the con` the connection; **Close Connection** cancels the import and closes the connection once the import has stopped. Nothing of the import is written to the PC

If the connection drops during the import, the import fails with `The remote connection closed during the import. Part of it may be on the se` Reconnect and run the import again.

The import stopped and an "Import Source Conflicts" window appeared

Several selected files **define the same element differently**. The conflict is between the input files, not with the project. The title shows `(N elements div`

1. Select a conflicting element in the tree on the left. The per-file values appear on the right.
2. Choose the reference file for each element. To apply one file to every undecided item, pick it in **Prefer source:** and click **Apply to remaining**.
3. Click **Resolve 3/7** (decided / total) at the bottom right to continue. The button is always clickable. Undecided items take the value from the last file a `DUPLICATE_ELEMENT_MISMATCH` warning.

To stop, click **Cancel Import**. The confirmation (**Cancel Import?**) reads `Nothing has been written yet; your project files stay untouched.`; nothir the conflict window.

A conflict window appeared after the import

A project element conflicts with an incoming one. The window is titled `Conflict Resolution: N conflicts`. Choose per element:

Button	Behavior
Keep Project Version	Keeps the existing project value
Use Imported Version	Overwrites with the imported value
Merge Fields	Choose which side to use field by field (shown only when the two sides differ)
Keep Both	Renames so that both are kept (shown only when the element allows it)

Switching from **Merge Fields** to **Keep Project Version** or **Use Imported Version** asks `Discard Field Edits?`; confirming discards the per-field choices. Wh

> **Clicking Resolve without deciding overwrites with the imported values.** This differs from the source-conflict window above. If items remain, **Unresolvec** `will be overwritten with the imported values.` **Overwrite & Continue** gives every undecided item the imported value; **Cancel** returns to the conf

To cancel, click **Cancel** left of **Resolve** (there is no `Cancel Import` button here). The `Cancel Import?` confirmation reads `Discards this import and r` `state.` **Cancel Import** restores the pre-import state; **Keep Resolving** returns to the window. On success the toast is `Import cancelled` with the line `Pro` `Rollback failed` with the reasons.

From **Import ECUC Configuration**, cancel works differently: the confirmation reads `Conflicting files keep your current values; already-importec` without conflict stay.

After you decide, the import continues and shows the toast `Import complete` with a line such as `3 replaced, 1 kept`. If applying fails, the whole proj shows the reasons.

The import finished but some data is missing

> **Where display and behavior differ**: the ordinary **Import** reports only a count (`Import complete: N elements imported from M file(s)`). **Its warning** (`UNRESOLVED_*`), skipped ECUC configurations, and discarded module-level parameters.

- To re-import only the ECUC configuration, right-click **Configuration** under **MODULE** in the **EDITOR** view and choose **Import ECUC Configuration**. This pa dialog (see the next entry).

An ECUC configuration was not imported

A configuration is skipped when its **module definition** is not in the project. If the same ARXML also holds the `ECUC-MODULE-DEF`, the definition is importec

Import ECUC Configuration reports the result in **ECUC Configuration Import (Partial)**:

```
Imported 3 module configuration(s).

Skipped, Module Definition not found:
  • Dcm
  • Dem

Rejected, definition mismatch:
  • Com: Container definition not found in "Com": ComSignalX (instance "ComSignal_0")
```

Category	Meaning	What to do
Skipped	That module's definition file is not in the project	Import the definition ARXML first
Rejected	The definition exists, but a container, parameter, or reference does not match the definition tree structurally	Align the definition version, or check the configur

Through the ordinary **Import**, these two categories stay warnings and are not shown.

I imported an ECUC module definition and some items are missing

Parameters and references directly under `ECUC-MODULE-DEF` are **discarded on purpose**: the R25-11 XSD has no slot for them there, so the export would vi are ignored for the same reason.

Module definitions come in with the ordinary **Import**, which writes any `ECUC-MODULE-DEF` it finds as a module definition. There is no separate menu entry

> **Where display and behavior differ**: no menu imports definitions alone, and the "discarded" warning is never shown. If items are missing, check the ARX under `ECUC-MODULE-DEF`.

Korean and other special characters in a DBC file are garbled

Detection recognizes **only a UTF-8 BOM**; without one the file is read as ISO-8859-1 (Latin-1). Save the file as UTF-8 with BOM.

20.5 Export

Run an export with **Export**, the second icon at the right of the **DESIGN** header in the **EDITOR** view, or right-click `design` in the Explorer ▸ **Export**. See 13. In

I clicked Export and it failed

Export needs **an active license and online validation**. On failure the line of the format in the **Export** dialog shows one of:

Cause shown	Meaning
`OFFLINE_EXPORT_DENIED: A server connection is required`	The server could not be reached
`PLAN_EXPIRED: Export is not permitted on this plan`	The license has expired
`INTEGRITY_EXPORT_DENIED: Installation integrity check failed`	The installation failed the integrity check

With an expired license the upgrade dialog usually blocks the menu first; `PLAN_EXPIRED` appears only when the server refuses afterwards. The same che the DataType export.

I do not know where the output files are created

In the folder of **FOLDER** in the **Export** dialog, by default `<project>/export/`. ARXML writes one `<project name>_*.arxml` file per kind of content the pr cluster. The result line reads, for example, `R25-11, 298 elements` or `3 files`.

Extract ECU (ECU node) or **Extract Machine** (HPC node) in the System editor writes to `<project>/export/<node name>/`, and the message lists the files `name>/`.

I edited an ARXML in the `export` folder and the design did not change

This is intended. `export` is a **one-way output folder**. PAIO never reads it, and the next export overwrites a file of the same name. Edit the model in its edit

> To bring in an outside ARXML, do not put it in `export`; use **Import** on the **DESIGN** header (13. Import and Export).

My latest edits are missing from the export

> **Where display and behavior differ**: ARXML export, including **Extract ECU** and **Extract Machine**, **reads the saved files in** `design/`. Unsaved content is not

I re-imported an exported ARXML and the diagram layout changed

This is intended. Node coordinates and sizes **are not written to ARXML**. System editor coordinates live in `design/IVC.io.model`, the VFB layout in `desig` On import, System editor coordinates are generated anew on a grid. To keep a layout, move or copy the project folder itself.

I want to export to DBC

Choose **DBC** in the **Export** dialog, or right-click one CAN cluster in the System editor and choose **Export to DBC** (see 13. Import and Export 13.6).

20.6 ECUC Generation

I cannot find ECUC GENERATION or Generate ECUC

The System editor's ECU properties panel has no **ECUC GENERATION** section and no **Generate ECUC** button. A Classic ECU's BSW configurations are generat which shows a preview before writing. Start it from:

- **PRTE AUTO CONFIGURATION** in the header of the ECU Configuration dialog of the VFB editor (08. VFB Editor 8.16).
- The **PRTE Auto Configuration** icon on an ECU row of the **ECU** list in the Module Configuration editor (12. ECUC 12.2.6).

Configurations can also be imported:

- Run **Import** on the **DESIGN** header with an ARXML that contains the module configurations.
- Right-click **Configuration** under **MODULE** in the **EDITOR** view and choose **Import ECUC Configuration** (see 20.4).

View and edit them in the editor that **MODULE** ▸ **Configuration** opens. See 12. ECUC.

20.7 Validation

There is no run button. The window's project is validated as soon as the window opens, and again:

- When you edit the model; it updates while you type as well
- When the active editor tab changes
- When an ARXML import finishes; here it reads the content on disk
- When you open another folder in the window

See 14. Validation.

The three messages shown when the result is empty

Tell these apart:

Message shown	Meaning
`No validation issues.`	Validation ran normally and found no violations
`Select a project in the Explorer to run validation.`	No target project has been determined
`Validation could not run. See the developer console for details.`	Validation could not run and failed

The third does not mean "no problems". The status bar Validation (`V`) badge then shows `!`, and the tooltip reads `Validation could not run. Open th`

For the second, check that a project is open (title `PAIO [<project>]`), then open one of its editors.

I do not know what the status bar badge means

Badge	Meaning
None	Zero violations
A number	The total violation count (the color follows error > warning > info). Above 99 it is shown as `99+`
`!`	Validation failed to run

The tooltip reads `Validation: N error(s), M warning(s), K info(s)`. The red badge beside the panel title counts **errors only** and is hidden when th

I clicked a violation row and it did not navigate anywhere

Two causes:

1. The rule has no navigation target. Clicking does nothing, silently.
2. The element is in an area without a target editor. Navigation works for:

Violation area	Editor that opens
`datatype.*`	DataType editor
`interface.*`	Interface editor
`vfb.*`	VFB editor
`ivc.*`, `system.*`	System editor
`signal.*`	Signal editor
`diagnostic.*`	Diagnostic editor

ECUC configuration and module definition violations have no click target; open the editor under **MODULE** yourself.

On success, the element is selected and the violated field is marked in the severity color. Clicking another row clears the mark.

20.8 AI

The AI chat button is disabled

The chat toggle sits at the right of the header (tooltip `AI Chat Panel`). If the tooltip reads `Plan expired`, the license is expired or read-only and the bu placeholder then reads `Plan expired, upgrade to continue`.

Nothing in the provider list is available

Name in the list	Kind	Requirement
Claude [OPUS]	CLI	`claude` CLI installed and logged in
Claude [GLM]	API	API key registered
Gemini	CLI	`gemini` CLI installed and logged in
Codex	CLI	`codex` CLI installed and logged in
Ollama	Local server	Ollama local server running

With none available, sending fails with `LLM provider is not available`.

Open the provider dialog with the gear in the chat panel header (tooltip **Manage AI accounts**) or **Manage AI Accounts...** in the command palette (`Ctrl+Sh` **VIEW** / **HELP**) does not have it.

Availability is rechecked every 30 seconds. After you start a login from the dialog, PAIO rechecks **every 5 seconds for up to 2 minutes** and stops as soon a minutes, close and reopen the dialog.

I do not like the model the AI produced

Press `Ctrl+Z`. An AI chat write is one undo step even across several files. **EDIT ▸ Undo** shows the file count, as in `Undo AI: 3 file(s)`, not the element

I cannot find Quick Start

This version has no **Quick Start** menu entry. Create a project with **PROJECT ▸ New Project...** (see 20.2) and build the model in the editors or with AI chat. S

20.9 Code and Build

Build reports that it cannot find a compiler

A local build uses the bundled `gcc` / `g++` first, then a compiler on `PATH`. With neither, the OS error is printed to the output panel. Install a compiler or che

On a remote build (WSL, container, SSH), exit code 127 or `command not found` adds this hint:

```
Hint: 'gcc' was not found in this environment. Install a C/C++ toolchain
(e.g. 'sudo apt install build-essential' on Debian/Ubuntu).
```

Defaults: options `-Wall -g -std=c++17`; `g++` for `.cpp`, `.hpp`, `.cc`, and `.cxx`, `gcc` otherwise; timeout 60 seconds locally, 120 seconds remotely.

I cannot build a single `.rs` file

Rust needs a `Cargo.toml` project. **Build & Run** on a single `.rs` file prints:

```
Rust files require a Cargo.toml project for Build & Run.
Create a Cargo project (cargo new <name>) or use the terminal:
  rustc "<file path>" && ./<output>
```

Build & Run walks up to 12 levels from the file, looking for `CMakeLists.txt` and then `Cargo.toml`. The hint appears only when neither is found.

I edited `code/Cargo.toml` directly and my change was reverted

This is intended. If `code` holds any `.rs` file (subfolders included, `target` and `.git` excluded), PAIO **regenerates** `code/Cargo.toml`, because rust-analyz

When it runs	`version` generated
Right after app startup, and whenever the window's project changes	`0.1.0`
When you open a `.rs` file in the code editor and rust-analyzer starts	`0.1.2`

Both write package name `paio-code`, edition `2021`, and one `[[bin]]` per `.rs` file. Only the version differs, so that line may flip back and forth; it does n in a subfolder of `code` and it is not overwritten.

Code completion does not work

The language server may still be starting, or it failed. Check for diagnostics in the **PROBLEMS** panel below the code editor. For Rust, see the `Cargo.toml` check 20.10 Remote.

For C and C++, check **IntelliSense** at the left of the status bar. If it reads `IntelliSense: Failed` or `IntelliSense: Stopped`, hover it for the reason, the

---

20.10 Remote

The remote language server is not deployed

The bundled binaries are for **glibc-based linux-x64** only. Deployment is refused if:

Condition	Result
`uname -s -m` is not `Linux` + `x86_64`	Refused (`not-linux-x64`)
`/lib/ld-musl-*` exists (Alpine and other musl distributions)	Refused (`not-linux-x64`)
The bundled binary is not in the package	Refused (`bundle-missing`)
The remote `$HOME` could not be resolved	Refused (`home-unresolved`)

On SSH the toast `No code intelligence` with the line `Remote is not glibc Linux x64` appears (once per workspace).

On WSL the deployment banner shows the error instead:

Condition	Error string
The distribution is not x86_64	`WSL distro is not x86_64 — only Linux x64 is supported`
A musl (Alpine) distribution	`WSL distro uses musl (Alpine) — only glibc is supported`

The screen reloads after I connect or disconnect

This is intended. Connecting, disconnecting, and returning to a local folder reload the window to reset its state.

> **Caution.** The reload does not ask about saving. Save before you connect or disconnect.

There is no Rebuild or Reopen Locally in the container menu

In a vECU container window, the remote indicator menu has only **Close Connection** (17.1.3). It copies the container's work back to the host project folder a the container window with its ⨯ copies the work back the same way (17.3.7, 17.10), so there is no **Reopen Folder Locally**. Attaching to an arbitrary running

To start with a fresh container, remove it from its card in the **vECU** view: the three-dot button (tooltip `More`) ▸ **REMOVE CONTAINER** (17.3.6). The workspac next **OPEN** builds a new container.

Build ECU is refused with "A Windows host is not an execution target"

ECUs are built and run in the project's vECU. In a local Windows window, **Build ECU** and **Run ECU** are refused with the toast `Not available here` and the l `target (PRTE RELEASE.md :249). Build and run the ECU in its vECU: open it from the vECU view.`; in an SSH window the line reads `ECU buil (remote-workspace plan §2 phase 3).` Open the vECU from the **vECU** view (17.3) and use **Build ECU** there. See 17.9.2 and 17. Remote Development (SSH,

---

20.11 Performance

A diagram with many nodes is slow

The System and VFB editors have two icon toggles in the toolbar at the bottom left of the canvas (names in their tooltips):

Tooltip	What turning it off removes
Toggle Grid	The background grid
Toggle Connector Animation	The animation flowing along the connectors

They change the display only, not the model. **Fit to View** in the same toolbar draws everything at once; zooming in on your work area draws less.

Many open tabs make the app slow

Up to 20 tabs can be open (see 20.3). Right-click a tab and use **Close Saved** to close only saved tabs, or **Close Others** / **Close All**, which ask about each un

---

20.12 Other Checks

Check	How
App version	The `v1.0.1` indicator at the right of the status bar, or **HELP** ▸ **About PAIO**
Product information and contact	**HELP** ▸ **About PAIO** shows the version and `contact@popcornsar.com`
Check for updates	**HELP** ▸ **Check for Updates**
Contact support	**HELP** ▸ **Report Issue** opens `https://autosar.io/support/qna` in your default browser
Return to the Welcome tab	**HELP** ▸ **Welcome**
Find a feature by command name	The command palette (`Ctrl+Shift+P`)

On macOS the OS menu bar also has a Help menu, whose **Report Issue** opens `https://github.com/popcornsar/PAIO/issues`. Windows and Linux use a h
menu is visible.

I do not know what the update check result means

On Windows, and on Linux installed from the `.deb` package, PAIO can download and install a new version itself. Look up the notification wording:

Result	Display
A new version is available	`PAIO <version> is available.`
Checking	`Checking for updates…`
Downloading	`Downloading PAIO <version>… <n>%`
Already downloaded	`PAIO <version> has already been downloaded.` under the notification
Preparing	`Updating PAIO <version>.`
Installing	`Updating PAIO <version>. PAIO restarts when the update finishes.`
Installing (Linux `.deb`)	`Enter an administrator password to install PAIO <version>.`
The window is closed for the install (system notification)	`Installing PAIO <version>: This takes about 5 minutes with nothing on screen. PAIO opens again when it finishe` (Windows) / `PAIO opens again when the install finishes.` (Linux `.deb`)
The password prompt was dismissed or failed (Linux `.deb`)	`The install was cancelled. Press Update to try again, or install it from the download page.`
Another program kept installing for 2 minutes (Linux `.deb`)	`Another program is installing software. Press Update again when it finishes.`
Other PAIO windows are open	`Close the other PAIO windows to update (<n> open).`
Tasks are running	`PAIO <version> installs when these tasks finish: <tasks>.`
Other windows and tasks cannot be checked	`PAIO cannot tell whether other PAIO windows or tasks are running, so it cannot update yet.`
Installation failed	`PAIO <version> could not be installed automatically. Install it from the download page.`
The installer failed the signature check	`The downloaded PAIO <version> installer could not be verified and was not installed. Install it from the downl page.`
The app cannot install here (Linux `.deb`)	`PAIO <version> cannot be installed from the app on this system. Install it from the download page.`
The package failed the check (Linux `.deb`)	`The downloaded PAIO <version> package could not be verified and was not installed. Install it from the downloa page.`
The install failed (Linux `.deb`)	`PAIO <version> could not be installed. Install it from the download page.`
The update was not applied	`PAIO <version> was not installed. You are still on PAIO <current>. Install it from the download page.`
The update was installed	`Updated to PAIO <version> (from <old>).`
Projects were closed for the update	`Projects closed for the update: <projects>.`
Reopening failed	`Could not reopen: <projects>.`
Up to date	`PAIO is up to date.`
Not signed in	`Sign in to check for updates.`
The sign-in could not be stored (Linux)	`Cannot check for updates: your sign-in could not be stored securely on this computer. Unlock or set up your keyring, then restart PAIO and sign in.`
No release for this platform	`No update is available for this platform yet.`
The check failed	`The update check did not complete.`
Development mode	`Auto-update is disabled in development mode.`

When **Update** is refused, a note appears under the notification:

Note	What to do
`Some changes could not be saved. PAIO was not updated.`	Save the open files, then press **Update** again
`Close the open dialog first, then press Update again.`	Close the dialog and press **Update** again
`Another PAIO window is updating.`	Wait for that window's update
`The update could not start.`	Press **Update** again, or use the download page

If the browser does not open for **Download page**, the note `Could not open the browser. Download PAIO <version> from <address>.` gives the add

Download page opens the download page in your default browser for a manual install. When the background check runs is described in 3.5.

20.13 Common Message Index

Search by the wording you saw on screen.

License and Integrity

Message	Entry
`Plan expired`	20.8
`Plan expired, upgrade to continue`	20.8
`License Expired`	20.1
`Upgrade Required` / `Trial Ended`	20.1
`Subscription Required` / `Subscription Expired`	20.1
`Signed in on Another Device`	20.1
`PAIO could not verify the integrity of its installation.`	20.1
`Can't reach the PAIO server (pars.popcornsar.com). Check your network or proxy.`	20.1
`Sign-in wasn't completed. Try again.`	3.2.5
`Too many attempts. Try again in a few minutes.`	3.2.5
`Couldn't sign in. Try again, or contact PopcornSAR support.`	3.2.5

Import and Export

Message	Entry
`Import rejected: an import is already running. Finish or cancel it first.`	20.4
`Import Source Conflicts`	20.4
`Nothing has been written yet; your project files stay untouched.`	20.4
`Conflict Resolution: N conflicts`	20.4
`Unresolved Conflicts`	20.4
`N undecided conflicts will be overwritten with the imported values.`	20.4
`Discard Field Edits?`	20.4
`Discards this import and restores your project to its pre-import state.`	20.4
`Conflicting files keep your current values; already-imported files stay applied.`	20.4
`Rollback failed`	20.4
`Import complete` / `Import failed` (toast)	20.4
`An ARXML import is running. Closing the connection cancels it.`	20.4
`The remote connection closed during the import. ...`	20.4
`ECUC Configuration Import (Partial)`	20.4
`OFFLINE_EXPORT_DENIED: A server connection is required`	20.5
`PLAN_EXPIRED: Export is not permitted on this plan`	20.5
`INTEGRITY_EXPORT_DENIED: Installation integrity check failed`	20.5

Editing, Validation, and Projects

Message	Entry
`Changed on disk` / `Unsaved edits kept`	20.3
`Permission denied`	20.2
`Do you want to save the changes to ...?`	20.3
`Name is required`	20.3
`Must start with a letter, then letters/digits/underscores only`	20.3
`Project name can only contain letters, numbers, hyphens, and underscores`	20.2
`Project "..." already exists`	20.2
`"..." is a reserved system name and cannot be used`	20.2
`Validation could not run. See the developer console for details.`	20.7
`Select a project in the Explorer to run validation.`	20.7
`No validation issues.`	20.7

Code and Remote

Message	Entry
`Rust files require a Cargo.toml project for Build & Run.`	20.9
`Hint: 'gcc' was not found in this environment.`	20.9
`No code intelligence` / `Remote is not glibc Linux x64`	20.10
`WSL distro uses musl (Alpine) — only glibc is supported`	20.10
`WSL distro is not x86_64 — only Linux x64 is supported`	20.10
`Not available here` / `A Windows host is not an execution target ...`	20.10
`ECU build/run over SSH is not supported yet ...`	20.10
`PAIO is up to date.` / `The update check did not complete.`	20.12

AI Workflow

Message	Entry
`Already running`	20.14
`Nothing to run` / `No nodes`	20.14
`Node '<name>' input 'in' is required but not connected`	20.14
`The workflow graph contains a cycle — execution order cannot be determined`	20.14
`No cached outputs for this upstream node ...`	20.14
`Node type '<type>' is not implemented yet`	20.14
`stopped by the provider's usage limit ...`	20.14
`stopped because the AI provider account is not logged in ...`	20.14
`No PRTE vECU image on this machine ...`	20.14
`PAIO is installing an update; nothing new can start.`	20.14

Search

Message	Entry
`Search cancelled: there are unsaved changes. Save them, or press Refresh to choose again.`	20.15
`The result set only contains a subset of all matches; refine the search.`	20.15
`<n> files were not searched (too large, binary or unreadable).`	20.15

20.14 AI Workflow

Clicking the Workflow icon does nothing

The rail icon opens the editor of the window's project. With no project open there is nothing to open: open a project first (21.3).

Run Workflow refuses to start

Read the toast.

- `Already running`: only one run per project; stop it with `Cancel Run`.
- `Nothing to run` with the line `No nodes`: add nodes.
- An expired plan refuses the run (`PLAN_EXPIRED`).
- While PAIO installs an update, nothing new starts: `PAIO is installing an update; nothing new can start.`

A node with an unconnected input fails with `Node '<name>' input 'in' is required but not connected`, and a loop that is not a feedback edge wit `execution order cannot be determined` (21.5.1).

A run stopped at the provider's usage limit

The `Finished:` line of the run log ends with `stopped by the provider's usage limit — <reason>`. The AI provider's quota for the account is used up is available again, run again; **Run from Here** on the stopped node takes the upstream values from the recorded run (21.8.1).

Run from Here fails with No cached outputs

An upstream node has never produced recorded outputs. Run the workflow (or that node) once, then use **Run from Here** again (21.8.1).

A node fails with Node type ... is not implemented yet

AA Design and **Condition Branch** do not run in this version. Disable them or remove them (21.12).

The AI nodes cannot sign in

The `Finished:` line of the run log ends with `stopped because the AI provider account is not logged in — <reason>`. Open the avatar > **AI Accou** `connected`, use **Connect** or **Paste Token** (21.11). Then run again.

The build or run nodes find no vECU image

The inspector's PRTE vECU image field reads `No PRTE vECU image on this machine. ...`. Start Docker and pull a PRTE vECU image; PAIO pulls nothing i

My files changed after I cancelled a run

`Cancel Run` in the toolbar rolls back every file the run wrote (the toast `Workflow cancelled` shows the line `Files rolled back`). Closing the tab with without rolling back (21.8.6).

Last Run is empty after I reopened the editor

LAST RUN shows the current session only. Read `doc/report/` and `.paio/workflow-runs/<runId>/` for earlier runs (21.9).

20.15 Search

Search does not find a value I just typed

The search reads the saved files on disk only (5.8). When tabs have unsaved changes, the `Search` dialog asks first: **Save All & Search** saves them and sear disk. If you chose **Cancel**, the message line reads `Search cancelled: there are unsaved changes. Save them, or press Refresh to choose agai` be asked again.

Nothing is found in `export`, `import`, or `workflow`

This is intended. Only the `design`, `doc`, and `code` folders are searched.

The result set only contains a subset of all matches

The search stops at 20,000 matches, and the message line reads `The result set only contains a subset of all matches; refine the search.` `Case` or `Match Whole Word`. The filter chips only narrow the results already found; they do not bring back the matches beyond the limit.

Some files were not searched

`<n> files were not searched (too large, binary or unreadable).` means that PAIO skipped files larger than 5 MB, binary files, and files it could n

Clicking a search result opened a read-only `*.io.model` instead of an editor

This is intended. PAIO stores and exports some values that no editor field shows, for example the resource-consumption name of an SW-C implementatio read-only, with the matched text selected (5.8.4).

Tutorials

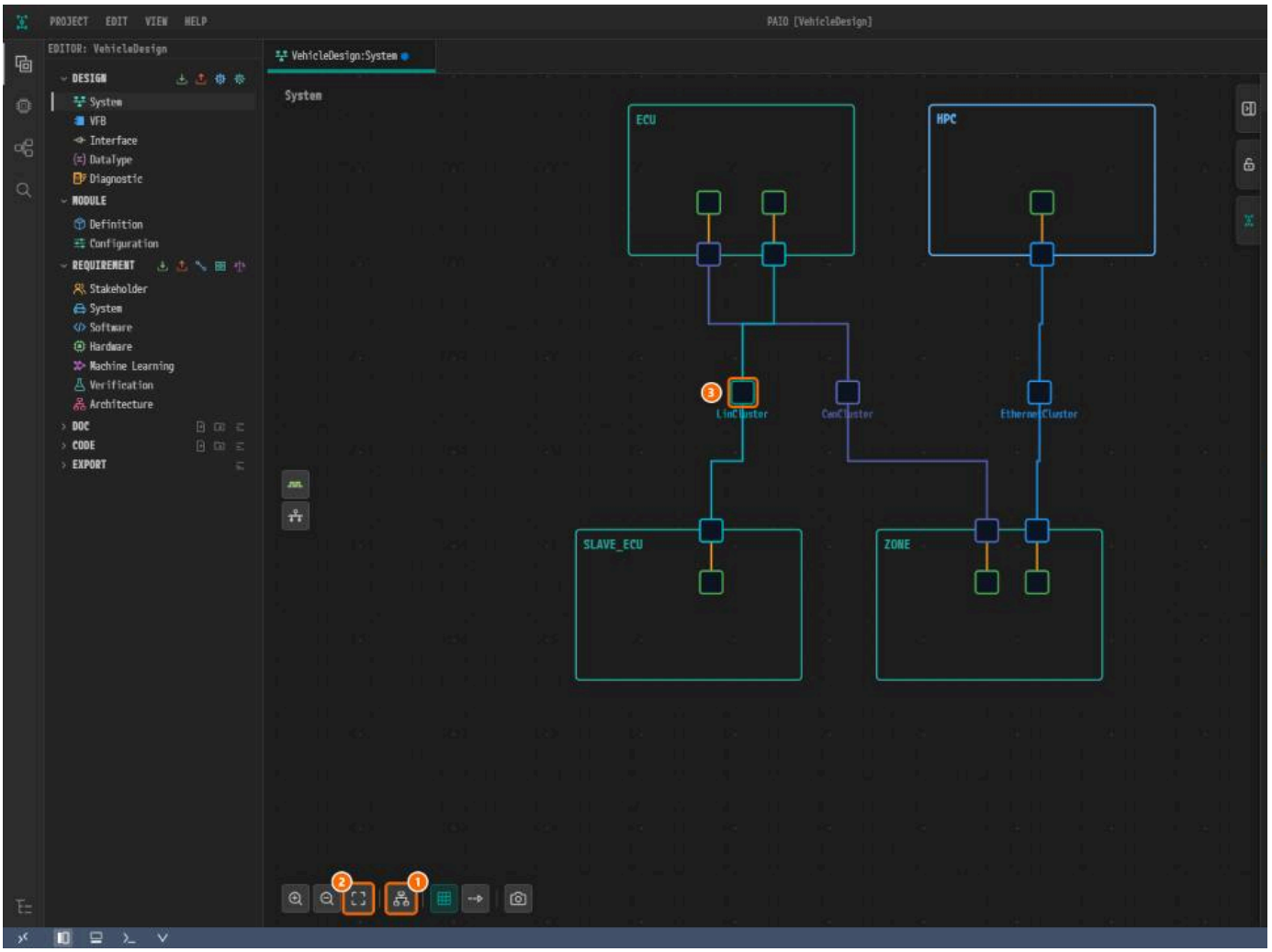
Each tutorial is an ordered series of chapters that builds a single model from start to finish. Unlike the editor reference, a tutorial follows one worked exa editor.

Tutorial	What you build
AUTOSAR System	The vehicle network `VehicleDesign` — three ECU nodes and one HPC joined by CAN, Ethernet (with VLAN and SOME/IP SD multicast) and LIN — from a PAIO wi export
AUTOSAR CP	Two software components on the node `ECU` of the project `VehicleDesign` from the AUTOSAR System tutorial, the BSW configuration with its CAN stack, thr
AUTOSAR AP	Three Adaptive Applications on the node `HPC` of the project `VehicleDesign` from the AUTOSAR CP tutorial — the SOME/IP service `Sensor`, the PARA platform and DLT — through to validation and code generation with PARA
PNC Design	A zonal vehicle with three Partial Network Clusters, from the topology in the System editor through validation to the ARXML export

The editor-by-editor reference is in Editors, and import, export and validation are covered in Workflow.

AUTOSAR System Tutorial

This tutorial builds the vehicle network `VehicleDesign` in PAIO: three ECU nodes and one HPC node joined by CAN, Ethernet and LIN, from an empty PAIO v
three parts on one project: AUTOSAR System → AUTOSAR CP → AUTOSAR AP.



The network you build: the ECU nodes `ECU`, `ZONE` and `SLAVE_ECU`, the HPC node `HPC`, and the clusters `CanCluster`, `EthernetCluster` and `LinCluster`.

What You Build

Each bus joins two nodes. The CP part later puts software components on `ECU`, and the AP part puts adaptive applications on `HPC`.

Cluster	Nodes	Main values
`CanCluster`	`ZONE`, `ECU`	Bus-off recovery L1 `100` ms, L2 `200` ms
`EthernetCluster`	`ZONE`, `HPC`	VLAN ID `13`, `HPC` address `192.168.0.11`
`LinCluster`	`ECU` (master), `SLAVE_ECU`	`LIN 2.0`, time base `1` ms

What You Need

- PAIO, installed and able to run.
- No prepared project. Chapter 1 starts from a PAIO window with no project open.
- A screen of 1920×1080 or larger.
- A rough idea of buses, controllers, connectors and clusters. Each chapter shows what to enter on which screen.

Chapters

Chapter	What you do	Editor used
1. Project and Nodes	Create the project `VehicleDesign` and place the four nodes	System editor
2. ECU Properties	Set the sleep and wake-up properties of the ECU nodes and the system version	System editor, SYSTEM panel
3. CAN Configuration	Join `ECU` and `ZONE` with `CanCluster` and enter the bus-off recovery values	System editor, communication cluster dialog
4. ETHERNET Configuration	Join `ZONE` and `HPC` with `EthernetCluster` and enter the addresses	System editor, communication cluster dialog
5. LIN Configuration	Join `ECU` and `SLAVE_ECU` with `LinCluster`	System editor
6. Validation	Fix three planted defects that the VALIDATION panel reports	VALIDATION panel, System editor
7. ARXML Export	Export the design and check the result in the XML editor	EDITOR view, XML editor
Appendix	Design checklist, value tables, file layout, common problems	—

How to Read This Tutorial

- **Follow the chapters in order.** Each one starts from the result of the previous one.
- **Repetitive work is shown once.** One item is shown as steps; the rest are given in value tables that you enter the same way.
- UI labels are written in **bold** as the screen shows them, and model element names and values in `code`.

Next Step

Start with Project and Nodes.

Project and Nodes

What You Will Do

In this chapter you create the project `VehicleDesign` and place four nodes in a 2x2 grid in the System editor: the ECU nodes `ECU`, `ZONE`, `SLAVE_ECU` and chapter 3 on.

Before You Begin

- Start from a PAIO window with no project open: the **EDITOR** view shows `No project open`. If PAIO opens an earlier project, choose **PROJECT** > **New Wi**
- Later chapters use these names as they are, so do not change them.

Element	Value to type	Where
Project name	`VehicleDesign`	**New Project** dialog
ECU node names (three)	`ECU`, `ZONE`, `SLAVE_ECU`	System editor property panel **NAME**
HPC node name	`HPC`	System editor property panel **NAME**

Creating a New Project

1. Click **PROJECT** in the menu bar, then **New Project...**.

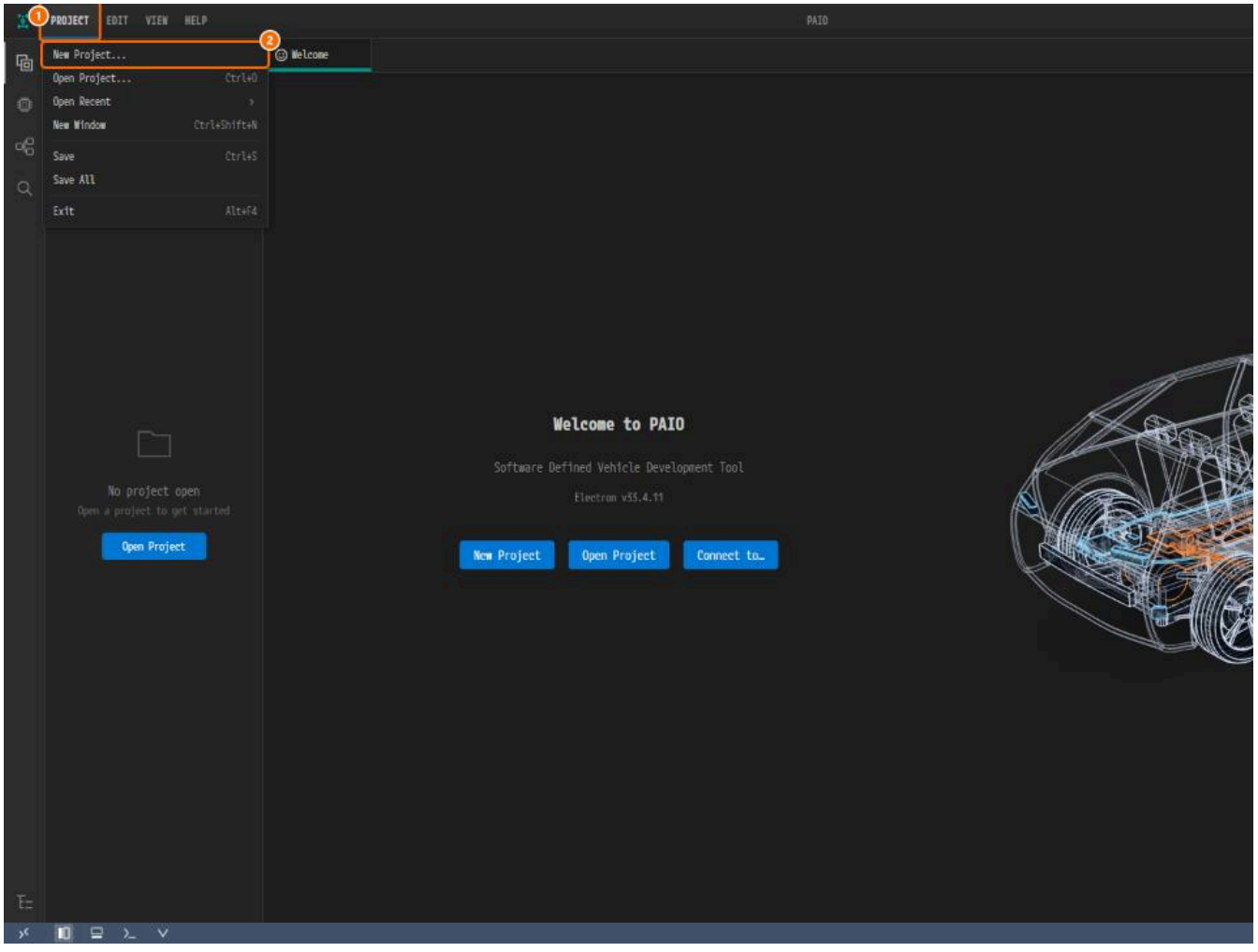


*Figure 1-1. ① **PROJECT**, ② **New Project...**.*

2. Type `VehicleDesign` in **Name**. In **Location**, type or **Browse...** to the folder that will hold the project. Click **Create**.

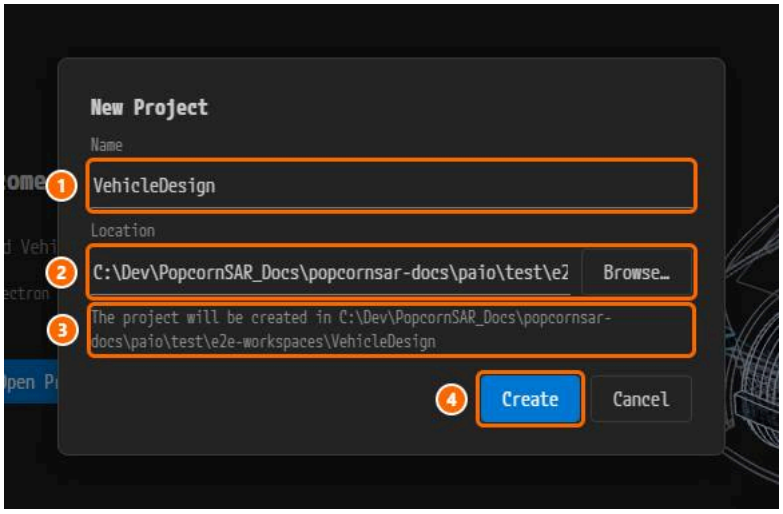


*Figure 1-2. ① **Name**, ② **Location** and **Browse...**, ③ the folder that will be created, ④ **Create**. The folder in the figure is only an example.*

3. Confirm that the title bar reads `PAIO [VehicleDesign]` and that **System** is under **DESIGN** in the **EDITOR** view.

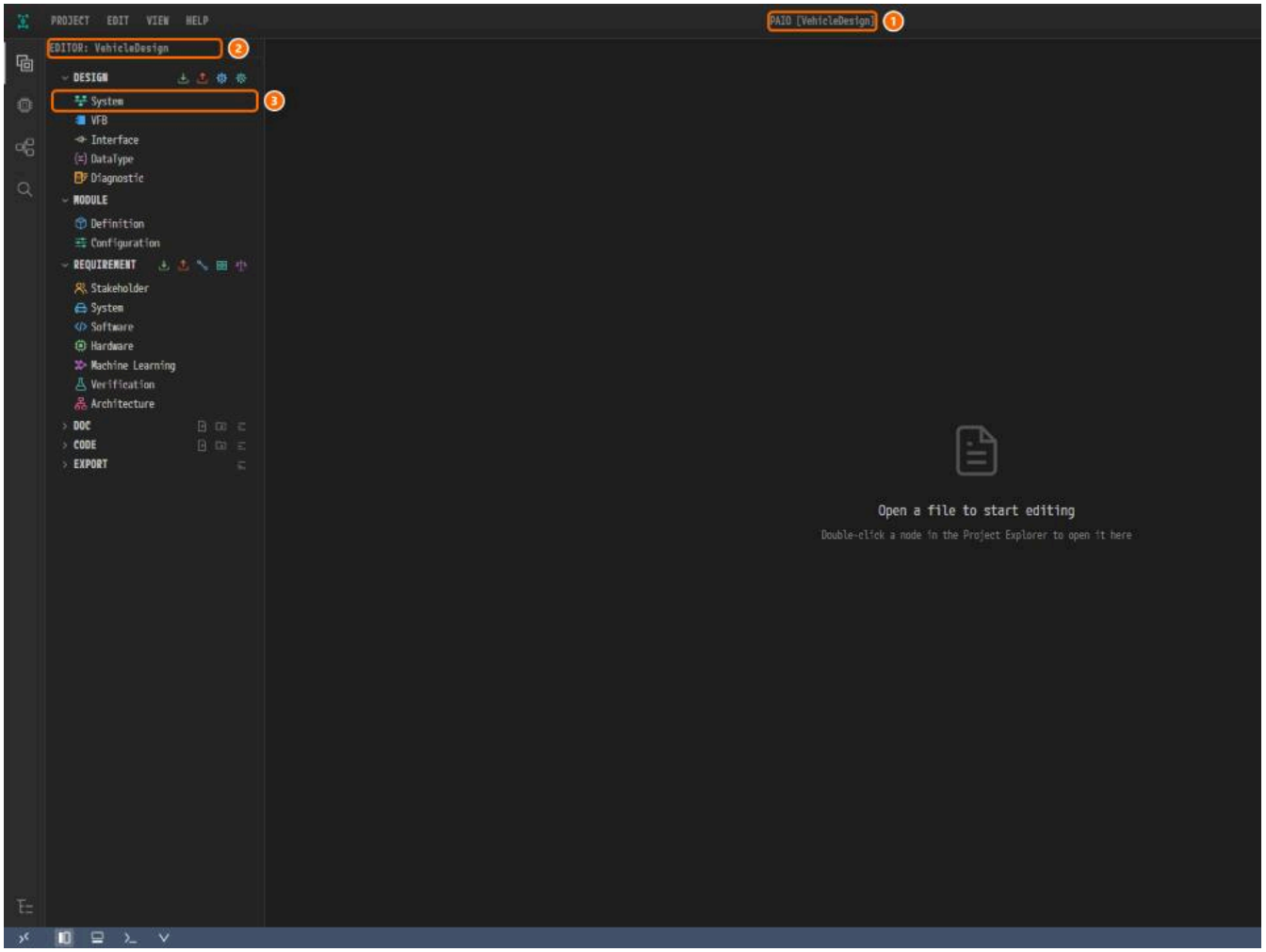


*Figure 1-3. ① The title bar, ② the view header `EDITOR: VehicleDesign`, ③ **System**.*

Creating the First ECU Node

1. Click **System** under **DESIGN**. The tab `VehicleDesign:System` opens with an empty canvas.
2. Right-click an empty spot at the **top left** of the canvas and click **Add ECU**.

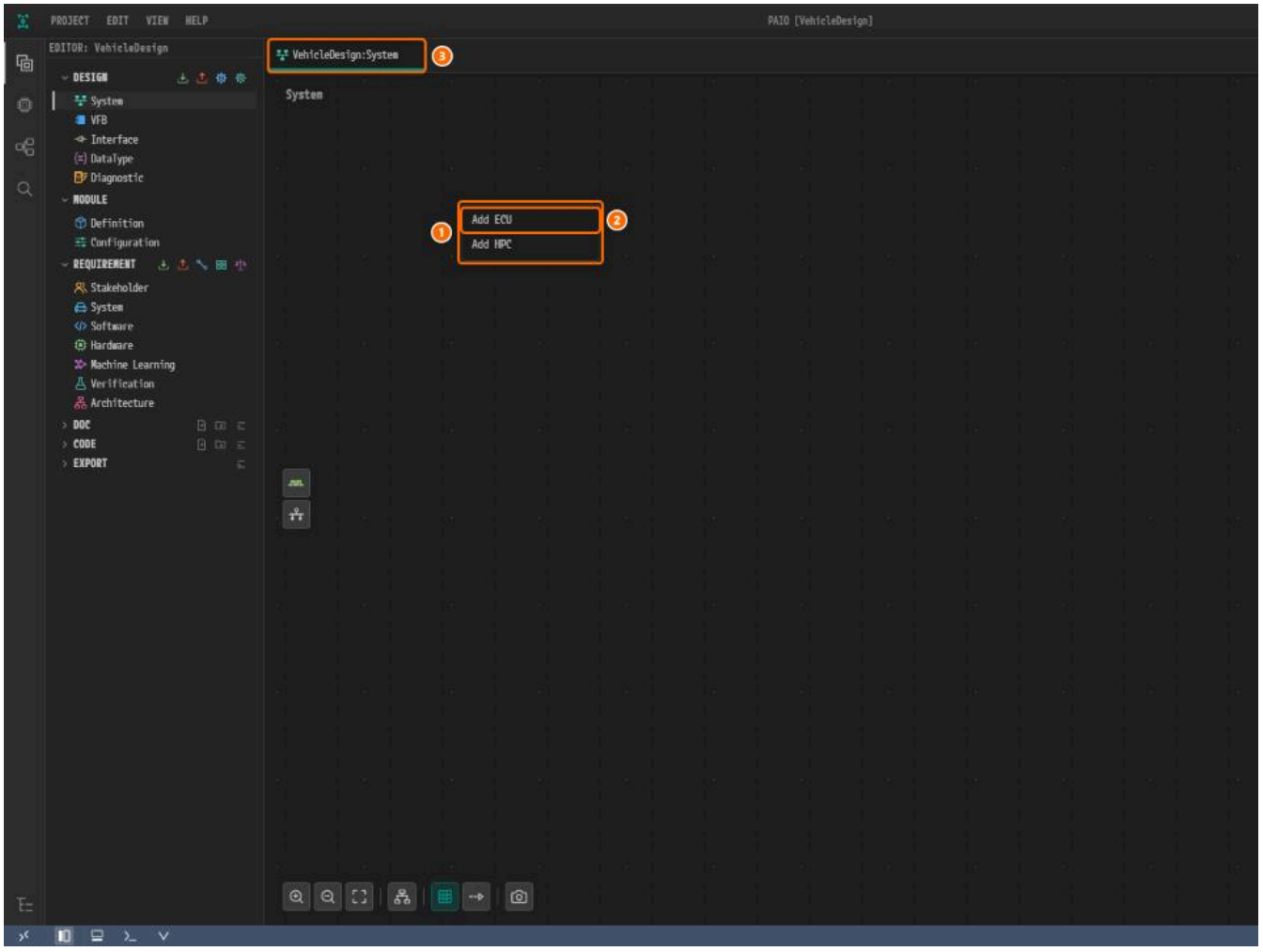


*Figure 1-4. ① The canvas menu, ② **Add ECU**, ③ the tab `VehicleDesign:System`.*

3. The node `ECU_0` appears. Click its header. A property panel titled **ECU** opens.

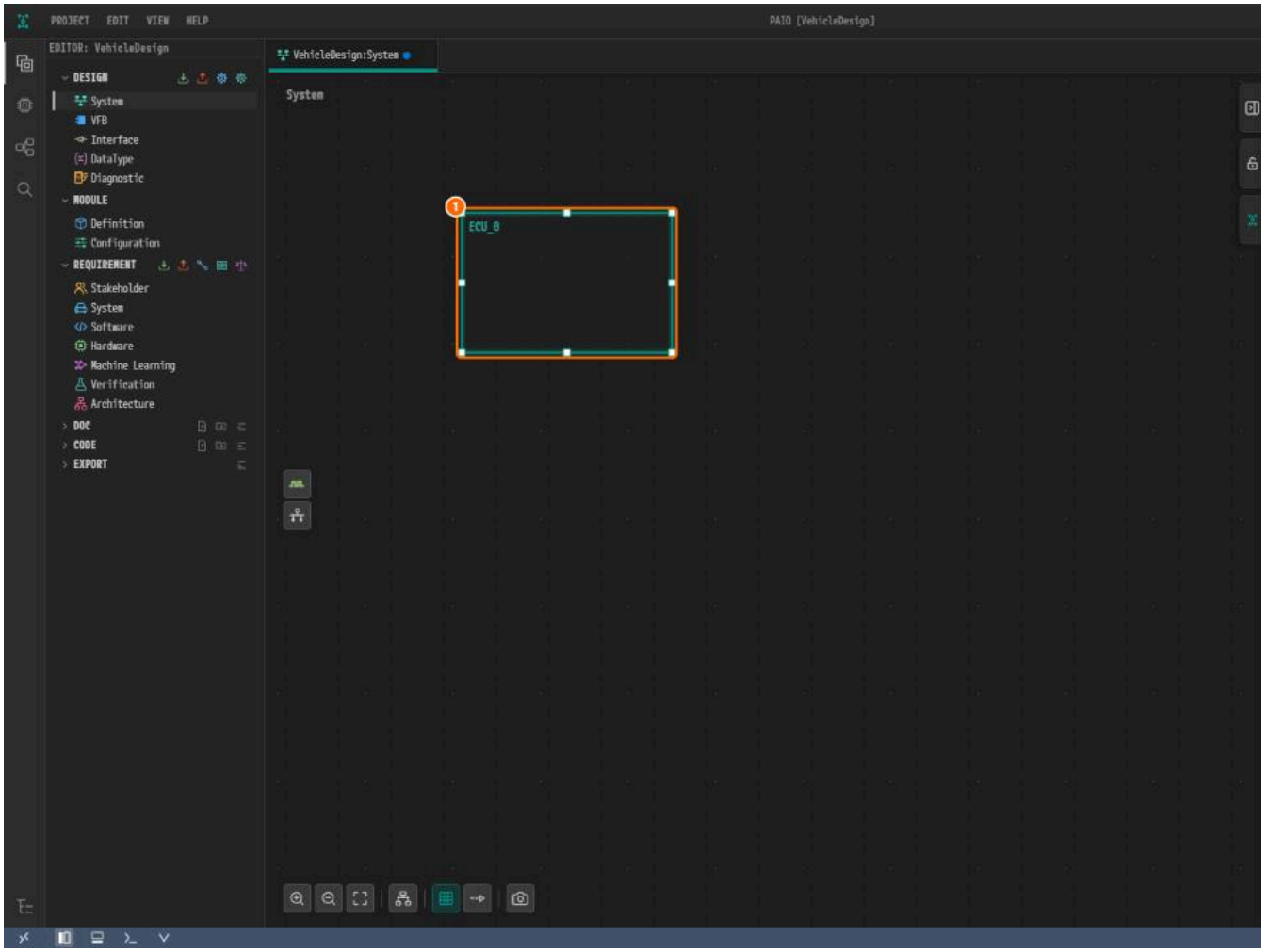


*Figure 1-5. ① The node `ECU_0`, ② **NAME**.*

4. Replace `ECU_0` in **NAME** with `ECU` and press `Tab`. The node header changes to `ECU`.

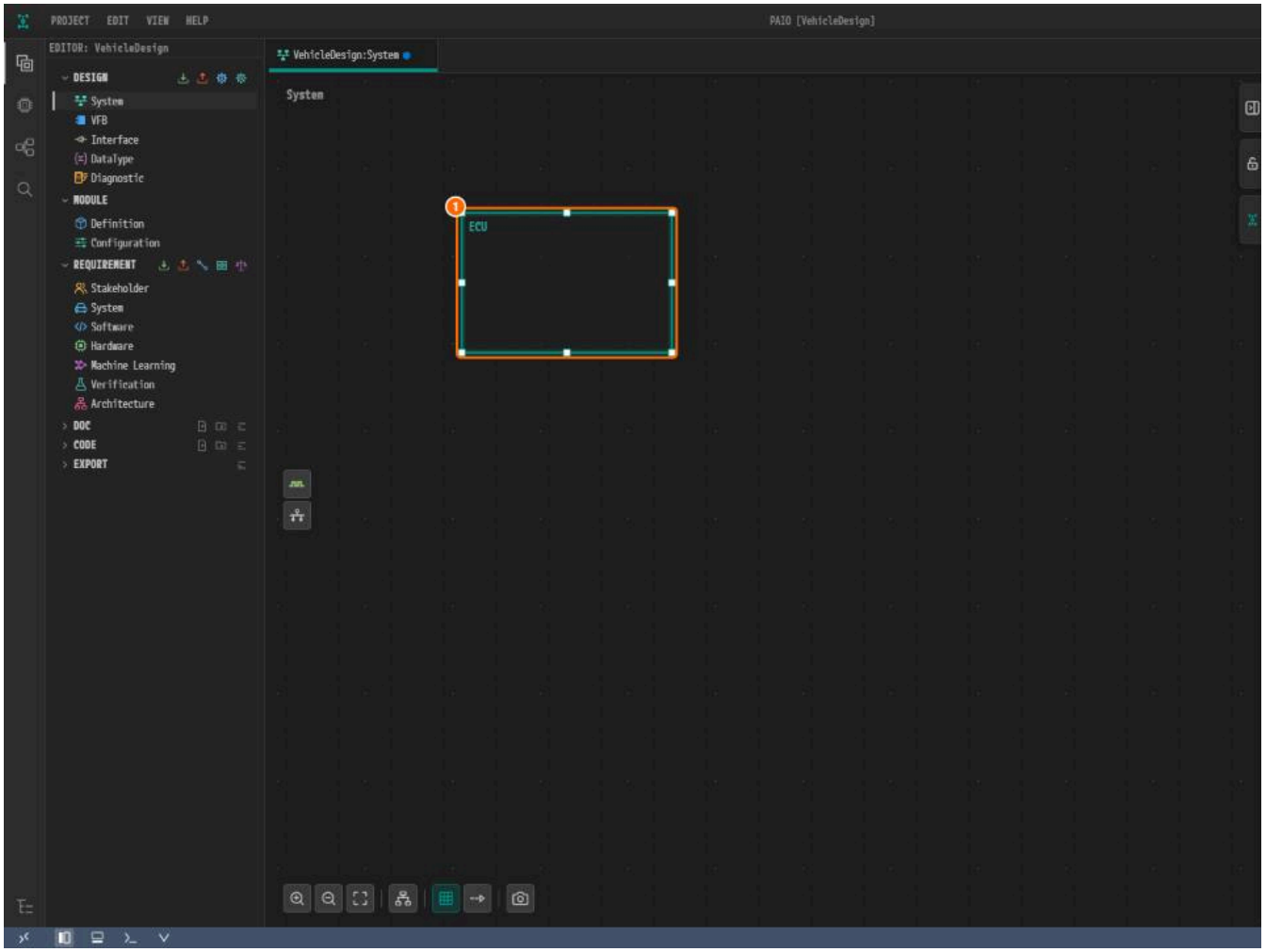


*Figure 1-6. ① The node `ECU`, ② **NAME**.*

Creating the Remaining Nodes

1. Right-click at the **top right** and click **Add ECU**. The new node is again named `ECU_0`.

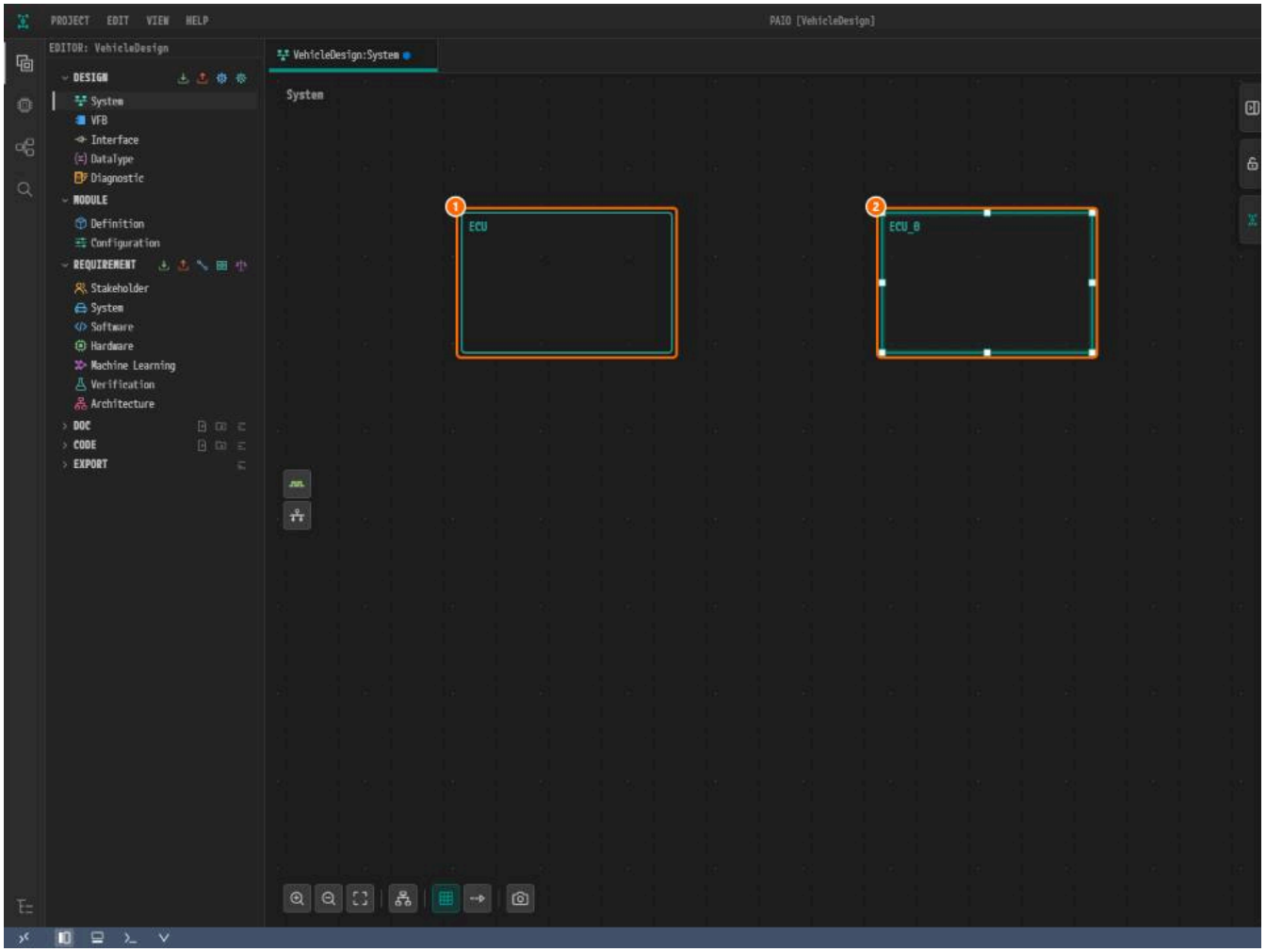


*Figure 1-7. ① `ECU`, ② the new node `ECU_0`, ③ **NAME**.*

2. Type `ZONE` in **NAME** and press `Tab`.

3. Create `SLAVE_ECU` the same way at the **bottom left**.

4. Right-click at the **bottom right** and click **Add HPC**.

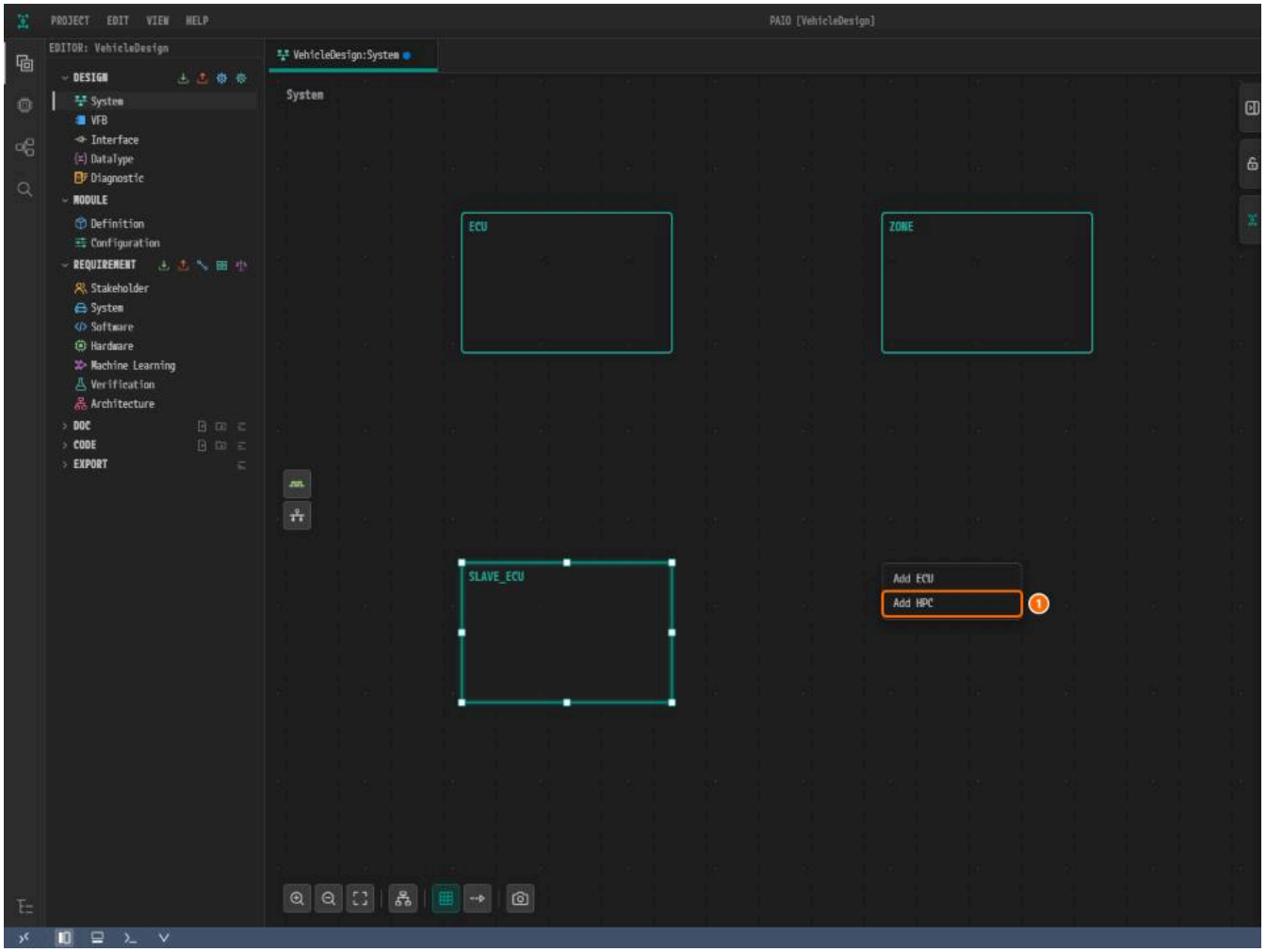


*Figure 1-8. ① **Add HPC.***

5. Click the header of the new node `HPC_0`. A panel titled **HPC** opens. Type `HPC` in **NAME** and press `Tab`.

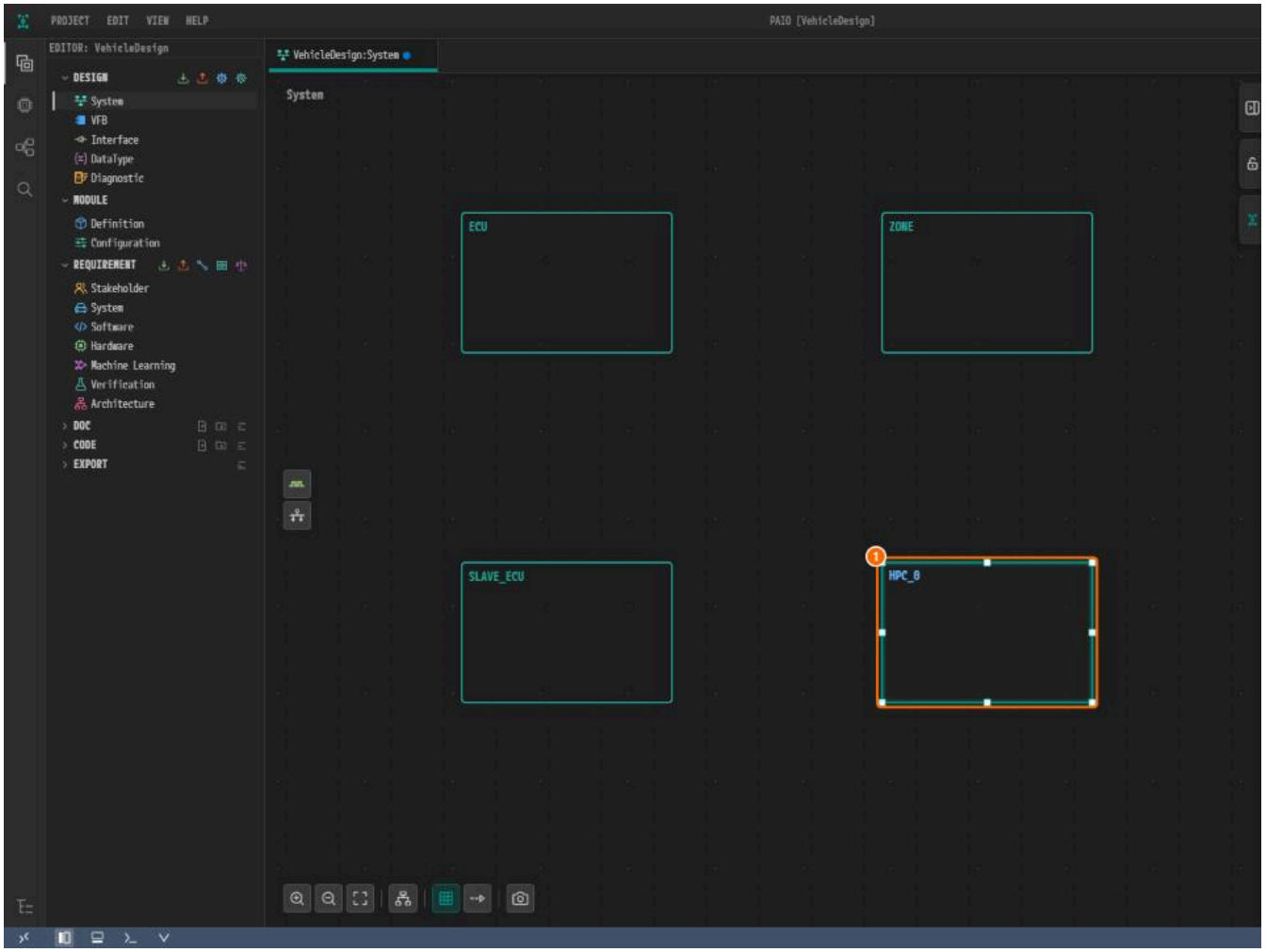


*Figure 1-9. ① The node `HPC_0`, ② the panel title **HPC**, ③ **NAME**. The HPC panel has no **POWER MANAGEMENT** section.*

Add the nodes in the order of this table.

Order	Right-click position	Menu item	Node created	Type into NAME
1	Top left of the canvas	**Add ECU**	`ECU_0`	`ECU`
2	Top right of the canvas	**Add ECU**	`ECU_0`	`ZONE`
3	Bottom left of the canvas	**Add ECU**	`ECU_0`	`SLAVE_ECU`
4	Bottom right of the canvas	**Add HPC**	`HPC_0`	`HPC`

> **Caution.** The ECU nodes are exported in the order you add them, so the line numbers quoted in ARXML Export match only in this order.

Saving

1. Click the empty canvas between the four nodes to clear the selection.
2. Press `Ctrl+S`. The modified mark on the tab disappears.

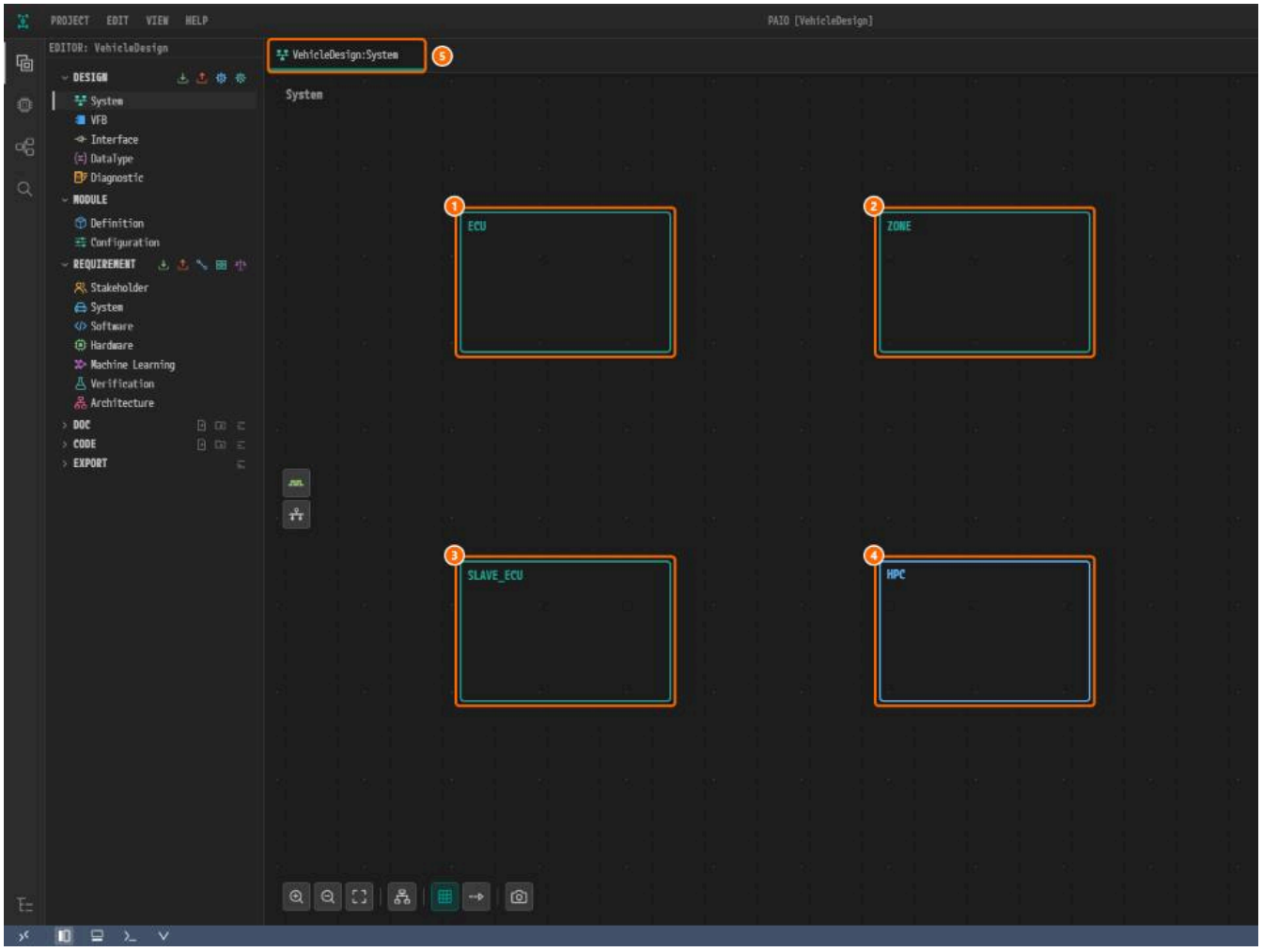


Figure 1-10. ① `ECU`, ② `ZONE`, ③ `SLAVE_ECU`, ④ `HPC`, ⑤ the tab without the modified mark.

Viewing the Validation Result

1. Click the **V** icon (`Toggle Validation`) in the footer. The **VALIDATION** panel shows `No validation issues.`.

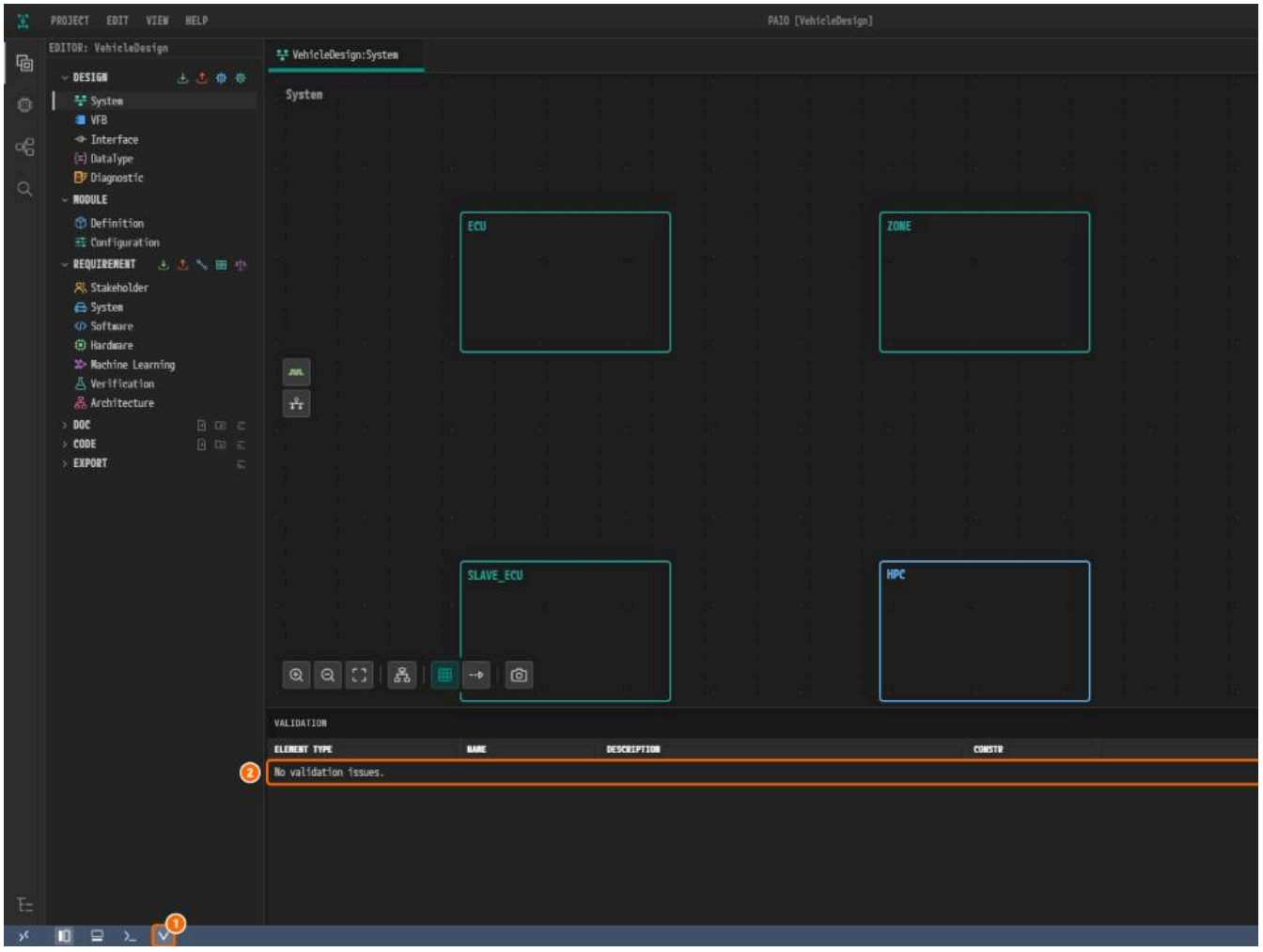


*Figure 1-11. ① The **V** icon, ② `No validation issues.`.*

Why Is It Designed This Way?

- **The names carry the roles.** `ECU` has CAN and a LIN master, `ZONE` has CAN and Ethernet, `SLAVE_ECU` is the LIN slave, and `HPC` is connected only to Eth
- **Only `HPC` is an HPC node.** It is stored as a machine design and has no **POWER MANAGEMENT** section, so the next chapter turns power management o
- **You place the nodes yourself.** A node appears where you right-click, so the 2x2 grid keeps your screen like the figures of later chapters.

Checklist

- [ ] The title bar reads `PAIO [VehicleDesign]`.
- [ ] The tab `VehicleDesign:System` holds `ECU`, `ZONE`, `SLAVE_ECU`, `HPC` in a 2x2 grid.
- [ ] Selecting `HPC` shows the panel title **HPC**; the other three show **ECU**.
- [ ] The tab `VehicleDesign:System` carries no modified mark.
- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In ECU Properties you turn on power management of the three ECU nodes and change the system version.

ECU Properties

What You Will Do

In this chapter you turn on sleep and wake-up over the bus for the three ECU nodes and change the system version to `1.0.0;POPCORNSAR`.

Before You Begin

- Start after finishing Project and Nodes. The tab `VehicleDesign:System` holds the four saved nodes.
- To match the figures, close the **VALIDATION** panel with the **V** icon in the footer.

Where	Item	Value
System editor, nodes `ECU`, `ZONE`, `SLAVE_ECU`	**POWER MANAGEMENT > SLEEP MODE SUPPORTED**	On
System editor, nodes `ECU`, `ZONE`, `SLAVE_ECU`	**POWER MANAGEMENT > WAKE-UP OVER BUS**	On
System editor, node `HPC`	—	No change (check only)
System editor, SYSTEM panel	**VERSION**	`1.0.0;POPCORNSAR`

Turning On Power Management of the ECU Nodes

1. Click the header of the node `ECU`. The panel shows the **POWER MANAGEMENT** section with both buttons off.

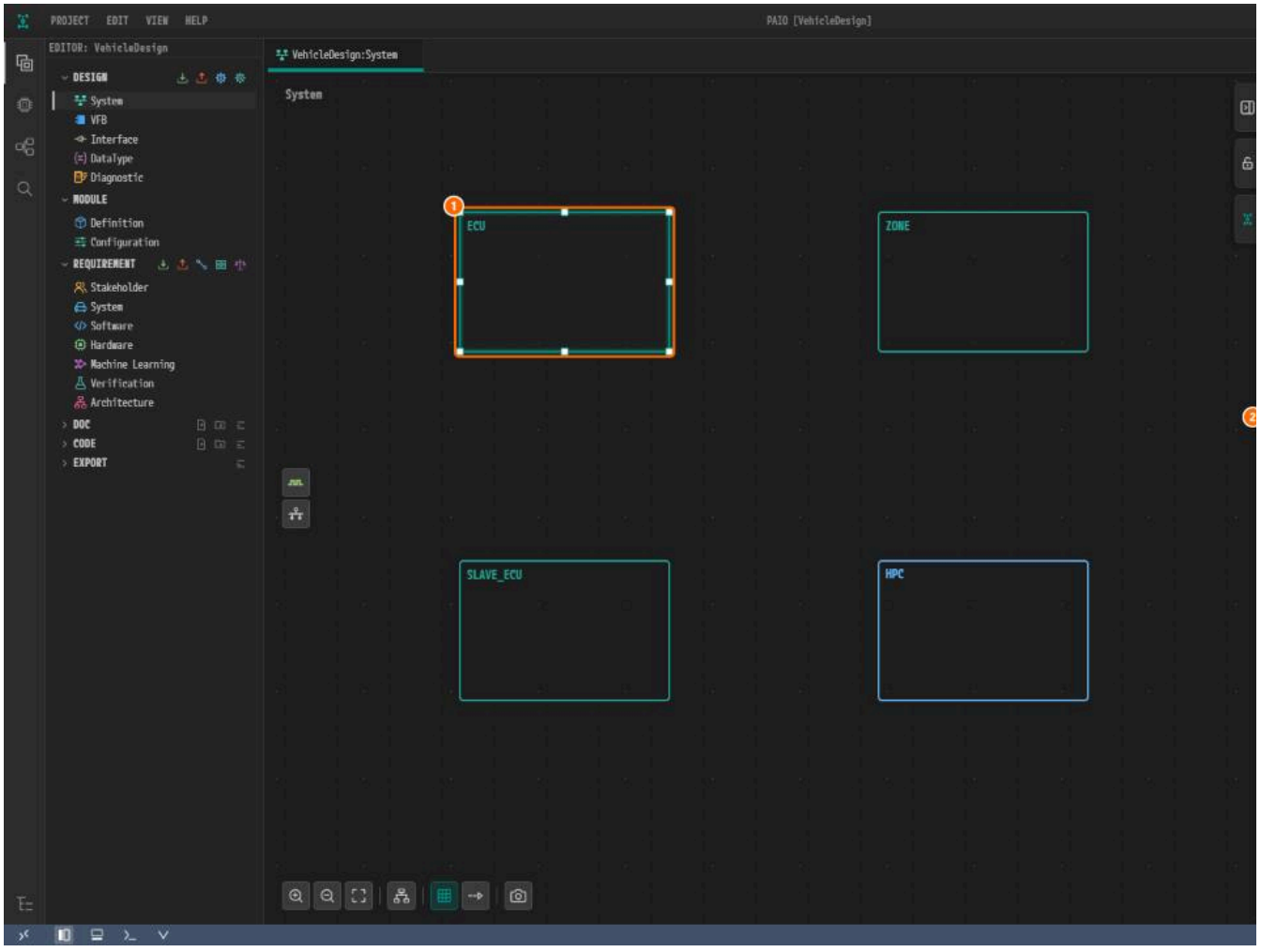


*Figure 2-1. ① The node `ECU`, ② **POWER MANAGEMENT**, ③ **SLEEP MODE SUPPORTED**, ④ **WAKE-UP OVER BUS**.*

2. Click **SLEEP MODE SUPPORTED** and **WAKE-UP OVER BUS**. Both turn on (filled background).

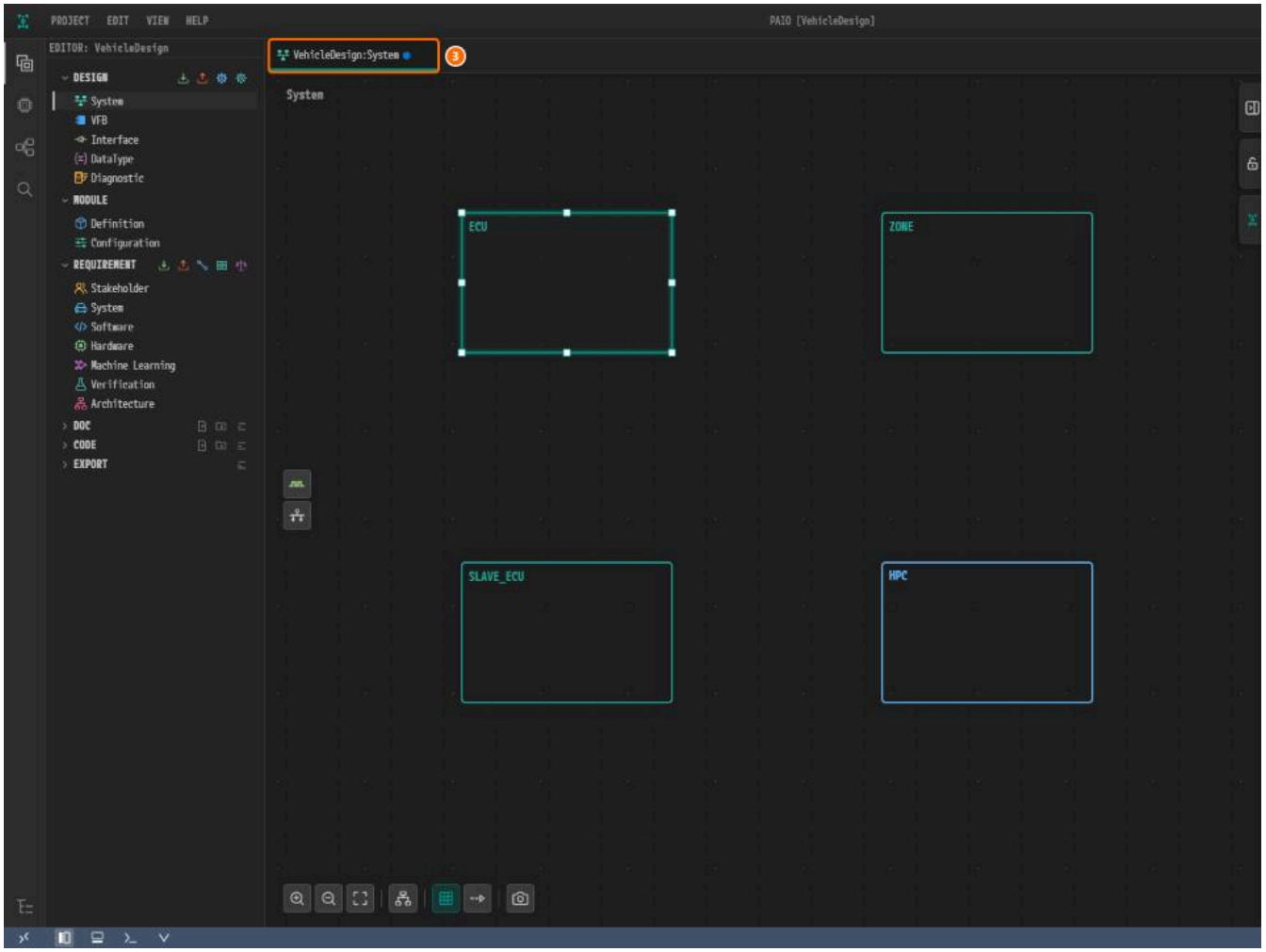


*Figure 2-2. ① **SLEEP MODE SUPPORTED**, ② **WAKE-UP OVER BUS**, ③ the tab with the modified mark.*

3. Turn on both toggles of `ZONE` and `SLAVE_ECU` the same way. All three panels are titled **ECU**, so check **NAME** to see which node you edit.

4. Click the header of `HPC`. Confirm that the panel title is **HPC** and that it has no **POWER MANAGEMENT** section. Change nothing.

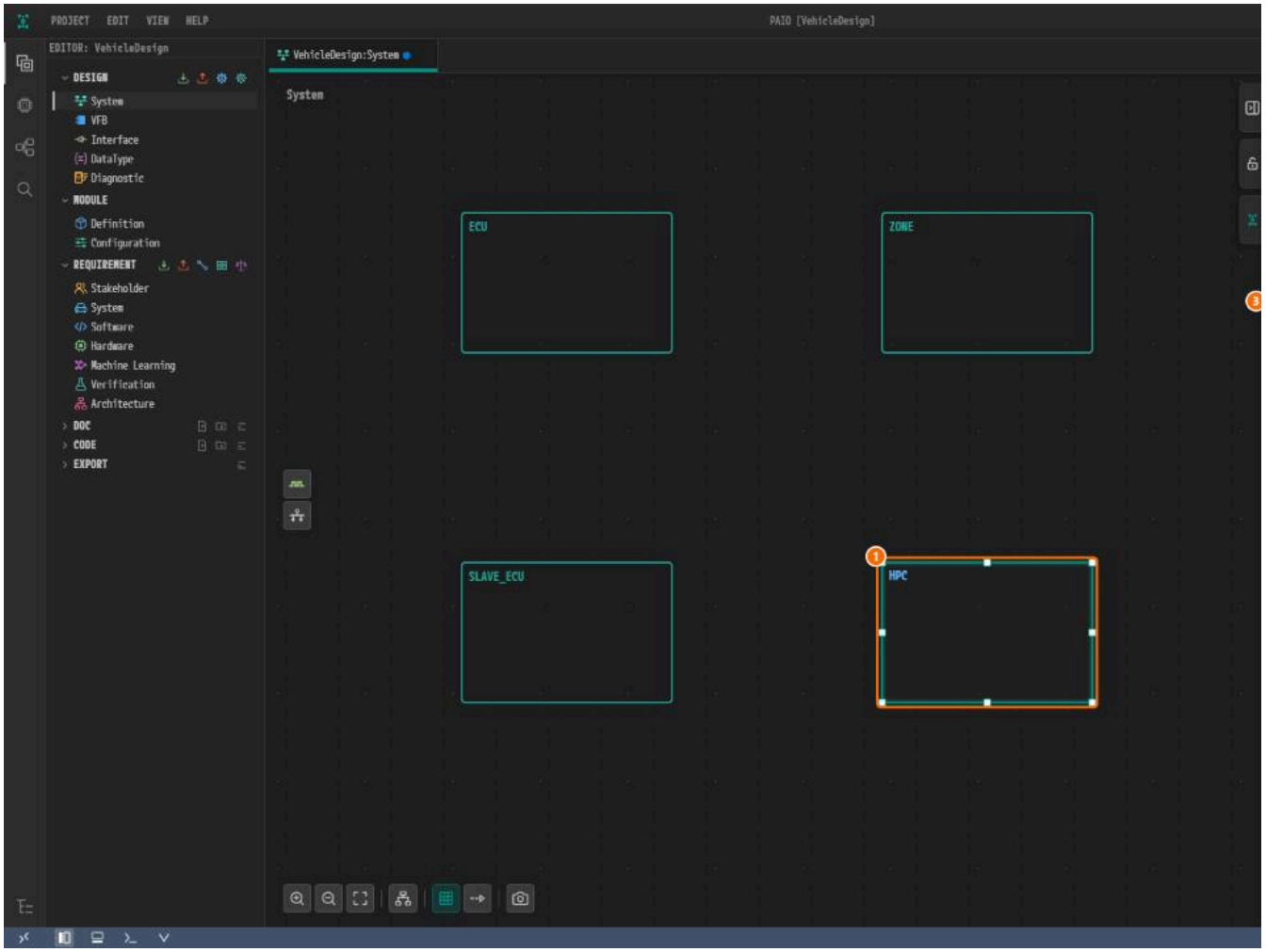


*Figure 2-3. ① The node `HPC`, ② the panel title **HPC**, ③ the panel items, without **POWER MANAGEMENT**.*

5. Click the empty canvas between the nodes and press `Ctrl+S`.

Changing the System Version

1. Click the **System** button (the PAIO logo) on the right edge of the canvas, the third of the three buttons there. The panel **SYSTEM** opens.

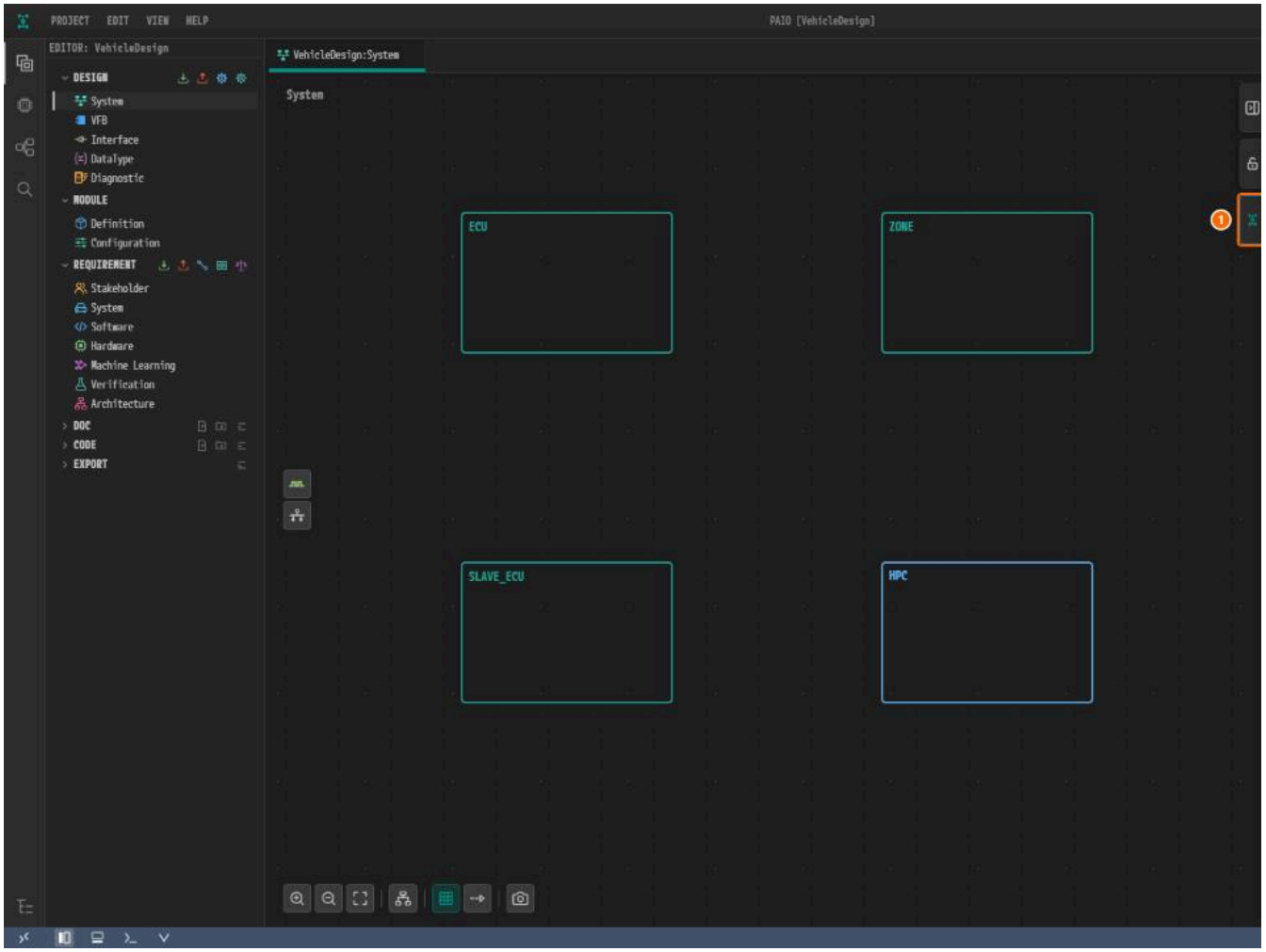


*Figure 2-4. ① The **System** button, ② the panel title **SYSTEM**, ③ **VERSION** with the default `1.0.0`.*

2. In **VERSION**, replace `1.0.0` with `1.0.0;POPCORNSAR` and press `Tab`. The value is saved as you type, so no modified mark appears and there is nothing

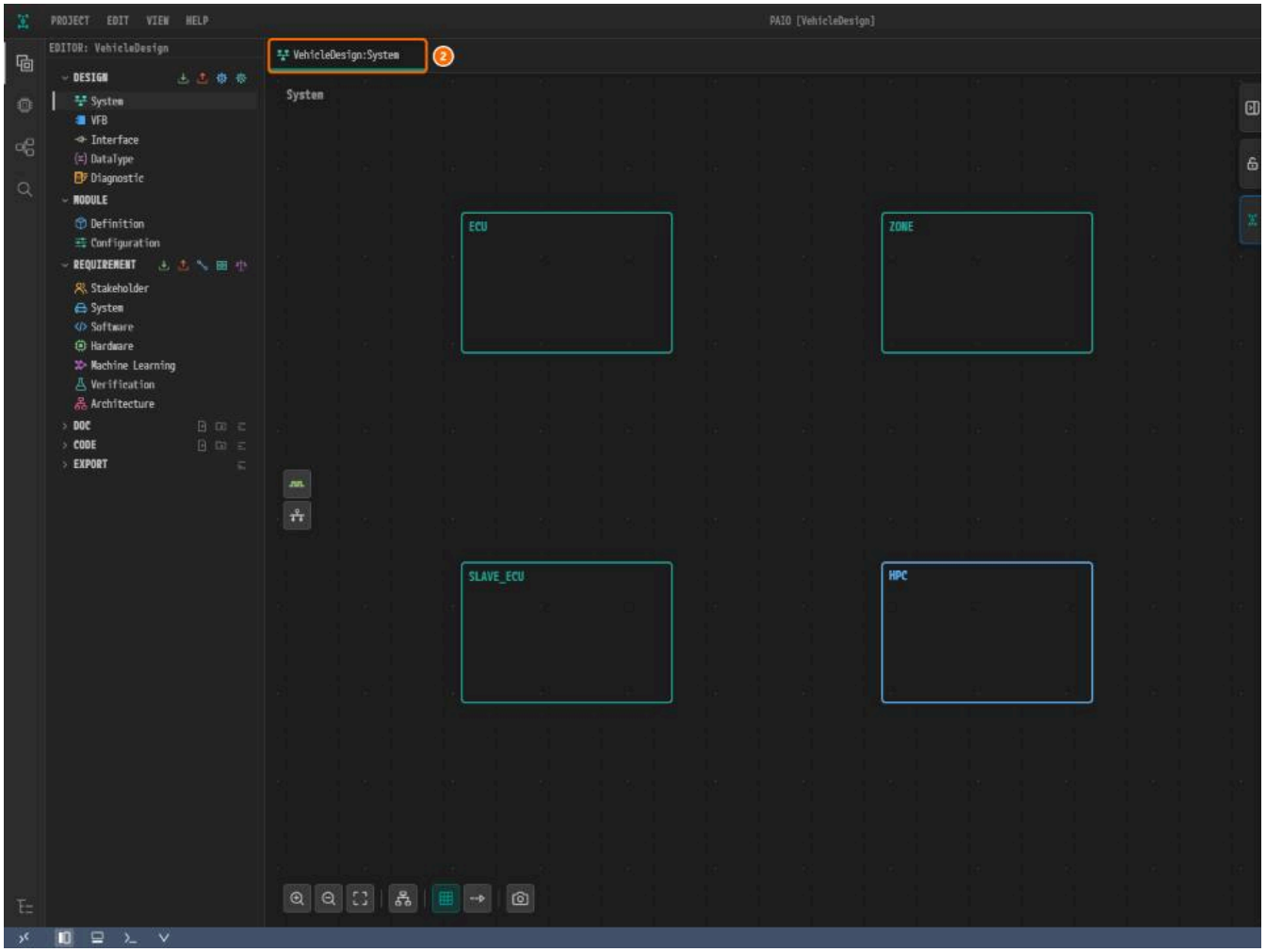


*Figure 2-5. ① **VERSION**, ② the tab without the modified mark.*

3. Confirm that **ECU MAPPING** lists `ECU`, `ZONE`, `SLAVE_ECU` and `HPC`, each with `No composition mapped`.

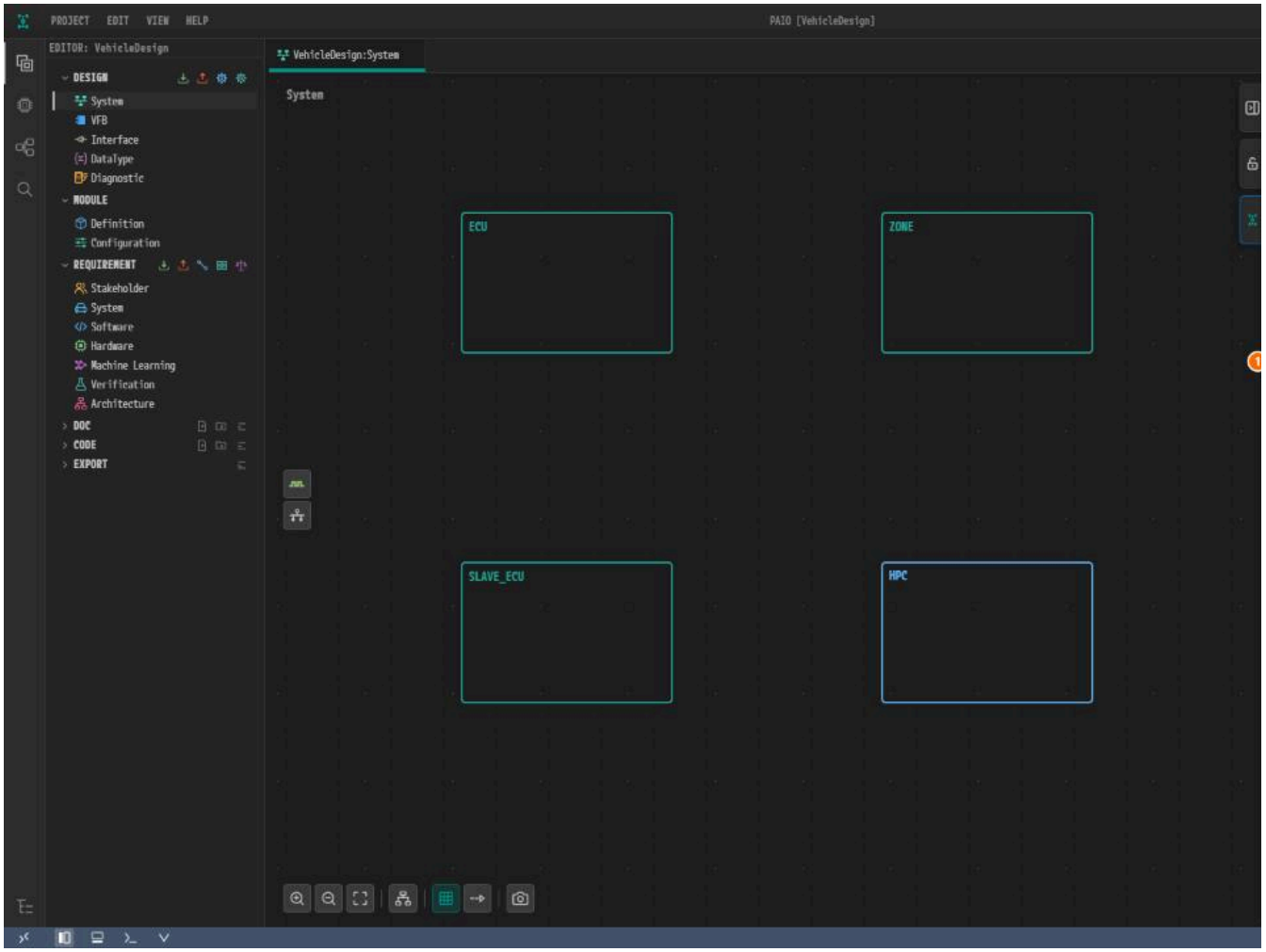


*Figure 2-6. ① The four rows of **ECU MAPPING**.*

Viewing the Validation Result

1. Click the **V** icon in the footer. The **VALIDATION** panel shows `No validation issues.`.

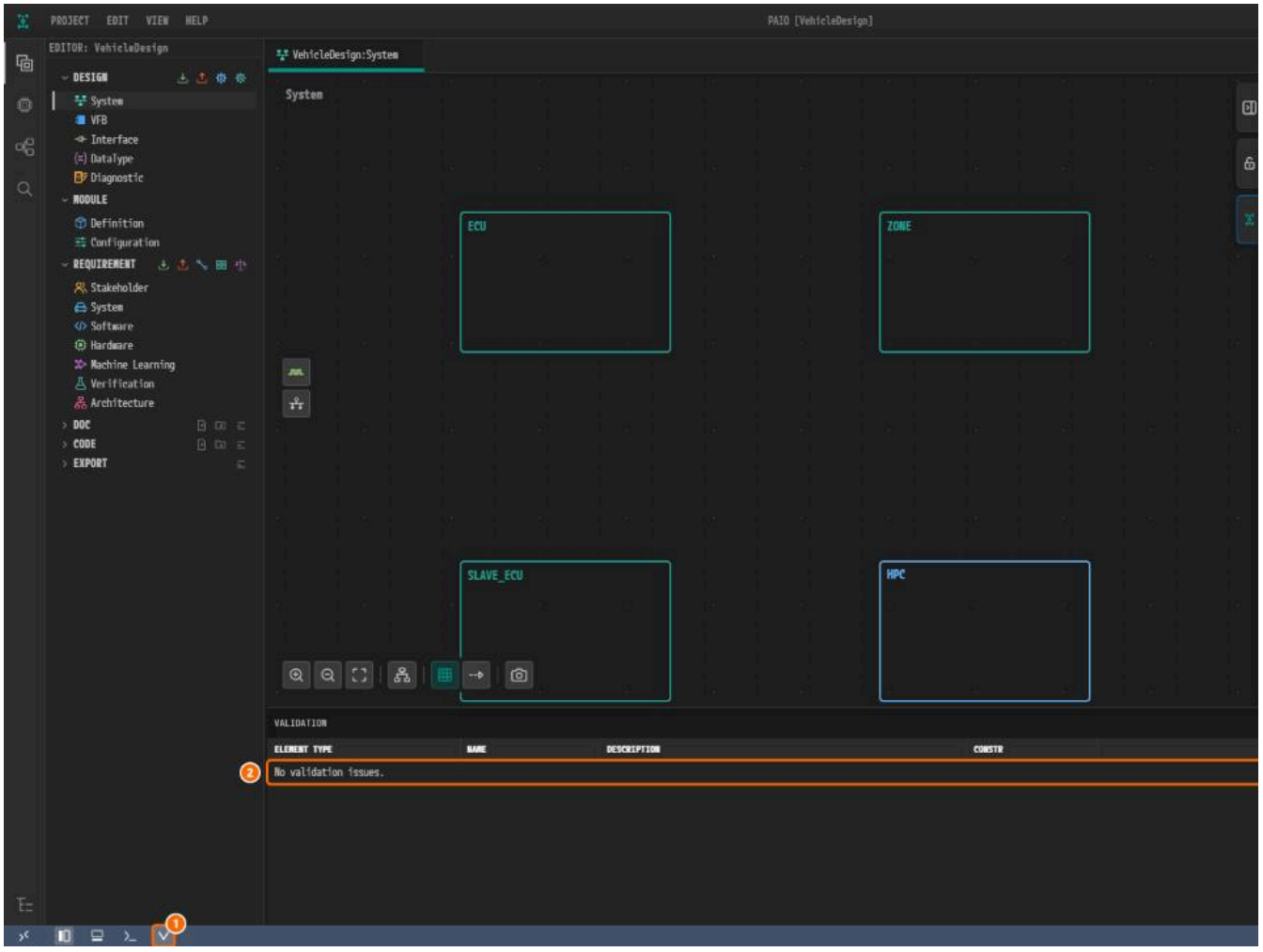


*Figure 2-7. ① The **V** icon, ② `No validation issues.`.*

Why Is It Designed This Way?

- **Sleep and wake-up on the three ECU nodes.** The reference design sets `SLEEP-MODE-SUPPORTED` and `WAKE-UP-OVER-BUS-SUPPORTED` to `true` on `ECU`
- **No power management on `HPC`.** An HPC node is a machine design, which has neither attribute.
- **The version `1.0.0;POPCORNSAR`** becomes `SYSTEM-VERSION` in the export, as in the reference design.
- **No partitions.** The reference design gives `ECU` the partitions `ECU_Partition0` and `ECU_Partition1`, but the System editor has no field for them an

Checklist

- [ ] `ECU`, `ZONE` and `SLAVE_ECU` each show **SLEEP MODE SUPPORTED** and **WAKE-UP OVER BUS** on.
- [ ] `HPC` shows the panel title **HPC** and no **POWER MANAGEMENT** section.
- [ ] **VERSION** in the SYSTEM panel is `1.0.0;POPCORNSAR`.
- [ ] The tab `VehicleDesign:System` carries no modified mark.
- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In CAN Configuration you join the nodes `ECU` and `ZONE` with the CAN cluster `CanCluster`.

CAN Configuration

What You Will Do

You join the nodes `ECU` and `ZONE` with a CAN bus: one CAN controller and one CAN connector on each node, and the CAN cluster `CanCluster` that conne
recovery values of the cluster, which the CAN state manager (CanSM) of `ECU` reads in the AUTOSAR CP Tutorial. Network management (NM) and the transp

Before You Begin

- Start after finishing ECU Properties. The four nodes have no controllers, connectors or clusters, and the **VALIDATION** panel shows `No validation is`
- Open the tab `VehicleDesign:System` (**EDITOR** > **DESIGN** > **System**). Close the **VALIDATION** panel with the **V** icon in the footer.
- A field in the property panel commits its value only when you leave it, so press the `Tab` key after each value.

Where	Item	Value
CAN CONTROLLER (node `ECU`)	**NAME**	`CanController`
CAN CONTROLLER (node `ECU`)	**SYNC JUMP WIDTH**, **PROPAGATION SEGMENT**, **TIME SEGMENT 1**, **TIME SEGMENT 2**	`0` in all four
CAN CONTROLLER (node `ZONE`)	**NAME**	`CanCommController`
CAN CONNECTOR (node `ECU`)	**NAME** / **CONTROLLER**	`CanConnector` / `CanController`
CAN CONNECTOR (node `ZONE`)	**NAME** / **CONTROLLER**	`CanConnector` / `CanCommController`
CAN CLUSTER	**NAME**	`CanCluster`
CAN CLUSTER	**CHANNEL NAME**	`PhysicalChannel` (default, leave as is)
CAN CLUSTER	**BOR TIME L1** / **BOR TIME L2**	`100` `ms` / `200` `ms`
CAN CLUSTER	**BOR COUNTER L1 TO L2** / **BOR TIME TX ENSURED**	`2` / `100` `ms`

Creating the CAN Controllers

Adding a Controller

1. Right-click an empty spot inside the node `ECU`. Do not right-click the node name or an element already placed.
2. Click **Add Controller > CAN**.

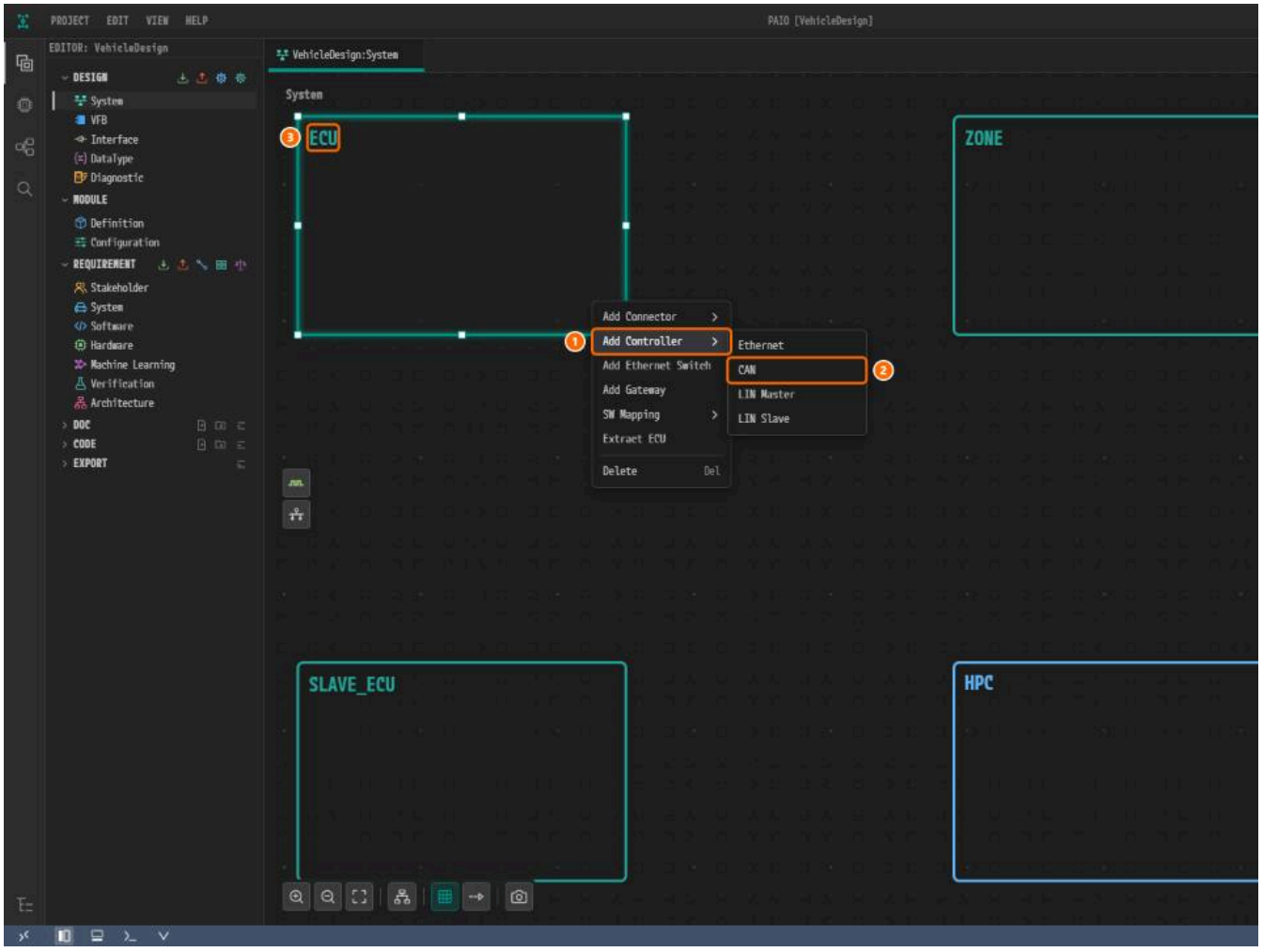


*Figure 3-1. ① **Add Controller**, ② **CAN**, ③ the node `ECU`.*

3. Click the new small square inside the node. The panel **CAN CONTROLLER** opens with **NAME** `ECU_Can_Controller_0`.

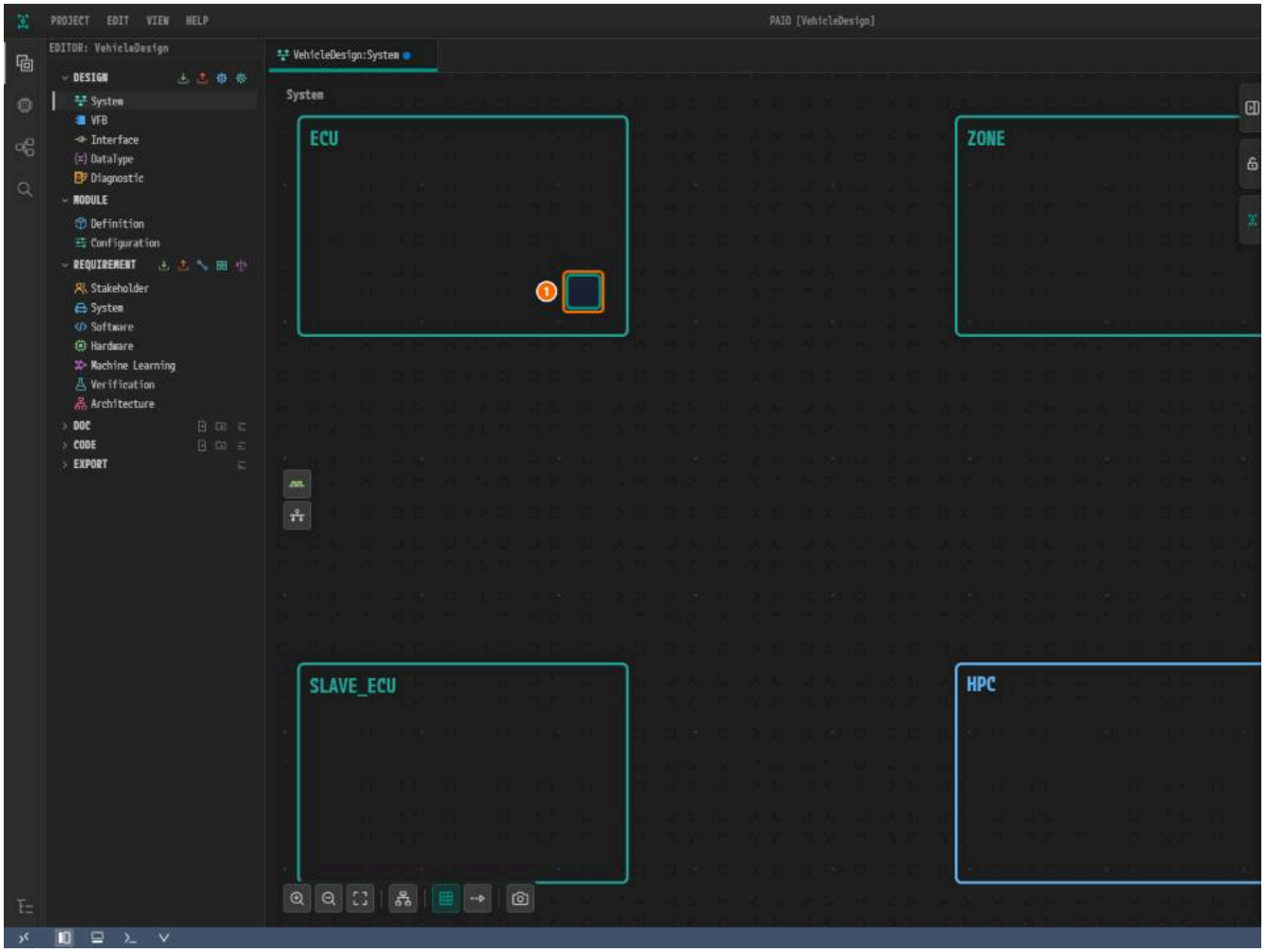


*Figure 3-2. ① The controller, ② **CAN CONTROLLER**, ③ **NAME**, ④ the **CAN / CAN FD** toggle, ⑤ **BIT TIMING CONFIGURATION**.*

Entering the Name and Bit Timing

1. Type `CanController` into **NAME**.
2. Leave the toggle at **CAN**.
3. Type `0` into **SYNC JUMP WIDTH**, **PROPAGATION SEGMENT**, **TIME SEGMENT 1** and **TIME SEGMENT 2**. Leave **BIT TIMING REQUIREMENTS** empty.

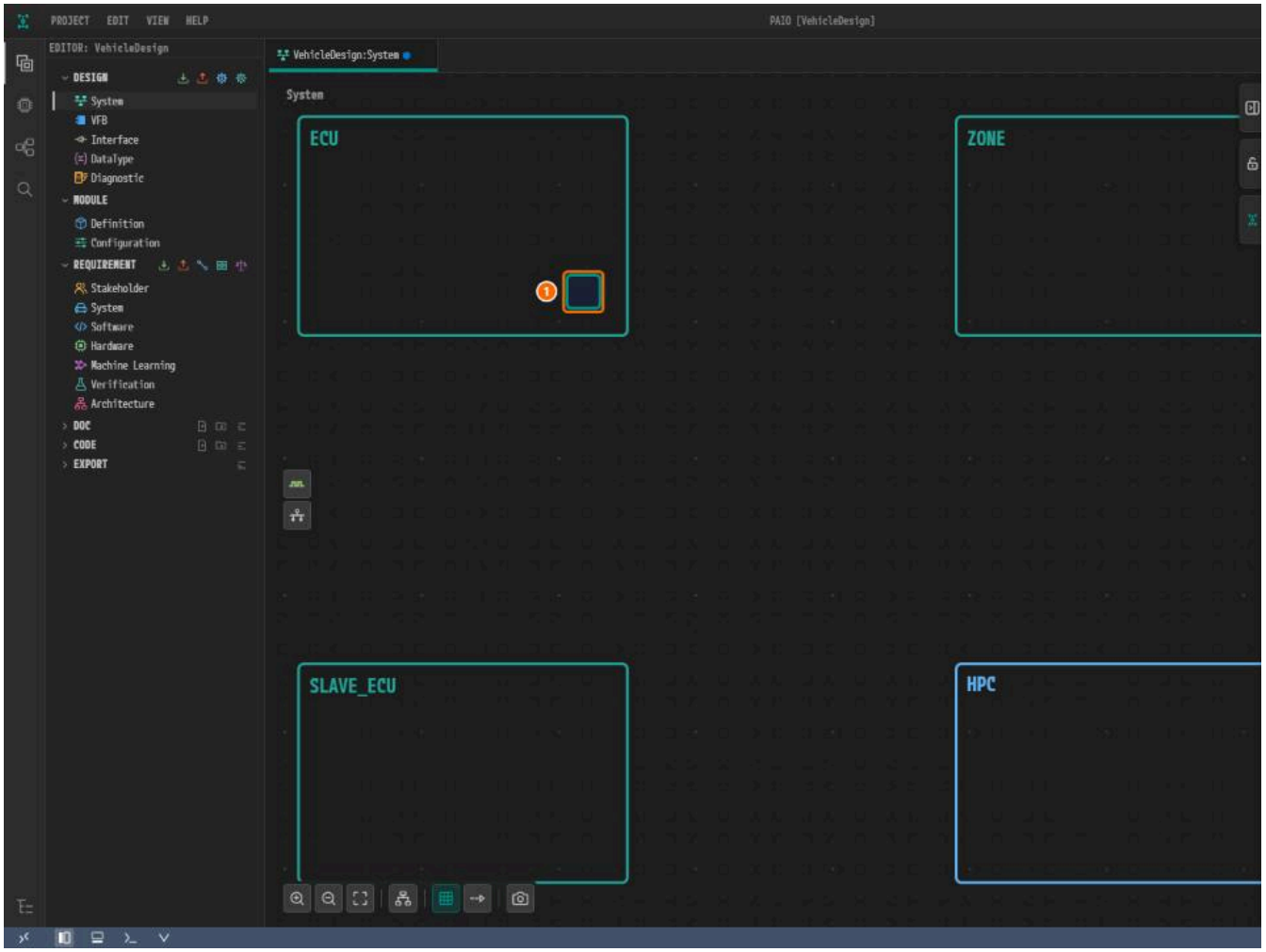


*Figure 3-3. ① The controller, ② the four bit timing fields `0`, ③ **NAME** = `CanController`.*

Creating a Controller on ZONE

1. Right-click an empty spot inside `ZONE`, click **Add Controller > CAN**, and click the new square.
2. Type `CanCommController` into **NAME**. Leave the bit timing fields empty.

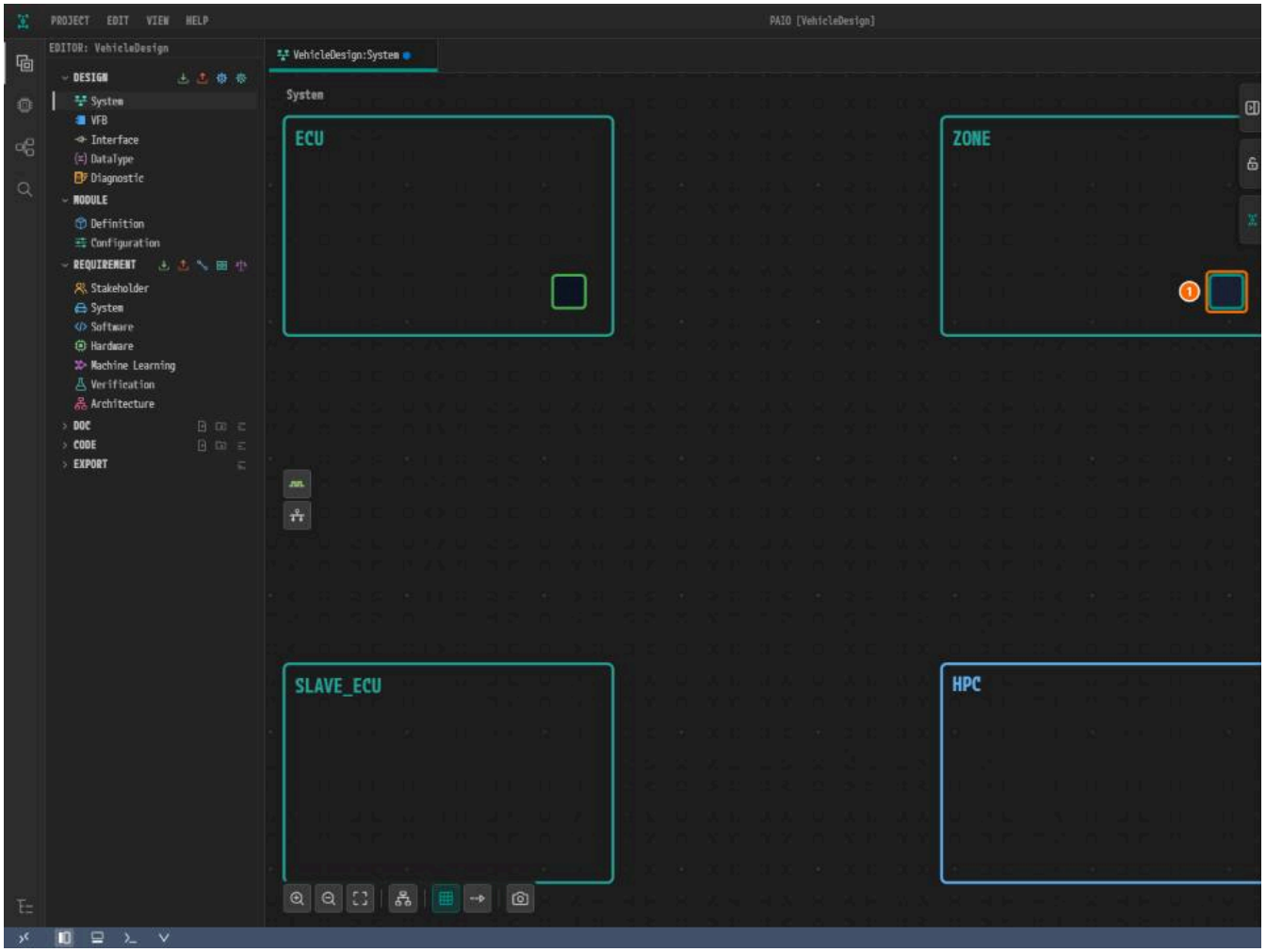


*Figure 3-4. ① The controller inside `ZONE`, ② **NAME** = `CanCommController`.*

Creating the CAN Connectors

Adding a Connector

1. Right-click an empty spot inside `ECU` and click **Add Connector > CAN**.

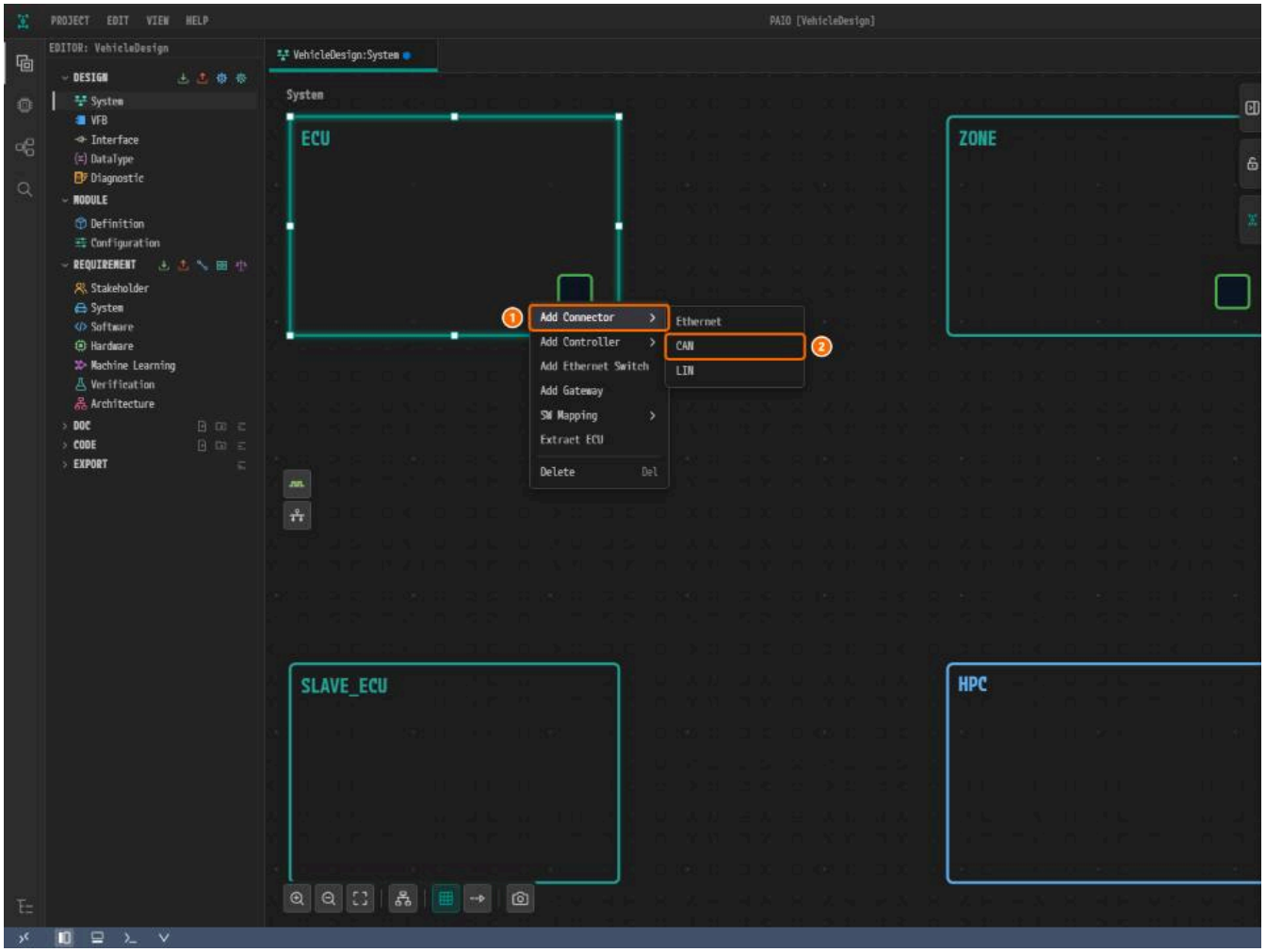


*Figure 3-5. ① **Add Connector**, ② **CAN**.*

2. Click the new connector square on the bottom edge of `ECU`. The panel **CAN CONNECTOR** opens. **CONTROLLER** is red with `Controller is required` u

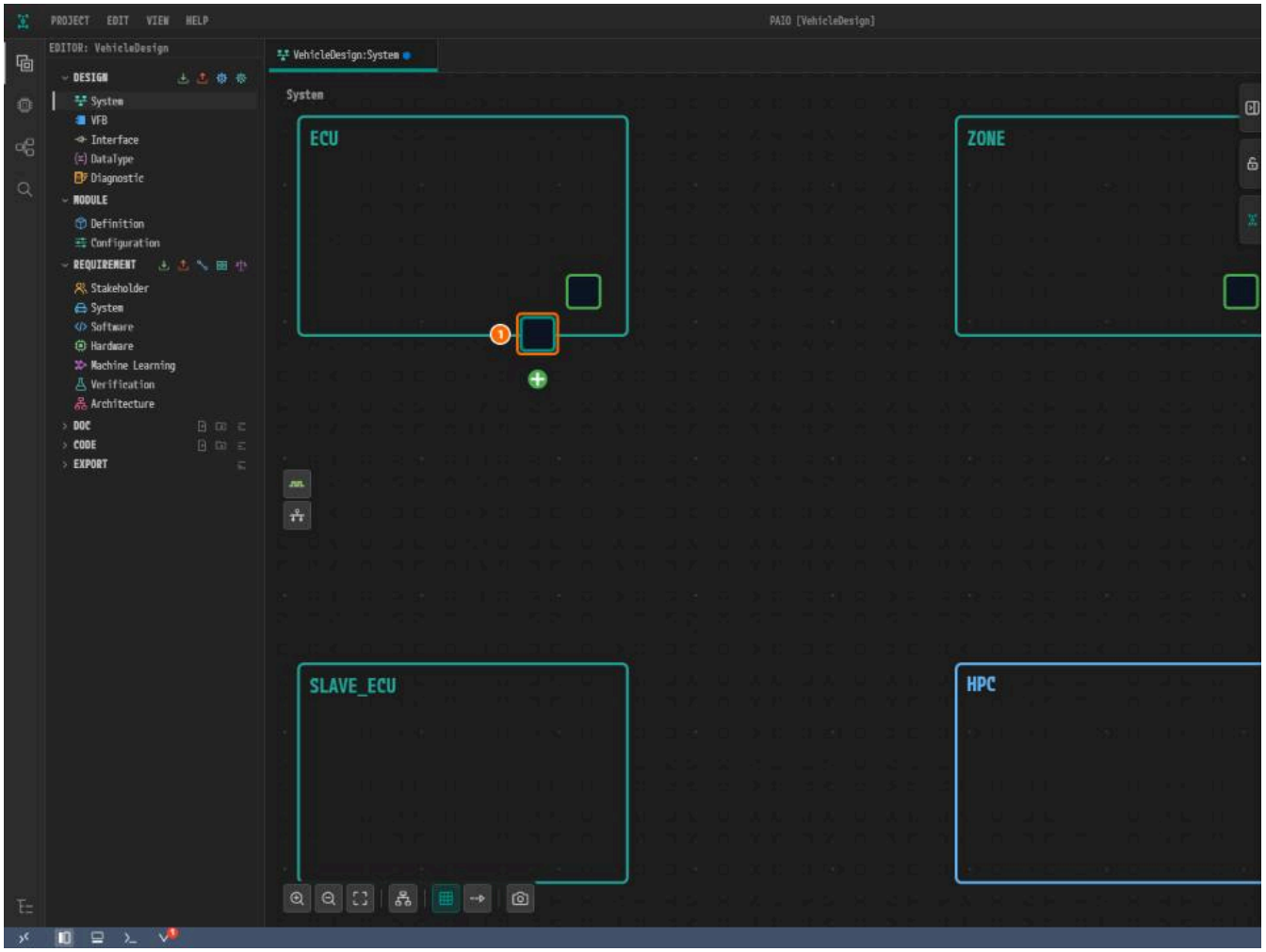


*Figure 3-6. ① The connector, ② **NAME**, ③ **CONTROLLER**, ④ **CONNECTED CHANNEL**.*

Setting the Name and Controller

1. Type `CanConnector` into **NAME**.
2. Click **CONTROLLER** and click `CanController` in the list. An orange line joins the connector and the controller.

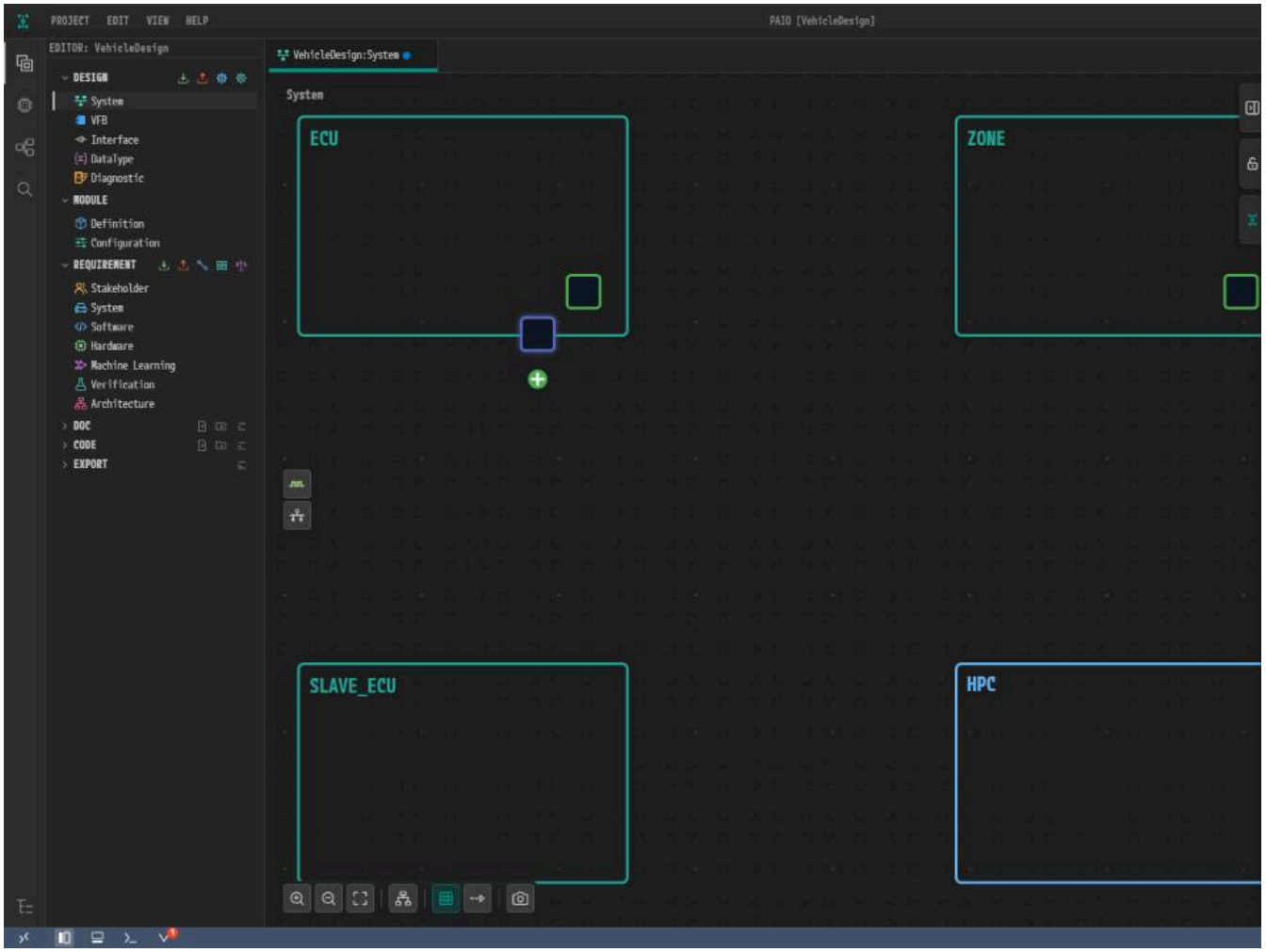


*Figure 3-7. ① `CanController`, ② **NAME**, ③ **CONTROLLER**.*

Creating a Connector on ZONE

1. Right-click an empty spot inside `ZONE`, click **Add Connector > CAN**, and click the new connector.
2. Type `CanConnector` into **NAME** and select `CanCommController` in **CONTROLLER**.

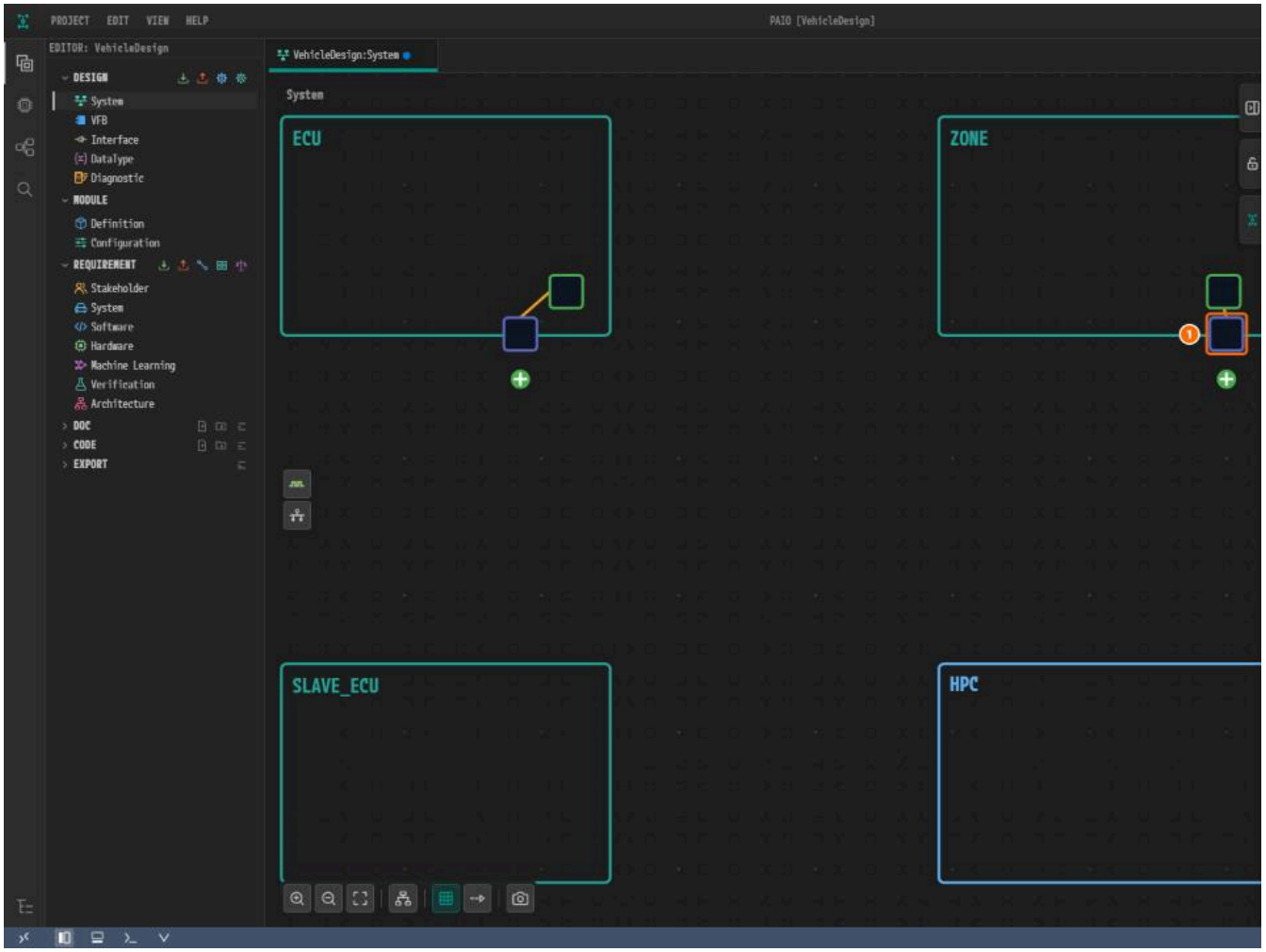


*Figure 3-8. ① The connector of `ZONE`, ② **NAME**, ③ **CONTROLLER**.*

The two connectors may share the name `CanConnector` because each belongs to its own node.

Creating the CAN Cluster

Creating the Cluster from the ZONE Connector

Create the cluster from `ZONE`, so that its connector comes first in the channel, as in the reference design.

1. Click the green **+** badge below `CanConnector` of `ZONE` and click **Create CommunicationCluster [Can]**.

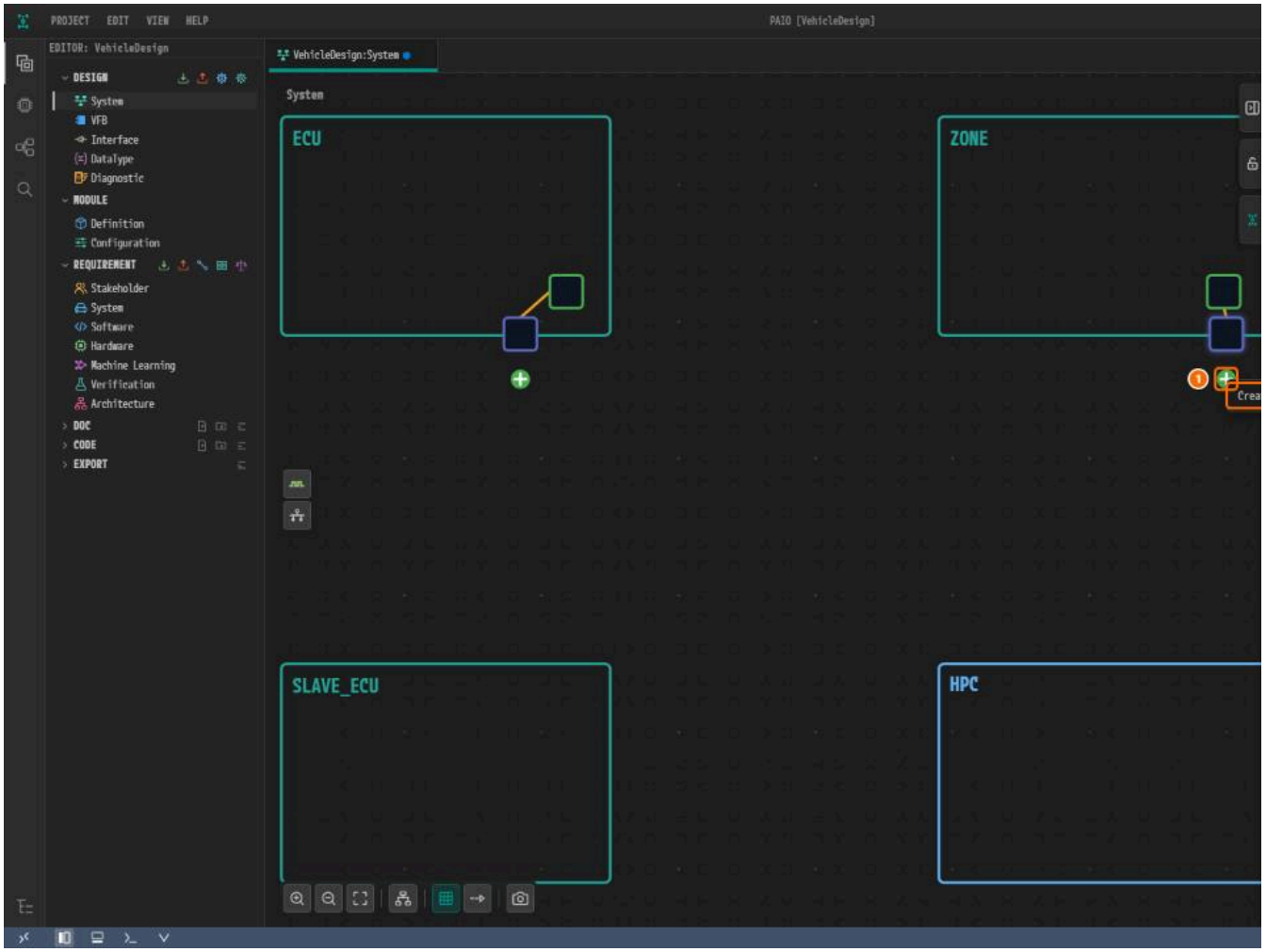


*Figure 3-9. ① The **+** badge, ② **Create CommunicationCluster [Can]**.*

2. Click the new cluster square below `ZONE` (click **Fit to View** if you cannot see it). The panel **CAN CLUSTER** opens.

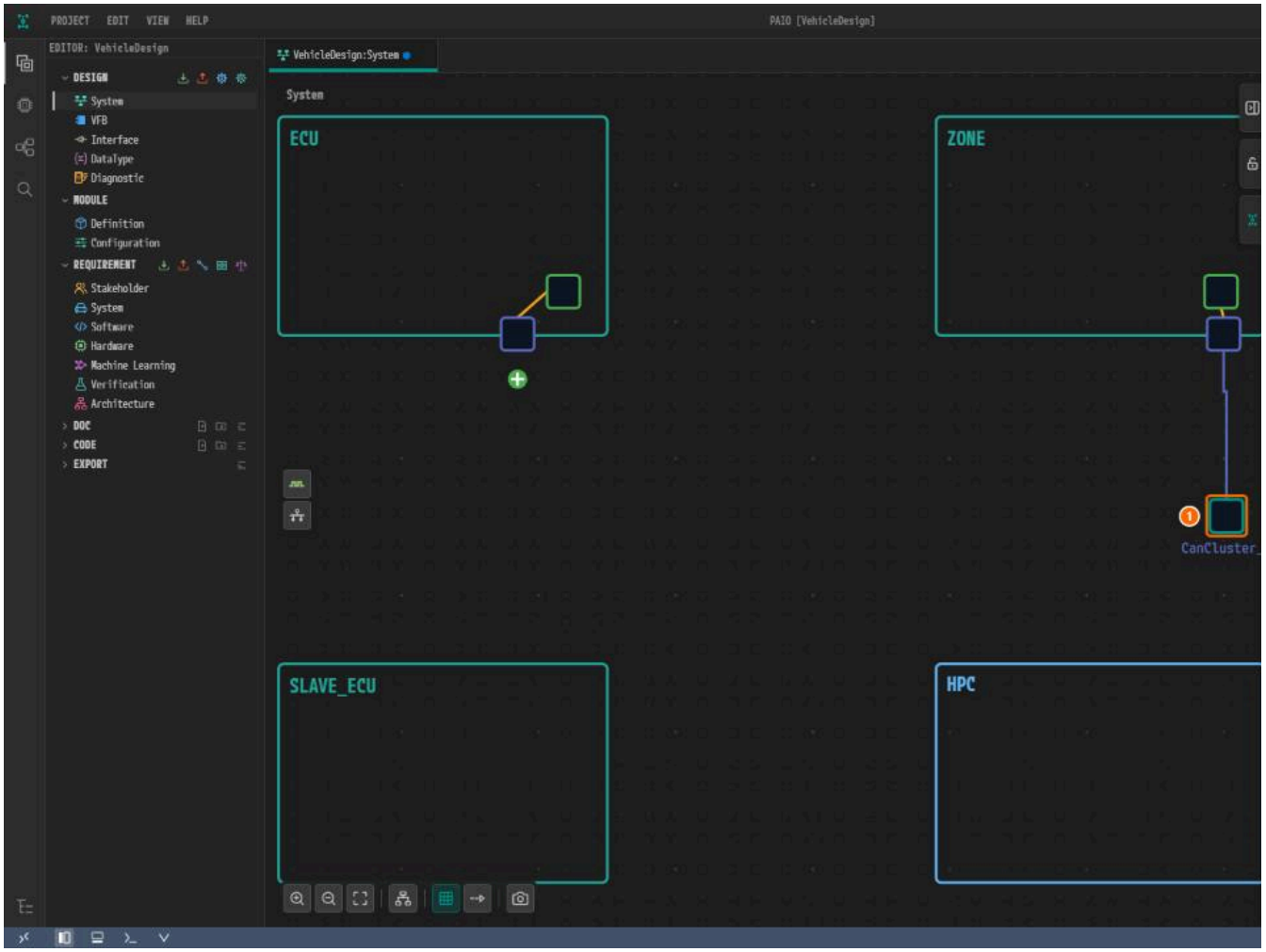


*Figure 3-10. ① The cluster, ② **NAME**, ③ **BAUDRATE**, ④ **CHANNEL NAME**.*

3. Type `CanCluster` into **NAME**. Leave **PROTOCOL** and **CHANNEL NAME** as they are.

> **Note.** No **BAUDRATE** button is on, but the saved cluster holds `500000`. See ARXML Export.

Connecting the ECU Connector to the Channel

1. Click the **+** badge below `CanConnector` of `ECU` and click **Connect to Channel... > CanCluster > PhysicalChannel**. Do not click **Create CommunicationCl**

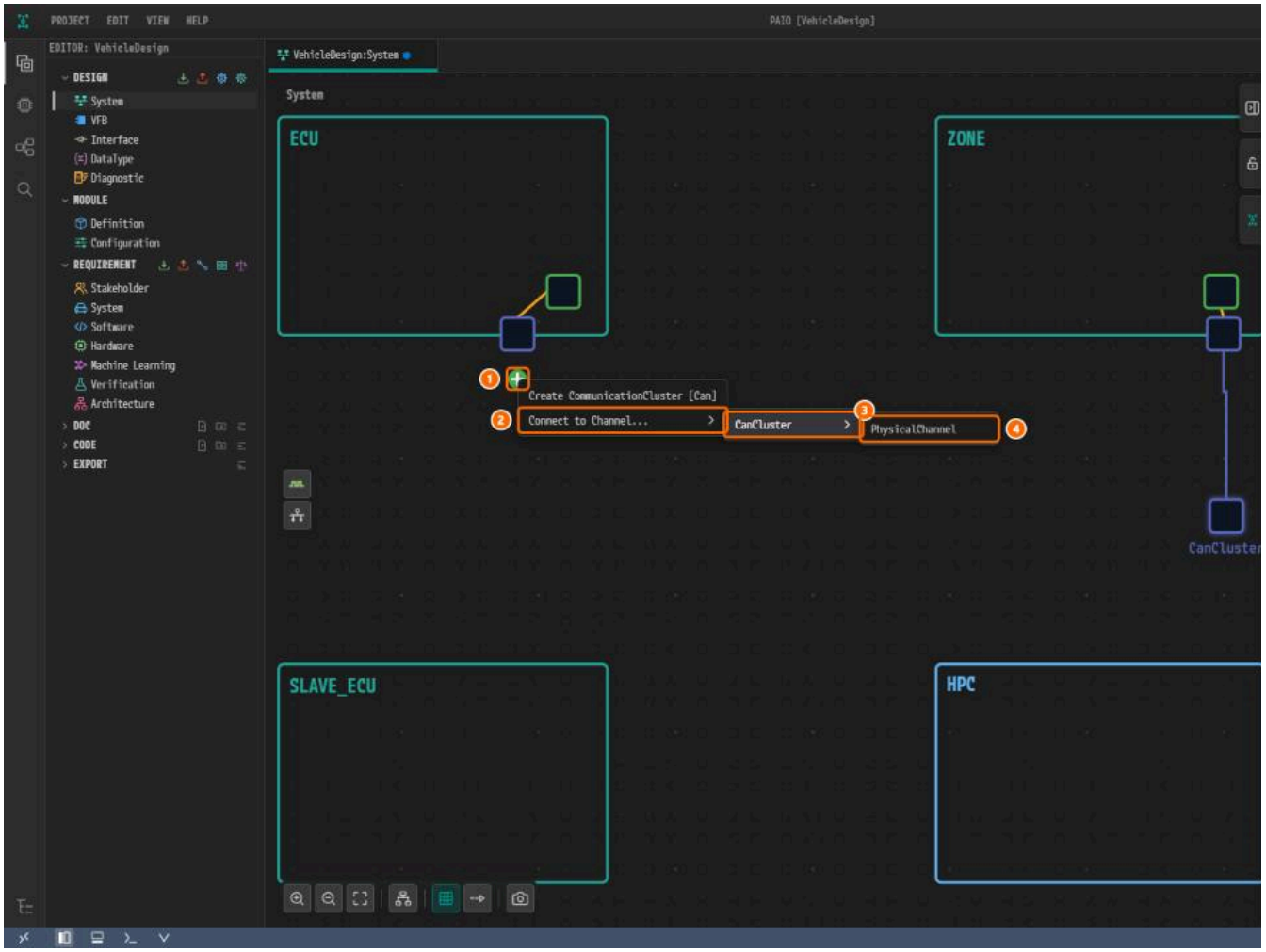


*Figure 3-11. ① The **+** badge, ② **Connect to Channel...**, ③ **CanCluster**, ④ **PhysicalChannel**.*

Tidying the Layout and Saving

1. Click **Auto Layout**, then **Fit to View**, in the toolbar below the canvas.
2. Click the cluster and check that **Connected Connectors (2)** lists `ZONE/CanConnector` and `ECU/CanConnector`.

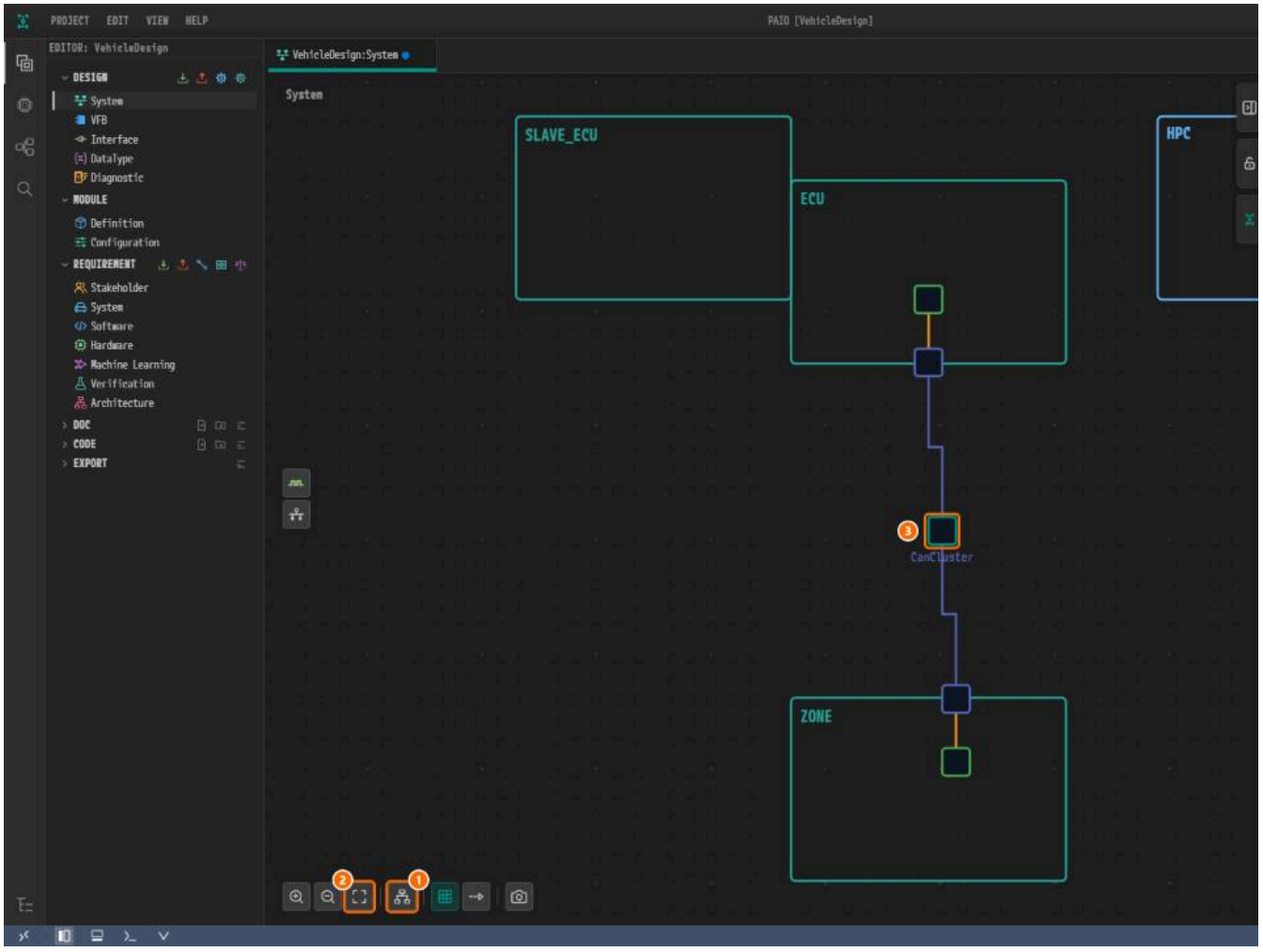


*Figure 3-12. ① **Auto Layout**, ② **Fit to View**, ③ the cluster, ④ **CHANNEL NAME** and the two connectors.*

3. In **BUS-OFF RECOVERY**, set the unit to `ms` first, then enter **BOR TIME L1** `100`, **BOR TIME L2** `200` and **BOR TIME TX ENSURED** `100`. Enter **BOR COUNTER L1** 1

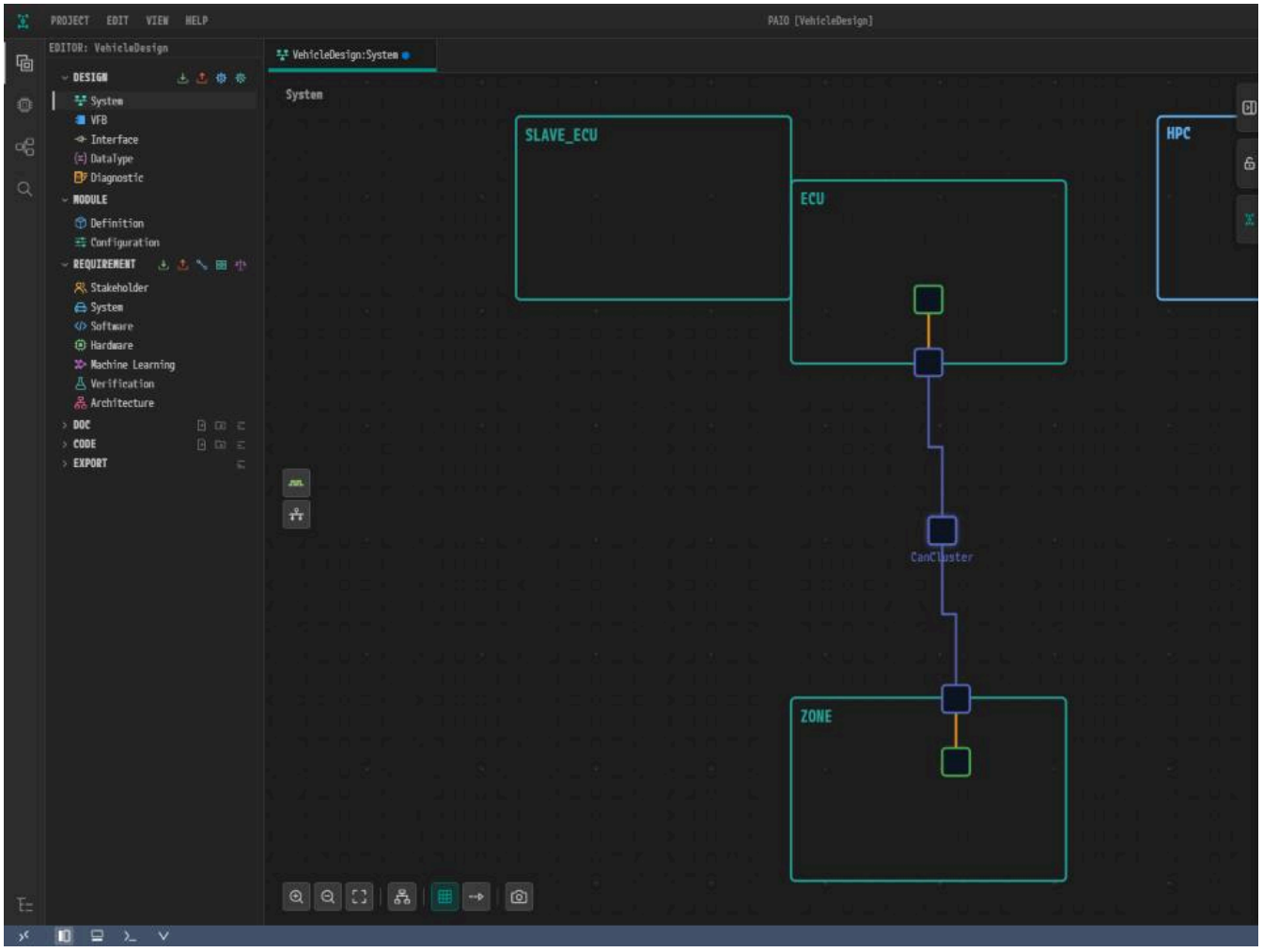


*Figure 3-13. ① **BOR TIME L1**, ② **BOR TIME L2**, ③ **BOR COUNTER L1 TO L2**, ④ **BOR TIME TX ENSURED.***

4. Click the empty canvas and press `Ctrl+S`.

Viewing the Validation Result

1. Click the **V** icon to open the **VALIDATION** panel. It shows `No validation issues.`.

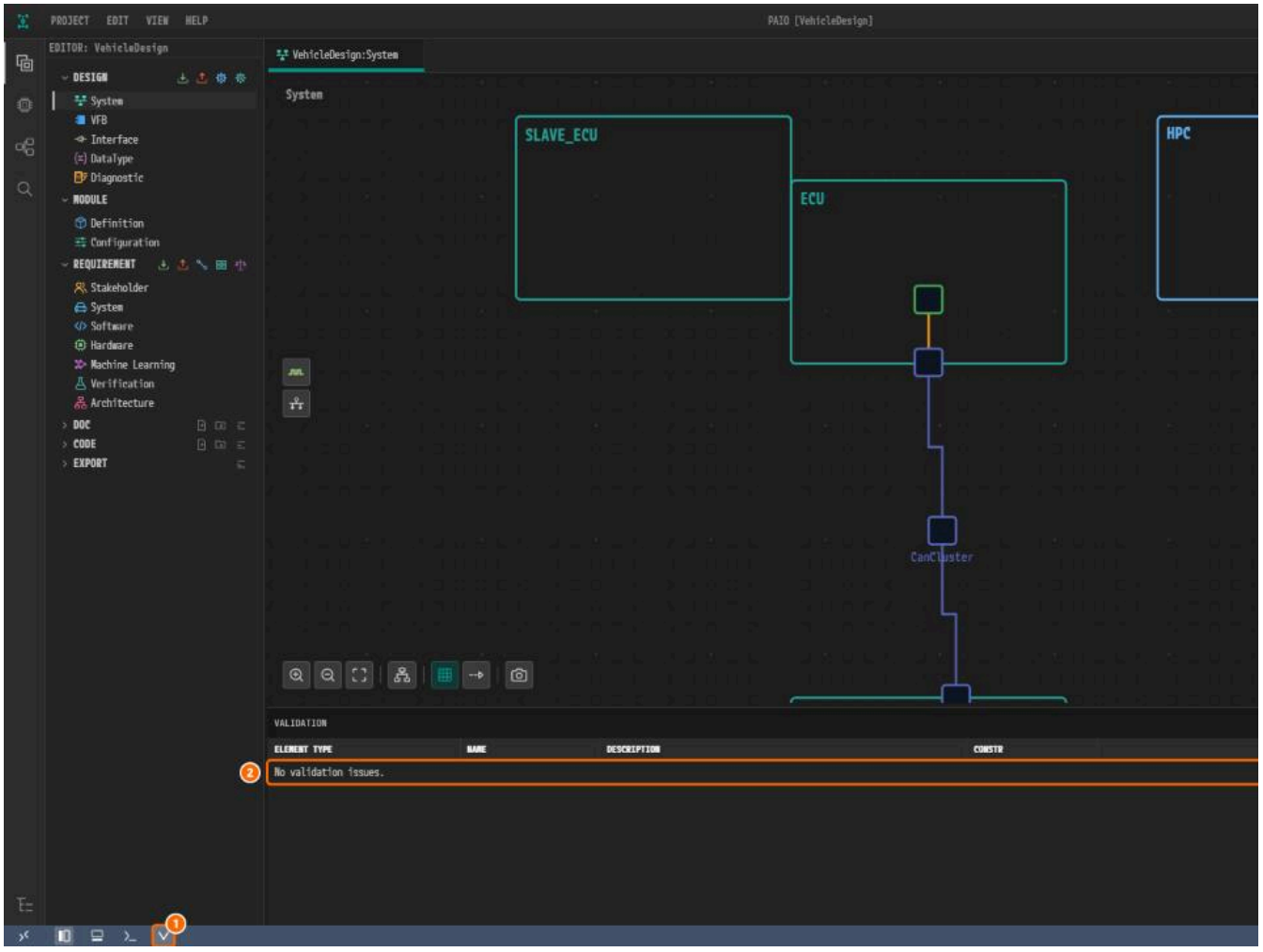


*Figure 3-14. ① The **V** icon, ② `No validation issues.`.*

Checking the Network Management Tab

Opening the Cluster Configuration Window

1. Close the **VALIDATION** panel.
2. Double-click `CanCluster`. The window **Communication Cluster Configuration [CanCluster]** opens.
3. Click the **NM CLUSTER** tab.

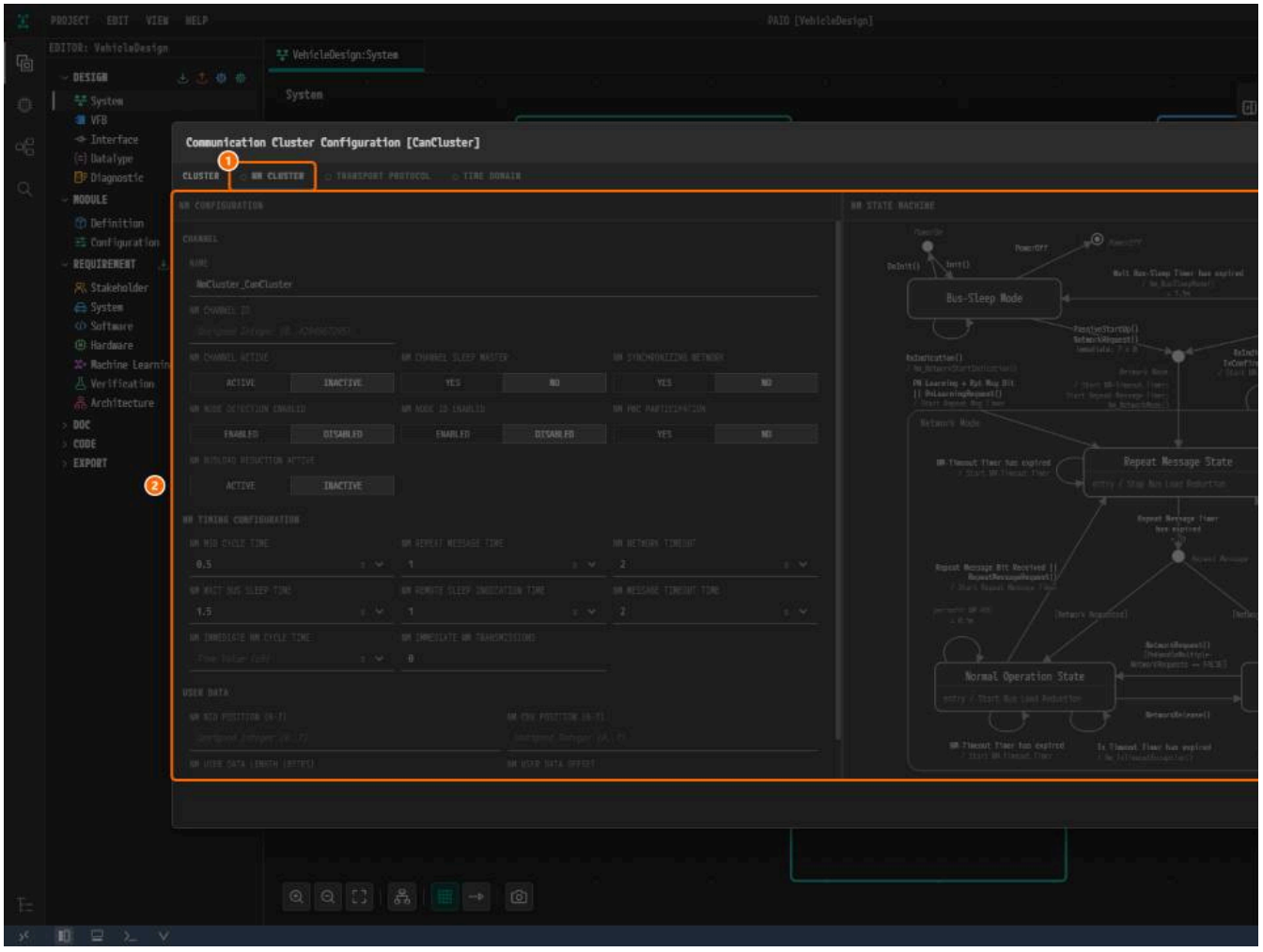


*Figure 3-15. ① **NM CLUSTER** with a hollow dot, ② the dimmed editing area.*

A hollow dot means the feature is off; **TRANSPORT PROTOCOL** and **TIME DOMAIN** have one too. The dimmed area is only a preview and is not stored.

> **Caution.** Do not right-click the tab and choose **Enable**. The reference design has no NM.

Checking the Transport Protocol Configuration

1. Click the **TRANSPORT PROTOCOL** tab. It also has a hollow dot.

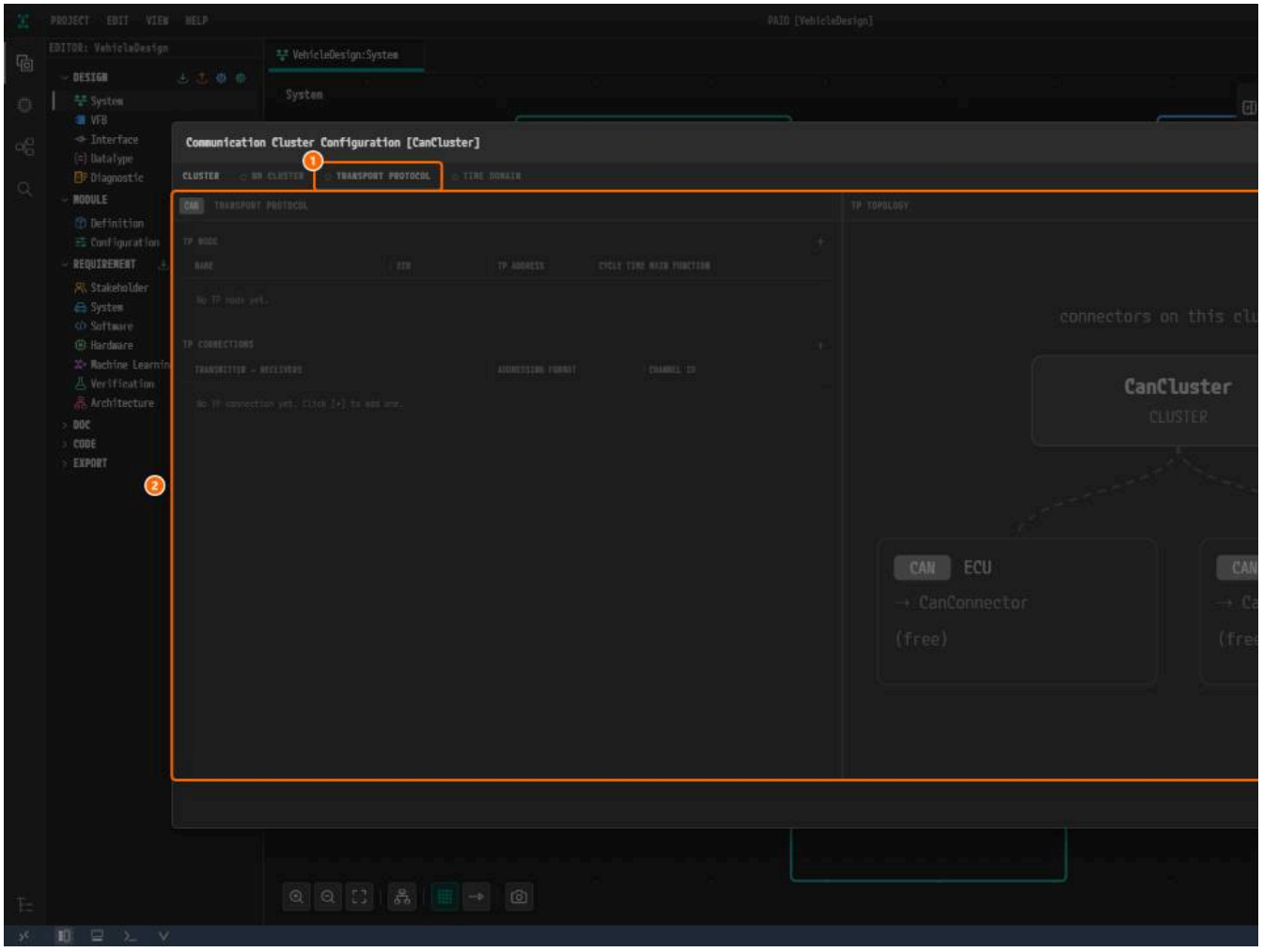


*Figure 3-16. ① **TRANSPORT PROTOCOL** with a hollow dot, ② the dimmed editing area.*

2. Click **Close**.

Validating Again

1. Open the **VALIDATION** panel. It still shows `No validation issues.`, and the tab `VehicleDesign:System` has no modified mark.

Why Is It Designed This Way?

- **Order of the connectors.** The reference design lists `ZONE` first, then `ECU`, so the cluster is created from `ZONE`.
- **Names.** The controllers are `CanController` and `CanCommController`; both connectors are `CanConnector`, as in the reference design.
- **Bit timing only on `ECU`.** `CanController` has four `0` values; `CanCommController` has none.
- **Bus-off recovery values.** The reference design has none, but PRTE needs them for CanSM when it generates the code of `ECU`. They are the values of
- **NM and TP stay off.** The reference design has neither, and a new cluster has both off.

Checklist

- [ ] `ECU` has `CanController` (bit timing `0`, **CAN**) and `CanConnector` (**CONTROLLER** = `CanController`).
- [ ] `ZONE` has `CanCommController` and `CanConnector` (**CONTROLLER** = `CanCommController`).
- [ ] `CanCluster` lists `ZONE/CanConnector` and `ECU/CanConnector`, in that order, on `PhysicalChannel`.
- [ ] **BUS-OFF RECOVERY** reads `100` `ms`, `200` `ms`, `2`, `100` `ms`.
- [ ] **NM CLUSTER**, **TRANSPORT PROTOCOL** and **TIME DOMAIN** have a hollow dot.
- [ ] No modified mark, and `No validation issues.`.

Next Step

In ETHERNET Configuration you join `ZONE` and `HPC` with the Ethernet cluster `EthernetCluster`.

ETHERNET Configuration

What You Will Do

You join the nodes `ZONE` and `HPC` with an Ethernet bus: one Ethernet controller and one Ethernet connector on each node, and the cluster `EthernetClu` VLAN ID `13`. Network management (NM) stays off.

Before You Begin

- Start after finishing CAN Configuration. The **VALIDATION** panel shows `No validation issues.`.
- If the tab `VehicleDesign:System` is not open, click **System** under **DESIGN** in the **EDITOR** view. Close the **VALIDATION** panel with the **V** icon in the foote
- If the node `HPC` is hidden under the property panel, click **Fit to View** in the toolbar below the canvas, then **Zoom Out** until all four nodes are visible.
- A field in the property panel commits its value only when you leave it. Press the `Tab` key after each value.

Where	Item	Value
ETHERNET CONTROLLER panel (node `ZONE`)	**NAME** / **MAC ADDRESS**	`EthernetCommController` / `11:22:EE:44:55:66`
ETHERNET CONTROLLER panel (node `HPC`)	**NAME** / **MAC ADDRESS**	`EthernetController` / `00:11:22:33:44:55`
ETHERNET CONNECTOR panel (node `ZONE`)	**NAME** / **CONTROLLER**	`EthernetConnector` / select `EthernetCommController`
ETHERNET CONNECTOR panel (node `ZONE`)	**UNICAST IPV4 ADDRESS** / **ADDRESS SOURCE**	`192.168.0.19` / click **FIXED**
ETHERNET CONNECTOR panel (node `HPC`)	**NAME** / **CONTROLLER**	`EthernetConnector` / select `EthernetController`
ETHERNET CONNECTOR panel (node `HPC`)	**UNICAST IPV4 ADDRESS** / **ADDRESS SOURCE**	`192.168.0.11` / click **FIXED**
ETHERNET CONNECTOR panel (node `HPC`)	**MULTICAST IPV4 ADDRESS**	`224.0.0.1`
ETHERNET CLUSTER panel	**NAME** / **VLAN ID**	`EthernetCluster` / `13`
ETHERNET CLUSTER panel	**PHYSICAL CHANNELS > CHANNEL NAME**	`PhysicalChannel` (default, leave as is)
Communication Cluster Configuration window	**NM CLUSTER**, **TIME DOMAIN** tabs	Disabled (default, leave as is)

Creating the Ethernet Controllers

1. Right-click an empty spot inside the node `ZONE` and click **Add Controller > Ethernet**.

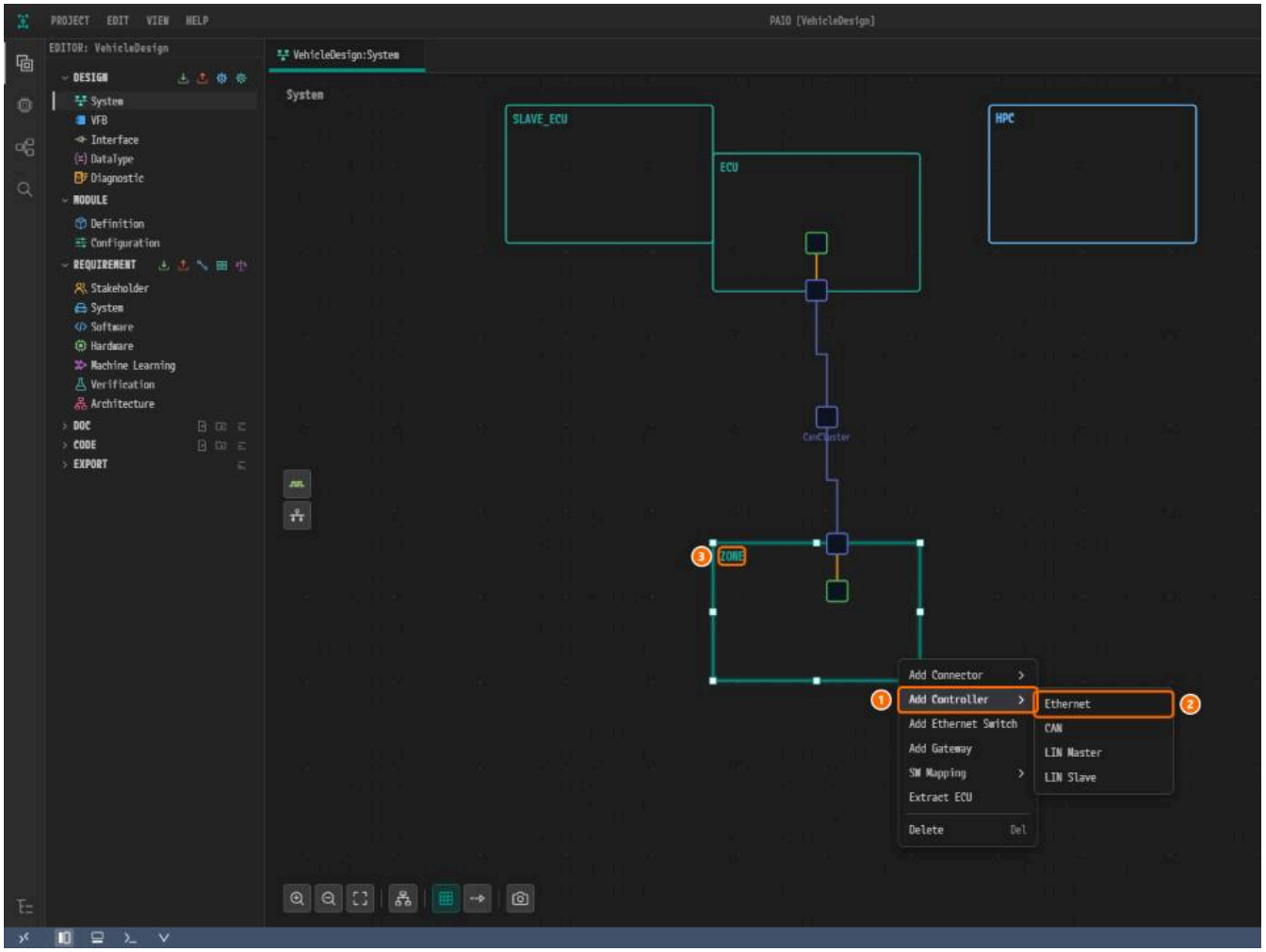


*Figure 4-1. ① **Add Controller**, ② **Ethernet**, ③ the node `ZONE`.*

2. A small square, the controller, appears inside the node. Click it; the **ETHERNET CONTROLLER** panel opens.

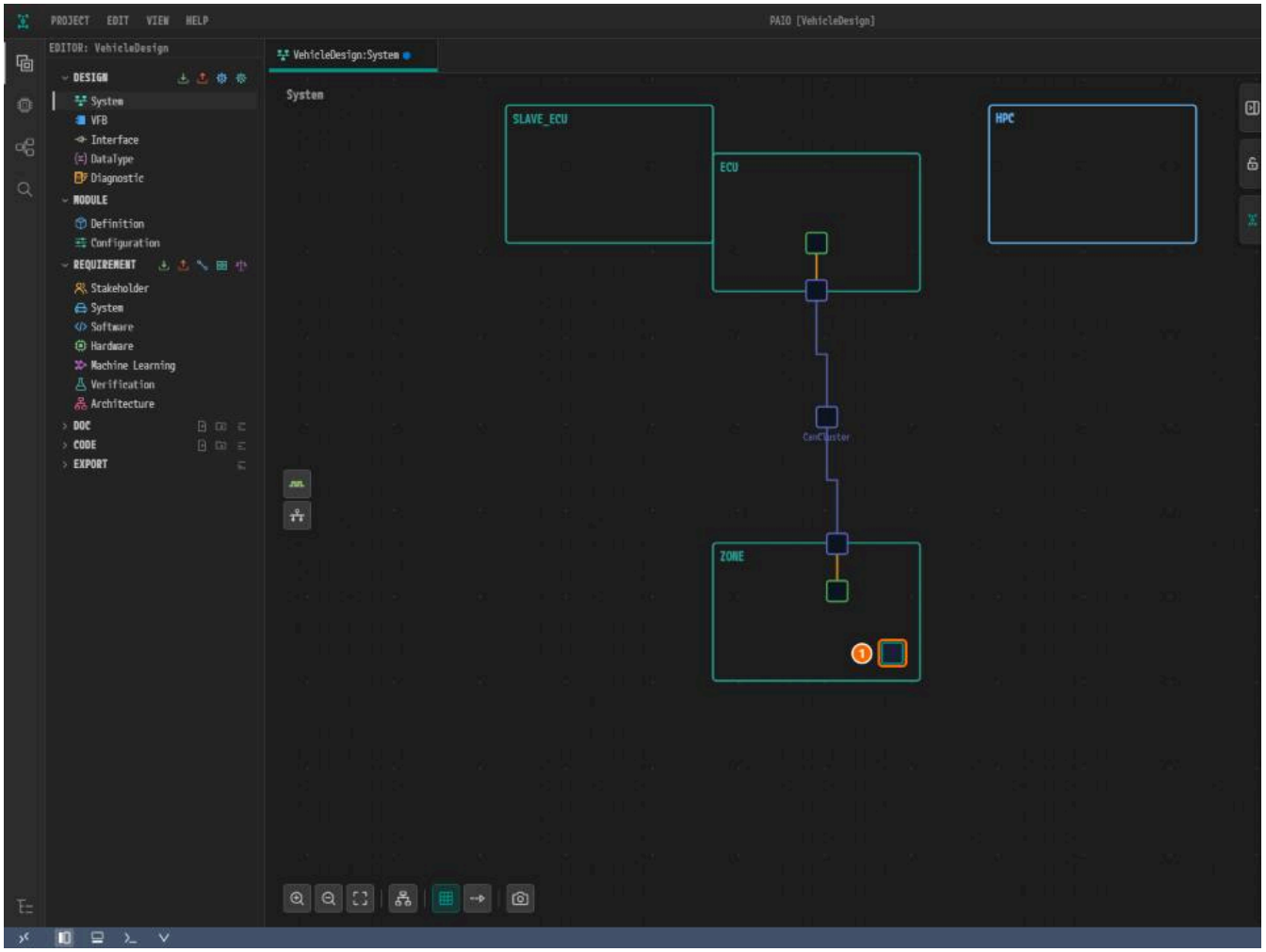


*Figure 4-2. ① The controller, ② **ETHERNET CONTROLLER**, ③ **NAME**, ④ **MAC ADDRESS**, ⑤ **MAC LAYER TYPE**.*

3. Type `EthernetCommController` into **NAME** and `11:22:EE:44:55:66` into **MAC ADDRESS**. The address is shown in lowercase; it is the same address. D

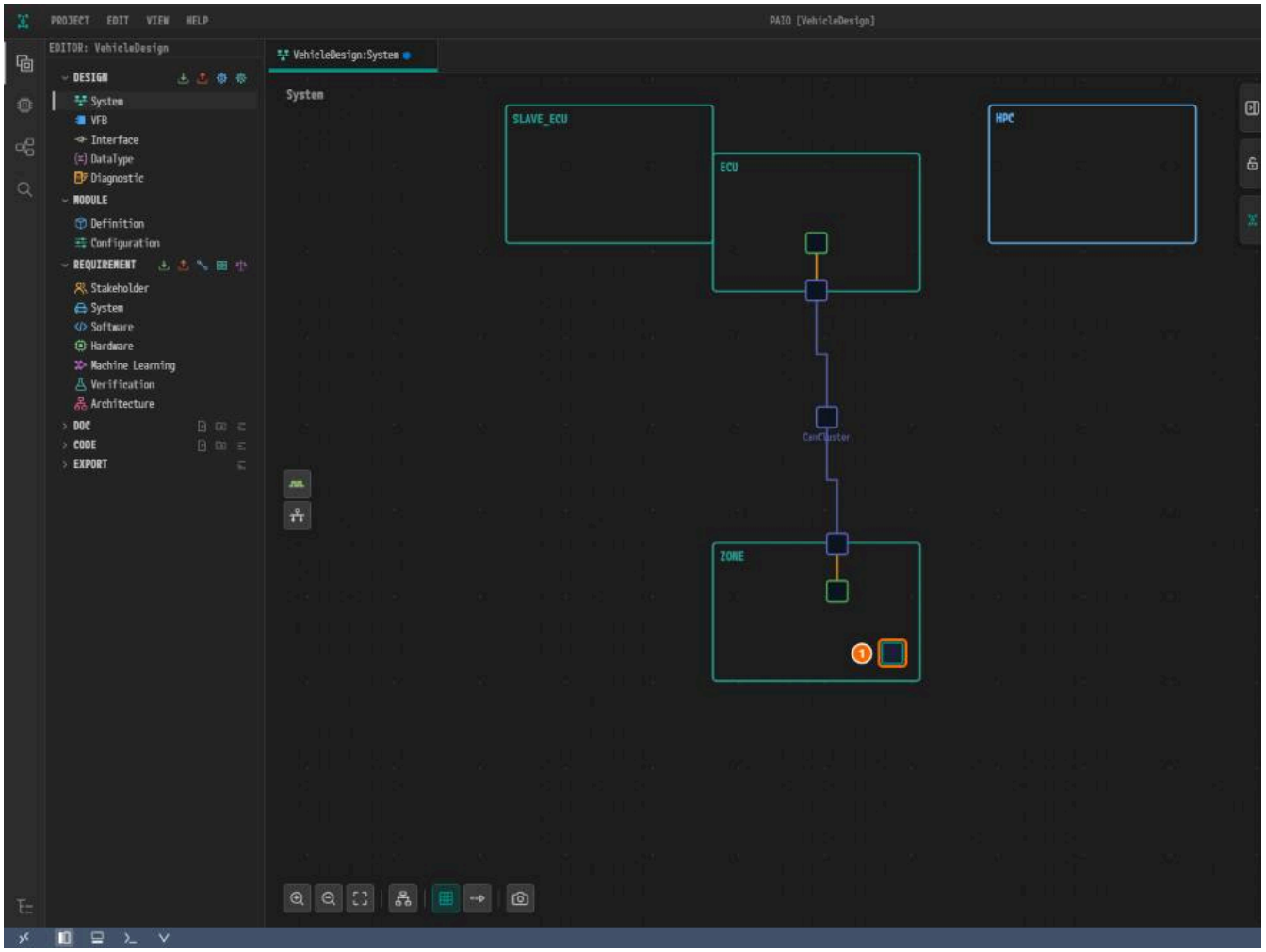


*Figure 4-3. ① The controller, ② **NAME**, ③ **MAC ADDRESS**, ④ **MAC LAYER TYPE**, left unclicked.*

> **Caution.** The toggle values `XMII`, `XGMII` and `XXGMII` are not AUTOSAR schema values (`X-MII`, `XG-MII`, `XXG-MII`), so clicking one exports a value

4. Right-click an empty spot inside the node `HPC` and click **Add Controller > Ethernet**.

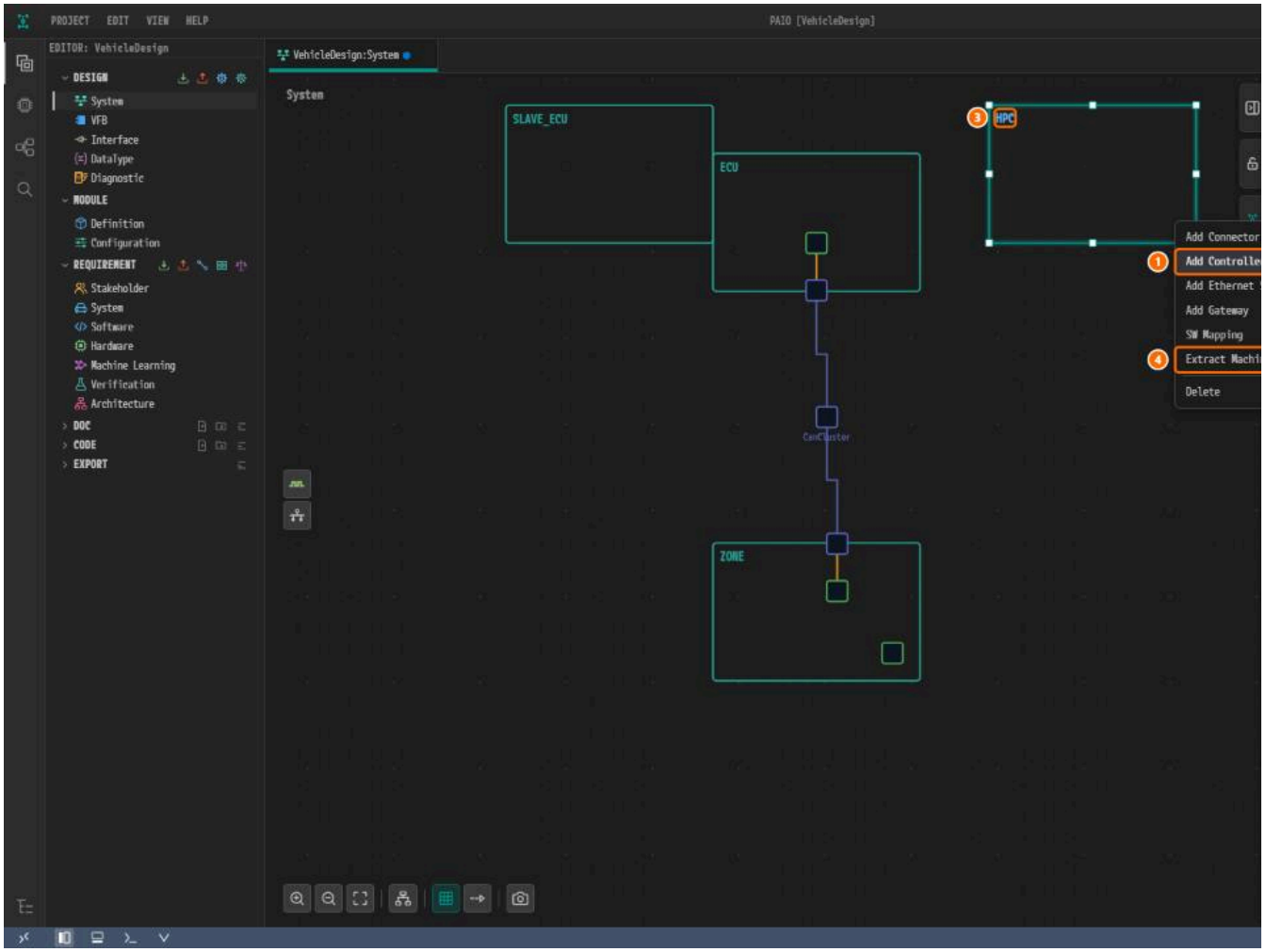


*Figure 4-4. ① **Add Controller**, ② **Ethernet**, ③ the node `HPC`, ④ **Extract Machine**.*

5. Click the new controller inside `HPC`. Type `EthernetController` into **NAME** and `00:11:22:33:44:55` into **MAC ADDRESS**. Again, do not click **MAC LAYE**

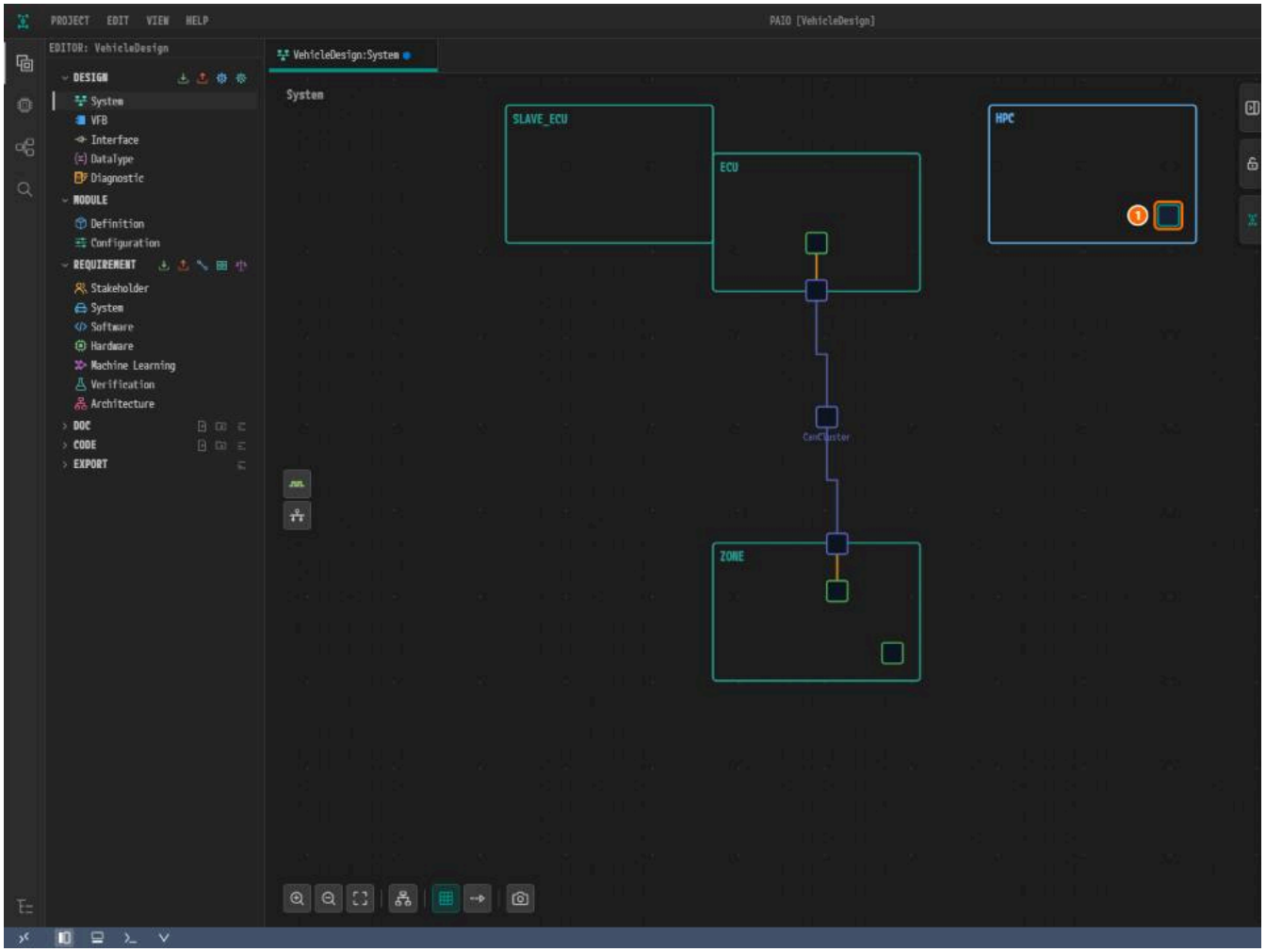


*Figure 4-5. ① The controller of `HPC`, ② **NAME**, ③ **MAC ADDRESS**.*

Creating the Ethernet Connectors

1. Right-click an empty spot inside `ZONE` and click **Add Connector > Ethernet**. A connector appears on the bottom edge of the node, with a green **+** bac (`Controller is required`) until you pick a controller.

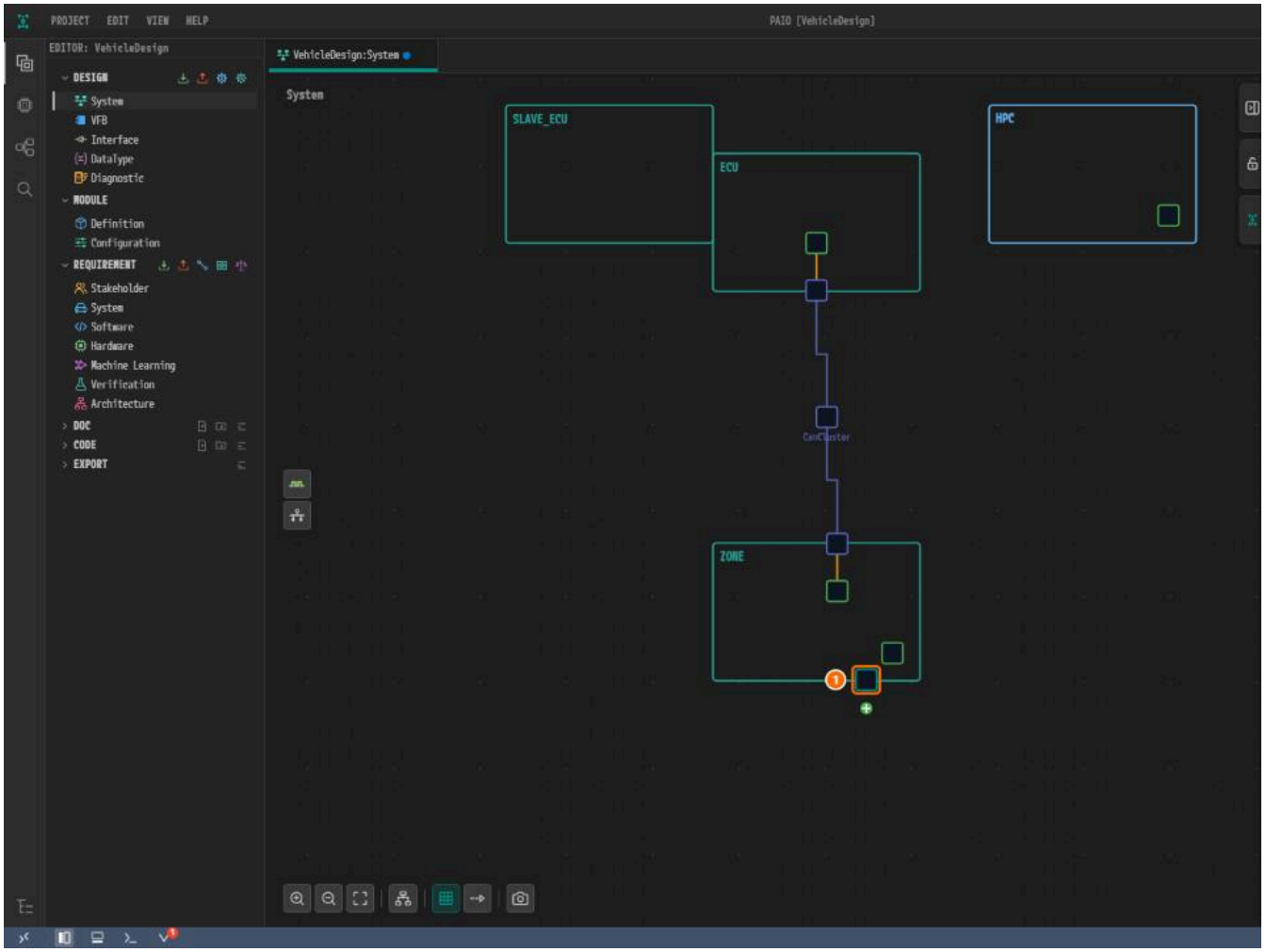


*Figure 4-6. ① The connector, ② **ETHERNET CONNECTOR**, ③ **NAME**, ④ **CONTROLLER**, ⑤ **NETWORK ENDPOINT**.*

2. Type `EthernetConnector` into **NAME**, pick `EthernetCommController` in the **CONTROLLER** list, type `192.168.0.19` into **UNICAST IPV4 ADDRESS** and cl joins the connector and the controller.

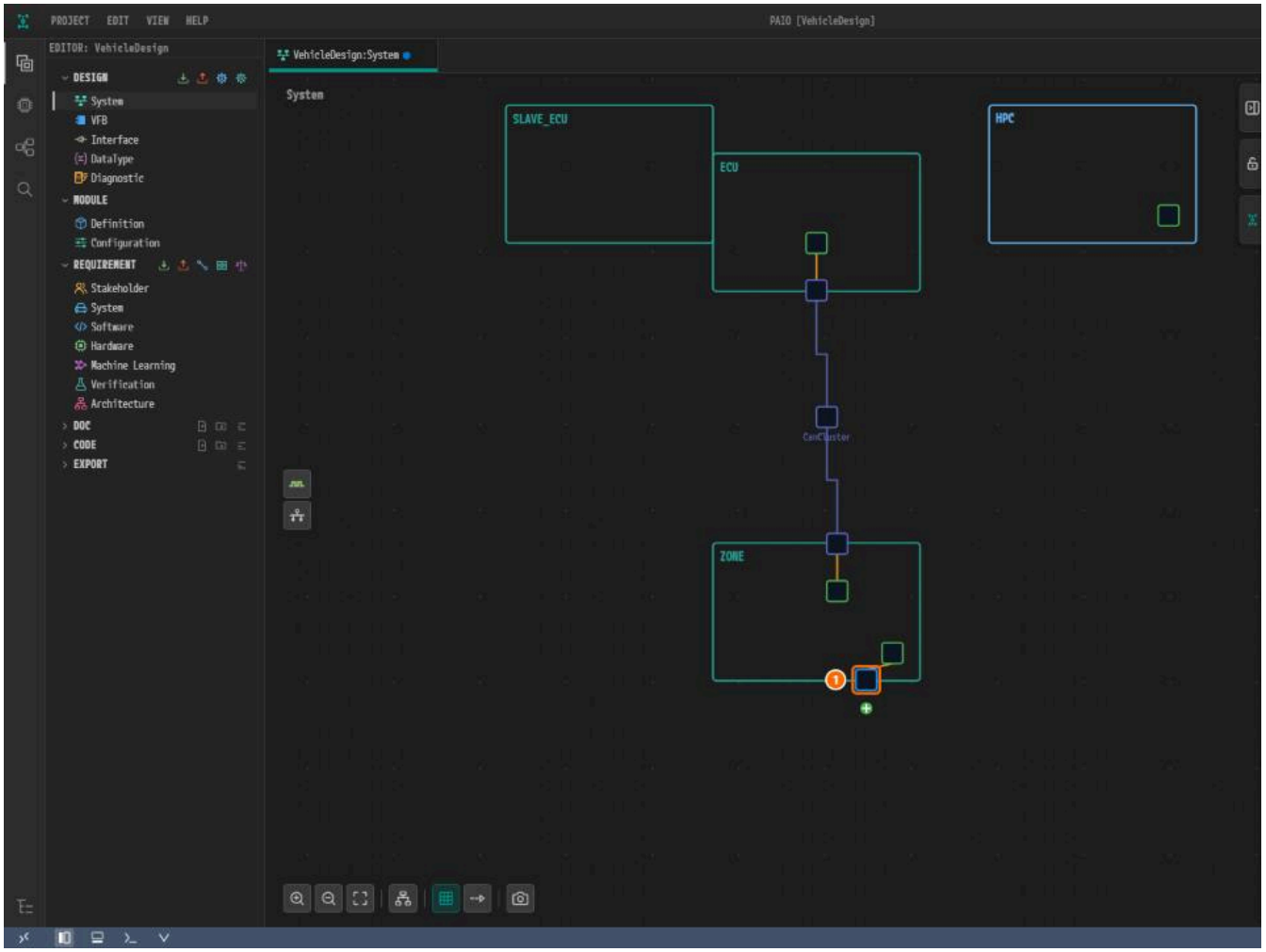


*Figure 4-7. ① The connector, ② **NAME**, ③ **CONTROLLER**, ④ **UNICAST IPV4 ADDRESS**, ⑤ **FIXED**.*

> **Caution. ADDRESS SOURCE** starts with no value on. Without **FIXED** the connector is saved with no address source.

3. Create the connector of `HPC` the same way: **NAME** `EthernetConnector`, **CONTROLLER** `EthernetController`, **UNICAST IPV4 ADDRESS** `192.168.0.11`, **ADDRESS** under **MULTICAST CONFIG**. Leave **MULTICAST PORT** empty.

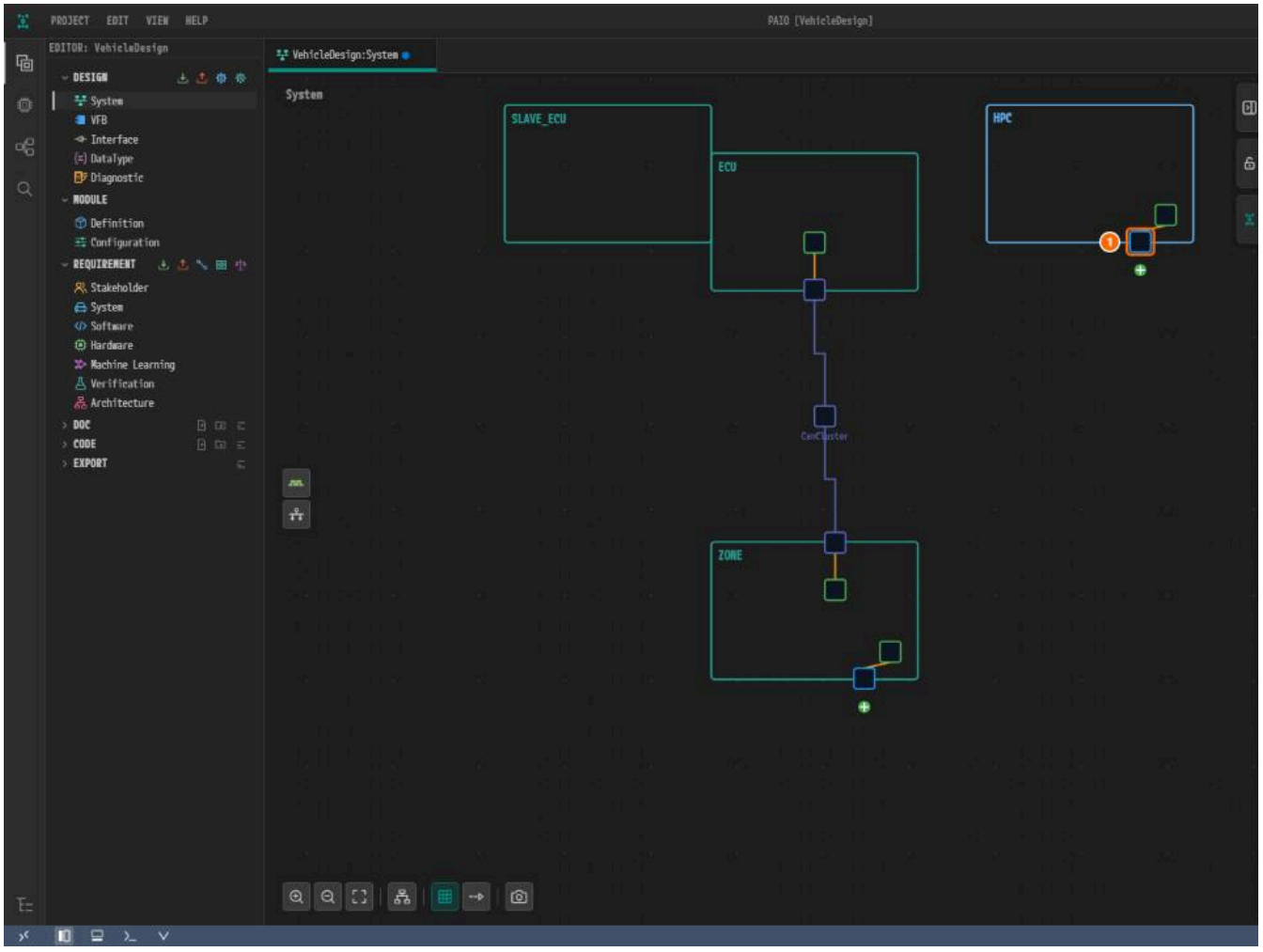


*Figure 4-8. ① The connector of `HPC`, ② **NAME**, ③ **CONTROLLER**, ④ **UNICAST IPV4 ADDRESS**, ⑤ **FIXED**, ⑥ **MULTICAST IPV4 ADDRESS**.*

Creating the Ethernet Cluster

1. Click the green **+** badge below the connector of `ZONE` and click **Create CommunicationCluster [Ethernet]**.

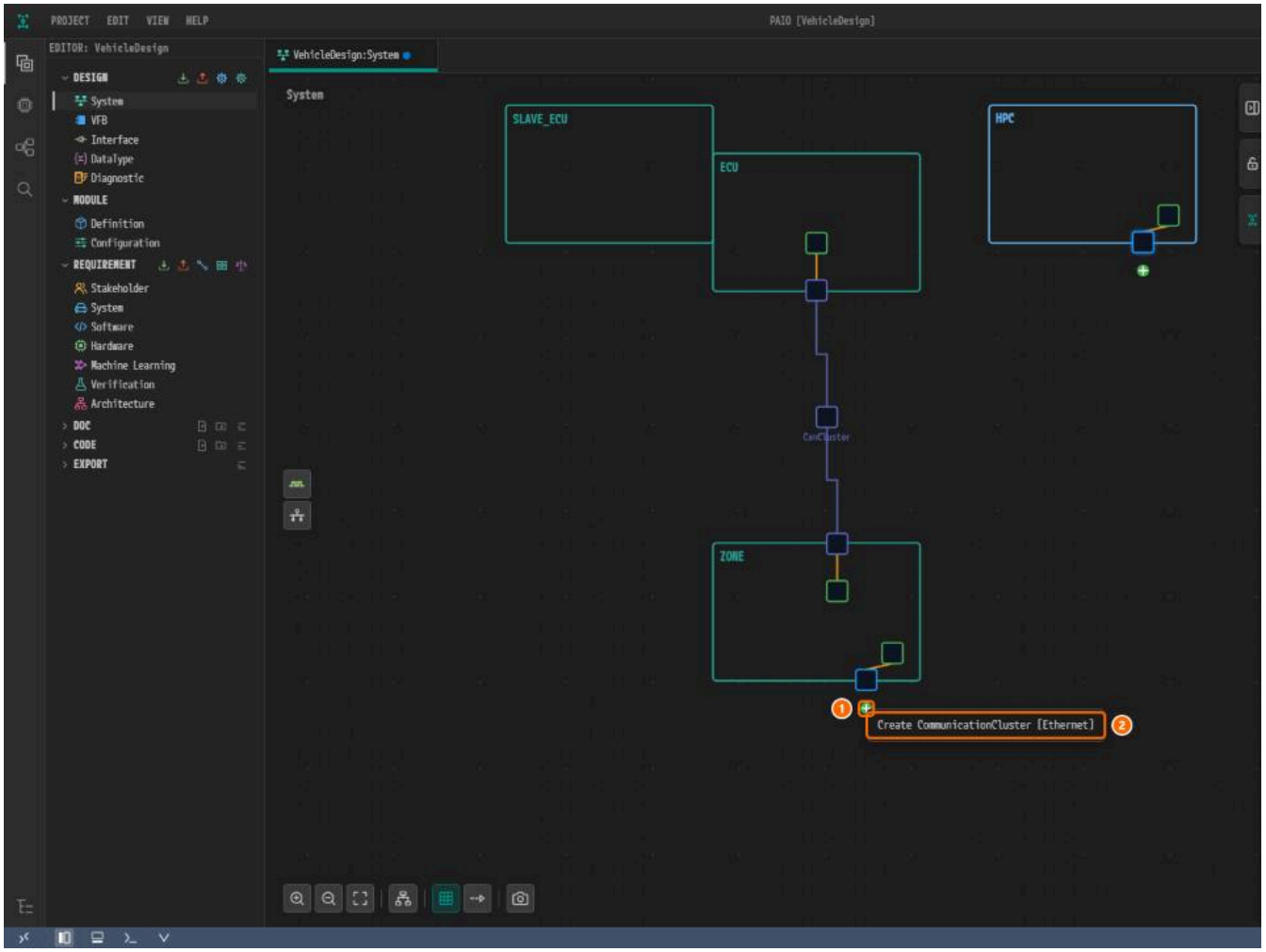


*Figure 4-9. ① The **+** badge, ② **Create CommunicationCluster [Ethernet]**.*

2. Click the new cluster square below `ZONE`. If you cannot see it, click **Fit to View**.

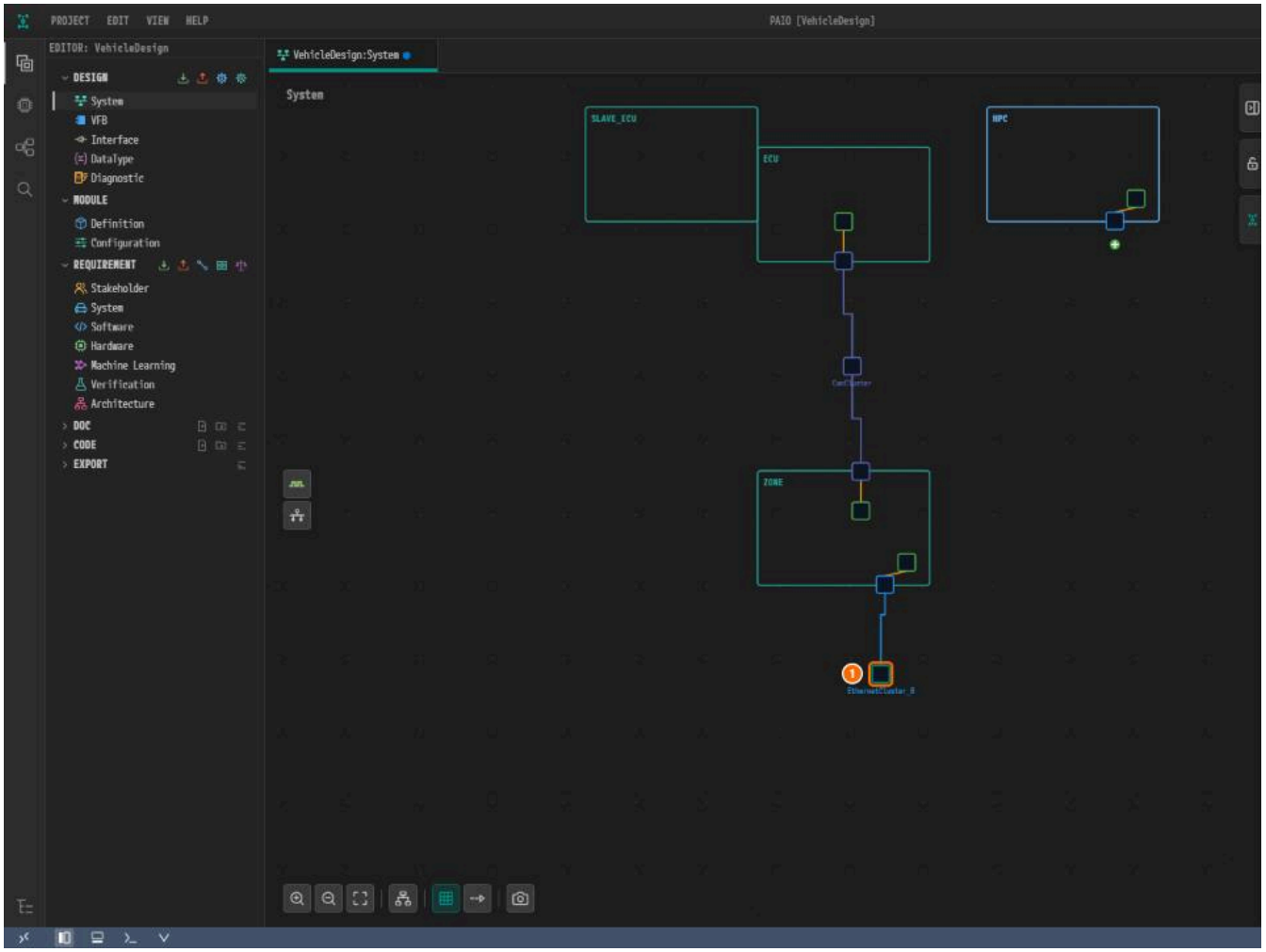


*Figure 4-10. ① The cluster, ② **NAME**, ③ **CHANNEL NAME** (`PhysicalChannel`), ④ **VLAN ID**.*

3. Type `EthernetCluster` into **NAME** and `13` into **VLAN ID**. Leave **CHANNEL NAME** as it is.

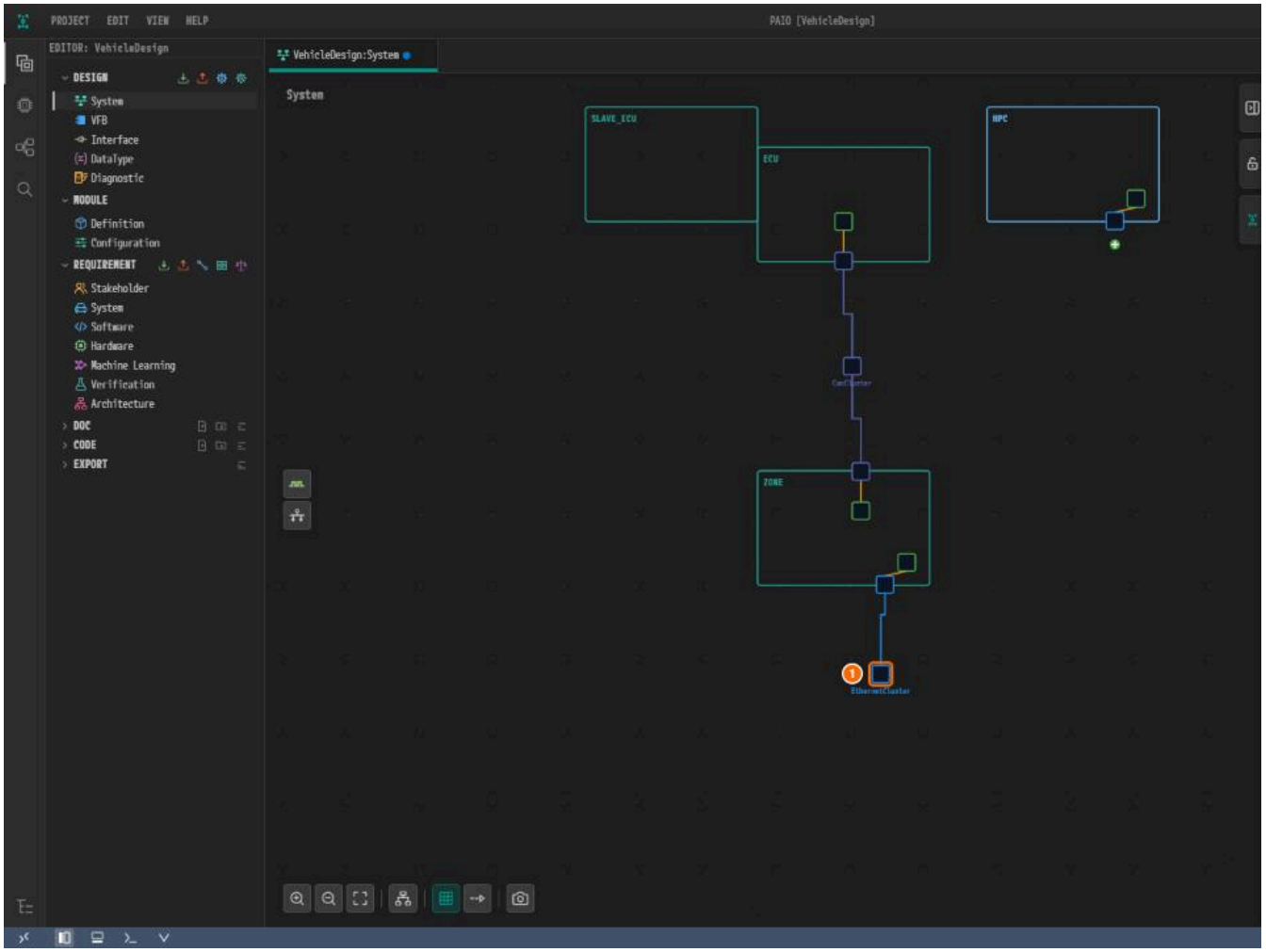


*Figure 4-11. ① The cluster, ② **NAME**, ③ **VLAN ID**.*

4. Click the **+** badge below the connector of `HPC` and click **Connect to Channel... > EthernetCluster > PhysicalChannel (VLAN: 13)**. If the badge is hard to

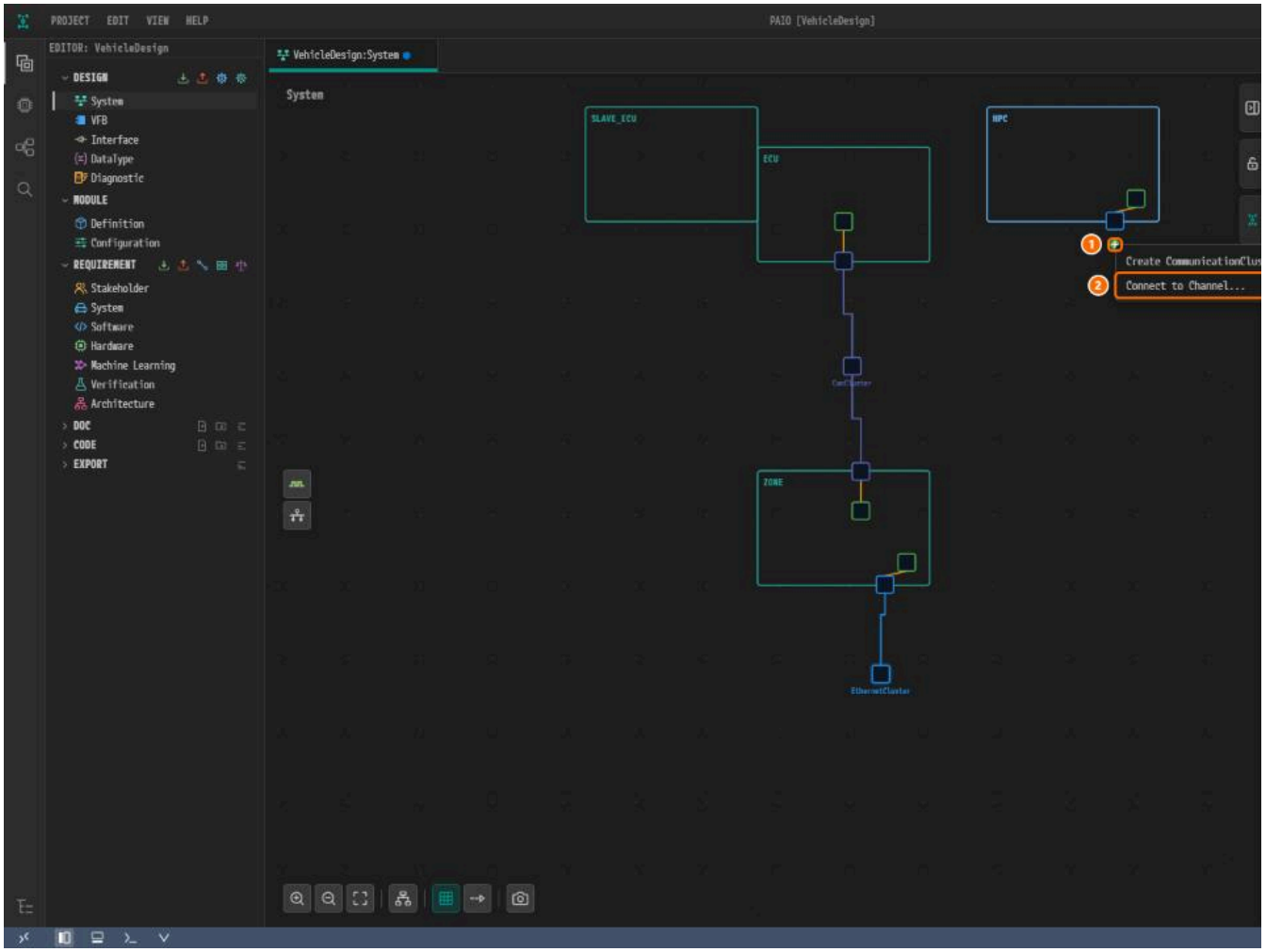


*Figure 4-12. ① The **+** badge, ② **Connect to Channel...**, ③ **EthernetCluster**, ④ **PhysicalChannel (VLAN: 13)**.*

> **Caution.** Do not click **Create CommunicationCluster [Ethernet]** here: it creates a second cluster.

5. Click **Auto Layout**, then **Fit to View**. Click the cluster and check that **Connected Connectors (2)** lists `ZONE/EthernetConnector` and `HPC/EthernetCo`

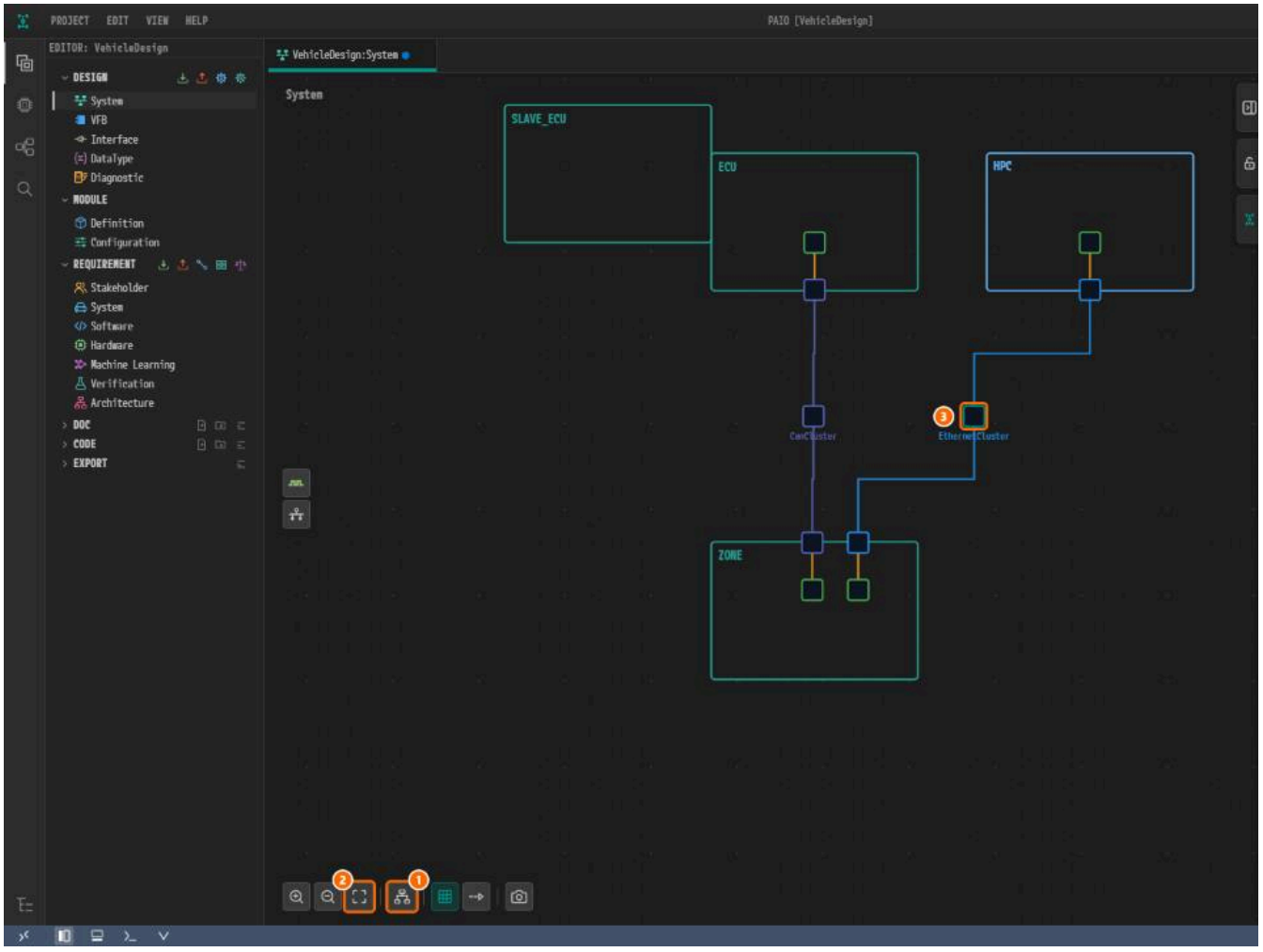


*Figure 4-13. ① **Auto Layout**, ② **Fit to View**, ③ the cluster, ④ the channel and the connected connectors. Your layout may differ slightly.*

6. Click an empty spot of the canvas and press `Ctrl+S`.

Checking Validation and Network Management

1. Click the **V** icon in the footer. The **VALIDATION** panel shows `No validation issues.`.

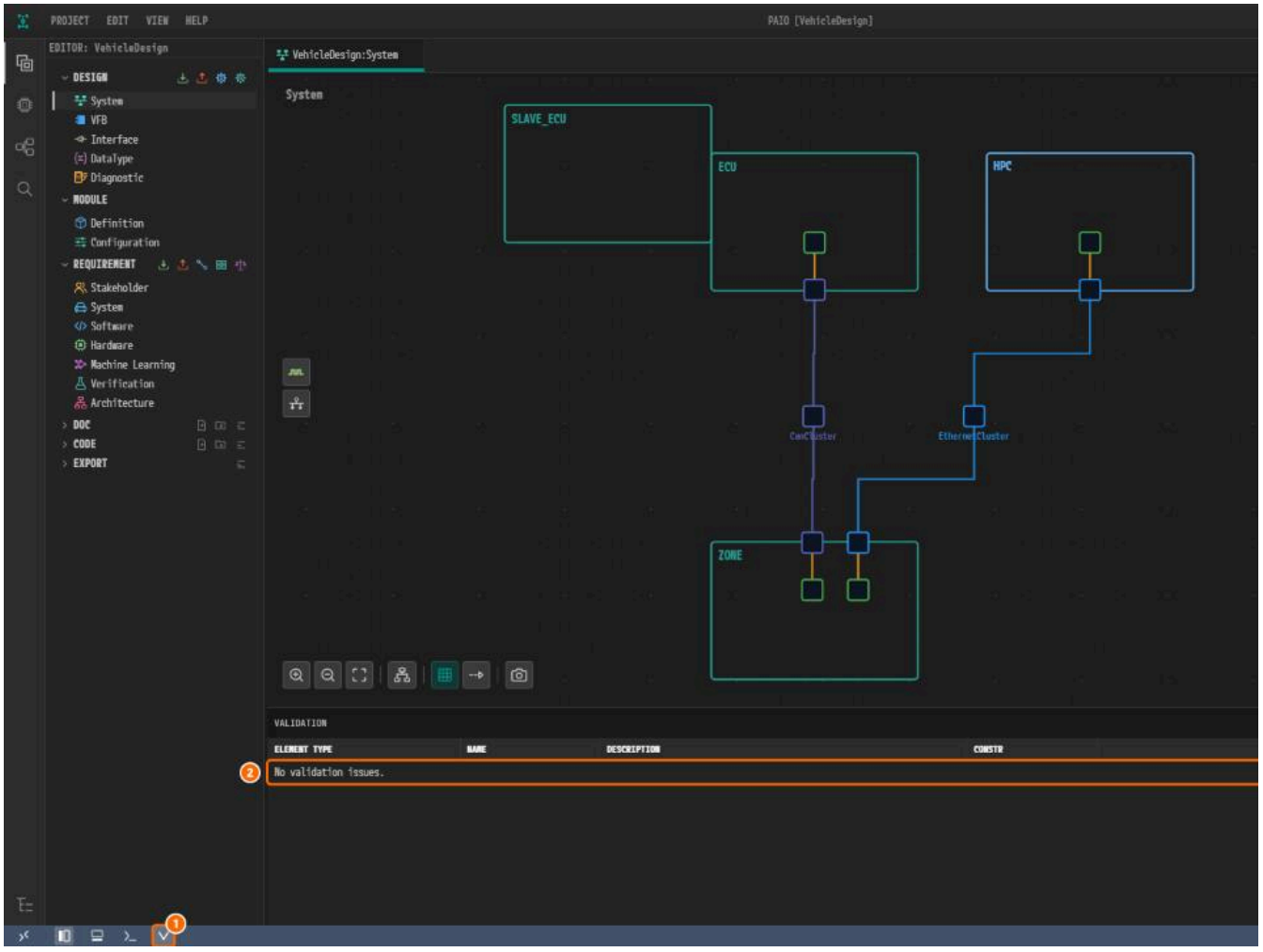


*Figure 4-14. ① The **V** icon, ② `No validation issues.`.*

2. Close the panel. Double-click the `EthernetCluster` square to open **Communication Cluster Configuration [EthernetCluster]**, then click the **NM CLUS**

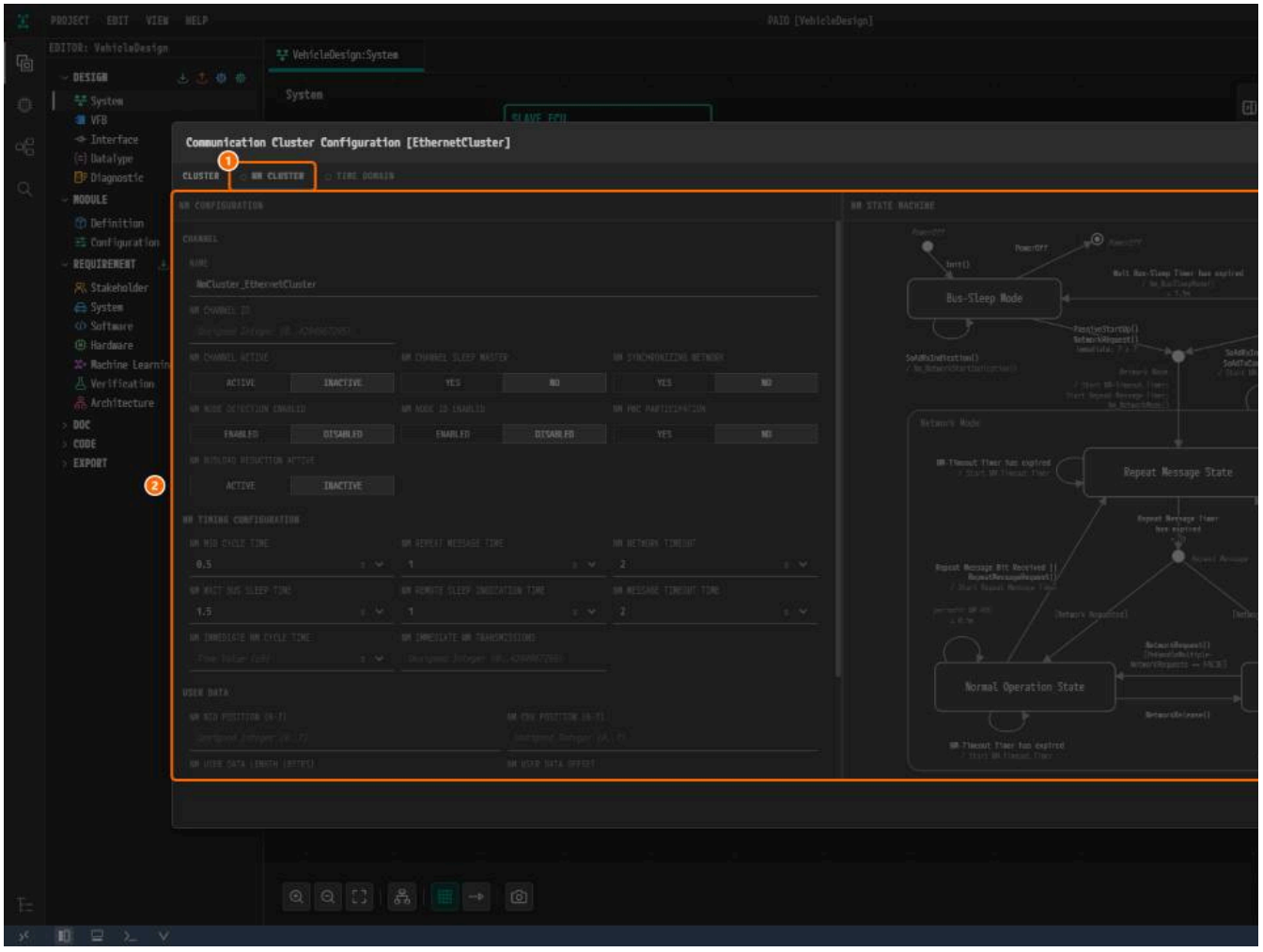


*Figure 4-15. ① **NM CLUSTER** with a hollow dot (off), ② the dimmed preview, which is not stored.*

> **Caution.** Do not choose **Enable** from the right-click menu of the tab. The reference design has no NM.

3. Click **Close**. The **VALIDATION** panel still shows `No validation issues.`.

Why Is It Designed This Way?

- **The cluster is created from `ZONE`.** The connector you start from becomes the first connector of the channel, so the order is `ZONE`, `HPC` as in the refe
- **Addresses go on the connectors.** The editor has no endpoint field. The reference endpoints `ZONE_Endpoint` (`192.168.0.19`), `HPC_Endpoint` (`192.1` (`224.0.0.1`, SOME/IP service discovery) are the three addresses you entered.
- **The `HPC` connector corrects the reference design.** There it has no controller and points to the address of `ZONE`; this tutorial uses the own controller
- **NM stays off.** A new cluster has NM off, and the reference design has no NM.
- **Some values have no input field.** The protocol name `IEEE 802.3`, the category `WIRED`, `WAKE-UP-BY-CONTROLLER-SUPPORTED` `false`, the VLAN name covers the difference.

Checklist

- [ ] `ZONE` has `EthernetCommController` (`11:22:ee:44:55:66`) and `EthernetConnector` (`192.168.0.19`, **FIXED**).
- [ ] `HPC` has `EthernetController` (`00:11:22:33:44:55`) and `EthernetConnector` (`192.168.0.11`, **FIXED**, multicast `224.0.0.1`).
- [ ] **MAC LAYER TYPE** of both controllers is off.
- [ ] `EthernetCluster` has **VLAN ID** `13` and lists `ZONE/EthernetConnector` and `HPC/EthernetConnector`.
- [ ] The **VALIDATION** panel shows `No validation issues.`, and the tab `VehicleDesign:System` carries no modified mark.

Next Step

In LIN Configuration you join the nodes `ECU` and `SLAVE_ECU` with the LIN cluster `LinCluster`.

LIN Configuration

What You Will Do

You join the nodes `ECU` and `SLAVE_ECU` with a LIN bus: a LIN master controller and connector on `ECU`, a LIN slave controller and connector on `SLAVE_ECU` both connectors. The idle timeout `10` ms of the reference design is below the minimum the System editor accepts, so the channel keeps no idle timeout

Before You Begin

- Start after finishing ETHERNET Configuration. `SLAVE_ECU` is still empty, and the **VALIDATION** panel shows `No validation issues.`.
- Open the tab `VehicleDesign:System` and close the **VALIDATION** panel.
- If `HPC` is hidden under the property panel, click **Fit to View** and then **Zoom Out** a few times. The figures show the canvas zoomed out.
- Press the `Tab` key after each value in the property panel.

Where	Item	Value
LIN MASTER (node `ECU`)	**NAME / PROTOCOL VERSION**	`LinController_Master` / `LIN 2.0`
LIN MASTER (node `ECU`)	**TIME BASE / TIME BASE JITTER**	`1` `ms` / `1` `ms`
LIN SLAVE (node `SLAVE_ECU`)	**NAME / PROTOCOL VERSION**	`LinSlaveController` / `LIN 2.0`
LIN CONNECTOR (node `ECU`)	**NAME / CONTROLLER**	`LinConnector_Master` / `LinController_Master`
LIN CONNECTOR (node `SLAVE_ECU`)	**NAME / CONTROLLER**	`LinConnector` / `LinSlaveController`
LIN CLUSTER	**NAME / IDLE TIMEOUT**	`LinCluster` / `10` `ms` (refused, not saved)
LIN CLUSTER	**CHANNEL NAME**	`PhysicalChannel` (default, leave as is)

Creating the LIN Controllers

Adding the Master Controller

1. Right-click an empty spot inside the node `ECU`.
2. Click **Add Controller > LIN Master**.

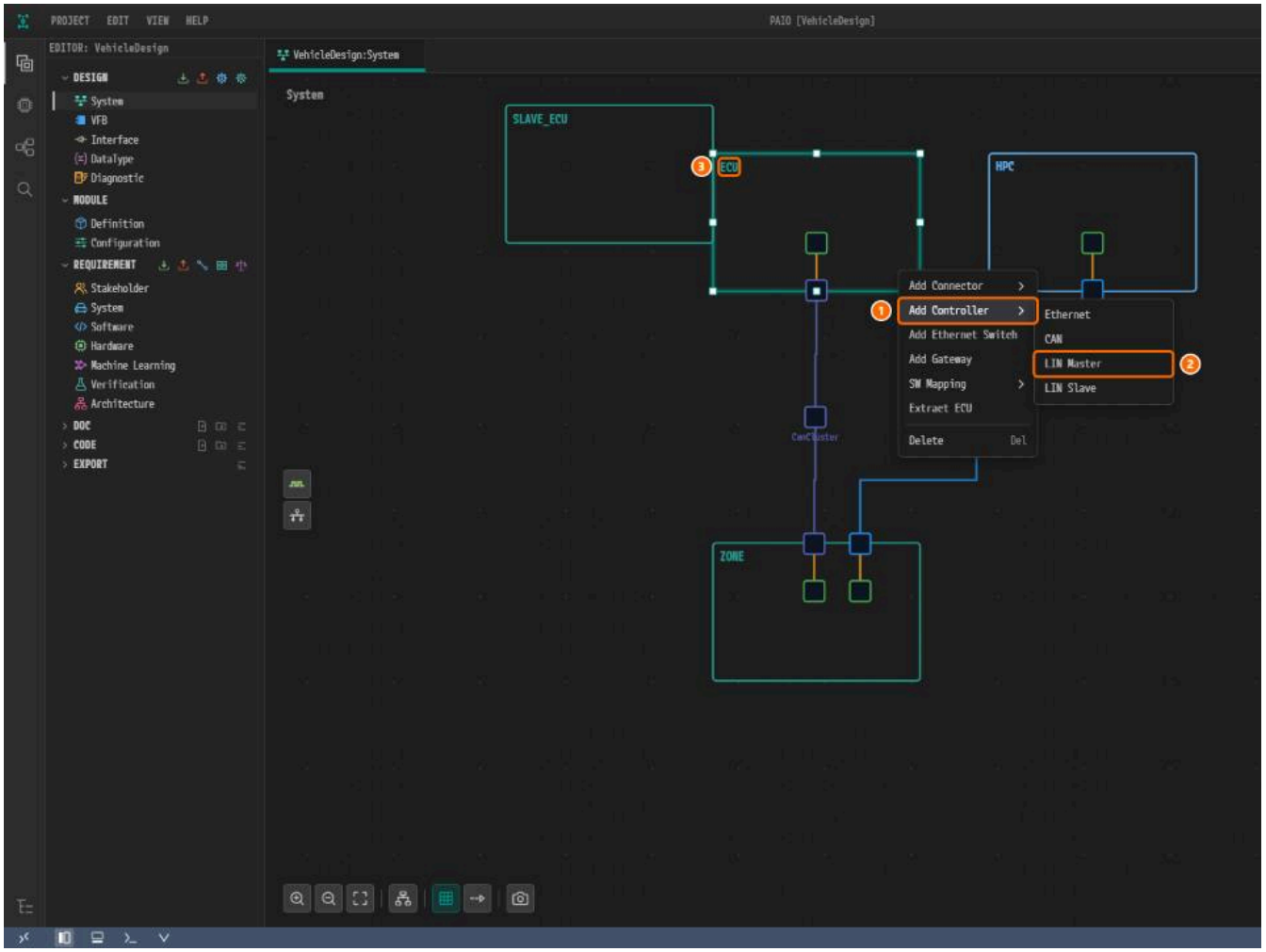


*Figure 5-1. ① **Add Controller**, ② **LIN Master**, ③ the node `ECU`.*

3. Click the new small square inside the node. The panel **LIN MASTER** opens with **NAME** `ECU_LinMaster_Controller_0`. The **V** icon shows the badge `1` u

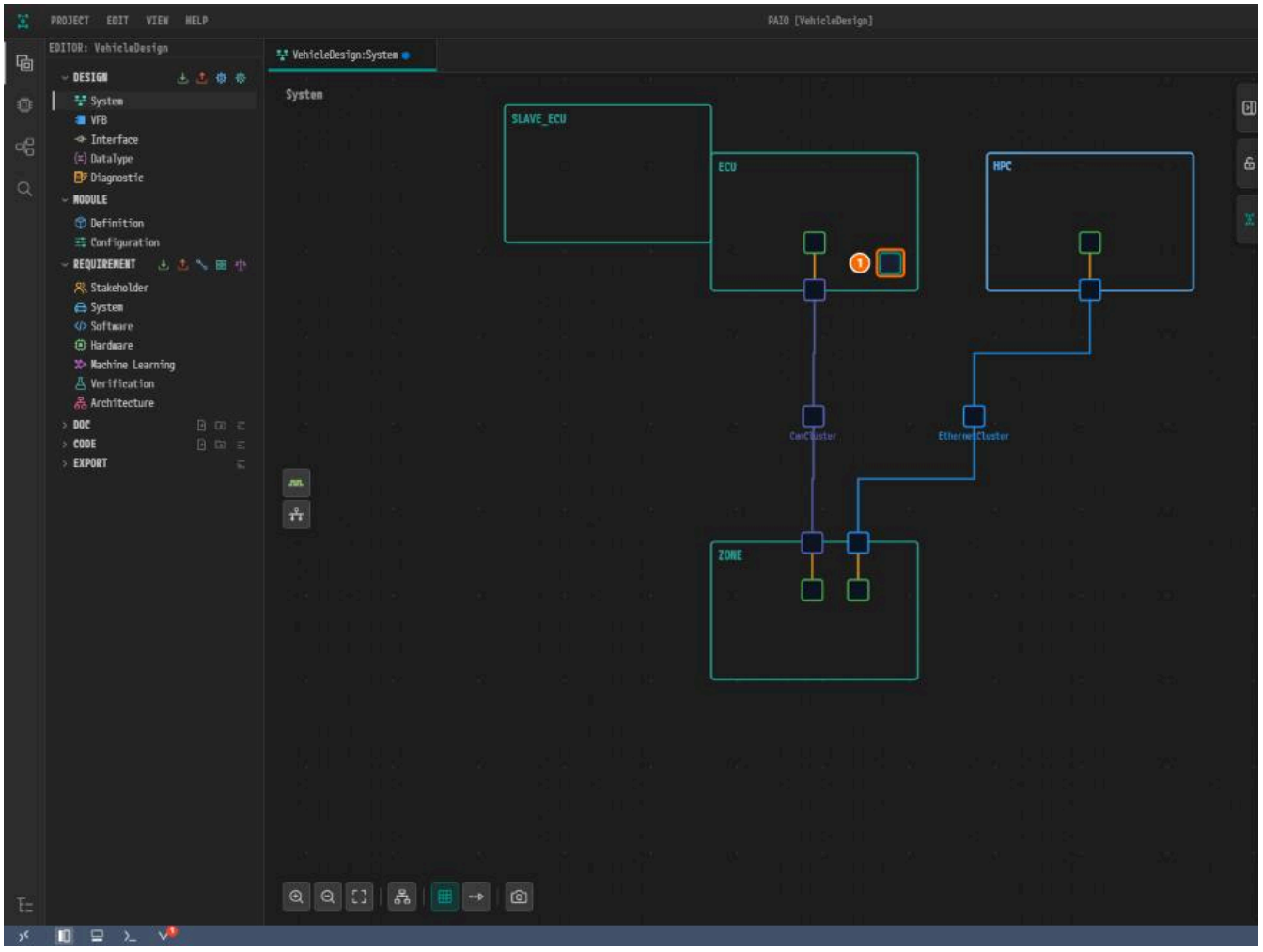


*Figure 5-2. ① The controller, ② **LIN MASTER**, ③ **NAME**, ④ **PROTOCOL VERSION**, ⑤ **TIME BASE**, ⑥ **TIME BASE JITTER**.*

Entering the Name, Protocol Version and Time Base

1. Type `LinController_Master` into **NAME**.
2. Click `LIN 2.0` under **PROTOCOL VERSION**. The badge on the **V** icon disappears.
3. Change the unit of **TIME BASE** from `s` to `ms` first, then type `1`.
4. Do the same for **TIME BASE JITTER**: unit `ms`, value `1`. Leave **WAKE-UP BY CONTROLLER** and **LIN SLAVES** alone.

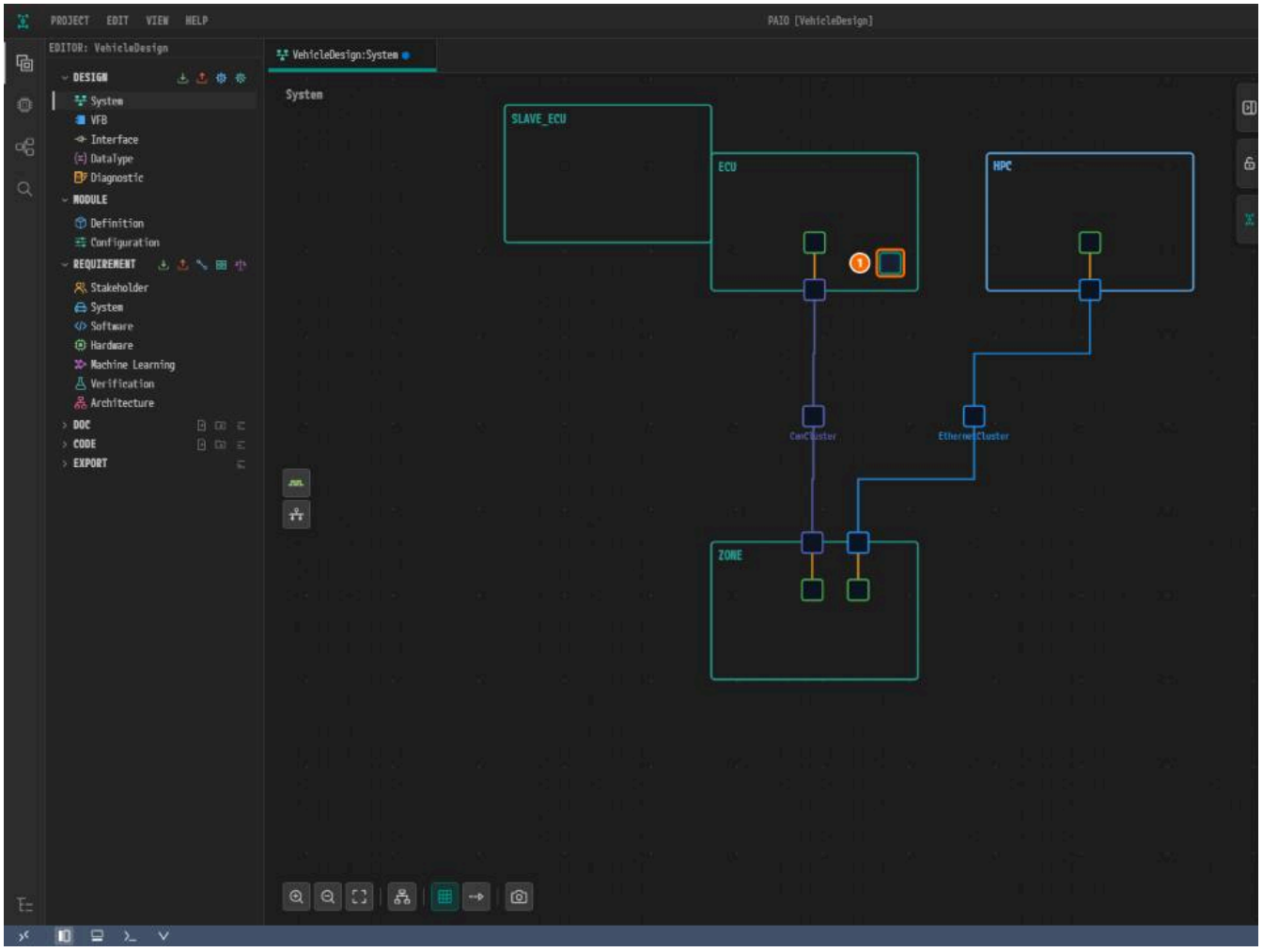


*Figure 5-3. ① The controller, ② **NAME**, ③ **PROTOCOL VERSION**, ④ **TIME BASE**, ⑤ **TIME BASE JITTER**.*

> **Caution.** Change the unit before typing the number. The unit defaults to `s`, and changing it after typing may recalculate the number.

Creating the Slave Controller on SLAVE_ECU

1. Right-click an empty spot inside `SLAVE_ECU`.
2. Click **Add Controller > LIN Slave**.

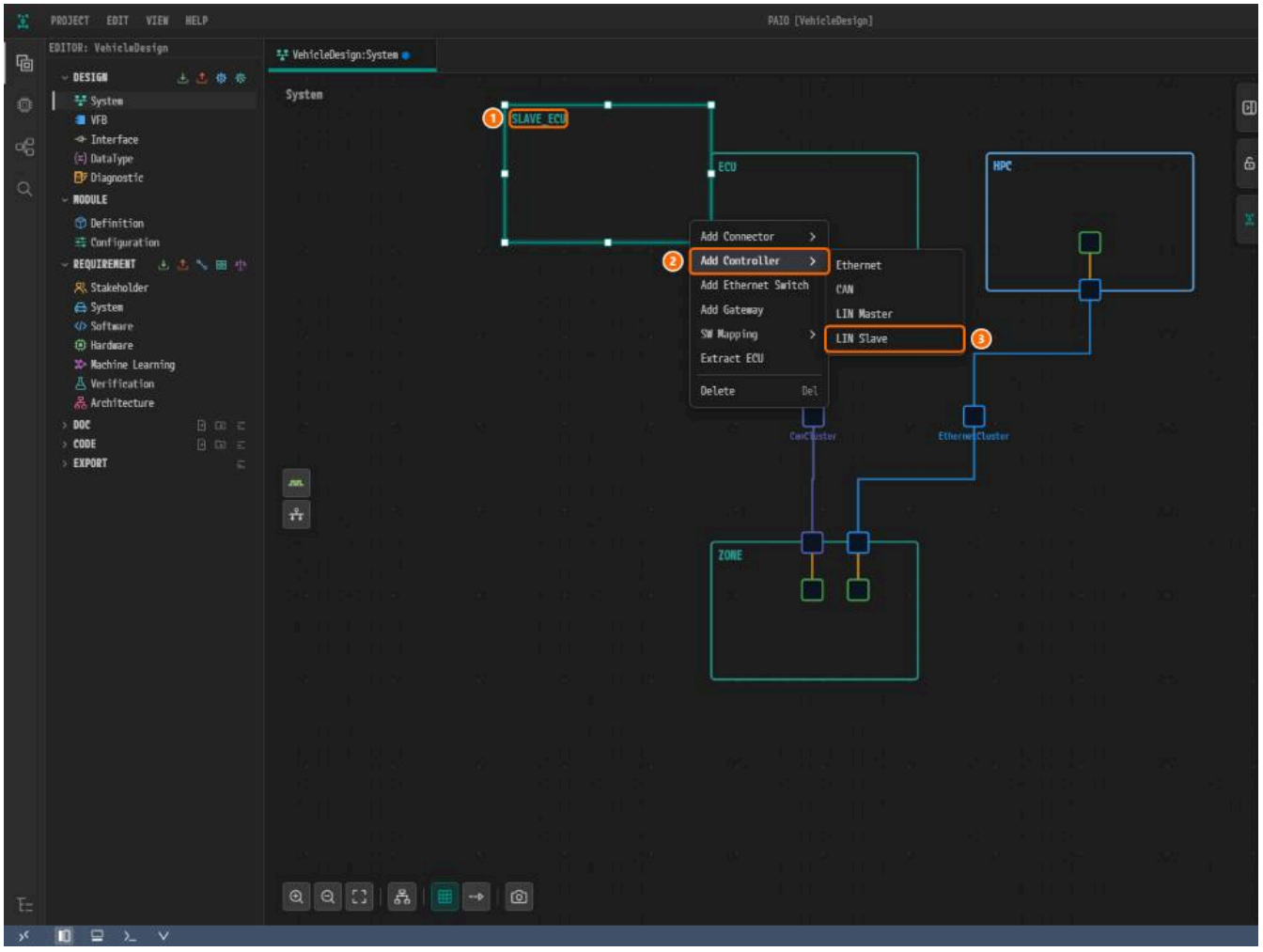


*Figure 5-4. ① The node `SLAVE_ECU`, ② **Add Controller**, ③ **LIN Slave**.*

3. Click the new square. The panel **LIN SLAVE** opens.

4. Type `LinSlaveController` into **NAME**.

5. Click `LIN 2.0` under **PROTOCOL VERSION**. Leave the rest as it is.

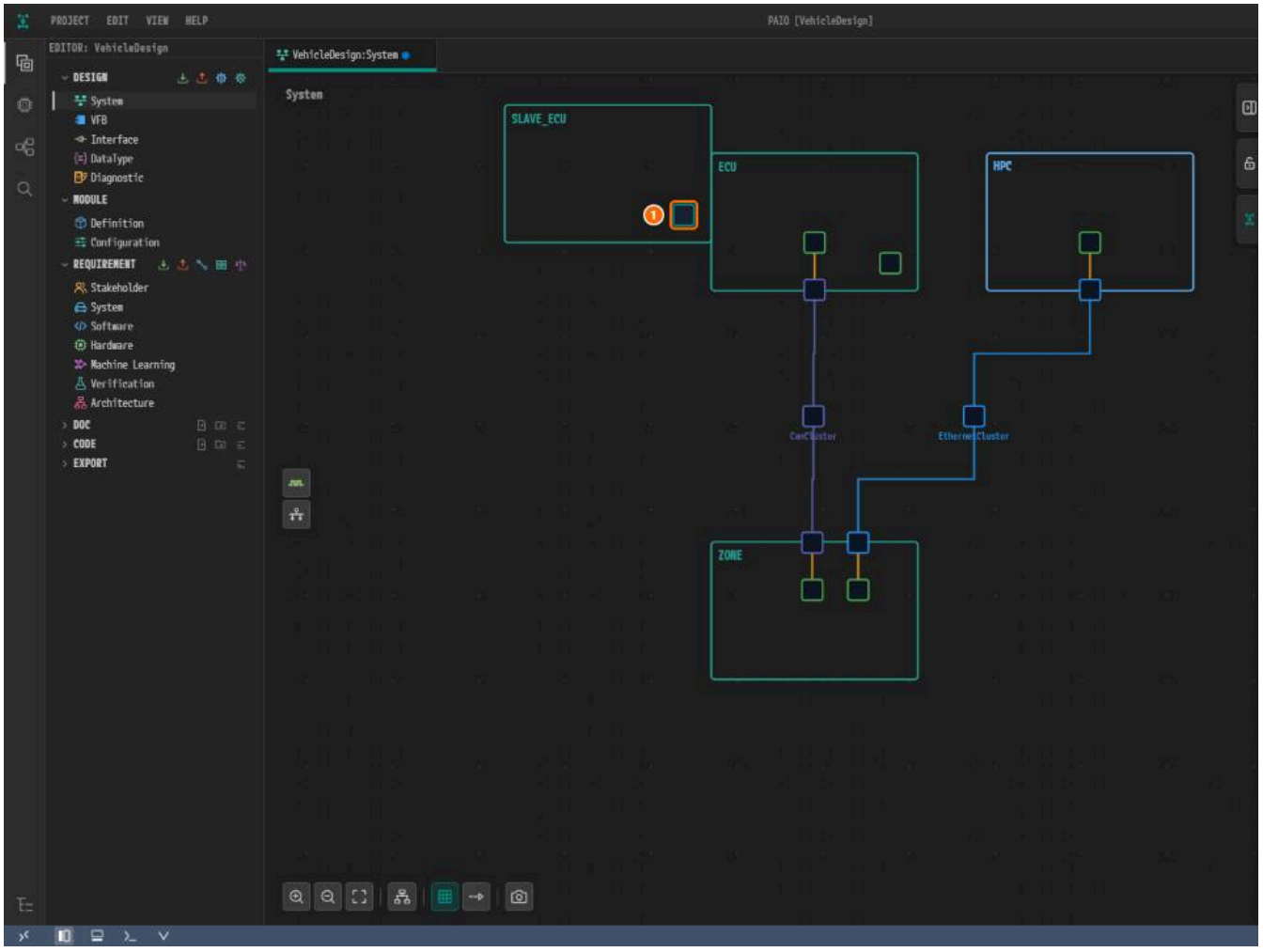


*Figure 5-5. ① The controller, ② **NAME**, ③ **PROTOCOL VERSION**.*

The slave panel has no **TIME BASE** fields.

Creating the LIN Connectors

Adding a Connector

1. Right-click an empty spot inside `ECU`.
2. Click **Add Connector > LIN**.

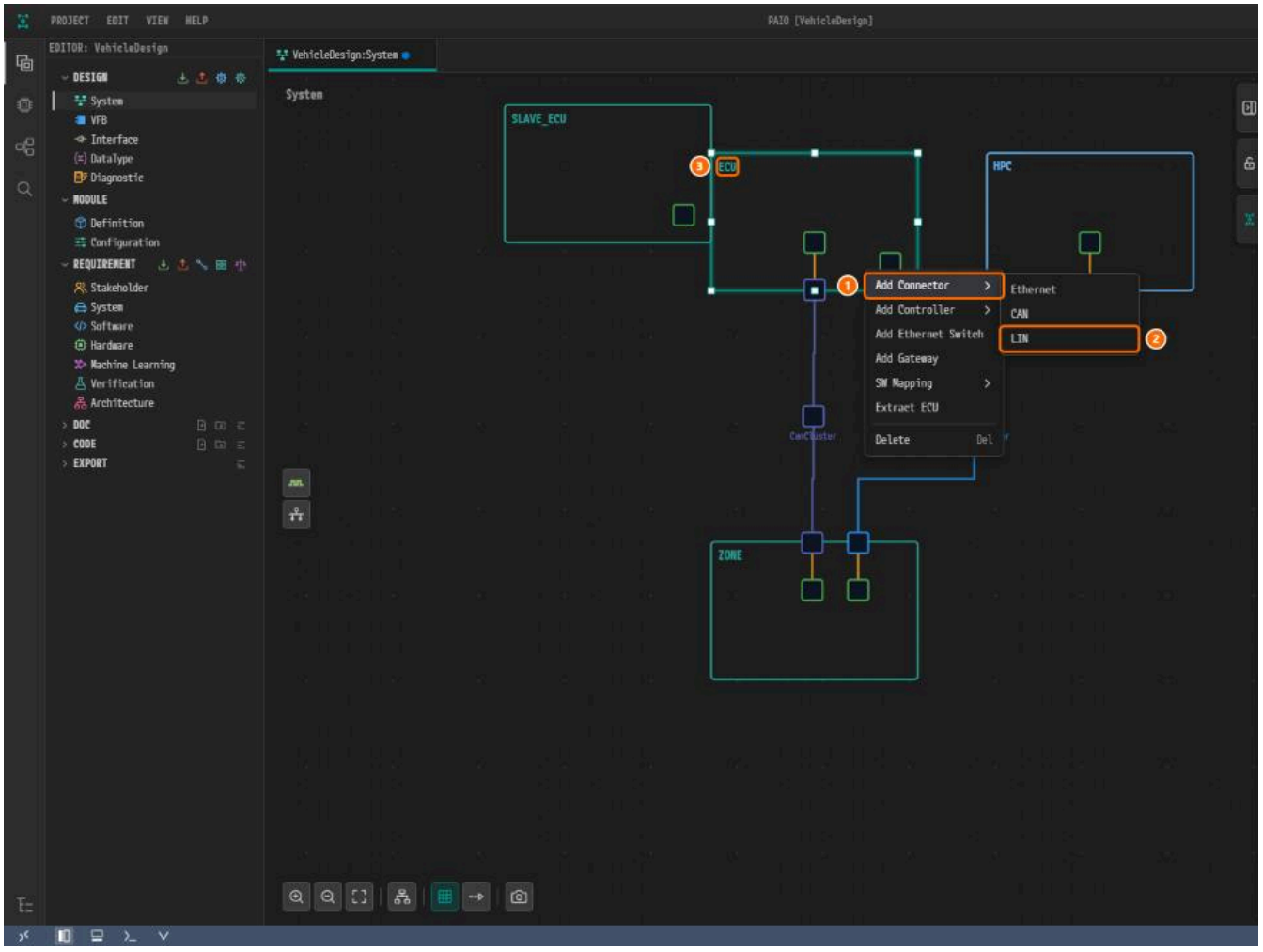


*Figure 5-6. ① **Add Connector**, ② **LIN**, ③ the node `ECU`.*

3. Click the new connector square on the bottom edge of `ECU`, to the right of `CanConnector`. The panel **LIN CONNECTOR** opens. **CONTROLLER** is red with

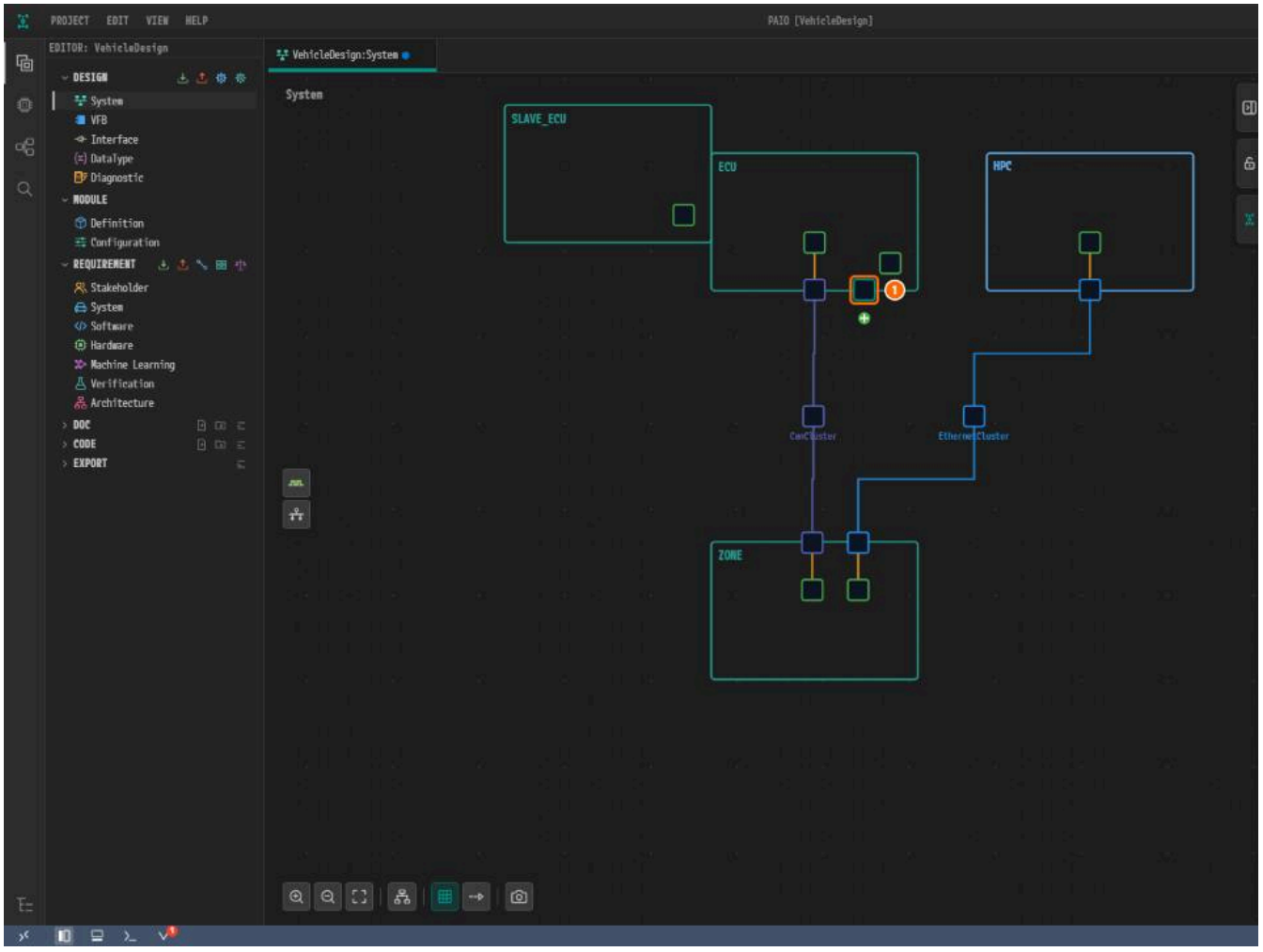


*Figure 5-7. ① The connector, ② **LIN CONNECTOR**, ③ **NAME**, ④ **CONTROLLER**, ⑤ **LIN CONNECTOR SETTINGS**.*

Setting the Name and Controller

1. Type `LinConnector_Master` into **NAME**.
2. Click **CONTROLLER** and click `LinController_Master`, the only item in the list.

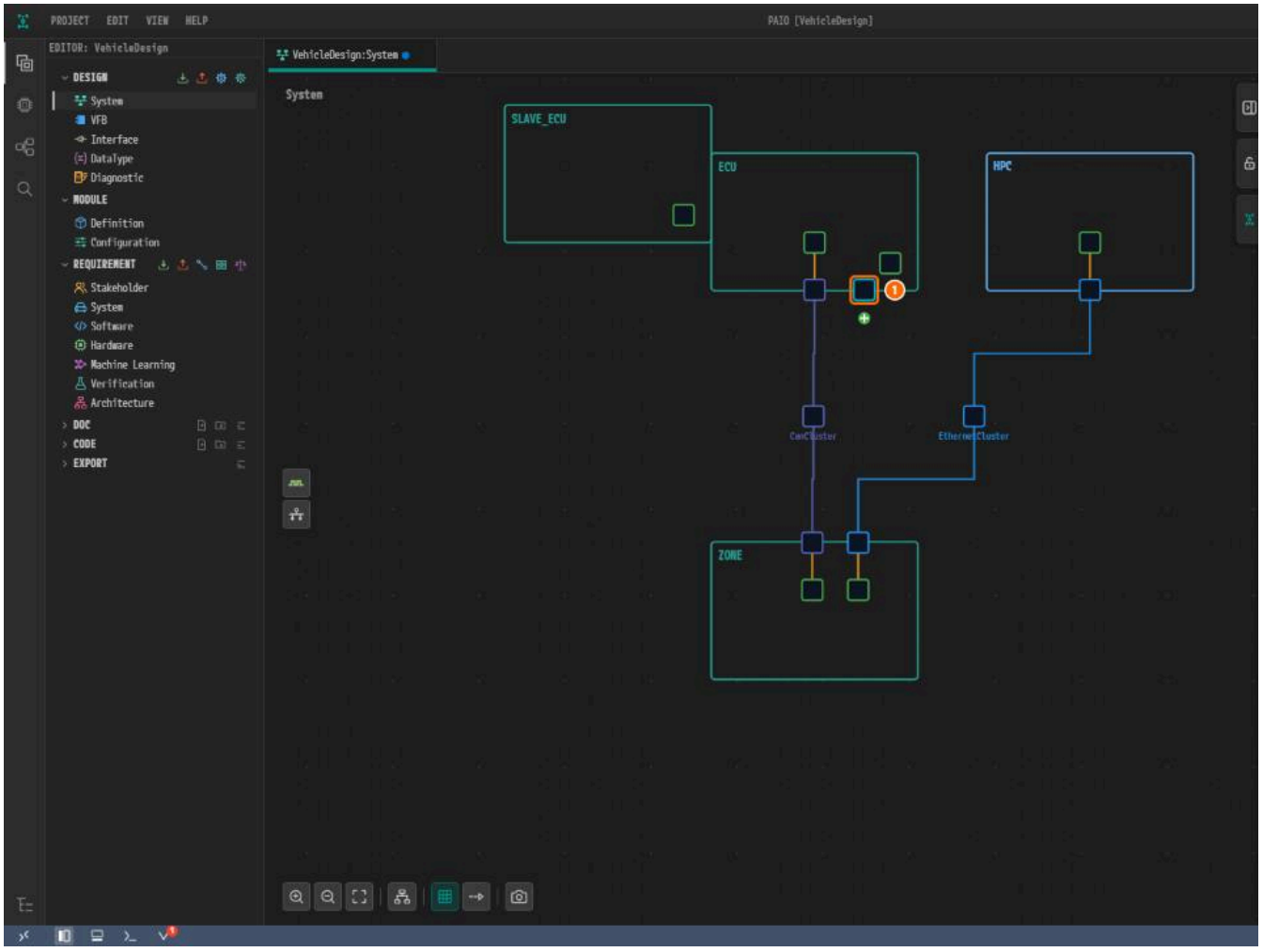


*Figure 5-8. ① The connector, ② **NAME**, ③ **CONTROLLER**, ④ `LinController_Master`.*

> **Note.** The connector menu has one **LIN** item. The controller you select decides whether the connector is on the master or the slave side.

Creating a Connector on SLAVE_ECU

1. Right-click an empty spot inside `SLAVE_ECU`, click **Add Connector > LIN**, and click the new connector.
2. Type `LinConnector` into **NAME**.
3. Select `LinSlaveController` in **CONTROLLER**.

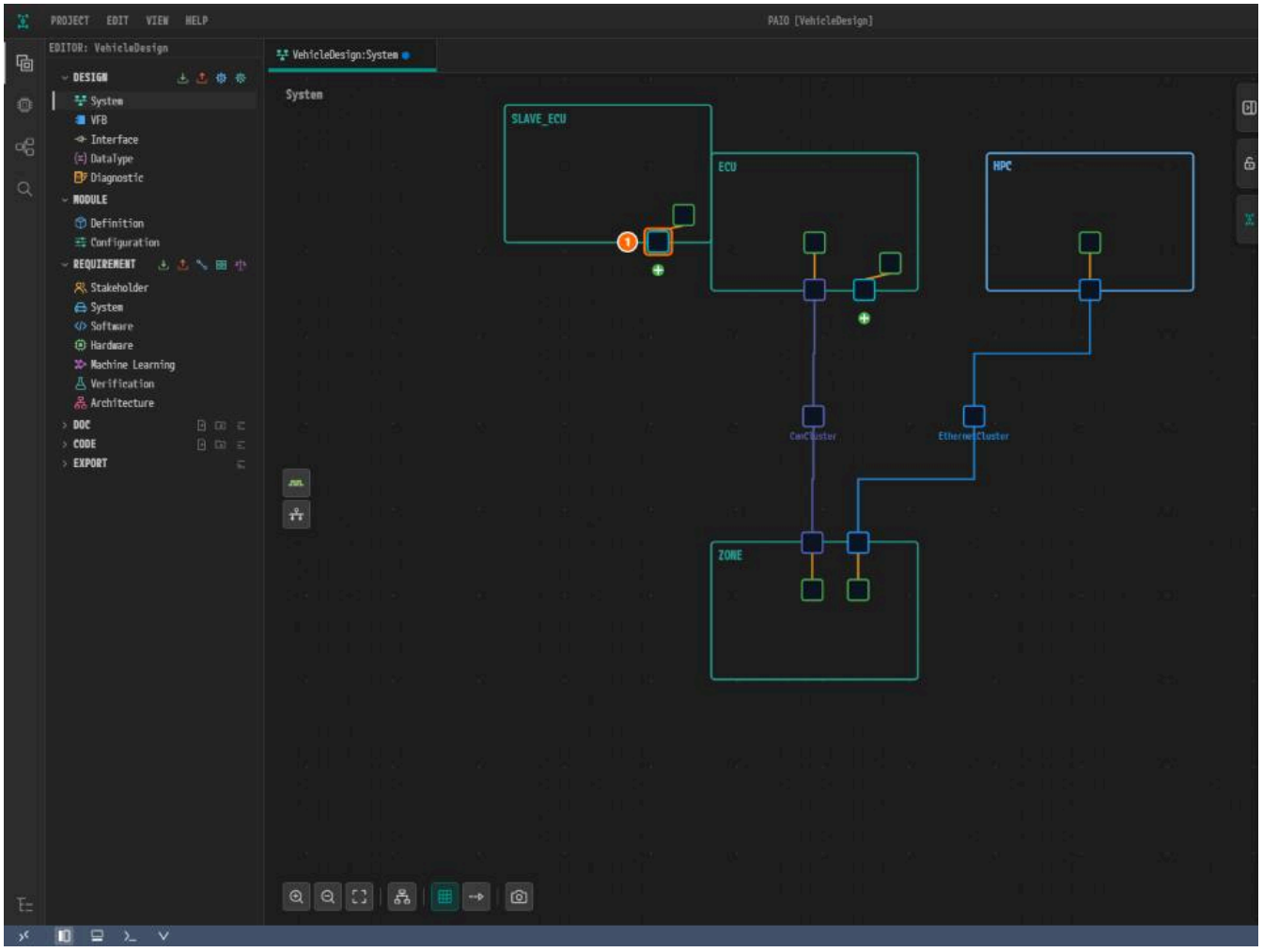


*Figure 5-9. ① The connector of `SLAVE_ECU`, ② **NAME**, ③ **CONTROLLER**.*

Creating the LIN Cluster

Creating the Cluster from the SLAVE_ECU Connector

Create the cluster from `SLAVE_ECU`, so that its connector comes first in the channel, as in the reference design.

1. Click the green **+** badge below `LinConnector` of `SLAVE_ECU`.
2. Click **Create CommunicationCluster [Lin]**.

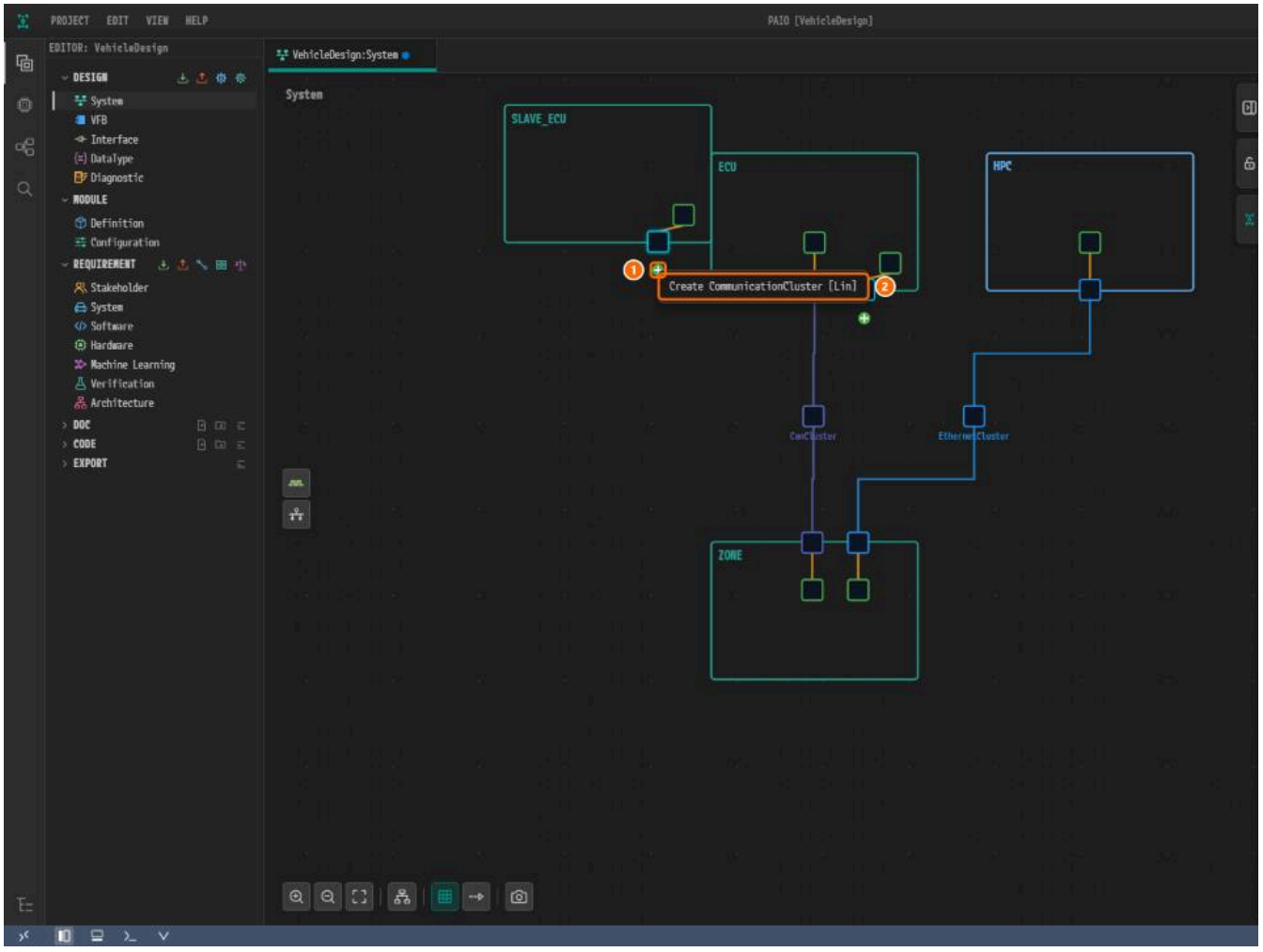


*Figure 5-10. ① The **+** badge, ② **Create CommunicationCluster [Lin]**.*

3. If the new cluster square is not visible, click **Fit to View** (and **Zoom Out**).

4. Click the cluster square. The panel **LIN CLUSTER** opens.

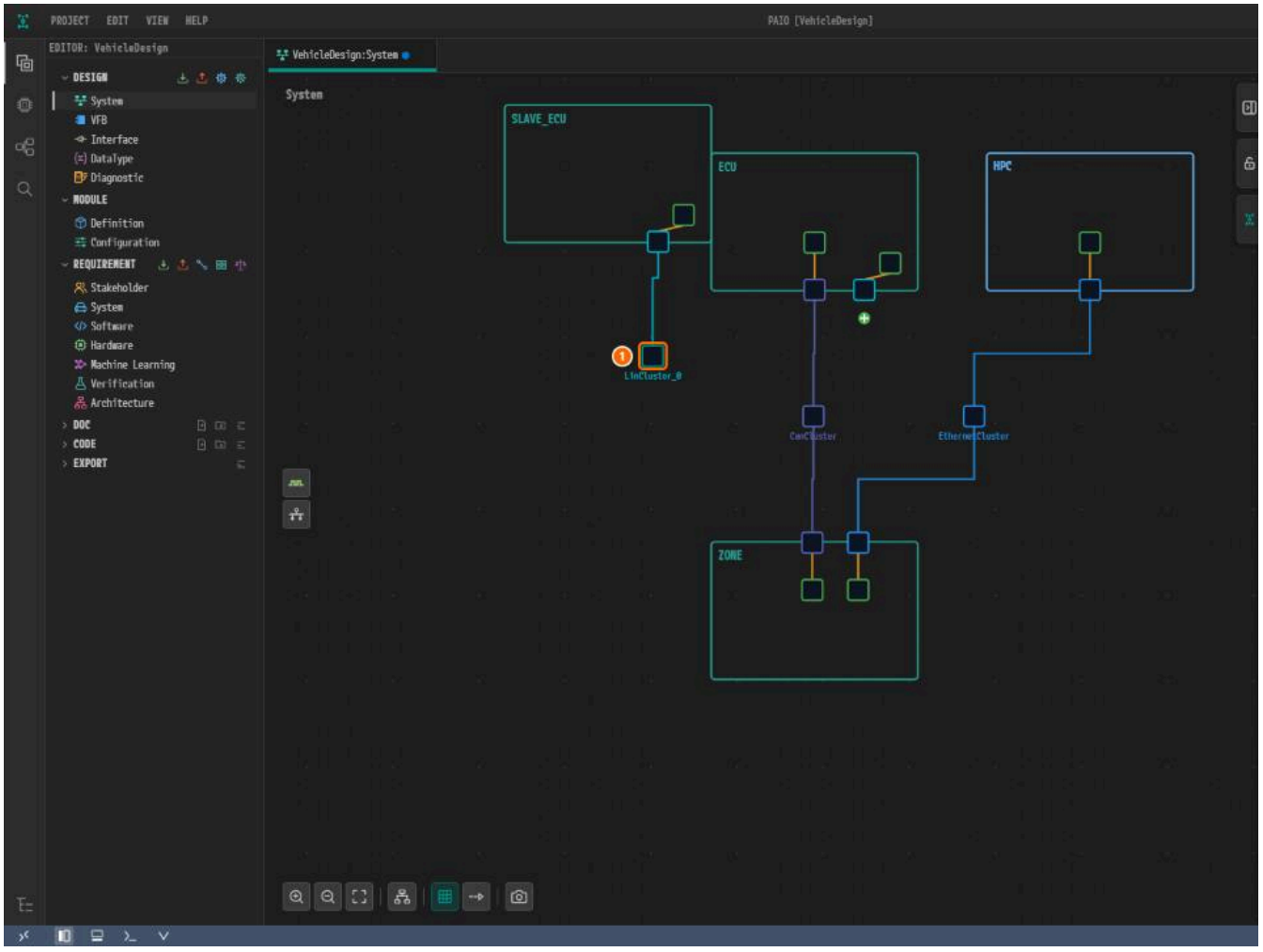


*Figure 5-11. ① The cluster, ② **NAME**, ③ **BAUDRATE (BPS)**, ④ **CHANNEL NAME**, ⑤ **IDLE TIMEOUT**.*

5. Type `LinCluster` into **NAME**.

6. Set the unit of **IDLE TIMEOUT** to `ms` and type `10`. The field turns red with `Must be at least 0.1` and the value is not saved. Leave the other fields a

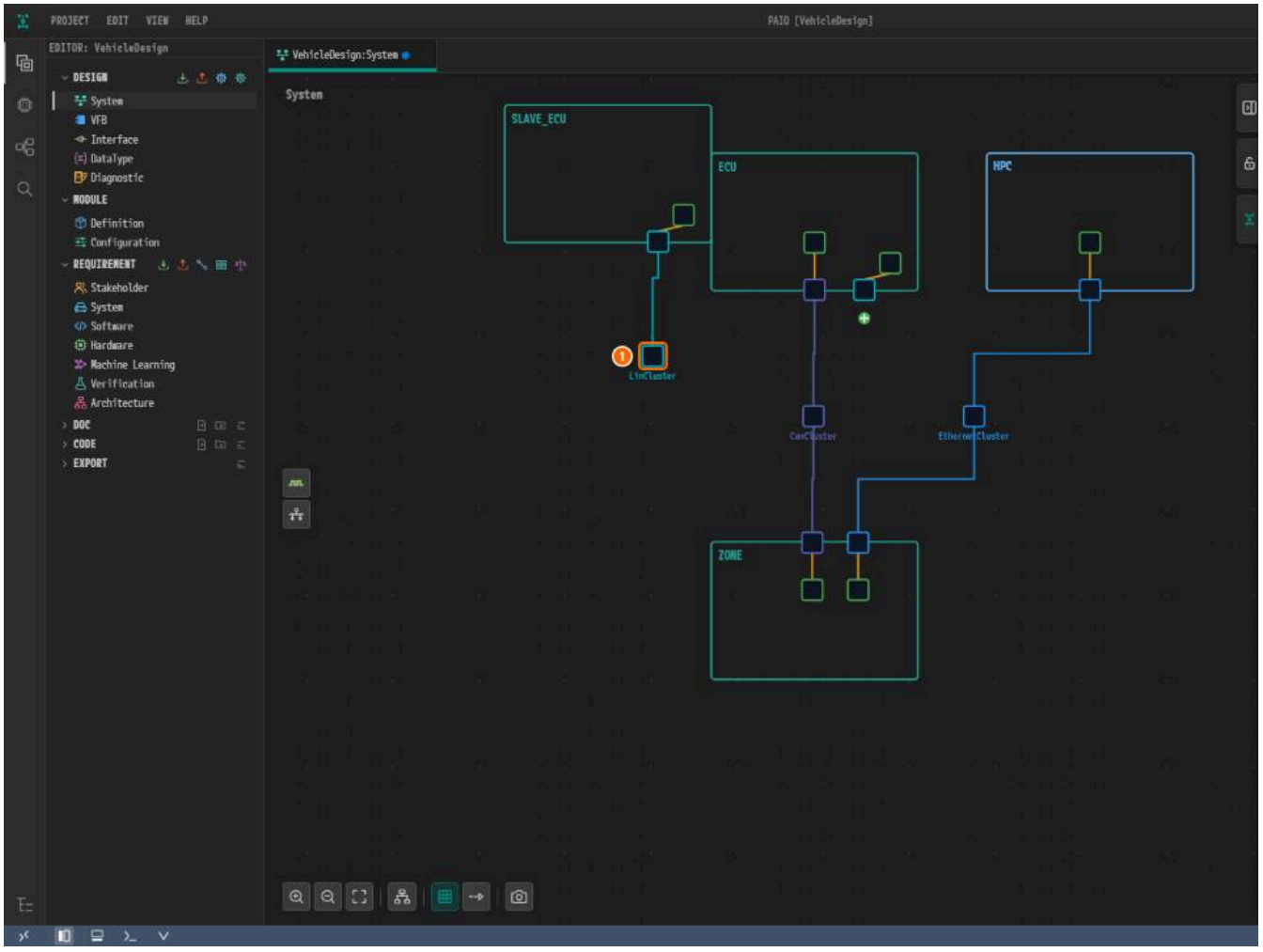


*Figure 5-12. ① The cluster, ② **NAME** = `LinCluster`, ③ **IDLE TIMEOUT** refused.*

If the field accepted `10` without turning red, the unit was still `s`: delete the number and press `Tab` to leave it empty.

> **Note.** No **BAUDRATE (BPS)** button is on, but the saved cluster holds `19200`. See ARXML Export.

Connecting the ECU Connector to the Channel

1. Click the **+** badge below `LinConnector_Master` of `ECU`.
2. Click **Connect to Channel... > LinCluster > PhysicalChannel**.

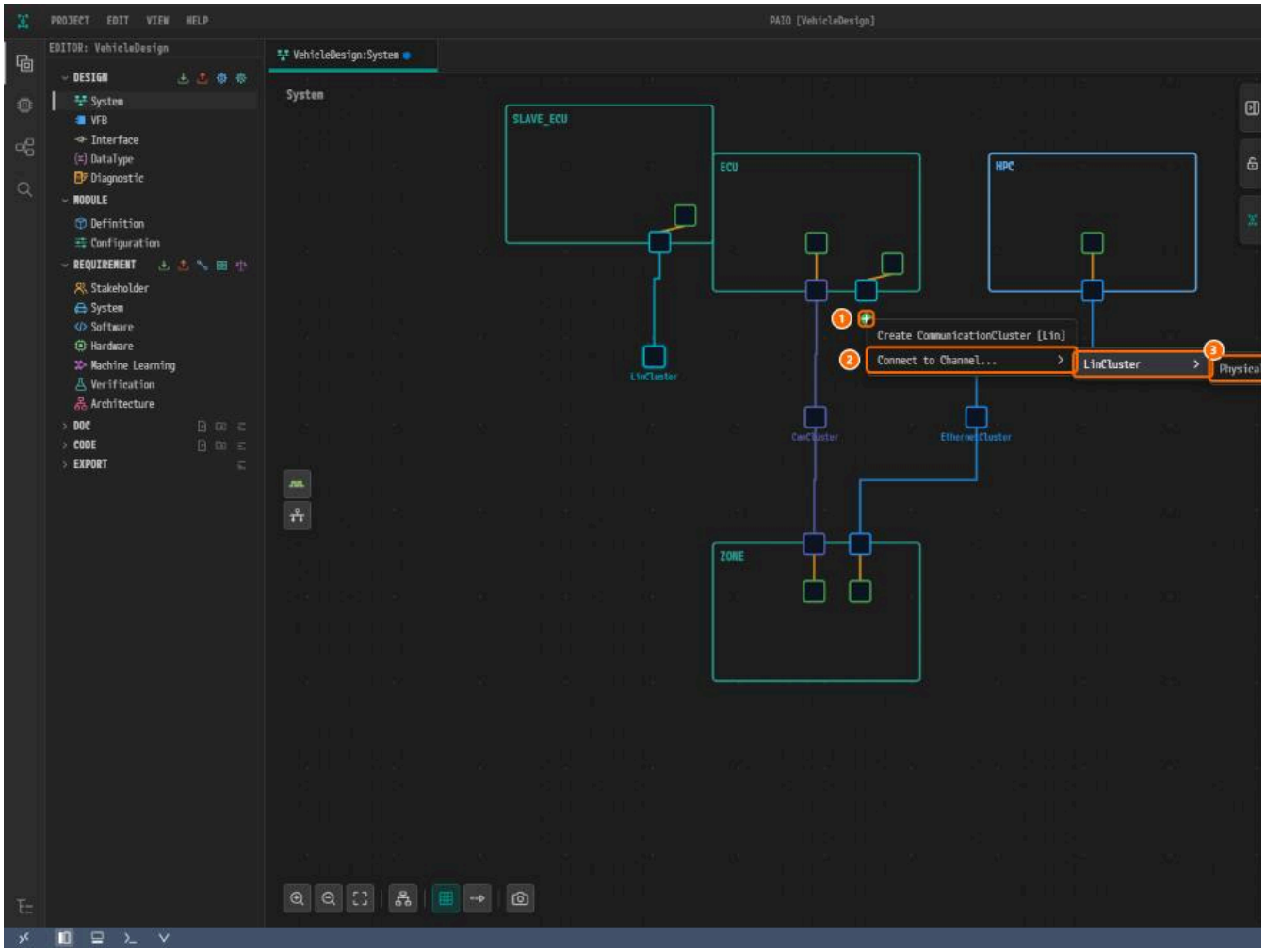


*Figure 5-13. ① The **+** badge, ② **Connect to Channel...**, ③ **LinCluster**, ④ **PhysicalChannel**.*

> **Caution.** Do not click **Create CommunicationCluster [Lin]** here: it makes a second cluster.

Tidying the Layout and Saving

1. Click **Auto Layout**, then **Fit to View** (and **Zoom Out** if needed).
2. Click the LIN cluster and check that **IDLE TIMEOUT** is empty and **Connected Connectors (2)** lists `SLAVE_ECU/LinConnector` and `ECU/LinConnector_M`

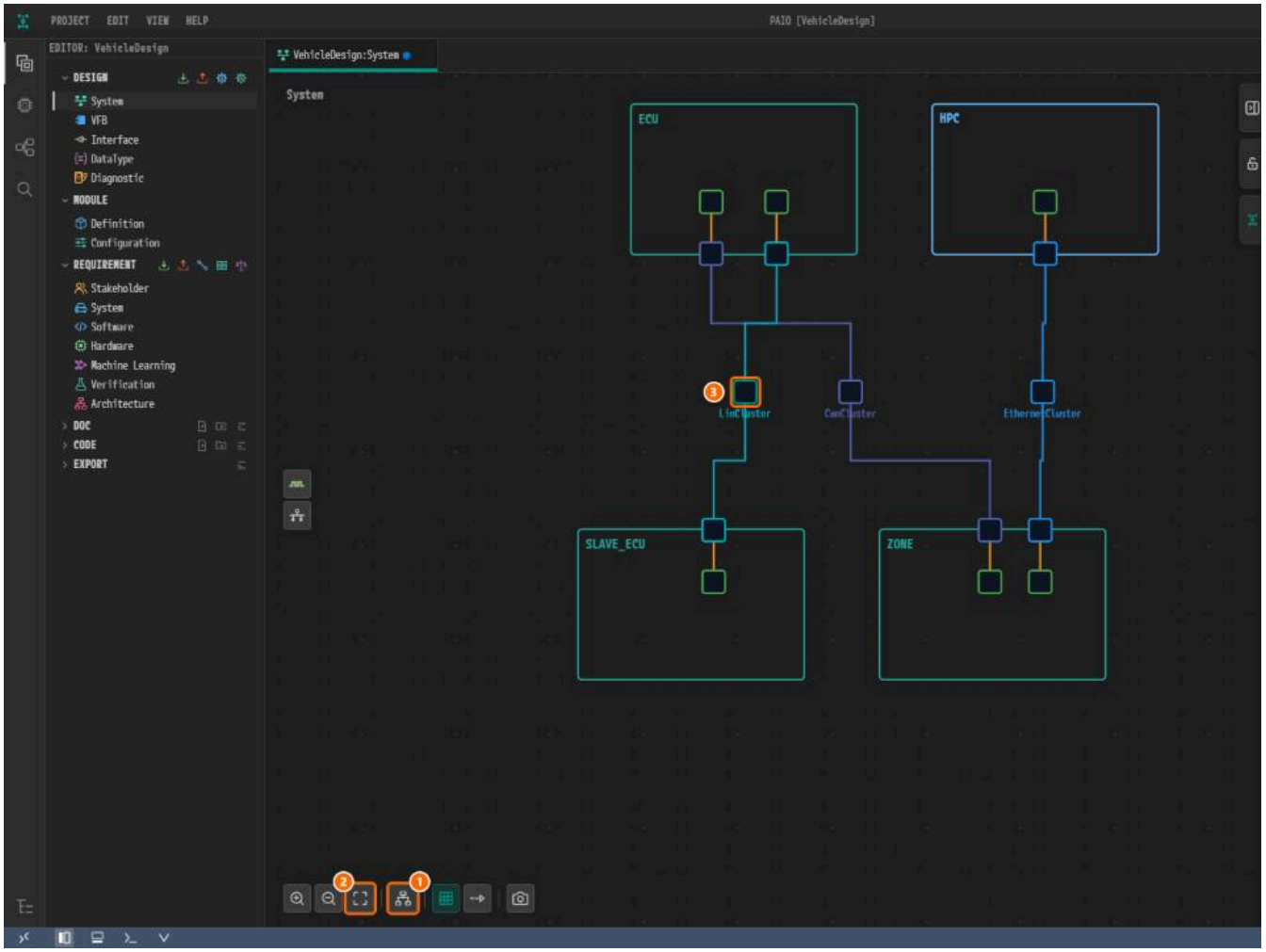


*Figure 5-14. ① **Auto Layout**, ② **Fit to View**, ③ the cluster, ④ **CHANNEL NAME**, **IDLE TIMEOUT** and the two connectors.*

3. Click the empty canvas and press `Ctrl+S`.

Viewing the Validation Result

1. Click the **V** icon. The **VALIDATION** panel shows `No validation issues.`.

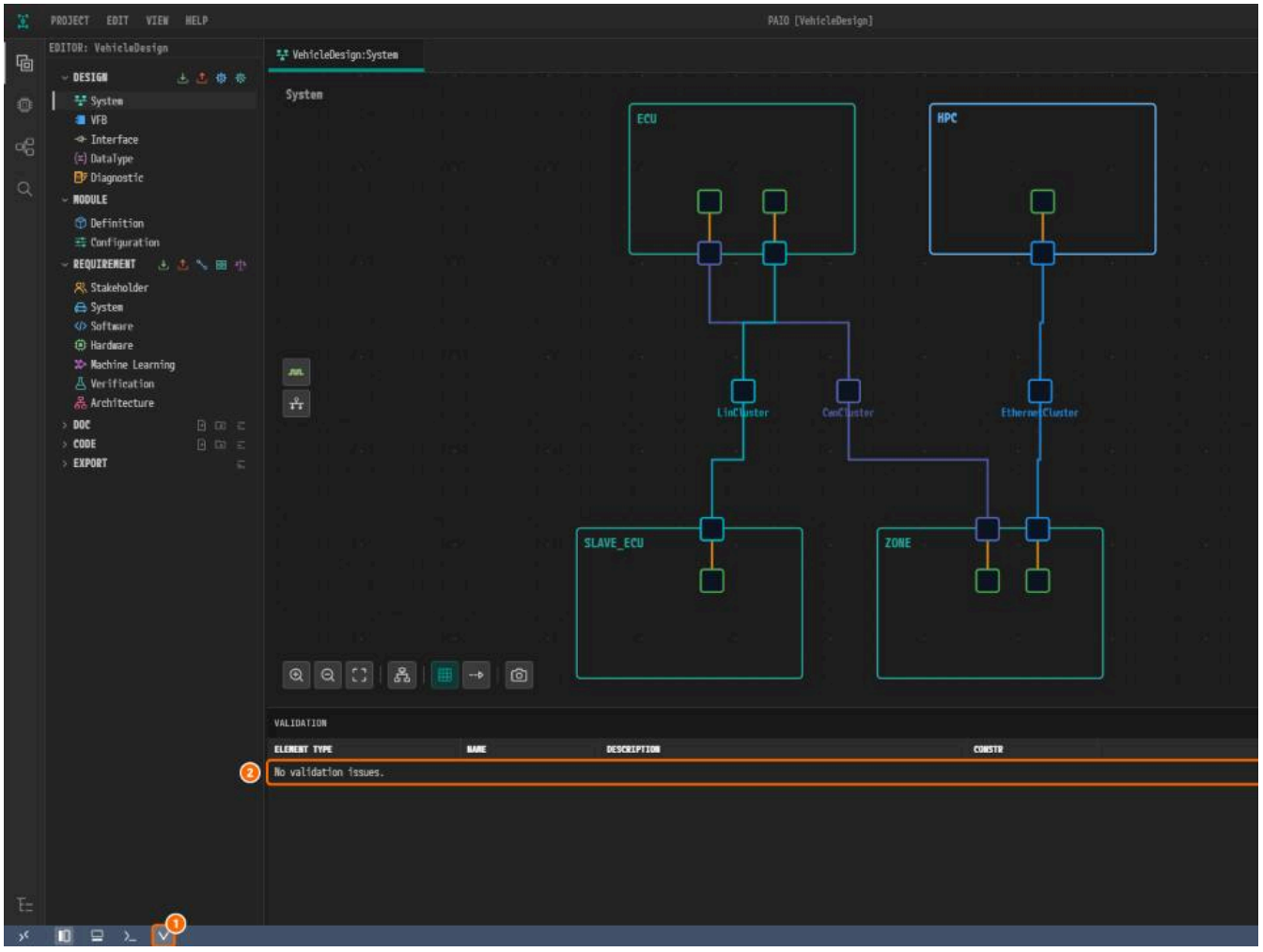


*Figure 5-15. ① The **V** icon, ② `No validation issues.`.*

A LIN cluster has no NM, and its TP is off, so there is nothing to check in the cluster configuration window.

Values Not Entered in This Chapter

The difference in the export result is covered in ARXML Export.

Value in the reference design	Location	System editor
`WAKE-UP-BY-CONTROLLER-SUPPORTED` = `true`	`LinController_Master` of `ECU`	**WAKE-UP BY CONTROLLER** is not clicked, so no value is saved
`BUS-IDLE-TIMEOUT-PERIOD` = `0.01`	`PhysicalChannel` of `LinCluster`	**IDLE TIMEOUT** accepts only `0.1` seconds or more

Why Is It Designed This Way?

- **Order of the connectors.** The reference design lists `SLAVE_ECU` first, then `ECU`, so the cluster is created from `SLAVE_ECU`.
- **Master and slave.** One LIN master sets the schedule and the slaves follow it; the reference design puts the master on `ECU`.
- **Time base only on the master.** `LinController_Master` has `0.001` seconds for both; `LinSlaveController` has only a protocol version.
- **Idle timeout empty. IDLE TIMEOUT** follows the AUTOSAR range of `LinIfBusIdleTimeoutPeriod` (`0.1` seconds or more).
- **No NADs, schedule tables or frames.** The reference design has none.

Checklist

- [ ] `ECU` has `LinController_Master` (`LIN 2.0`, `1` `ms`, `1` `ms`) and `LinConnector_Master` (**CONTROLLER** = `LinController_Master`).
- [ ] `SLAVE_ECU` has `LinSlaveController` (`LIN 2.0`) and `LinConnector` (**CONTROLLER** = `LinSlaveController`).
- [ ] `LinCluster` has an empty **IDLE TIMEOUT** and lists `SLAVE_ECU/LinConnector` and `ECU/LinConnector_Master`, in that order, on `PhysicalChannel`.
- [ ] **CONNECTED CHANNEL** of both connectors is `LinCluster / PhysicalChannel`.
- [ ] No modified mark, and `No validation issues.`.

Next Step

In Validation you validate the finished topology and jump from an error row to its element.

Validation

What You Will Do

You put three defects into the topology finished in LIN Configuration, read how the **VALIDATION** panel reports them, and fix them until the panel shows `N`

Before You Begin

- Start after finishing LIN Configuration. The **VALIDATION** panel shows `No validation issues.`.
- Make the three defects below in the System editor (**EDITOR** > **DESIGN** > **System**) and save with `Ctrl+S`. Then close the tab `VehicleDesign:System` w

Element	Field	Original value	Make the defect
`EthernetConnector` of `HPC`	**UNICAST IPV4 ADDRESS**	`192.168.0.11`	Type `224.0.0.1` and press `Tab`
`LinConnector` of `SLAVE_ECU`	**CONTROLLER**	`LinSlaveController`	Click the **×** (Clear Controller) inside the field
`LinSlaveController` of `SLAVE_ECU`	**PROTOCOL VERSION**	`LIN 2.0`	Click `LIN 2.0` again to turn it off

Reading the VALIDATION Panel

1. PAIO validates the open project by itself, so the **V** icon in the footer already carries the red badge `3`. Click it.

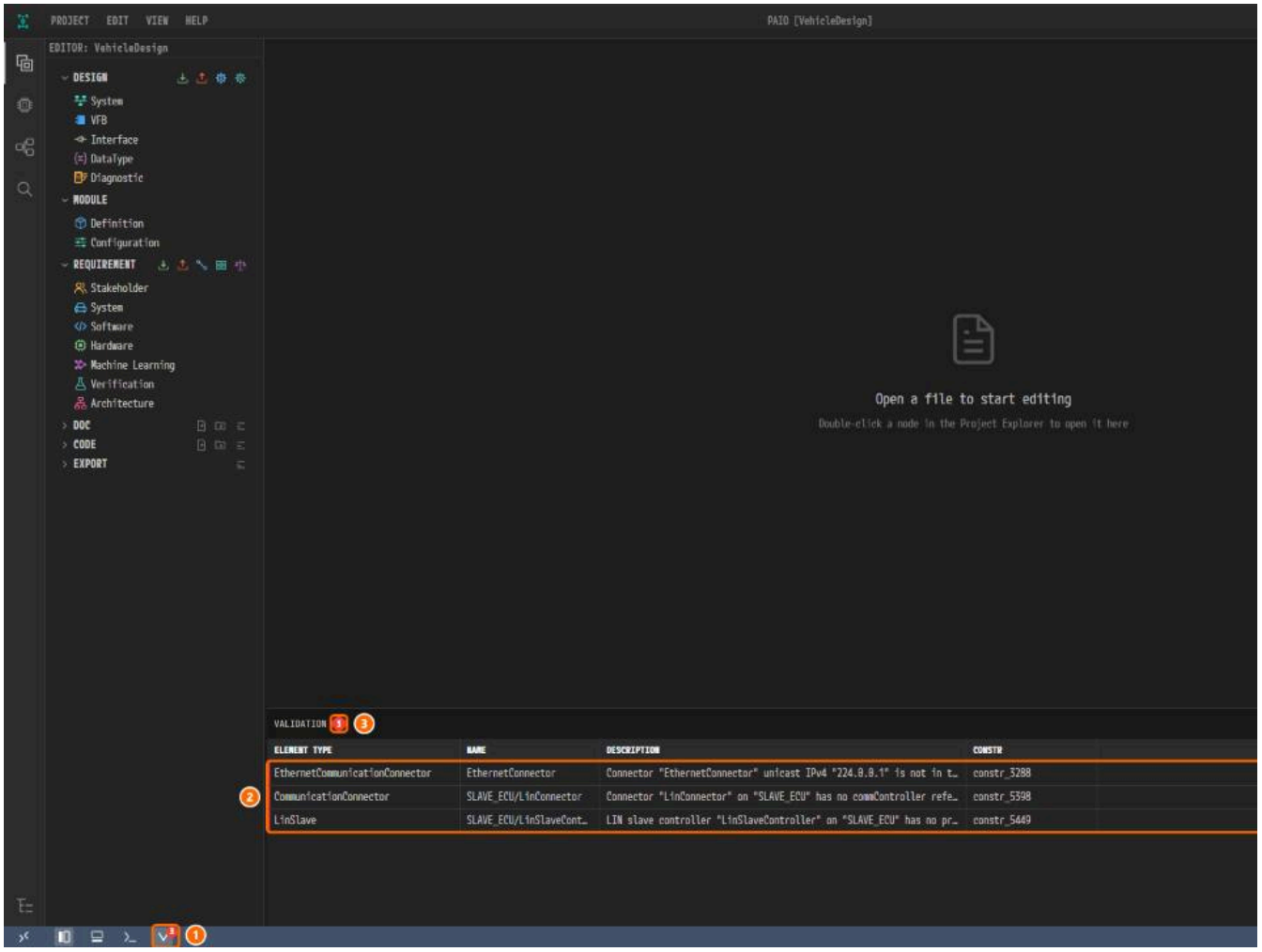


*Figure 6-1. ① The **V** icon (badge `3`), ② the three error rows, ③ the panel badge `3`.*

2. If the text is cut off, drag the right edge of the **NAME** and **DESCRIPTION** headers to widen them (optional).

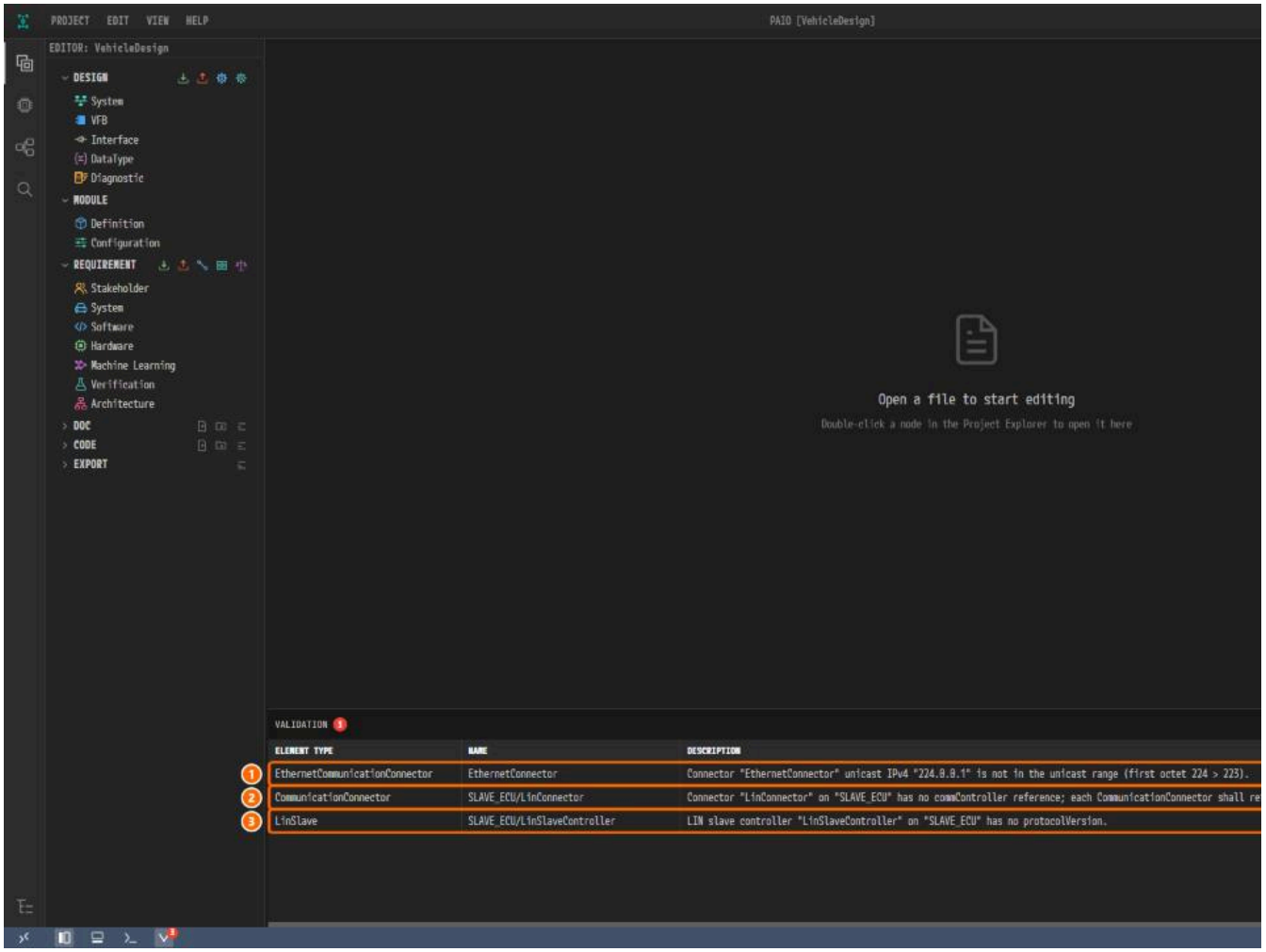


Figure 6-2. ① The `constr_3288` row, ② the `constr_5398` row, ③ the `constr_5449` row.

#	ELEMENT TYPE	NAME	DESCRIPTION
1	`EthernetCommunicationConnector`	`EthernetConnector`	`Connector "EthernetConnector" unicast IPv4 "224.0.0.1" is not in the unicast`
2	`CommunicationConnector`	`SLAVE_ECU/LinConnector`	`Connector "LinConnector" on "SLAVE_ECU" has no commController reference; each reference exactly one communication controller.`
3	`LinSlave`	`SLAVE_ECU/LinSlaveController`	`LIN slave controller "LinSlaveController" on "SLAVE_ECU" has no protocolVersi`

A single click on a row moves you to the element, or close to it. The panel updates as soon as you fix a value, even before saving.

Fixing the Unicast Address of the HPC Connector

1. Click the `constr_3288` row. The tab `VehicleDesign:System` opens with the connector of `ZONE` selected, not the one of `HPC`: both are named `Ethe` connector.

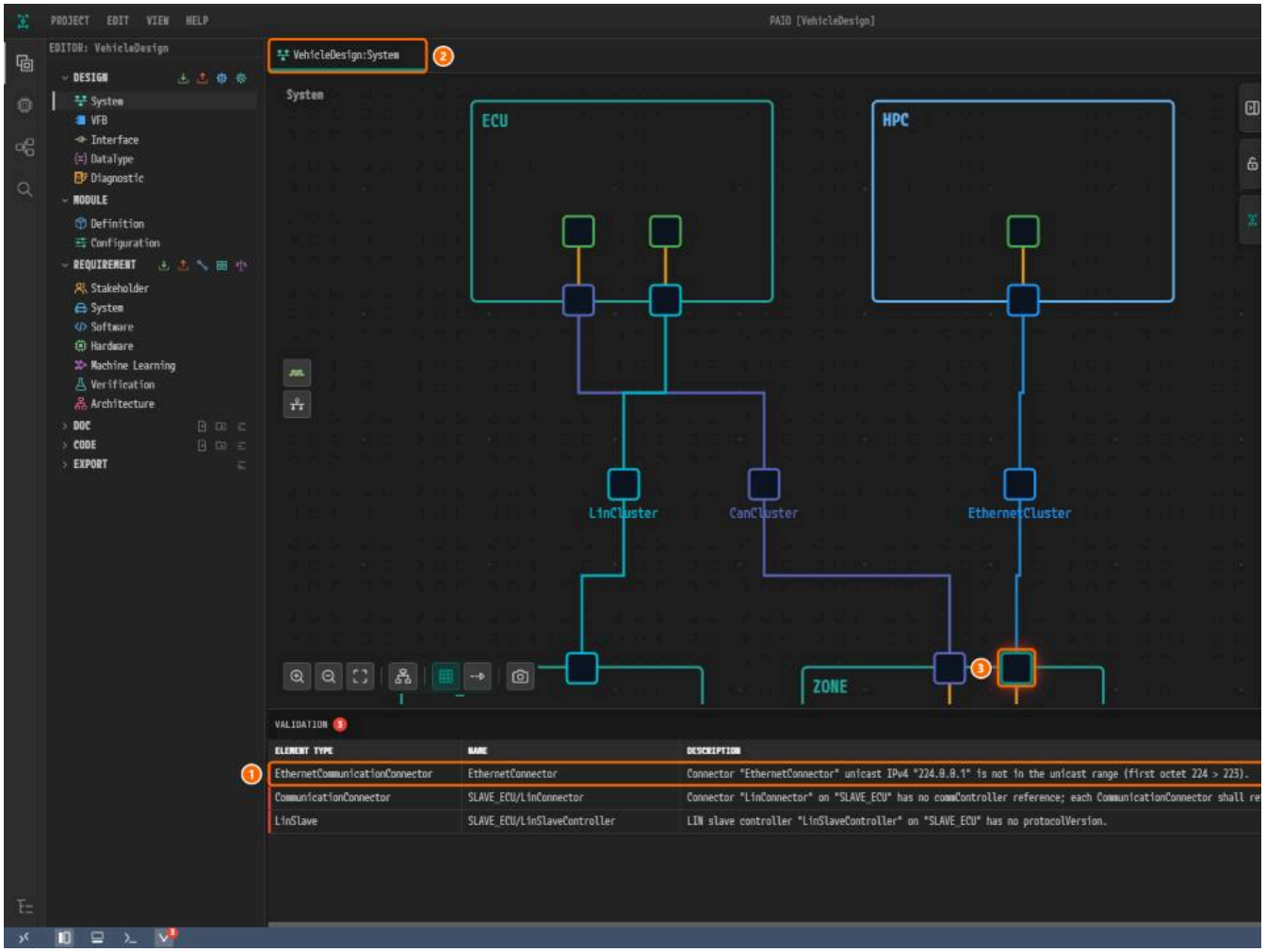


*Figure 6-3. ① The `constr_3288` row, ② the tab `VehicleDesign:System`, ③ the connector of `ZONE`, ④ **UNICAST IPV4 ADDRESS** (`192.168.0.19`).*

2. Click **Fit to View**, then click the connector on the bottom edge of `HPC`. Its **UNICAST IPV4 ADDRESS** is `224.0.0.1`.

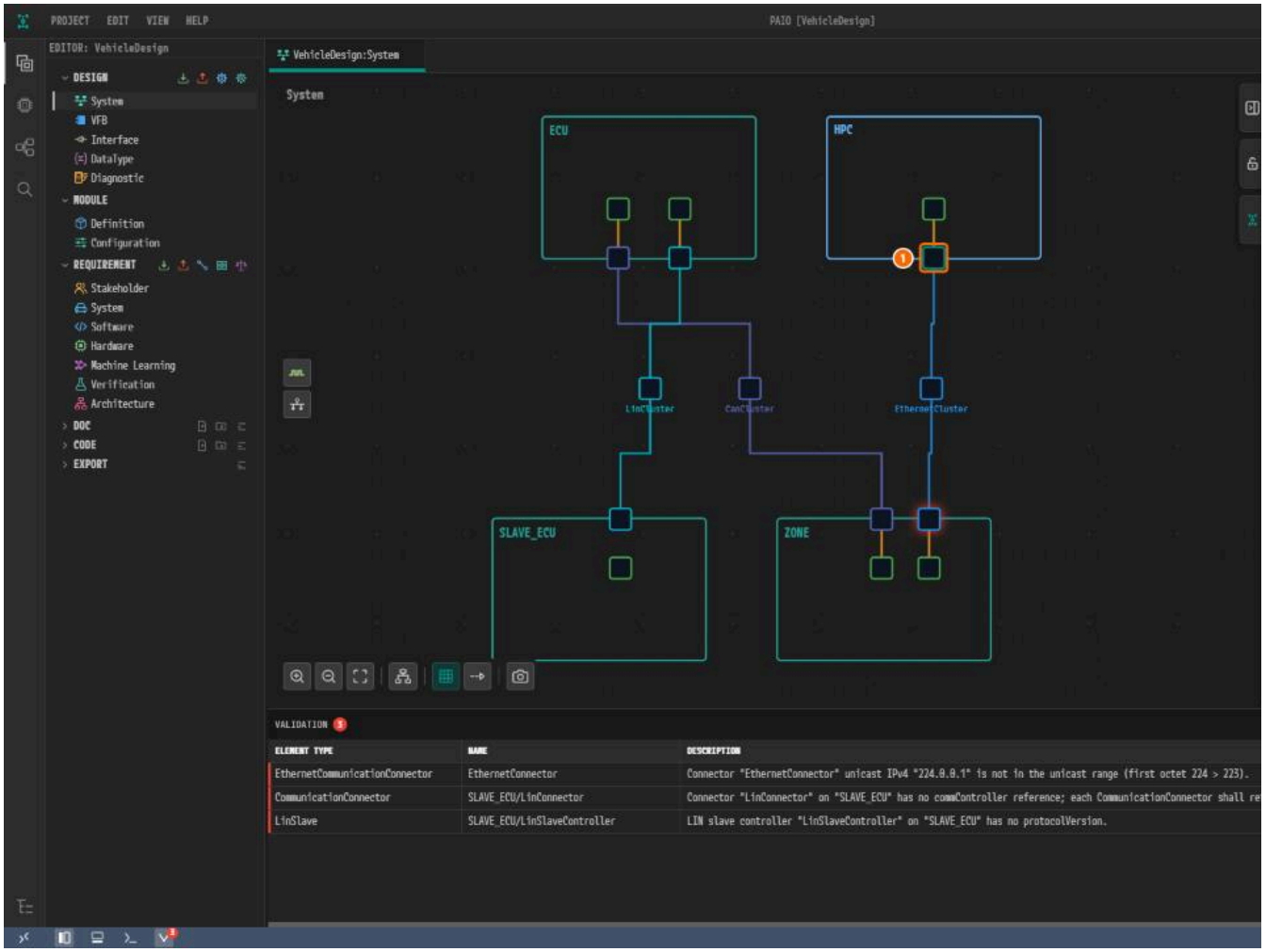


*Figure 6-4. ① The connector of `HPC`, ② **UNICAST IPV4 ADDRESS**, ③ **MULTICAST IPV4 ADDRESS**.*

3. Replace the value of **UNICAST IPV4 ADDRESS** with `192.168.0.11` and press `Tab`. The panel drops to two rows.

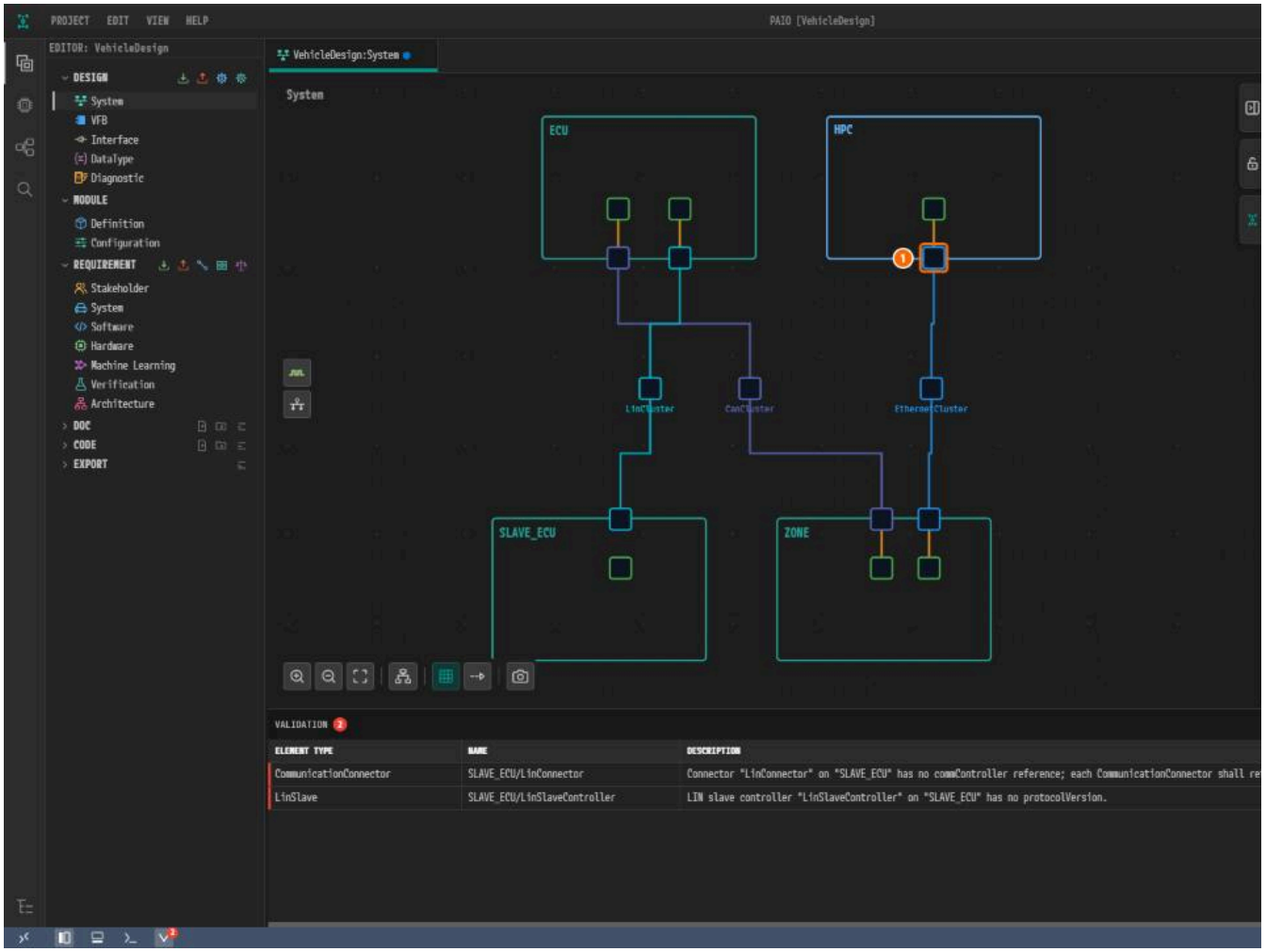


*Figure 6-5. ① The connector of `HPC`, ② **UNICAST IPV4 ADDRESS** = `192.168.0.11`.*

Assigning a Controller to the SLAVE_ECU Connector

1. Click the `constr_5398` row. It selects `LinConnector` of `SLAVE_ECU` directly. **CONTROLLER** shows `Select controller...` in red.

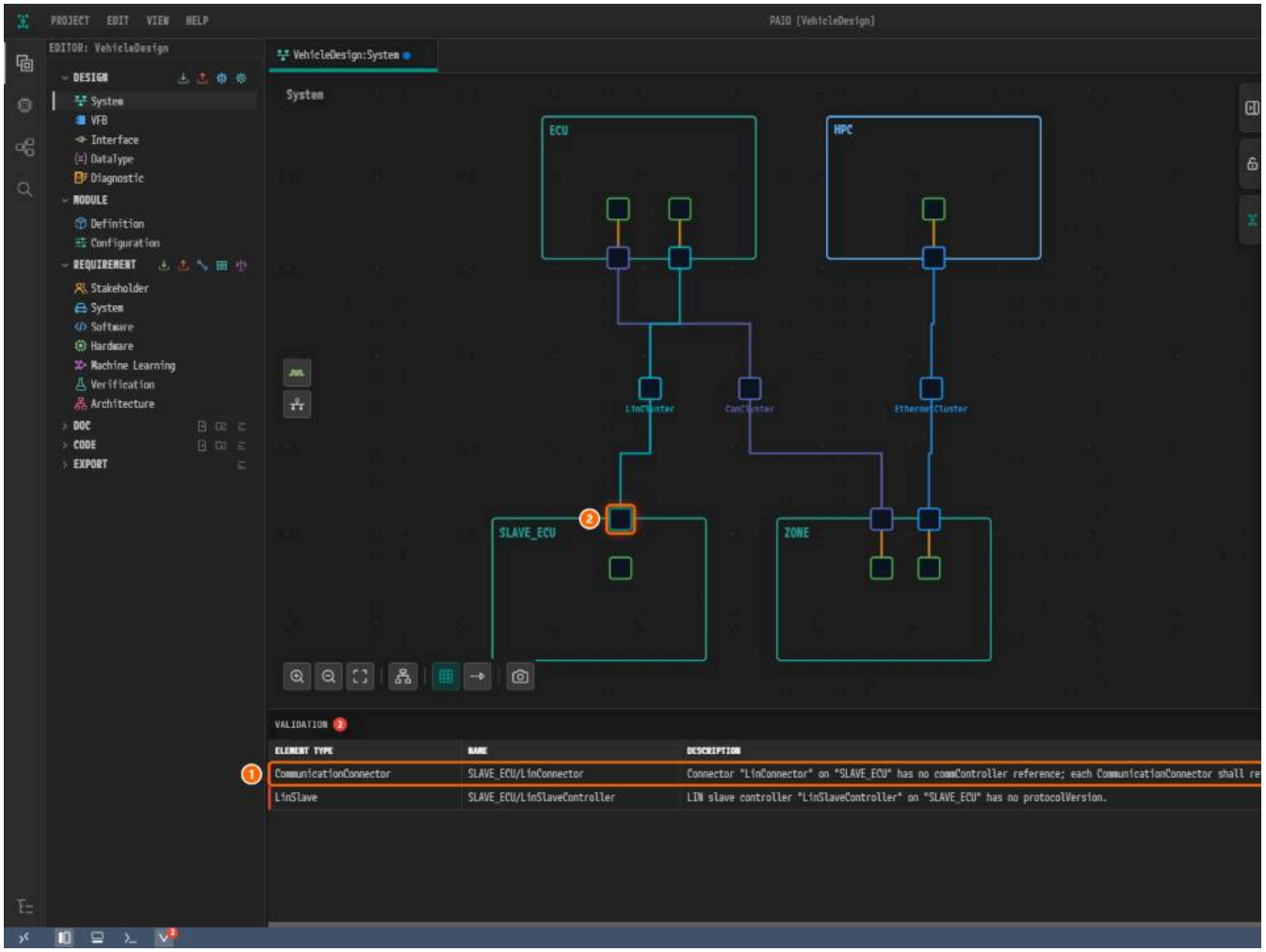


*Figure 6-6. ① The `constr_5398` row, ② the connector of `SLAVE_ECU`, ③ **CONTROLLER**.*

2. Click the **CONTROLLER** field.

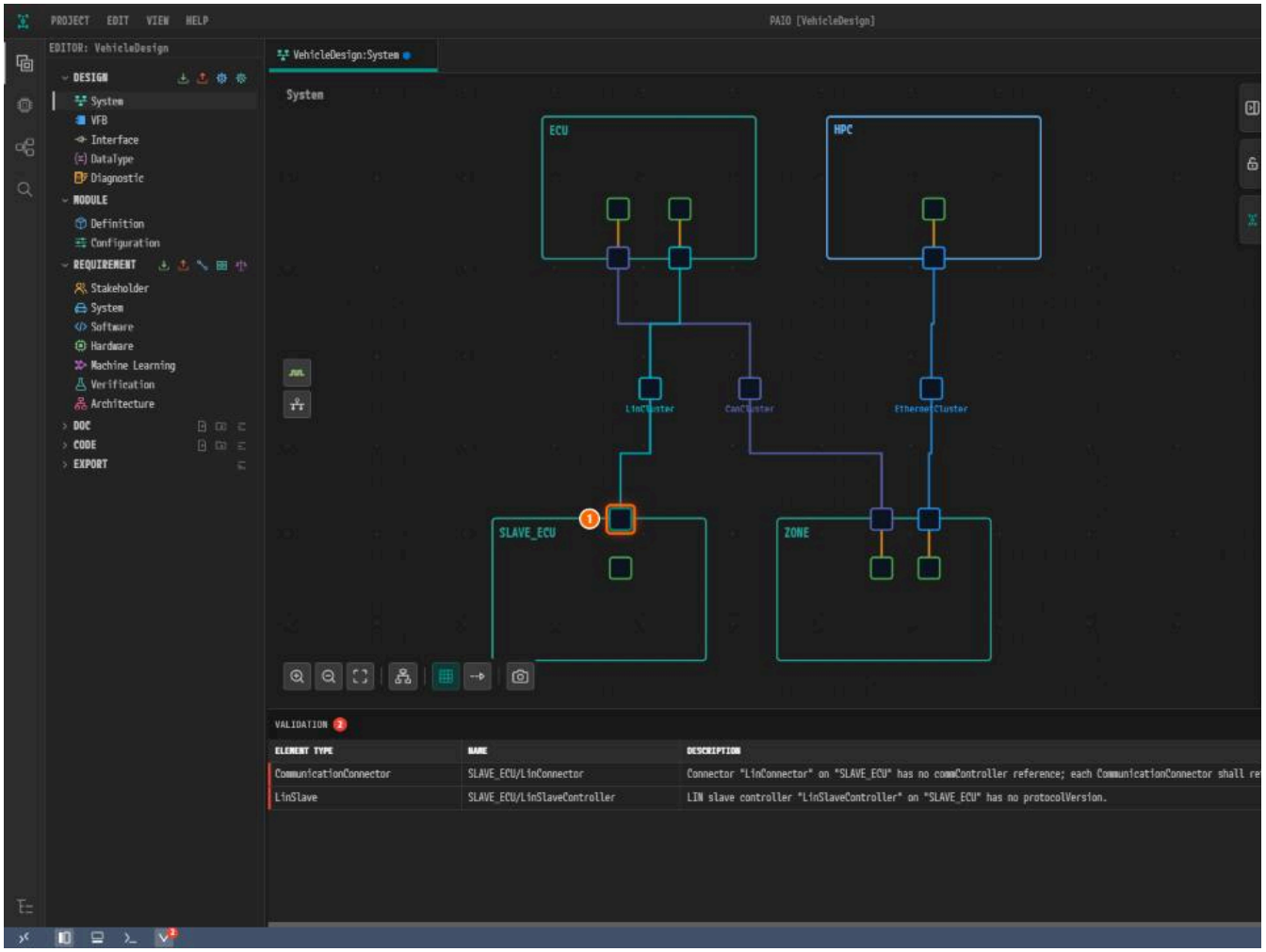


*Figure 6-7. ① The connector, ② **CONTROLLER**, ③ `LinSlaveController`.*

3. Click `LinSlaveController`. The panel drops to one row.

Choosing the Protocol Version of the LIN Slave Controller

1. Click the `constr_5449` row. It selects the node `SLAVE_ECU` (panel **ECU**), not the controller.

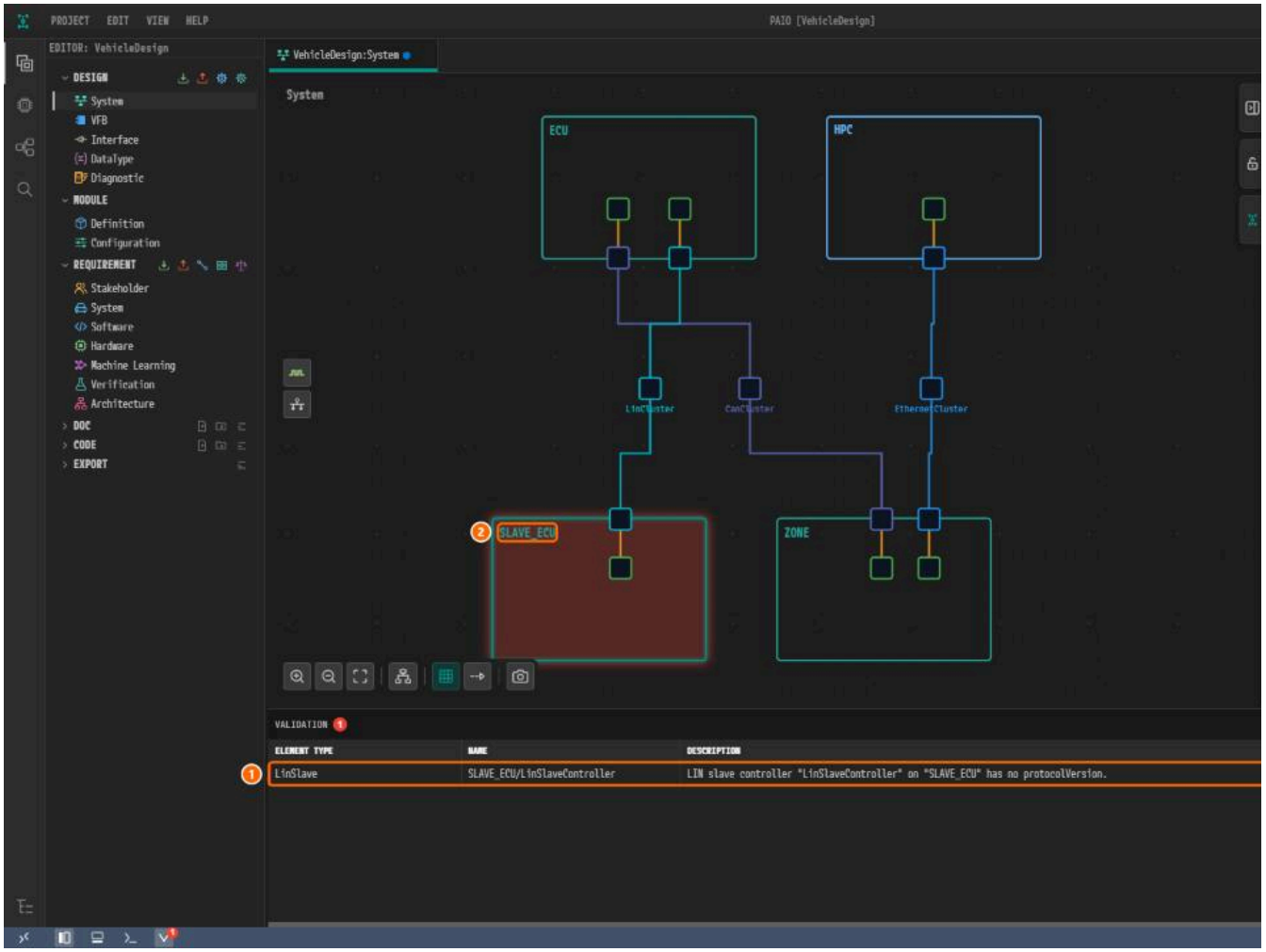


*Figure 6-8. ① The `constr_5449` row, ② the node `SLAVE_ECU`, ③ the panel title **ECU**.*

2. Click the controller `LinSlaveController` inside the node and click `LIN 2.0` under **PROTOCOL VERSION**. The panel shows `No validation issues.`.

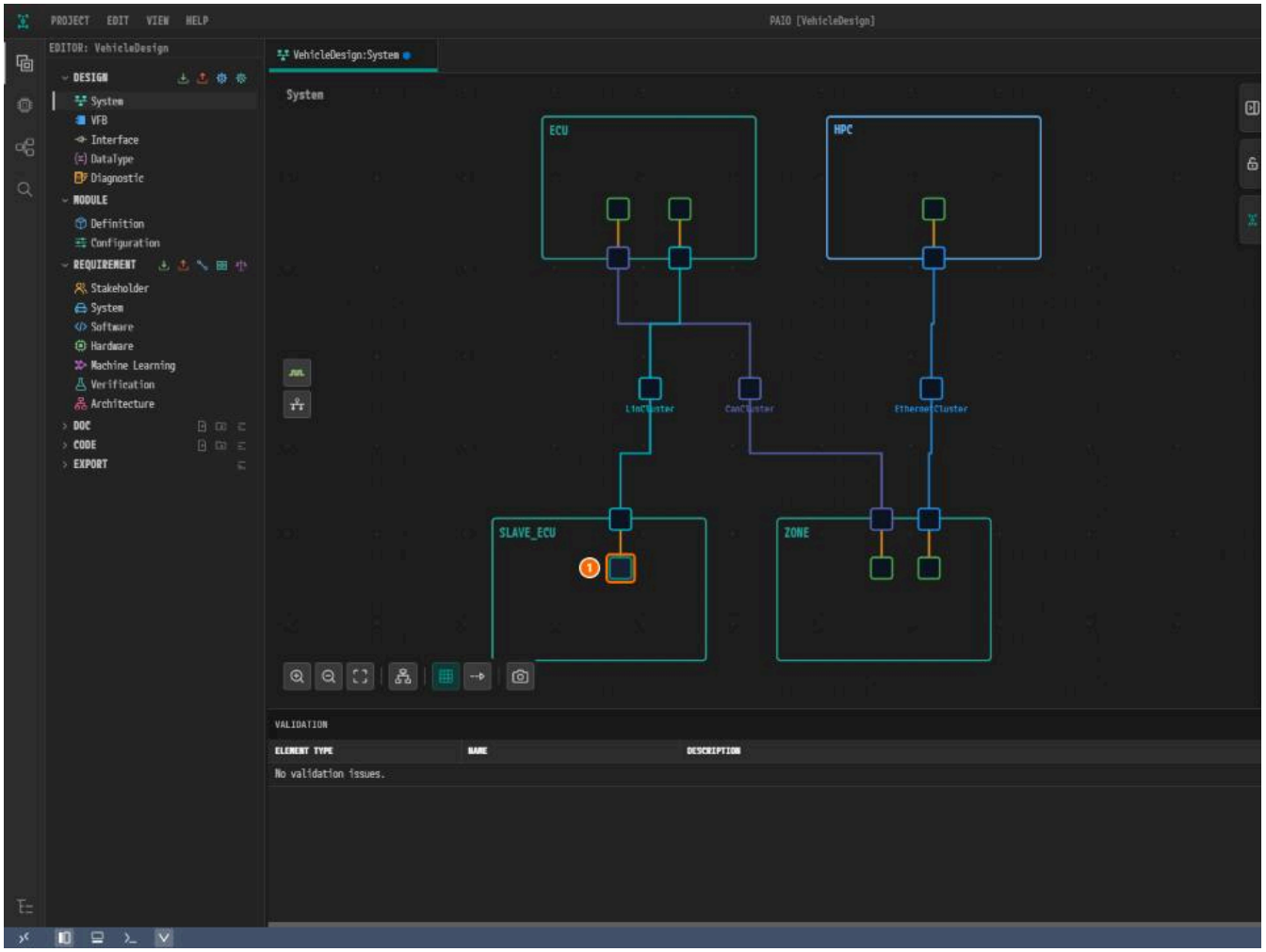


*Figure 6-9. ① The controller, ② **PROTOCOL VERSION** = `LIN 2.0`.*

Saving and Validating Again

1. Click an empty spot of the canvas and press `Ctrl+S`. The panel still shows `No validation issues.`, with no badge.

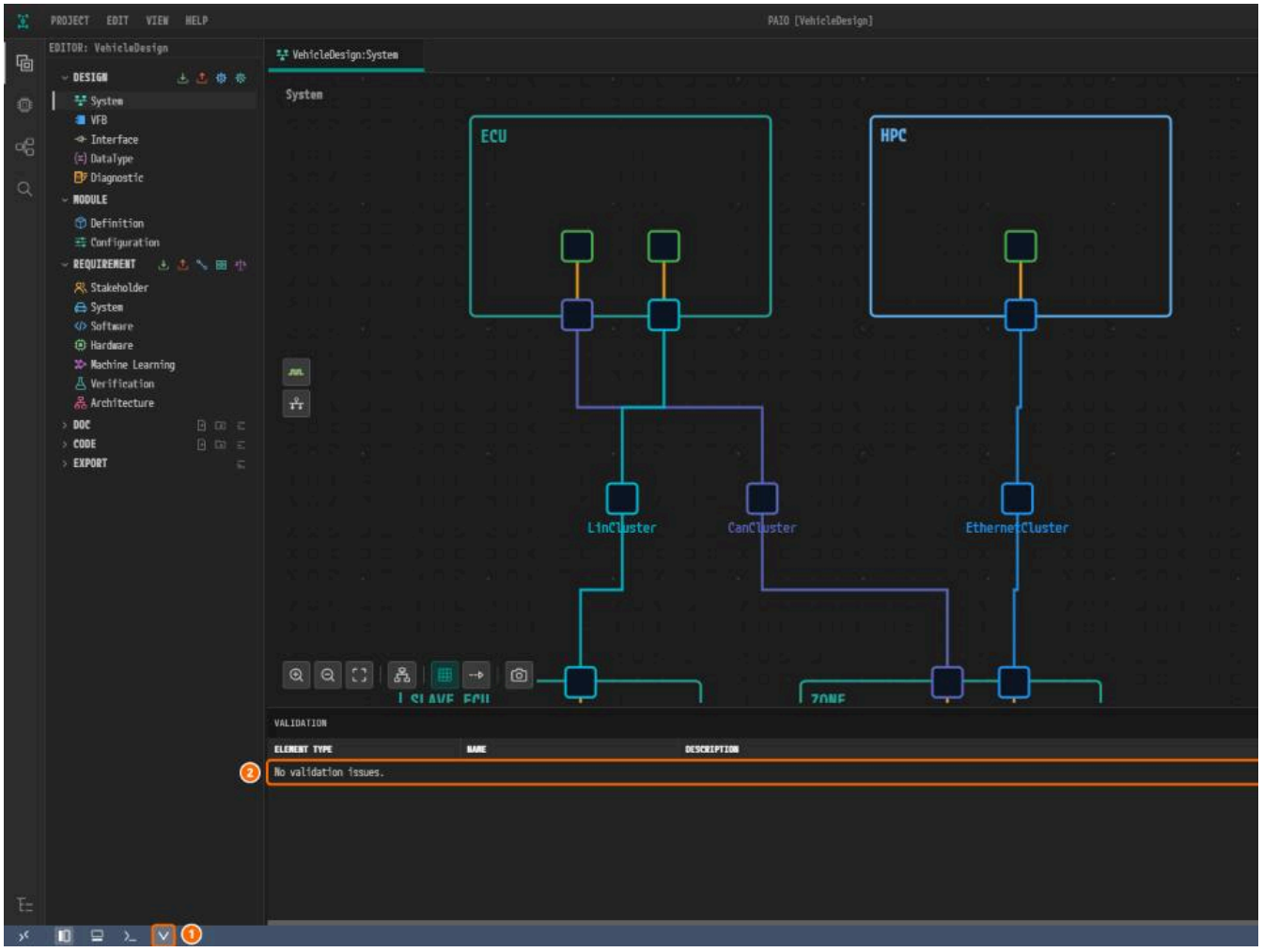


*Figure 6-10. ① The **V** icon without a badge, ② `No validation issues.`. The canvas zoom has returned to the one before **Fit to View**; the design is u*

Why Is It Designed This Way?

- **A unicast address must have a first octet of 223 or less.** `224.0.0.1` is the SOME/IP service discovery multicast address from ETHERNET Configuratio
- **A connector must refer to exactly one controller.** It attaches to the bus through its controller.
- **A LIN controller needs a protocol version.** `LIN 2.0` corresponds to `LIN20` in the reference design.

Checklist

- [ ] The **VALIDATION** panel shows `No validation issues.`, and the **V** icon has no badge.
- [ ] `EthernetConnector` of `HPC` has **UNICAST IPV4 ADDRESS** `192.168.0.11`; the one of `ZONE` still has `192.168.0.19`.
- [ ] `LinConnector` of `SLAVE_ECU` has **CONTROLLER** `LinSlaveController`, and `LinSlaveController` has **PROTOCOL VERSION** `LIN 2.0`.
- [ ] The tab `VehicleDesign:System` carries no modified mark.

Next Step

In ARXML Export you export the topology to ARXML files and compare them with the reference design.

ARXML Export

What You Will Do

You export the topology to four ARXML files in the project's `export` folder. Then you check in the XML editor what your screen values became, and see h `System_Design.arxml`.

Before You Begin

- Start after finishing Validation. The **VALIDATION** panel shows `No validation issues.`; close it with the **V** icon in the footer.
- The `export` folder is empty. Close the tab `VehicleDesign:System` with its **×**, so that no editor is open.

What to find	File	Search string	Find widget	Line
HPC node `HPC`	`VehicleDesign_COM.arxml`	`<MACHINE-DESIGN>`	`1 of 1`	109
The three Ethernet endpoints	`VehicleDesign_COM.arxml`	`<NETWORK-ENDPOINTS>`	`1 of 1`	191
VLAN `VLAN_13`	`VehicleDesign_COM.arxml`	`<VLAN>`	`1 of 1`	237

Exporting the Project

1. In the **EDITOR** view, point at the second icon on the right of the **DESIGN** header. Its tooltip is **Export**.

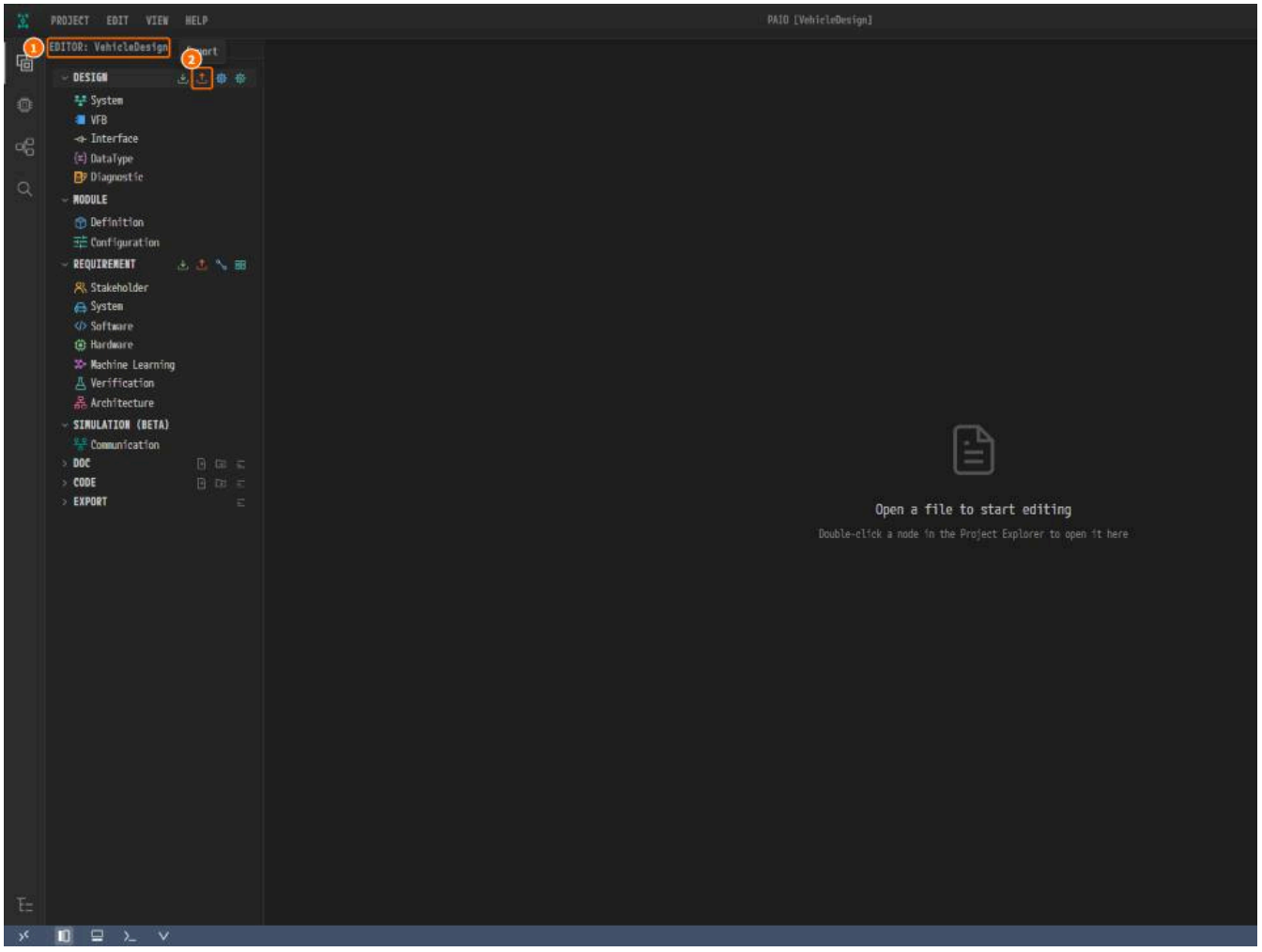


*Figure 7-1. ① The view header `EDITOR: VehicleDesign`, ② the **Export** icon and its tooltip.*

2. Click **Export**. The **Export** dialog opens. Leave **FOLDER** at `export` and **FORMAT** at **ARXML (R25-11)**, and click **Export**. The dialog shows `✓ Complete!` an

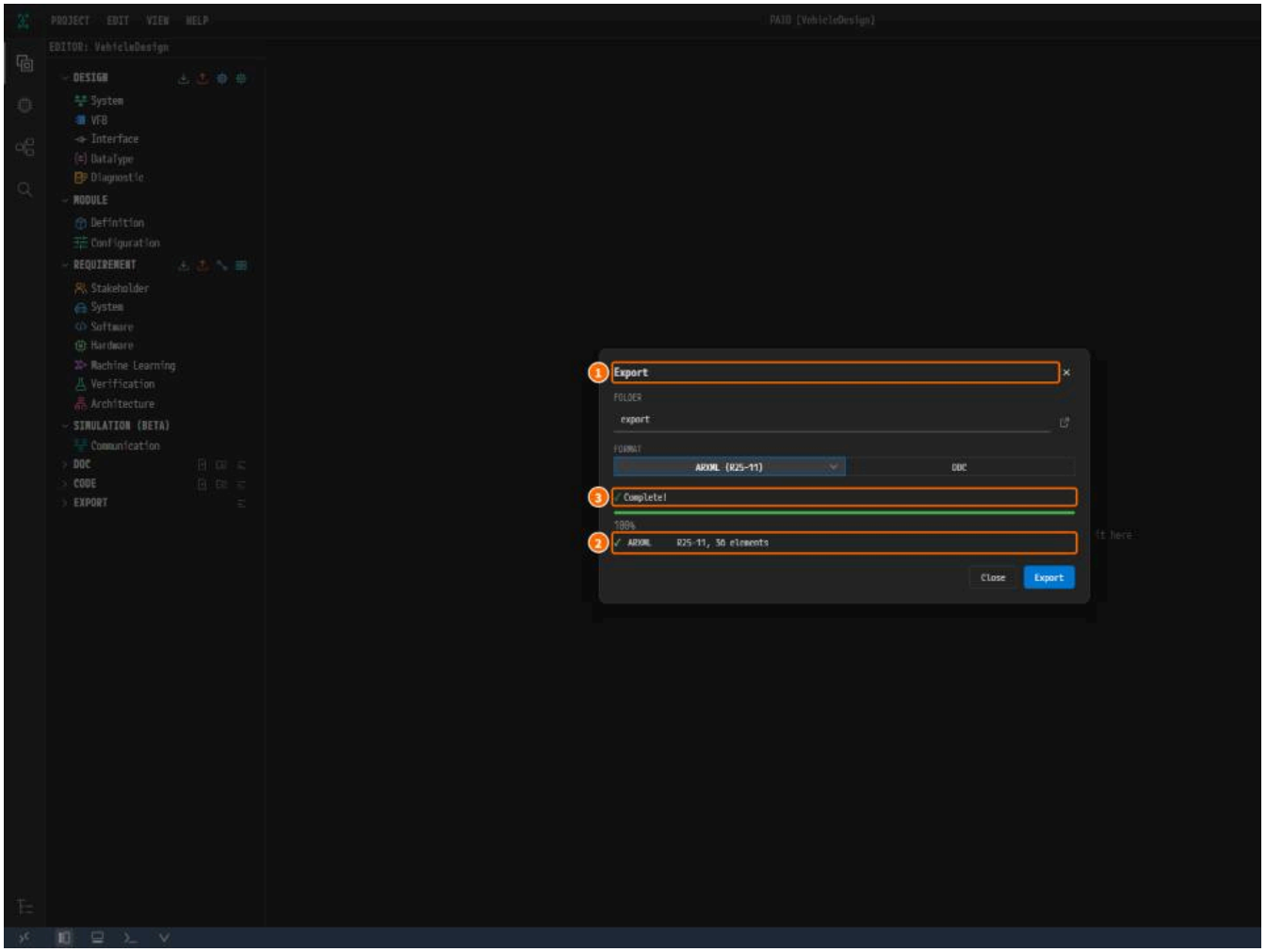


*Figure 7-2. ① **Export**, ② the result line, ③ `✓ Complete!`.*

3. Click **Close**.

> **Note.** The export writes `<project name>_*.arxml` into the folder of **FOLDER** and does not change the files in `design`.

Viewing the Four Files in the export Folder

1. Click the **EXPORT** header to expand it. It lists four files.

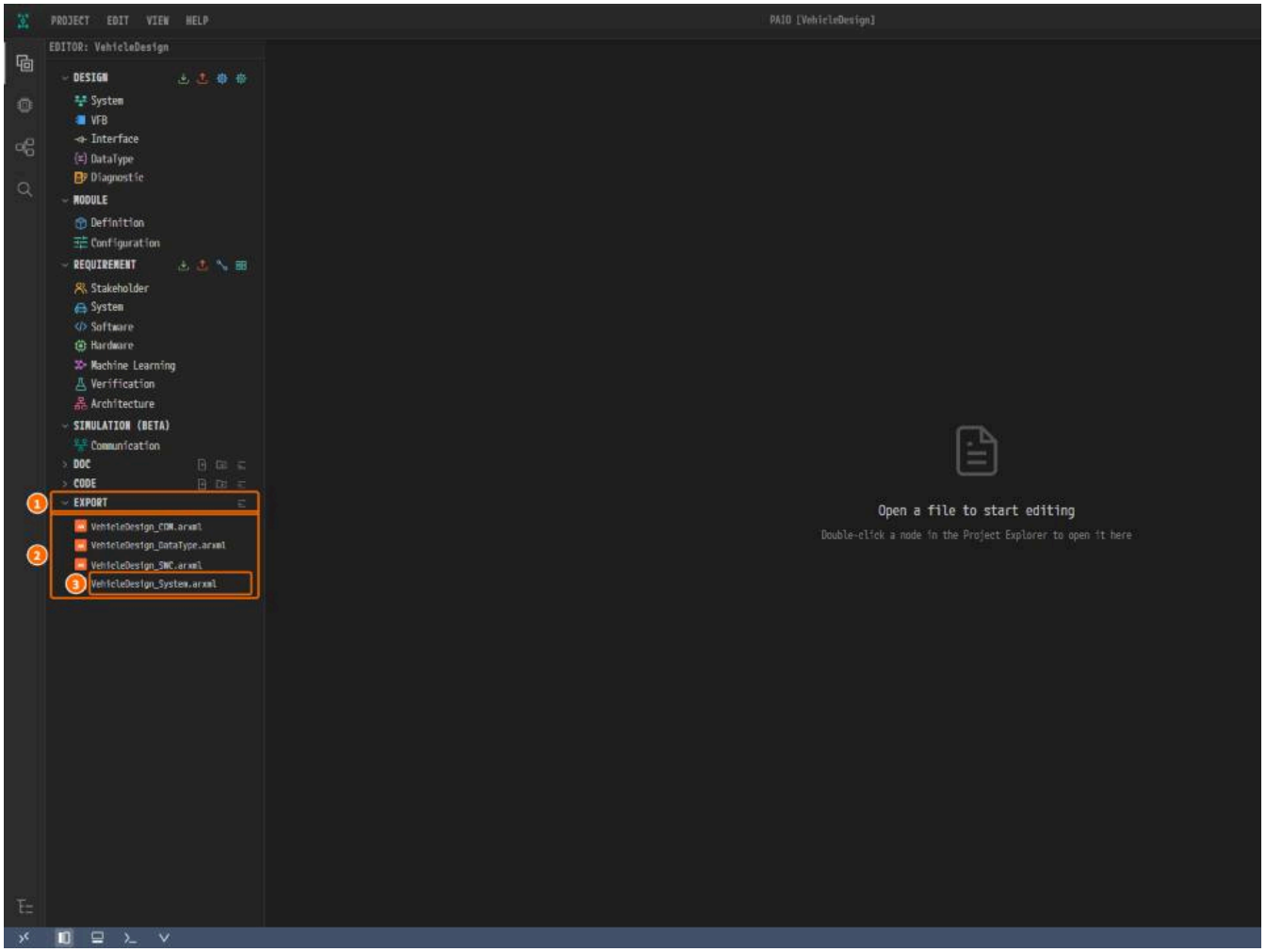


*Figure 7-3. ① **EXPORT**, ② the four files, ③ `VehicleDesign_System.arxml`, which you open next.*

File	Contents
`VehicleDesign_COM.arxml`	The nodes `ECU`, `ZONE`, `SLAVE_ECU`, `HPC` with their controllers and connectors, the three clusters, the endpoints and the VLAN
`VehicleDesign_DataType.arxml`	Basic data types PAIO puts into every export. This design does not use them
`VehicleDesign_SWC.arxml`	One empty package `SwComponentType`
`VehicleDesign_System.arxml`	The system `System`

Viewing the System System

1. In **EXPORT**, double-click `VehicleDesign_System.arxml`. The file opens read-only in the XML editor.
2. Check that the status bar shows `Schema: R25-11` and `✓ No problems`. The editor checks the file against the AUTOSAR schema when it opens it.

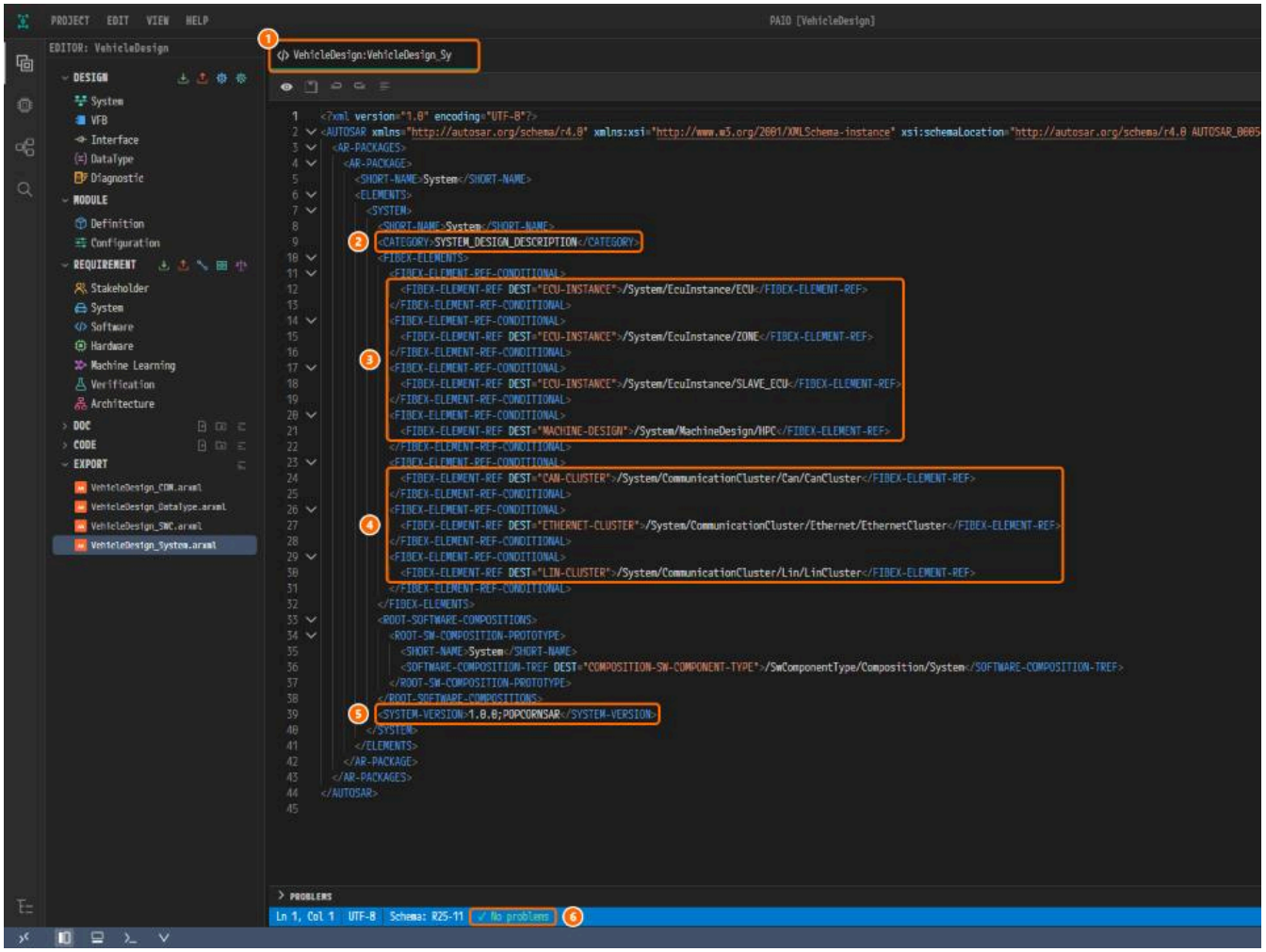


Figure 7-4. Lines 1-45. ① The editor tab, ② line 9 `SYSTEM_DESIGN_DESCRIPTION`, ③ lines 12-21, the four nodes, ④ lines 24-30, the three clusters, ⑤ lin `No problems`.

Line	Content	Where it comes from
9	`CATEGORY` `SYSTEM_DESIGN_DESCRIPTION`	Fixed by the export
12-21	References to `ECU`, `ZONE`, `SLAVE_ECU`, then `HPC`	The nodes on the canvas, ECU nodes in the order you added them (Project and Nodes)
24-30	References to `CanCluster`, `EthernetCluster`, `LinCluster`	CAN Configuration, ETHERNET Configuration, LIN Configuration
33-38	Root composition `System`	Not entered. It points at a composition that is in none of the four files; the schema check does not fo
39	`SYSTEM-VERSION` `1.0.0;POPCORNSAR`	**VERSION** in ECU Properties

Viewing the ECU Node ECU

1. In **EXPORT**, double-click `VehicleDesign_COM.arxml`. The status bar shows `✓ No problems`.
2. Scroll down until line 10 `<ECU-INSTANCE>` is at the top.

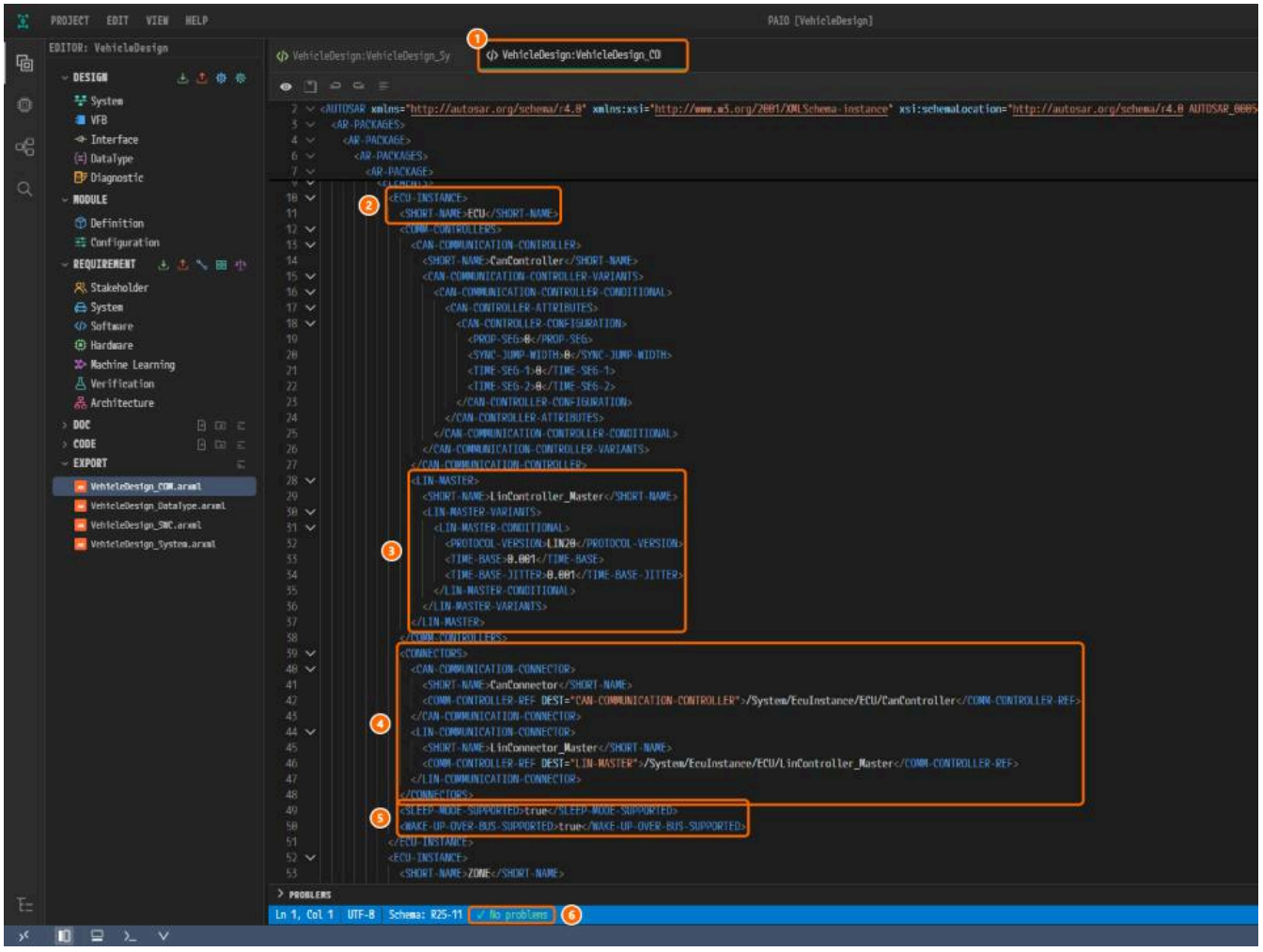


Figure 7-5. Lines 10-51. ① The active tab, ② `ECU-INSTANCE` `ECU`, ③ `LinController_Master`, ④ `CONNECTORS`, ⑤ `SLEEP-MODE-SUPPORTED` and `WAKE-U` `problems`. The top rows of the editor are pinned headers of the enclosing elements.

Line	Content	Where it comes from
13-27	`CanController`, the four bit timing values `0`	CAN Configuration
28-37	`LinController_Master`: `LIN20`, `TIME-BASE` and `TIME-BASE-JITTER` `0.001`	**PROTOCOL VERSION** `LIN 2.0`, `1` `ms`, `1` `ms` in LIN Configuration
40-47	`CanConnector` → `CanController`, `LinConnector_Master` → `LinController_Master`	The connectors' **CONTROLLER**
49-50	`SLEEP-MODE-SUPPORTED`, `WAKE-UP-OVER-BUS-SUPPORTED` `true`	ECU Properties

`ZONE` (lines 52-82) and `SLAVE_ECU` (lines 83-103) have the same shape. The two partitions of `ECU` in the reference design are missing: the System editor

Viewing the HPC Node HPC

1. Click inside the editor and press `Ctrl+F`.
2. Type `<MACHINE-DESIGN>`. The find widget shows `1 of 1` and selects line 109.
3. Scroll down until line 109 is at the top.

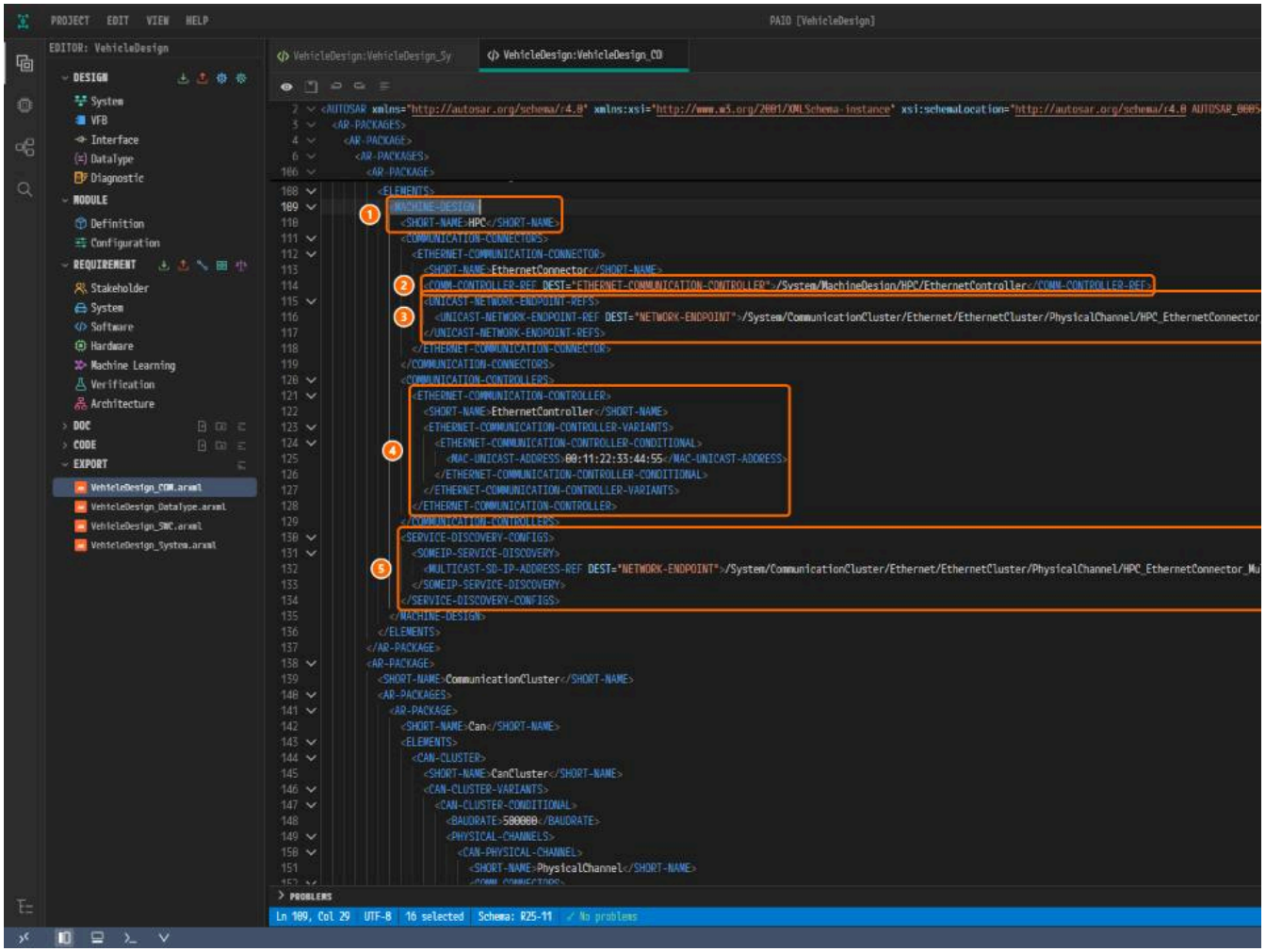


Figure 7-6. ① `MACHINE-DESIGN` `HPC`, ② line 114 `COMM-CONTROLLER-REF`, ③ lines 115-117 `HPC_EthernetConnector_Endpoint`, ④ `EthernetController` `CONFIGS`, ⑥ the find widget.

An HPC node is exported as a `MACHINE-DESIGN`, not an `ECU-INSTANCE`. Line 114 points at `/System/MachineDesign/HPC/EthernetController` and line 116 (`192.168.0.11`): this is where the tutorial corrects the reference design (see "Where the HPC Connector Is Corrected" below). The export creates `SERVI` the connector has a **MULTICAST IPV4 ADDRESS**.

Viewing the Ethernet Endpoints

1. Replace the search string with `<NETWORK-ENDPOINTS>`. The find widget shows `1 of 1` and selects line 191.
2. Scroll down until line 191 is at the top.

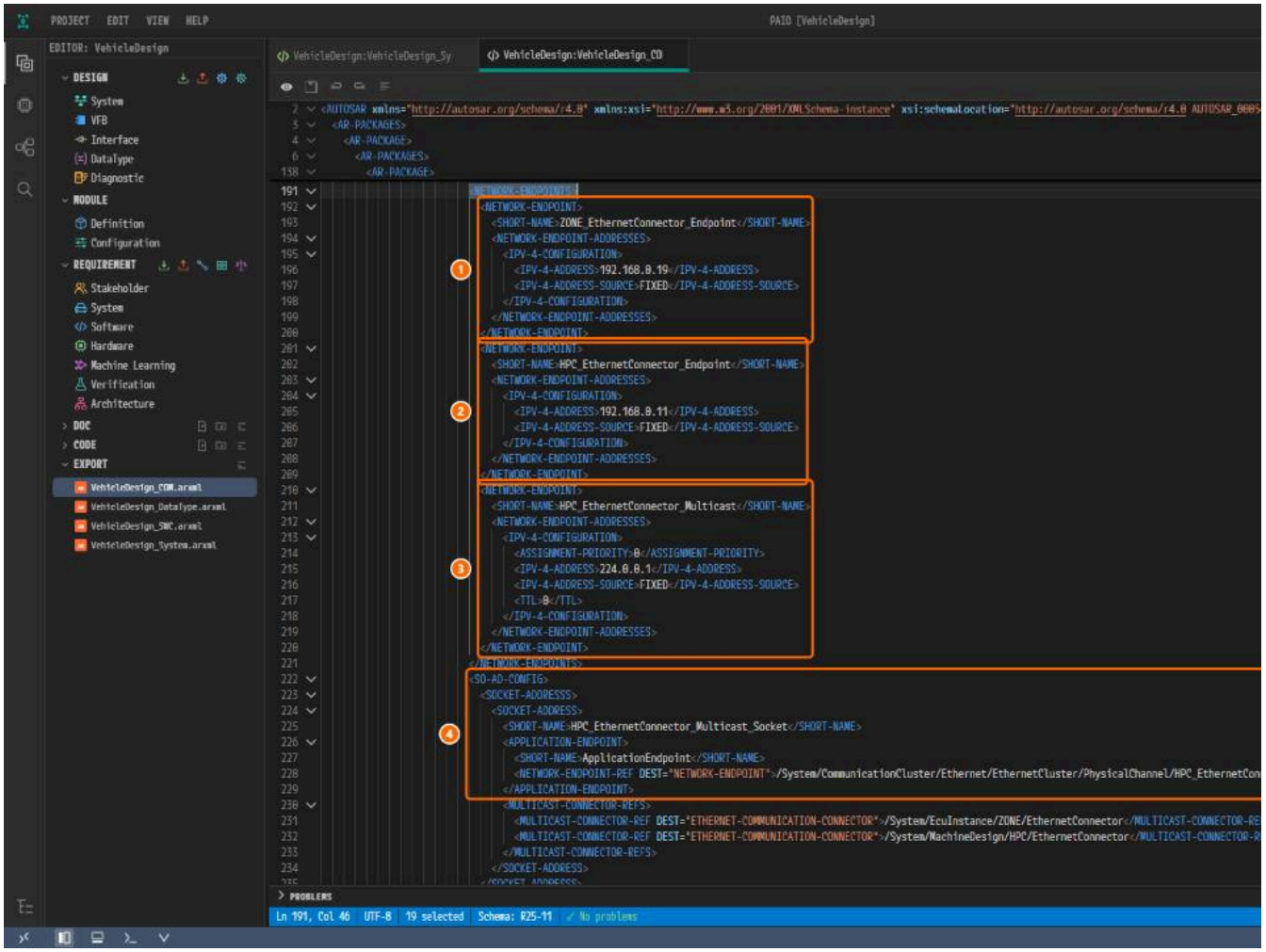


Figure 7-7. ① `ZONE_EthernetConnector_Endpoint` (`192.168.0.19`), ② `HPC_EthernetConnector_Endpoint` (`192.168.0.11`), ③ `HPC_EthernetConne` `SO-AD-CONFIG`.

All three endpoints are `FIXED`. You do not enter their names: the export names them `<node>_<connector>_Endpoint` and `<node>_<connector>_Multic` and the SD socket for the multicast address.

Viewing the VLAN and the LIN Cluster

1. Replace the search string with `<VLAN>`. The find widget shows `1 of 1` and selects line 237.

2. Scroll down until line 237 is at the top.

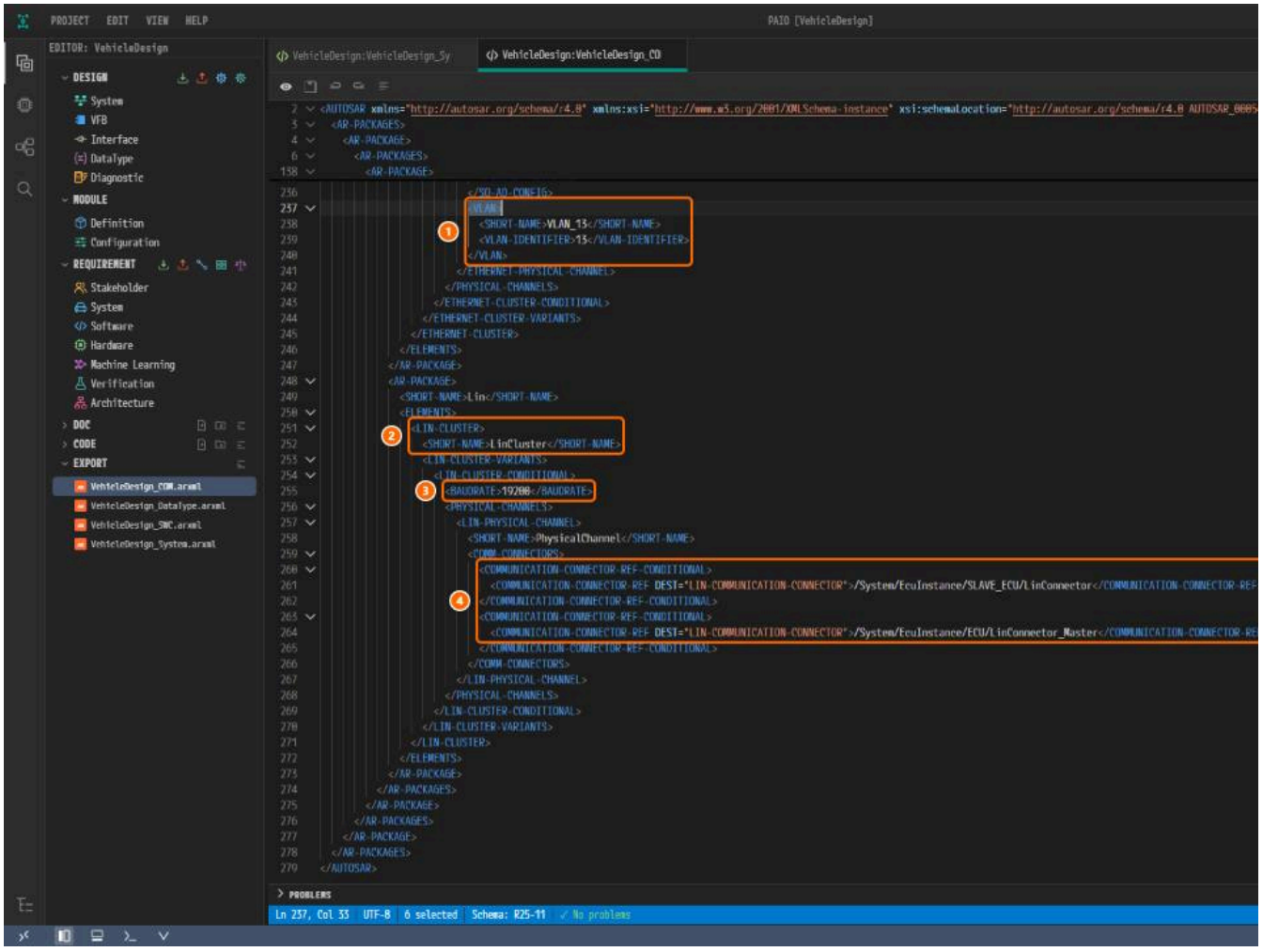


Figure 7-8. ① `VLAN_13`, `VLAN-IDENTIFIER` `13`, ② `LIN-CLUSTER` `LinCluster`, ③ `BAUDRATE` `19200`, ④ the references to `SLAVE_ECU/LinConnector` ar

3. Press `Escape` to close the find widget.

The export names the VLAN `VLAN_<ID>`. `BAUDRATE` `19200` is a value PAIO saves into a new LIN cluster, and the LIN channel has no `BUS-IDLE-TIMEOUT-PER` `CanCluster` (lines 144-170) holds `BAUDRATE` `500000` and the `BUS-OFF-RECOVERY` values of CAN Configuration.

Where the HPC Connector Is Corrected

Item	Reference design	Your export
`COMM-CONTROLLER-REF`	None	`/System/MachineDesign/HPC/EthernetController` (line 114)
Unicast endpoint	`ZONE_Endpoint` = `192.168.0.19`, the address of `ZONE`	`HPC_EthernetConnector_Endpoint` = `192.168.0.11` (line 116)

- In the reference design the connector has no controller, and both Ethernet connectors would share `192.168.0.19` while `HPC_Endpoint` stays unus Configuration.
- Validation does not report this defect: `constr_5398` checks only connectors of ECU nodes.

How Your Export Differs from the Reference Design

The reference design is one file with 29 named elements; your export is four files with 79. 27 elements have a counterpart. Only the two partitions are mi

Difference	
Nodes moved to `/System/EcuInstance` (`HPC` to `/System/MachineDesign`), clusters to `/System/CommunicationCluster/Can`, `/Ethernet`, `/Lin`	Namir
`CATEGORY` `SYSTEM_DESCRIPTION` → `SYSTEM_DESIGN_DESCRIPTION`; references in another order	Namir
`ZONE_Endpoint`, `HPC_Endpoint`, `SOMEIP_SD_Multicast`, `Vlan_13` → `ZONE_EthernetConnector_Endpoint`, `HPC_EthernetConnector_Endpoint`, `HPC_EthernetConnector_Multicast`, `VLAN_13`	Namir
MAC address `11:22:EE:44:55:66` → `11:22:ee:44:55:66`; `CanCommController` has no empty `CAN-COMMUNICATION-CONTROLLER-CONDITIONAL`	Namir
`ECU_Partition0`, `ECU_Partition1` missing	No fie
`CATEGORY` `WIRED` missing (Ethernet channel, controllers, connectors); `PROTOCOL-NAME` `IEEE 802.3` missing; `WAKE-UP-BY-CONTROLLER-SUPPORTED` `false` of `HPC` missing	No fie
`BUS-IDLE-TIMEOUT-PERIOD` `0.01` of the LIN channel missing	**IDLE T**
`MAC-LAYER-TYPE` `X-MII` missing	Tutori schen
`WAKE-UP-BY-CONTROLLER-SUPPORTED` `true` of `LinController_Master` missing; `PROTOCOL-NAME` `CAN-FD` missing; `CREATE-ECU-WAKEUP-SOURCE` `false` of the `HPC` connector missing	Tutori not ex
`BUS-OFF-RECOVERY` added to `CanCluster`	Tutori
`HPC` connector: controller reference added, unicast endpoint changed	Defec
`BAUDRATE` `500000` and `19200`, `SERVICE-DISCOVERY-CONFIGS`, SD socket (2), `ASSIGNMENT-PRIORITY` / `TTL` `0`, 49 basic data types, root composition `System`	Gener
No `UUID` attributes; different file layout and element order	Not d

Your export has no NM or TP elements; neither does the reference design.

Why Is It Designed This Way?

- **The XML editor checks the schema on open.** `✓ No problems` confirms the structure, but not that reference targets exist.
- **Most differences are locations and names** set by the export rules.
- **The `HPC` connector does not follow the reference design**, because the reference design is wrong there.

Checklist

- [ ] **EXPORT** lists `VehicleDesign_COM.arxml`, `VehicleDesign_DataType.arxml`, `VehicleDesign_SWC.arxml` and `VehicleDesign_System.arxml`.
- [ ] Both open files show `Schema: R25-11` and `✓ No problems`.
- [ ] Line 39 of `VehicleDesign_System.arxml` is `SYSTEM-VERSION` `1.0.0;POPCORNSAR`.
- [ ] In `VehicleDesign_COM.arxml`, line 114 points at `/System/MachineDesign/HPC/EthernetController` and line 116 at `HPC_EthernetConnector_Endpoi`
- [ ] The three endpoints are `192.168.0.19`, `192.168.0.11` and `224.0.0.1`, and `VLAN_13` has `VLAN-IDENTIFIER` `13`.
- [ ] Neither tab carries the modified mark `•`.

Next Step

The main part of the tutorial ends here. Continue with the Appendix.

Appendix

What You Will Do

This appendix is reference material, not a chapter to follow. Use it to review your design or to find which value belongs to which chapter.

Before You Begin

- The values here are the same as in chapters 1 to 7. If one looks different, follow the chapter.

Checklist of the Finished Design

Nodes and System

- [ ] The System editor holds `ECU`, `ZONE`, `SLAVE_ECU` (panel title **ECU**) and `HPC` (panel title **HPC**).
- [ ] The three ECU nodes have **SLEEP MODE SUPPORTED** and **WAKE-UP OVER BUS** on. `HPC` has no **POWER MANAGEMENT** section.
- [ ] **VERSION** in the SYSTEM panel is `1.0.0;POPCORNSAR`, and **ECU MAPPING** lists the four nodes.

CAN Bus

- [ ] `ECU` has `CanController` and `CanConnector`; `ZONE` has `CanCommController` and `CanConnector`.
- [ ] **Connected Connectors (2)** of `CanCluster` lists `ZONE/CanConnector`, `ECU/CanConnector`.
- [ ] **BUS-OFF RECOVERY** of `CanCluster` is `100` `ms`, `200` `ms`, `2`, `100` `ms`.

Ethernet Bus

- [ ] `ZONE`: `EthernetCommController` (`11:22:ee:44:55:66`), `EthernetConnector` (`192.168.0.19`, **FIXED**).
- [ ] `HPC`: `EthernetController` (`00:11:22:33:44:55`), `EthernetConnector` (`192.168.0.11`, **FIXED**, multicast `224.0.0.1`).
- [ ] **MAC LAYER TYPE** is off on both controllers.
- [ ] `EthernetCluster` has **VLAN ID** `13` and lists `ZONE/EthernetConnector`, `HPC/EthernetConnector`.

LIN Bus

- [ ] `ECU`: `LinController_Master` (`LIN 2.0`, `1` `ms`, `1` `ms`), `LinConnector_Master`.
- [ ] `SLAVE_ECU`: `LinSlaveController` (`LIN 2.0`), `LinConnector`.
- [ ] `LinCluster` has an empty **IDLE TIMEOUT** and lists `SLAVE_ECU/LinConnector`, `ECU/LinConnector_Master`.

Validation and Export

- [ ] The **VALIDATION** panel shows `No validation issues.`.
- [ ] The `export` folder has the four ARXML files, and the XML editor shows `✓ No problems` for them.
- [ ] No tab carries a modified mark.

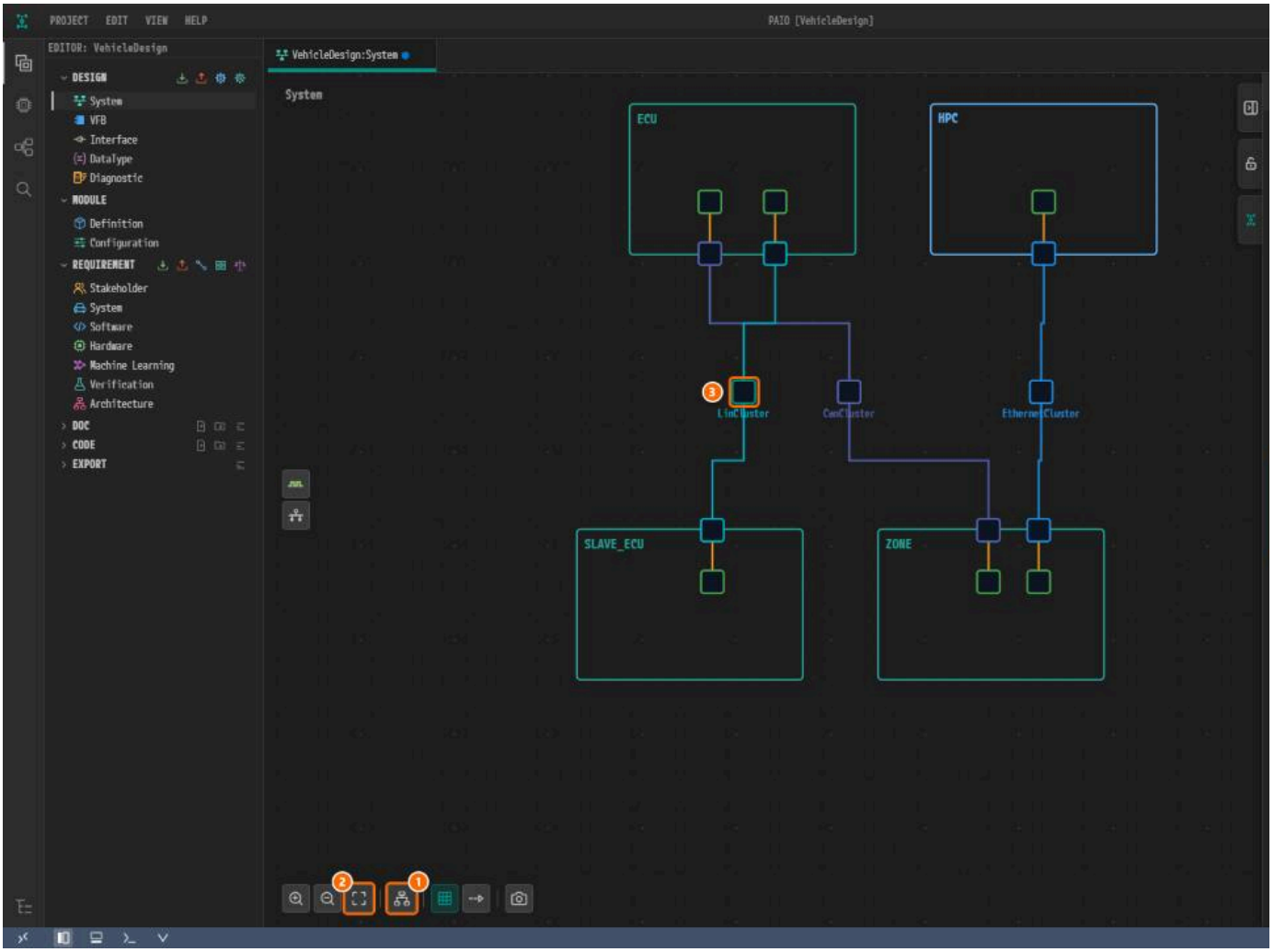


*Figure 8-1. The finished design (Figure 5-14 of LIN Configuration). ① **Auto Layout**, ② **Fit to View**, ③ `LinCluster`, ④ **CHANNEL NAME**, the empty **IDLE TIMEOU** are not design values.*

All Value Tables

Project and System (Chapters 1 and 2)

Item	Value	Chapter
Project name	`VehicleDesign`	1
System name	`System` (default of a new project, unchanged)	1
SYSTEM panel **VERSION**	`1.0.0;POPCORNSAR` (default `1.0.0`)	2
SYSTEM panel **ECU MAPPING**	`ECU`, `ZONE`, `SLAVE_ECU`, `HPC` (all `No composition mapped`, only checked)	2

Nodes and Power Management (Chapters 1 and 2)

Order added	Right-click position	Menu item	Default name	NAME	Panel title	SLEEP MODE SUPPORTED / WAKE-UP OVER BUS
1	Top left of the canvas	**Add ECU**	`ECU_0`	`ECU`	**ECU**	On / on
2	Top right of the canvas	**Add ECU**	`ECU_0`	`ZONE`	**ECU**	On / on
3	Bottom left of the canvas	**Add ECU**	`ECU_0`	`SLAVE_ECU`	**ECU**	On / on
4	Bottom right of the canvas	**Add HPC**	`HPC_0`	`HPC`	**HPC**	No section

Keep the order `ECU` → `ZONE` → `SLAVE_ECU` → `HPC`. A node appears where you right-click, and **Auto Layout** is used from CAN Configuration onward.

Partition of the reference design	`EXEC-IN-USER-MODE`	This tutorial
`ECU_Partition0` (`ECU`)	`true`	Not created. The System editor has no input field for it
`ECU_Partition1` (`ECU`)	`false`	Not created. The System editor has no input field for it

Controllers (Chapters 3 to 5)

Node	Menu	Panel title	Default name	NAME	Oth
`ECU`	**Add Controller > CAN**	**CAN CONTROLLER**	`ECU_Can_Controller_0`	`CanController`	**CAN** on, **SYNC JUMP WIDTH**, **PROPAGAT SEGMENT 2** all `0`
`ZONE`	**Add Controller > CAN**	**CAN CONTROLLER**	`ZONE_Can_Controller_0`	`CanCommController`	**CAN** on, the four bit timing fields left
`ZONE`	**Add Controller > Ethernet**	**ETHERNET CONTROLLER**	`ZONE_Ethernet_Controller_0`	`EthernetCommController`	**MAC ADDRESS** `11:22:EE:44:55:66` e **MAC LAYER TYPE** not clicked
`HPC`	**Add Controller > Ethernet**	**ETHERNET CONTROLLER**	`HPC_Ethernet_Controller_0`	`EthernetController`	**MAC ADDRESS** `00:11:22:33:44:55`, **M**
`ECU`	**Add Controller > LIN Master**	**LIN MASTER**	`ECU_LinMaster_Controller_0`	`LinController_Master`	**PROTOCOL VERSION** `LIN 2.0`, **TIME BA UP BY CONTROLLER** not clicked
`SLAVE_ECU`	**Add Controller > LIN Slave**	**LIN SLAVE**	`SLAVE_ECU_LinSlave_Controller_0`	`LinSlaveController`	**PROTOCOL VERSION** `LIN 2.0`. The fiel

For **TIME BASE** and **TIME BASE JITTER**, change the unit box to `ms` first and then type `1`. The saved value is `0.001`, in seconds.

Connectors (Chapters 3 to 5)

Node	Menu	Default name	NAME	CONTROLLER	Cluster / channel
`ECU`	**Add Connector > CAN**	`ECU_Can_Connector_0`	`CanConnector`	`CanController`	`CanCluster` / `PhysicalChannel`
`ZONE`	**Add Connector > CAN**	`ZONE_Can_Connector_0`	`CanConnector`	`CanCommController`	`CanCluster` / `PhysicalChannel`
`ZONE`	**Add Connector > Ethernet**	`ZONE_Ethernet_Connector_0`	`EthernetConnector`	`EthernetCommController`	`EthernetCluster` / `PhysicalChann`
`HPC`	**Add Connector > Ethernet**	`HPC_Ethernet_Connector_0`	`EthernetConnector`	`EthernetController`	`EthernetCluster` / `PhysicalChann`
`ECU`	**Add Connector > LIN**	`ECU_Lin_Connector_0`	`LinConnector_Master`	`LinController_Master`	`LinCluster` / `PhysicalChannel`
`SLAVE_ECU`	**Add Connector > LIN**	`SLAVE_ECU_Lin_Connector_0`	`LinConnector`	`LinSlaveController`	`LinCluster` / `PhysicalChannel`

After connecting, **CONNECTED CHANNEL** shows `EthernetCluster / PhysicalChannel (VLAN: 13)` on the Ethernet connectors and `LinCluster / Phys`

Clusters and Channels (Chapters 3 to 5)

Cluster	Created from	Default name	Panel title	CHANNEL NAME	Channel value
`CanCluster`	`CanConnector` of `ZONE` > + > **Create CommunicationCluster [Can]**	`CanCluster_0`	**CAN CLUSTER**	`PhysicalChannel` (default)	—
`CanCluster` (bus-off recovery)	**CAN CLUSTER** panel > **BUS-OFF RECOVERY**	—	**CAN CLUSTER**	—	**BOR TIME L1** `100` `ms`, **BOR TIME L2** `200` `ms` **BOR COUNTER L1 TO L2** `2`, **BOR TIME TX ENSURED** `100` `ms` (stored `0.1`, `0.2`, `2`, `0.` **MAIN FUNCTION PERIOD** empty
`EthernetCluster`	`EthernetConnector` of `ZONE` > + > **Create CommunicationCluster [Ethernet]**	`EthernetCluster_0`	**ETHERNET CLUSTER**	`PhysicalChannel` (default)	**VLAN ID** `13`
`LinCluster`	`LinConnector` of `SLAVE_ECU` > + > **Create CommunicationCluster [Lin]**	`LinCluster_0`	**LIN CLUSTER**	`PhysicalChannel` (default)	**IDLE TIMEOUT** empty (`10` `ms` is refused)

Connect the second connector with **+** > **Connect to Channel...** > the cluster name > **PhysicalChannel** (**PhysicalChannel (VLAN: 13)** for Ethernet). You do not `CanCluster` and `19200` into `LinCluster`.

Endpoints and VLAN (Chapter 4)

Reference design element	Value	Where you enter it	Export name
`ZONE_Endpoint`	`192.168.0.19` (`FIXED`)	**UNICAST IPV4 ADDRESS** and **FIXED** of **ADDRESS SOURCE** on the `ZONE` connector	`ZONE_EthernetConnector_Endpo`
`HPC_Endpoint`	`192.168.0.11` (`FIXED`)	**UNICAST IPV4 ADDRESS** and **FIXED** of **ADDRESS SOURCE** on the `HPC` connector	`HPC_EthernetConnector_Endpoi`
`SOMEIP_SD_Multicast`	`224.0.0.1`	**MULTICAST IPV4 ADDRESS** on the `HPC` connector. **MULTICAST PORT** is left empty	`HPC_EthernetConnector_Multica`
`Vlan_13`	`VLAN-IDENTIFIER` `13`	**VLAN ID** of the channel	`VLAN_13`

NM Clusters and TP Configurations (Chapters 3 to 5)

NM, TP and the time domain are turned on per cluster with **Enable** in the **Communication Cluster Configuration** window (double-click the cluster). The refe tutorial leaves them all off.

Cluster	Tabs of the window	Tabs left off	C
`CanCluster`	**CLUSTER**, **NM CLUSTER**, **TRANSPORT PROTOCOL**, **TIME DOMAIN**	**NM CLUSTER**, **TRANSPORT PROTOCOL**, **TIME DOMAIN** (hollow dot in front of the name)	3
`EthernetCluster`	**CLUSTER**, **NM CLUSTER**, **TIME DOMAIN**	**NM CLUSTER**, **TIME DOMAIN** (hollow dot in front of the name)	4
`LinCluster`	**CLUSTER**, **TRANSPORT PROTOCOL**	**TRANSPORT PROTOCOL**. Chapter 5 does not open the window	5

A LIN cluster has no **NM CLUSTER** tab, because NM does not apply to LIN.

Defects in the Validation Chapter (Chapter 6)

Validation starts from a design with the three defects below and restores the values of LIN Configuration.

Location	Defect value	Fixed value	Validation error
`EthernetConnector` of `HPC` > **UNICAST IPV4 ADDRESS**	`224.0.0.1`	`192.168.0.11`	`constr_3288`
`LinConnector` of `SLAVE_ECU` > **CONTROLLER**	Empty (`Select controller...`)	`LinSlaveController`	`constr_5398`
`LinSlaveController` of `SLAVE_ECU` > **PROTOCOL VERSION**	Empty (no button on)	`LIN 2.0`	`constr_5449`

Project File Layout

```
VehicleDesign/
├── VehicleDesign.io.prj
├── design/
│   ├── IVC.io.model                    Chapters 1 to 6
│   ├── SYSTEM.io.model                 Chapters 1, 2
│   └── VFB.io.model                    Chapter 1
└── export/
    ├── VehicleDesign_COM.arxml         Chapter 7
    ├── VehicleDesign_DataType.arxml    Chapter 7
    ├── VehicleDesign_SWC.arxml         Chapter 7
    └── VehicleDesign_System.arxml      Chapter 7
```

- `IVC.io.model` holds the nodes, controllers, connectors and clusters (and view state, which is not exported). `SYSTEM.io.model` holds **VERSION**. This
- The export writes only to `export/`. The folders `doc`, `code`, `workflow` and `import` are not used.

Differences from the Reference Design

Your export holds almost the same design as `System_Design.arxml`: 27 of its 29 named elements have a counterpart, and only the two partitions are mi
Export Differs from the Reference Design" of ARXML Export.

Kind	Examples
Deliberate departures	`HPC` connector with its own controller and `192.168.0.11`; no `MAC-LAYER-TYPE`; `BUS-OFF-RECOVERY` on `CanCluster` for the AUTOSAR CP Tutorial
PAIO naming rules	Packages `/System/EcuInstance`, `/System/MachineDesign`, `/System/CommunicationCluster/…`; endpoint names `<node>_<connector>_Endpoint`; '
Added by PAIO	`BAUDRATE` `500000` and `19200`, SD configuration and socket, 49 basic data types, root composition `System`
No field on screen	Partitions, `CATEGORY` `WIRED`, `PROTOCOL-NAME` `IEEE 802.3`, LIN `BUS-IDLE-TIMEOUT-PERIOD` `0.01`

Common Problems

Symptom or validation message	Cause	Fix
`constr_3288`: unicast IPv4 `224.0.0.1` is not in the unicast range	The `HPC` connector holds a multicast address	Click the `HPC` connector on the canvas, enter `192.168.0.11`, `Ctrl+S`
`constr_5398`: `LinConnector` on `SLAVE_ECU` has no commController reference	**CONTROLLER** is empty	Choose `LinSlaveController`, `Ctrl+S`
`constr_5449`: `LinSlaveController` has no protocolVersion	No **PROTOCOL VERSION** button is on	Click the controller, then `LIN 2.0`, `Ctrl+S`
TIME BASE becomes 1 s, **IDLE TIMEOUT** 10 s	The unit box defaults to `s`	Set the unit to `ms` before typing; leave **IDLE TIMEOUT** empty
An extra cluster appears	**Create CommunicationCluster** was used on the second connector	Use **Connect to Channel...** for the second connector
Connected Connectors order is reversed	The cluster was created from the other connector	Create CAN and Ethernet clusters from `ZONE`, the LIN cluster from `SLAVE_ECU`
Schema error on `XMII` in the export	**MAC LAYER TYPE** was clicked	Do not click **MAC LAYER TYPE**
ADDRESS SOURCE is all off	**FIXED** was not clicked	Click **FIXED** on both Ethernet connectors
A value is not applied	Fields commit only when you leave them	Press `Tab` after typing
Line numbers in ARXML Export do not match	The nodes were added in another order	Add `ECU` → `ZONE` → `SLAVE_ECU` → `HPC`
The **+** badge of the `HPC` connector is hard to click	The badge is small, and near the right edge the menu opens to the left	Zoom in, or **Zoom Out** once more and reopen the men

Checklist

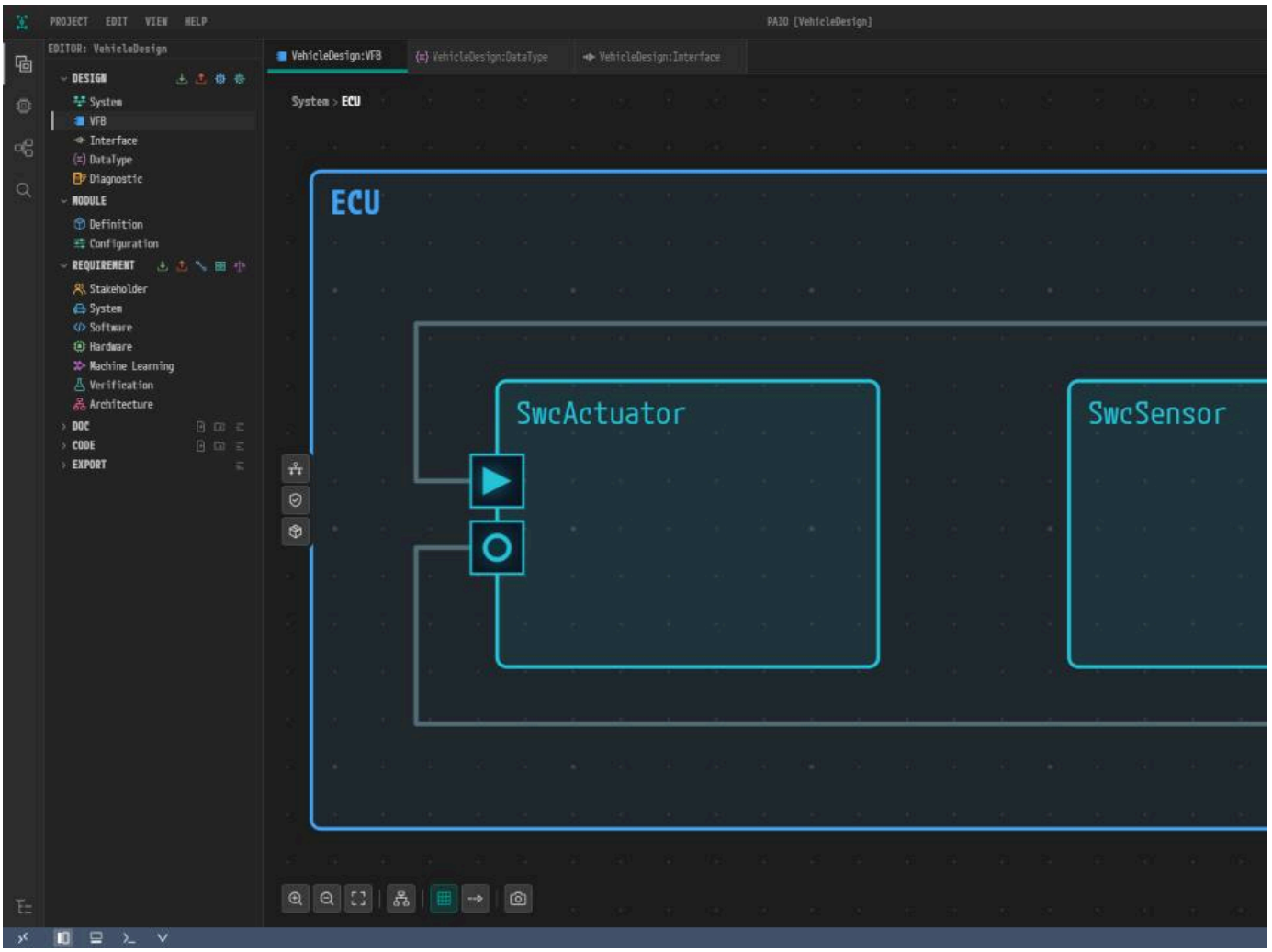
- [ ] Every item of "Checklist of the Finished Design" matches.
- [ ] Your values match "All Value Tables".

Next Step

The AUTOSAR CP Tutorial continues with this project. To start it from a finished copy, download VehicleDesign-system.zip.

AUTOSAR CP Tutorial

This tutorial puts two software components on the ECU node `ECU` of the project `VehicleDesign` in PAIO, configures the BSW with its CAN stack and netw
CONFIGURATION, and generates the ECU's code with PRTE, the Classic Platform code generator built into PAIO. It is the second of three parts on one projec
AP.



The software you build: `SwcSensor` and `SwcActuator` in the composition `ECU`, joined by two connectors.

What You Build

`SwcSensor` has a 20 ms runnable that may write a speed value and ask `SwcActuator` for a scale value; `SwcActuator` receives the value and answers the
bodies of the runnables are code you write later.

SW-C	Port	Interface	What it does
`SwcSensor`	`PpSpeed` (provides)	`IfSpeed` (sender-receiver)	May write `speed`, started every 20 ms
`SwcSensor`	`RpScale` (requires)	`IfScale` (client-server)	May call `GetScale`
`SwcActuator`	`RpSpeed` (requires)	`IfSpeed` (sender-receiver)	Runs when `speed` arrives
`SwcActuator`	`PpScale` (provides)	`IfScale` (client-server)	Answers `GetScale`

What You Need

- PAIO, installed and able to run.
- The project `VehicleDesign` finished in the AUTOSAR System Tutorial. To skip that tutorial, download VehicleDesign-system.zip, unpack it and open t
  the **PROJECT** menu.

- For chapter 6, the Rte values file VehicleDesign_Rte.arxml. Save it before you start; if the link opens it in a browser tab, use Save As.
- A screen of 1920×1080 or larger.
- A rough idea of ports, runnables and the RTE. Each chapter shows what to enter on which screen.

Chapters

Chapter	What you do	Editor used
1. Composition and ECU Mapping	Create the composition `ECU` and map it to the node `ECU`	VFB editor
2. Data Type	Create the data type `SpeedType`	DataType editor
3. Interfaces	Create the interfaces `IfSpeed` and `IfScale`	Interface editor
4. SW-Cs and Ports	Create `SwcSensor` and `SwcActuator` with two ports each and connect them	VFB editor
5. Internal Behavior	Add four runnables with their access points and RTE events	SWC Internal Behavior
6. PRTE Setup	Bring in the BSW modules, set `EcuM`, `BswM`, `Mcu` and `Det`, and import the Rte values file	ECU Configuration, Configuration
7. Os Configuration	Enter the OS settings, the tasks, the alarms and the counter by hand	Configuration
8. Rte Configuration	Map the RTE events to the tasks and complete the BSW module instances	Configuration
9. Communication Configuration	Enter `EcuC`, `Can`, `CanIf`, `ComM`, `CanSM`, `Nm`, `CanNm`, `LSduR`, `PduR` and `Com` by hand	Configuration
10. Validation and Code Generation	Validate the design and generate the code with PRTE	VALIDATION panel, PRTE Code Generation
Appendix	All values, common problems	—

How to Read This Tutorial

- **Follow the chapters in order.** Each one starts from the result of the previous one.
- **Repetitive work is shown once.** One item is shown as steps; the rest are given in value tables that you enter the same way.
- UI labels are written in **bold** as the screen shows them, and model element names and values in `code`.

Next Step

Start with Composition and ECU Mapping.

Composition and ECU Mapping

What You Will Do

In this chapter you create the composition `ECU` that holds this ECU's software and map it to the node `ECU`. Once it is mapped, the ECU icon on the comp where chapter 6 brings in the BSW modules.

Before You Begin

- Start from the project `VehicleDesign` finished in the AUTOSAR System Tutorial. The node `ECU` was made there and is connected to the CAN bus and
- If you skipped that tutorial, download the finished project VehicleDesign-system.zip, unpack it and open the folder `VehicleDesign` with **Open Proje**

Where	Item	Value
Composition panel	**NAME, AS PROTOTYPE**	`ECU`
Composition menu	**ECU/HPC Mapping**	**ECU [ECU]**

Creating the Composition

1. Click **VFB** under **DESIGN**. The tab `VehicleDesign:VFB` opens.

2. Right-click an empty spot on the canvas and click **Add Composition**.

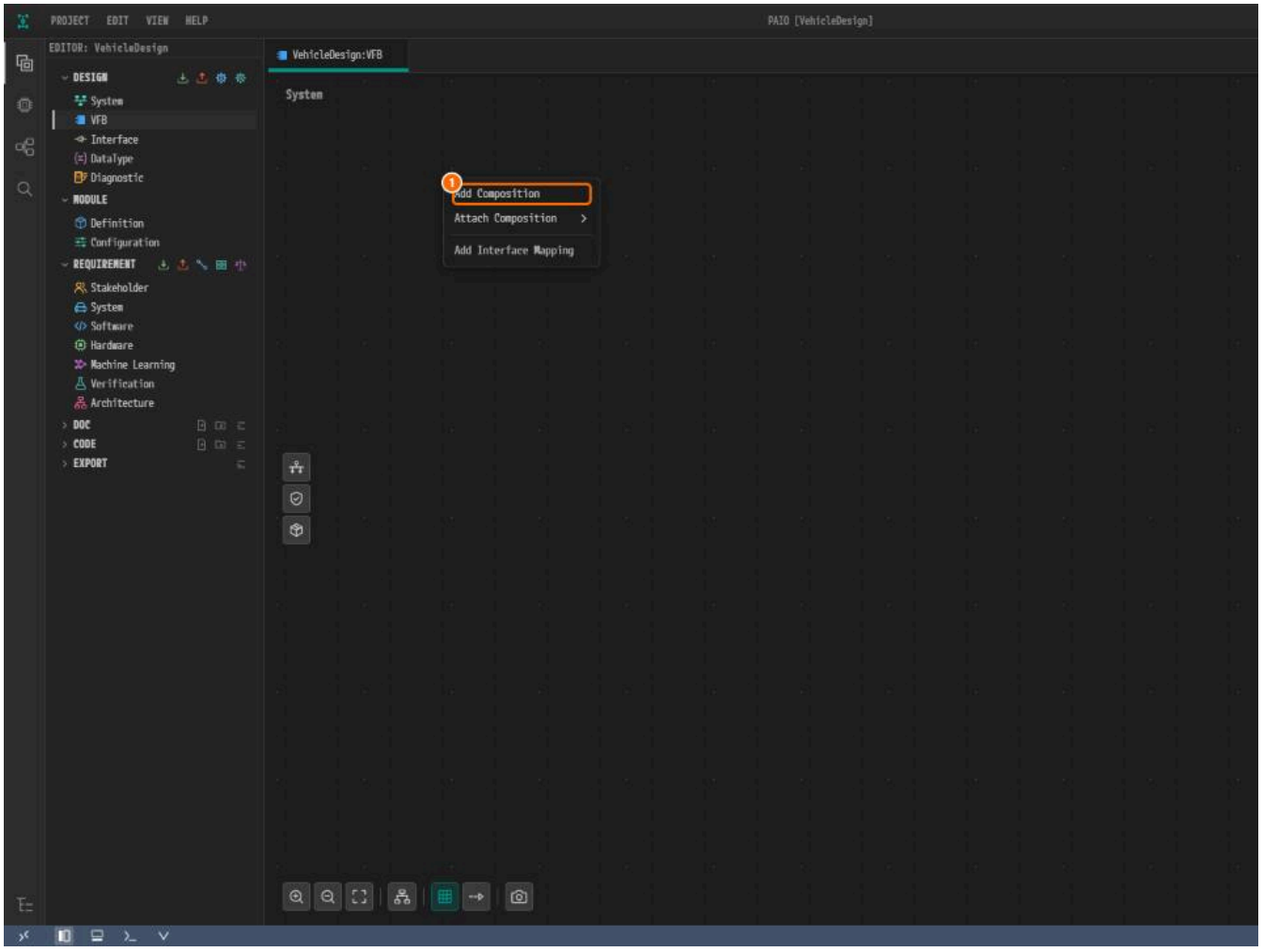


*Figure 1-1. ① **Add Composition** in the canvas menu.*

3. Click the new composition. In the panel, enter `ECU` in both **NAME** and **AS PROTOTYPE**, pressing `Enter` after each, and save with `Ctrl+S`.

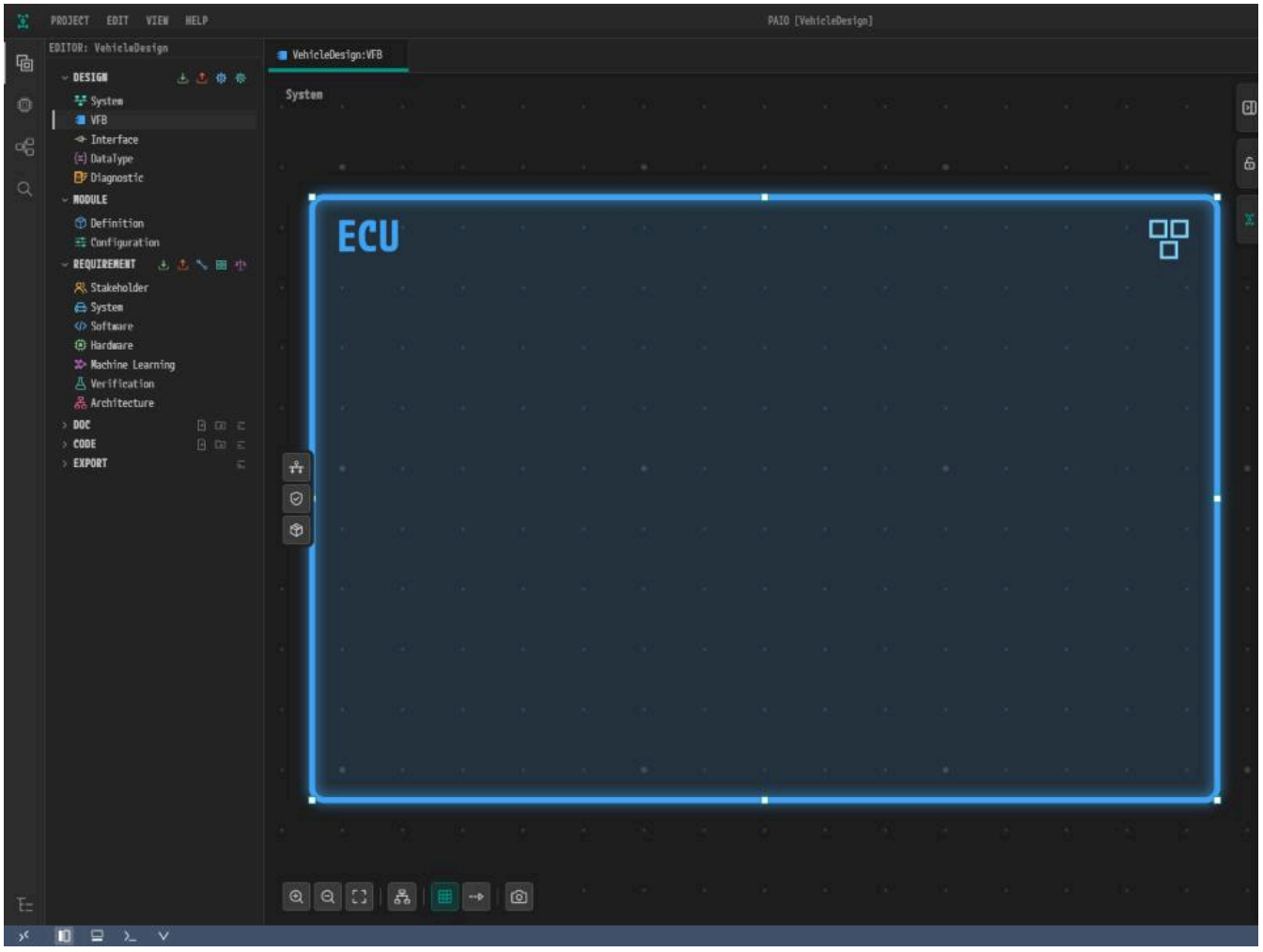


*Figure 1-2. ① **NAME**, ② **AS PROTOTYPE**.*

Mapping It to the ECU Node

1. Right-click inside the composition and hover over **ECU/HPC Mapping**. The submenu lists the four nodes of the System editor. Click **ECU [ECU]**. Save wi

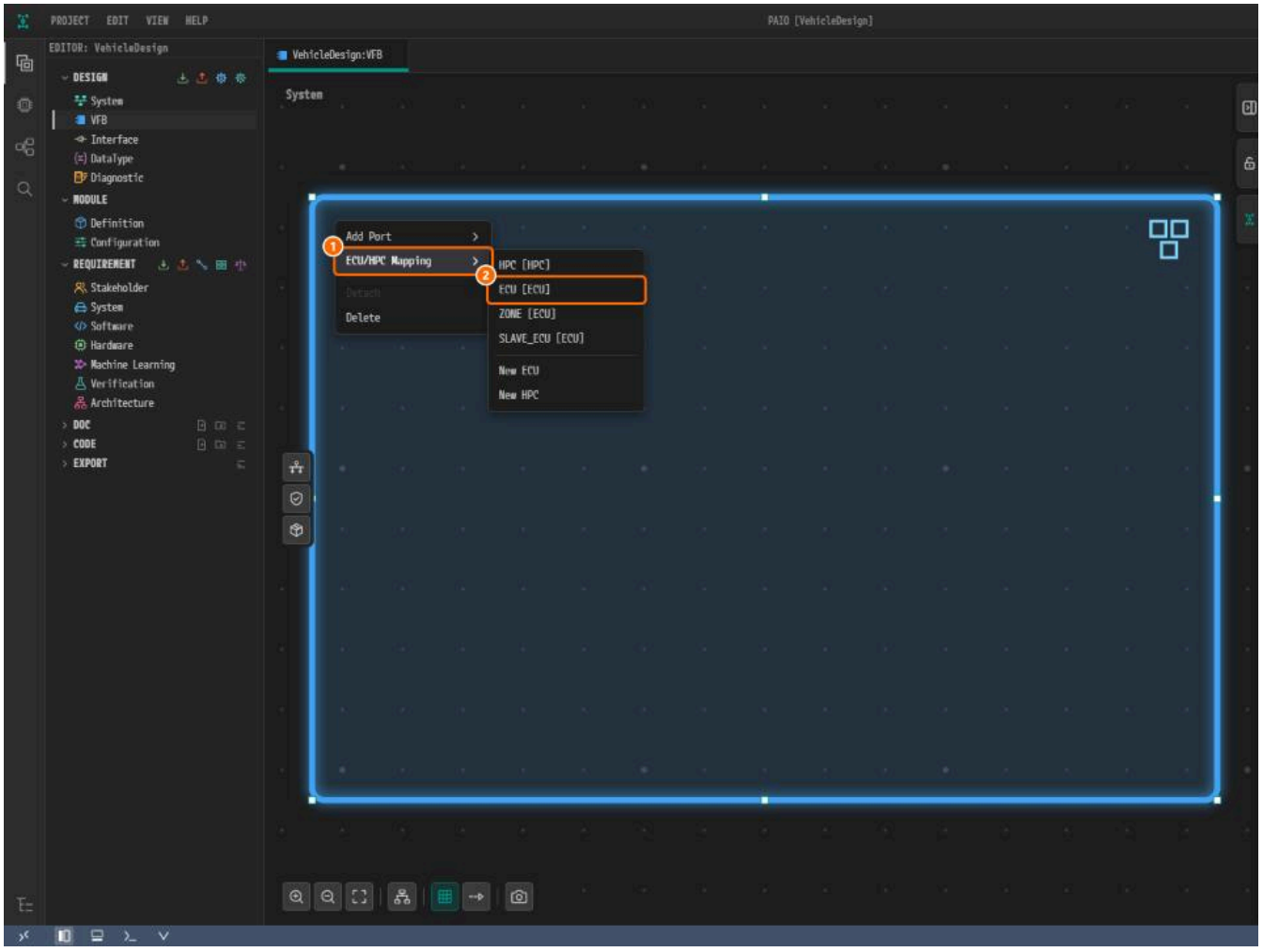


*Figure 1-3. ① **ECU/HPC Mapping**, ② **ECU [ECU]**.*

2. Double-click the composition to go inside it. The breadcrumb reads **System > ECU**, and an ECU icon appears at the top right of the frame.

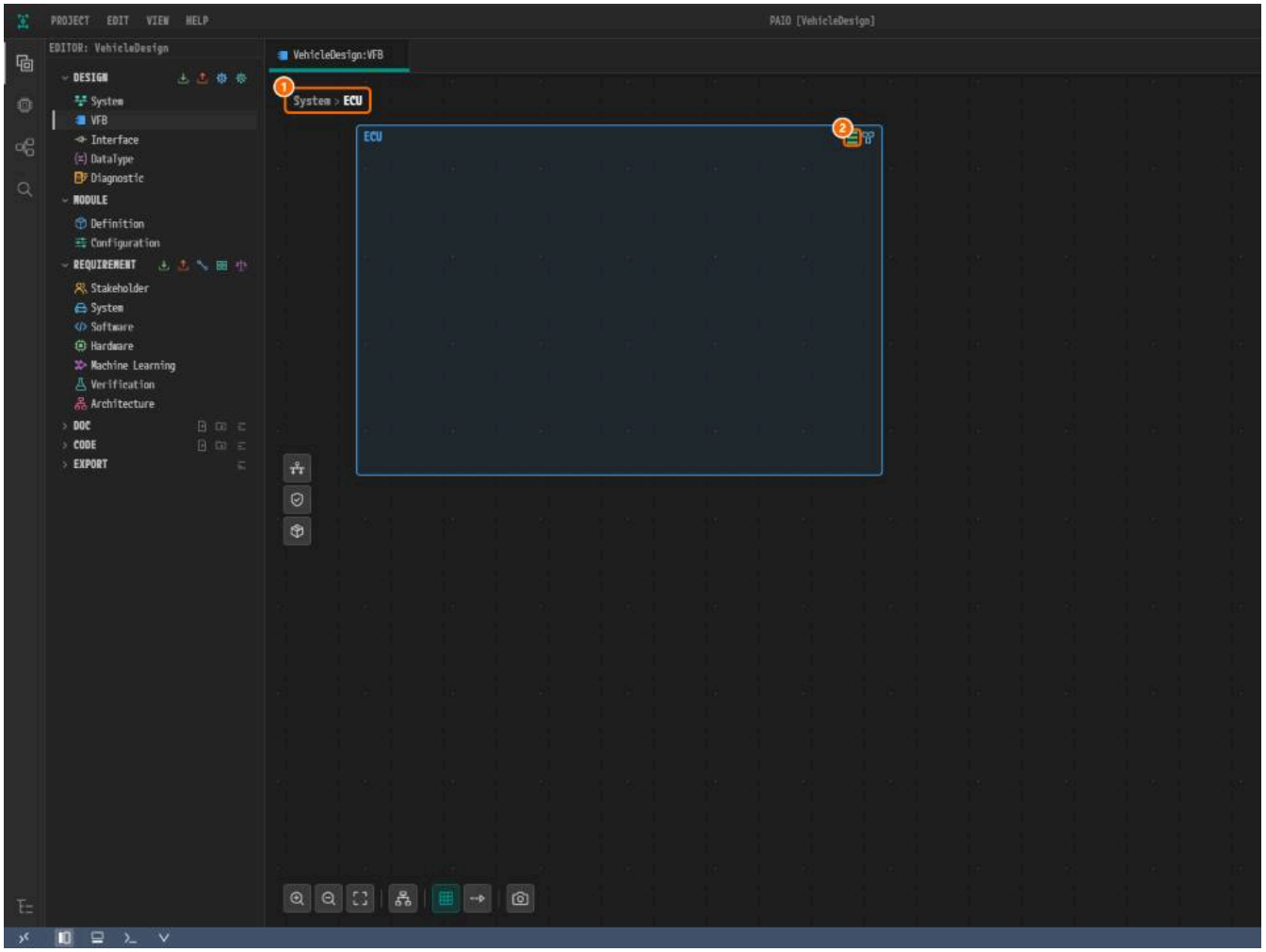


Figure 1-4. ① The breadcrumb, ② the ECU icon. The icon appears because the composition is mapped to an ECU node.

Why Is It Designed This Way?

- **The BSW comes later.** The BSW modules and their configuration are brought in once, in chapter 6, from this ECU's **ECU Configuration** window. Chapte

Checklist

- [ ] Inside the composition `ECU`, the breadcrumb reads **System > ECU** and the ECU icon is shown.

Next Step

In Data Type you create the data type the two SW-Cs exchange.

Data Type

What You Will Do

In this chapter you create the data type `SpeedType`, an alias of `uint16`. Both interfaces in the next chapter use it.

Before You Begin

- Start after finishing Composition and ECU Mapping.

Where	Item	Value
DataType editor	**NAME**	`SpeedType`
DataType editor	**CATEGORY**	**ALIAS**
DataType editor	**BASETYPE**	`uint16`

Creating SpeedType

1. Click **DataType** under **DESIGN**. The list holds the primitive types. Click **+** above the list.

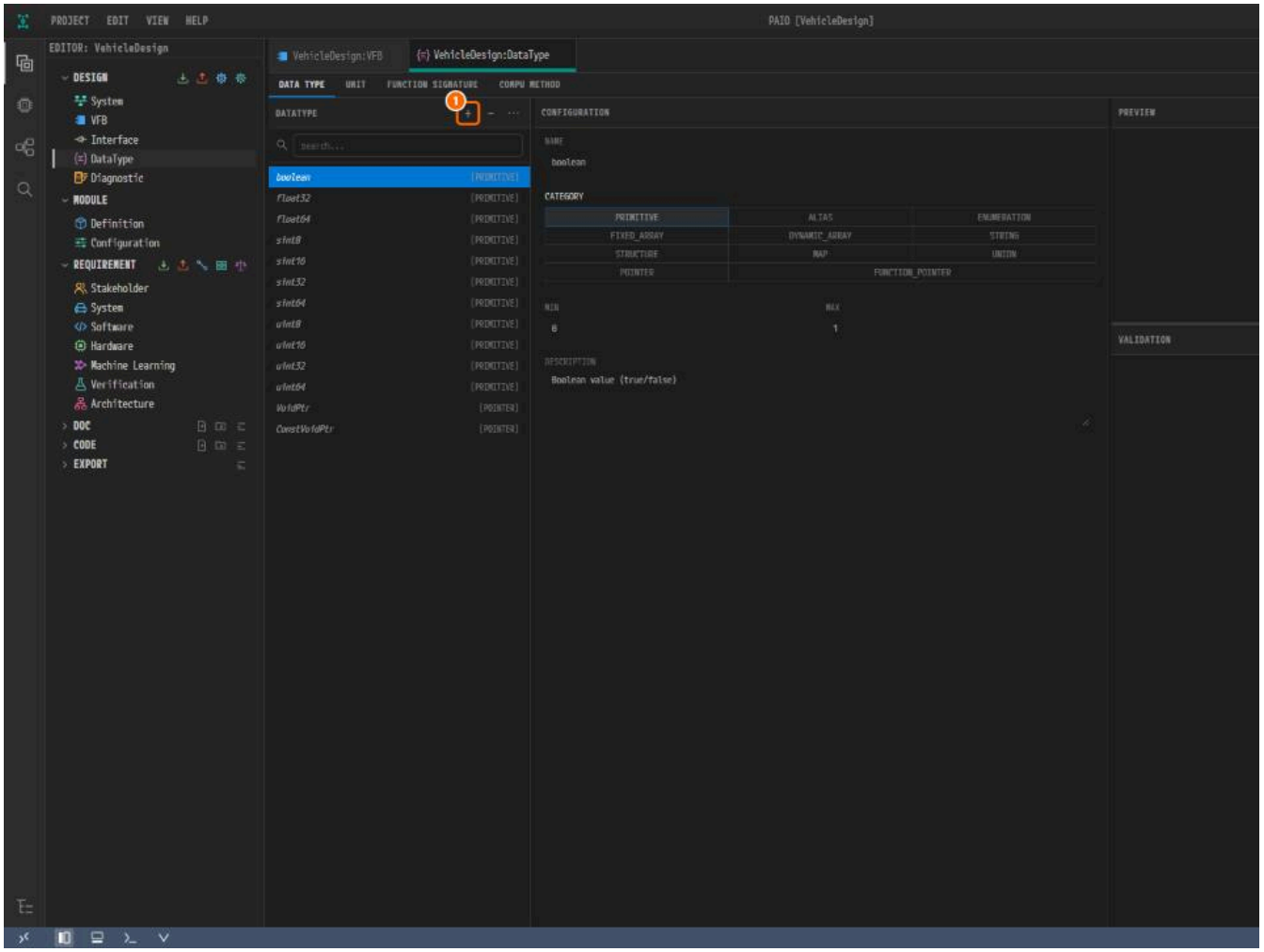


Figure 2-1. ① + (Add DataType).

2. Enter `SpeedType` in **NAME** and press `Tab`, then click **ALIAS** under **CATEGORY**. Open **BASETYPE**, type `uint16`, and click `uint16` in the list.

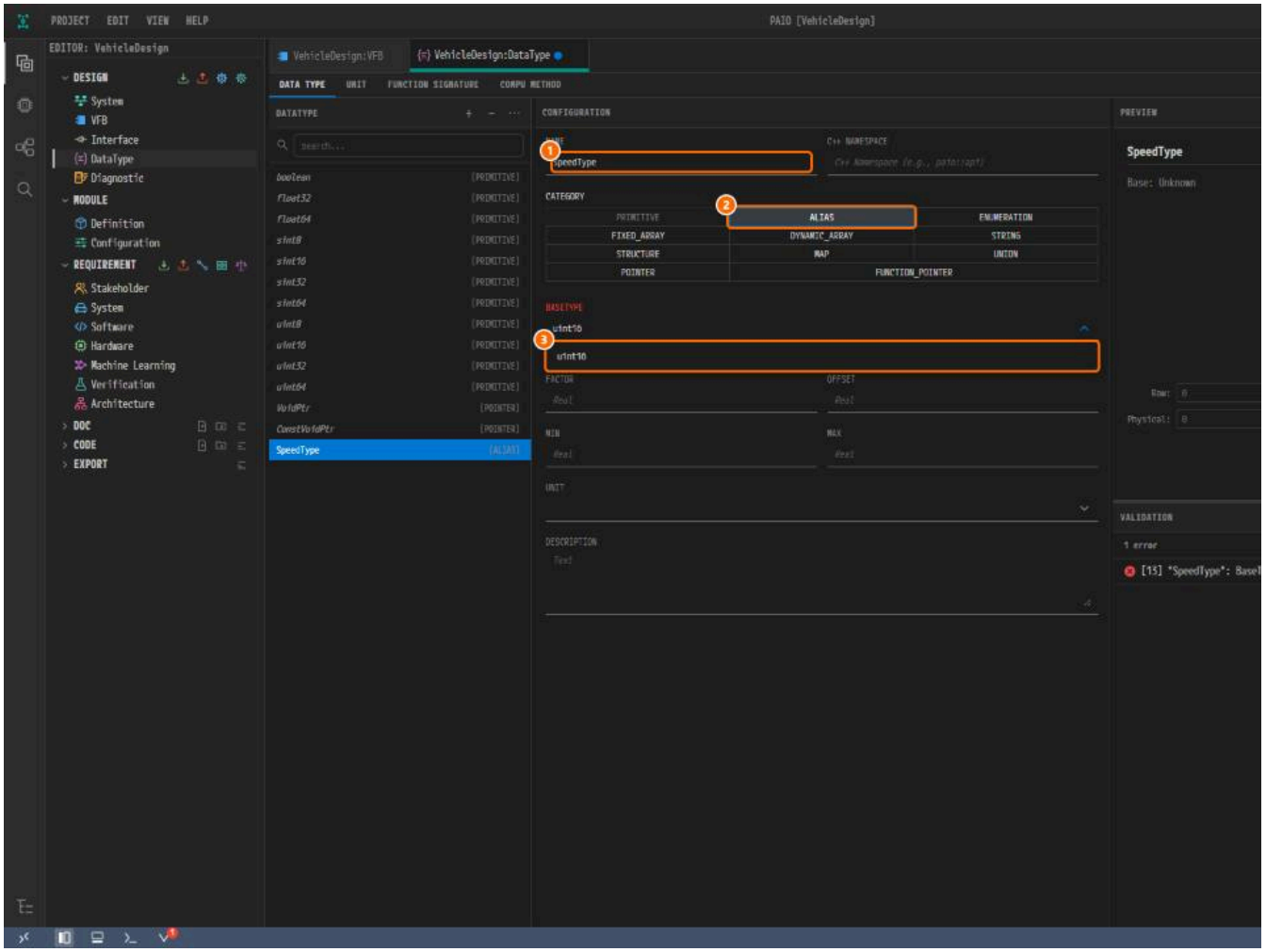


*Figure 2-2. ① **NAME**, ② **ALIAS**, ③ `uint16` in the **BASETYPE** list.*

3. Save with `Ctrl+S`.

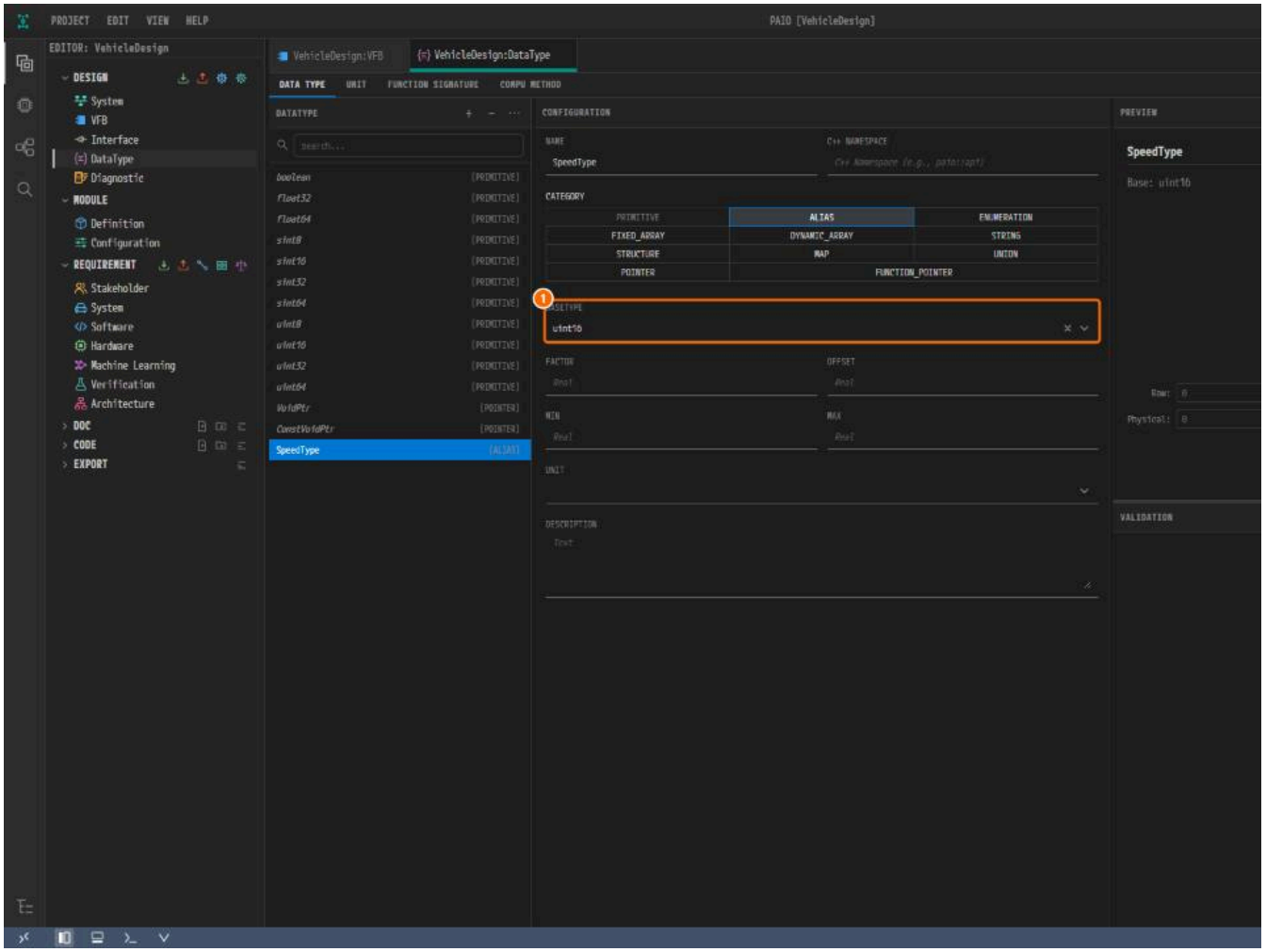


*Figure 2-3. ① **BASETYPE** = `uint16`.*

Why Is It Designed This Way?

- **An alias of `uint16`.** A named type says what the value means, and every use of it changes together if the base type ever does.

Checklist

- [ ] The list holds `SpeedType`, and selecting it shows **ALIAS** and **BASETYPE** `uint16`.
- [ ] The tab `VehicleDesign:DataType` carries no modified mark.

Next Step

In Interfaces you create the two interfaces that use `SpeedType`.

Interfaces

What You Will Do

In this chapter you create the two interfaces the SW-Cs communicate through: the sender-receiver interface `IfSpeed` for the speed value, and the clier

Before You Begin

- Start after finishing Data Type.

Interface	Tab	Element	Name	Type or direction
`IfSpeed`	**SENDER-RECEIVER**	Data element	`speed`	**TYPE** `SpeedType`
`IfScale`	**CLIENT-SERVER**	Operation	`GetScale`	
`IfScale`	**CLIENT-SERVER**	Argument of `GetScale`	`scale`	**DIRECTION** `OUT`, **TYPE** `SpeedType`

Creating IfSpeed

1. Click **Interface** under **DESIGN**, click the **SENDER-RECEIVER** tab, and click **+** above the **INTERFACES** list.

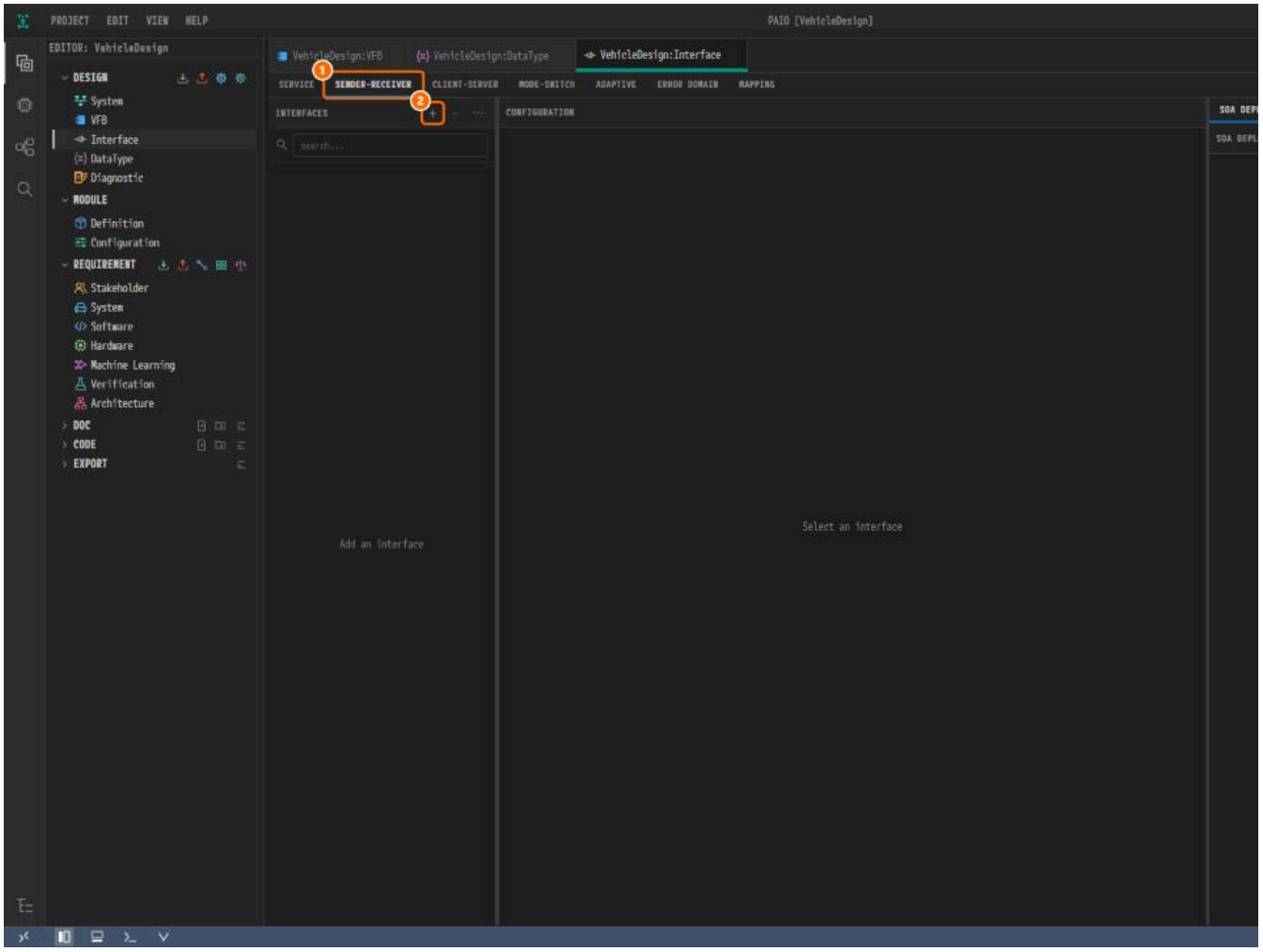


*Figure 3-1. ① **SENDER-RECEIVER**, ② **+** (Add Interface).*

2. Enter `IfSpeed` in **NAME** and press `Tab`.

3. Click **+** on the **DATA ELEMENTS** header. In the new row, enter `speed` in **NAME**, open **TYPE**, type `SpeedType` and click it in the list.

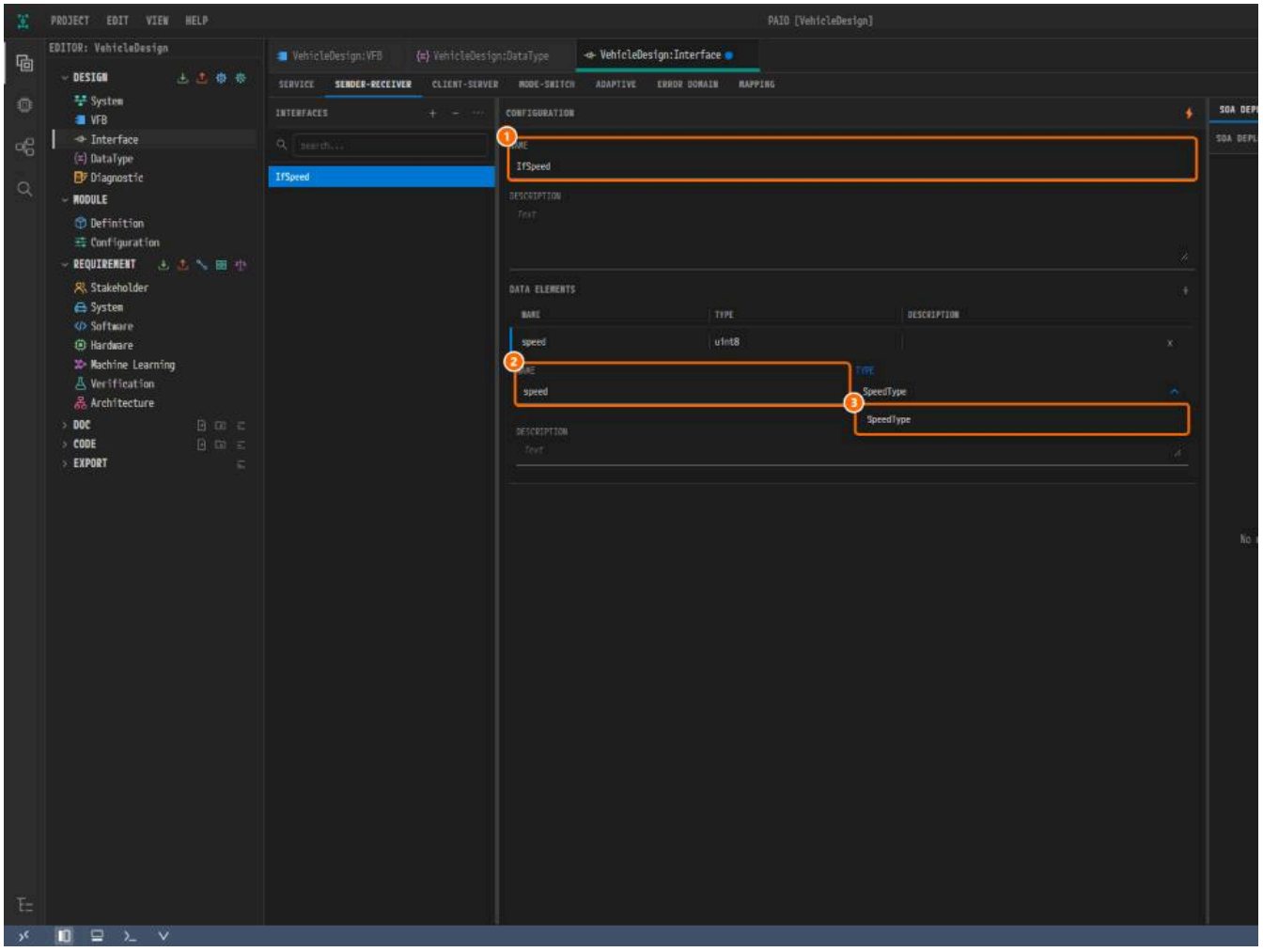


*Figure 3-2. ① The interface **NAME**, ② the data element **NAME**, ③ `SpeedType` in the **TYPE** list.*

Creating IfScale

1. Click the **CLIENT-SERVER** tab and click **+** above the **INTERFACES** list. Enter `IfScale` in **NAME** and press `Tab`.
2. Click **+** on the **OPERATIONS** header and enter `GetScale` in the operation's **NAME**.
3. Click **+** on the **ARGUMENTS** header. Enter `scale` in the argument's **NAME**, select `OUT` in **DIRECTION**, and select `SpeedType` in **TYPE**.
4. Save with `Ctrl+S`.

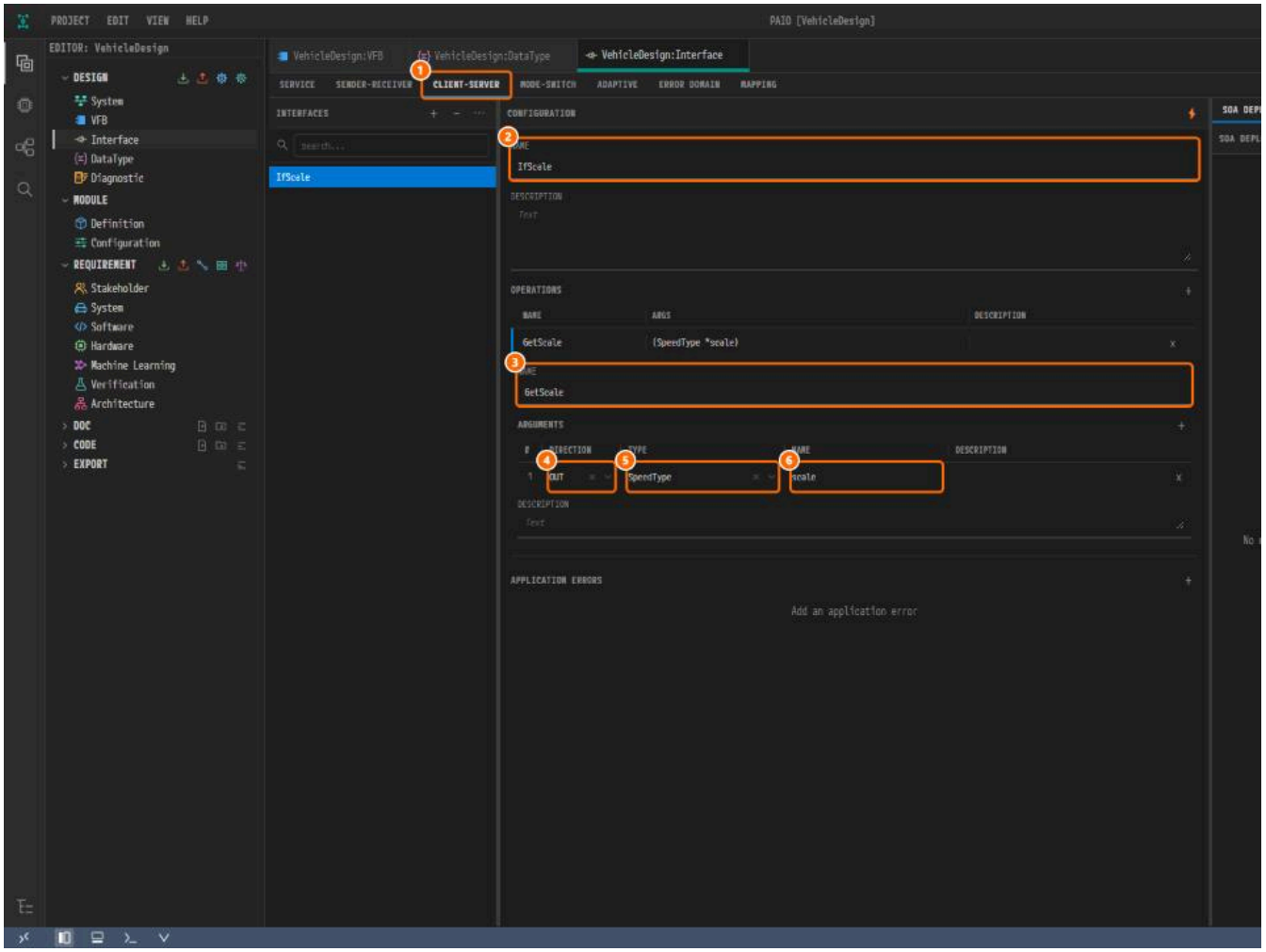


*Figure 3-3. ① **CLIENT-SERVER**, ② the interface **NAME**, ③ the operation **NAME**, ④ **DIRECTION**, ⑤ **TYPE**, ⑥ the argument **NAME**. The **ARGS** column of the o argument is passed as a pointer.*

Why Is It Designed This Way?

- **Two kinds of interface.** Sender-receiver sends a value and does not wait for anyone. Client-server asks and gets an answer. This design uses one of

Checklist

- [ ] The **SENDER-RECEIVER** tab lists `IfSpeed` with the data element `speed` of type `SpeedType`.
- [ ] The **CLIENT-SERVER** tab lists `IfScale` with the operation `GetScale` and its argument `scale`.
- [ ] The tab `VehicleDesign:Interface` carries no modified mark.

Next Step

In SW-Cs and Ports you create the two SW-Cs and type their ports with these interfaces.

SW-Cs and Ports

What You Will Do

In this chapter you create two application software components, `SwcSensor` and `SwcActuator`, inside the composition `ECU`, give each of them two port each interface.

Before You Begin

- Start after finishing Interfaces.
- Open the tab `VehicleDesign:VFB`. If you are not inside the composition `ECU`, double-click it.

SW-C	Port	Add Port item	PORT INTERFACE	Setting
`SwcSensor`	`PpSpeed`	**PPort**	`IfSpeed`	**NON-QUEUED**
`SwcSensor`	`RpScale`	**RPort**	`IfScale`	**CLIENT**
`SwcActuator`	`RpSpeed`	**RPort**	`IfSpeed`	**NON-QUEUED**
`SwcActuator`	`PpScale`	**PPort**	`IfScale`	**SERVER**

Creating the SW-Cs

1. Right-click an empty spot inside the frame. Hover over **Add SWC** and click **Application**.

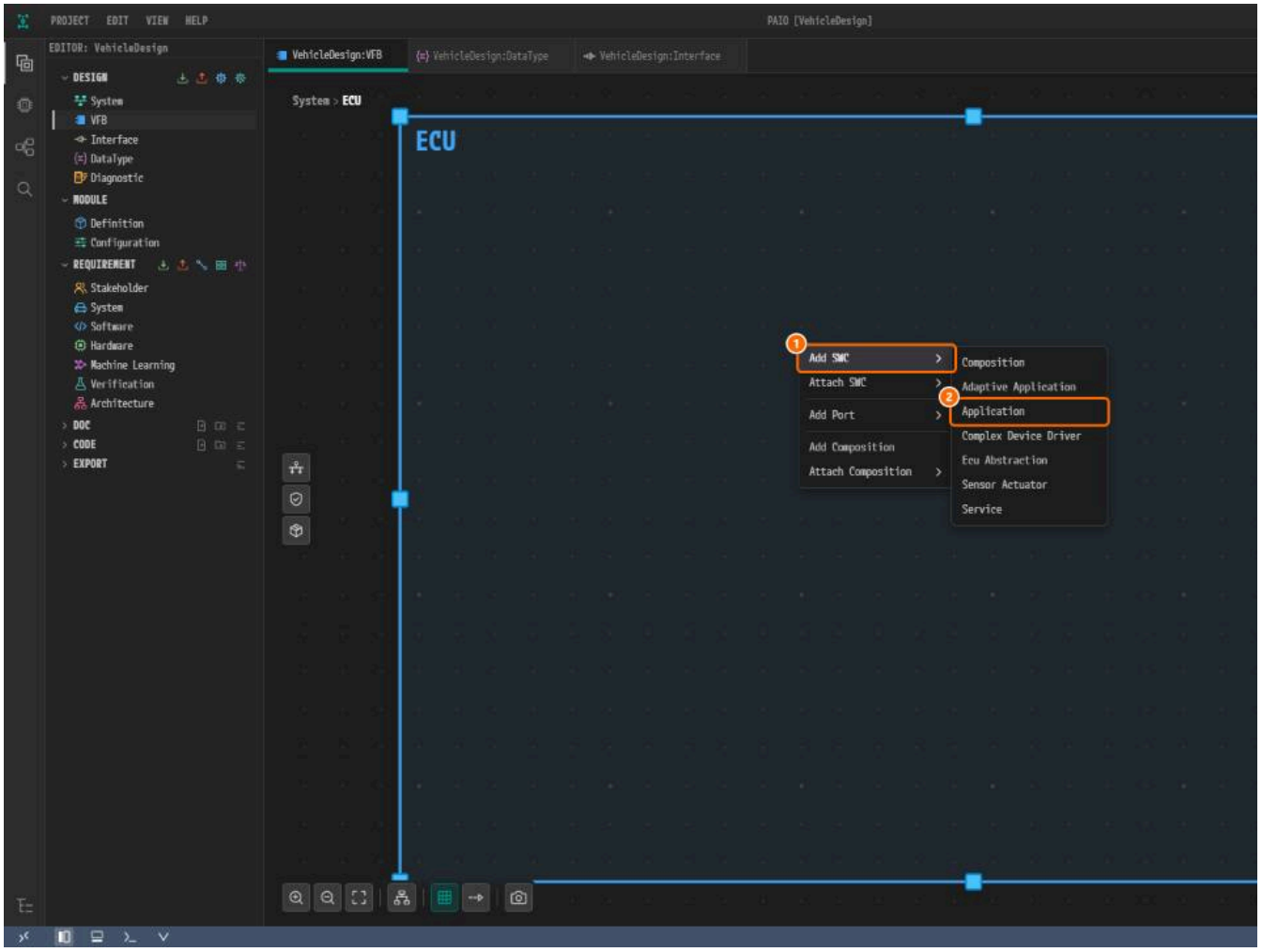


*Figure 4-1. ① **Add SWC**, ② **Application**.*

2. Click the new SW-C. In the panel, enter `SwcSensor` in both **NAME** and **AS PROTOTYPE**.

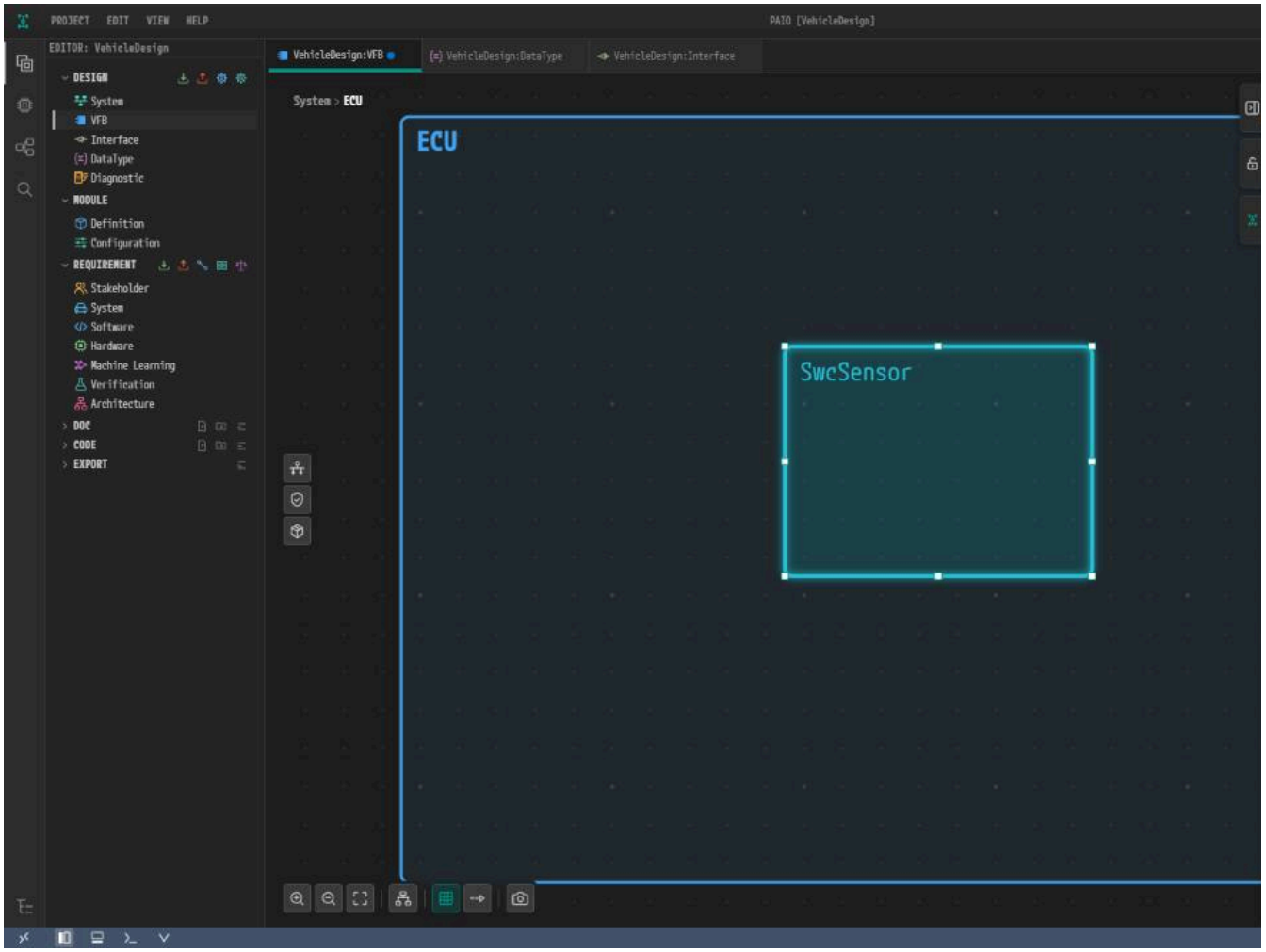


*Figure 4-2. ① **NAME**, ② **AS PROTOTYPE**.*

3. Create `SwcActuator` the same way, below `SwcSensor`.

Creating the Ports

Follow the port `PpSpeed` first.

1. Right-click near the right edge of `SwcSensor`. Hover over **Add Port** and click **PPort**. A port appears on the edge.

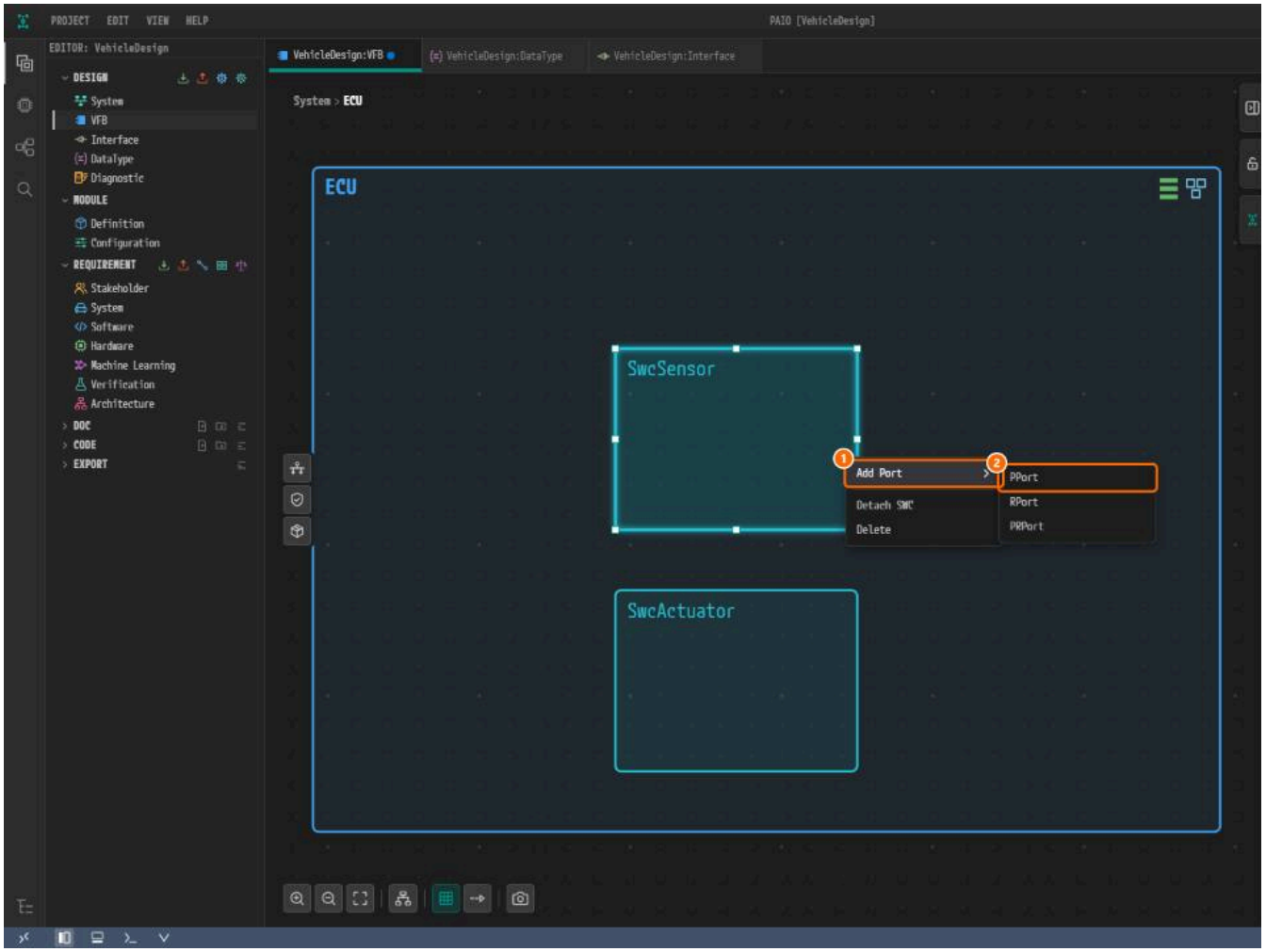


*Figure 4-3. ① **Add Port**, ② **PPort**.*

2. Click the new port. Enter `PpSpeed` in **NAME**, open **PORT INTERFACE** and click `IfSpeed`.

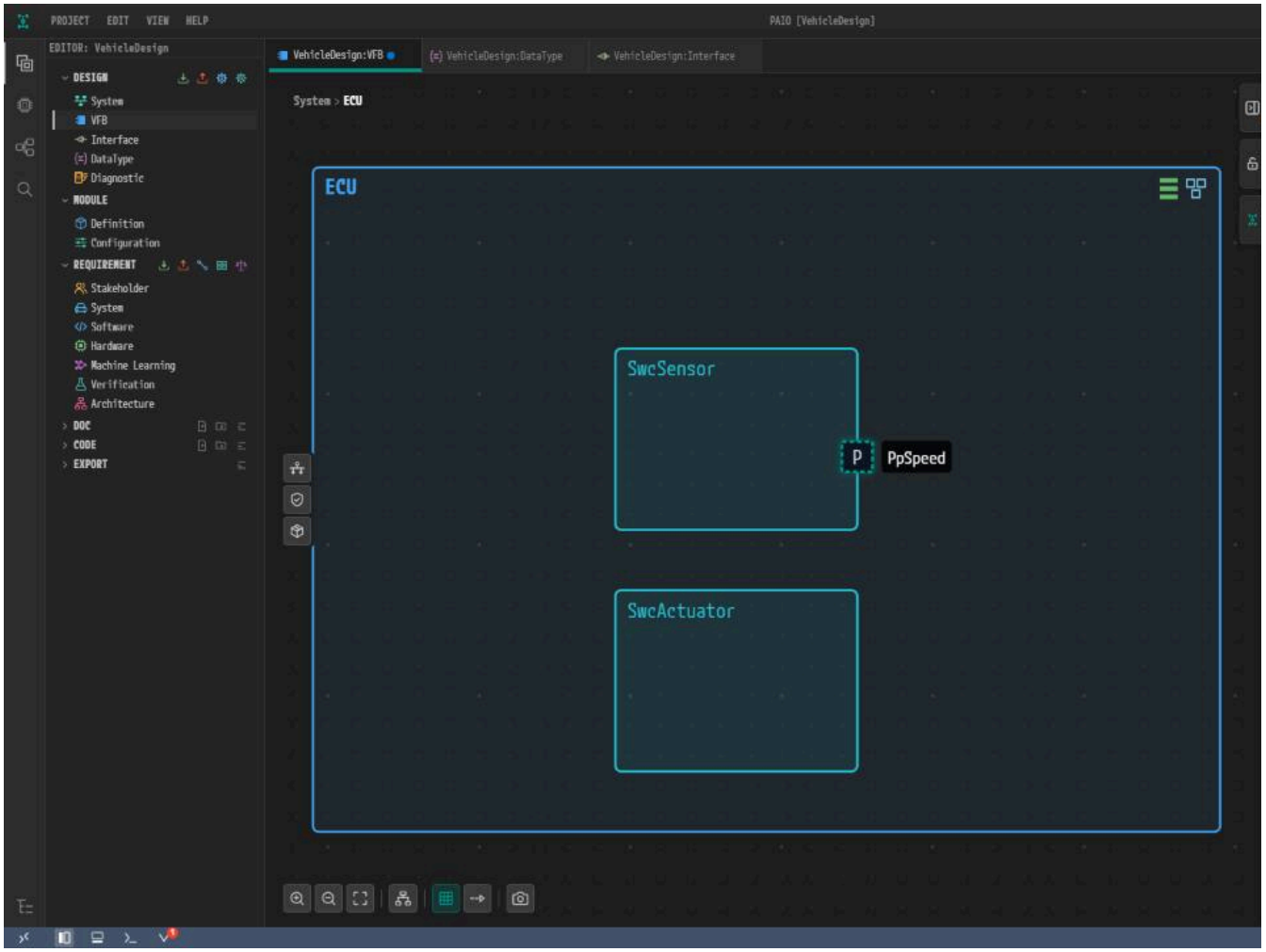


*Figure 4-4. ① **NAME**, ② `IfSpeed` in the **PORT INTERFACE** list.*

3. A row for the data element `speed` appears in the panel. Click **NON-QUEUED** in its **COMSPEC** column.

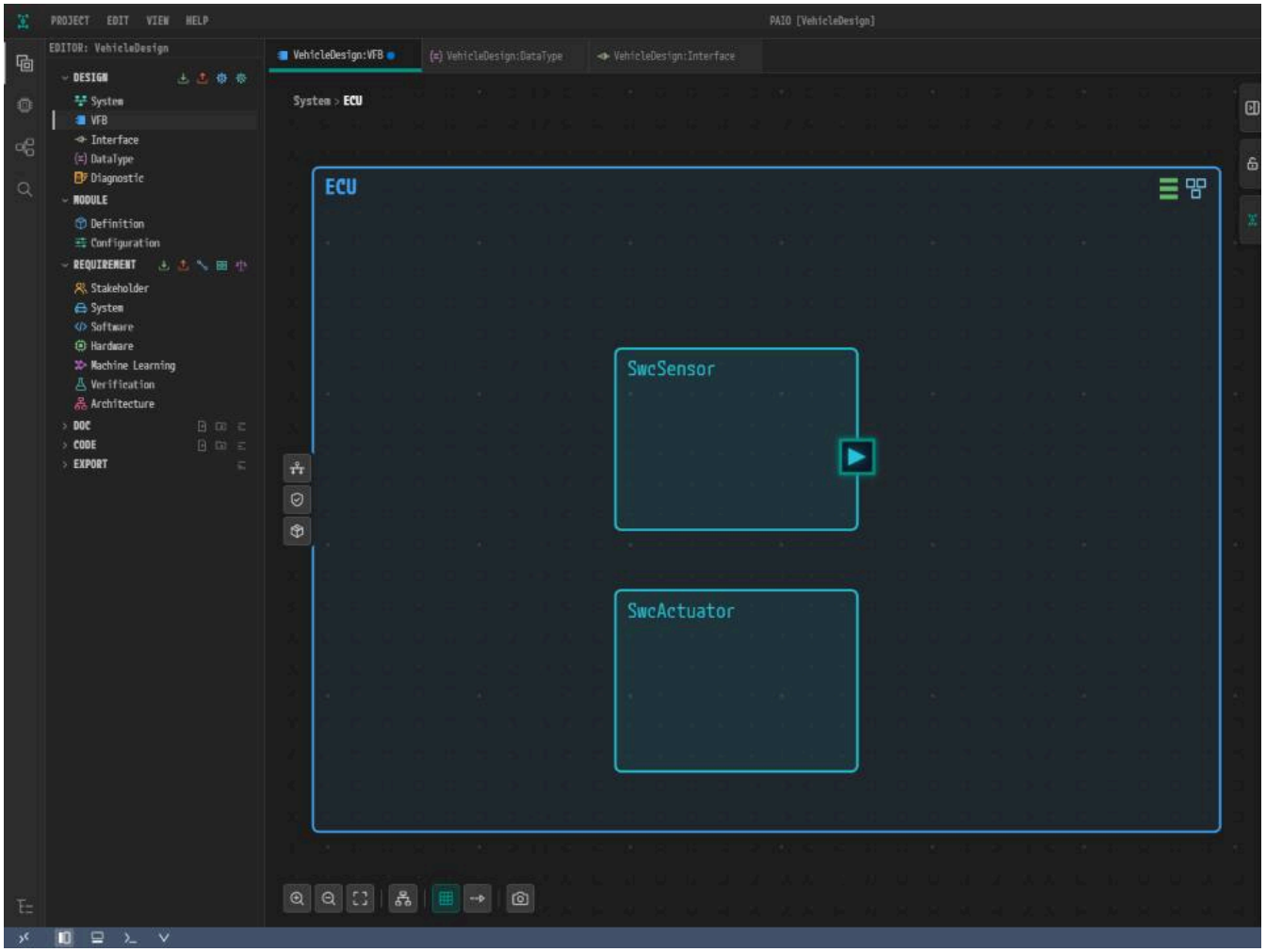


*Figure 4-5. ① **NON-QUEUED**.*

4. Create `RpScale` the same way with **RPort**, near the left edge. On a client-server port the row holds the operation `GetScale`. Click **CLIENT**.

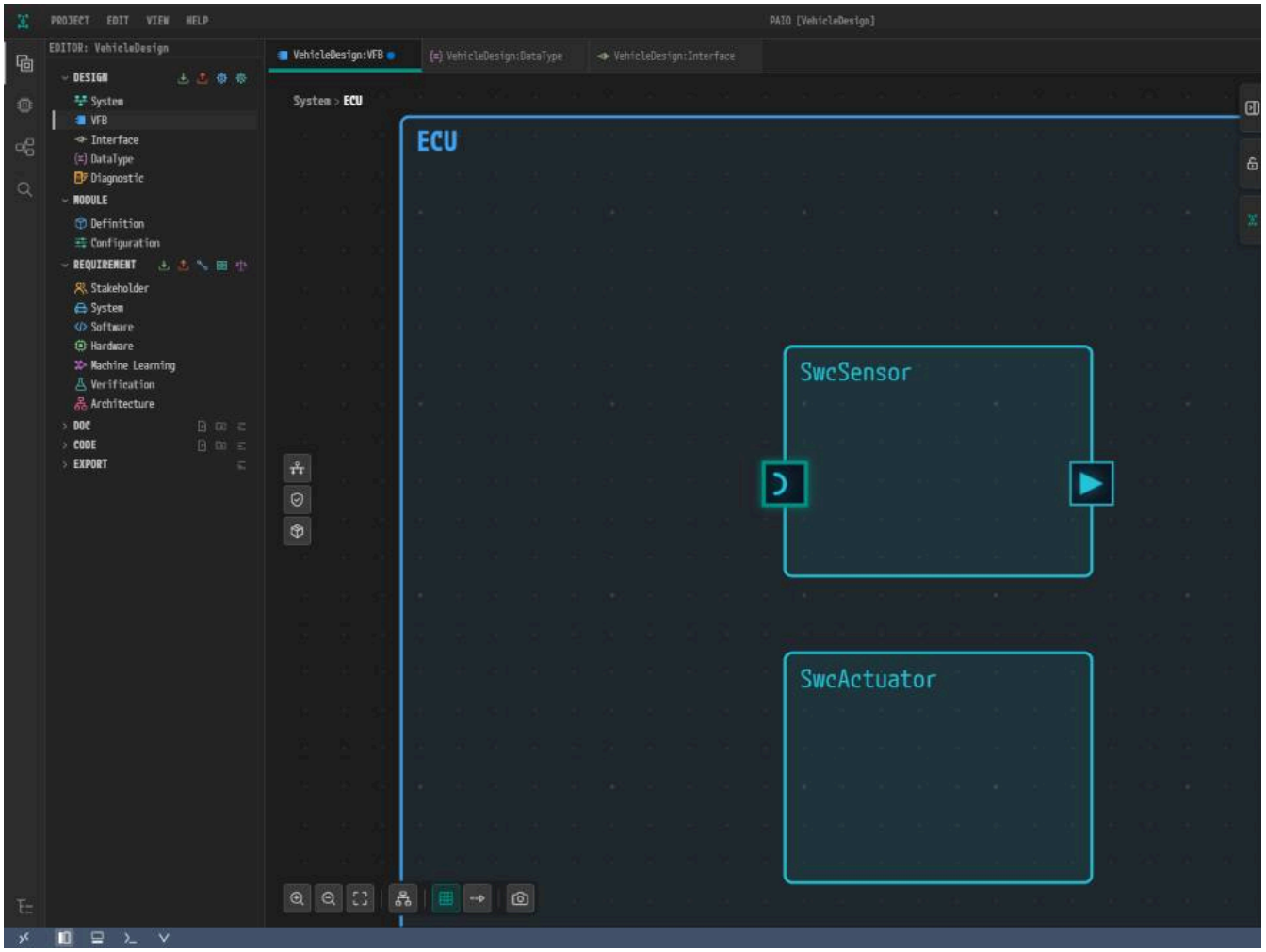


*Figure 4-6. ① **CLIENT**.*

5. Create `RpSpeed` and `PpScale` on `SwcActuator` from the table above.

6. Press `Esc`, then save with `Ctrl+S`. Click **Auto Layout** in the toolbar under the canvas to arrange the frame, and save again.

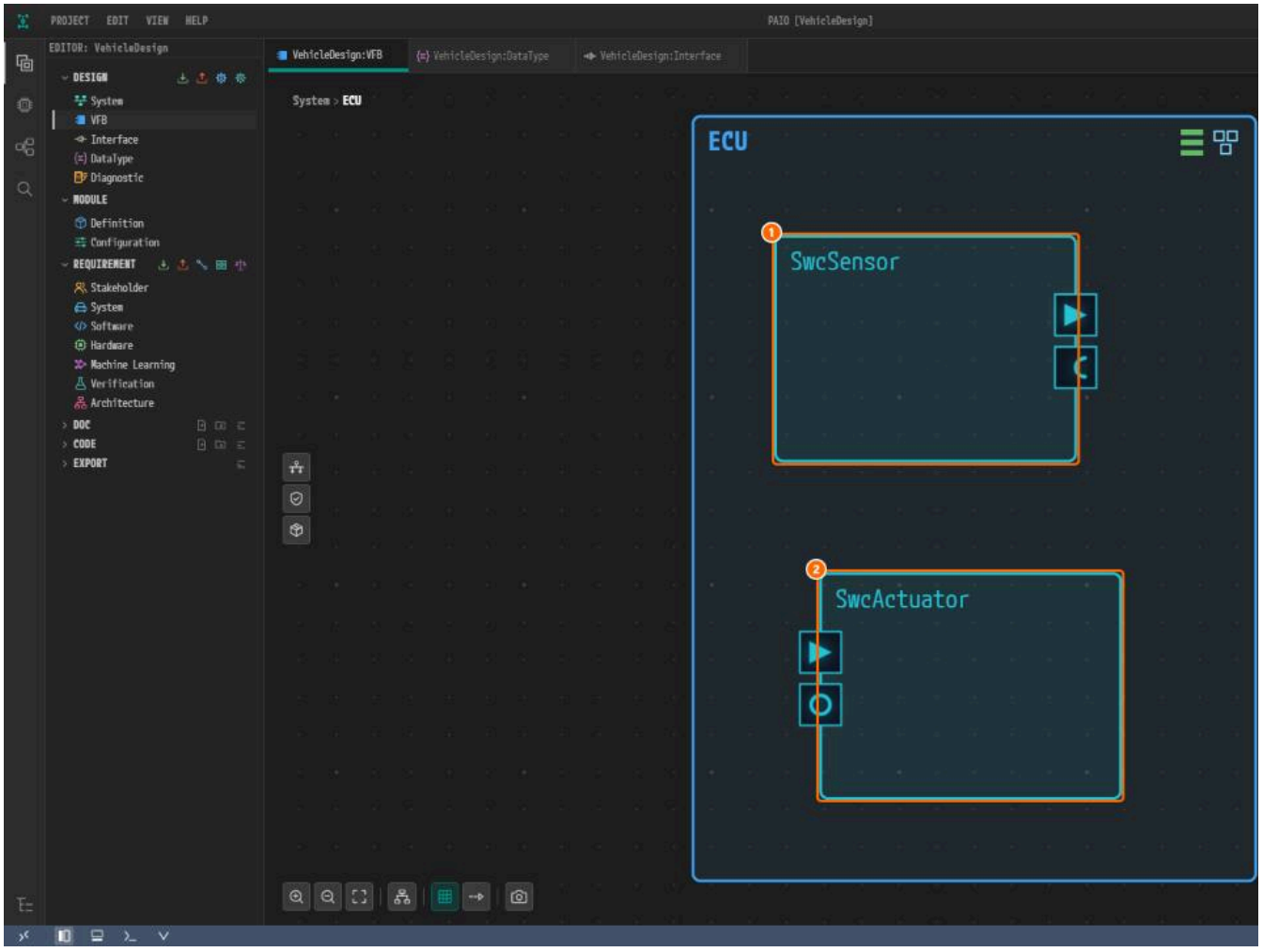


Figure 4-7. ① `SwcSensor`, ② `SwcActuator`. Auto Layout decides the positions, so the order on your screen may differ.

Connecting the Ports

Join the two SW-Cs with two connectors, one for each interface.

From	To	Connector name (filled in automatically)
`SwcSensor` / `PpSpeed`	`SwcActuator` / `RpSpeed`	`Assembly_SwcSensor_PpSpeed_SwcActuator_RpSpeed`
`SwcActuator` / `PpScale`	`SwcSensor` / `RpScale`	`Assembly_SwcActuator_PpScale_SwcSensor_RpScale`

1. Hold `Shift` and drag from the port `PpSpeed` to the port `RpSpeed`.

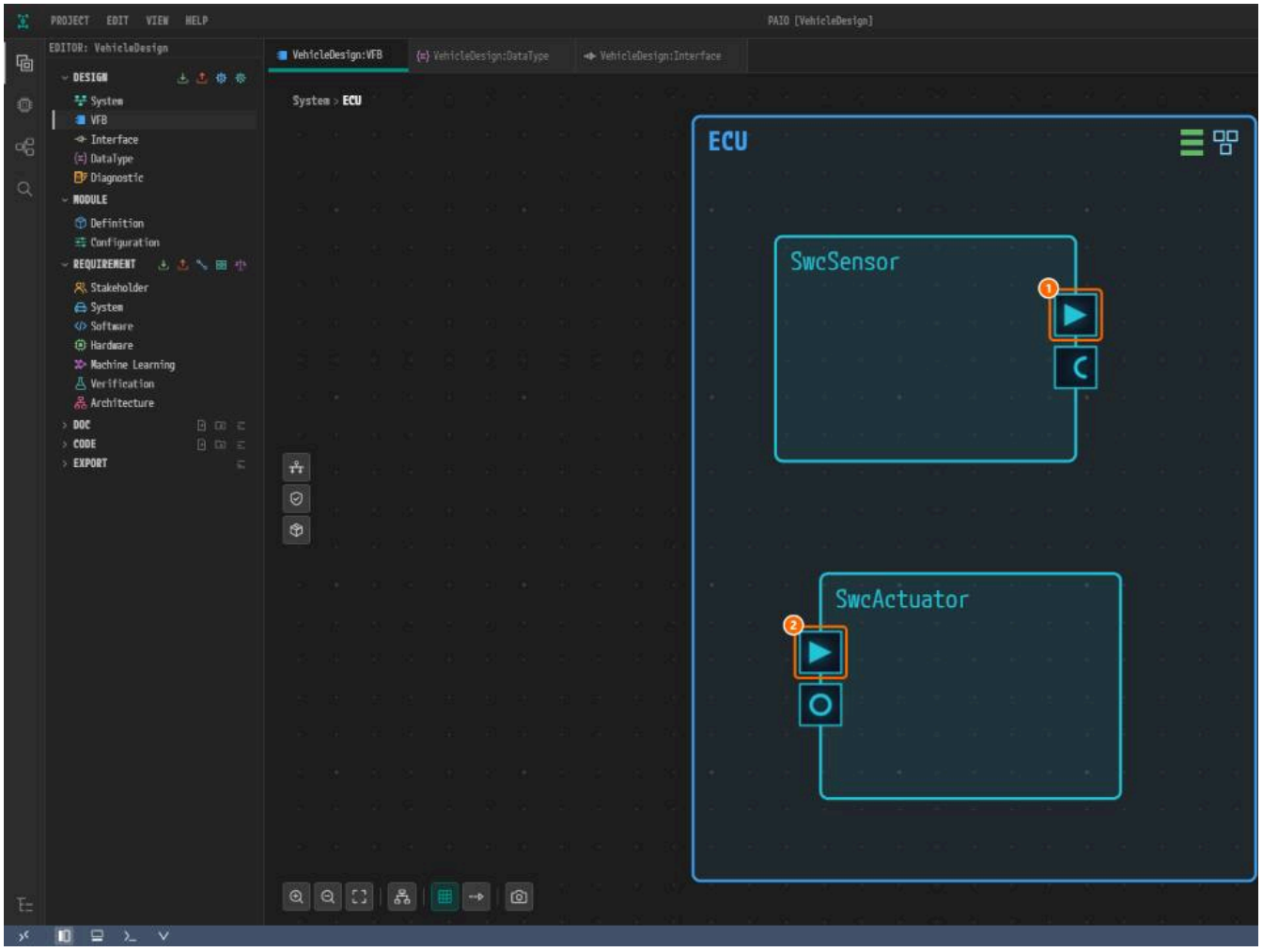


Figure 4-8. ① `PpSpeed`, ② `RpSpeed`.

2. Hold `Shift` and drag from the port `PpScale` to the port `RpScale`.

3. Click **Auto Layout** in the toolbar under the canvas and save with `Ctrl+S`.

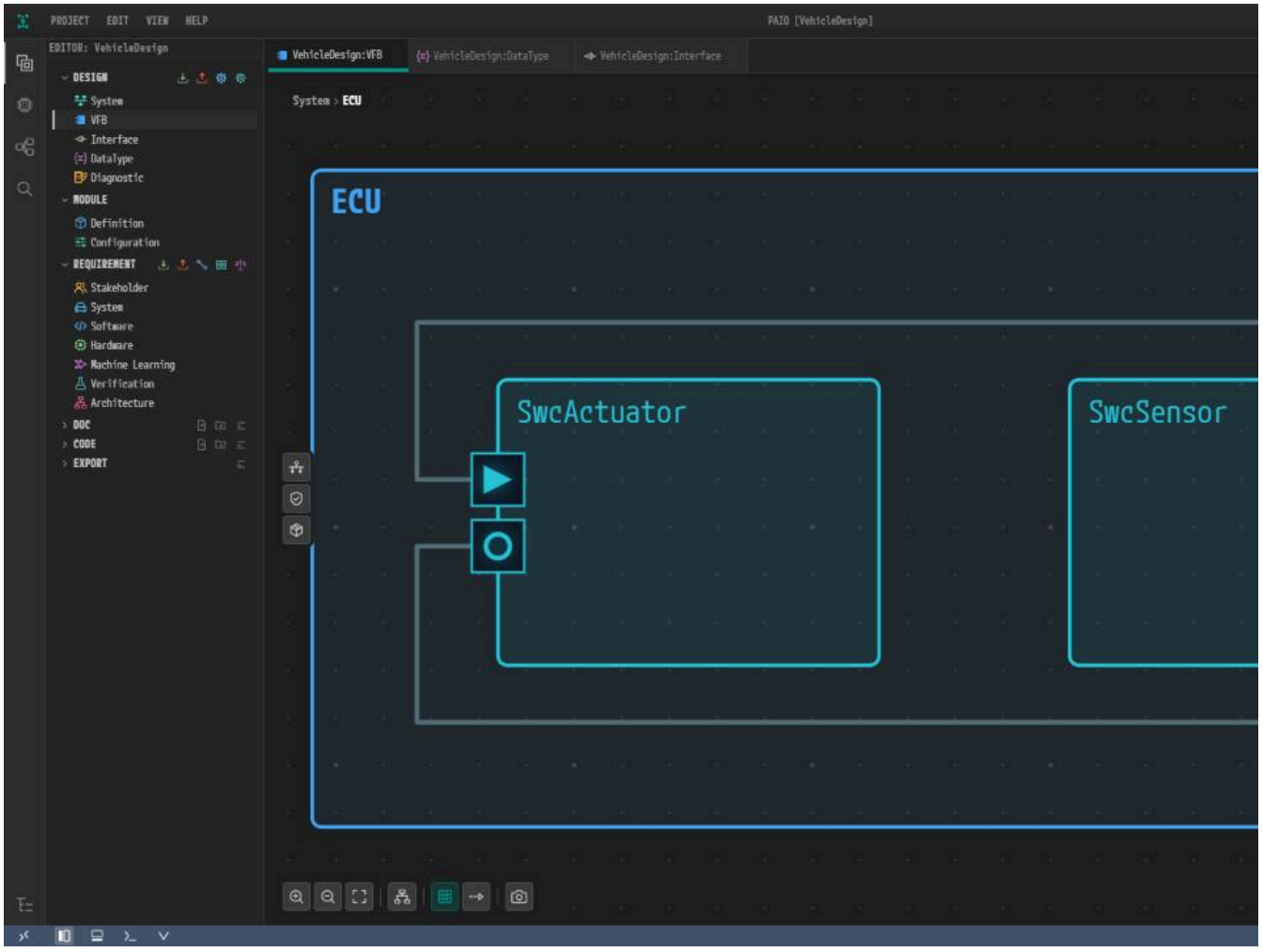


Figure 4-9. Both connectors. The positions may differ on your screen.

Why Is It Designed This Way?

- **The naming.** `Pp` marks a provided port and `Rp` a required port, so a name alone tells the direction.
- **NON-QUEUED.** The receiver wants the latest speed, not every speed.
- **Always from the provided port to the required port.** A connector joins a provider and a requirer of the same interface. That is why the second drag ru actuator is the server.

Checklist

- [ ] `SwcSensor` has the ports `PpSpeed` and `RpScale`.
- [ ] `SwcActuator` has the ports `RpSpeed` and `PpScale`.
- [ ] Selecting a port shows its **PORT INTERFACE** and the setting from the table.
- [ ] Two connectors join the SW-Cs.
- [ ] The tab `VehicleDesign:VFB` carries no modified mark.

Next Step

In Internal Behavior you add the runnables that use these ports.

Internal Behavior

What You Will Do

In this chapter you add the runnables of both SW-Cs. A runnable is a C function the RTE calls. An access point says which port it uses, and an RTE event s

Before You Begin

- Start after finishing SW-Cs and Ports, inside the composition `ECU` on the tab `VehicleDesign:VFB`.

SW-C	Runnable (NAME, SYMBOL)	Access point	RTE event	
`SwcSensor`	`Sensor_Cyclic`	**DataSendPoint** from `PpSpeed`, **SyncServerCallPoint** from `RpScale`	**TimingEvent** `SensorCyclicTiming`	**PE**
`SwcActuator`	`Actuator_Init`		**InitEvent** `ActuatorInitEvent`	
`SwcActuator`	`Actuator_OnData`	**DataReceivedPoint** from `RpSpeed`	**DataReceivedEvent** `ActuatorOnDataEvent`	**R-I**
`SwcActuator`	`Actuator_GetScale`		**OperationInvokedEvent** `ActuatorGetScaleEvent`	**P-I**

`Actuator_GetScale` also gets **CAN BE INVOKED** = **TRUE**.

SwcSensor

1. Double-click `SwcSensor`. The **SWC Internal Behavior [SwcSensor]** window opens.
2. Right-click the SW-C body and click **Add Runnable**. Click **Auto Layout** in the window's toolbar if the runnable overlaps a port.

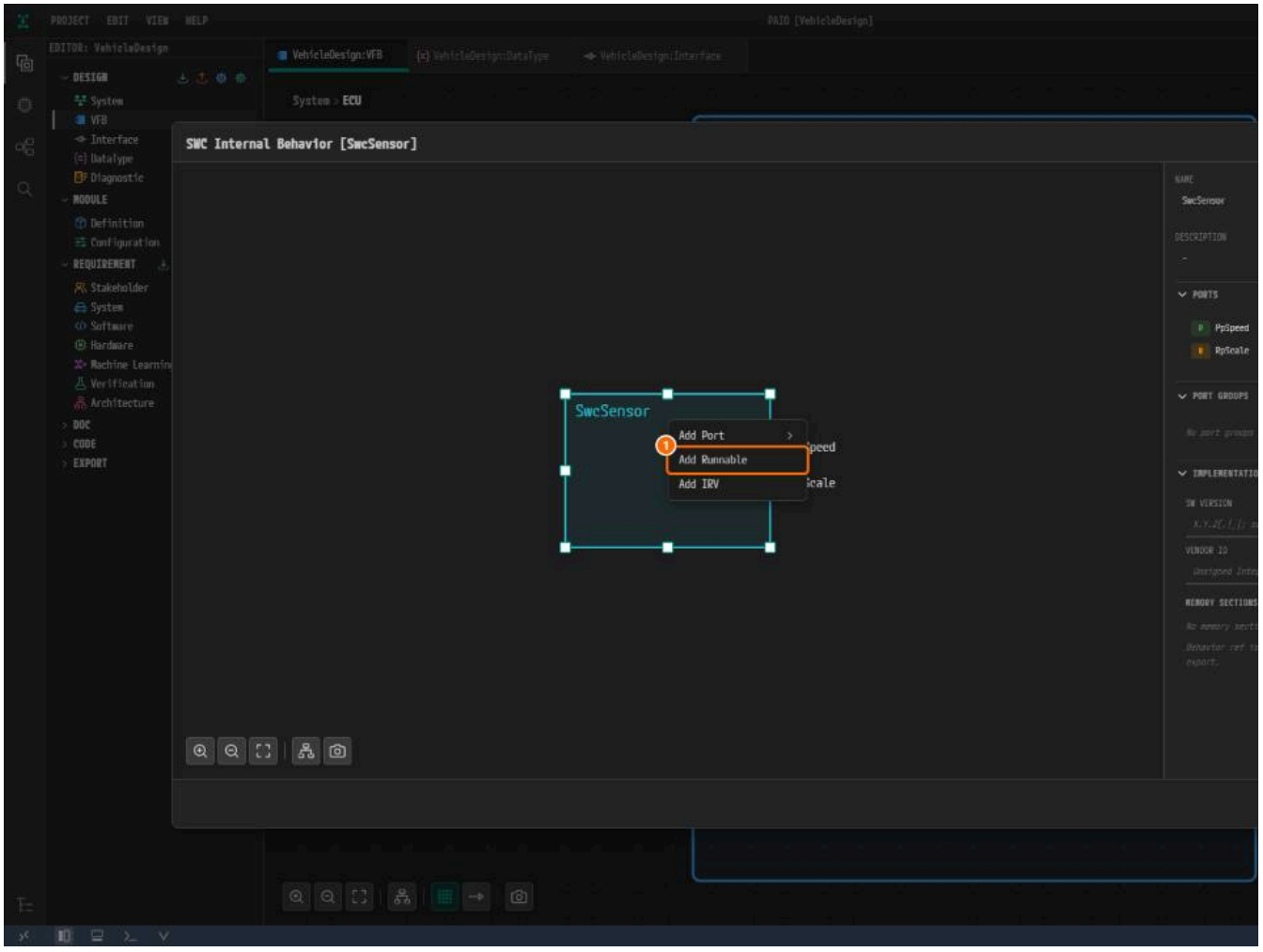


*Figure 5-1. ① **Add Runnable**.*

3. Click the runnable, then enter `Sensor_Cyclic` in **NAME** and **SYMBOL**.

4. Hold `Shift` and drag from the port `PpSpeed` to the runnable. Click **DataSendPoint** in the menu that opens.

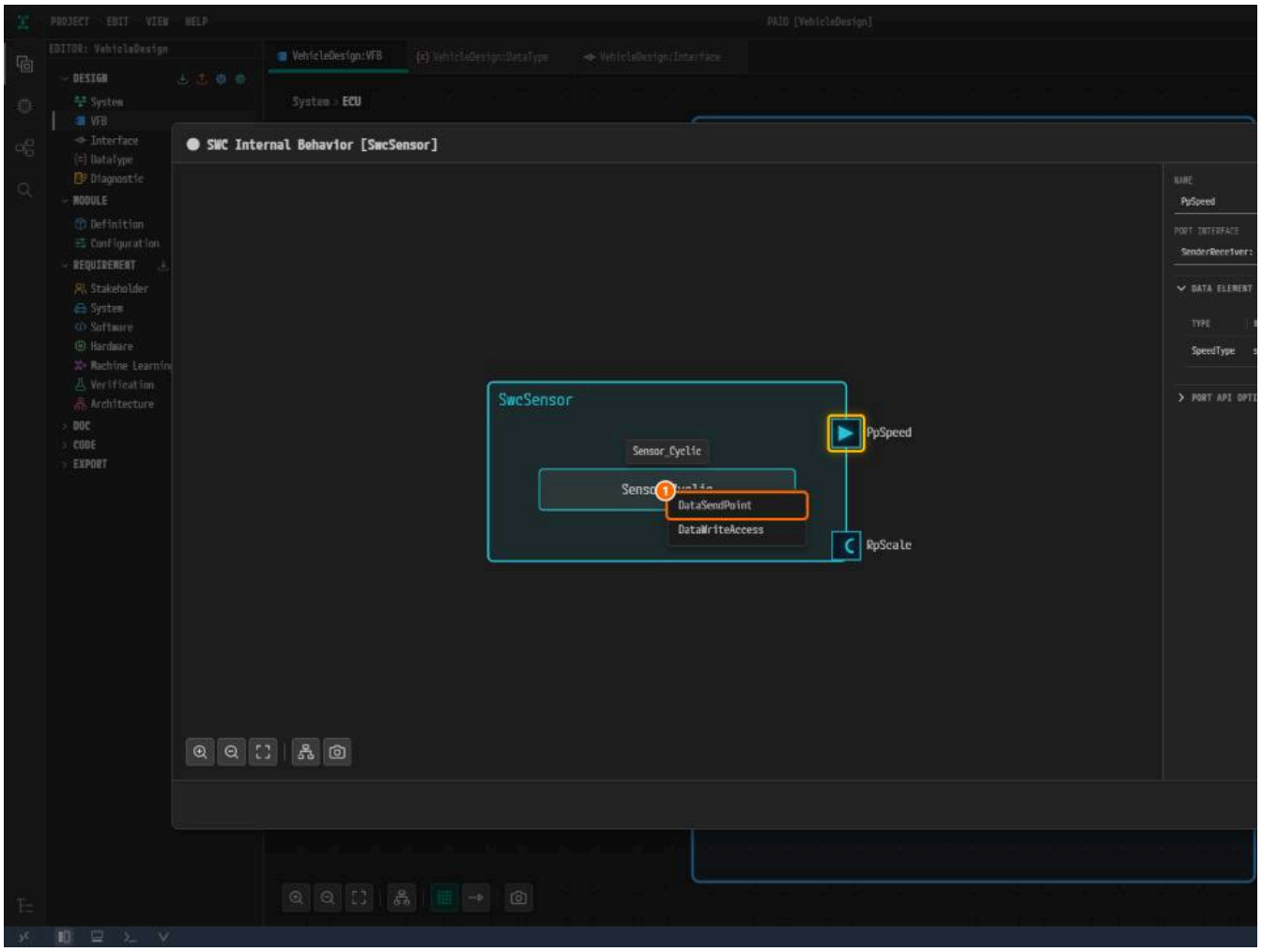


*Figure 5-2. ① **DataSendPoint**.*

5. Hold `Shift` and drag from the port `RpScale` to the runnable, and click **SyncServerCallPoint**.

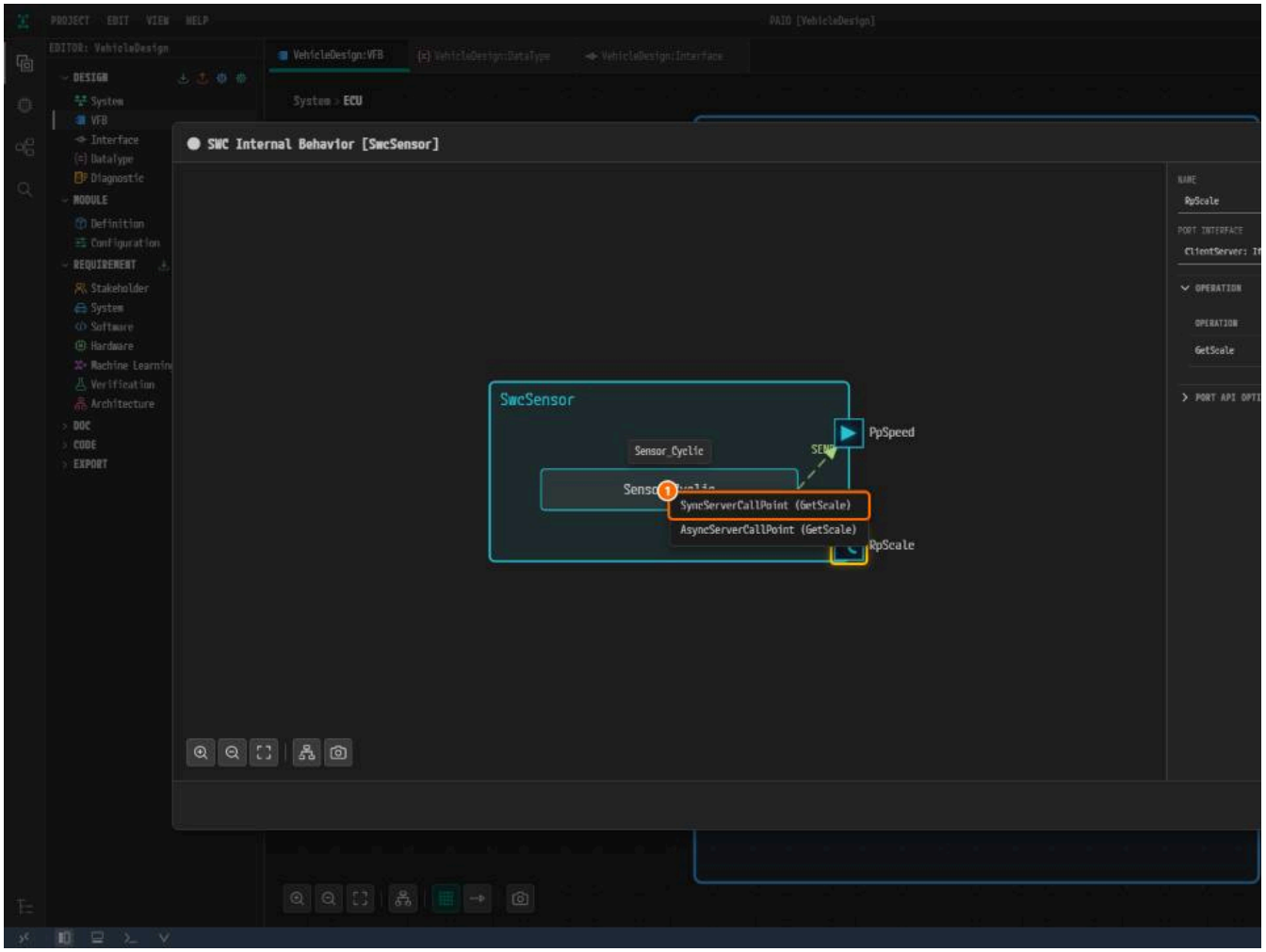


*Figure 5-3. ① **SyncServerCallPoint**.*

ACCESS POINTS now lists `Rte_Write_PpSpeed_speed` and `Rte_Call_RpScale_GetScale`. These are the RTE functions the runnable can call. You writ

6. Click the runnable again, click **+** on the **RTE EVENTS** header, and click **TimingEvent**.

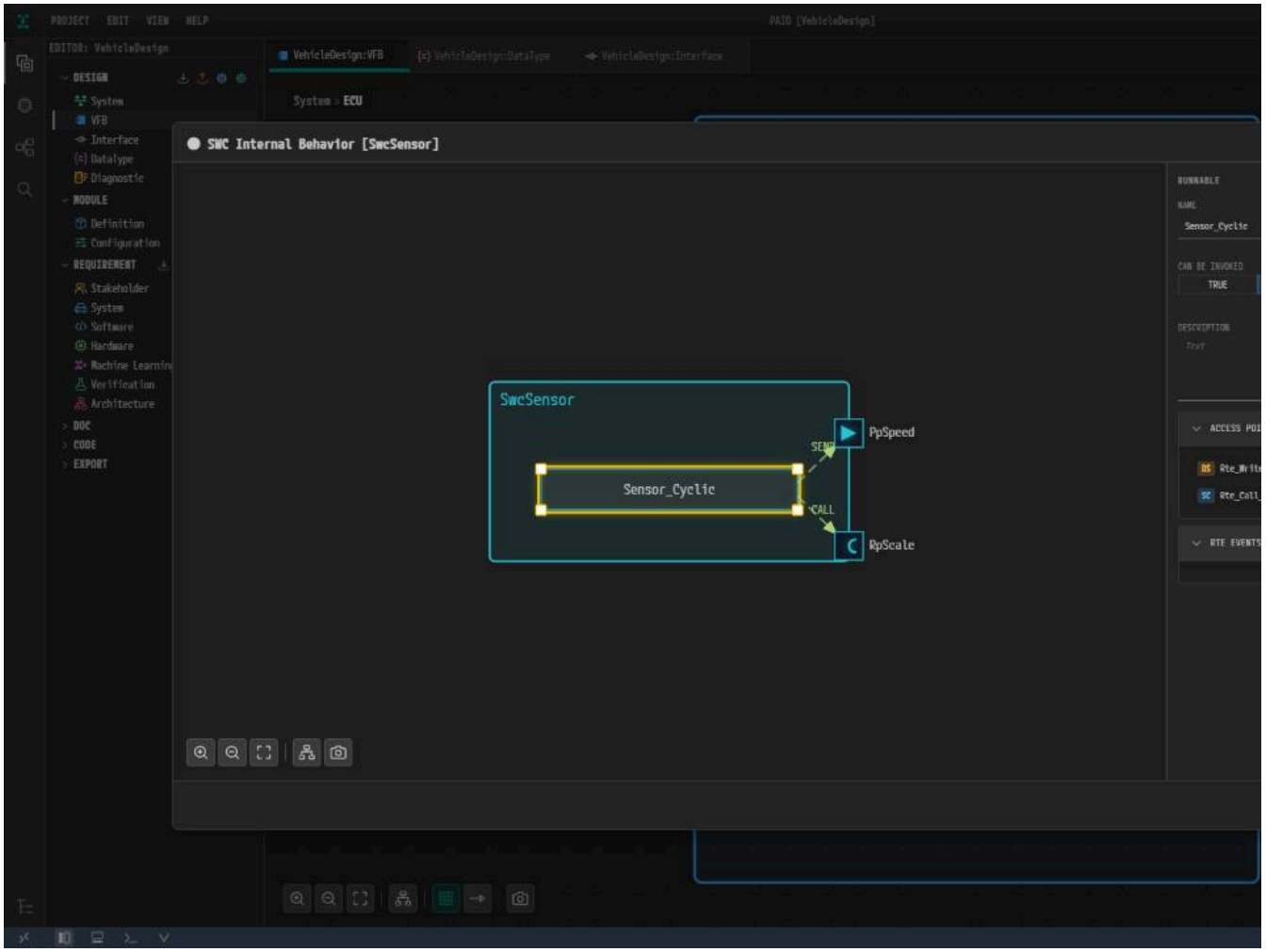


*Figure 5-4. ① **+** (Add RTE Event), ② **TimingEvent**.*

7. Enter `SensorCyclicTiming` in **NAME** and `0.02` in **PERIOD**, leaving the unit at `s`.

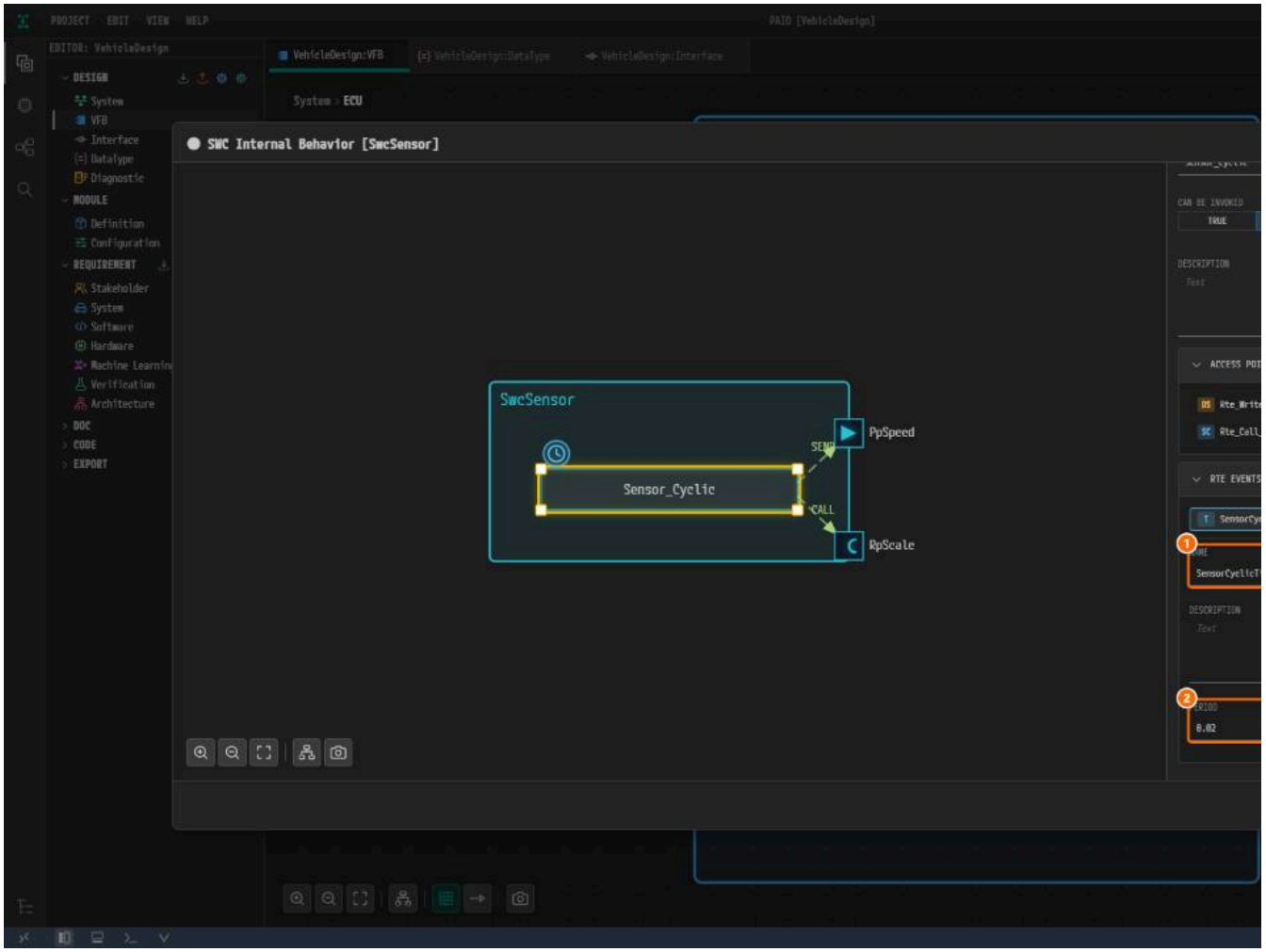


*Figure 5-5. ① **NAME**, ② **PERIOD**.*

8. Click **Auto Layout**, and click an empty spot of the canvas. Save with `Ctrl+S` and click **Close**.

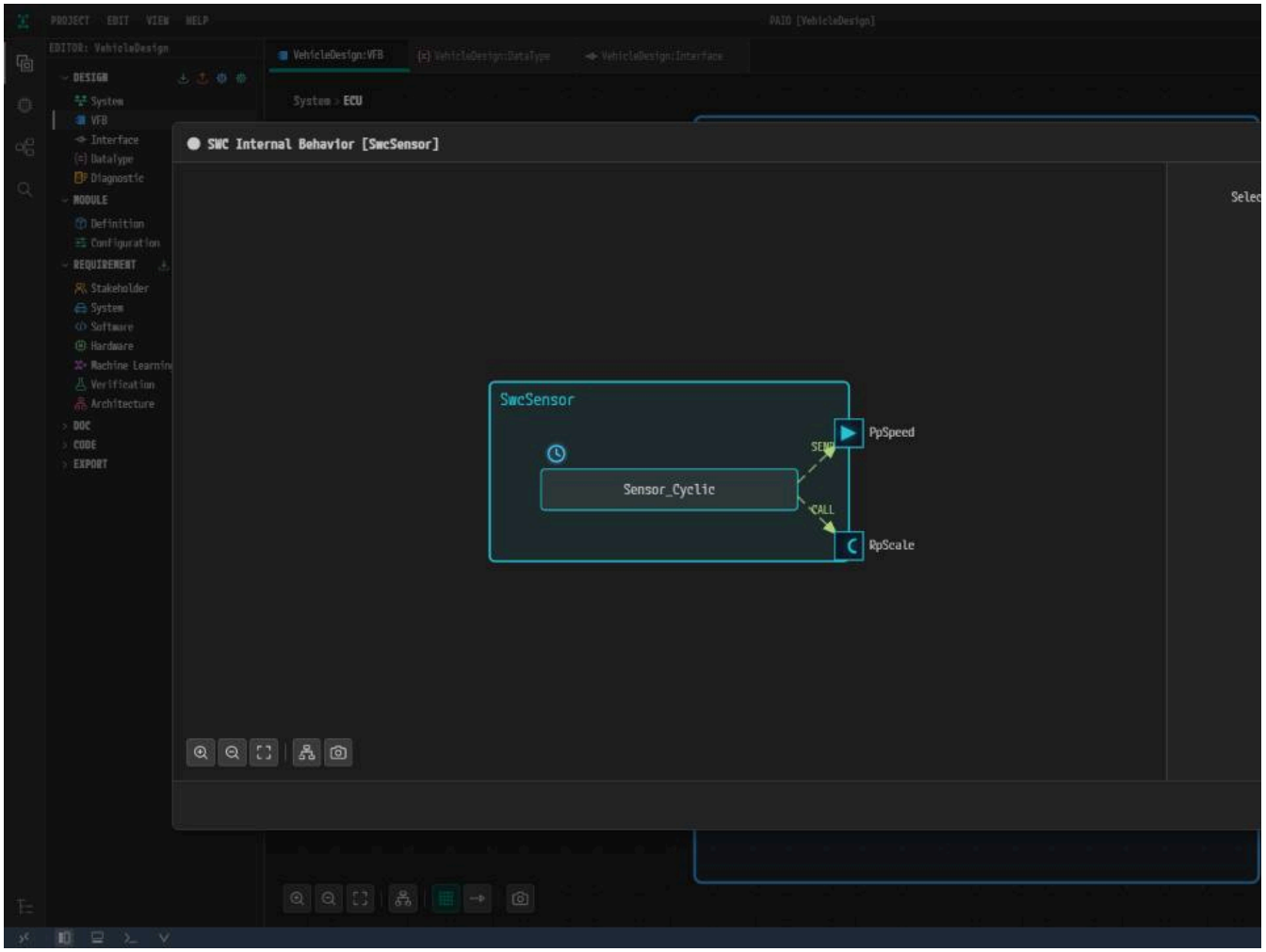


Figure 5-6. `SwcSensor` with its runnable and access points.

SwcActuator

Double-click `SwcActuator` and add its three runnables from the table in the same way.

1. For `Actuator_OnData`, the **DataReceivedEvent** asks for a port and a data element. Pick them first, then enter the event's **NAME**.

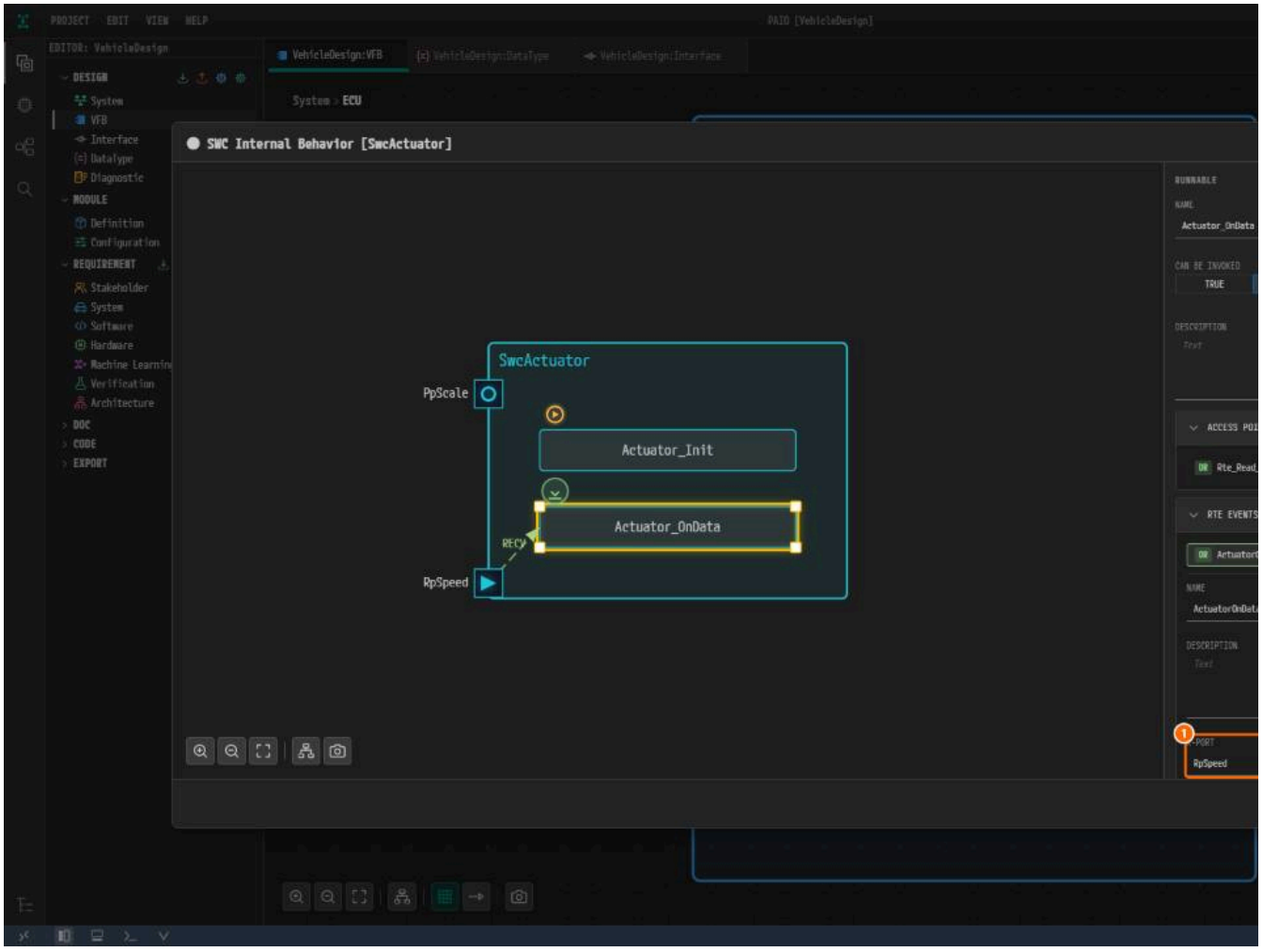


*Figure 5-7. ① **R-PORT**, ② **DATA ELEMENT**.*

2. For `Actuator_GetScale`, click **TRUE** under **CAN BE INVOKED**.

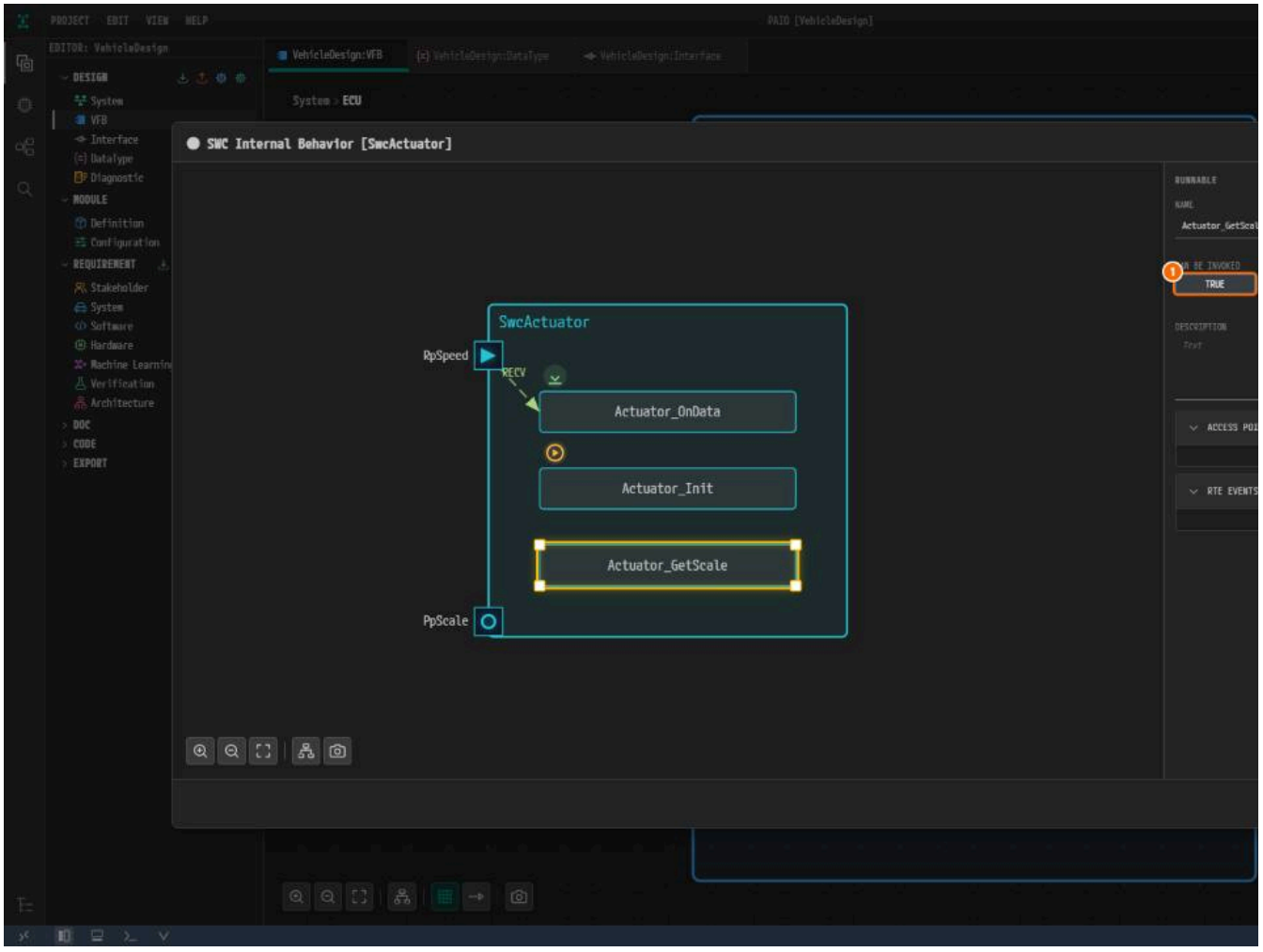


*Figure 5-8. ① **TRUE**.*

3. The **OperationInvokedEvent** asks for a port and an operation.

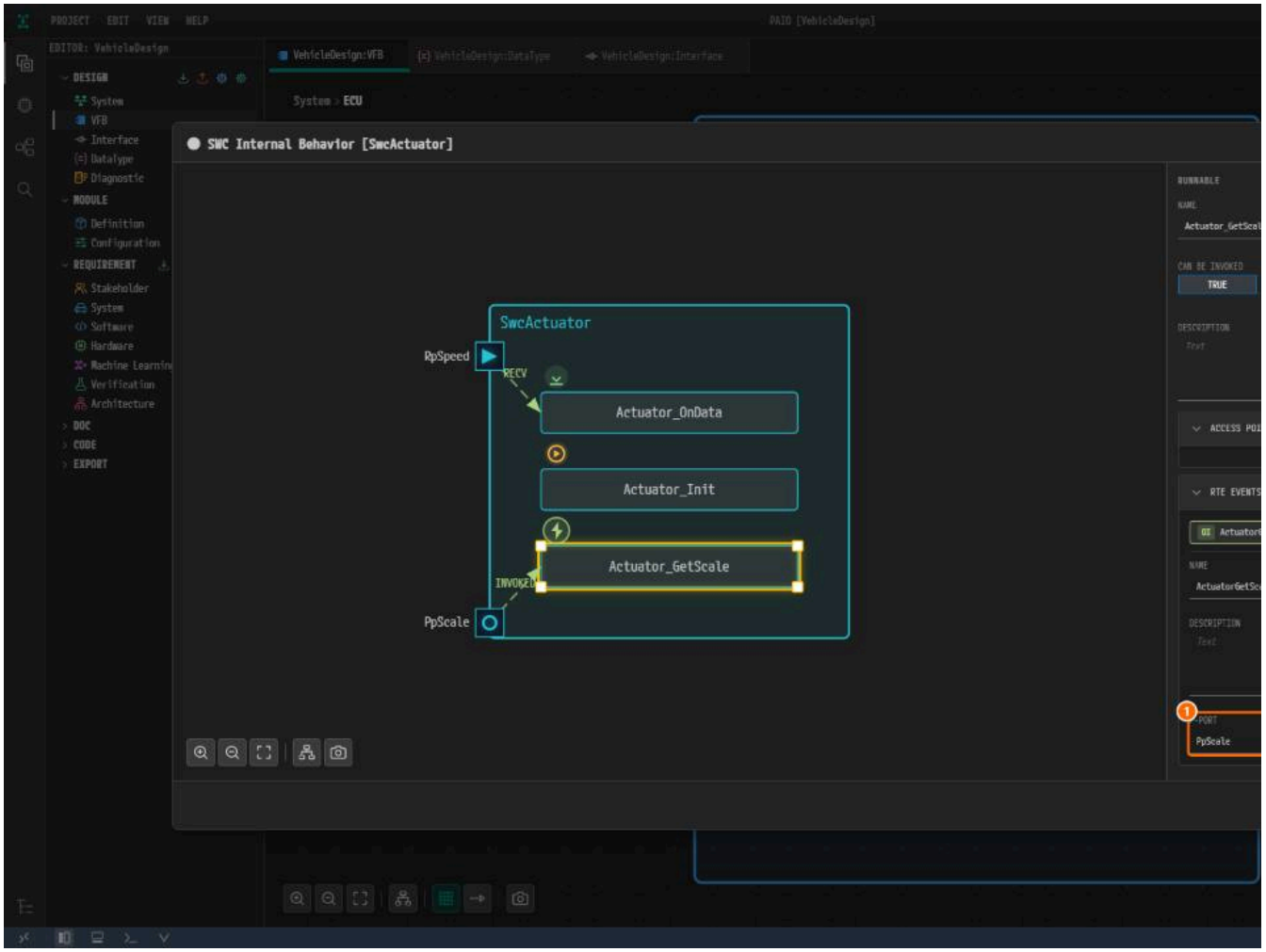


*Figure 5-9. ① **P-PORT**, ② **OPERATION**.*

4. Click **Auto Layout**, click an empty spot of the canvas, save with `Ctrl+S` and click **Close**. Save the tab with `Ctrl+S` as well.

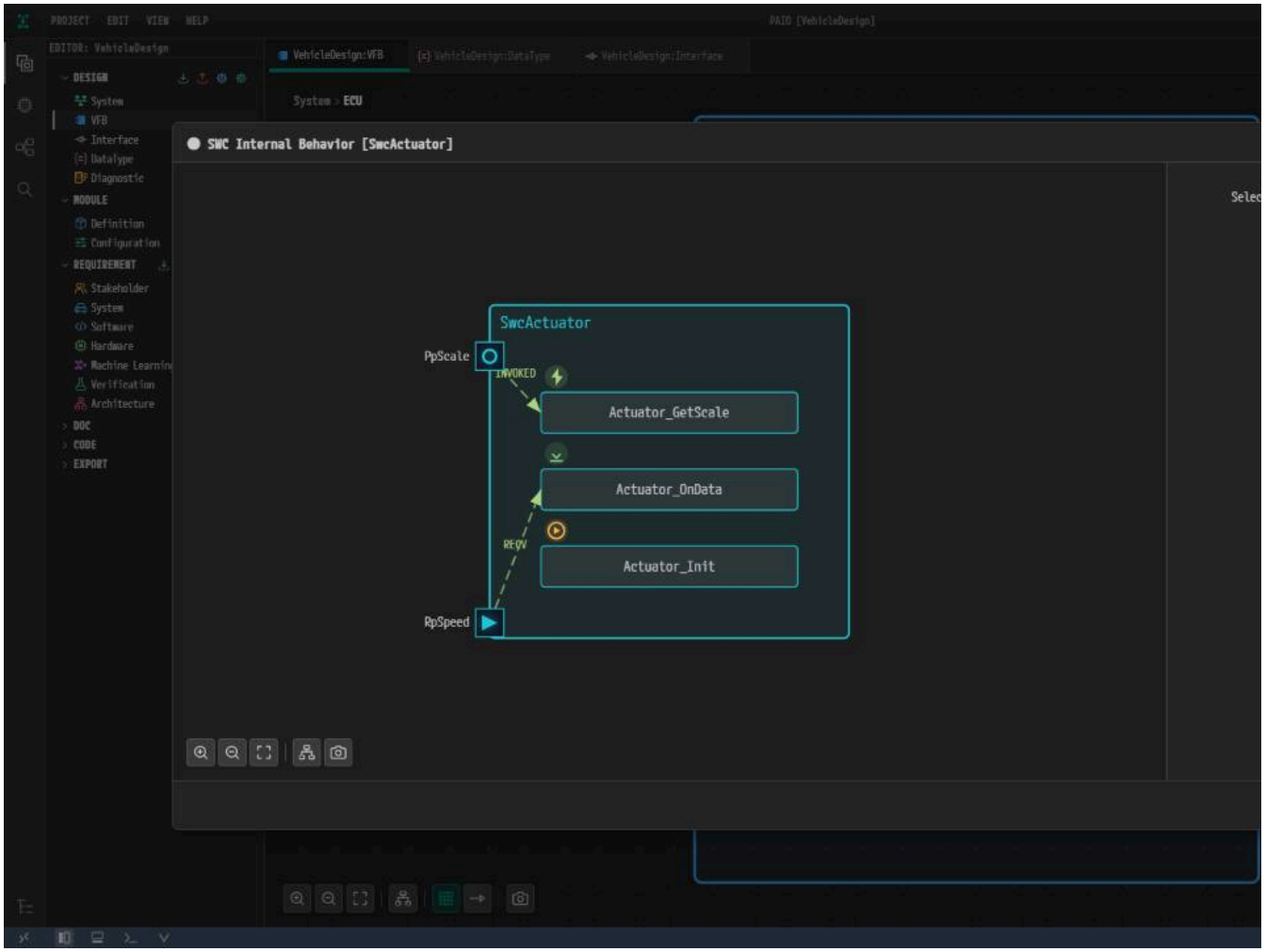


Figure 5-10. `SwcActuator` with its three runnables.

> **Caution.** Do not leave out **TRUE** on `Actuator_GetScale`. Chapter 8 leaves its event unmapped to any task, and the code generator accepts that only f concurrently. Otherwise generation stops with `RTE_CS_SERVER_UNMAPPED_NOT_CONCURRENT`.

Why Is It Designed This Way?

- `Sensor_Cyclic` **runs on a timer.** The RTE starts it every 20 ms, and the access points allow it to write the speed and ask for the scale. The calls then
- `Actuator_OnData` **runs on arrival.** It does not poll: the event starts it when `speed` arrives.
- `Actuator_Init` **runs once at start.** It is the place for one-time setup.
- `Actuator_GetScale` **runs when called.** It runs directly inside the caller's call, without a task of its own.

Checklist

- [ ] `SwcSensor` has the runnable `Sensor_Cyclic` with two access points and the event `SensorCyclicTiming`.
- [ ] `SwcActuator` has the runnables `Actuator_Init`, `Actuator_OnData` and `Actuator_GetScale`, each with its event.
- [ ] `Actuator_GetScale` shows **TRUE** under **CAN BE INVOKED**.

Next Step

In PRTE Setup you bring in the BSW modules and set the start-up modules `EcuM`, `BswM`, `Mcu` and `Det`.

PRTE Setup

What You Will Do

In this chapter you prepare the ECU's BSW configuration (ECUC) in three steps.

1. In the VFB **ECU Configuration** window, bring in the BSW modules with **PRTE Platform Import**, the network management modules `CanNm` and `Nm` includ empty configuration per module.
2. In the **ECU Configuration** editor, set the start-up and hardware values of `EcuM`, `BswM`, `Mcu` and `Det` by hand. The design does not decide them; they
3. Import the Rte values file. It carries the six `Rte` references the property panel does not offer yet.

> **Caution.** This tutorial does not use **PRTE AUTO CONFIGURATION**. You configure the BSW by hand in the ECU Configuration editor, module by module, so t click **PRTE AUTO CONFIGURATION**, the **PRTE Auto Configuration** icon of the ECU Configuration editor or **Auto-Generate**, and do not apply the **ECU Configu** design and would overwrite what you enter.

Before You Begin

- Start after finishing Internal Behavior.
- Save the file VehicleDesign_Rte.arxml to your computer.

> **Caution.** The file holds values for an ECU named `ECU`, the node `ECU` of the AUTOSAR System Tutorial. Do not rename the node.

Where	Item	Value
PRTE Platform Import window	Module map	The suggested selection, plus `CanNm` and `Nm`
ECU Configuration [ECU] window		Close with `Esc`
ECU Configuration editor	`EcuM`, `BswM`, `Mcu`, `Det`	The tables below
Conflict Resolution window	**Module Definition**	**All → Use Imported**

Importing the BSW Modules

1. In the tab `VehicleDesign:VFB`, go inside the composition `ECU`. Click the ECU icon at the top right of the frame to open the **ECU Configuration [ECU]** right. Do not click **PRTE AUTO CONFIGURATION** next to it.

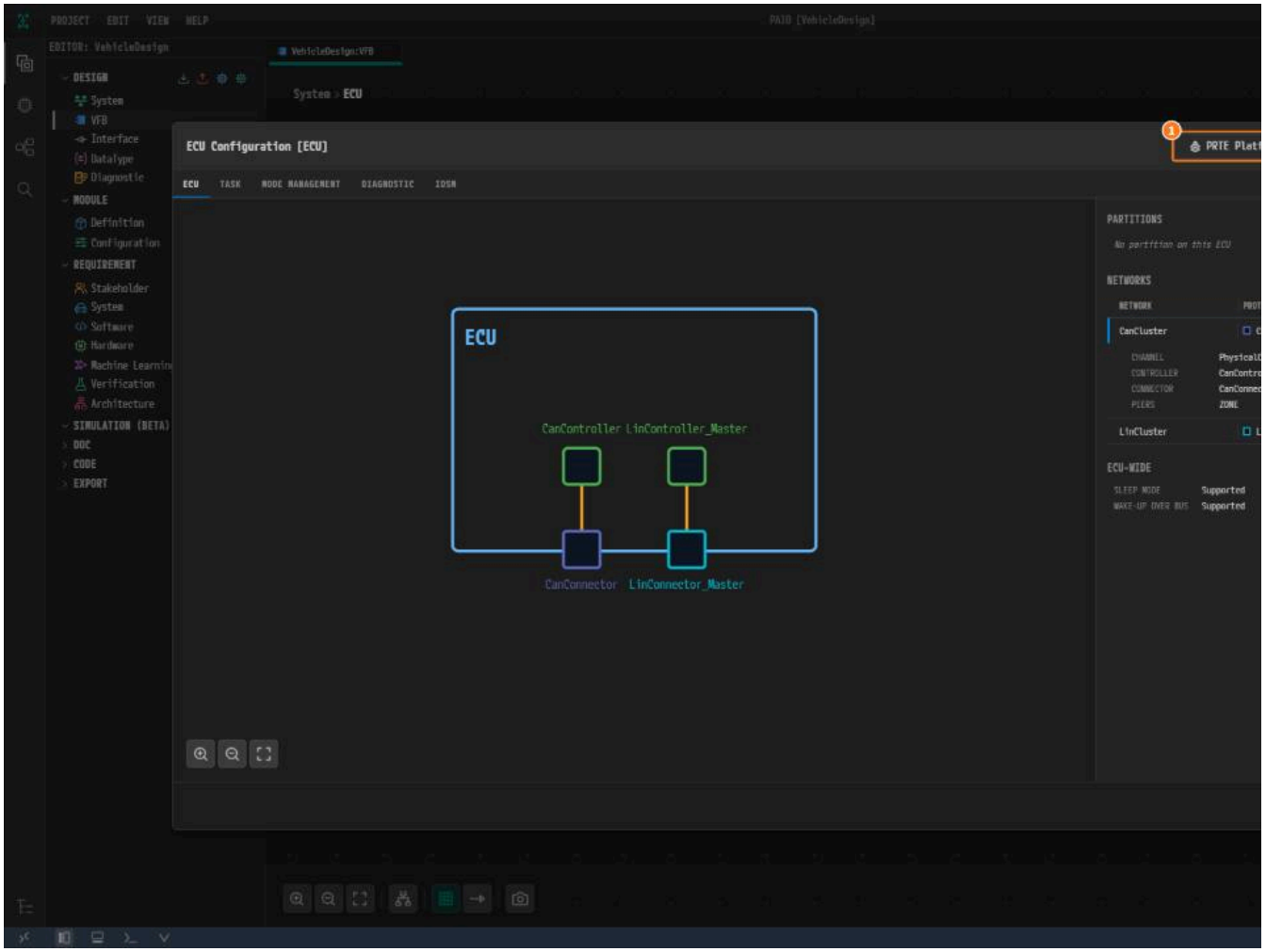


*Figure 6-1. ① **PRTE Platform Import.***

2. The **PRTE Platform Import** window shows a map of the BSW modules. The bright ones are selected already, from the design. The CAN stack is selecte `PduR`. The network management modules `CanNm` and `Nm` are not. Click `CanNm`: `Nm` is selected with it, and the line at the bottom reads `Selected Can` `Nm` is the module AUTOSAR calls the Network Management Interface (NmIf); PRTE names it `Nm`.

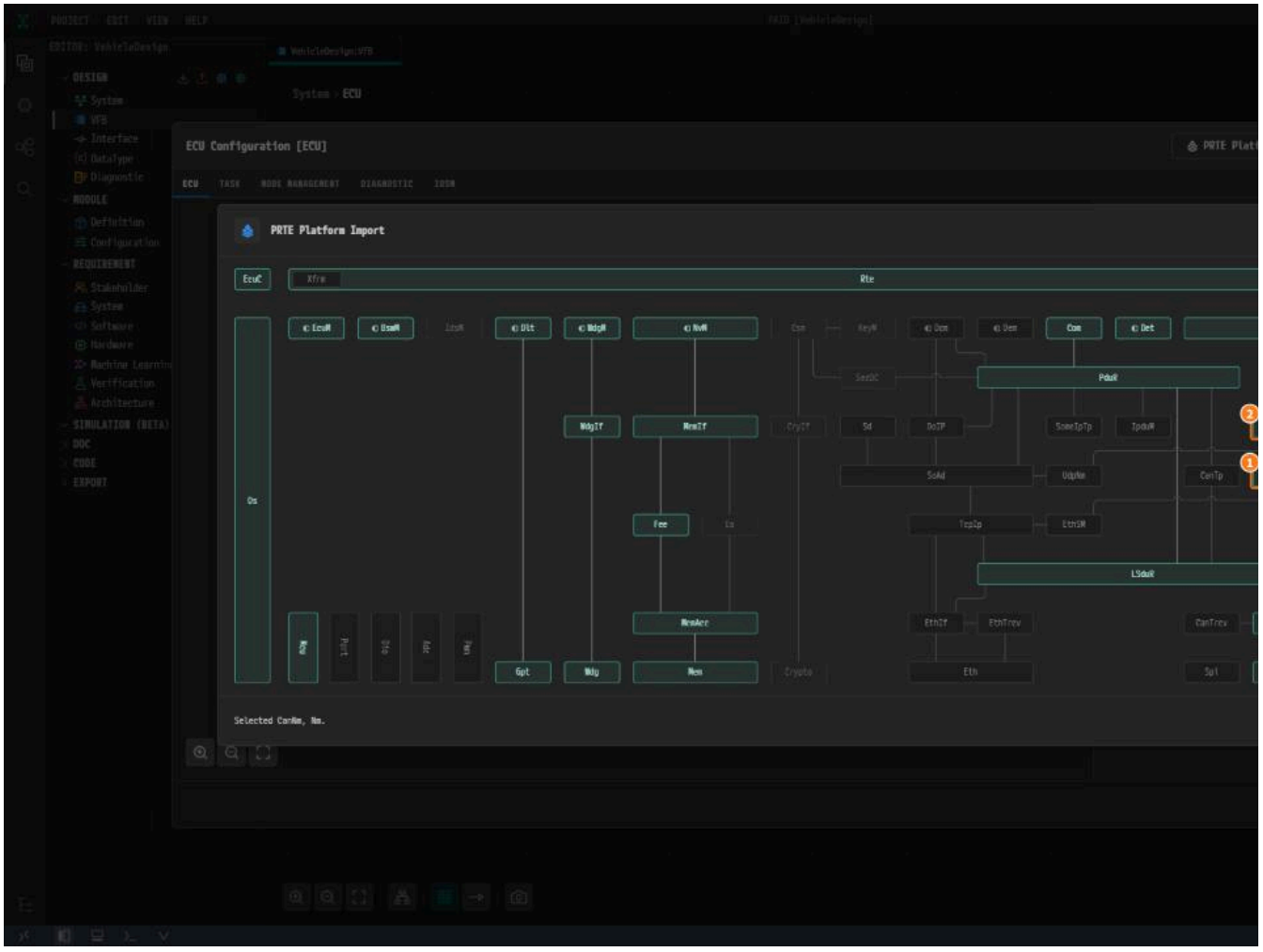


*Figure 6-2. ① `CanNm`, ② `Nm`, ③ **Import**.*

Wait until the progress reaches `Complete`.

3. Press `Esc` to close the **ECU Configuration** window. Do not press `Ctrl+S` in this window: there it applies the window. Closing with `Esc` writes nothing

> **Note.** The import also adds the composition `PRTE_ECU` with the BSW service components to the frame, and the interfaces and data types of the BSW editors now list many entries you did not create. Selecting `CanNm` and `Nm` adds only their definitions and empty configurations, no service componen

Opening the ECU Configuration Editor

Click **Configuration** under **MODULE**. The ECU Configuration editor opens. The **ECU** list on the left shows the ECUs. The imported modules are under `ECU`. Th holds the SW-C pairs of the import.

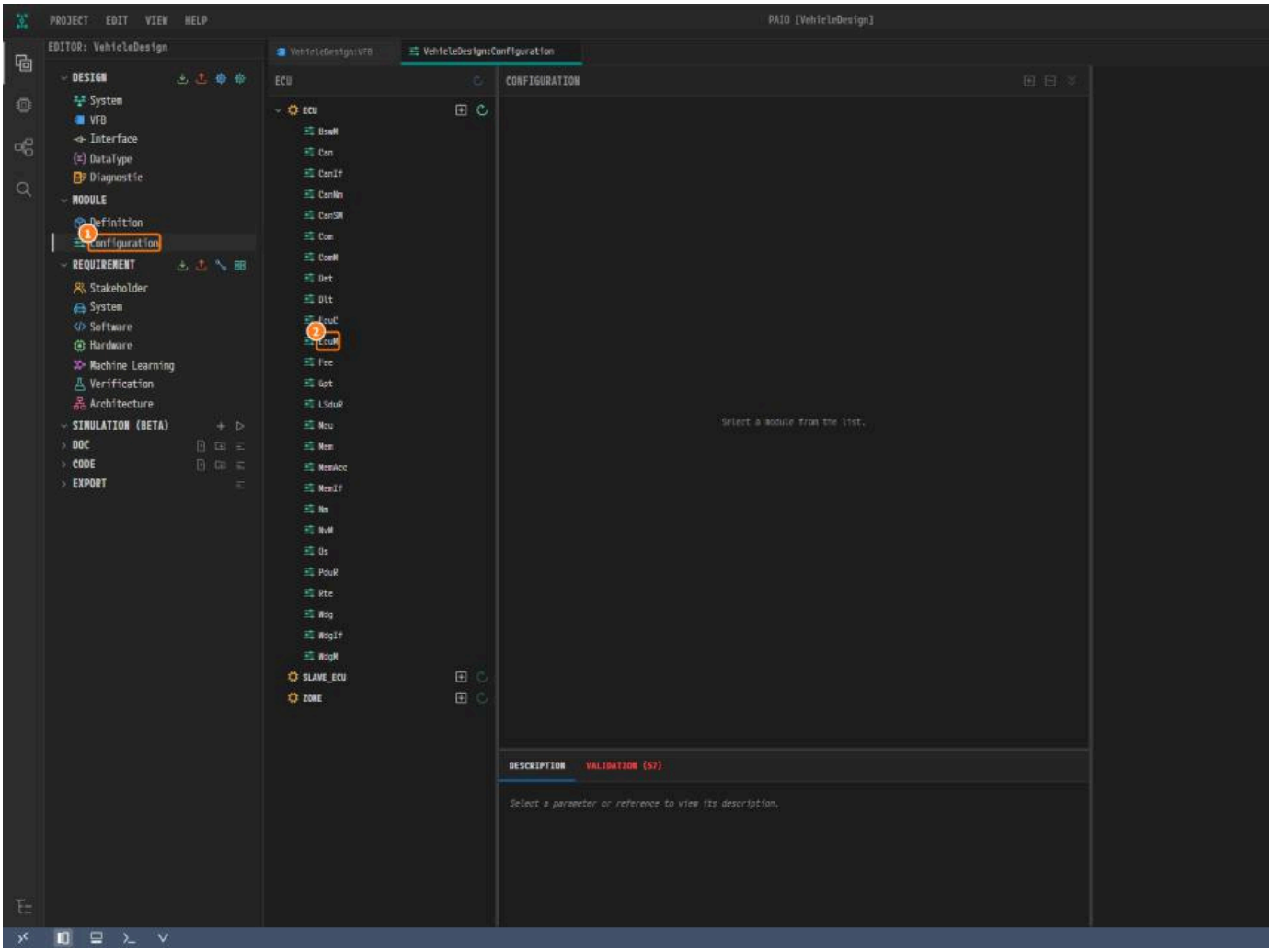


*Figure 6-3. ① **Configuration**, ② `EcuM`, one of the modules of `ECU`.*

At the right end of each ECU row are two icons, **Add Module** and **PRTE Auto Configuration** (the circular arrow). Do not click **PRTE Auto Configuration**.

The editor works the same way in every module of this chapter and of the chapters after it:

- **+** at the right end of a row in the **CONFIGURATION** tree adds a container of that definition, with its default values, and selects it. Change only the valu pick a reference from its list.
- A bold name in the **Container** column is not the name **+** gives. Type it in **Name** in the panel and press `Tab`.
- A reference whose target does not exist yet is not in its list. Type the path shown in the table and press `Tab`. Until the target exists the path is shov also adds `(unresolved)` to it.

Setting Up EcuM

1. Click `EcuM` in the list. The **CONFIGURATION** tree lists the container definitions of `EcuM`, all empty. Click **+** on the `EcuMGeneral` row. The editor adds `Ec`

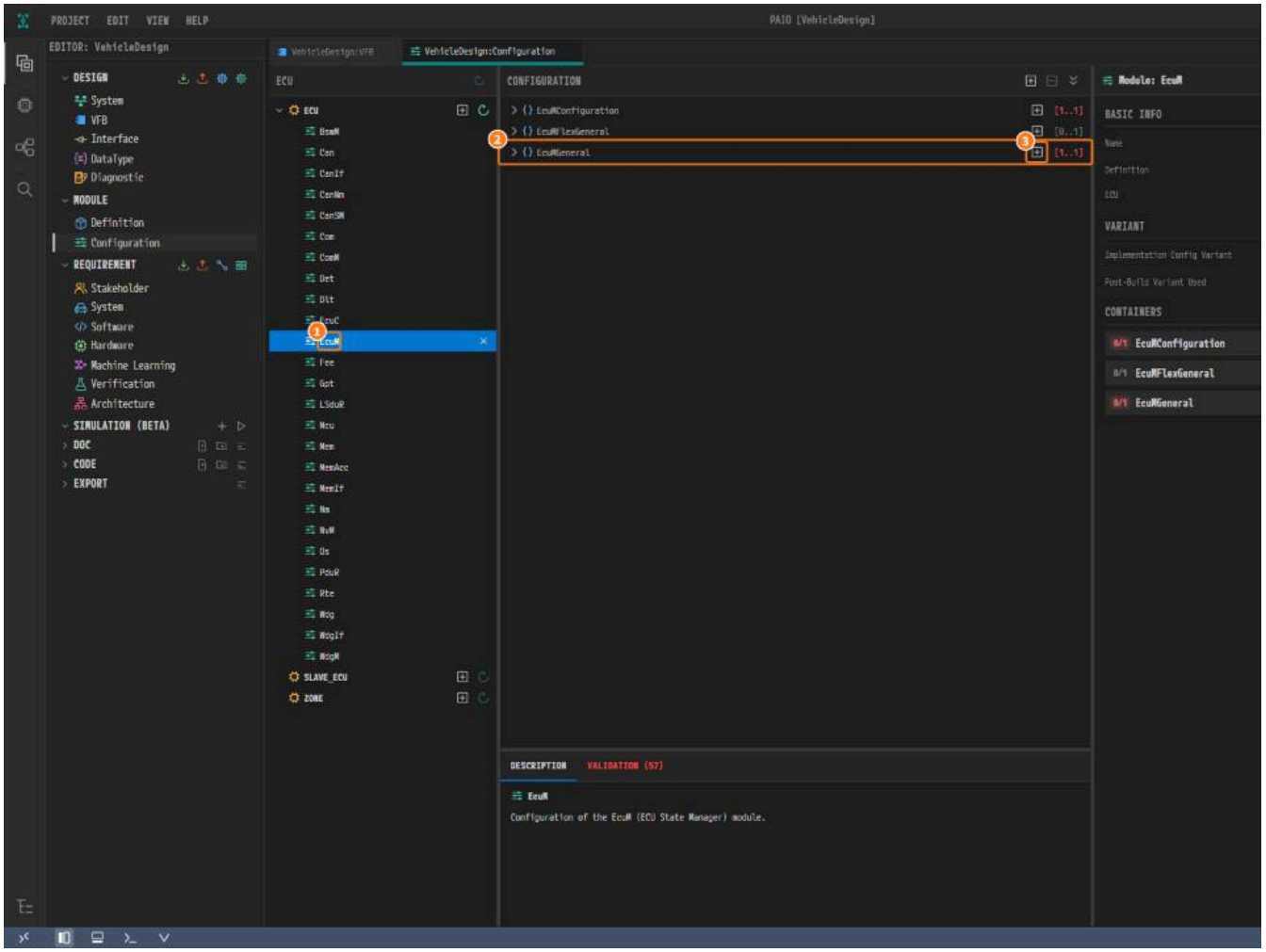


Figure 6-4. ① `EcuM`, ② the `EcuMGeneral` row, ③ +.

2. In the panel on the right, click **TRUE** for **EcuMDevErrorDetect** and type `0.01` in **EcuMMainFunctionPeriod**. Leave **EcuMVersionInfoApi** at **FALSE**.

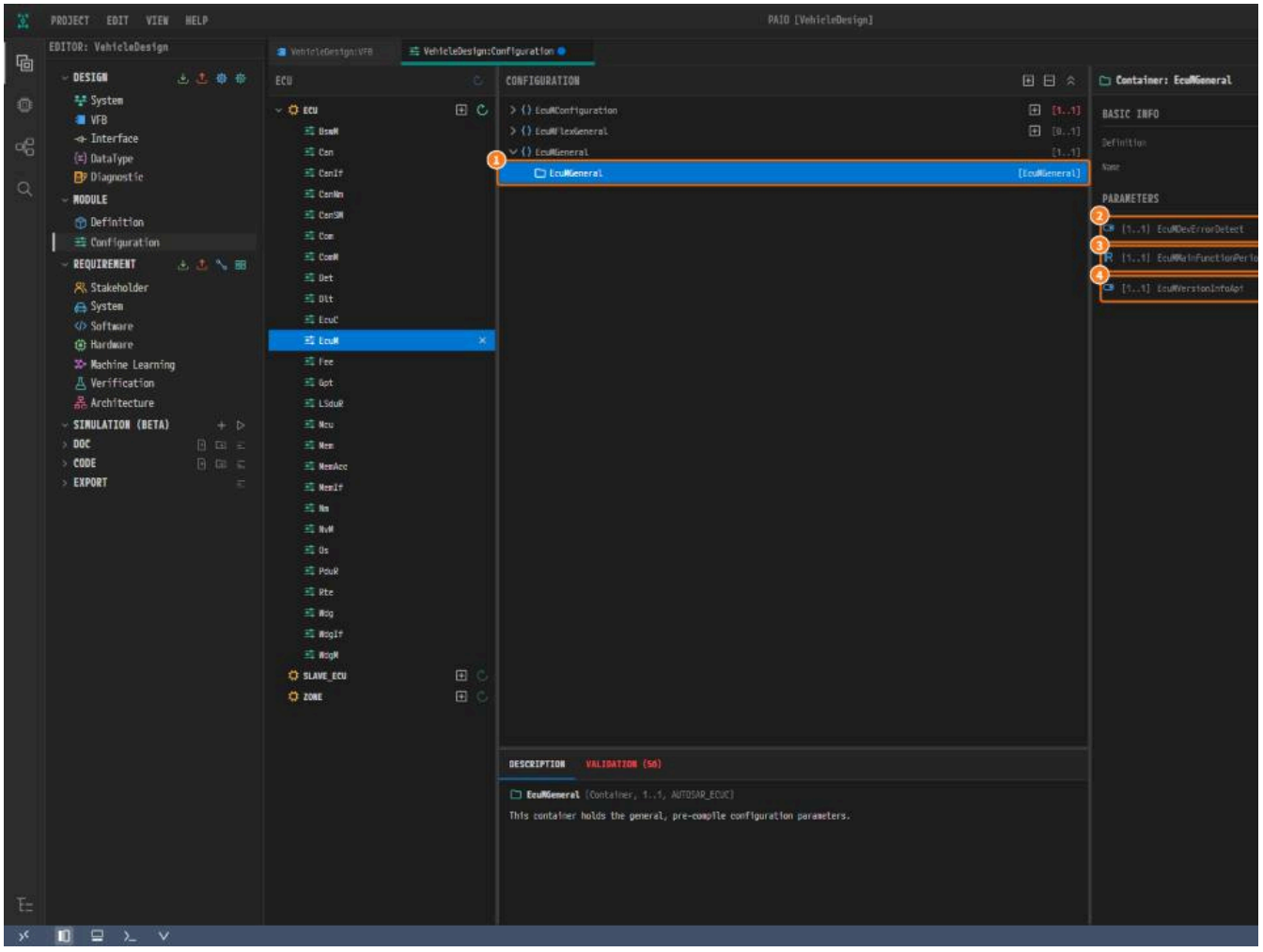


*Figure 6-5. ① `EcuMGeneral`, ② **EcuMDevErrorDetect**, ③ **EcuMMainFunctionPeriod**, ④ **EcuMVersionInfoApi**.*

3. Add `EcuMFlexGeneral`, `EcuMConfiguration` and, under it, `EcuMCommonConfiguration` the same way, each with its values from the table. In `EcuMCo` in **EcuMDefaultAppMode** and `Os/RES_AUTOSAR_ECUM` in **EcuMOSResource**. You create both in chapter 7, so both stay yellow until then, and `Os/RES_A`

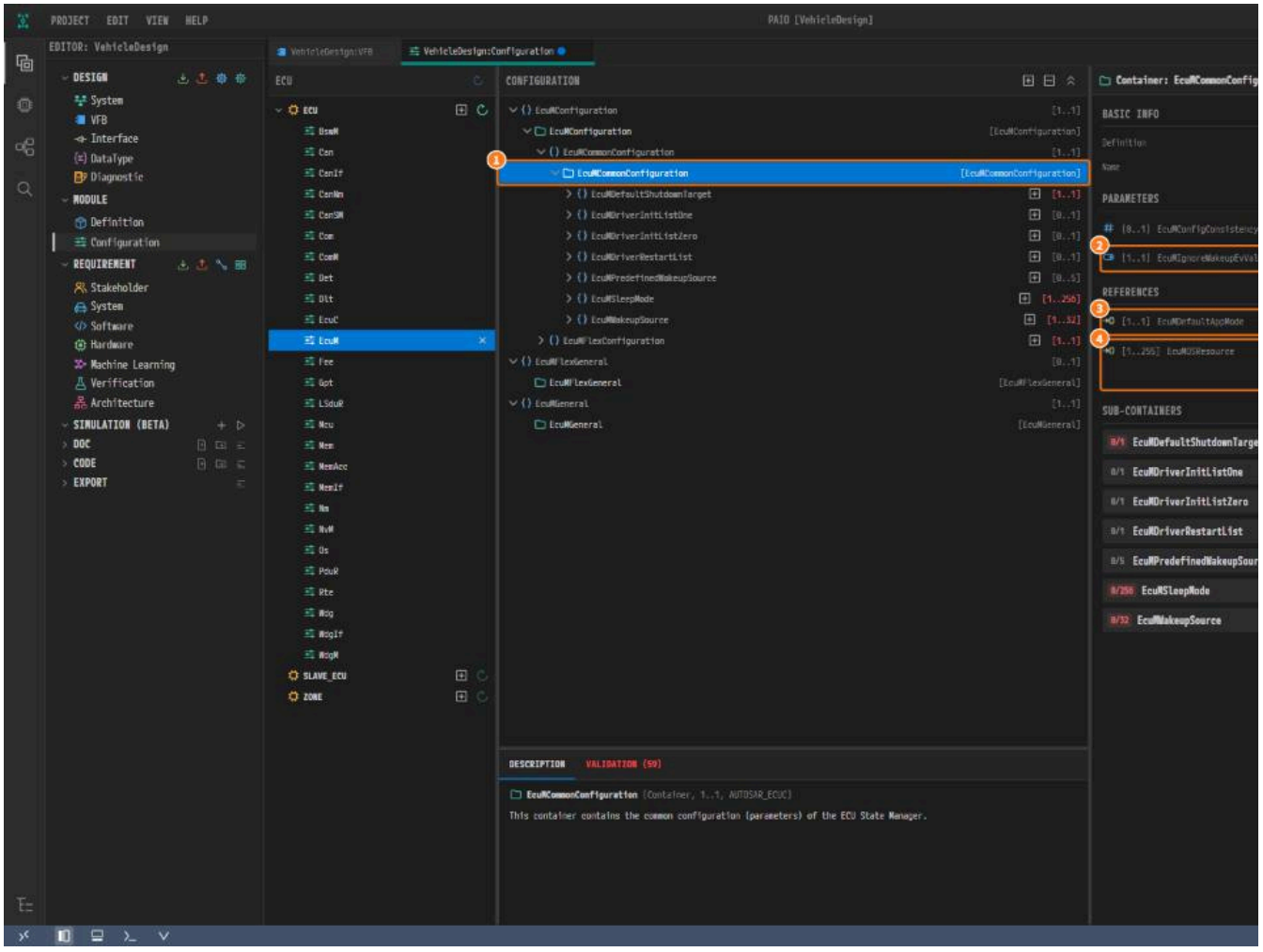


*Figure 6-6. ① `EcuMCommonConfiguration`, ② **EcuMIgnoreWakeupEvValOffPreOS**, ③ **EcuMDefaultAppMode**, ④ **EcuMOSResource**.*

4. Under `EcuMCommonConfiguration`, add `EcuMDriverInitListOne`. Under it, click **+** on the `EcuMDriverInitItem` row.

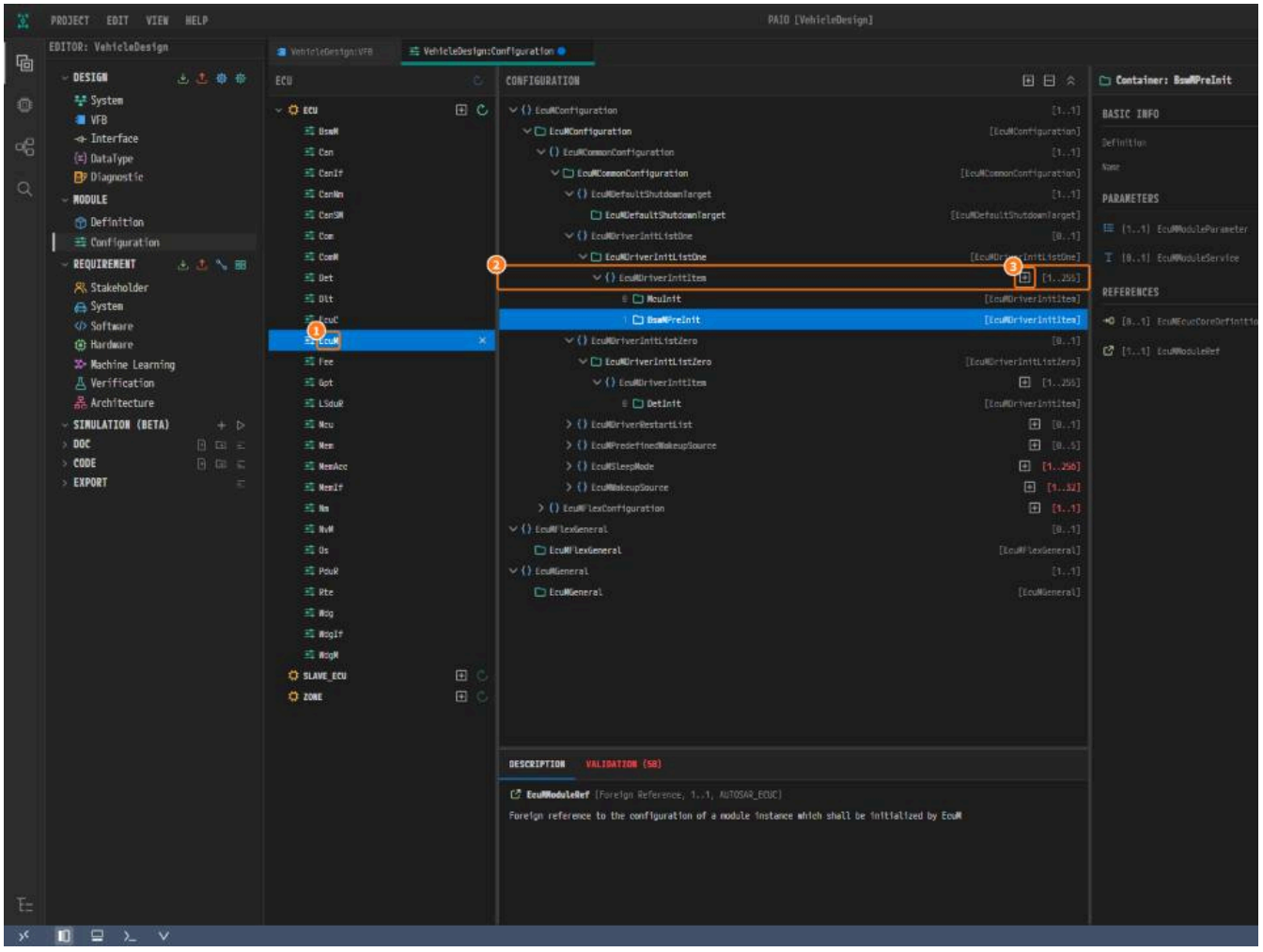


*Figure 6-7. ① `EcuM`, ② the `EcuMDriverInitItem` row under `EcuMDriverInitListOne`, ③ **+**.*

5. The editor adds `EcuMDriverInitItem_0`. In **Name**, replace it with the name in the table, choose the **EcuMModuleParameter** of the table (`POSTBUILD`) for **EcuMModuleRef**. Add the items in the order of the table: it is the order in which `EcuM` initializes the drivers.

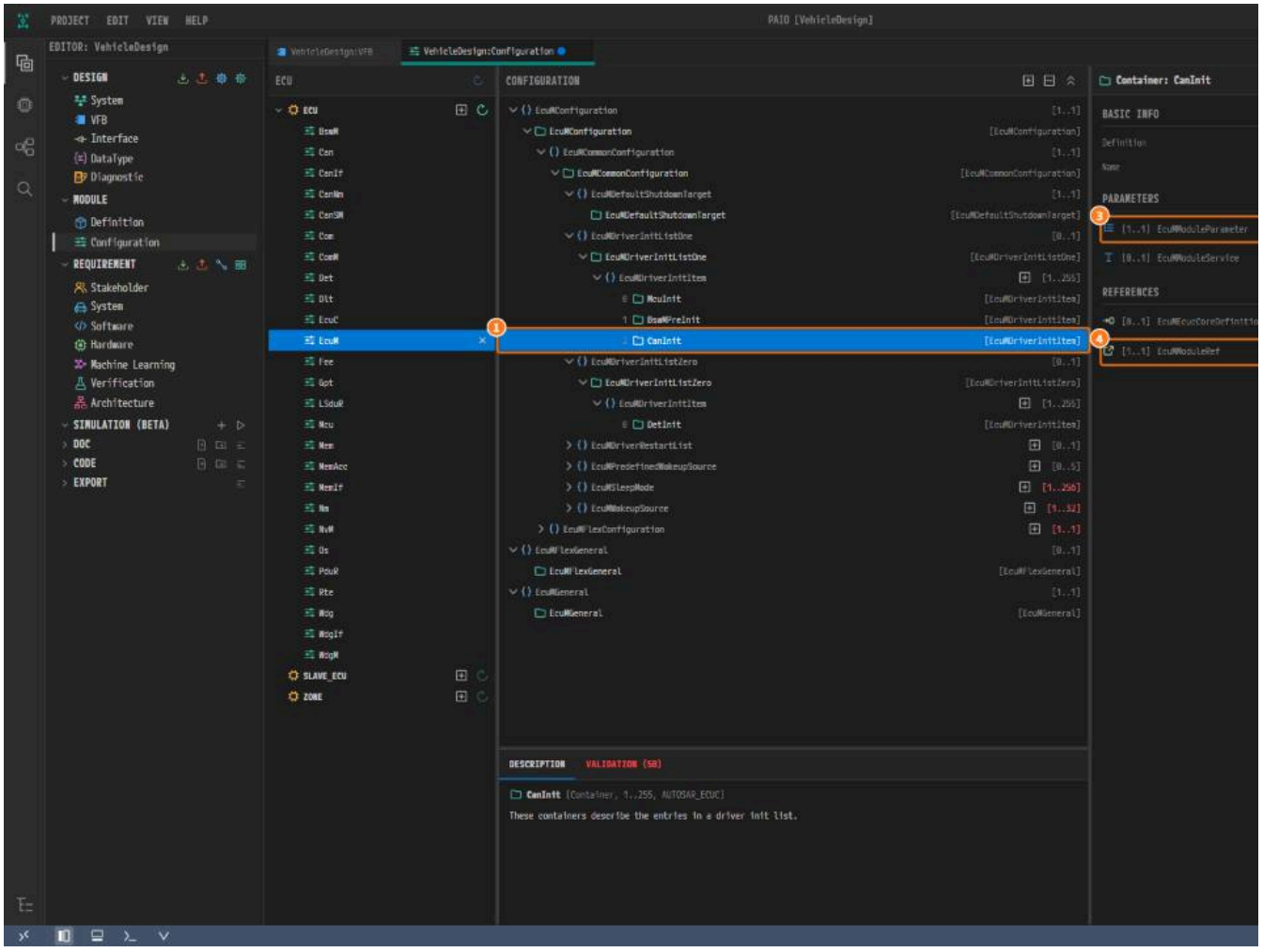


*Figure 6-8. ① `CanInit`, ② **Name**, ③ **EcuMModuleParameter**, ④ **EcuMModuleRef**.*

6. Add the remaining containers from the table and save with `Ctrl+S`.

Container	+ on	
`EcuMGeneral`	`EcuMGeneral`	**EcuMDevErrorDetect TRUE**, **EcuMMainFunction**
`EcuMFlexGeneral`	`EcuMFlexGeneral`	**EcuMAlarmClockPresent FALSE**, **EcuMModeHar** **EcuMSetProgrammableInterrupts FALSE**
`EcuMConfiguration`	`EcuMConfiguration`	None
`EcuMConfiguration/EcuMCommonConfiguration`	`EcuMCommonConfiguration`	**EcuMDefaultAppMode** `Os/OSDEFAULTAPPMODE`
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDefaultShutdownTarget`	`EcuMDefaultShutdownTarget`	**EcuMDefaultShutdownTarget** `EcuMShutdownTa` `EcuM/EcuMConfiguration/EcuMFlexConfigur` `EcuM/EcuMConfiguration/EcuMCommonConfig`
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListZero`	`EcuMDriverInitListZero`	None
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListZero/DetInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne`	`EcuMDriverInitListOne`	None
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/McuInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/BswMPreInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/CanInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/CanIfInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/LSduRInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/PduRInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/ComInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/CanNmInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/NmInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `NULL_PTR`, **EcuMModu**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/ComMInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverInitListOne/CanSMInit`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverRestartList`	`EcuMDriverRestartList`	None
`EcuMConfiguration/EcuMCommonConfiguration/EcuMDriverRestartList/DetRestart`	`EcuMDriverInitItem`	**EcuMModuleParameter** `POSTBUILD_PTR`, **EcuM**
`EcuMConfiguration/EcuMCommonConfiguration/Power`	`EcuMPredefinedWakeupSource`	**EcuMPredefinedWakeupSource** `POWER`, **EcuMR** `Mcu/McuPublishedInformation/PowerOn`
`EcuMConfiguration/EcuMCommonConfiguration/Reset`	`EcuMPredefinedWakeupSource`	**EcuMPredefinedWakeupSource** `RESET`, **EcuMR** `Mcu/McuPublishedInformation/Software`
`EcuMConfiguration/EcuMCommonConfiguration/WakeupTimer`	`EcuMWakeupSource`	**EcuMWakeupSourceId** `5`, **EcuMValidationTime** **EcuMWakeupSourcePolling FALSE**
`EcuMConfiguration/EcuMCommonConfiguration/Halt`	`EcuMSleepMode`	**EcuMSleepModeId** `0`, **EcuMSleepModeSuspen** `Mcu/McuModuleConfiguration/Sleep`, **EcuMW** `EcuM/EcuMConfiguration/EcuMCommonConfig`
`EcuMConfiguration/EcuMCommonConfiguration/Poll`	`EcuMSleepMode`	**EcuMSleepModeId** `1`, **EcuMSleepModeSuspen** `Mcu/McuModuleConfiguration/Normal`, **EcuM** `EcuM/EcuMConfiguration/EcuMCommonConfig`
`EcuMConfiguration/EcuMFlexConfiguration`	`EcuMFlexConfiguration`	**EcuMNormalMcuModeRef** `Mcu/McuModuleCon`

Container	+ on	
`EcuMConfiguration/EcuMFlexConfiguration/McuReset`	`EcuMResetMode`	**EcuMResetModeId** `0`
`EcuMConfiguration/EcuMFlexConfiguration/VecuMaster`	`EcuMFlexUserConfig`	**EcuMFlexUser** `0`
`EcuMConfiguration/EcuMFlexConfiguration/GoDownUsers`	`EcuMGoDownAllowedUsers`	**EcuMGoDownAllowedUserRef** `EcuM/EcuMConfi`
`EcuMConfiguration/EcuMFlexConfiguration/SupervisionFault`	`EcuMShutdownCause`	**EcuMShutdownCauseId** `4`

Setting Up BswM

`BswM` starts the RTE once `EcuM` reports RUN: a mode request port watches the state of `EcuM`, a condition compares it with `ECUM_STATE_RUN`, and a rule r

1. Click `BswM` in the list. Add `BswMGeneral` and enter its values from the table.

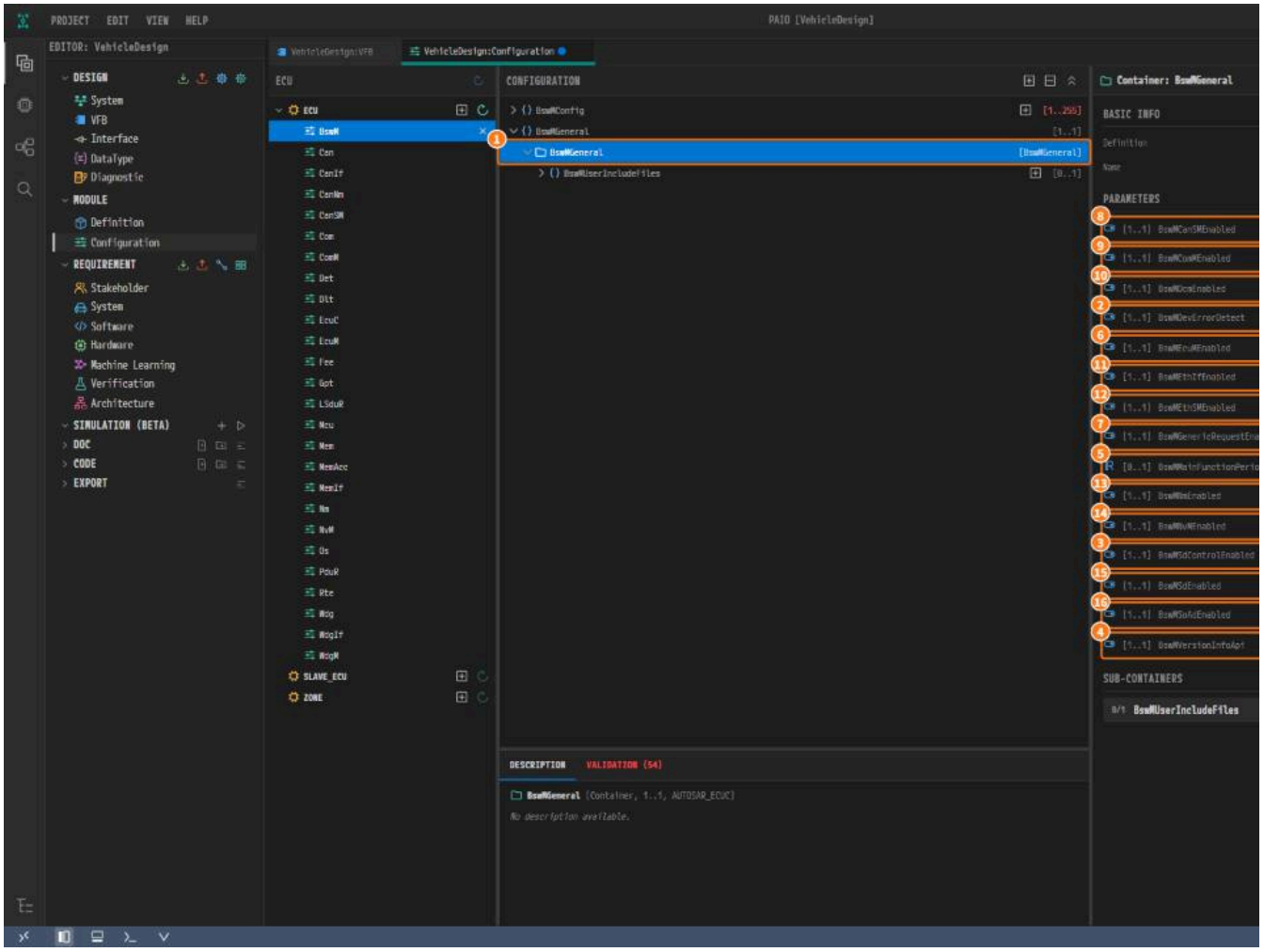


*Figure 6-9. ① `BswMGeneral`, ② to ⑯ its values. Click **TRUE** where the table says **TRUE**, type `0.01` in **BswMMainFunctionPeriod** and leave the others a*

2. Add the containers of `BswMConfig` in the order of the table. A rule names its expression and its action list; the action list `AlRteStart` comes after t

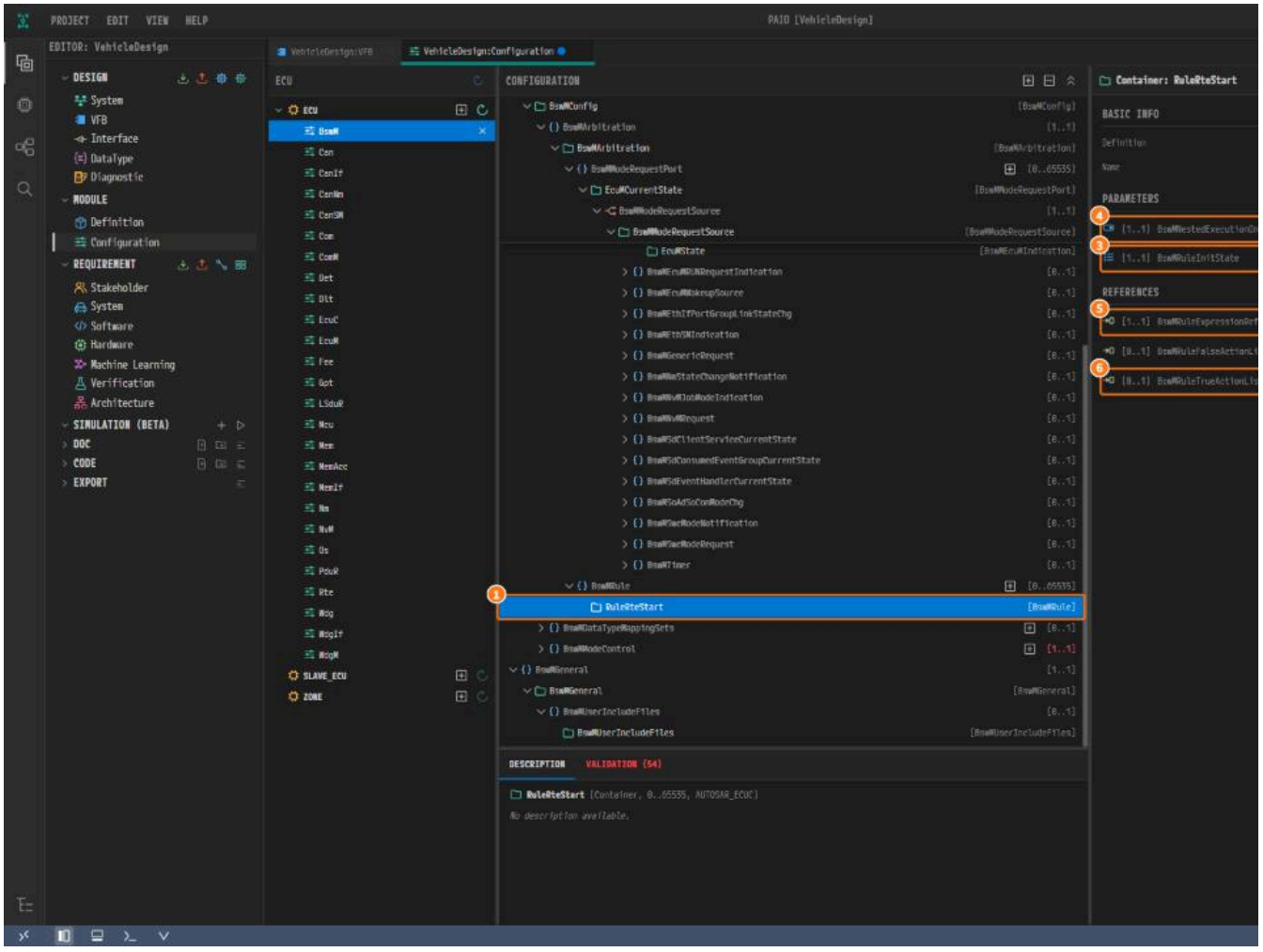


*Figure 6-10. ① `RuleRteStart`, ② **Name**, ③ **BswMRuleInitState**, ④ **BswMNestedExecutionOnly**, ⑤ **BswMRuleExpressionRef**, ⑥ **BswMRuleTrueActionLis***

3. Save with `Ctrl+S`.

Container	+ on	V
`BswMGeneral`	`BswMGeneral`	**BswMDevErrorDetect TRUE**, **BswMVersionInfoApi TRU TRUE**, **BswMGenericRequestEnabled TRUE**, **BswMCan BswMNmEnabled TRUE**
`BswMGeneral/BswMUserIncludeFiles`	`BswMUserIncludeFiles`	**BswMUserIncludeFile** `EcuM.h`
`BswMConfig`	`BswMConfig`	None
`BswMConfig/BswMArbitration`	`BswMArbitration`	None
`BswMConfig/BswMArbitration/EcuMCurrentState`	`BswMModeRequestPort`	**BswMRequestProcessing** `BSWM_IMMEDIATE`
`BswMConfig/BswMArbitration/EcuMCurrentState/BswMModeRequestSource`	`BswMModeRequestSource`	None
`BswMConfig/BswMArbitration/EcuMCurrentState/BswMModeRequestSource/EcuMState`	`BswMEcuMIndication`	None
`BswMConfig/BswMArbitration/CondEcuMStateRun`	`BswMModeCondition`	**BswMConditionType** `BSWM_EQUALS`, **BswMConditionI** `BswM/BswMConfig/BswMArbitration/EcuMCurrentS`
`BswMConfig/BswMArbitration/CondEcuMStateRun/BswMConditionValue`	`BswMConditionValue`	None
`BswMConfig/BswMArbitration/CondEcuMStateRun/BswMConditionValue/Value`	`BswMBswMode`	**BswMBswRequestedMode** `ECUM_STATE_RUN`
`BswMConfig/BswMArbitration/ExprEcuMRun`	`BswMLogicalExpression`	**BswMArgumentRef** `BswM/BswMConfig/BswMArbitra`
`BswMConfig/BswMArbitration/RuleRteStart`	`BswMRule`	**BswMRuleInitState** `BSWM_UNDEFINED`, **BswMRuleExp** `BswM/BswMConfig/BswMArbitration/ExprEcuMRun`, `BswM/BswMConfig/BswMModeControl/AlRteStart`
`BswMConfig/BswMModeControl`	`BswMModeControl`	None
`BswMConfig/BswMModeControl/RteStart`	`BswMAction`	None
`BswMConfig/BswMModeControl/RteStart/BswMAvailableActions`	`BswMAvailableActions`	None
`BswMConfig/BswMModeControl/RteStart/BswMAvailableActions/Kind`	`BswMRteStart`	None
`BswMConfig/BswMModeControl/AlRteStart`	`BswMActionList`	**BswMActionListExecution** `BSWM_TRIGGER`
`BswMConfig/BswMModeControl/AlRteStart/Item0`	`BswMActionListItem`	**BswMActionListItemIndex** `0`, **BswMAbortOnFail TRUI** `BswM/BswMConfig/BswMModeControl/RteStart`

Setting Up Mcu and Det

1. Click `Mcu` in the list and add its containers from the table. `CoreClock` is the clock the CAN driver refers to in chapter 9.

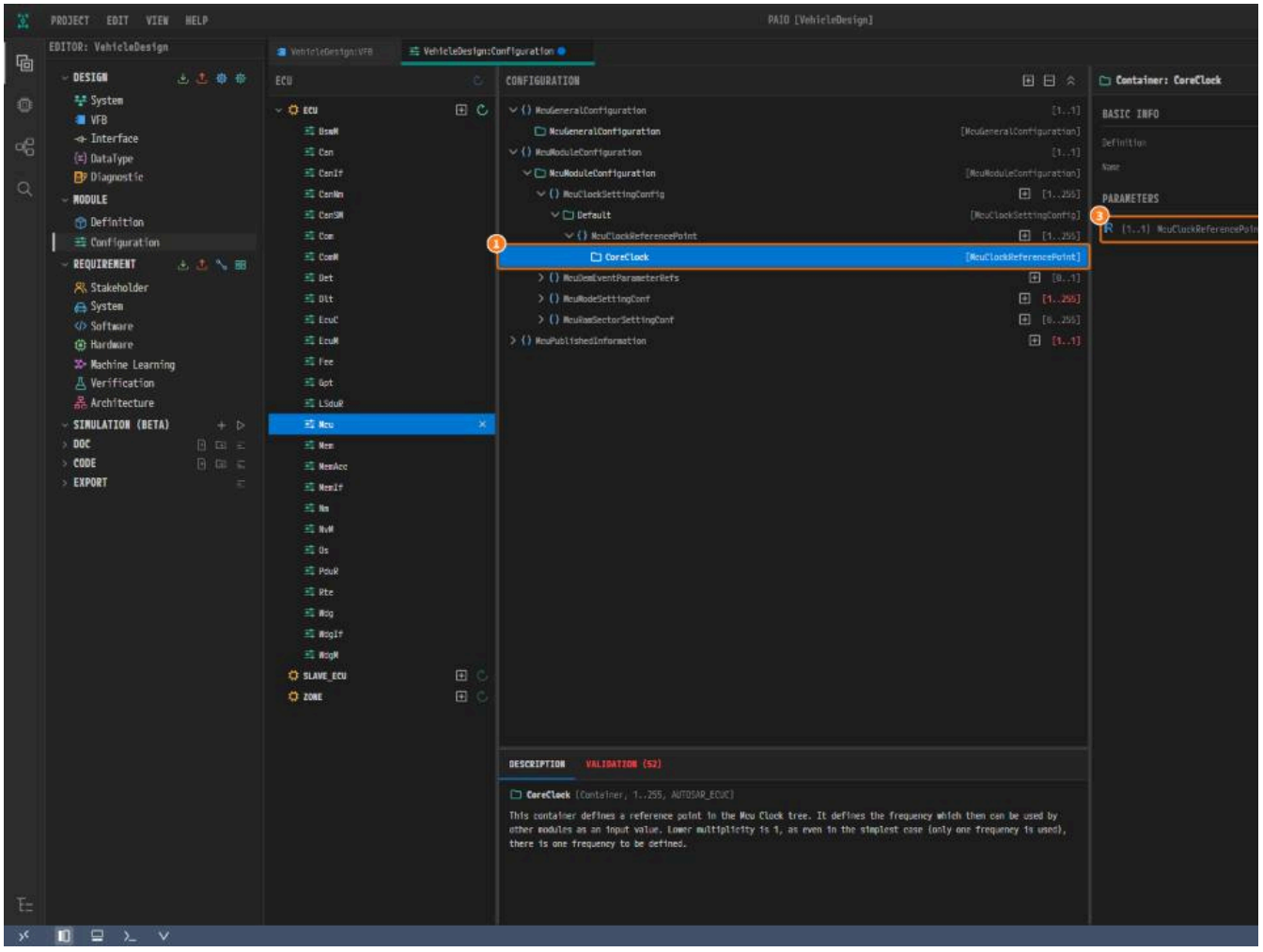


*Figure 6-11. ① `CoreClock`, ② **Name**, ③ **McuClockReferencePointFrequency**.*

2. Click `Det` in the list and add `DetGeneral`.

3. Save with `Ctrl+S`.

Container	+ on	Values
`McuGeneralConfiguration`	`McuGeneralConfiguration`	**McuDevErrorDetect** TRUE, **McuGetRamStateApi** TRUE, **McuNoPll** FALSE, **McuPerformResetApi**
`McuModuleConfiguration`	`McuModuleConfiguration`	**McuClockSrcFailureNotification** `DISABLED`, **McuNumberOfMcuModes** `2`, **McuRamSectors** `1`
`McuModuleConfiguration/Default`	`McuClockSettingConfig`	**McuClockSettingId** `0`
`McuModuleConfiguration/Default/CoreClock`	`McuClockReferencePoint`	**McuClockReferencePointFrequency** `100000000`
`McuModuleConfiguration/Normal`	`McuModeSettingConf`	**McuMode** `0`, **McuLowPowerMode** FALSE
`McuModuleConfiguration/Sleep`	`McuModeSettingConf`	**McuMode** `1`, **McuLowPowerMode** TRUE
`McuModuleConfiguration/Sector0`	`McuRamSectorSettingConf`	**McuRamDefaultValue** `0`, **McuRamSectionBaseAddress** `0`, **McuRamSectionSize** `256`
`McuPublishedInformation`	`McuPublishedInformation`	None
`McuPublishedInformation/PowerOn`	`McuResetReasonConf`	**McuResetReason** `0`
`McuPublishedInformation/Software`	`McuResetReasonConf`	**McuResetReason** `2`

Container	+ on	Values
`DetGeneral`	`DetGeneral`	**DetVersionInfoApi TRUE**

Importing the Rte Values File

Six references of `Rte` cannot be entered in the property panel yet: their lists do not offer the target, and a typed path is not kept.

Container	Reference	Target
`BswMInstance/RteEventToTaskMapping_0`	**RteEventRef**	The BswM poll event `BswM`
`RteBswModuleInstance_0`, `_3`, `_4`, `_5`, `_6` (`BswM`, `CanIf`, `CanSM`, `ComM`, `Com`), each `RteBswExclusiveAreaImpl_0`	**RteBswExclusiveAreaRef**	The module's BSW exclusiv

The Rte values file brings these containers with their references. Importing `Rte` replaces its configuration whole, so the file also carries the SW-C type a created; otherwise the import would remove them.

1. Click the **Import** icon in the **DESIGN** header of the **EDITOR** view.

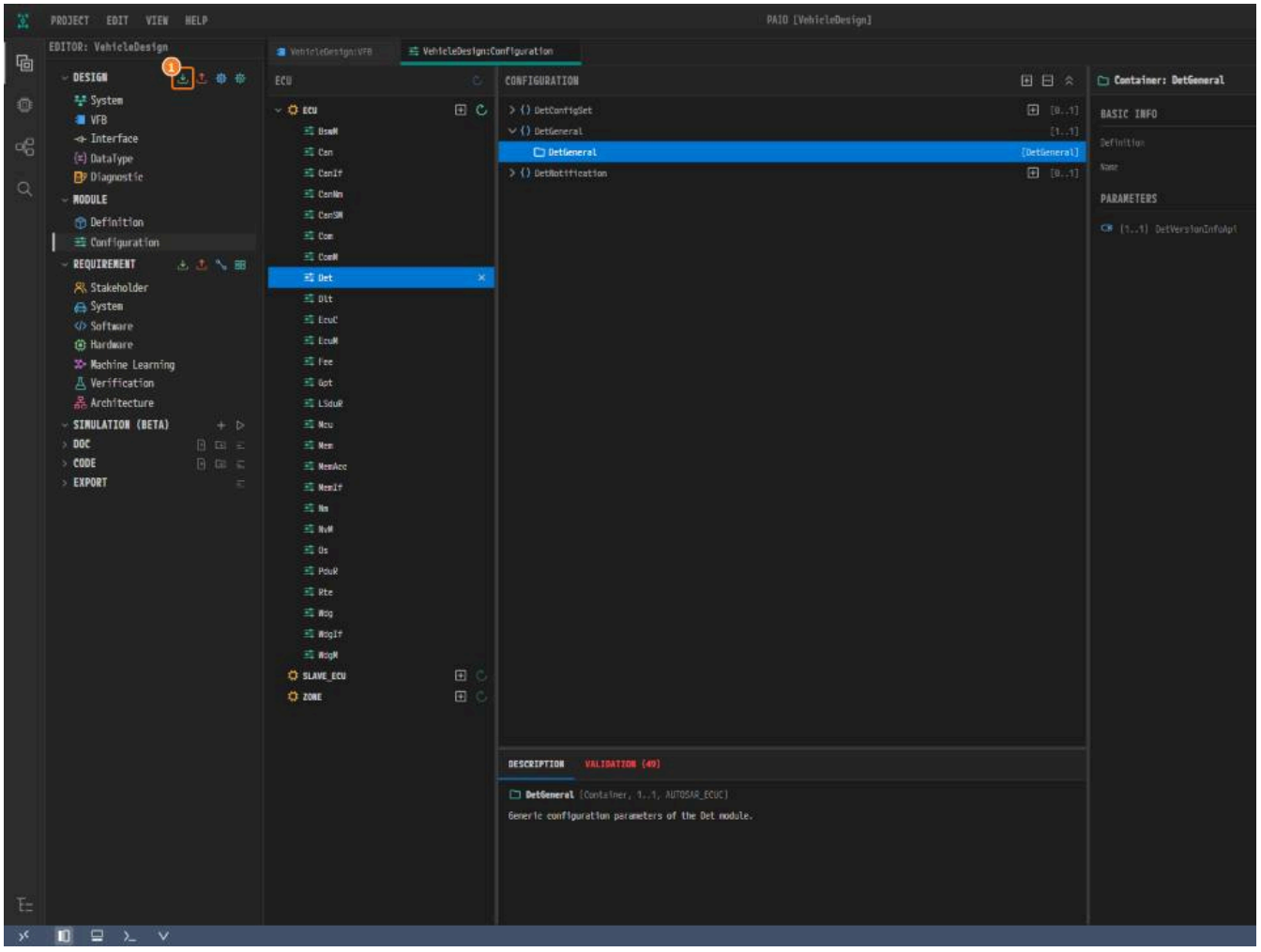


*Figure 6-12. ① **Import**.*

2. Choose the saved file `VehicleDesign_Rte.arxml` in the file dialog. The **Conflict Resolution** window opens with `Rte`, the one module of the file. The Click **Module Definition** on the left, then **All → Use Imported**. When the button reads `Resolve 1/1`, click it.

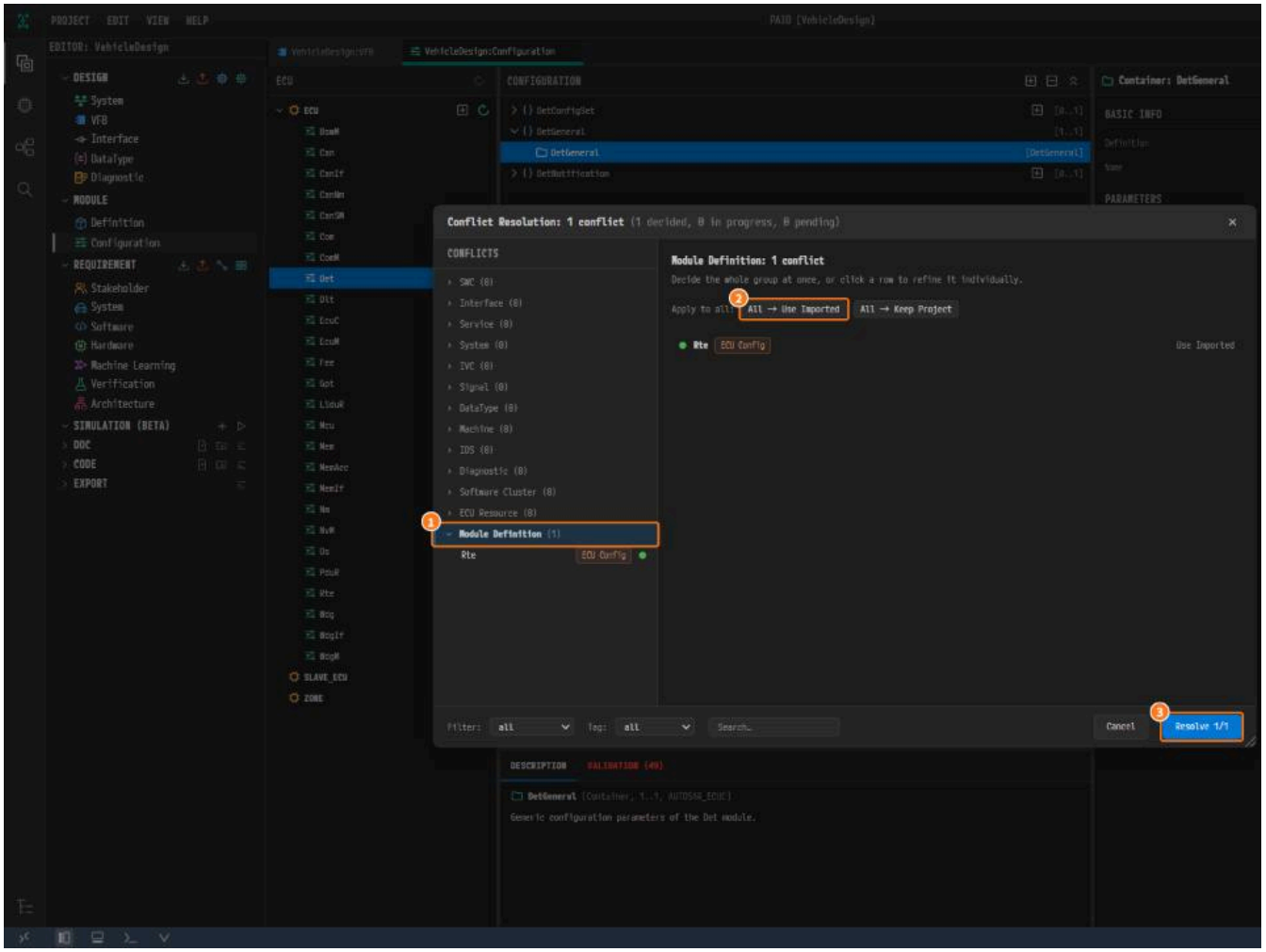


*Figure 6-13. ① **Module Definition**, ② **All → Use Imported**, ③ `Resolve 1/1`.*

The message `Import complete` appears at the bottom right, with `1 replaced` under it. You complete the `Rte` containers in chapter 8.

Why Is It Designed This Way?

- **No AUTO CONFIGURATION.** Every BSW value comes from you, so each one is visible in the editor. Configured this way from an empty project, validatior of 15 modules with no errors.
- **Platform Import is not configuration.** It creates the module definitions, empty configurations and the BSW service components. It sets no BSW value
- **The five modules PRTE requires.** PRTE needs the configuration of `Os`, `Rte`, `EcuM`, `BswM` and `Mcu` to generate code. If any one is empty, generation st
- **The Rte values file is temporary.** It holds only the six references the property panel does not offer yet, and the SW-C pairs the import would otherwi you enter the references in chapter 8 like every other value.
- **Network management at start-up. BswMNmEnabled** and the items `CanNmInit` and `NmInit` serve the start-up of network management at run time; F `CanNm` and `Nm` need no exclusive area reference: neither has a BSW exclusive area that PRTE requires.

Checklist

- [ ] The composition `PRTE_ECU` is inside the frame.
- [ ] `EcuM` shows `EcuMConfiguration`, `EcuMFlexGeneral` and `EcuMGeneral`, and `EcuMDriverInitListOne` holds the eleven items from `McuInit` to `CanSI`
- [ ] `BswM` holds `RuleRteStart` and `AlRteStart`.
- [ ] The message `Import complete` with `1 replaced` appeared.
- [ ] The tab `VehicleDesign:Configuration` carries no modified mark.

Next Step

In Os Configuration you create the tasks, alarms and counter of `Os` by hand.

Os Configuration

What You Will Do

`Os` decides what runs and when. In this chapter you create its containers by hand in the ECU Configuration editor: the settings of the operating system that `EcuM` names, one counter, four tasks and three alarms that activate the periodic tasks.

Before You Begin

- Start after finishing PRTE Setup.
- Click **Configuration** under **MODULE** to open the ECU Configuration editor.
- **+** at the right end of a row in the **CONFIGURATION** tree adds a container of that definition, with its default values, and selects it. Change only the valu pick a reference from its list.
- A bold name in the **Container** column is not the name **+** gives. Type it in **Name** in the panel and press `Tab`.

Container	+ on	Values
`OsOS`	`OsOS`	**OsStatus** `EXTENDED`, **OsStackMonitoring** **TRUE**, **OsUseGetServiceId** **TRUE**, **OsUseParameterAcc**
`OsOS/OsHooks`	`OsHooks`	**OsErrorHook**, **OsPostTaskHook**, **OsPreTaskHook**, **OsShutdownHook**, **OsStartupHook** all **FALSE**
`OSDEFAULTAPPMODE`	`OsAppMode`	None
`RES_AUTOSAR_ECUM`	`OsResource`	**OsResourceProperty** `STANDARD`
`OsCounter_0`	`OsCounter`	**OsCounterMaxAllowedValue** `1000000`, **OsCounterMinCycle** `1`, **OsCounterTicksPerBase** `1`, Os
`OsTask_0`	`OsTask`	**OsTaskActivation** `1`, **OsTaskPeriod** `0.01`, **OsTaskPriority** `2`, **OsTaskSchedule** `FULL`, **OsTaskSt**
`OsTask_EcuMStartup`	`OsTask`	**OsTaskActivation** `1`, **OsTaskPriority** `3`, **OsTaskSchedule** `FULL`, **OsTaskStackSize** `32768`
`OsTask_EcuMStartup/OsTaskAutostart`	`OsTaskAutostart`	**OsTaskAppModeRef** `Os/OSDEFAULTAPPMODE`
`OsTask_2`	`OsTask`	**OsTaskActivation** `1`, **OsTaskPeriod** `0.02`, **OsTaskPriority** `1`, **OsTaskSchedule** `FULL`, **OsTaskSt**
`OsTask_3`	`OsTask`	**OsTaskActivation** `1`, **OsTaskPeriod** `0.01`, **OsTaskPriority** `1`, **OsTaskSchedule** `FULL`, **OsTaskSt**
`OsAlarm_0`	`OsAlarm`	**OsAlarmCounterRef** `Os/OsCounter_0`
`OsAlarm_0/OsAlarmAction`	`OsAlarmAction`	None
`OsAlarm_0/OsAlarmAction/OsAlarmActivateTask`	`OsAlarmActivateTask`	**OsAlarmActivateTaskRef** `Os/OsTask_0`
`OsAlarm_1`	`OsAlarm`	**OsAlarmCounterRef** `Os/OsCounter_0`
`OsAlarm_1/OsAlarmAction`	`OsAlarmAction`	None
`OsAlarm_1/OsAlarmAction/OsAlarmActivateTask`	`OsAlarmActivateTask`	**OsAlarmActivateTaskRef** `Os/OsTask_2`
`OsAlarm_2`	`OsAlarm`	**OsAlarmCounterRef** `Os/OsCounter_0`
`OsAlarm_2/OsAlarmAction`	`OsAlarmAction`	None
`OsAlarm_2/OsAlarmAction/OsAlarmActivateTask`	`OsAlarmActivateTask`	**OsAlarmActivateTaskRef** `Os/OsTask_3`

Setting Up the Operating System

1. Click `Os` in the list. The **CONFIGURATION** tree lists the container definitions of `Os`, all empty. Click **+** on the `OsOS` row. The editor adds `OsOS` and selec

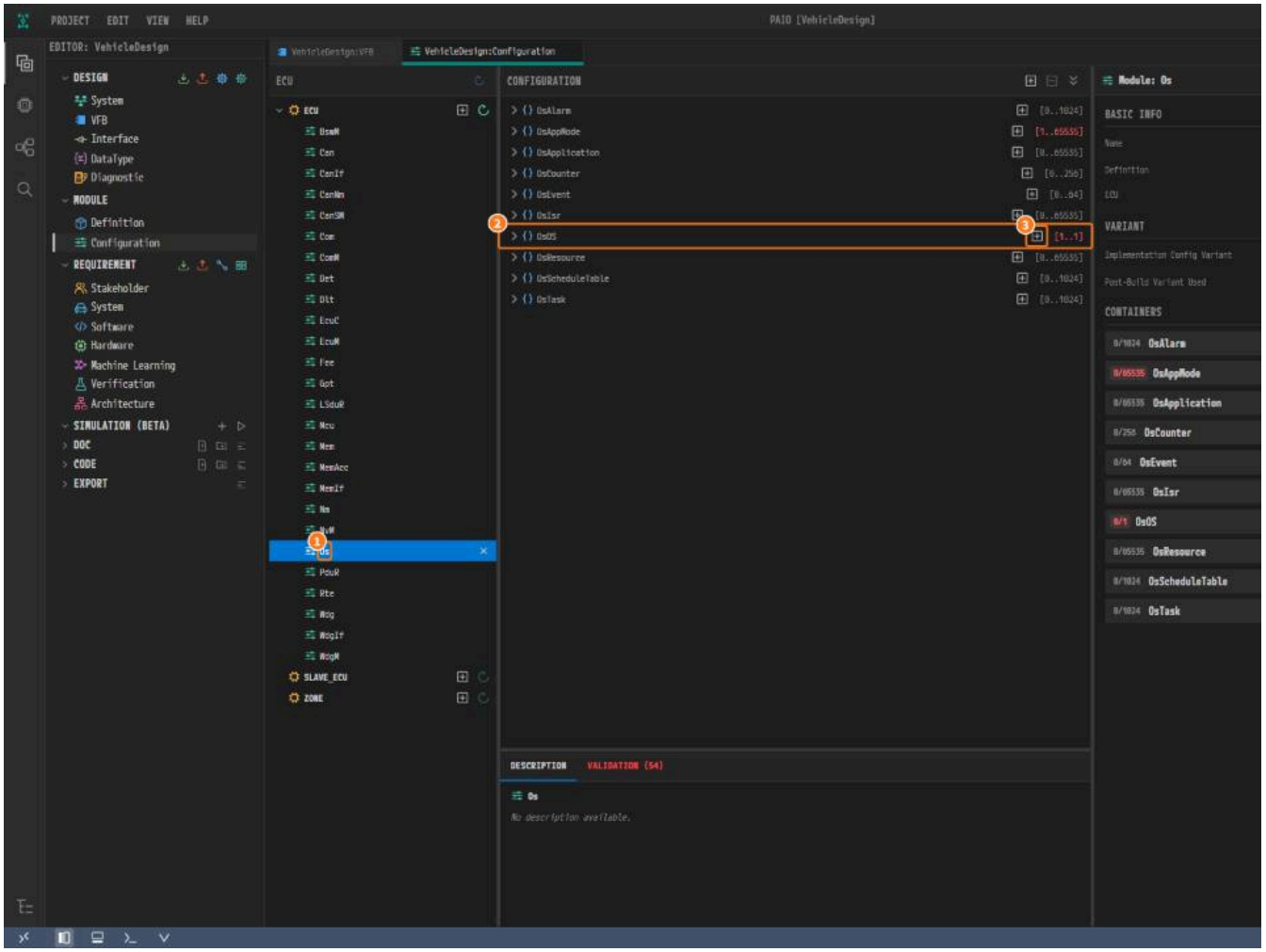


Figure 7-1. ① `Os`, ② the `OsOS` row, ③ +.

2. In the panel on the right, choose `EXTENDED` for **OsStatus**. Click **TRUE** for **OsStackMonitoring**, **OsUseGetServiceId** and **OsUseParameterAccess**.

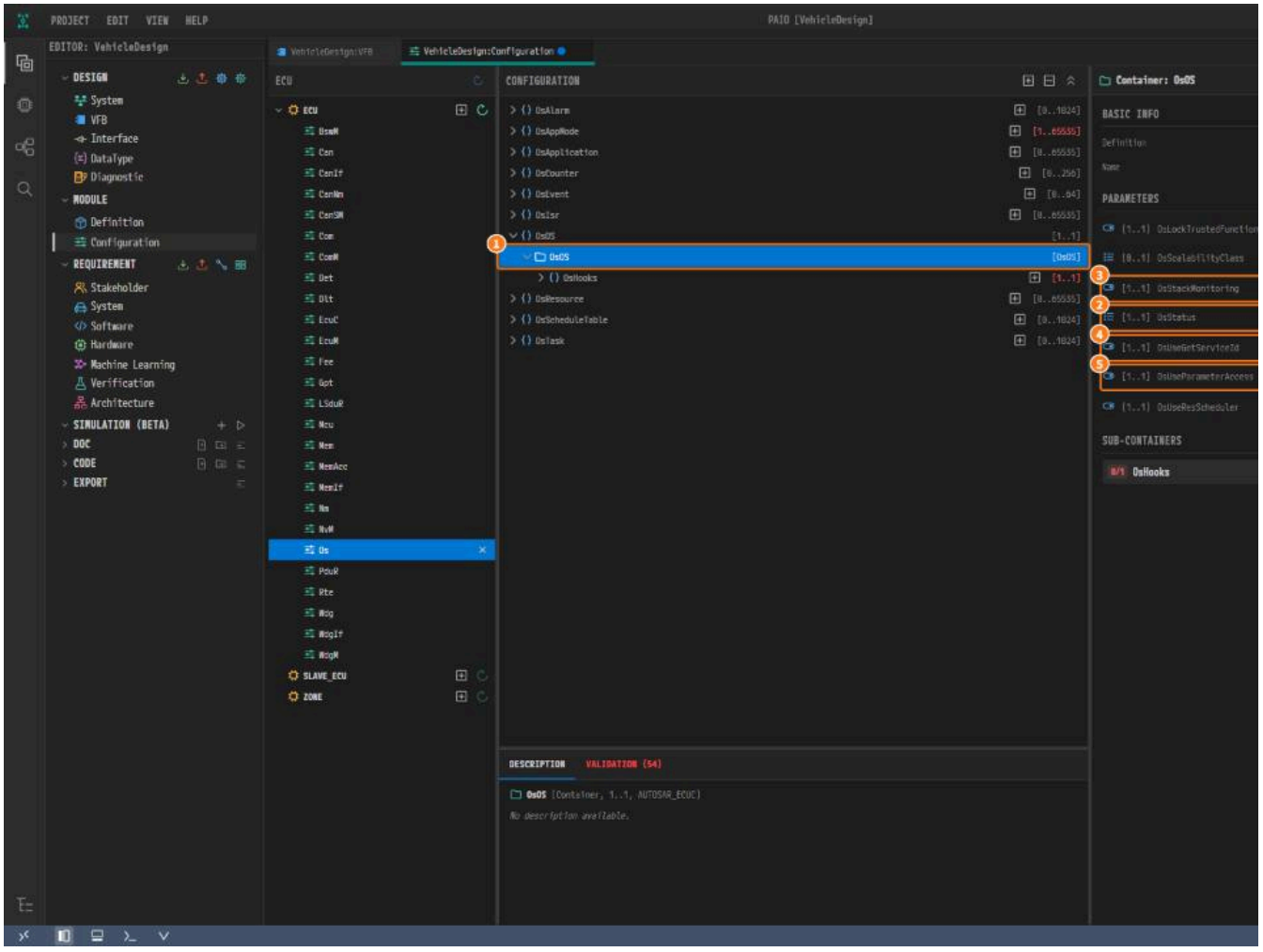


*Figure 7-2. ① `OsOS`, ② **OsStatus**, ③ **OsStackMonitoring**, ④ **OsUseGetServiceId**, ⑤ **OsUseParameterAccess**.*

3. Under `OsOS`, click **+** on the `OsHooks` row and click **FALSE** for the five hooks.
4. Click **+** on the `OsAppMode` row. In **Name**, replace the given name with `OSDEFAULTAPPMODE` and press `Tab`.
5. Click **+** on the `OsResource` row. Rename it to `RES_AUTOSAR_ECUM` the same way and choose `STANDARD` for **OsResourceProperty**.

   The two names must be exact: the `EcuM` configuration you entered in PRTE Setup refers to them.
6. Click **+** on the `OsCounter` row and enter the values of `OsCounter_0` from the table.

Creating the Tasks

1. Click **+** on the `OsTask` row. The editor adds `OsTask_0` and selects it.

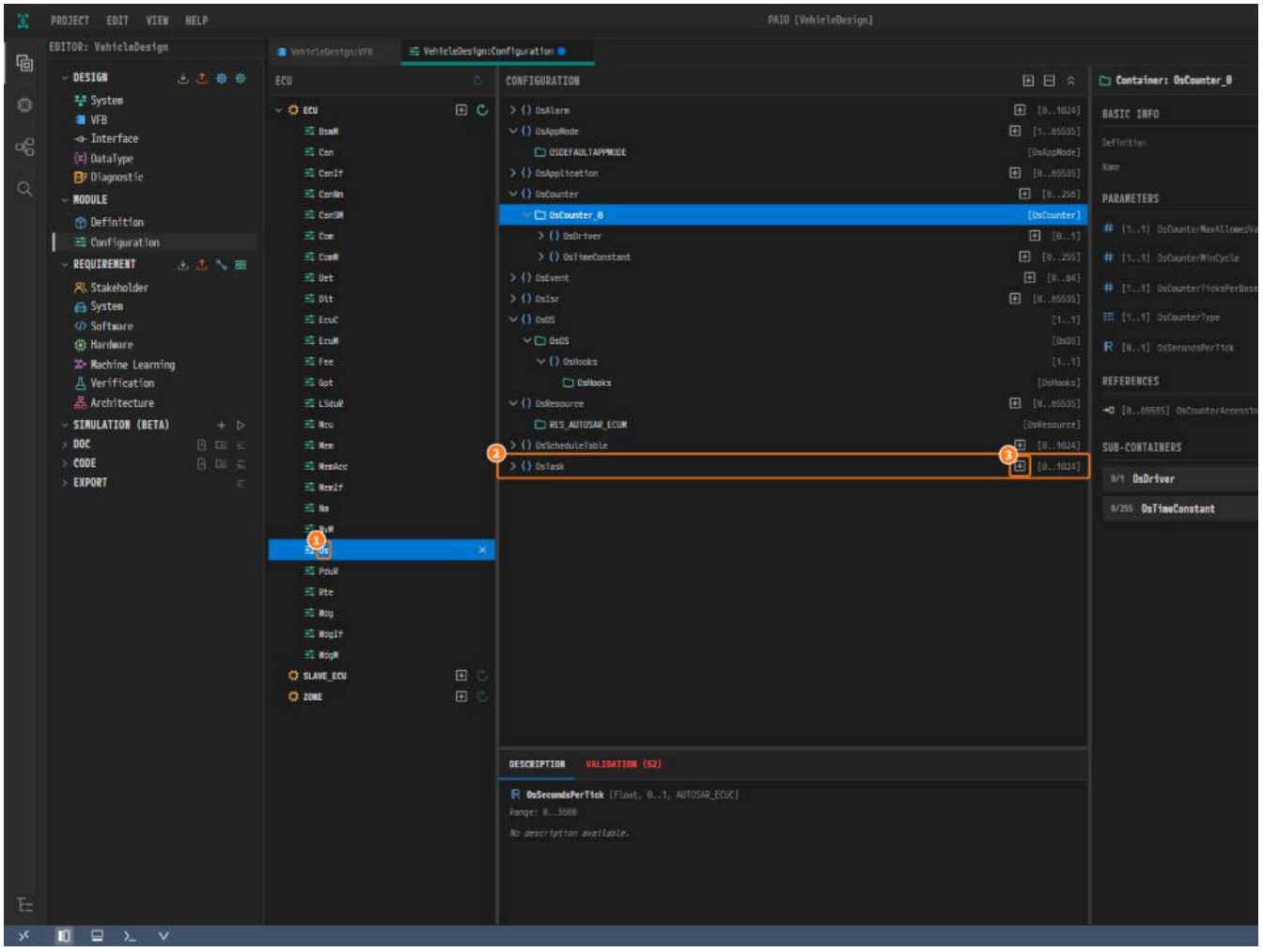


Figure 7-3. ① `Os`, ② the `OsTask` row, ③ +.

2. Enter the values of `OsTask_0` from the table.

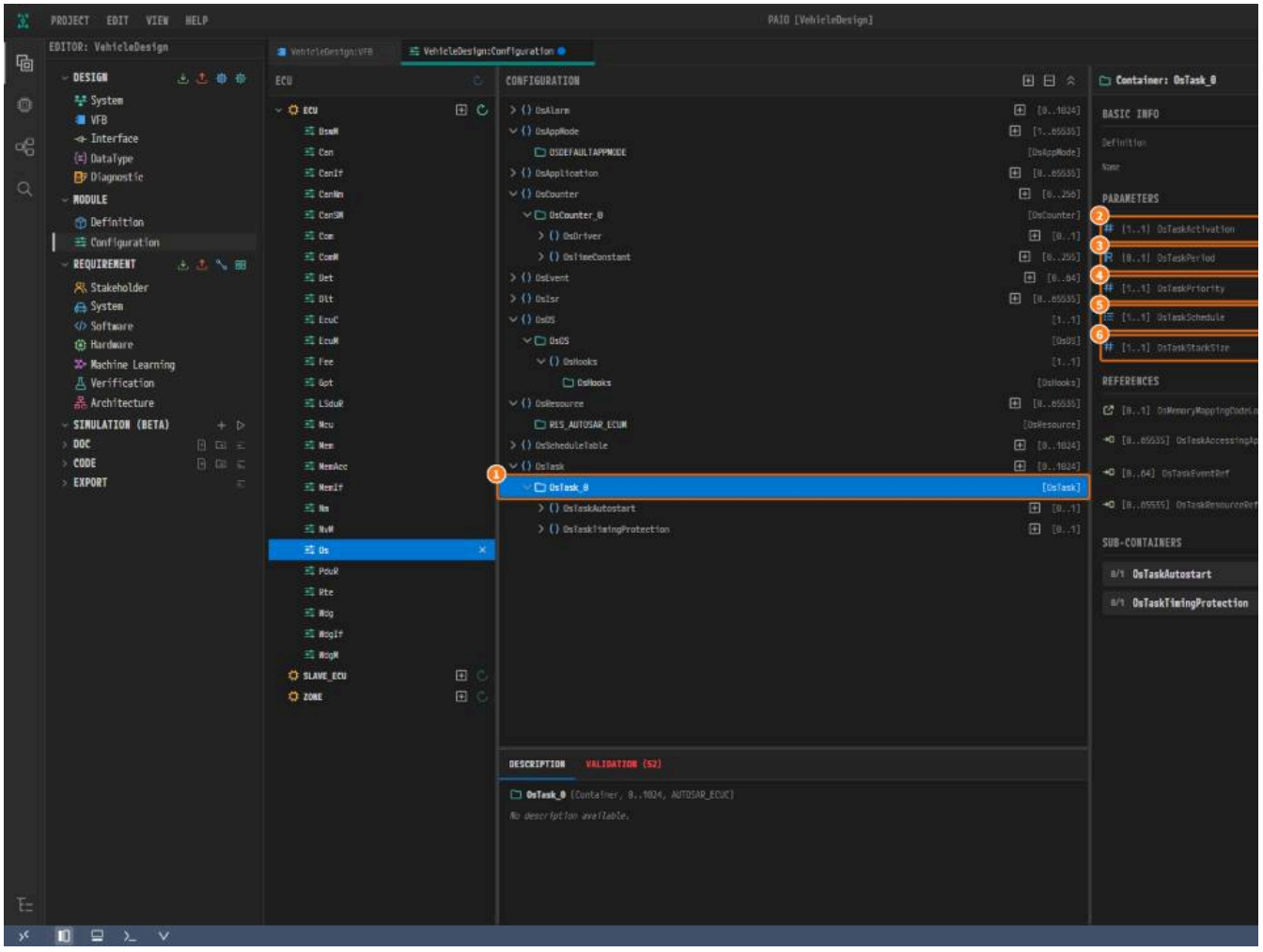


*Figure 7-4. ① `OsTask_0`, ② **OsTaskActivation**, ③ **OsTaskPeriod**, ④ **OsTaskPriority**, ⑤ **OsTaskSchedule**, ⑥ **OsTaskStackSize**.*

3. Click **+** on the `OsTask` row again. The editor adds `OsTask_1`. In **Name**, replace it with `OsTask_EcuMStartup` and press `Tab`. Enter its values from the t

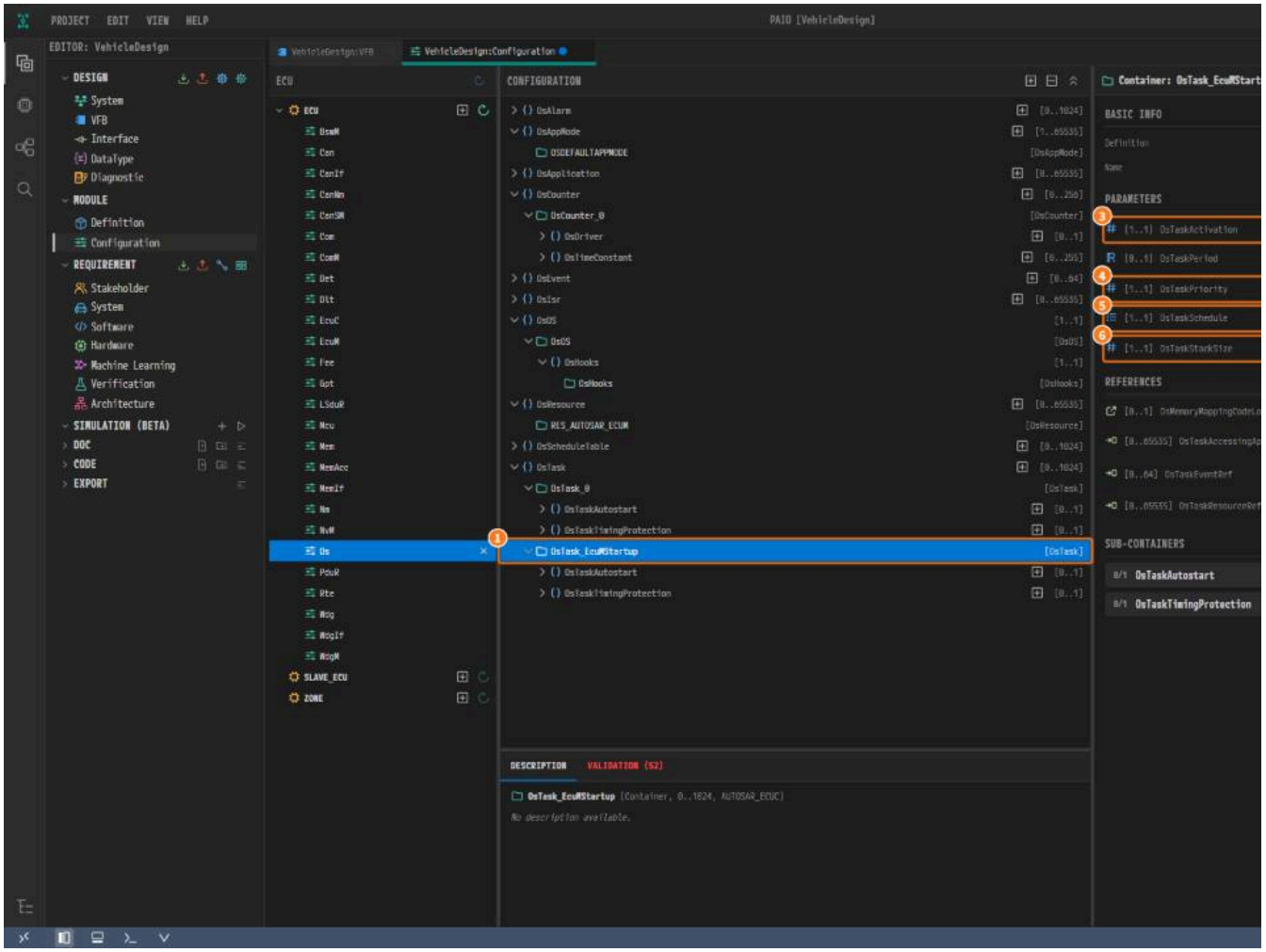


*Figure 7-5. ① `OsTask_EcuMStartup`, ② **Name**, ③ **OsTaskActivation**, ④ **OsTaskPriority**, ⑤ **OsTaskSchedule**, ⑥ **OsTaskStackSize**.*

4. Under `OsTask_EcuMStartup`, click **+** on the `OsTaskAutostart` row. In the new `OsTaskAutostart`, pick `Os/OSDEFAULTAPPMODE` for **OsTaskAppModeRe**

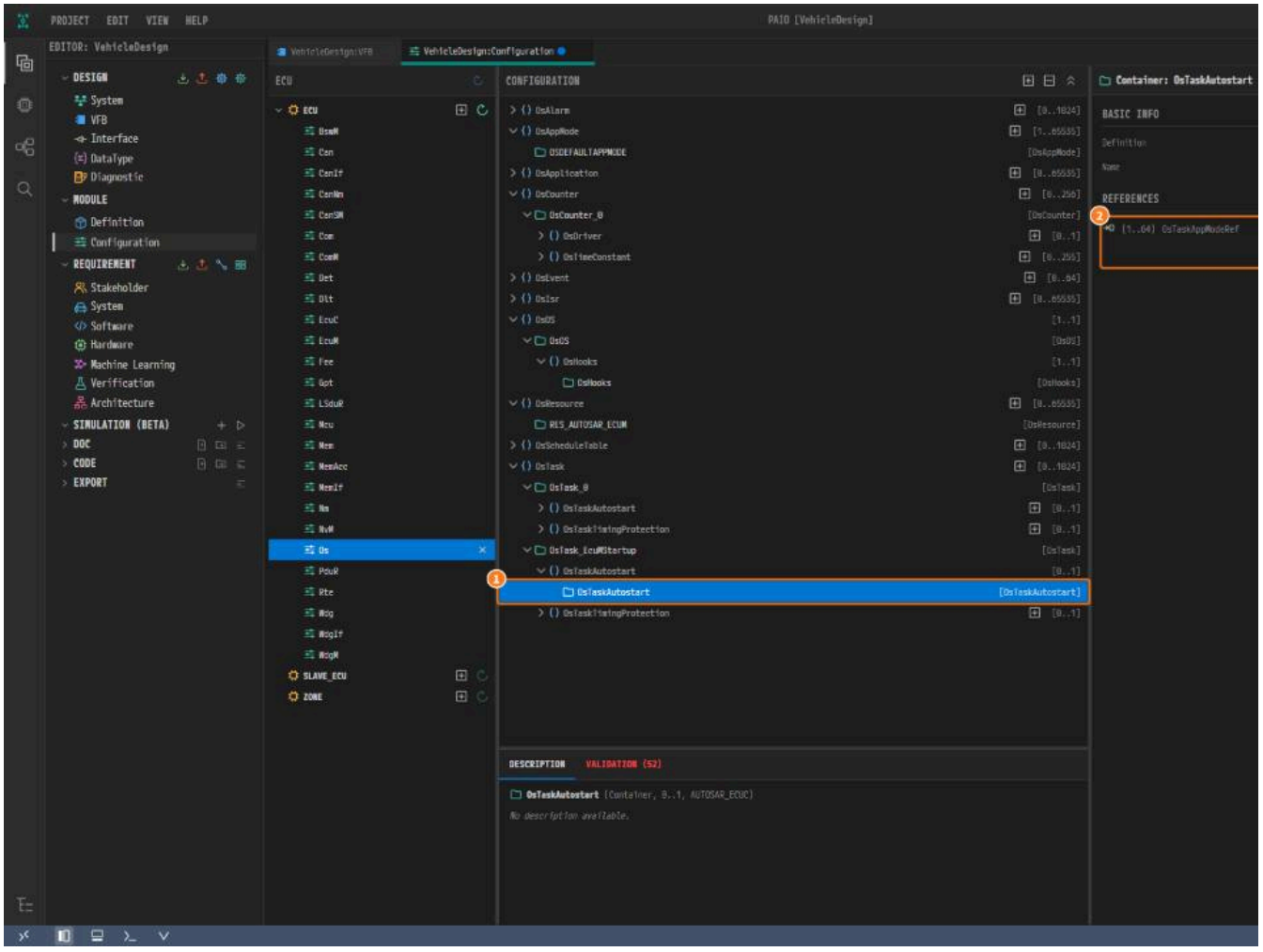


*Figure 7-6. ① `OsTaskAutostart`, ② **OsTaskAppModeRef**.*

5. Click **+** on the `OsTask` row again. The editor names the new task `OsTask_1` again, because that number is free. Rename it to `OsTask_2` and enter its
6. Create `OsTask_3` the same way: **+** gives `OsTask_1`, which you rename to `OsTask_3`.

Creating the Alarms

1. Click **+** on the `OsAlarm` row. In the new `OsAlarm_0`, pick `Os/OsCounter_0` for **OsAlarmCounterRef**.

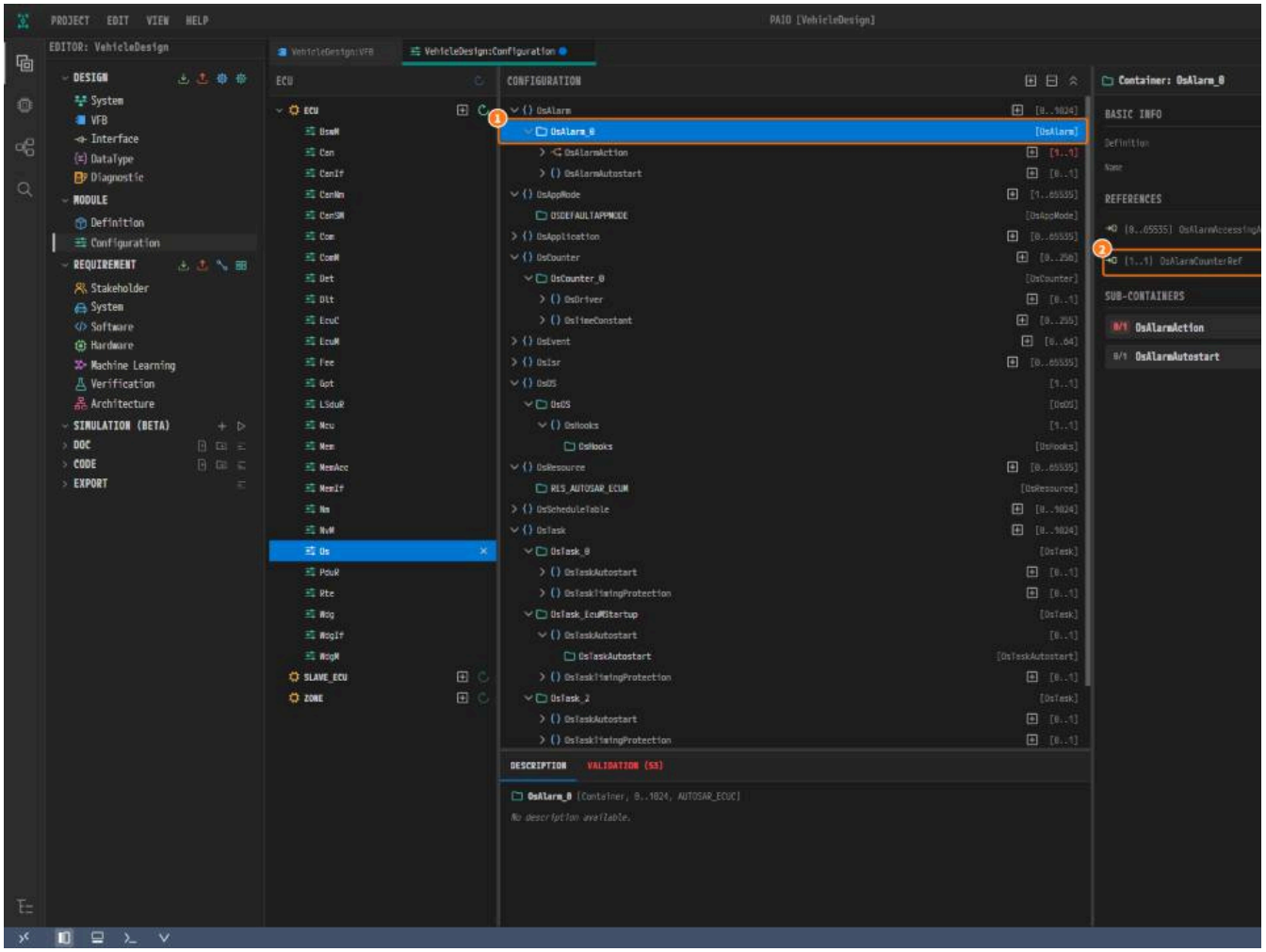


*Figure 7-7. ① `OsAlarm_0`, ② **OsAlarmCounterRef**.*

2. Under `OsAlarm_0`, click **+** on the `OsAlarmAction` row. Under the new `OsAlarmAction`, click **+** on the `OsAlarmActivateTask` row.

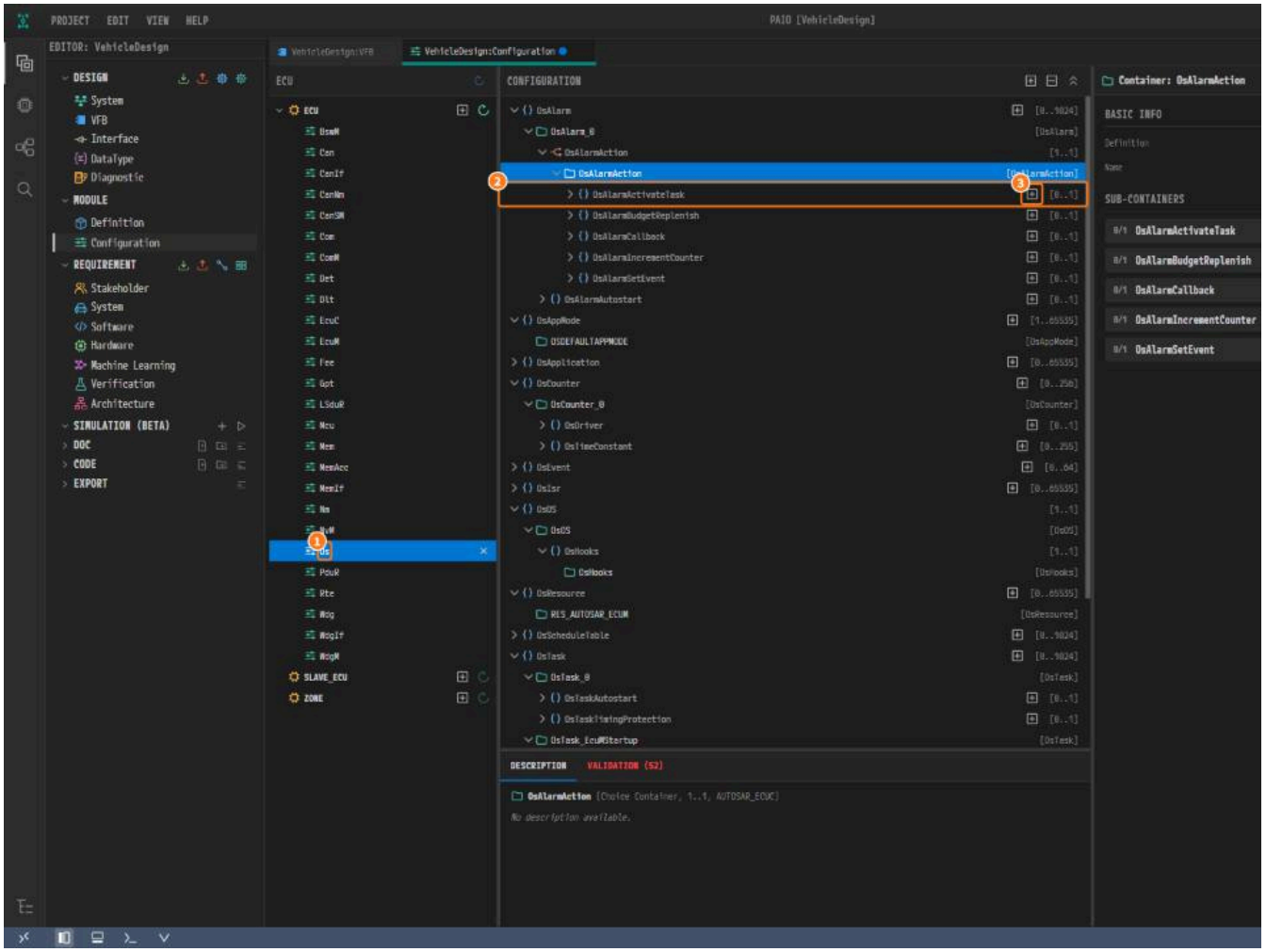


*Figure 7-8. ① `Os`, ② the `OsAlarmActivateTask` row under `OsAlarm_0/OsAlarmAction`, ③ **+**.*

3. In the new `OsAlarmActivateTask`, pick `Os/OsTask_0` for **OsAlarmActivateTaskRef**.

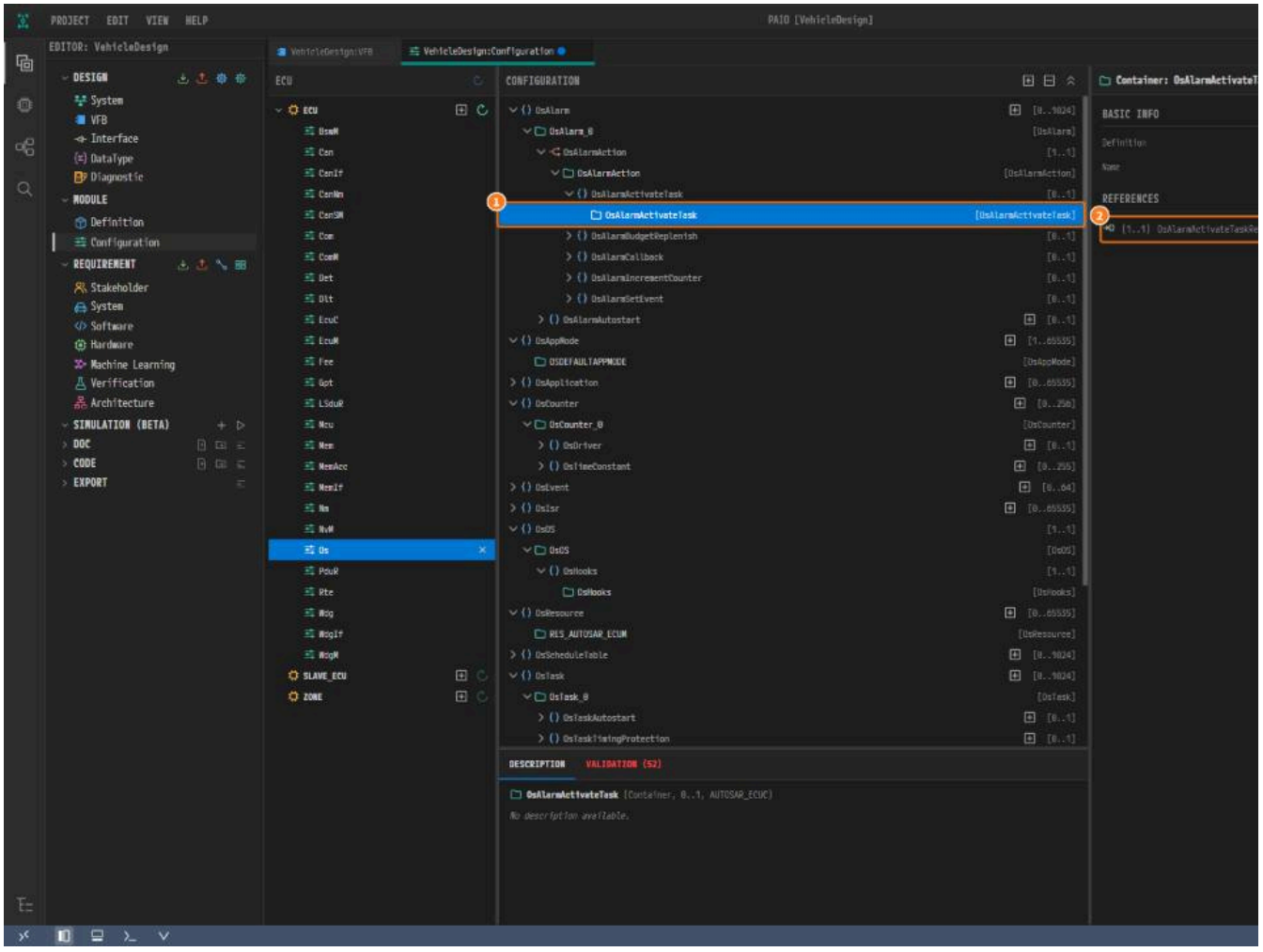


*Figure 7-9. ① `OsAlarmActivateTask`, ② **OsAlarmActivateTaskRef**.*

4. Create `OsAlarm_1` and `OsAlarm_2` the same way from the table. `OsAlarm_1` activates `OsTask_2`, and `OsAlarm_2` activates `OsTask_3`.
5. Save with `Ctrl+S`.

Why Is It Designed This Way?

- **Fixed names.** The `EcuM` of PRTE Setup names `OSDEFAULTAPPMODE` and `RES_AUTOSAR_ECUM`. PAIO writes the start-up task source `src/EcuM_StartupT` `OsTask_EcuMStartup`.
- **One task per timing.** `OsTask_0` runs the BSW main functions every 10 ms, `OsTask_2` the sensor runnable every 20 ms, and `OsTask_3` the actuator ru events to these tasks and alarms.
- **The start-up task.** `OsTask_EcuMStartup` has no period. **OsTaskAutostart** starts it once in `OSDEFAULTAPPMODE`, and its priority `3` is the highest.
- **One counter.** `OsCounter_0` ticks every 0.01 s and drives all three alarms.
- **EXTENDED.** The extended status makes the OS services report errors, which helps during development.

Checklist

- [ ] The **CONFIGURATION** tree of `Os` holds `OsOS`, `OSDEFAULTAPPMODE`, `RES_AUTOSAR_ECUM` and `OsCounter_0`.
- [ ] It holds the four tasks `OsTask_0`, `OsTask_EcuMStartup`, `OsTask_2` and `OsTask_3`.
- [ ] `OsAlarm_0`, `OsAlarm_1` and `OsAlarm_2` activate `OsTask_0`, `OsTask_2` and `OsTask_3`.
- [ ] The tab `VehicleDesign:Configuration` carries no modified mark.

Next Step

In Rte Configuration you map the runnables and BSW main functions to these tasks.

Rte Configuration

What You Will Do

In this chapter you configure the module `Rte` by hand. The Rte maps each runnable event of the SW-Cs and each main function of the BSW, the main fur alarm that activates it.

Before You Begin

- Start after finishing Os Configuration. The Rte references pick the Os tasks and alarms made there.
- Open the tab `VehicleDesign:Configuration` and click **MODULE** > **Configuration** > `Rte`.
- "+" means the **+** at the right end of a row in the **CONFIGURATION** tree. It adds the container and selects it. Only the values given here are changed.
- References are picked from their dropdown, except `RteBswImplementationRef` and `RteBswEventRef`: their dropdown offers nothing, so type the p
- The panel may show only the end of a long path.

Some containers already exist because the Rte values file of PRTE Setup made them. Do not add those again; only complete them.

Origin	Containers
Rte values file	The SW-C instances (`EcuMInstance`, `BswMInstance`, ..., `SwcSensorInstance`, `SwcActuatorInstance`)
Rte values file	`BswMInstance/RteEventToTaskMapping_0`
Rte values file	`RteBswModuleInstance_0`, `_3`, `_4`, `_5`, `_6`, each with `RteBswExclusiveAreaImpl_0`
"+" (26)	`RteGeneration`, `RteBswGeneral`, `RteInitializationBehavior_0`, three `RteEventToTaskMapping`, `RteModeMachineInstanceConfig_0`, `RteOsInteraction`, t `RteBswModuleInstance_1`, `_2` and `_7`, twelve `RteBswEventToTaskMapping`

Checking What the Rte Values File Brought

1. Click `Rte`. The tree already holds the SW-C instances and five BSW module instances.

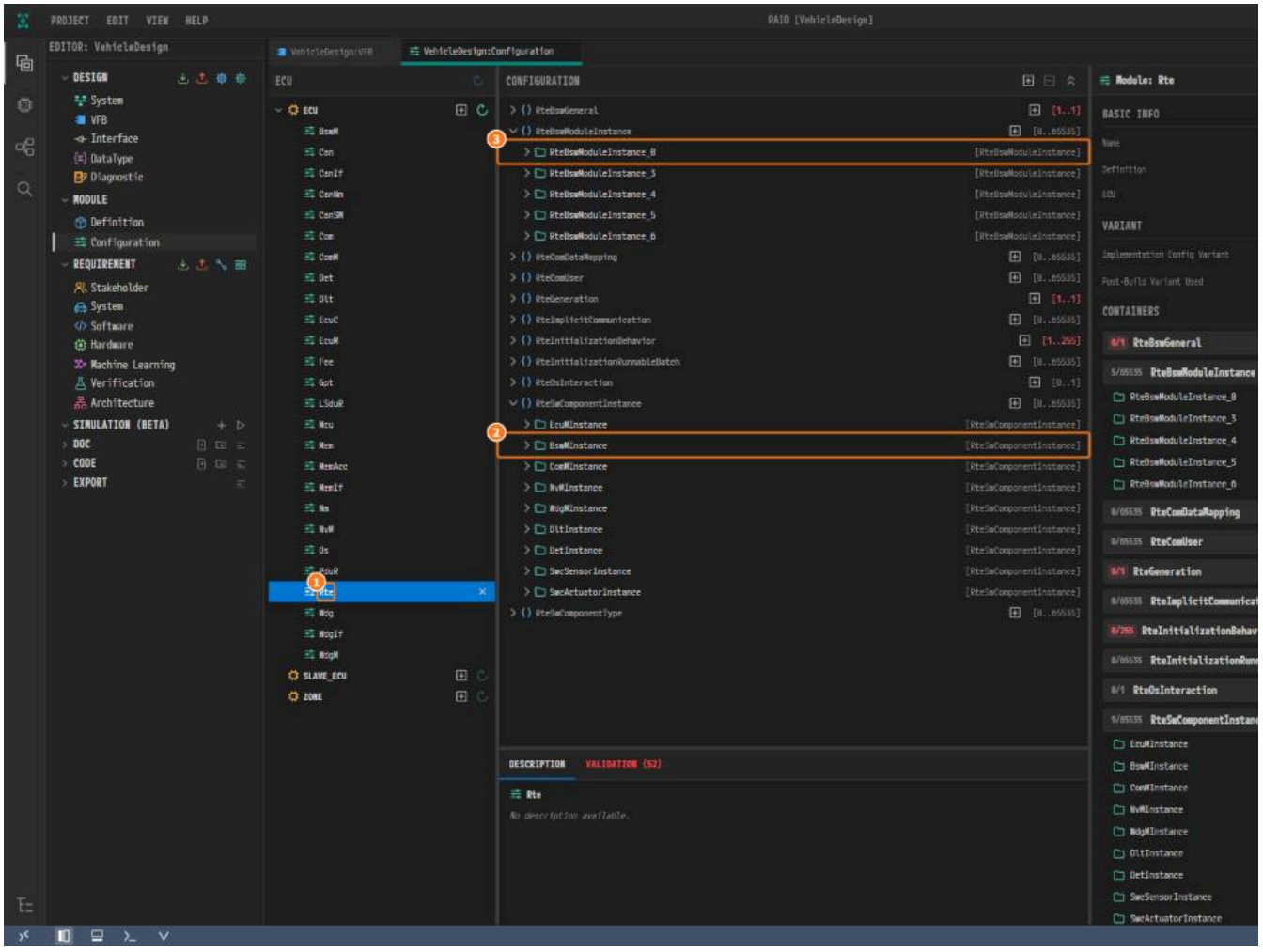


Figure 8-1. ① `Rte`, ② `BswMInstance` (from the Rte values file), ③ `RteBswModuleInstance_0` (from the Rte values file).

Generation and Initialization

1. Press "+" on the `RteGeneration` row.

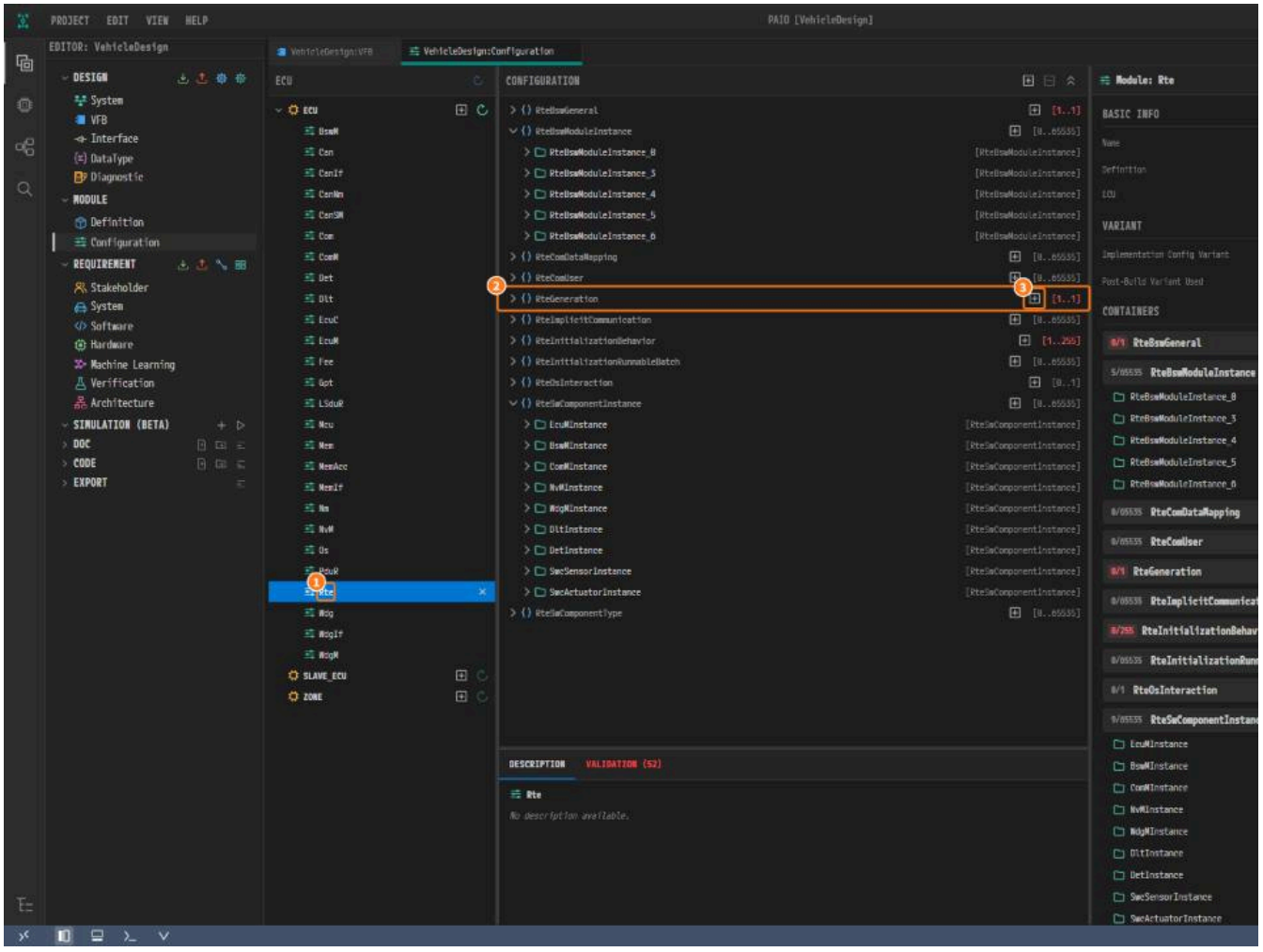


Figure 8-2. ① `Rte`, ② `RteGeneration`, ③ `+`.

2. In `RteCodeVendorId` type `79` and press `Tab`.

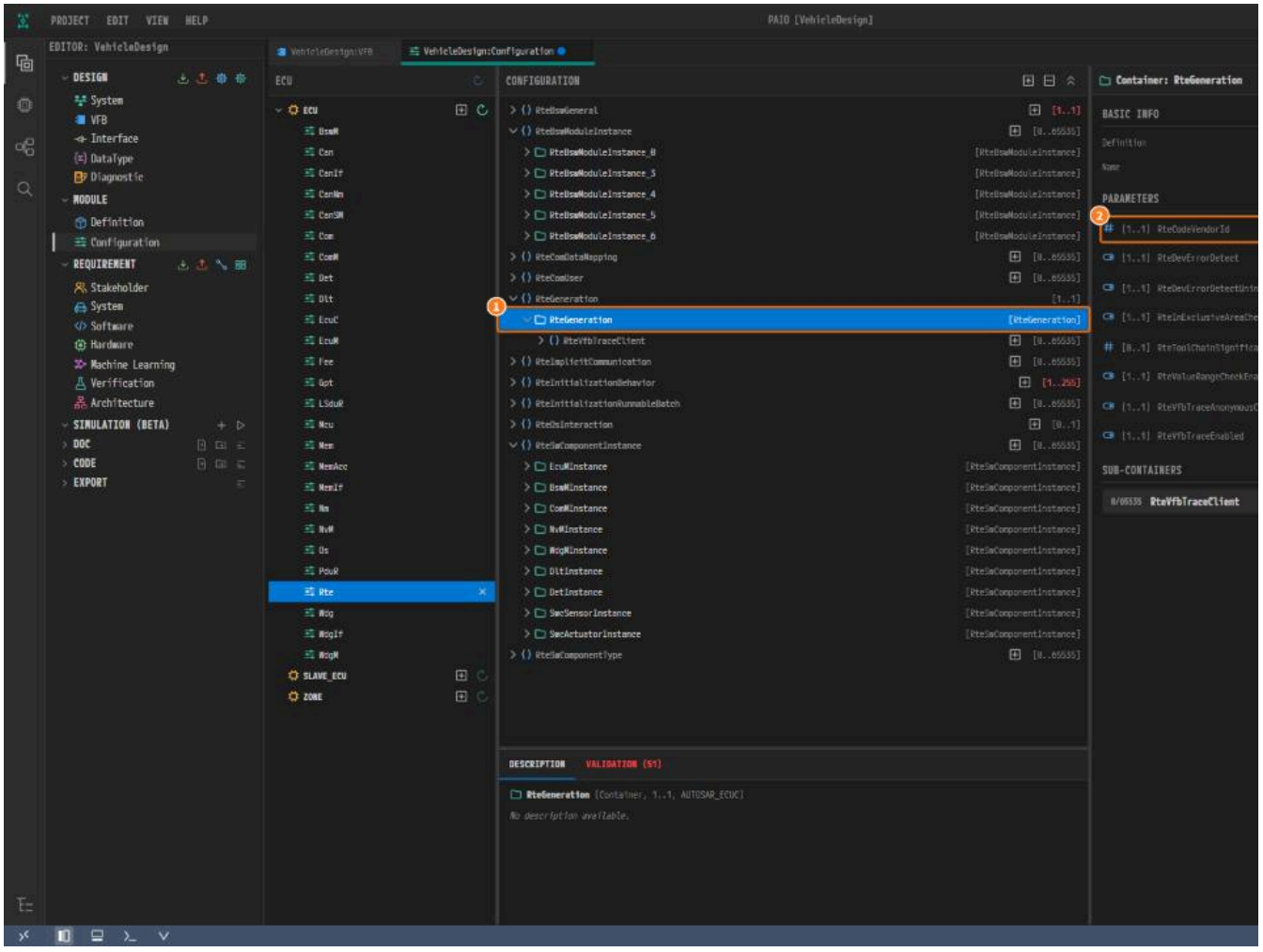


Figure 8-3. ① `RteGeneration`, ② `RteCodeVendorId`.

3. Press "+" on `RteBswGeneral`. It needs no values.
4. Press "+" on `RteInitializationBehavior`. Set `RteInitializationStrategy` to `RTE_INITIALIZATION_STRATEGY_AT_DATA_DECLARATION` and type

SW-C Event Mappings

`RteEventIsMappedToTask` is the trap here: "+" writes `FALSE`, and every mapping that names a task needs `TRUE`.

1. Under `SwcSensorInstance`, press "+" on `RteEventToTaskMapping`.

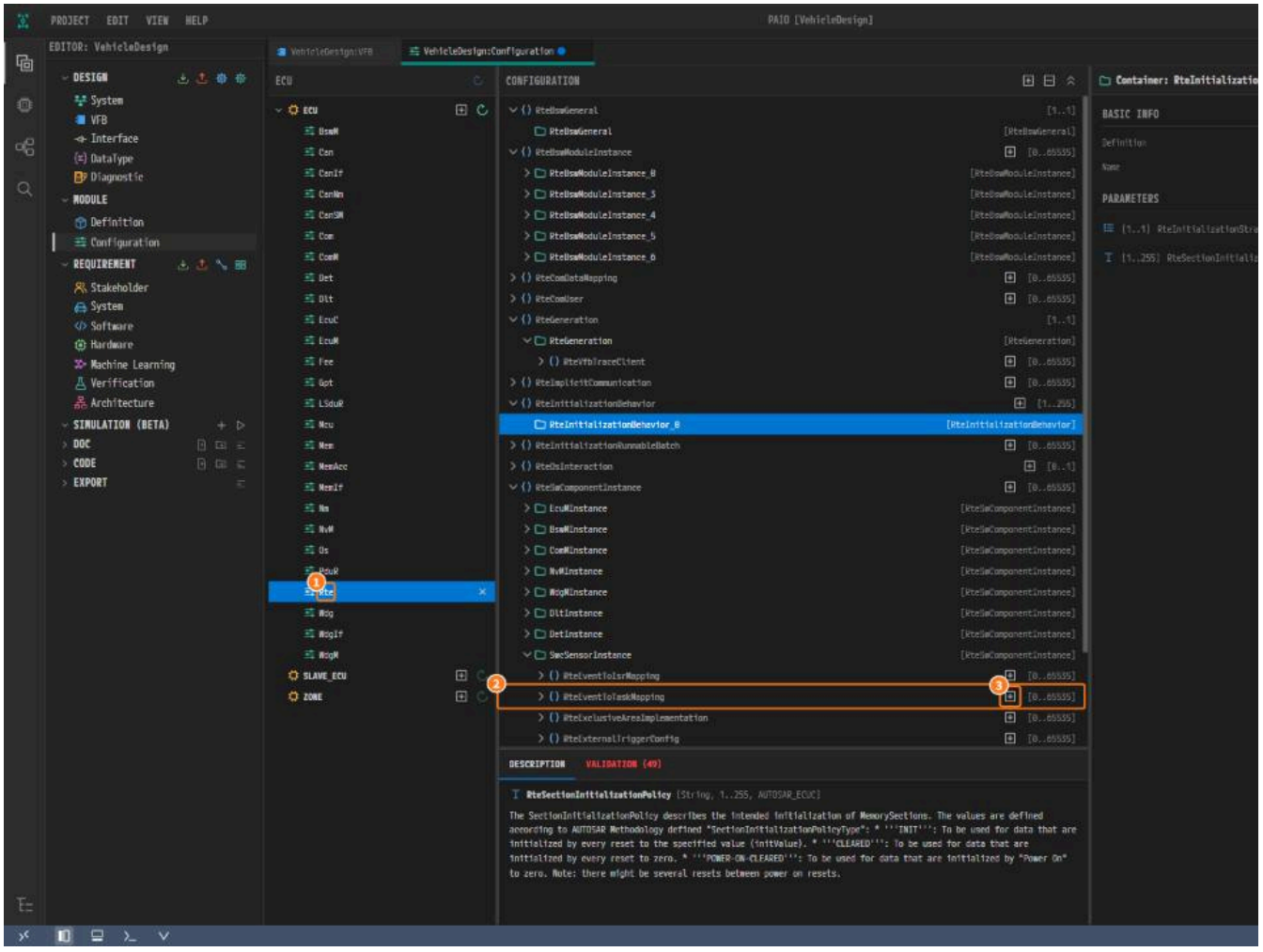


*Figure 8-4. ① `Rte`, ② `RteEventToTaskMapping` under `SwcSensorInstance`, ③ **+**.*

2. Press **TRUE** at `RteEventIsMappedToTask`, then pick the references from the table.

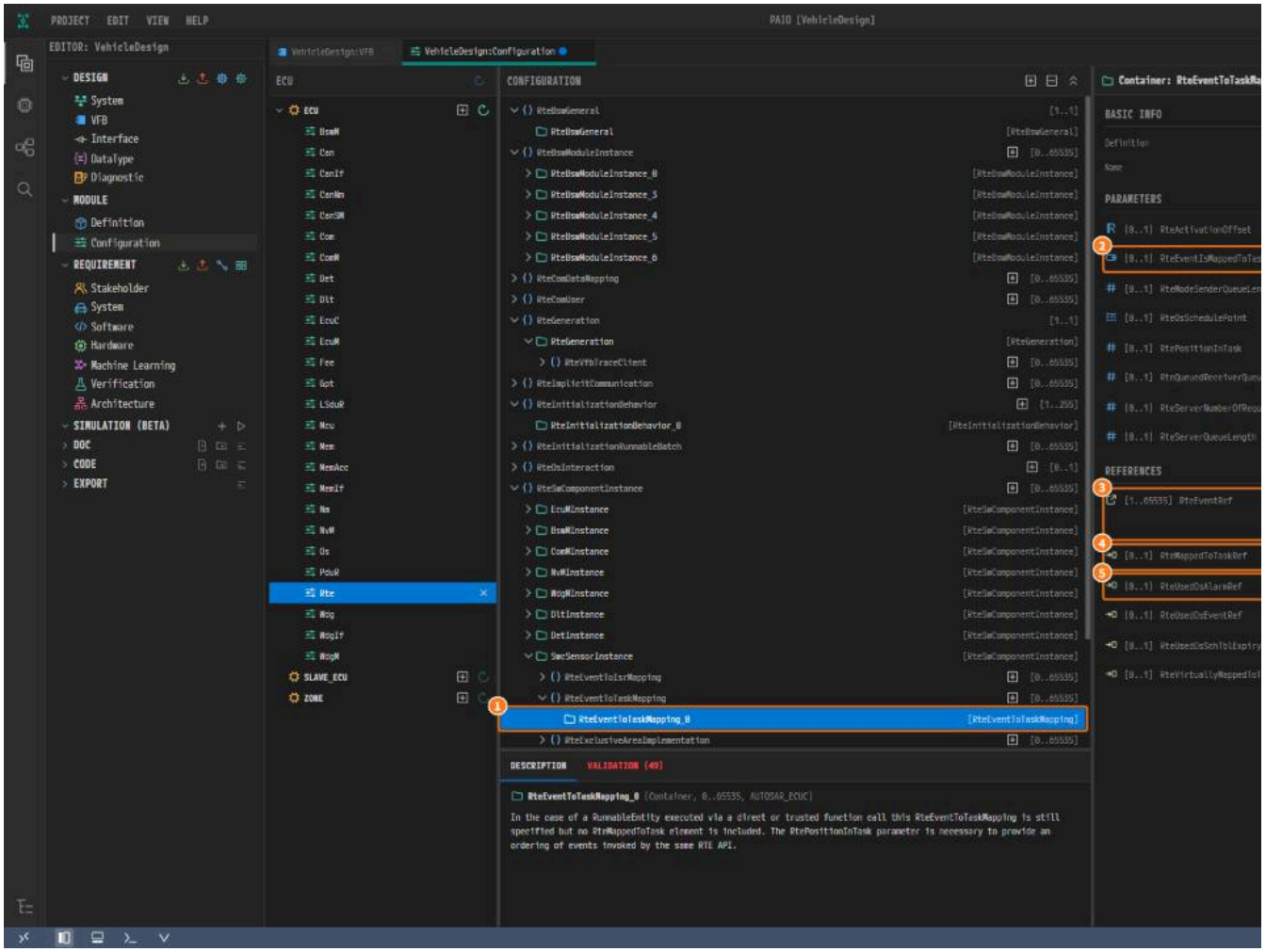


Figure 8-5. ① `RteEventToTaskMapping_0`, ② `RteEventIsMappedToTask`, ③ `RteEventRef`, ④ `RteMappedToTaskRef`, ⑤ `RteUsedOsAlarmRef`.

3. Add the two mappings of `SwcActuatorInstance` the same way.

4. Select `BswMInstance/RteEventToTaskMapping_0` (from the Rte values file). Its `RteEventRef` is already set; fill the rest.

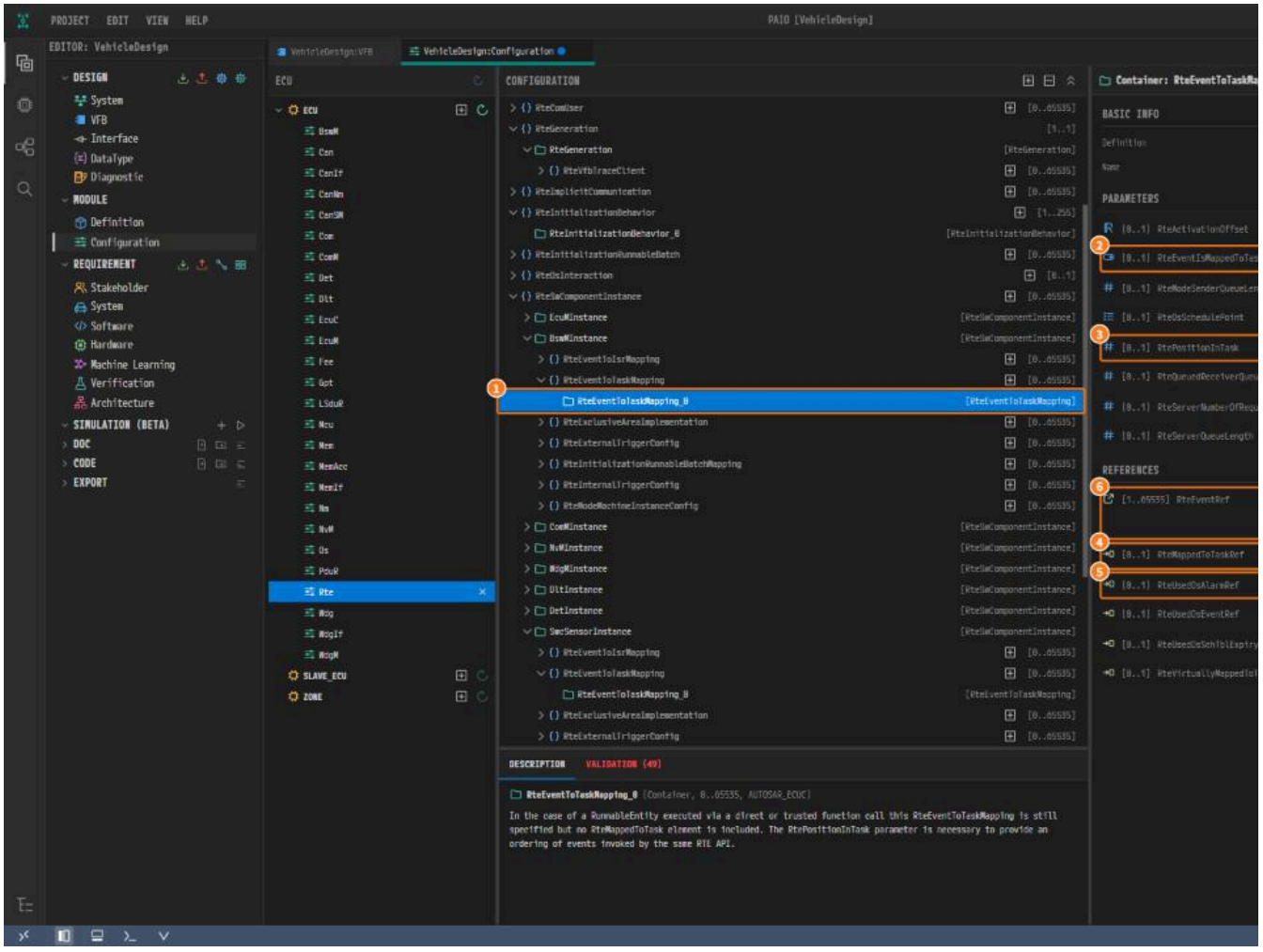


Figure 8-6. ① `RteEventToTaskMapping_0`, ② `RteEventIsMappedToTask`, ③ `RtePositionInTask`, ④ `RteMappedToTaskRef`, ⑤ `RteUsedOsAlarmRef`,

`RteEventIsMappedToTask` is `TRUE` in every row.

Container	`RtePositionInTask`	`RteEventRef`
`SwcSensorInstance/RteEventToTaskMapping_0` ("+")	-	`SwComponentType/Atomic/Application/SwcSensor/InternalBehavior/SensorCyclicTiming`
`SwcActuatorInstance/RteEventToTaskMapping_0` ("+")	`2`	`SwComponentType/Atomic/Application/SwcActuator/InternalBehavior/ActuatorInitEven`
`SwcActuatorInstance/RteEventToTaskMapping_1` ("+")	`1`	`SwComponentType/Atomic/Application/SwcActuator/InternalBehavior/ActuatorOnDataEv`
`BswMInstance/RteEventToTaskMapping_0` (Rte values file)	`0`	(from the Rte values file)

Do not map `ActuatorGetScaleEvent`, the operation-invoked event of `Actuator_GetScale`. The code generator accepts an unmapped event only for a s which you set in Internal Behavior with **CAN BE INVOKED** = **TRUE**.

EcuM Mode Machine

1. Under `EcuMInstance`, press "+" on `RteModeMachineInstanceConfig`.

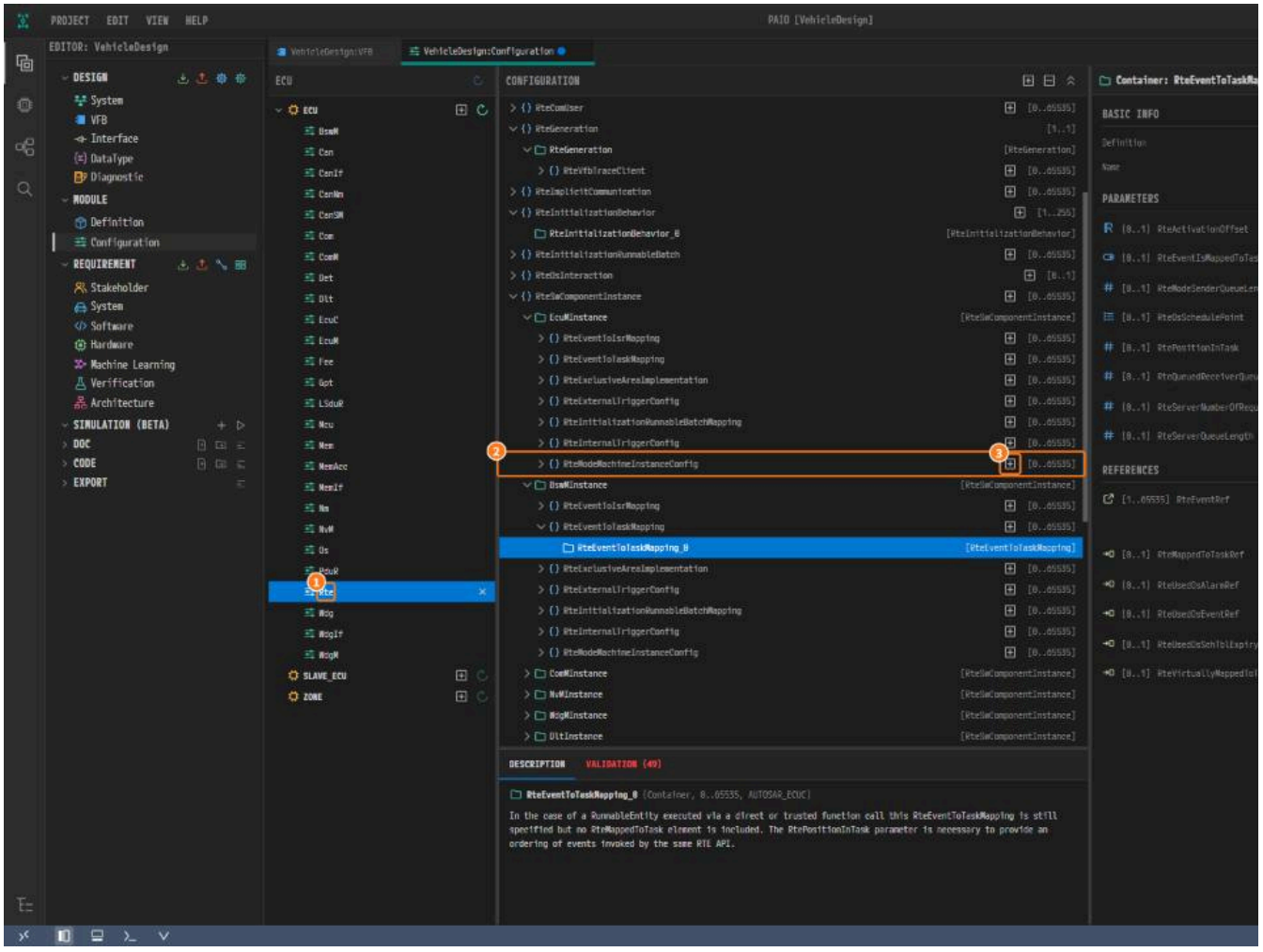


Figure 8-7. ① `Rte`, ② `RteModeMachineInstanceConfig` under `EcuMInstance`, ③ +.

2. In `RteModeMachineQueueLength` type `1` and press `Tab`.

3. In `RteSwcModeManagerRef` pick `PortInterface/ModeSwitch/EcuM_CurrentMode/currentMode`.

4. Select another tree node, then `RteModeMachineInstanceConfig_0` again. Only now the **Context Refs** row appears. Pick `SwComponentType/Atomic/S` **PROVIDED-PORT-PROTOTYPE**).

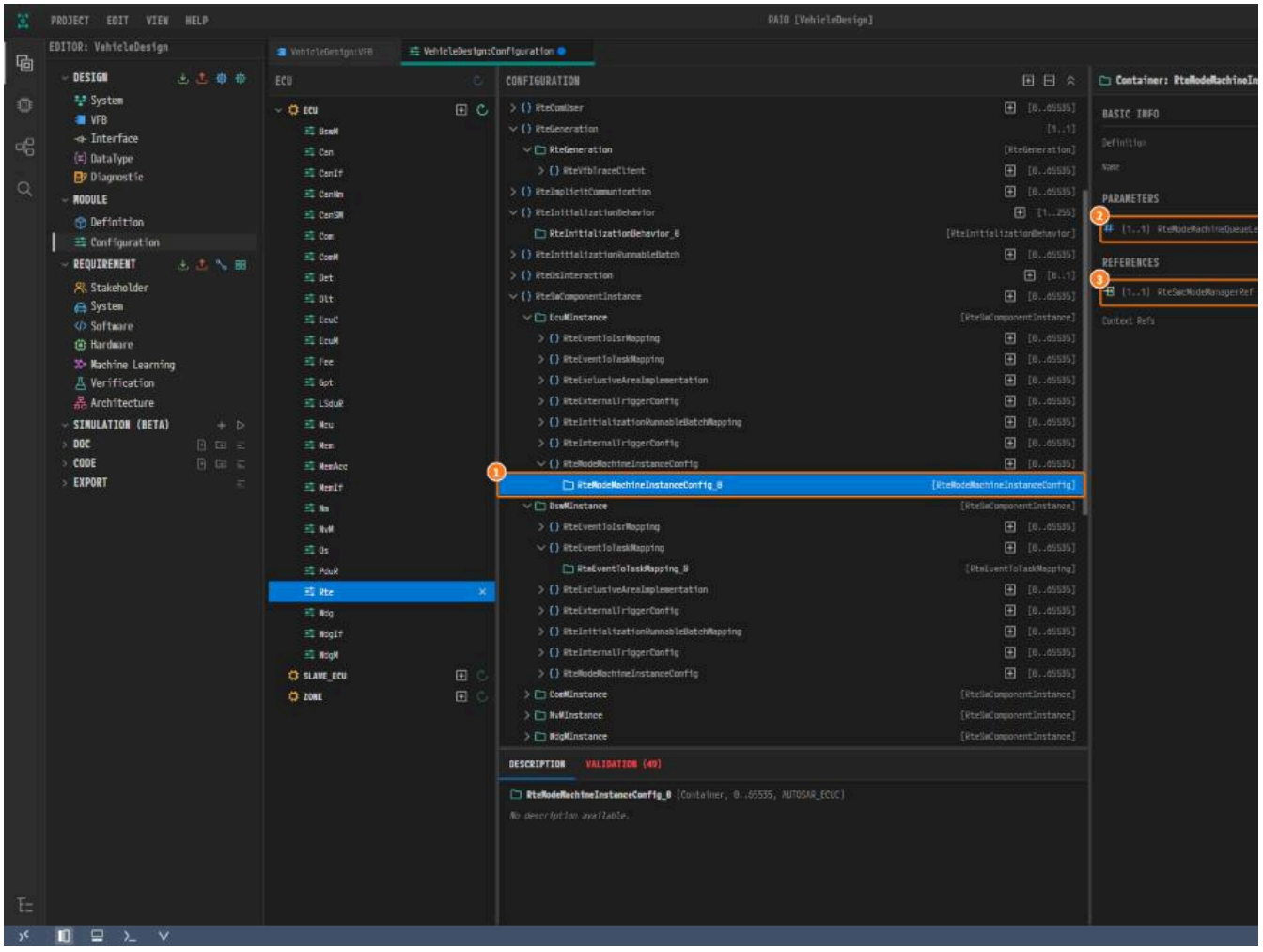


*Figure 8-8. ① `RteModeMachineInstanceConfig_0`, ② `RteModeMachineQueueLength`, ③ `RteSwcModeManagerRef`, ④ **Context Refs**.*

Os Activations

1. Press "+" on `RteOsInteraction`. It needs no values.
2. Under it, press "+" on `RteUsedOsActivation`. In the new `RteUsedOsActivation_0`, enter the values of the table.

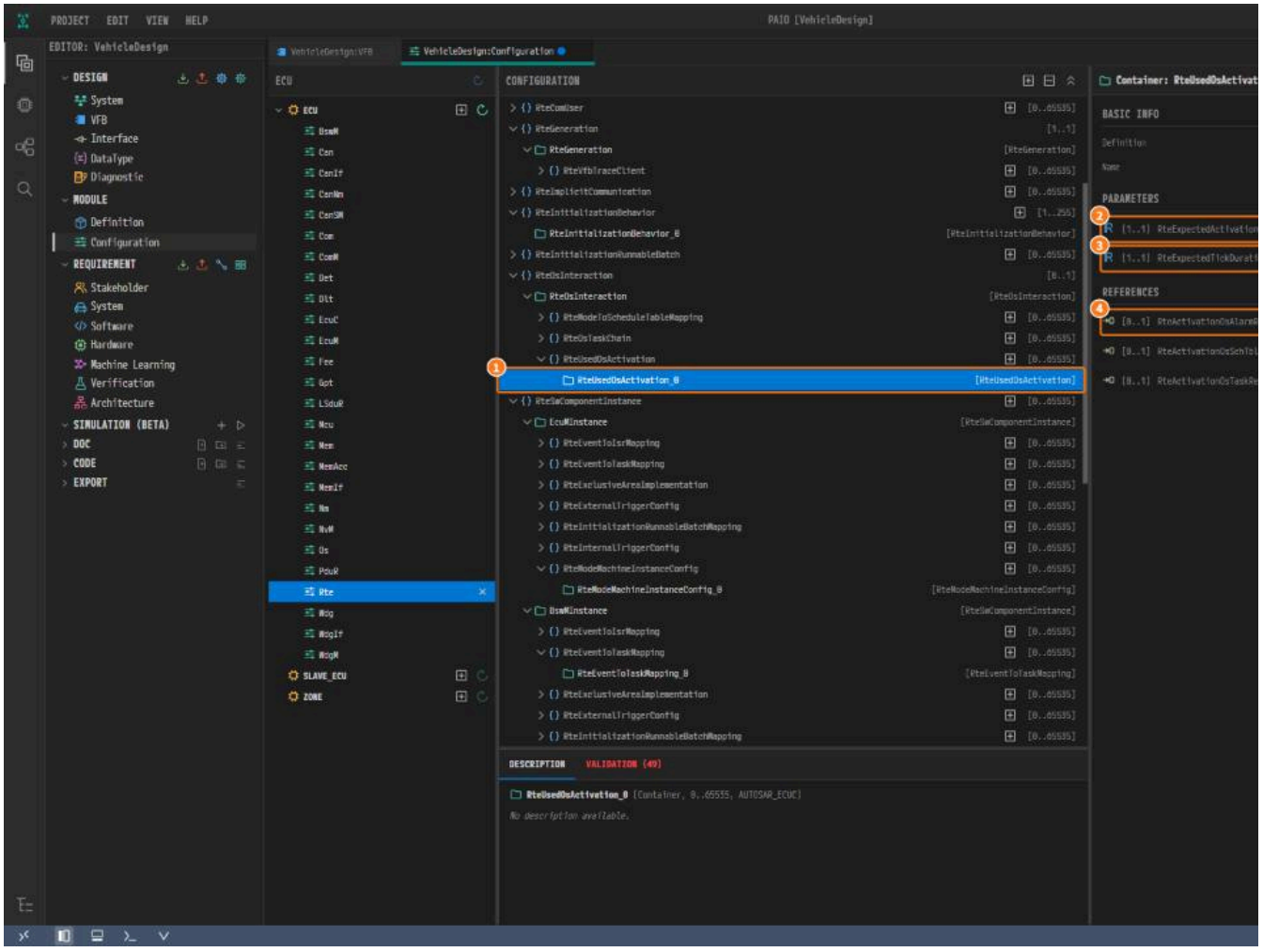


Figure 8-9. ① `RteUsedOsActivation_0`, ② `RteExpectedActivationOffset`, ③ `RteExpectedTickDuration`, ④ `RteActivationOsAlarmRef`.

3. Add `_1` and `_2` the same way from the table.

Container	`RteExpectedActivationOffset`	`RteExpectedTickDuration`	`RteActivationOsAlarmRef`
`RteUsedOsActivation_0`	`0`	`0.01`	`Os/OsAlarm_0`
`RteUsedOsActivation_1`	`0`	`0.01`	`Os/OsAlarm_1`
`RteUsedOsActivation_2`	`0`	`0.01`	`Os/OsAlarm_2`

BSW Module Instances

The Rte values file brought `_0`, `_3`, `_4`, `_5` and `_6`. "+" gives the smallest free number, so add the instances in this order: EcuM's (`_1`), Can's (`_2`), then C

1. Select `RteBswModuleInstance_0` (from the Rte values file). Type `BswModuleDescription/BswM/BswMImplementation` in `RteBswImplementationRef`
2. Select its `RteBswExclusiveAreaImpl_0` (from the Rte values file). Set `RteExclusiveAreaImplMechanism` to `ALL_INTERRUPT_BLOCKING`. `RteBswExcl`

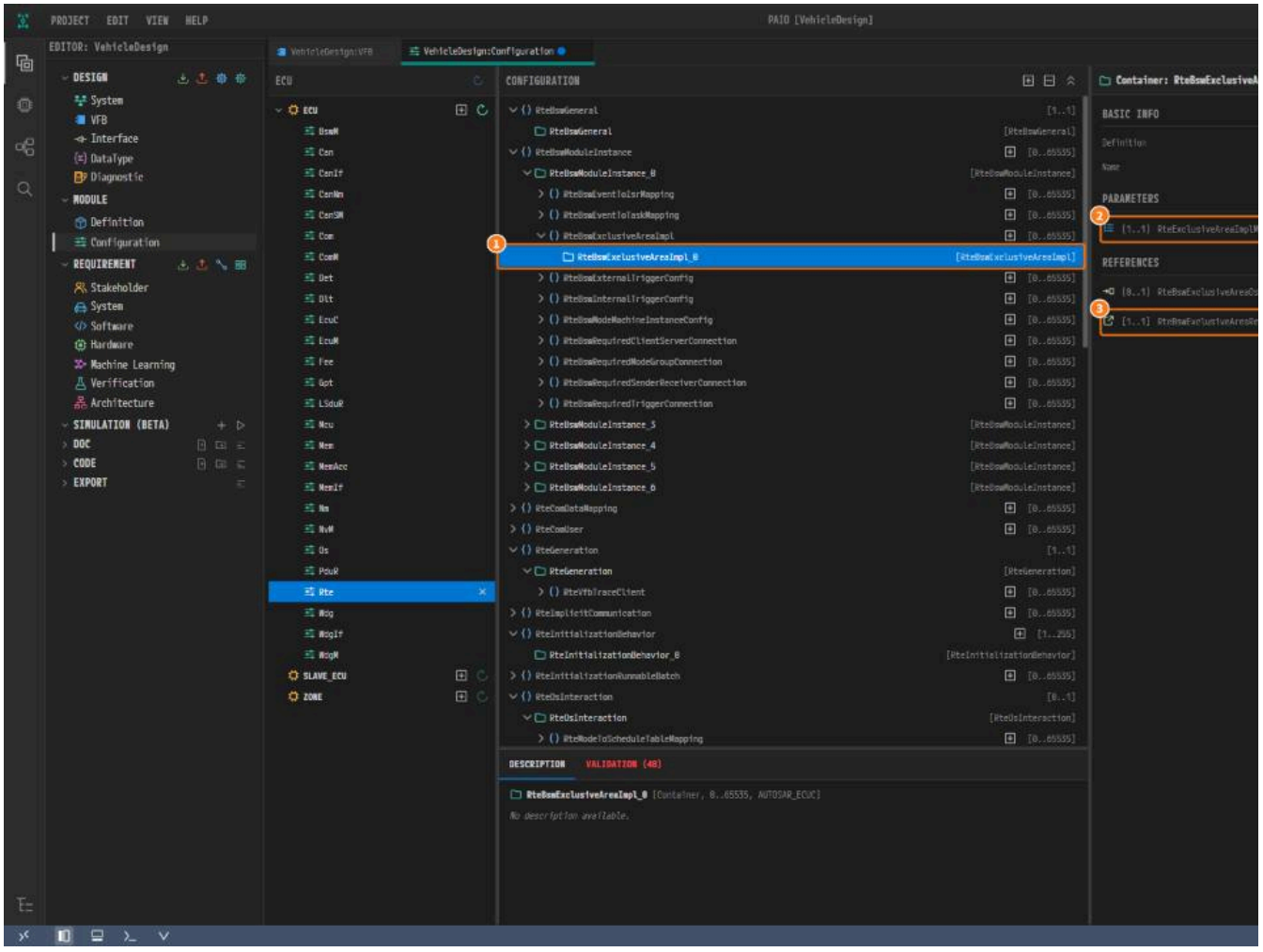


Figure 8-10. ① `RteBswExclusiveAreaImpl_0`, ② `RteExclusiveAreaImplMechanism`, ③ `RteBswExclusiveAreaRef` (from the Rte values file).

3. Complete `_3`, `_4`, `_5` and `_6` the same way from the table.

4. Press "+" on `RteBswModuleInstance`. Type the EcuM path from the table in `RteBswImplementationRef` of `RteBswModuleInstance_1` and press `Tab`

5. Press "+" on `RteBswModuleInstance` again.

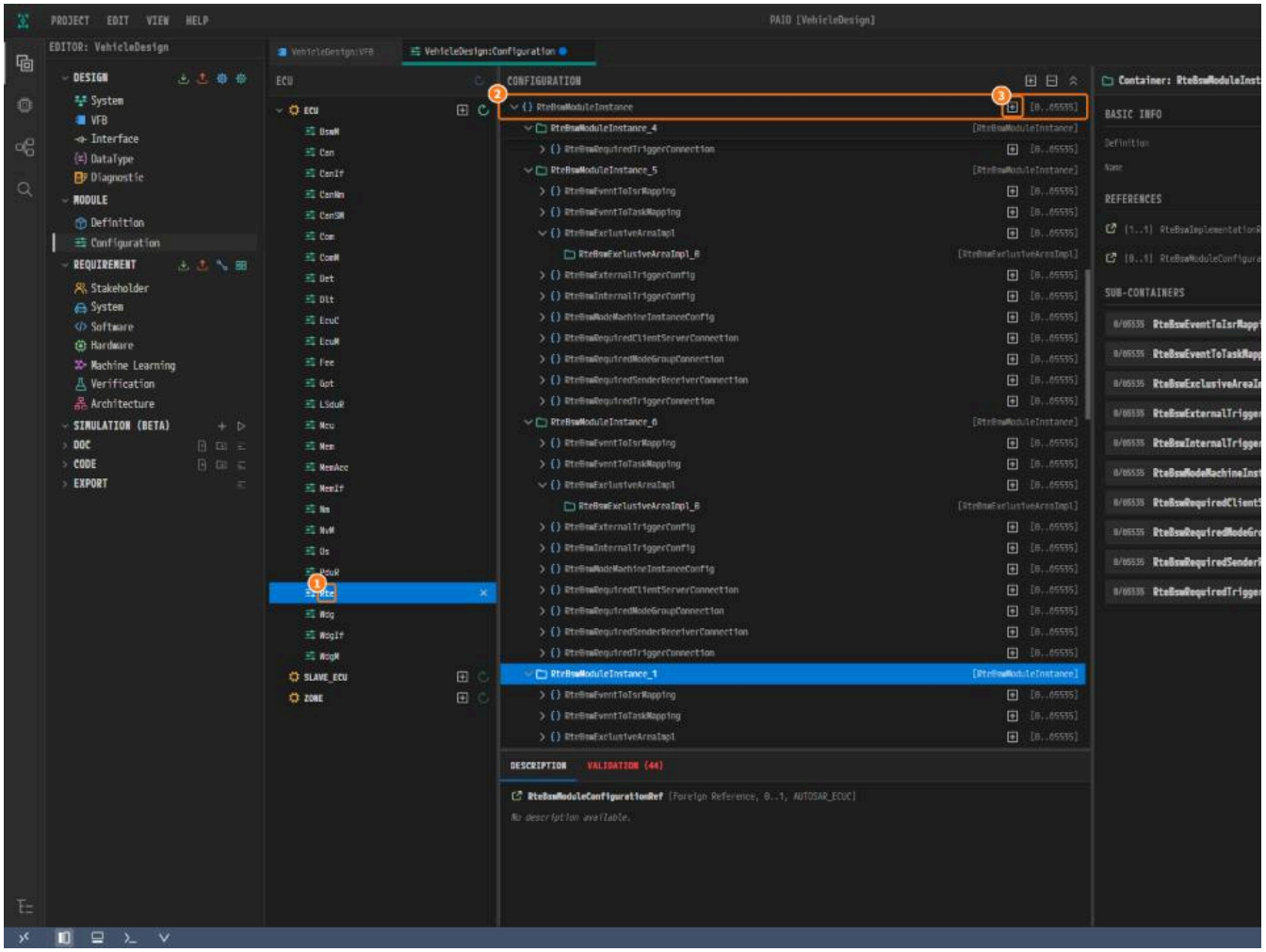


Figure 8-11. ① `Rte`, ② `RteBswModuleInstance`, ③ `+`.

6. Type `BswModuleDescription/Can/CanImplementation` in `RteBswImplementationRef` of `RteBswModuleInstance_2` and press `Tab`.

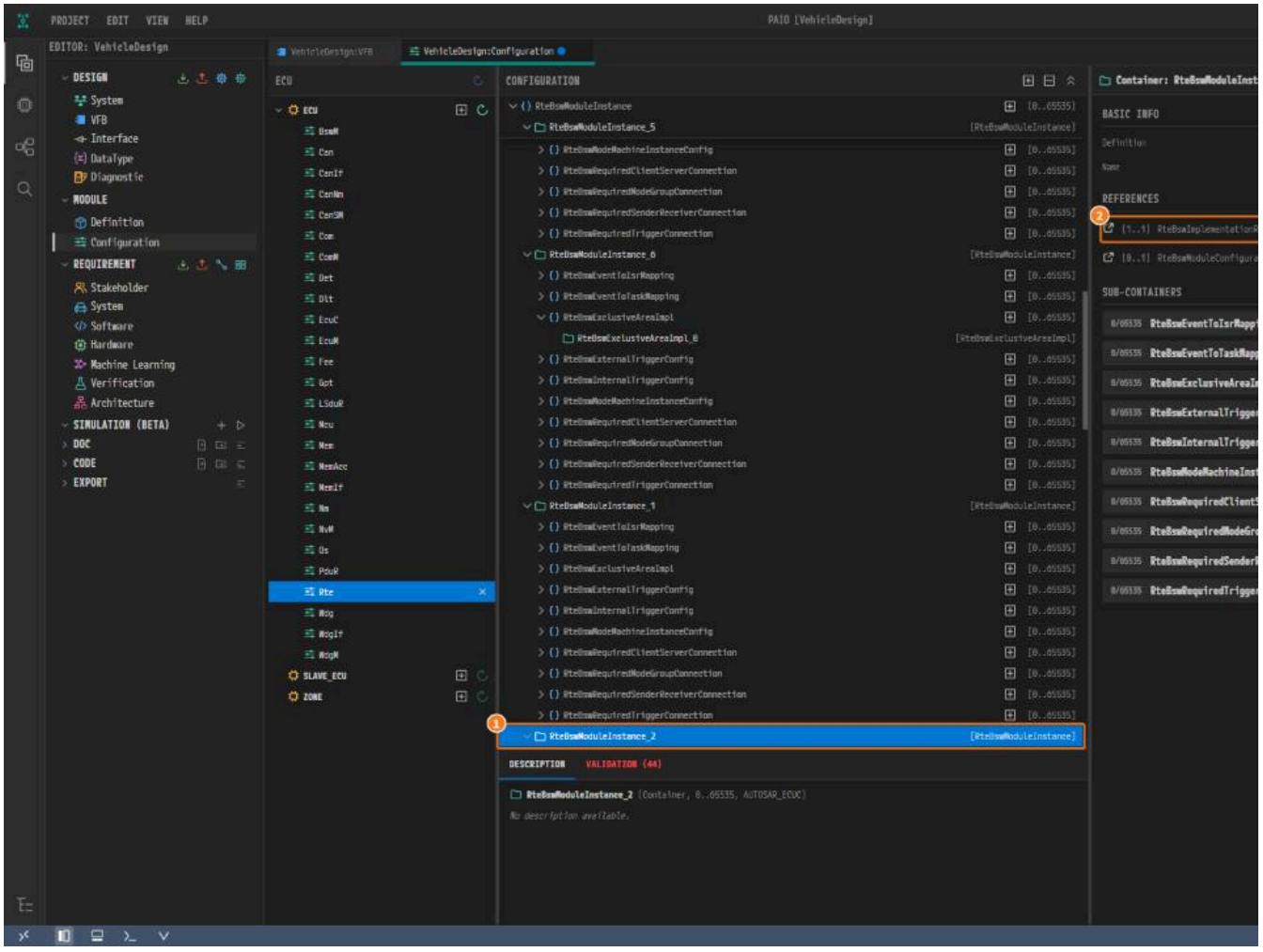


Figure 8-12. ① `RteBswModuleInstance_2`, ② `RteBswImplementationRef`.

7. Press "+" on `RteBswModuleInstance` once more. The editor names it `RteBswModuleInstance_7`, because `_3` to `_6` are taken. Type the CanNm path and press `Tab`.

Instance	Origin	`RteBswImplementationRef` (typed)	`RteExclusiveAreaImplMechanism` of `RteBswExclusiveAreaImpl_`
`RteBswModuleInstance_0`	Rte values file	`BswModuleDescription/BswM/BswMImplementation`	`ALL_INTERRUPT_BLOCKING`
`RteBswModuleInstance_1`	"+"	`BswModuleDescription/EcuM/EcuMImplementation`	-
`RteBswModuleInstance_2`	"+"	`BswModuleDescription/Can/CanImplementation`	-
`RteBswModuleInstance_3`	Rte values file	`BswModuleDescription/CanIf/CanIfImplementation`	`ALL_INTERRUPT_BLOCKING`
`RteBswModuleInstance_4`	Rte values file	`BswModuleDescription/CanSM/CanSMImplementation`	`ALL_INTERRUPT_BLOCKING`
`RteBswModuleInstance_5`	Rte values file	`BswModuleDescription/ComM/ComMImplementation`	`ALL_INTERRUPT_BLOCKING`
`RteBswModuleInstance_6`	Rte values file	`BswModuleDescription/Com/ComImplementation`	`ALL_INTERRUPT_BLOCKING`
`RteBswModuleInstance_7`	"+"	`BswModuleDescription/CanNm/CanNmImplementation`	-

BSW Event Mappings

`RteBswEventIsMappedToTask` has the same trap: "+" writes `FALSE`; press **TRUE**.

1. Under `RteBswModuleInstance_2`, press "+" on `RteBswEventToTaskMapping`.
2. Press **TRUE** at `RteBswEventIsMappedToTask` and type `4` in `RteBswPositionInTask`.

3. Type `BswModuleDescription/Can/Can/CanInternalBehavior/CanMainWriteTimingEvent` in `RteBswEventRef` and press `Tab`.

4. Pick `Os/OsTask_0` in `RteBswMappedToTaskRef` and `Os/OsAlarm_0` in `RteBswUsedOsAlarmRef`.

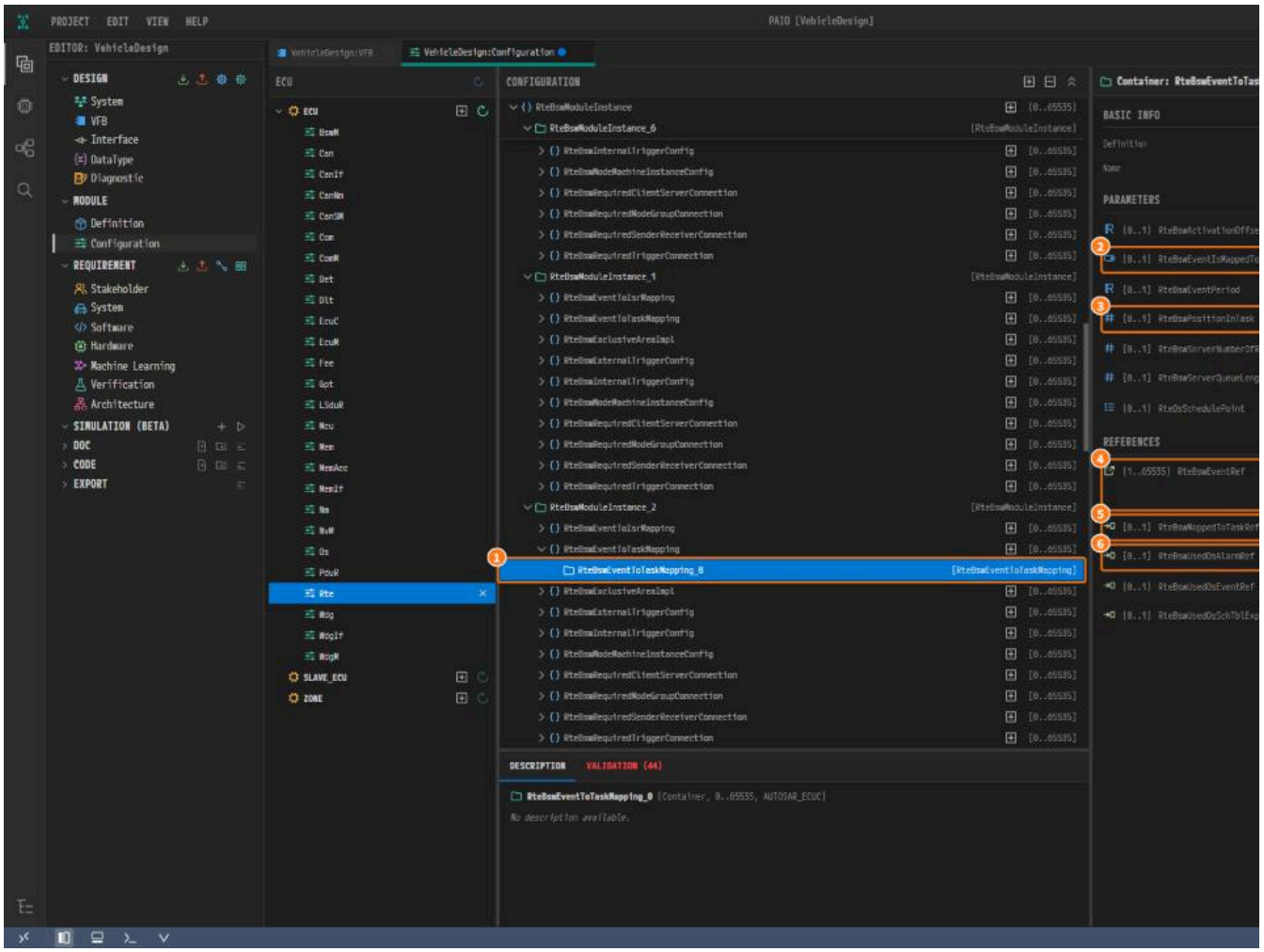


Figure 8-13. ① `RteBswEventToTaskMapping_0`, ② `RteBswEventIsMappedToTask`, ③ `RteBswPositionInTask`, ④ `RteBswEventRef`, ⑤ `RteBswMappedT`

5. Add the other mappings the same way from the table. Every one is added with "+", has `RteBswEventIsMappedToTask` `TRUE`, `RteBswMappedToTaskRe` `Os/OsAlarm_0`.

6. The last one is the main function of `CanNm`: under `RteBswModuleInstance_7`, press "+" on `RteBswEventToTaskMapping`, press **TRUE**, type `13` in `Rte` table in `RteBswEventRef`, and pick the task and alarm.

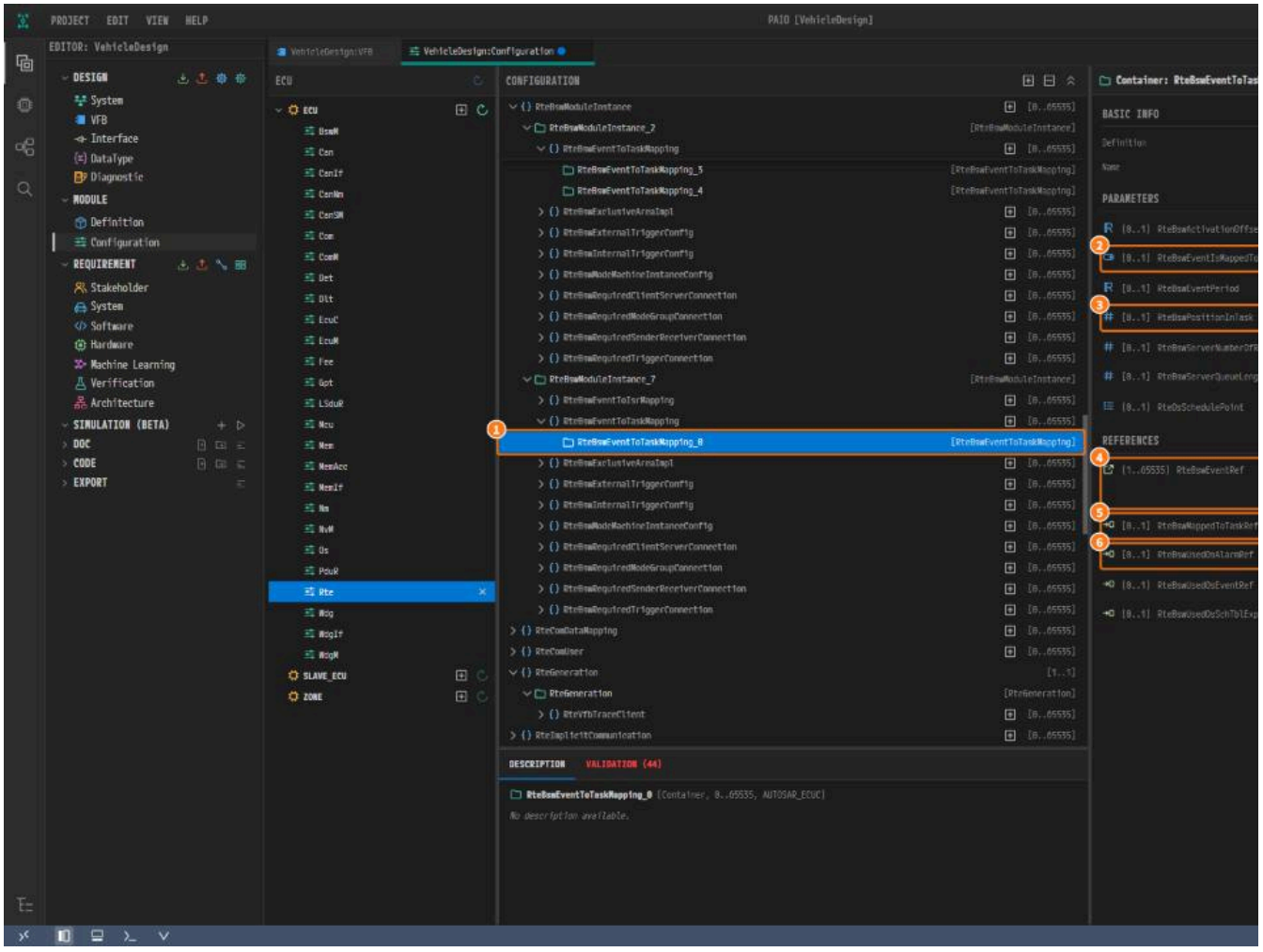


Figure 8-14. ① `RteBswEventToTaskMapping_0`, ② `RteBswEventIsMappedToTask`, ③ `RteBswPositionInTask`, ④ `RteBswEventRef`, ⑤ `RteBswMapped`

7. Save the module with `Ctrl+S`.

Mapping	`RteBswPositionInTask`	`RteBswEventRef` (typed)
`RteBswModuleInstance_0/RteBswEventToTaskMapping_0`	2	`BswModuleDescription/BswM/BswM/BswMInternalBehavior/BswMMainTimingEvent`
`RteBswModuleInstance_1/RteBswEventToTaskMapping_0`	3	`BswModuleDescription/EcuM/EcuM/EcuMInternalBehavior/EcuMMainTimingEvent`
`RteBswModuleInstance_2/RteBswEventToTaskMapping_0`	4	`BswModuleDescription/Can/Can/CanInternalBehavior/CanMainWriteTimingEven`
`RteBswModuleInstance_2/RteBswEventToTaskMapping_1`	5	`BswModuleDescription/Can/Can/CanInternalBehavior/CanMainReadTimingEvent`
`RteBswModuleInstance_2/RteBswEventToTaskMapping_2`	6	`BswModuleDescription/Can/Can/CanInternalBehavior/CanMainModeTimingEvent`
`RteBswModuleInstance_2/RteBswEventToTaskMapping_3`	7	`BswModuleDescription/Can/Can/CanInternalBehavior/CanMainBusOffTimingEve`
`RteBswModuleInstance_2/RteBswEventToTaskMapping_4`	8	`BswModuleDescription/Can/Can/CanInternalBehavior/CanMainWakeupTimingEve`
`RteBswModuleInstance_4/RteBswEventToTaskMapping_0`	9	`BswModuleDescription/CanSM/CanSM/CanSMInternalBehavior/CanSMMainTimingE`
`RteBswModuleInstance_5/RteBswEventToTaskMapping_0`	10	`BswModuleDescription/ComM/ComM/ComMInternalBehavior/ComMMainFunctionTim`
`RteBswModuleInstance_6/RteBswEventToTaskMapping_0`	11	`BswModuleDescription/Com/Com/ComInternalBehavior/ComMainFunctionRxTimin`
`RteBswModuleInstance_6/RteBswEventToTaskMapping_1`	12	`BswModuleDescription/Com/Com/ComInternalBehavior/ComMainFunctionTxTimin`
`RteBswModuleInstance_7/RteBswEventToTaskMapping_0`	13	`BswModuleDescription/CanNm/CanNm/CanNmInternalBehavior/CanNmMainTimingE`

Why Is It Designed This Way?

- **Mappings.** A runnable or a BSW main function runs only when an Rte mapping puts it into an Os task.
- **TRUE on every mapping.** PRTE rejects a mapping that names a task while its flag says `FALSE` (`RTE_EVENT_MAPPING_TASK_FLAG_MISMATCH`, `RTE_BSW_I`
- **Com events named after containers.** `Com` has one main function event per `ComMainFunctionRx` and `ComMainFunctionTx` container, so the event na you add in Communication Configuration.
- **An instance for CanNm.** `CanNm` has a main function, so PRTE requires its BSW module instance (`RTE_BSW_INSTANCE_MISSING` without it). `Nm` has non
- **The Rte values file.** It carries only what the panel cannot take (the BswM poll event and the BSW exclusive areas) and the SW-C instances, which the
- **Positions.** `RteBswPositionInTask` sets the call order of the BSW main functions inside `OsTask_0`.

Checklist

- [ ] Every `RteEventToTaskMapping` and `RteBswEventToTaskMapping` shows **TRUE** at its mapped-to-task flag.
- [ ] `RteBswModuleInstance_0` to `_7` each have an `RteBswImplementationRef`.
- [ ] `RteModeMachineInstanceConfig_0` shows the **Context Refs** row filled.
- [ ] The tab `VehicleDesign:Configuration` carries no modified mark.

Next Step

In Communication Configuration you configure the CAN communication stack, from `EcuC` and the CAN driver to `LSduR`, `PduR` and `Com`.

Communication Configuration

What You Will Do

In the AUTOSAR System Tutorial you connected the node `ECU` to the CAN cluster `CanCluster`, so PRTE Platform Import in chapter 6 brought the modules documents. In this chapter you fill those ten modules by hand, from the bottom up.

- `EcuC` holds the PDUs that several modules share. You create the two network management PDUs `PduNmTx` and `PduNmRx` here.
- `Can` is the driver of the CAN controller and its two hardware objects, one for transmitting and one for receiving.
- `CanIf` is the interface between the driver and the modules above it. It gives each PDU its CAN id.
- `ComM` manages the communication channels of the ECU.
- `CanSM` manages the state of the CAN network and its recovery from bus-off.
- `Nm` is the network management layer that does not depend on the bus type. It hands the channel to `CanNm`.
- `CanNm` is the network management of CAN. It sends and receives the NM PDUs `PduNmTx` and `PduNmRx` of `EcuC`.
- `LSduR` passes the NM PDUs between `CanIf` and `CanNm`. You declare the two modules it talks to and one path for each NM PDU.
- `PduR` routes application PDUs between the communication modules. This design has none, so it gets only its two mandatory containers.
- `Com` packs signals into PDUs and unpacks them. It also gets only its two mandatory containers.

Before You Begin

- Start after finishing Rte Configuration.
- Click **Configuration** under **MODULE** to open the ECU Configuration editor.
- Configure the modules in this order. A module further down picks containers of the modules above it from its lists.

Order	Module	Containers you add	Refers to
1	`EcuC`	6	—
2	`Can`	6	`Mcu` (from PRTE Setup)
3	`CanIf`	12	`Can`, `EcuC`
4	`ComM`	4	—
5	`CanSM`	4	`ComM`, `CanIf`
6	`Nm`	7	`ComM`
7	`CanNm`	4	`ComM`, `EcuC`
8	`LSduR`	10	`CanIf`, `CanNm`, `EcuC`
9	`PduR`	2	—
10	`Com`	2	—

The tables of each section list every value you enter. Time values are in seconds.

> **Note.** The **+** at the right end of a row in the **CONFIGURATION** tree adds a container of that definition and selects it. The new container already holds the values in the tables. Press `Tab` after typing into a text field, and pick a reference from its list. Make one container at a time: **+**, the name if the table gi container. Save each module once with `Ctrl+S` when it is done.

> **Caution.** Do not press the **PRTE Auto Configuration** icon next to `ECU` in the module list. This tutorial configures the BSW by hand, and that icon writes

> **Caution.** Many **TRUE/FALSE** items of `Nm` and `CanNm` have no default value, so **+** leaves them empty and PRTE stops on each empty one. Click **FALSE** (or **T** them. Clicking a button that is already selected clears the value; click it once more in that case.

EcuC Configuration

Configure `EcuC` first. `CanIf`, `CanNm` and `LSduR` pick its PDUs from their lists. `EcuC` itself produces no generated file.

Container	How you add it	Item	Value
`EcucConfigSet`	+ on `EcucConfigSet`		Add only
`EcucPduCollection`	+ on `EcucPduCollection` under `EcucConfigSet`	**PduIdTypeEnum**	`UINT16`
		PduLengthTypeEnum	`UINT32`
`PduNmTx`	+ on `Pdu` under `EcucPduCollection`	**Name**	`PduNmTx`
		PduLength	`8`
`PduNmRx`	+ on `Pdu` under `EcucPduCollection`	**Name**	`PduNmRx`
		PduLength	`8`
`EcucHardware`	+ on `EcucHardware`		Add only
`Core0`	+ on `EcucCoreDefinition` under `EcucHardware`	**Name**	`Core0`
		EcucCoreId	`0`

1. Click `EcuC` in the module list. Add `EcucConfigSet` with its **+**, then `EcucPduCollection` under it, and pick its two values from the table.
2. Under `EcucPduCollection`, click the **+** on the `Pdu` row.

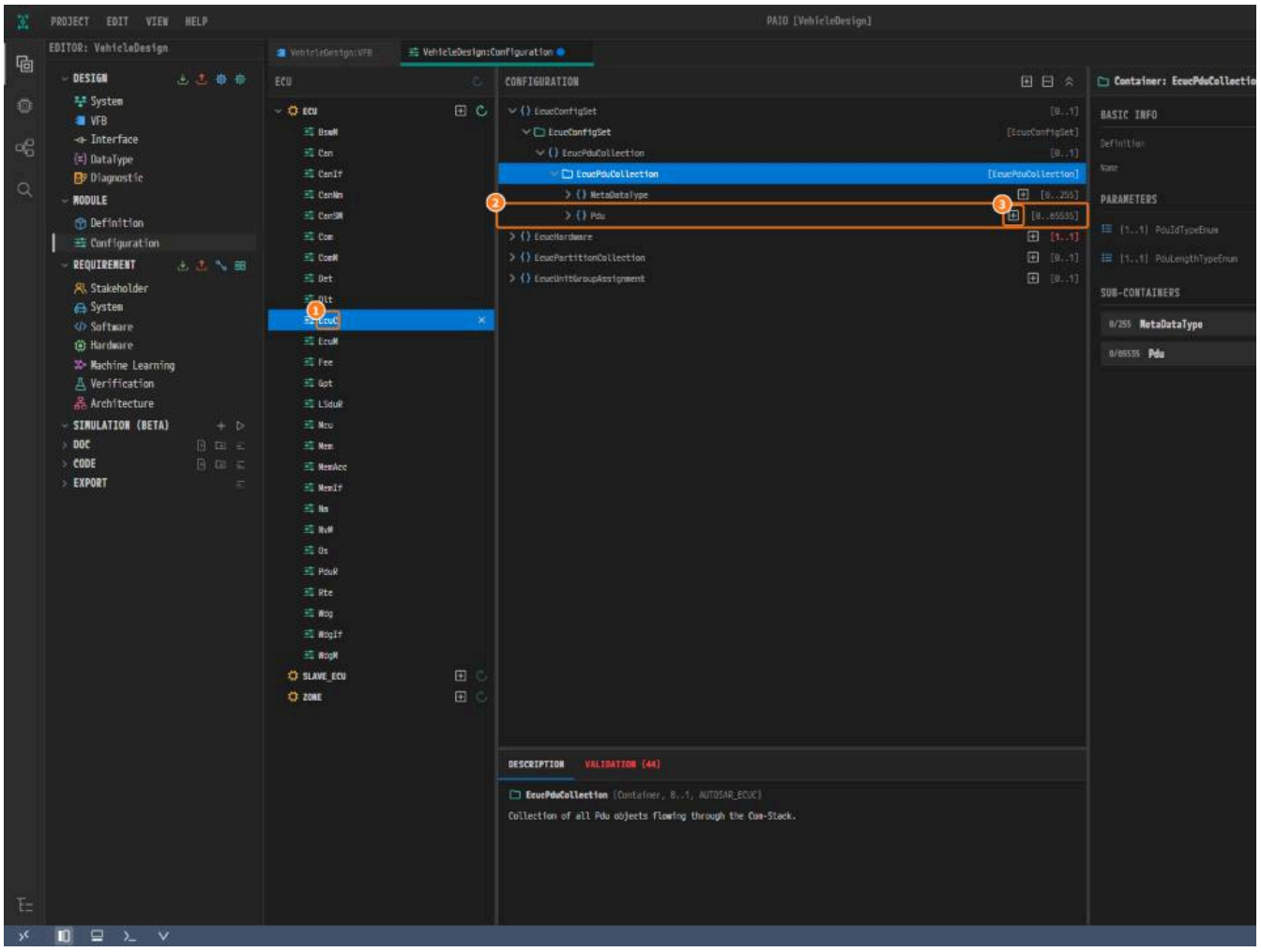


*Figure 9-1. ① `EcuC`, ② `Pdu`, ③ **+**.*

3. The editor adds `Pdu_0` and selects it. Type `PduNmTx` into **Name** and press `Tab`. Type `8` into **PduLength** and press `Tab`.

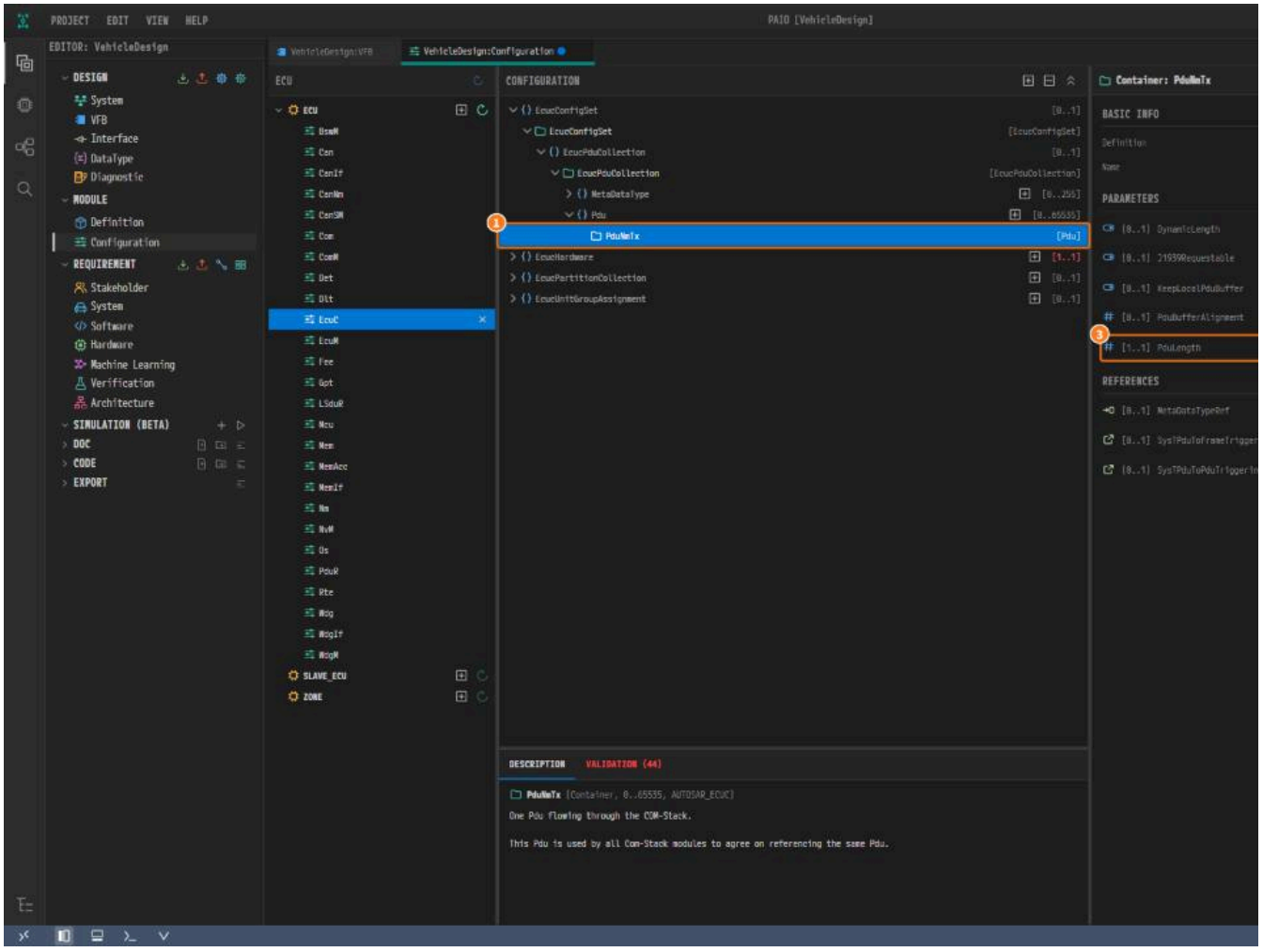


*Figure 9-2. ① `PduNmTx`, ② **Name**, ③ **PduLength**.*

4. Create `PduNmRx` the same way. The **+** gives `Pdu_0` again, because that name is free, and you rename it to `PduNmRx`.

5. Add `EcucHardware` with its **+**. Under it, click the **+** on the `EcucCoreDefinition` row, rename the new `EcucCoreDefinition_0` to `Core0` and type `0` it holds anything. Save with `Ctrl+S`.

Can Configuration

Container	How you add it	Item	
`CanGeneral`	+ on `CanGeneral`	**CanGlobalTimeSupport**	**FALSE**
		CanIndex	`0`
		CanMainFunctionModePeriod	`0.01`
		CanMultiplexedTransmission	**FALSE**
		CanTimeoutDuration	`0.001`
`CanConfigSet`	+ on `CanConfigSet`		Add only
`CanController_0`	+ on `CanController` under `CanConfigSet`	**CanBusoffProcessing**	`POLLING`
		CanControllerActivation	**TRUE**
		CanControllerBaseAddress	`0`
		CanControllerId	`0`
		CanRxProcessing	`POLLING`
		CanTxProcessing	`POLLING`
		CanWakeupProcessing	`POLLING`
		CanWakeupSupport	**FALSE**
		CanControllerDefaultBaudrate	`Can/CanConfigSe`
		CanCpuClockRef	`Mcu/McuModuleCo`
`CanControllerBaudrateConfig_0`	+ on `CanControllerBaudrateConfig` under `CanController_0`	**CanControllerBaudRate**	`500`
		CanControllerPropSeg, **CanControllerSeg1**, **CanControllerSeg2**, **CanControllerSyncJumpWidth**	`0`
`CanHardwareObject_0`	+ on `CanHardwareObject` under `CanConfigSet`	**CanHandleType**	`BASIC`
		CanIdType	`STANDARD`
		CanObjectId	`0`
		CanObjectType	`TRANSMIT`
		CanControllerRef	`Can/CanConfigSe`
`CanHardwareObject_1`	+ on `CanHardwareObject` under `CanConfigSet`	**CanHandleType**	`BASIC`
		CanIdType	`STANDARD`
		CanObjectId	`1`
		CanObjectType	`RECEIVE`
		CanControllerRef	`Can/CanConfigSe`

1. Click `Can` in the module list. In the **CONFIGURATION** tree, click the **+** on the `CanGeneral` row.

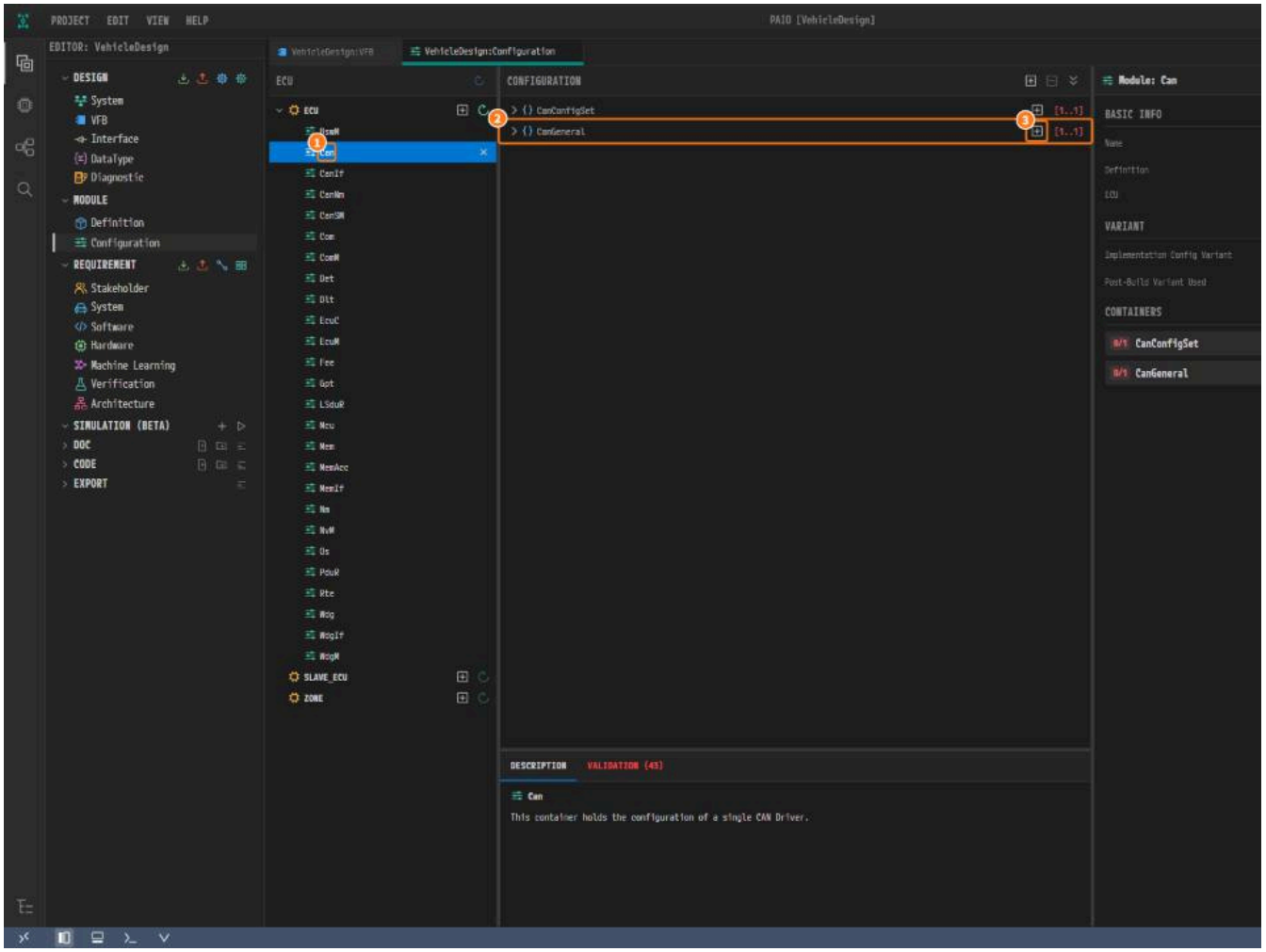


Figure 9-3. ① `Can`, ② `CanGeneral`, ③ +.

2. Enter the five values of `CanGeneral` from the table.

> **Caution.** Clicking a **TRUE** or **FALSE** button that is already selected clears the value, and the row gets a red border. Click the same button once more

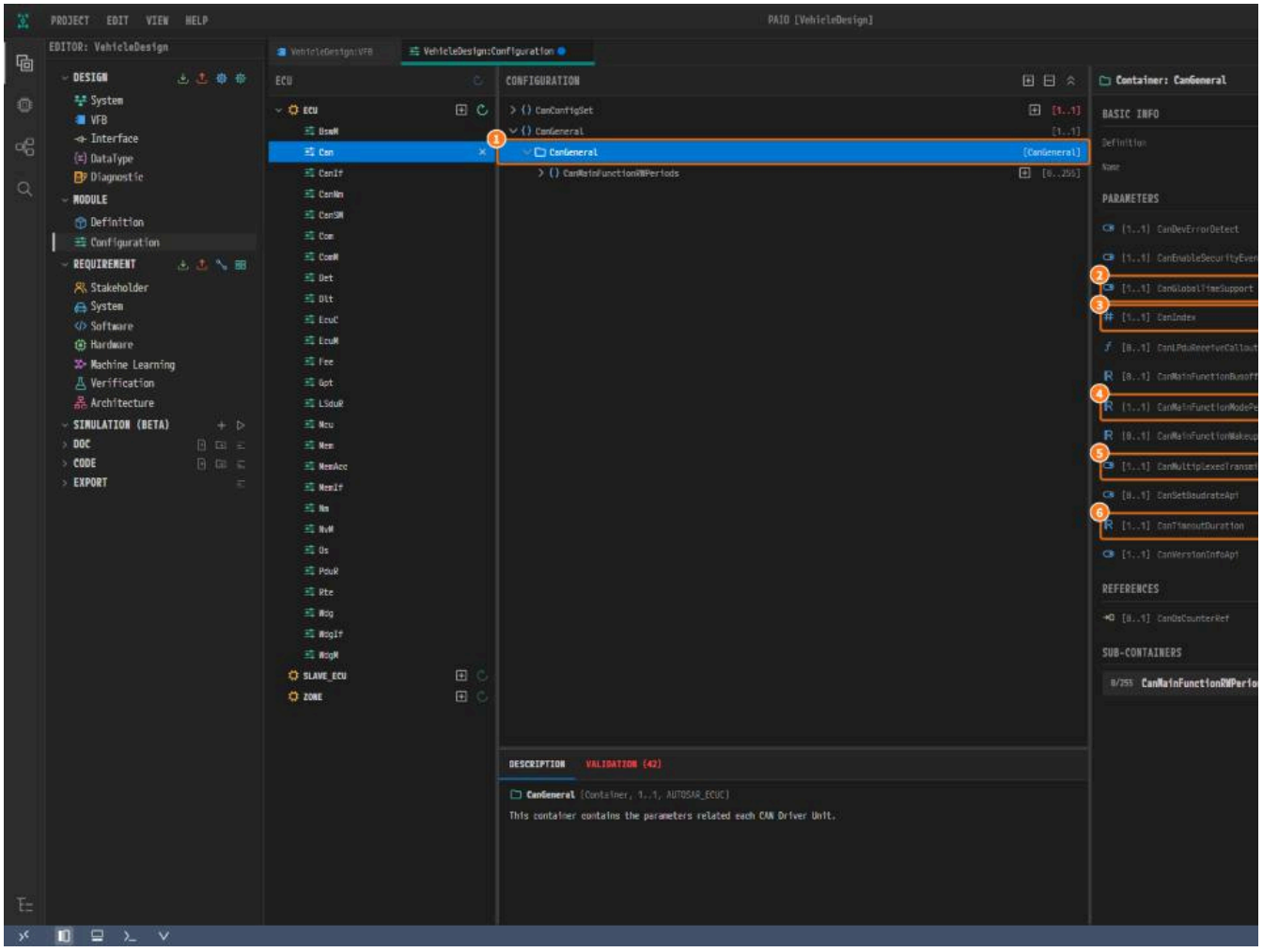


*Figure 9-4. ① `CanGeneral`, ② **CanGlobalTimeSupport**, ③ **CanIndex**, ④ **CanMainFunctionModePeriod**, ⑤ **CanMultiplexedTransmission**, ⑥ **CanTimeoutI***

3. Click the **+** on the `CanConfigSet` row. Under it, click the **+** on the `CanController` row, and under the new `CanController_0`, the **+** on the `CanContr` this baud rate configuration, so it must exist first.

4. Click `CanController_0` and enter its ten values from the table. **CanControllerDefaultBaudrate** picks the `CanControllerBaudrateConfig_0` of step

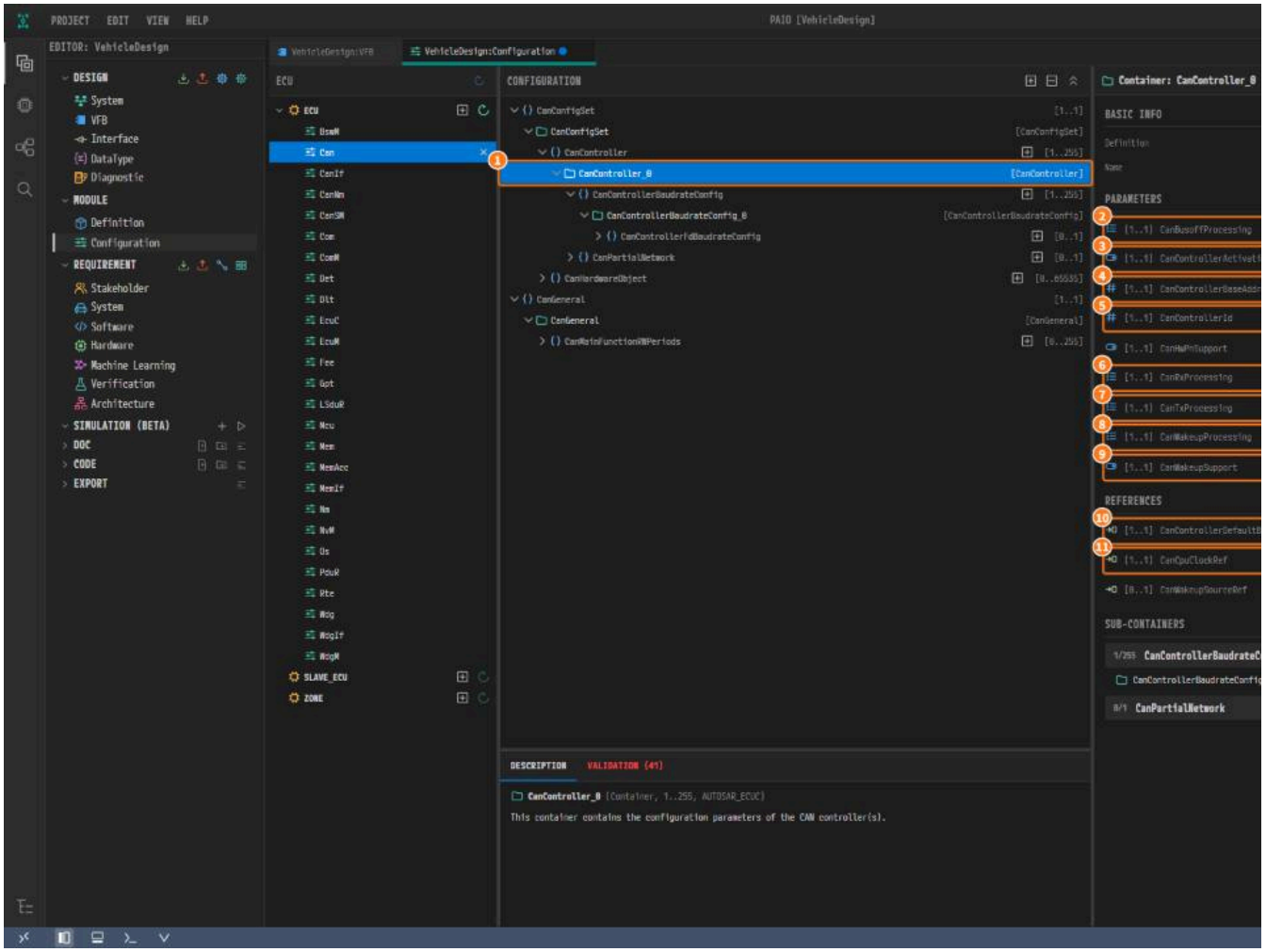


*Figure 9-5. ① `CanController_0`, ② **CanBusoffProcessing**, ③ **CanControllerActivation**, ④ **CanControllerBaseAddress**, ⑤ **CanControllerId**, ⑥ **CanRxPro**
CanWakeupProcessing, ⑨ **CanWakeupSupport**, ⑩ **CanControllerDefaultBaudrate**, ⑪ **CanCpuClockRef**.*

5. Click `CanControllerBaudrateConfig_0` and enter its five values.

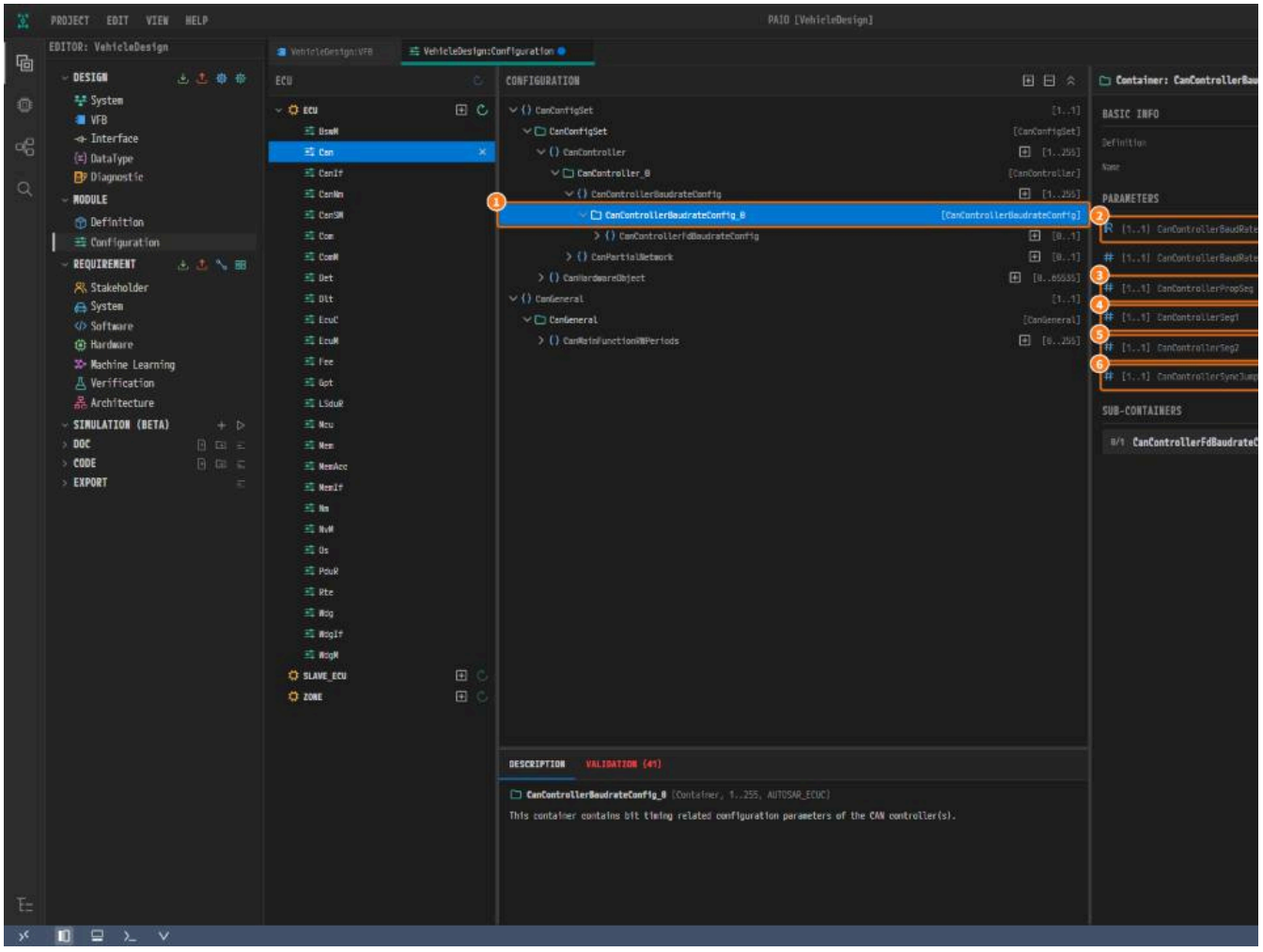


*Figure 9-6. ① `CanControllerBaudrateConfig_0`, ② **CanControllerBaudRate**, ③ **CanControllerPropSeg**, ④ **CanControllerSeg1**, ⑤ **CanControllerSeg2**, ⑥*

6. Under `CanConfigSet`, click the **+** on the `CanHardwareObject` row.

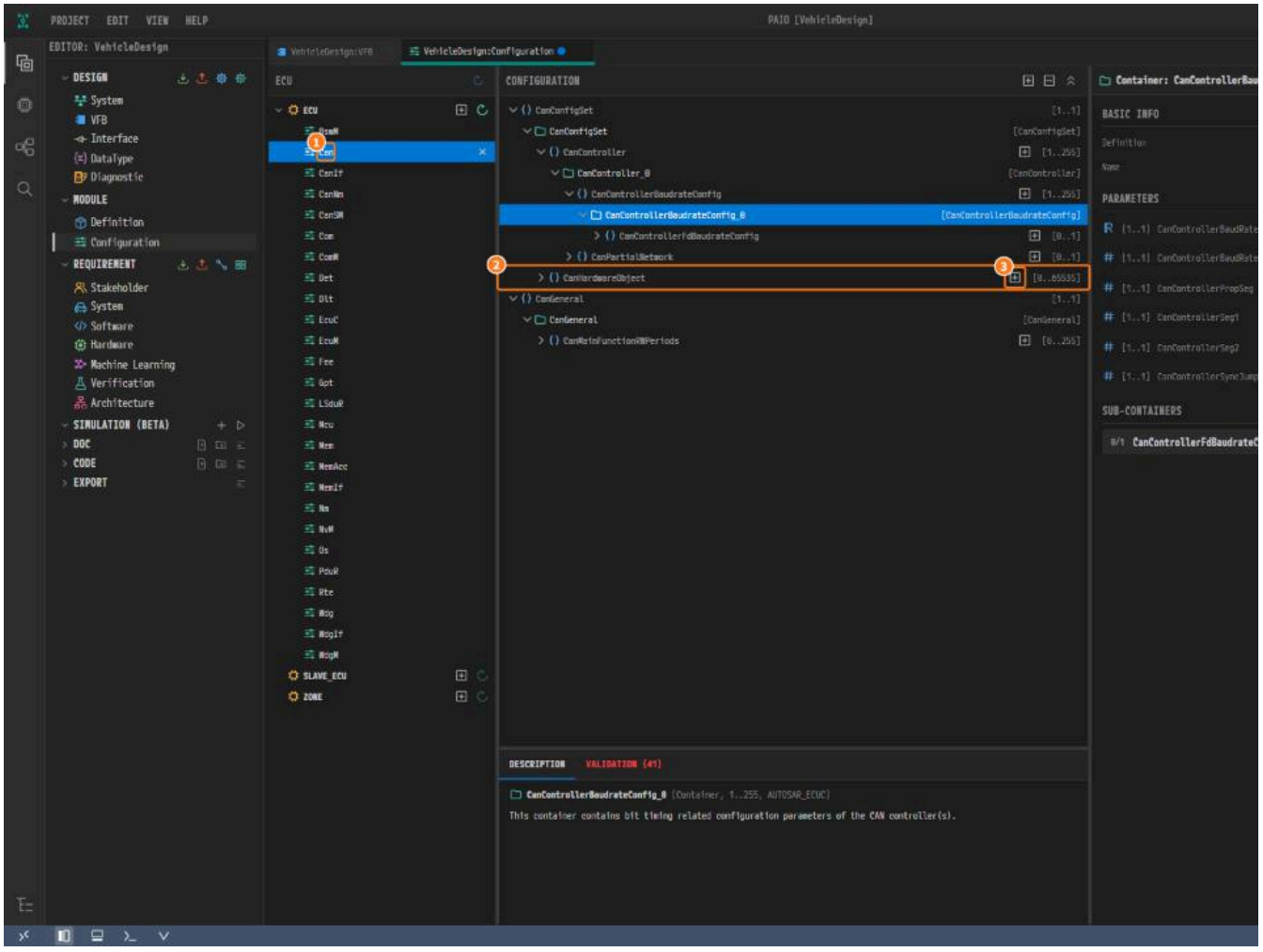


Figure 9-7. ① `Can`, ② `CanHardwareObject`, ③ +.

7. Enter the five values of `CanHardwareObject_0` from the table. This is the transmit object.

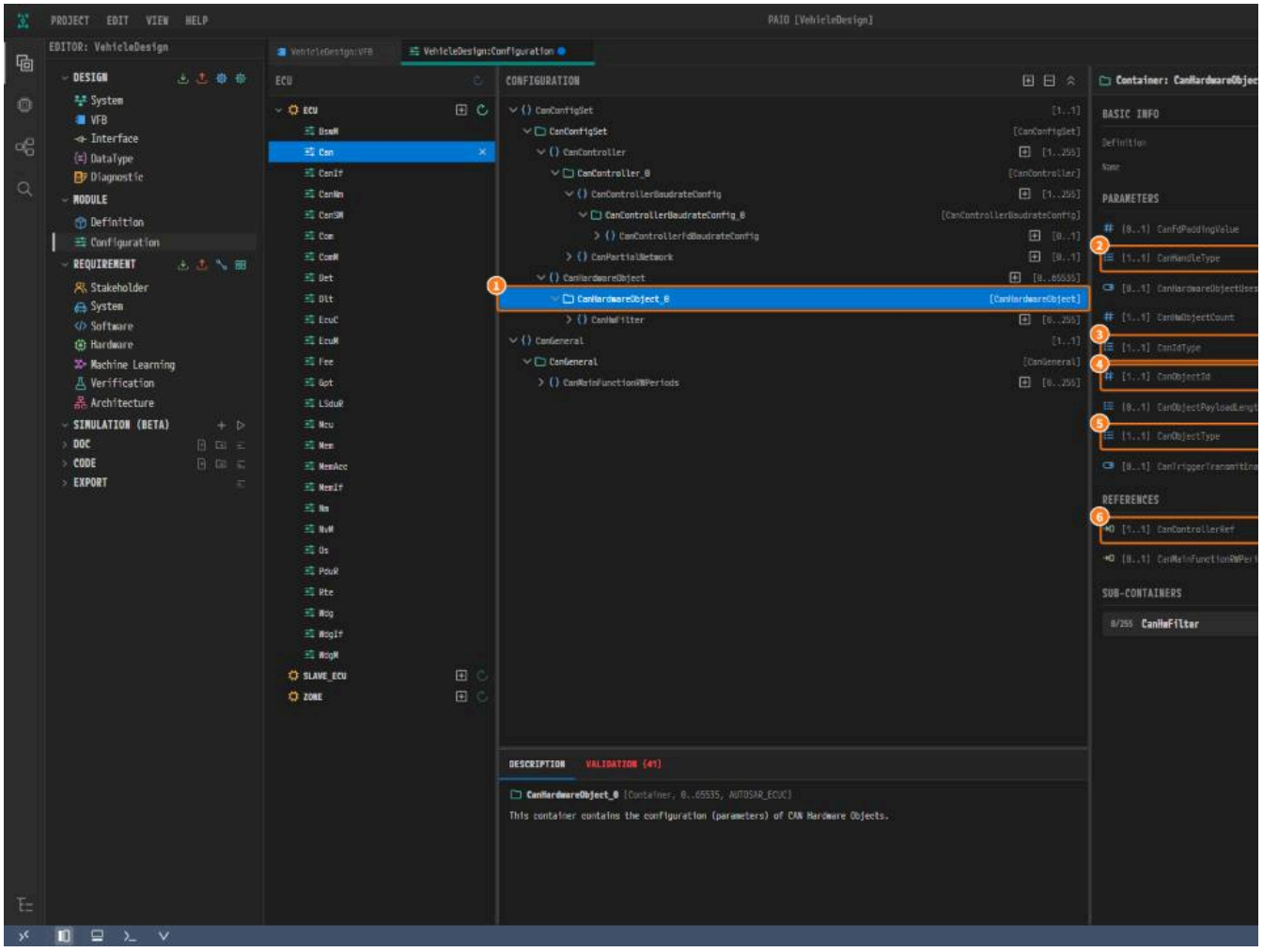


*Figure 9-8. ① `CanHardwareObject_0`, ② **CanHandleType**, ③ **CanIdType**, ④ **CanObjectId**, ⑤ **CanObjectType**, ⑥ **CanControllerRef**.*

8. Create the receive object `CanHardwareObject_1` the same way from the table. Save with `Ctrl+S`.

CanIf Configuration

Container	How you add it	Item	Value
`CanIfPublicCfg`	+ on `CanIfPublicCfg`	**CanIfGlobalTimeSupport**	**FALSE**
		CanIfPublicHandleTypeEnum	`UINT16`
		CanIfPublicTxConfirmPollingSupport	**TRUE**
		CanIfPublicTxBuffering	**TRUE**
`CanIfPrivateCfg`	+ on `CanIfPrivateCfg`	**CanIfPrivateSoftwareFilterType**	`LINEAR`
`CanIfDispatchCfg`	+ on `CanIfDispatchCfg`	**CanIfDispatchUserCtrlBusOffUL**	`CAN_SM`
		CanIfDispatchUserCtrlModeIndicationUL	`CAN_SM`
`CanIfInitCfg`	+ on `CanIfInitCfg`	**CanIfInitCfgSet**	`CanIf_Config`
`CanIfInitHohCfg_0`	+ on `CanIfInitHohCfg` under `CanIfInitCfg`		Add only
`CanIfCtrlDrvCfg_0`	+ on `CanIfCtrlDrvCfg`	**CanIfCtrlDrvNameRef**	`Can/CanGeneral`
		CanIfCtrlDrvInitHohConfigRef	`CanIf/CanIfInitCfg/CanIfInitHohCfg_0`
`CanIfCtrlCfg_0`	+ on `CanIfCtrlCfg` under `CanIfCtrlDrvCfg_0`	**CanIfCtrlId**	`0`
		CanIfCtrlCanCtrlRef	`Can/CanConfigSet/CanController_0`
`CanIfHthCfg_0`	+ on `CanIfHthCfg` under `CanIfInitHohCfg_0`	**CanIfHthCanCtrlIdRef**	`CanIf/CanIfCtrlDrvCfg_0/CanIfCtrlCfg_0`
		CanIfHthIdSymRef	`Can/CanConfigSet/CanHardwareObject_0`
`CanIfHrhCfg_0`	+ on `CanIfHrhCfg` under `CanIfInitHohCfg_0`	**CanIfHrhSoftwareFilter**	**TRUE**
		CanIfHrhCanCtrlIdRef	`CanIf/CanIfCtrlDrvCfg_0/CanIfCtrlCfg_0`
		CanIfHrhIdSymRef	`Can/CanConfigSet/CanHardwareObject_1`
`CanIfBufferCfg_0`	+ on `CanIfBufferCfg` under `CanIfInitCfg`	**CanIfBufferSize**	`1`
		CanIfBufferHthRef	`CanIf/CanIfInitCfg/CanIfInitHohCfg_0/CanIfHth`
`CanIfTxPduCfg_0`	+ on `CanIfTxPduCfg` under `CanIfInitCfg`	**CanIfTxPduCanId**	`1281`
		CanIfTxPduId	`0`
		CanIfTxPduType	`STATIC`
		CanIfTxPduBufferRef	`CanIf/CanIfInitCfg/CanIfBufferCfg_0`
		CanIfTxPduRef	`EcuC/EcucConfigSet/EcucPduCollection/PduNmTx`
`CanIfRxPduCfg_0`	+ on `CanIfRxPduCfg` under `CanIfInitCfg`	**CanIfRxPduCanId**	`1282`
		CanIfRxPduDataLength	`8`
		CanIfRxPduId	`0`
		CanIfRxPduHrhIdRef	`CanIf/CanIfInitCfg/CanIfInitHohCfg_0/CanIfHrh`
		CanIfRxPduRef	`EcuC/EcucConfigSet/EcucPduCollection/PduNmRx`

1. Click `CanIf` in the module list. Add `CanIfPublicCfg` with its **+** and enter its four values. **CanIfPublicTxBuffering** must be **TRUE**: the transmit PDU use buffering PRTE stops with `CANIF_BUFFERING_GATE_INCONSISTENT`.

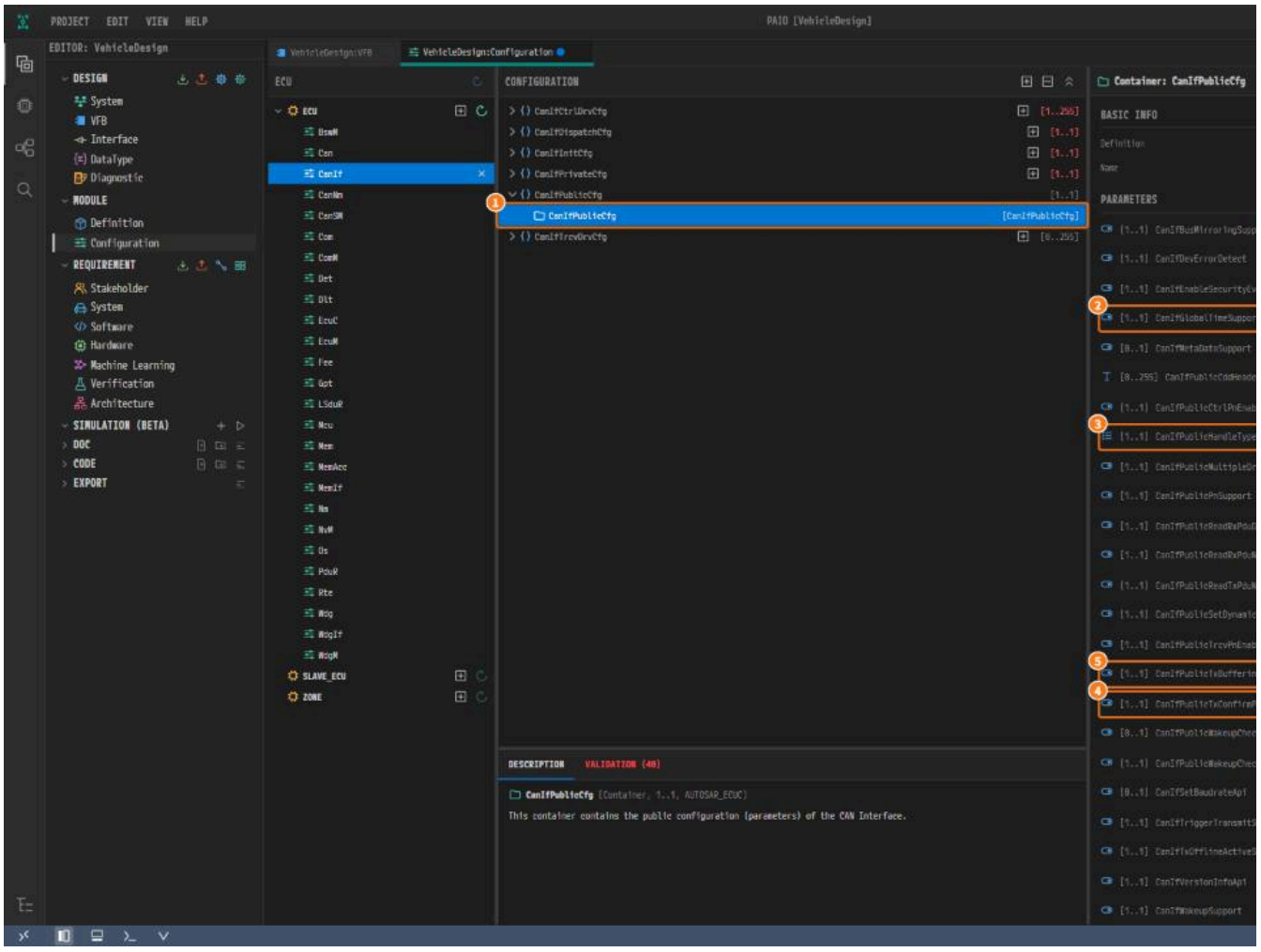


*Figure 9-9. ① `CanIfPublicCfg`, ② **CanIfGlobalTimeSupport**, ③ **CanIfPublicHandleTypeEnum**, ④ **CanIfPublicTxConfirmPollingSupport**, ⑤ **CanIfPublicT***

2. Add `CanIfPrivateCfg` and pick `LINEAR`. Then click the **+** on the `CanIfDispatchCfg` row.

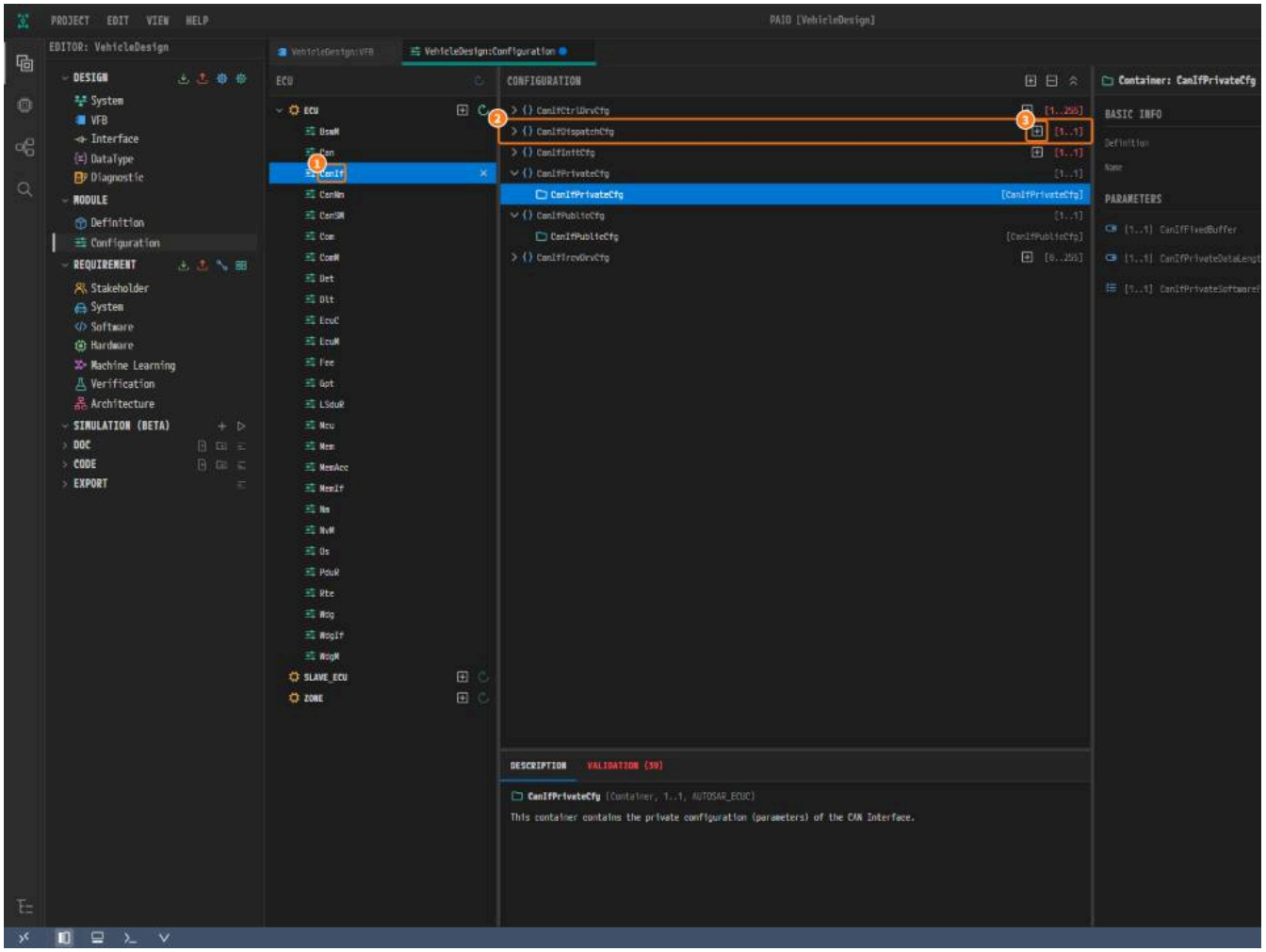


Figure 9-10. ① `CanIf`, ② `CanIfDispatchCfg`, ③ +.

3. Pick `CAN_SM` in both lists. `CanSM` then receives the bus-off and mode reports of the controller.

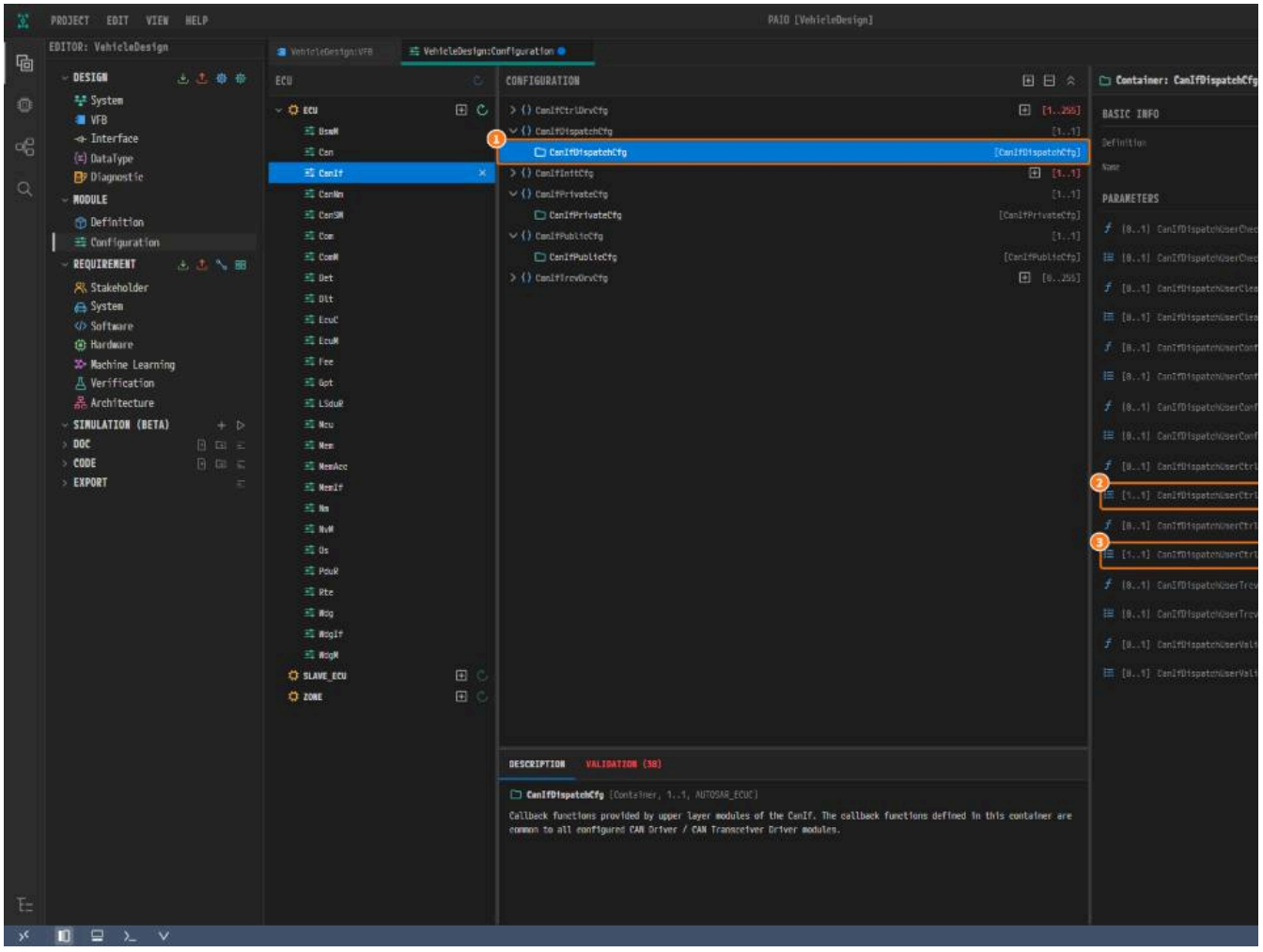


*Figure 9-11. ① `CanIfDispatchCfg`, ② **CanIfDispatchUserCtrlBusOffUL**, ③ **CanIfDispatchUserCtrlModeIndicationUL**.*

4. Add `CanIfInitCfg`, type `CanIf_Config` into **CanIfInitCfgSet**, and add `CanIfInitHohCfg_0` under it.

5. Add `CanIfCtrlDrvCfg_0` and pick its two references from the table.

6. Under `CanIfCtrlDrvCfg_0`, add `CanIfCtrlCfg_0`, type `0` into **CanIfCtrlId** and pick `Can/CanConfigSet/CanController_0` in **CanIfCtrlCanCtrlRef**.

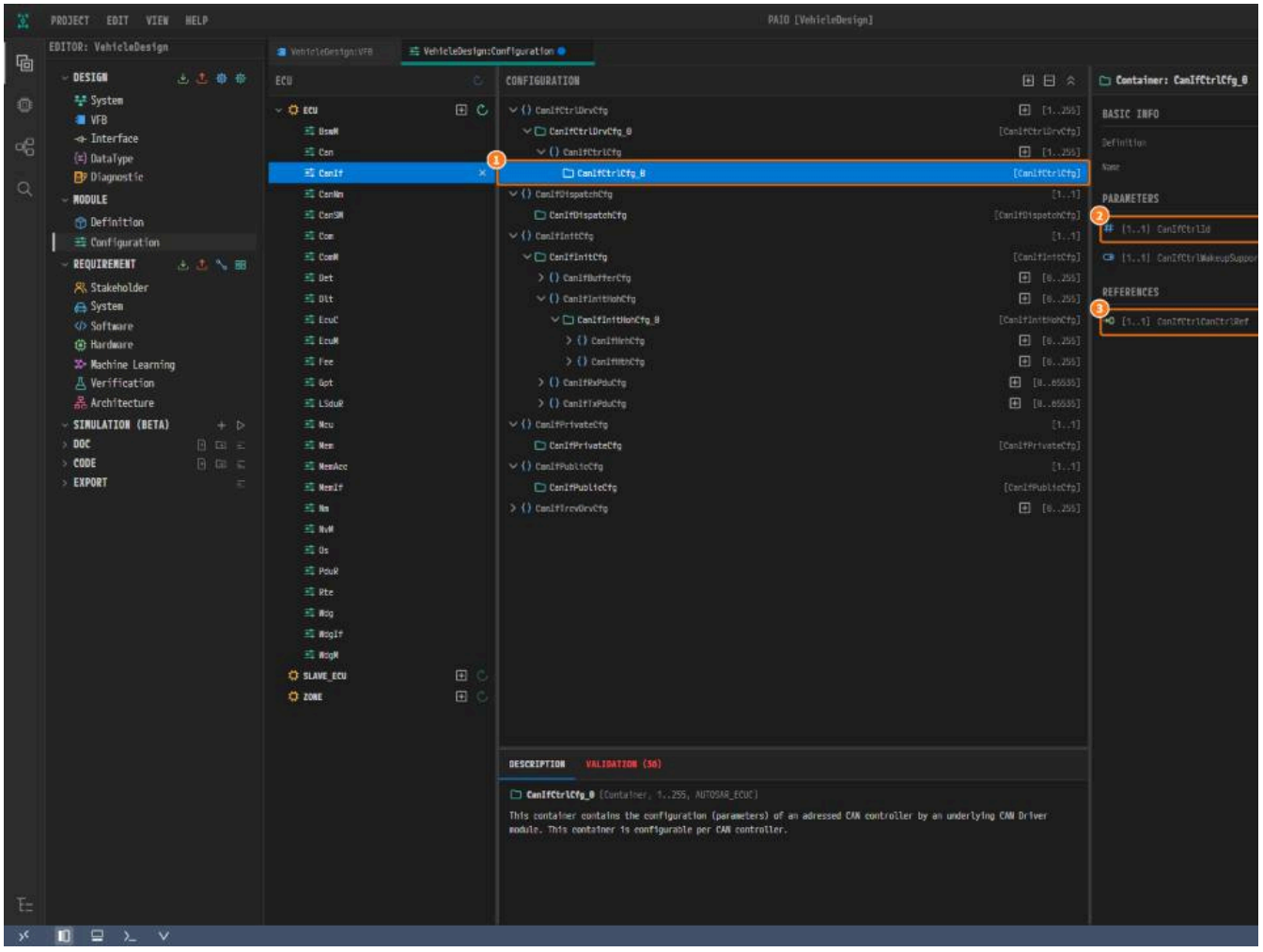


*Figure 9-12. ① `CanIfCtrlCfg_0`, ② **CanIfCtrlId**, ③ **CanIfCtrlCanCtrlRef**.*

7. Create `CanIfHthCfg_0`, `CanIfHrhCfg_0` and `CanIfBufferCfg_0` the same way from the table. The transmit handle points to `CanHardwareObject_0` and the buffer to the transmit handle.

8. Under `CanIfInitCfg`, click the **+** on the `CanIfTxPduCfg` row.

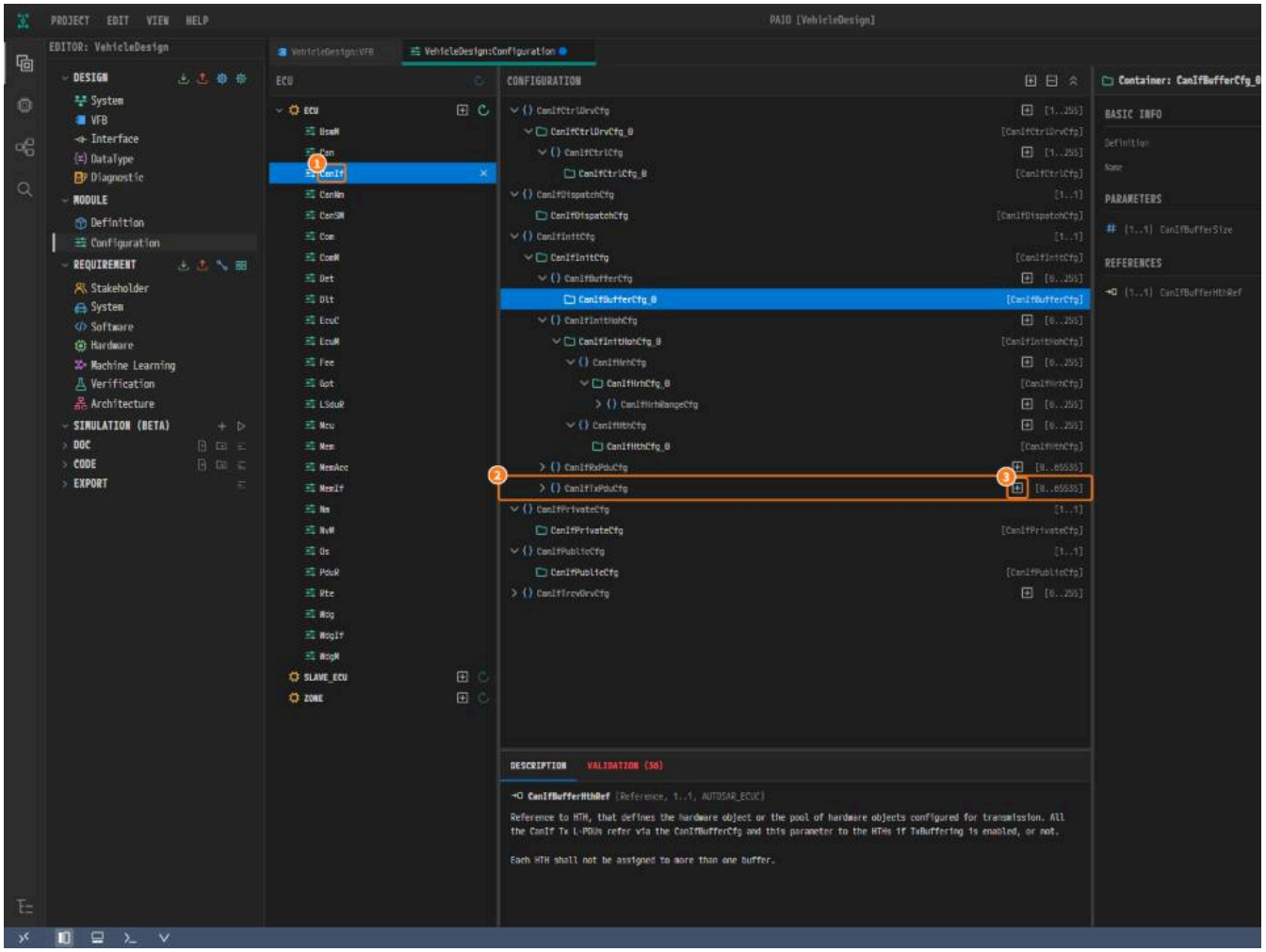


Figure 9-13. ① `CanIf`, ② `CanIfTxPduCfg`, ③ +.

9. Enter the five values of `CanIfTxPduCfg_0` from the table. `1281` is the CAN id `0x501`. **CanIfTxPduRef** picks the `PduNmTx` you made in `EcuC`.

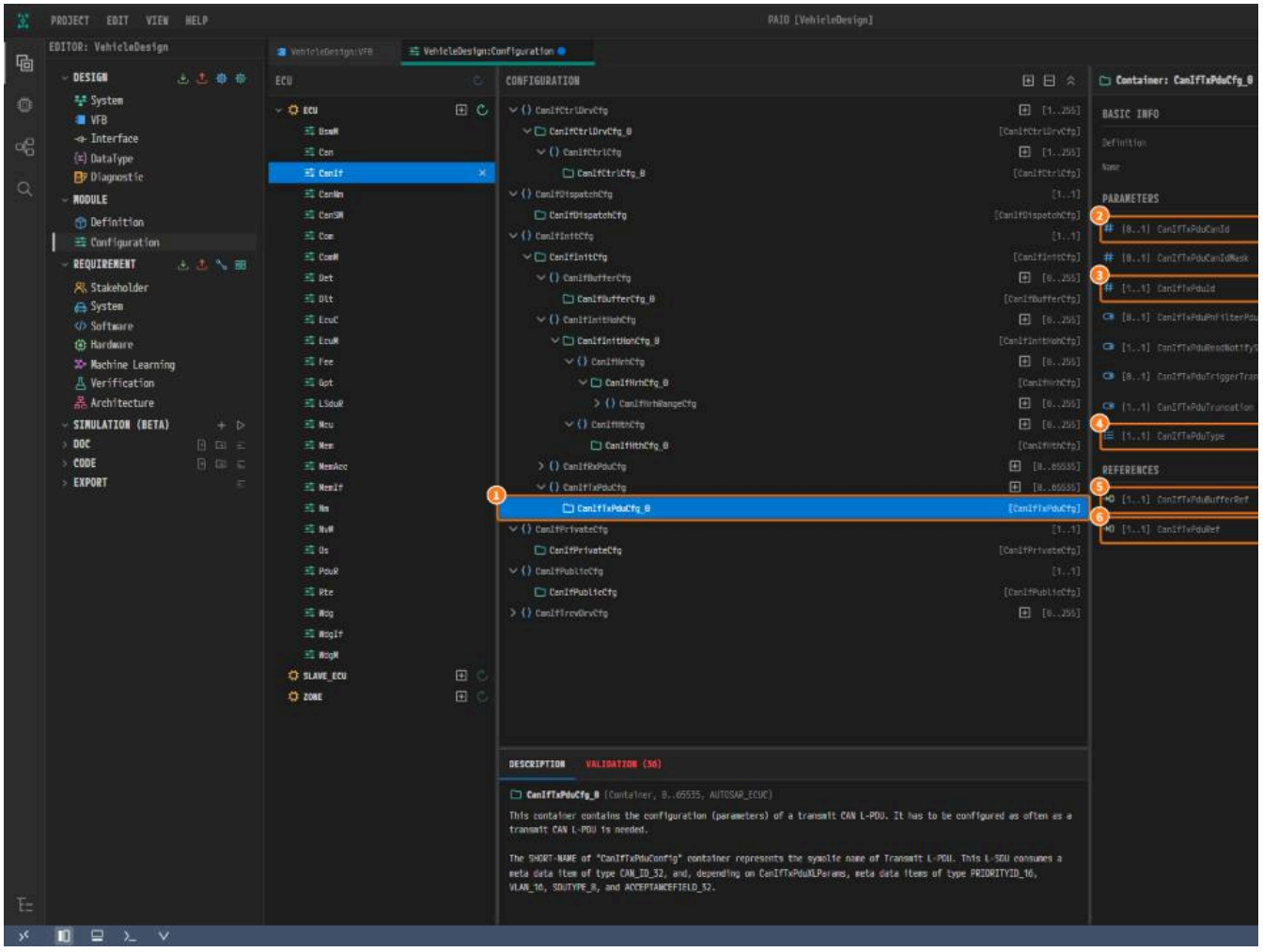


*Figure 9-14. ① `CanIfTxPduCfg_0`, ② **CanIfTxPduCanId**, ③ **CanIfTxPduId**, ④ **CanIfTxPduType**, ⑤ **CanIfTxPduBufferRef**, ⑥ **CanIfTxPduRef**.*

10. Create `CanIfRxPduCfg_0` the same way from the table. `1282` is the CAN id `0x502`, and **CanIfRxPduRef** picks `PduNmRx`. Save with `Ctrl+S`.

ComM Configuration

Container	How you add it	Item	Value
`ComMGeneral`	+ on `ComMGeneral`		Add only
`ComMConfigSet`	+ on `ComMConfigSet`		Add only
`ComMChannel_0`	+ on `ComMChannel` under `ComMConfigSet`	**ComMBusType**	`COMM_BUS_TYPE_CAN`
		ComMChannelId	`0`
		ComMNoWakeUpInhibitionNvmStorage	**FALSE**
`ComMNetworkManagement`	+ on `ComMNetworkManagement` under `ComMChannel_0`	**ComMNmVariant**	`FULL`

1. Click `ComM` in the module list. Add `ComMGeneral` and `ComMConfigSet` with their +. Under `ComMConfigSet`, click the + on the `ComMChannel` row.

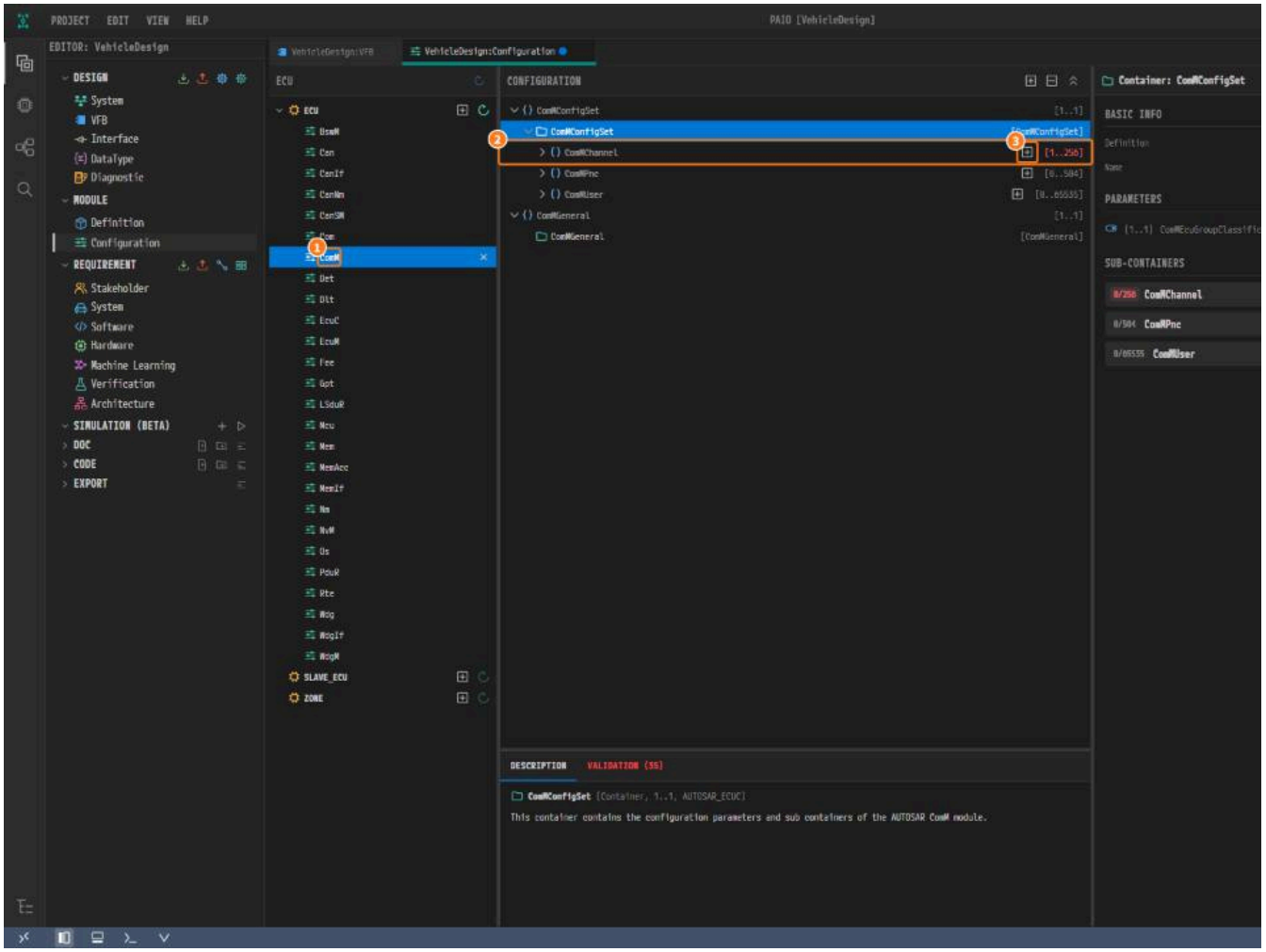


Figure 9-15. ① `ComM`, ② `ComMChannel`, ③ +.

2. Enter the three values of `ComMChannel_0` from the table.

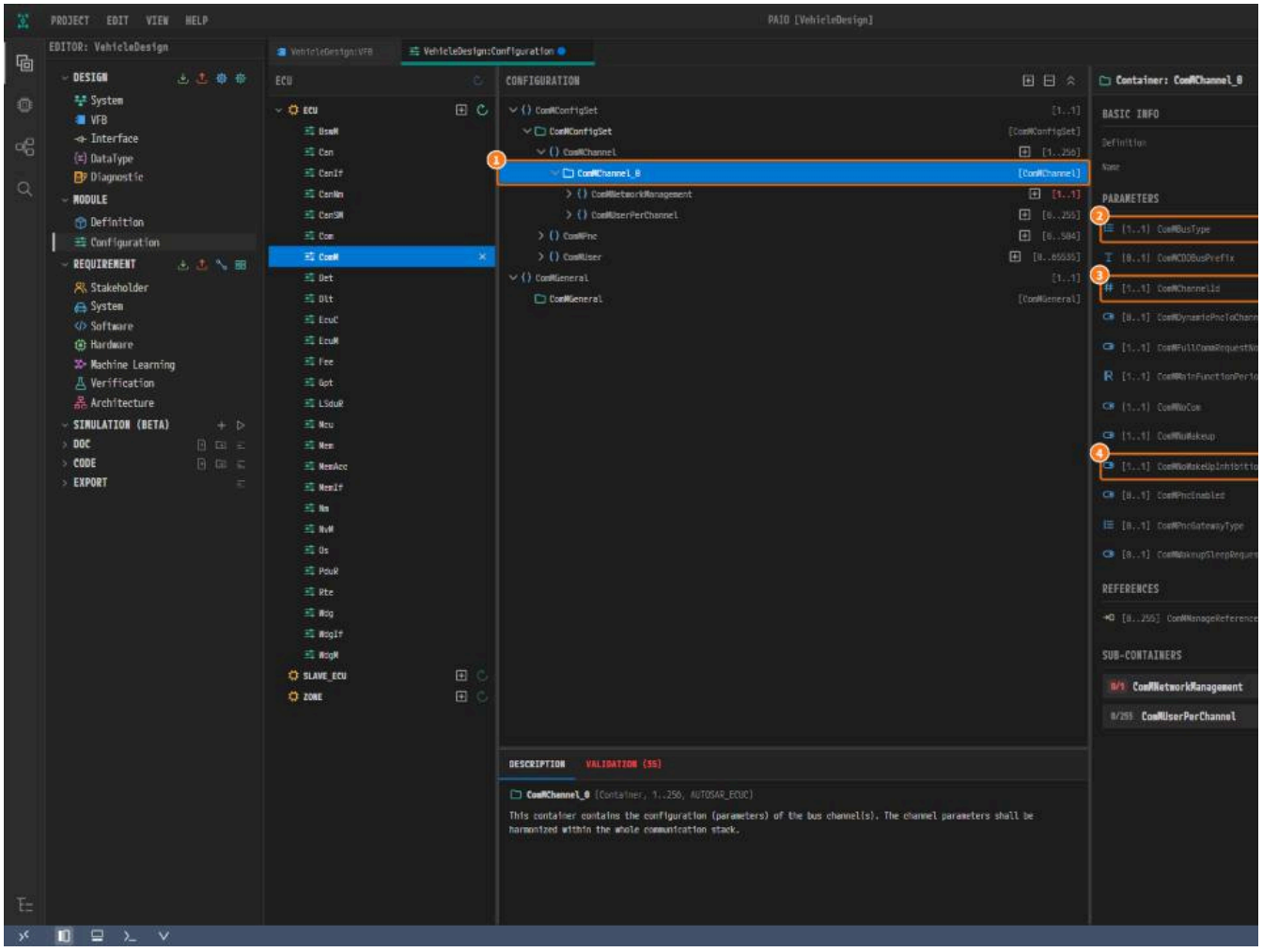


*Figure 9-16. ① `ComMChannel_0`, ② **ComMBusType**, ③ **ComMChannelId**, ④ **ComMNoWakeUpInhibitionNvmStorage**.*

3. Under `ComMChannel_0`, add `ComMNetworkManagement`. **ComMNmVariant** must be `FULL`, because this ECU runs `CanNm`. The **+** already writes `FULL`; che

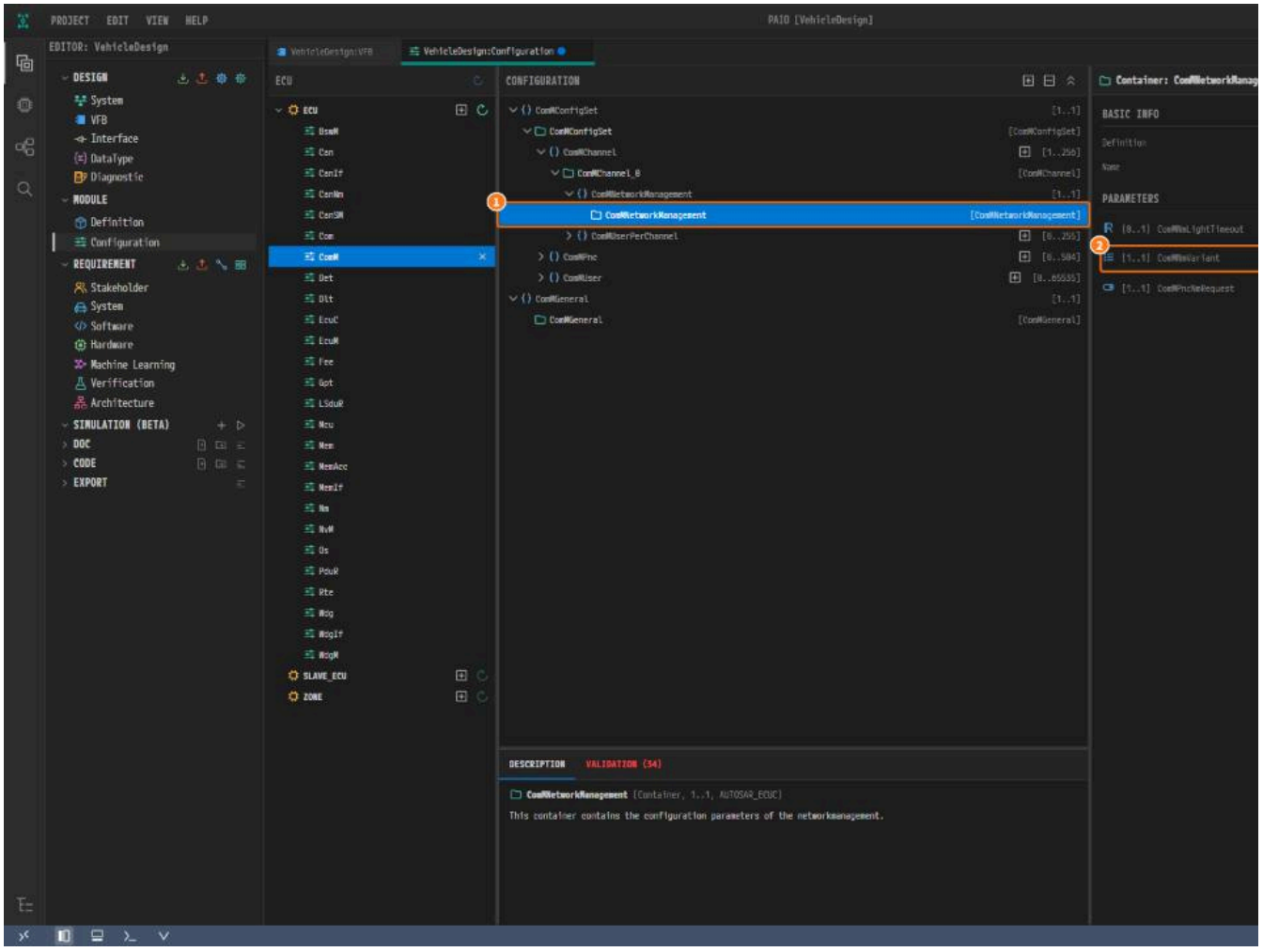


*Figure 9-17. ① `ComMNetworkManagement`, ② **`ComMNmVariant`**.*

CanSM Configuration

The four bus-off recovery values of `CanSMManagerNetwork_0` are the values you entered on `CanCluster` in the System tutorial (L1 `100` ms, L2 `200` ms, c hand and keep them equal.

Container	How you add it	Item	Value
`CanSMGeneral`	+ on `CanSMGeneral`	**CanSMMainFunctionTimePeriod**	`0.01`
`CanSMConfiguration`	+ on `CanSMConfiguration`	**CanSMModeRequestRepetitionMax**	`3`
		CanSMModeRequestRepetitionTime	`0.02`
`CanSMManagerNetwork_0`	+ on `CanSMManagerNetwork` under `CanSMConfiguration`	**CanSMBorCounterL1ToL2**	`2`
		CanSMBorTimeL1	`0.1`
		CanSMBorTimeL2	`0.2`
		CanSMBorTimeTxEnsured	`0.1`
		CanSMBorTxConfirmationPolling	**TRUE**
		CanSMComMNetworkHandleRef	`ComM/ComMConfigSet/ComMChannel_0`
`CanSMController_0`	+ on `CanSMController` under `CanSMManagerNetwork_0`	**CanSMControllerId**	`CanIf/CanIfCtrlDrvCfg_0/CanIfCtrlCfg_`

1. Click `CanSM` in the module list. Add `CanSMGeneral` and `CanSMConfiguration` with their **+** and enter their values from the table. Under `CanSMConfigu` `CanSMManagerNetwork` row.

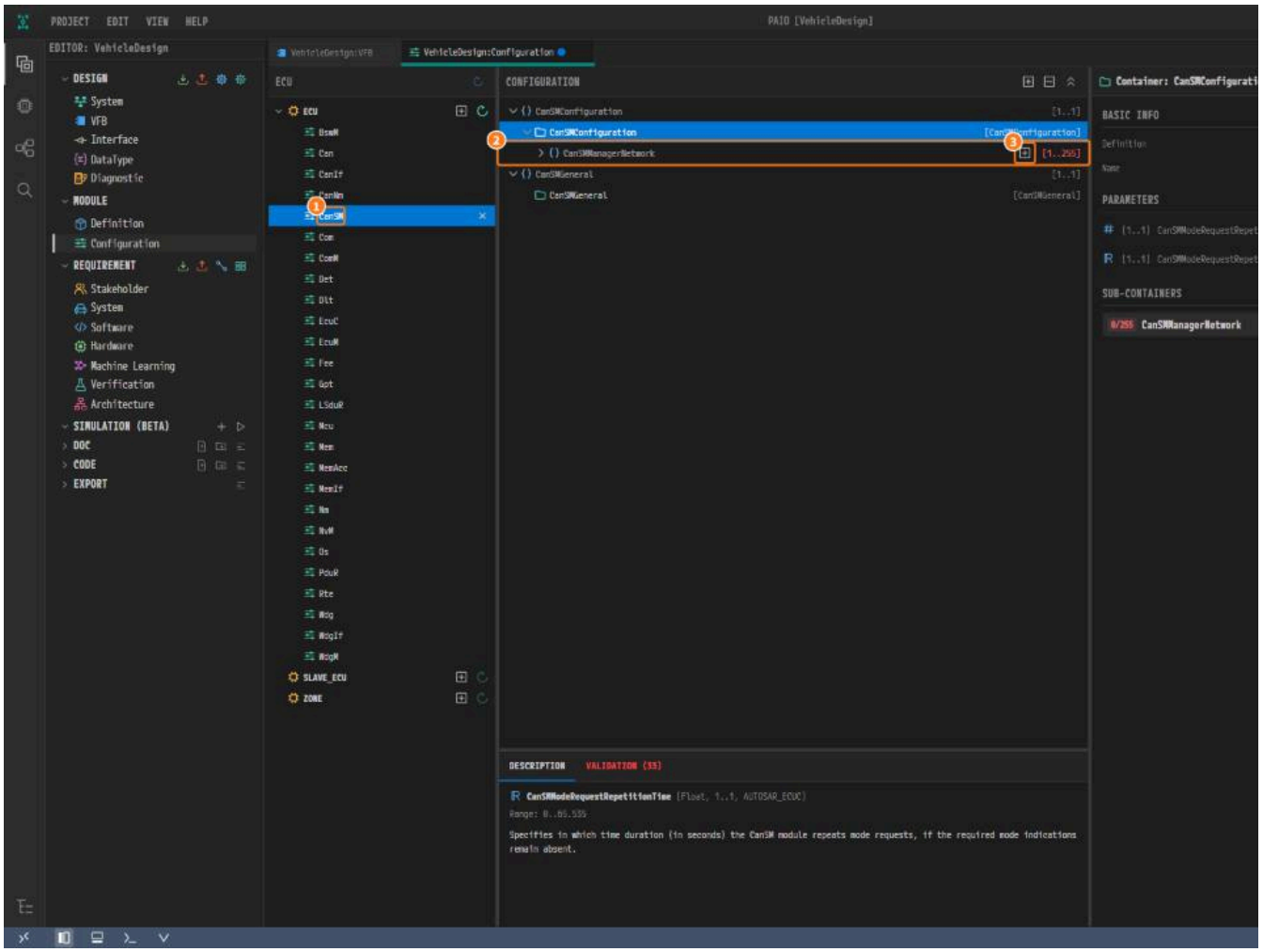


*Figure 9-18. ① `CanSM`, ② `CanSMManagerNetwork`, ③ **+**.*

2. Enter the six values of `CanSMManagerNetwork_0` from the table. **CanSMComMNetworkHandleRef** picks the channel you made in `ComM`.

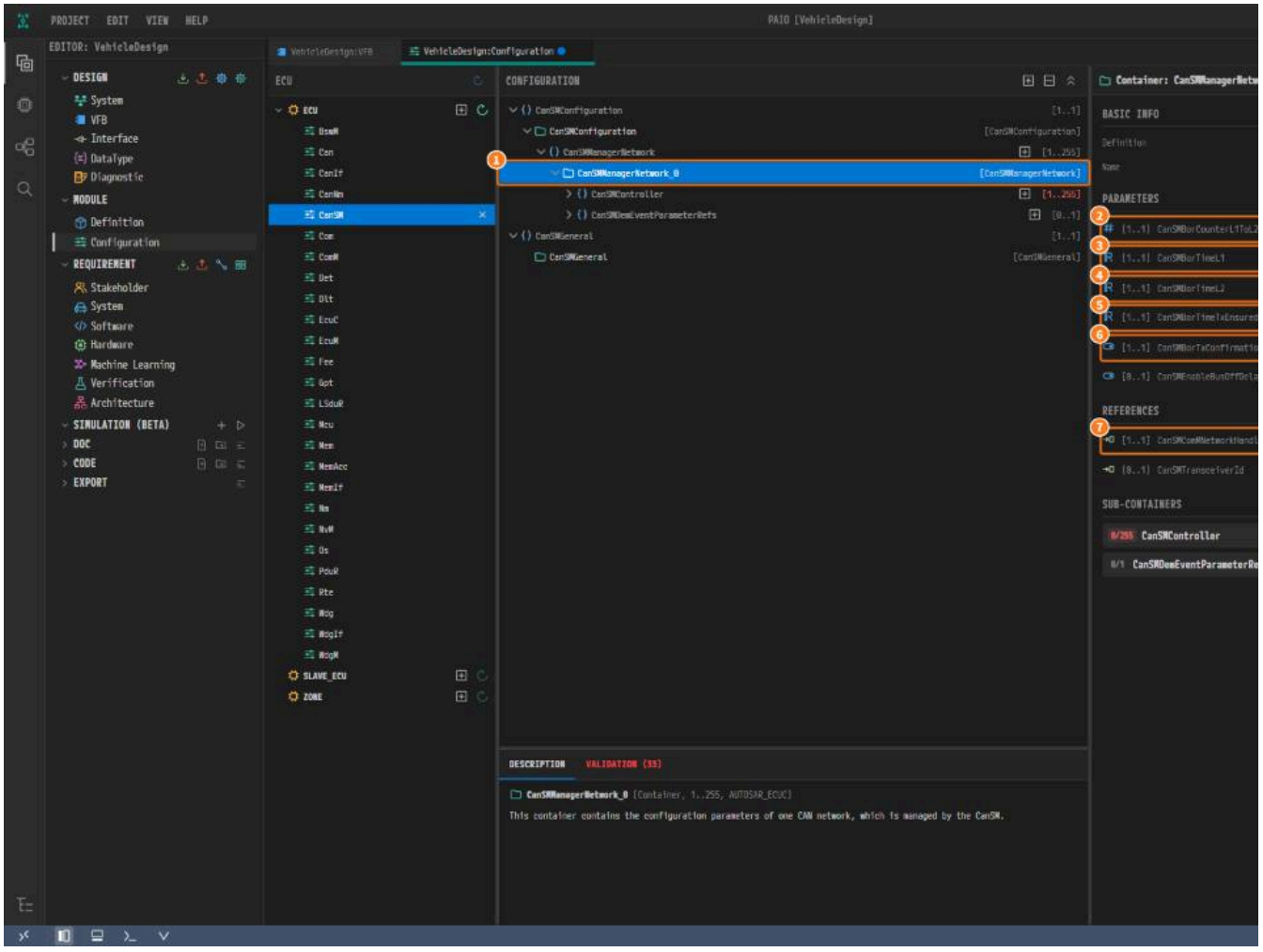


*Figure 9-19. ① `CanSMManagerNetwork_0`, ② **CanSMBorCounterL1ToL2**, ③ **CanSMBorTimeL1**, ④ **CanSMBorTimeL2**, ⑤ **CanSMBorTimeTxEnsured**, ⑥ **CanS**
CanSMComMNetworkHandleRef.*

3. Under `CanSMManagerNetwork_0`, add `CanSMController_0` and pick `CanIf/CanIfCtrlDrvCfg_0/CanIfCtrlCfg_0` in **CanSMControllerId**. Save with `C`

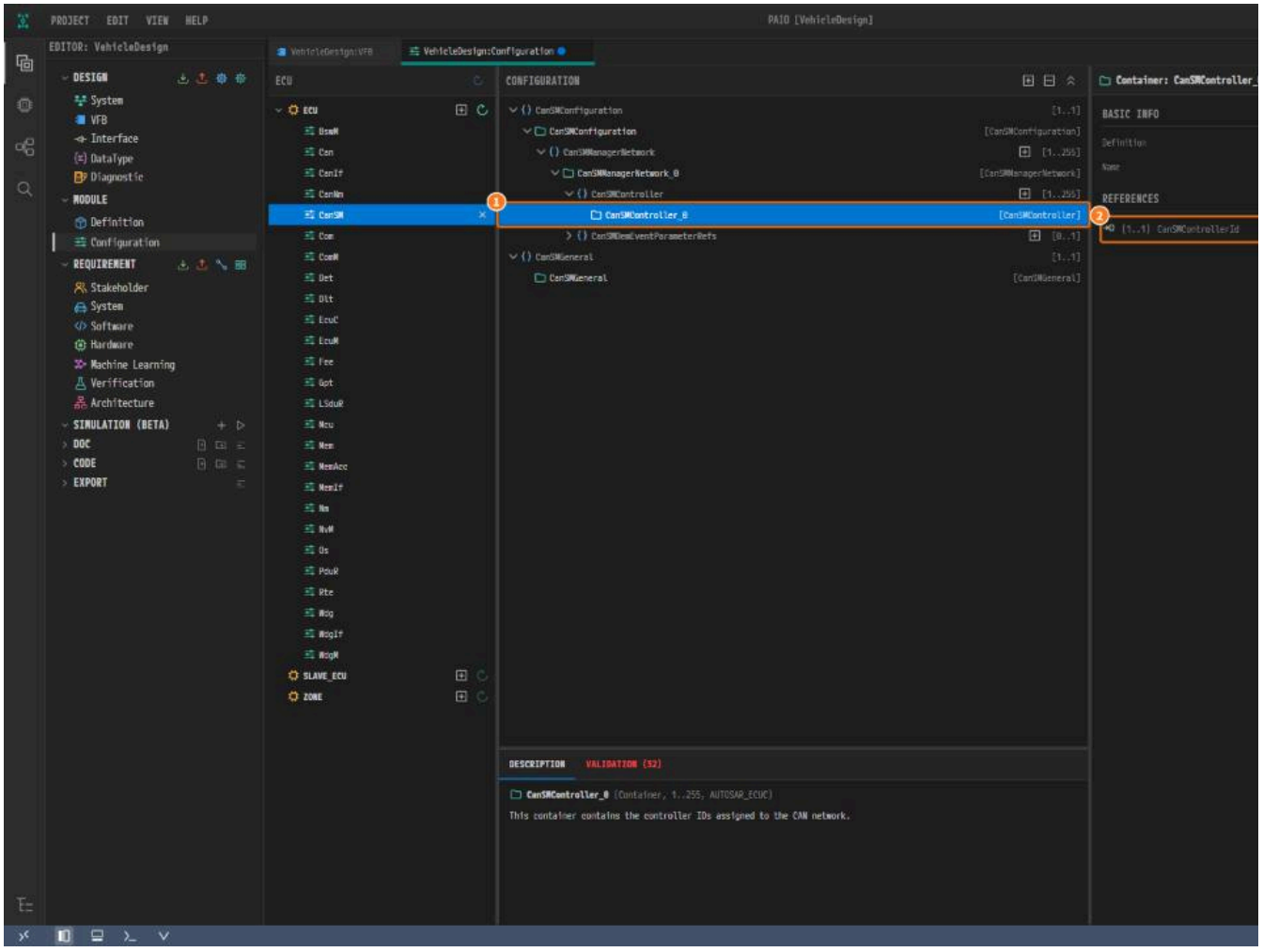


*Figure 9-20. ① `CanSMController_0`, ② **CanSMControllerId**.*

Nm Configuration

Container	How you add it	Item
`NmGlobalConfig`	+ on `NmGlobalConfig`	
`NmGlobalConstants`	+ on `NmGlobalConstants` under `NmGlobalConfig`	**NmNumberOfChannels**
`NmGlobalFeatures`	+ on `NmGlobalFeatures` under `NmGlobalConfig`	**NmBusSynchronizationEnabled**, **NmComControlEnabled**, **NmCoordinatorSyncSupport**, **NmDynamicPncToChannelMappingSupport**, **NmPduRxIndicationEnabled**, **NmRemoteSleepIndEnabled**, **NmStateChangeIndEnabled**, **NmUserDataEnabled**
`NmGlobalProperties`	+ on `NmGlobalProperties` under `NmGlobalConfig`	**NmCycletimeMainFunction**
`NmChannelConfig_0`	+ on `NmChannelConfig`	**NmChannelSleepMaster**, **NmPassiveModeEnabled**, **NmStateReportEnabled**, **NmSynchronizingNetwork**
		NmComMChannelRef
`NmBusType`	+ on `NmBusType` under `NmChannelConfig_0`	
`NmStandardBusNmConfig`	+ on `NmStandardBusNmConfig` under `NmBusType`	**NmStandardBusType**

1. Click `Nm` in the module list. Add `NmGlobalConfig` with its **+**. Under it, add `NmGlobalConstants` and type `1` into **NmNumberOfChannels**.

2. Under `NmGlobalConfig`, add `NmGlobalFeatures` and click **FALSE** on each of its eight items in the table. None of them has a default value.

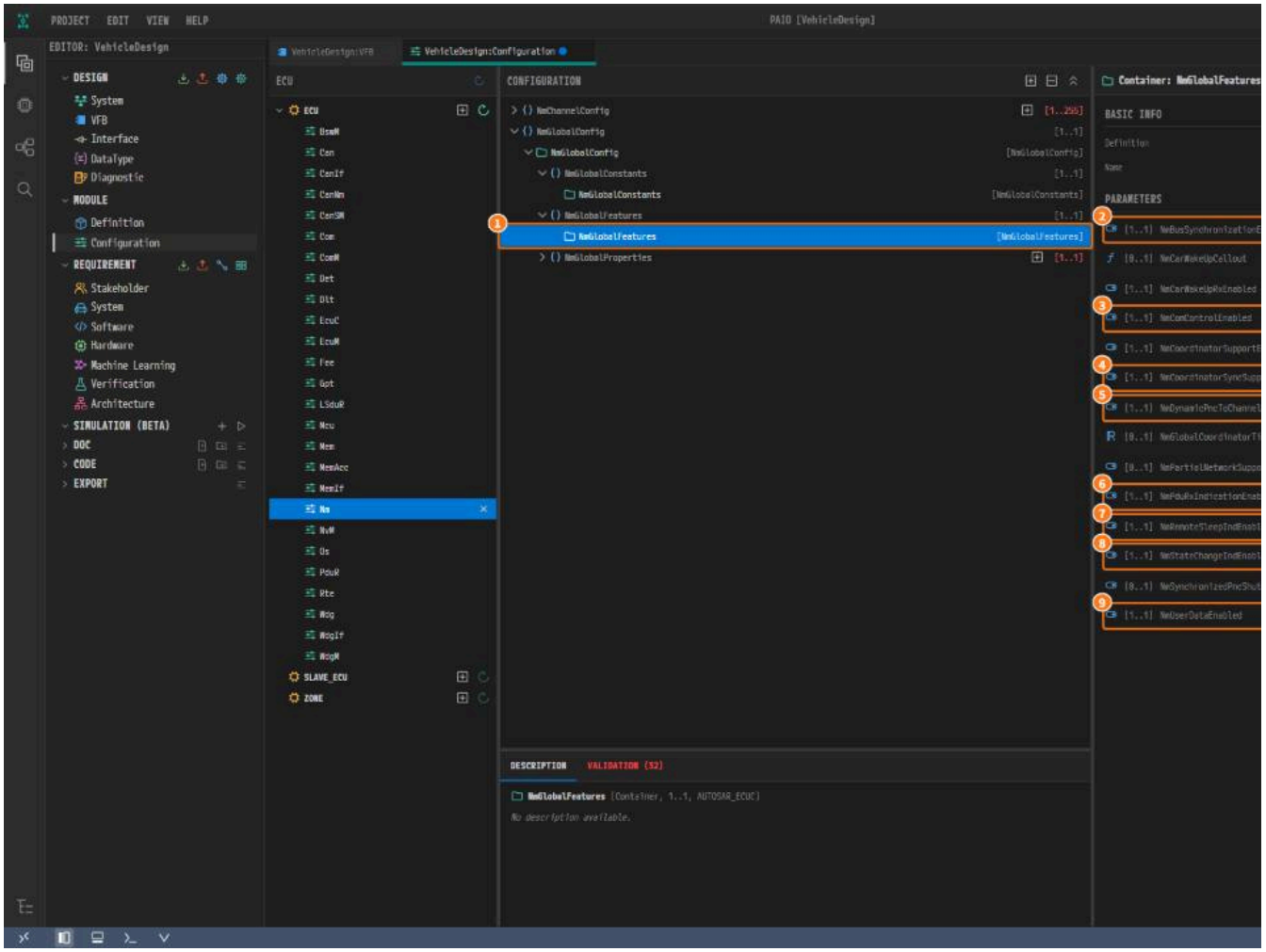


*Figure 9-21. ① `NmGlobalFeatures`, ② **NmBusSynchronizationEnabled**, ③ **NmComControlEnabled**, ④ **NmCoordinatorSyncSupport**, ⑤ **NmDynamicPncT** **NmPduRxIndicationEnabled**, ⑦ **NmRemoteSleepIndEnabled**, ⑧ **NmStateChangeIndEnabled**, ⑨ **NmUserDataEnabled**.*

3. Under `NmGlobalConfig`, add `NmGlobalProperties` and type `0.01` into **NmCycletimeMainFunction**. Then click the **+** on the `NmChannelConfig` row at

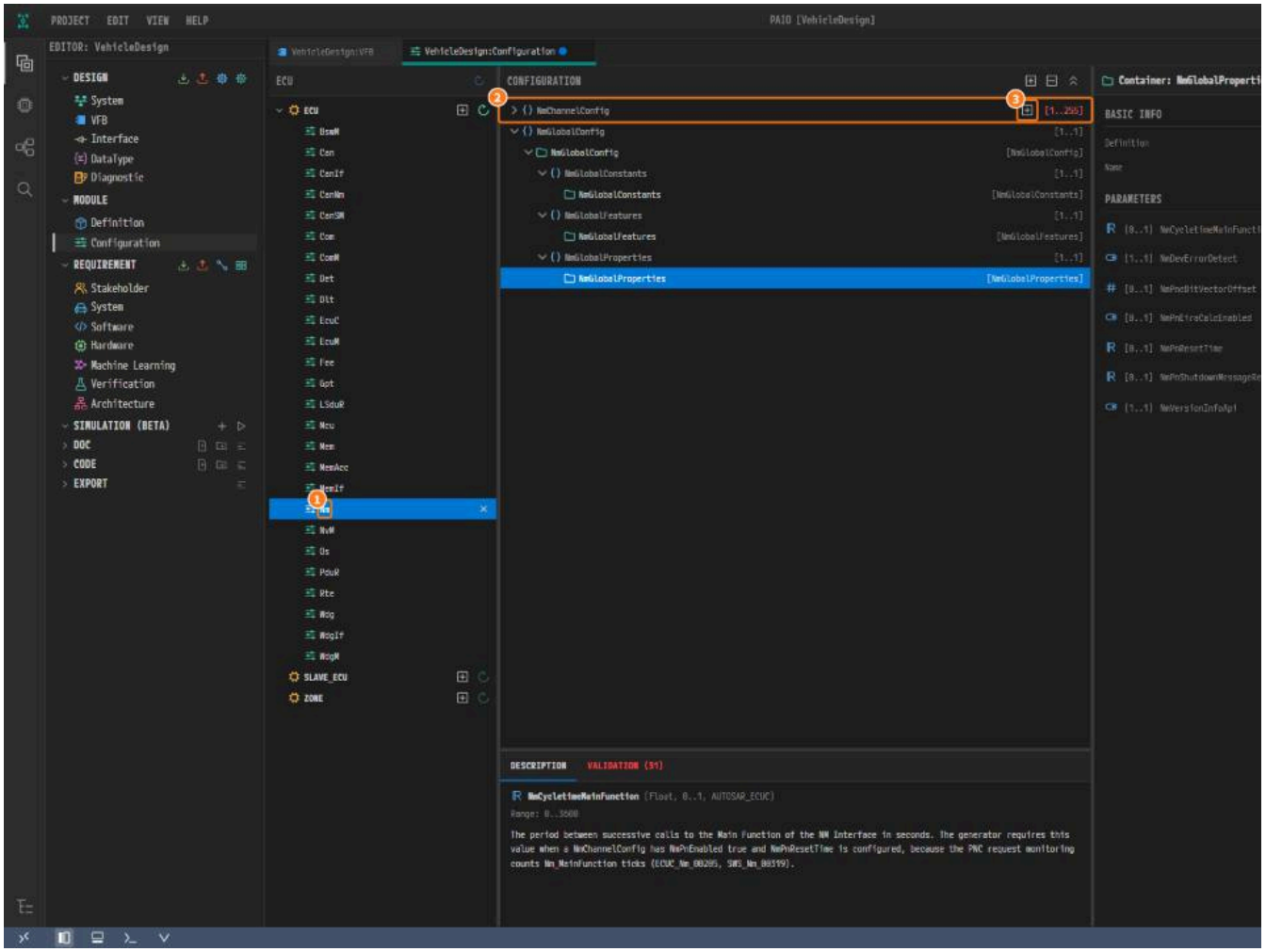


Figure 9-22. ① `Nm`, ② `NmChannelConfig`, ③ `+`.

4. Click **FALSE** on the four items of `NmChannelConfig_0` and pick `ComM/ComMConfigSet/ComMChannel_0` in **NmComMChannelRef**.

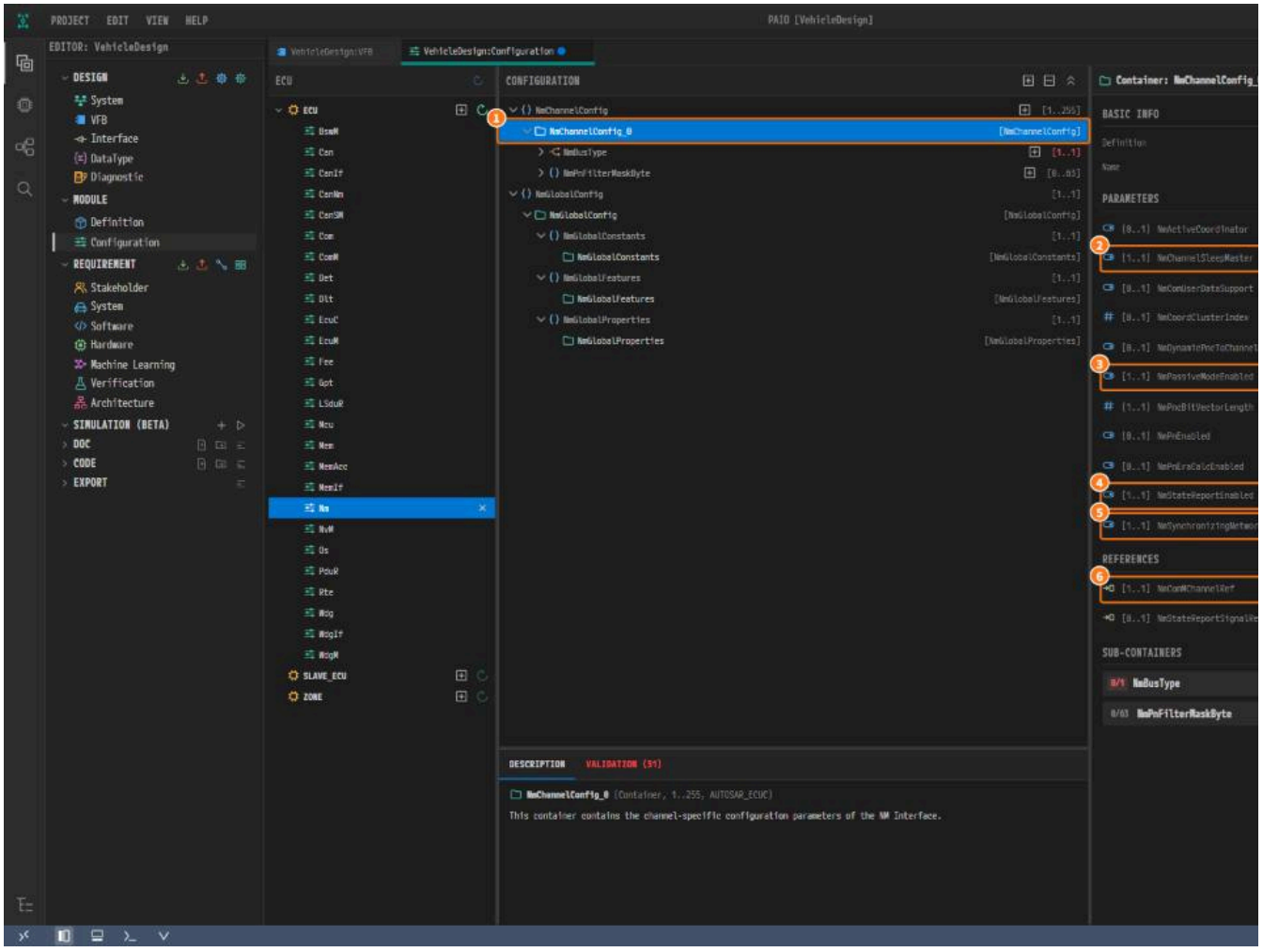


*Figure 9-23. ① `NmChannelConfig_0`, ② **NmChannelSleepMaster**, ③ **NmPassiveModeEnabled**, ④ **NmStateReportEnabled**, ⑤ **NmSynchronizingNetwork**,*

5. Under `NmChannelConfig_0`, add `NmBusType`, and under it `NmStandardBusNmConfig`. Pick `NM_BUSNM_CANNM` in **NmStandardBusType**. Save with `Ctrl+S`

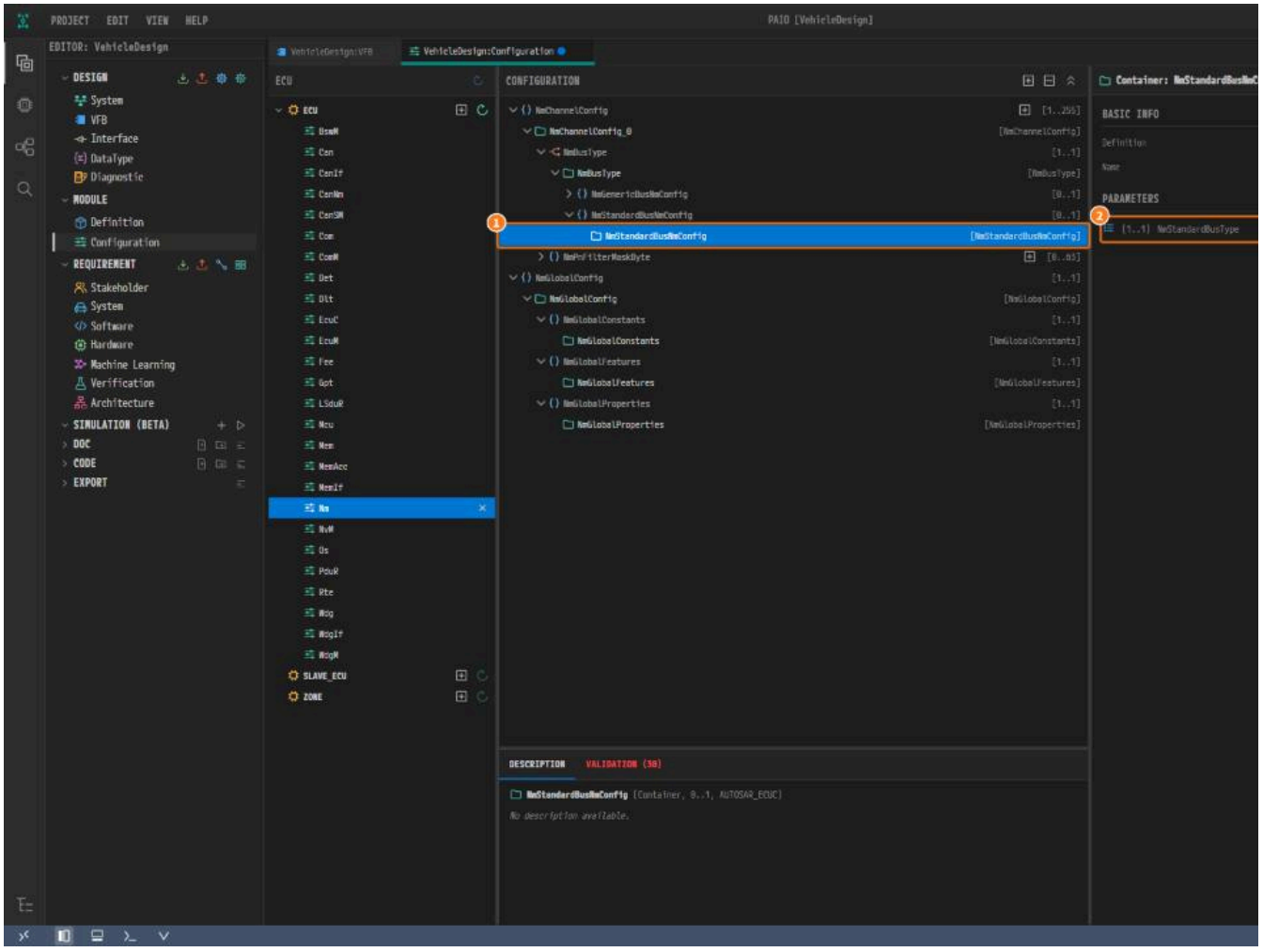


*Figure 9-24. ① `NmStandardBusNmConfig`, ② **`NmStandardBusType`**.*

CanNm Configuration

Container	How you add it	Item
`CanNmGlobalConfig`	+ on `CanNmGlobalConfig`	**CanNmBusLoadReductionEnabled**, **CanNmBusSynchronizationEnabled**, **CanNmComControlEnabled**, **CanNmComUserDataSupport**, **CanNmCoordinatorSyncSupport**, **CanNmDynamicPncToChannelMappingSupport**, **CanNmImmediateRestartEnabled**, **CanNmImmediateTxconfEnabled**, **CanNmPassiveModeEnabled**, **CanNmPduRxIndicationEnabled**, **CanNmRemoteSleepIndEnabled**, **CanNmStateChangeIndEnabled**, **CanNmUserDataEnabled**
		CanNmMainFunctionPeriod
`CanNmChannelConfig_0`	+ on `CanNmChannelConfig` under `CanNmGlobalConfig`	**CanNmMsgCycleTime**
		CanNmTimeoutTime
		CanNmRepeatMessageTime
		CanNmWaitBusSleepTime
		CanNmRemoteSleepIndTime
		CanNmImmediateNmCycleTime
		CanNmImmediateNmTransmissions
		CanNmMsgCycleOffset
		CanNmMsgReducedTime
		CanNmNodeId
		CanNmNodeIdEnabled
		CanNmPduCbvPosition
		CanNmPduNidPosition
		CanNmBusLoadReductionActive, **CanNmNodeDetectionEnabled**, **CanNmRepeatMsgIndEnabled**
		CanNmCarWakeUpFilterEnabled
		CanNmComMNetworkHandleRef
`CanNmRxPdu_0`	+ on `CanNmRxPdu` under `CanNmChannelConfig_0`	**CanNmRxPduId**
		CanNmRxPduRef
`CanNmTxPdu`	+ on `CanNmTxPdu` under `CanNmChannelConfig_0`	**CanNmTxConfirmationPduId**
		CanNmTxPduRef

1. Click `CanNm` in the module list. Add `CanNmGlobalConfig` with its **+**. Click **FALSE** on its thirteen items in the table, which have no default value, and typ

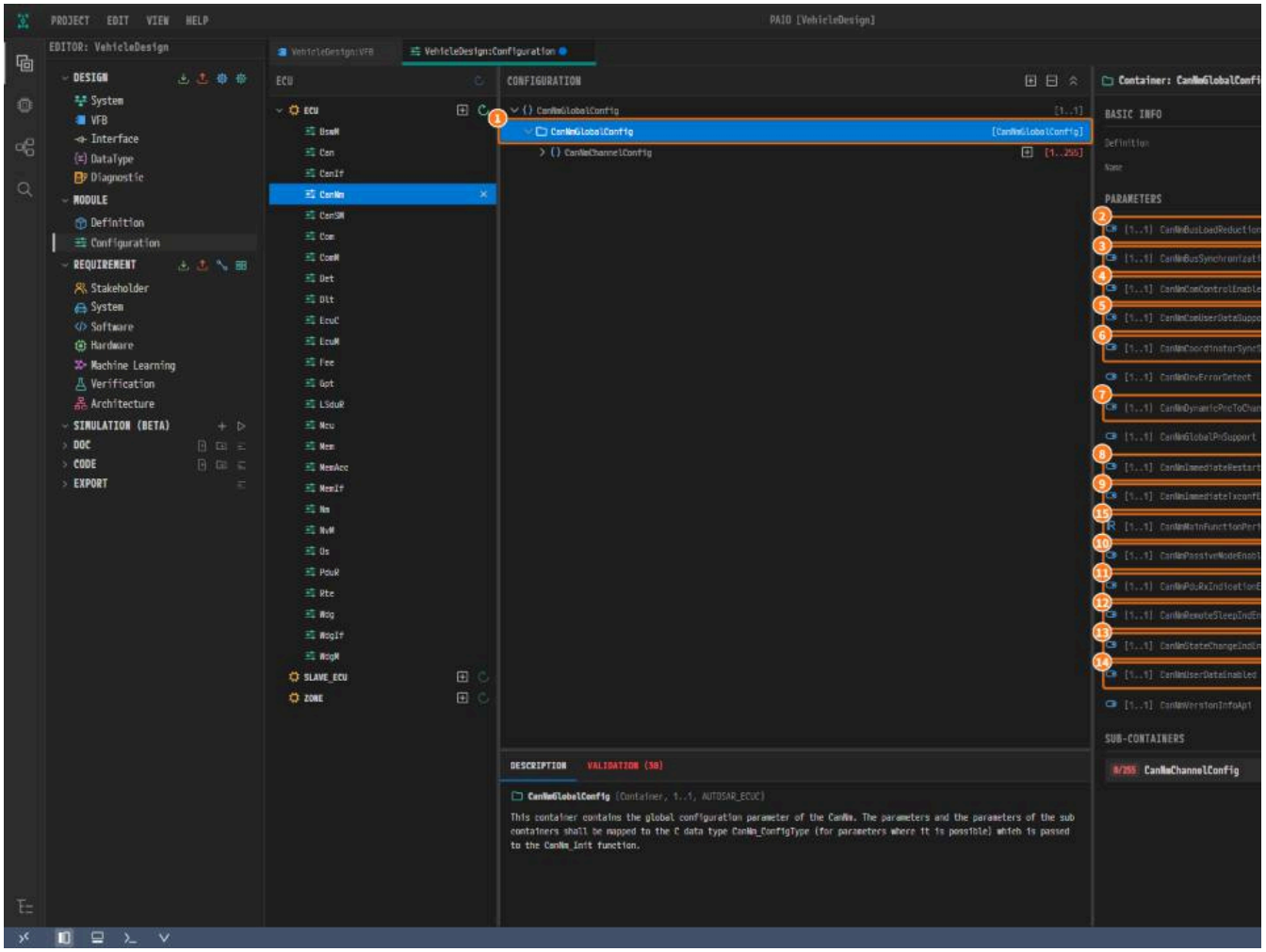


*Figure 9-25. ① `CanNmGlobalConfig`, ② **CanNmBusLoadReductionEnabled**, ③ **CanNmBusSynchronizationEnabled**, ④ **CanNmComControlEnabled**, ⑤ **Ca**
CanNmCoordinatorSyncSupport, ⑦ **CanNmDynamicPncToChannelMappingSupport**, ⑧ **CanNmImmediateRestartEnabled**, ⑨ **CanNmImmediateTxconfE**
CanNmPduRxIndicationEnabled, ⑫ **CanNmRemoteSleepIndEnabled**, ⑬ **CanNmStateChangeIndEnabled**, ⑭ **CanNmUserDataEnabled**, ⑮ **CanNmMainFun***

2. Under `CanNmGlobalConfig`, click the **+** on the `CanNmChannelConfig` row.

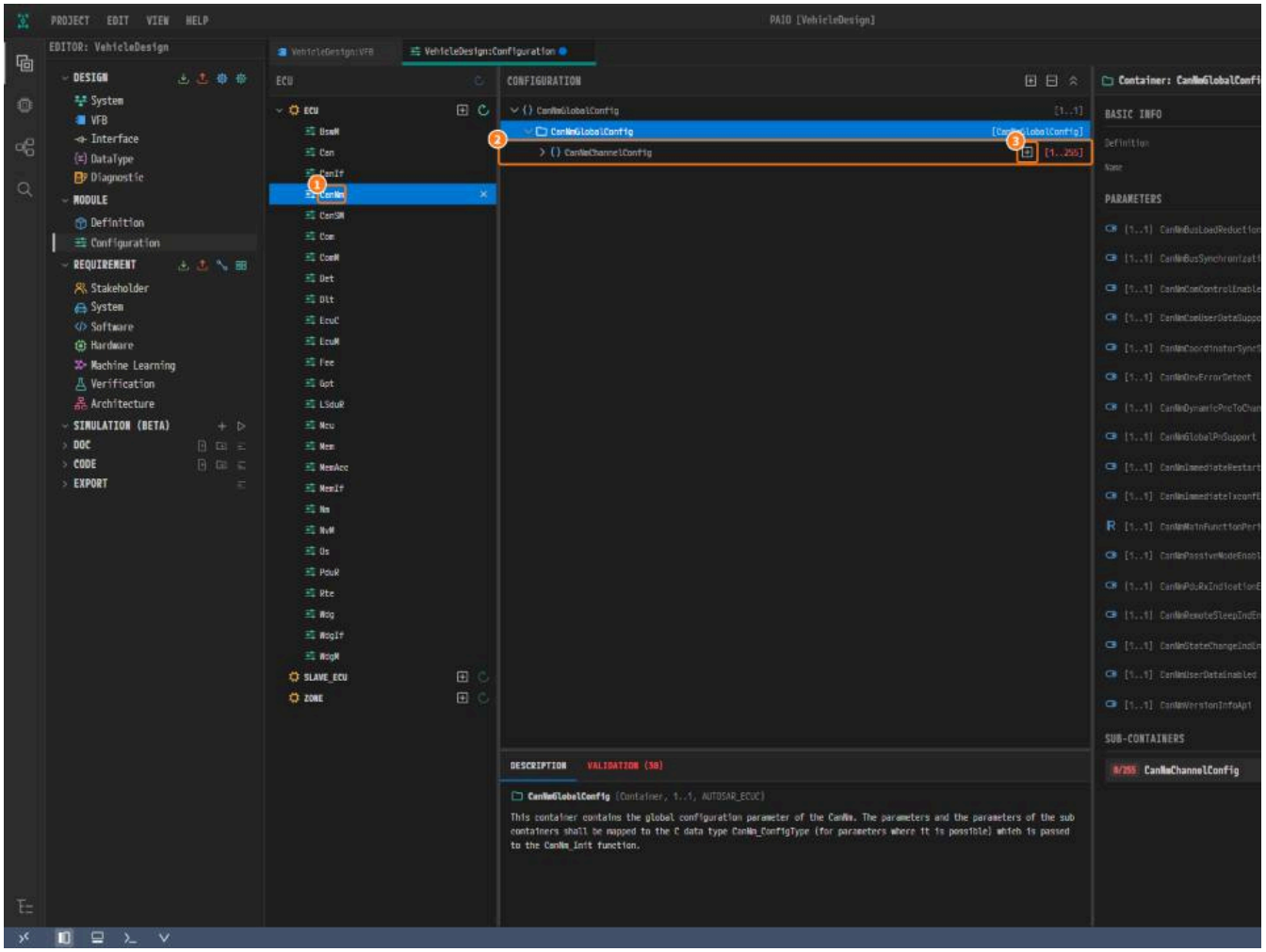


*Figure 9-26. ① `CanNm`, ② `CanNmChannelConfig`, ③ **+**.*

3. Enter the values of `CanNmChannelConfig_0` from the table: the times and numbers, **TRUE** on **CanNmNodeIdEnabled**, the two byte positions, **FALSE** or channel of `ComM` in **CanNmComMNetworkHandleRef**.

4. **CanNmCarWakeUpFilterEnabled** must stay empty. The **+** writes a value into it, so click its selected button once more to clear it. With a value there, PF

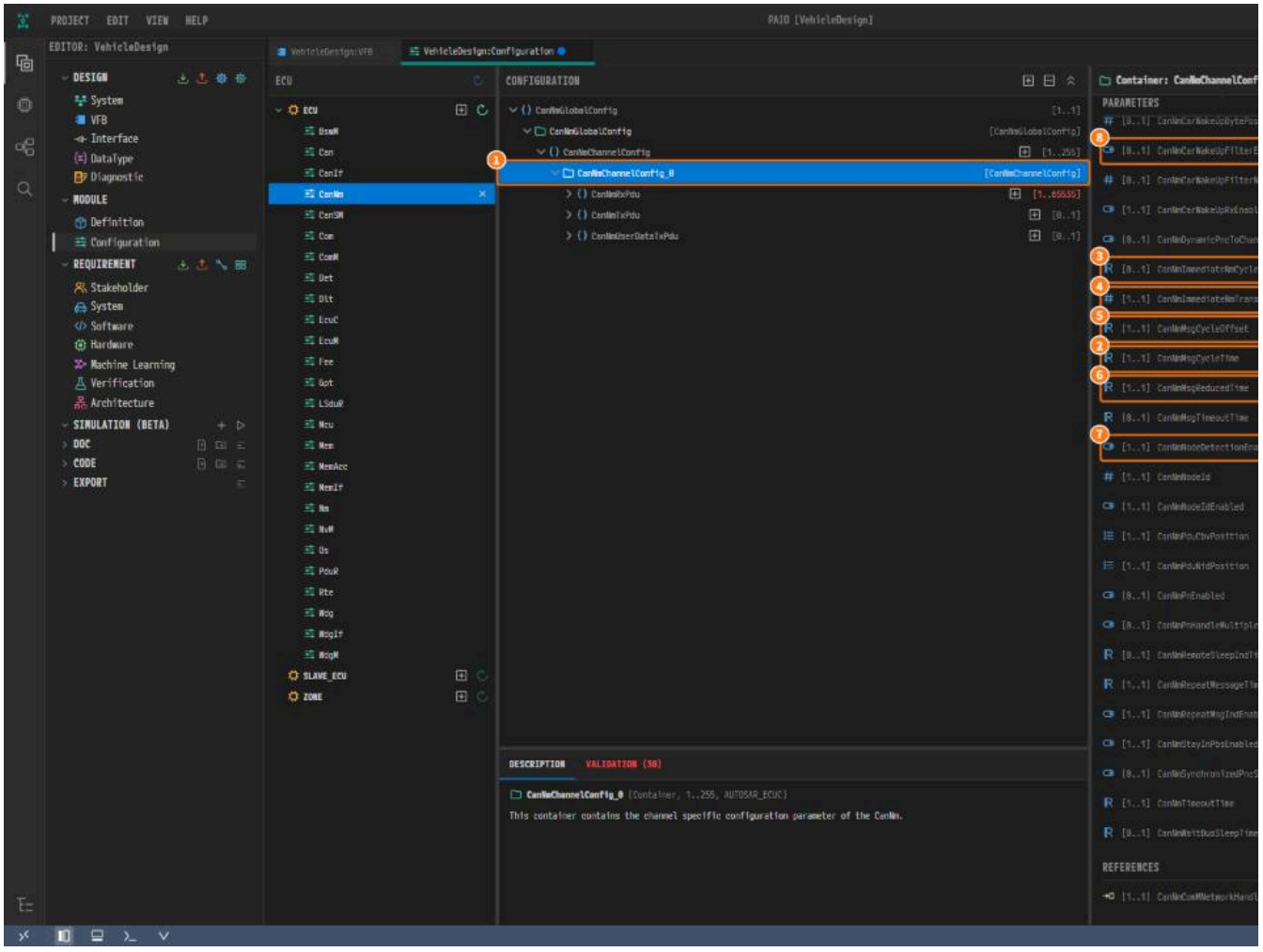


*Figure 9-27. ① `CanNmChannelConfig_0`, ② **CanNmMsgCycleTime**, ③ **CanNmImmediateNmCycleTime**, ④ **CanNmImmediateNmTransmissions**, ⑤ **CanNmI**
CanNmNodeDetectionEnabled, ⑧ **CanNmCarWakeUpFilterEnabled** with neither button selected.*

5. Under `CanNmChannelConfig_0`, add `CanNmRxPdu_0`, type `0` into **CanNmRxPduId** and pick `PduNmRx` in **CanNmRxPduRef**.

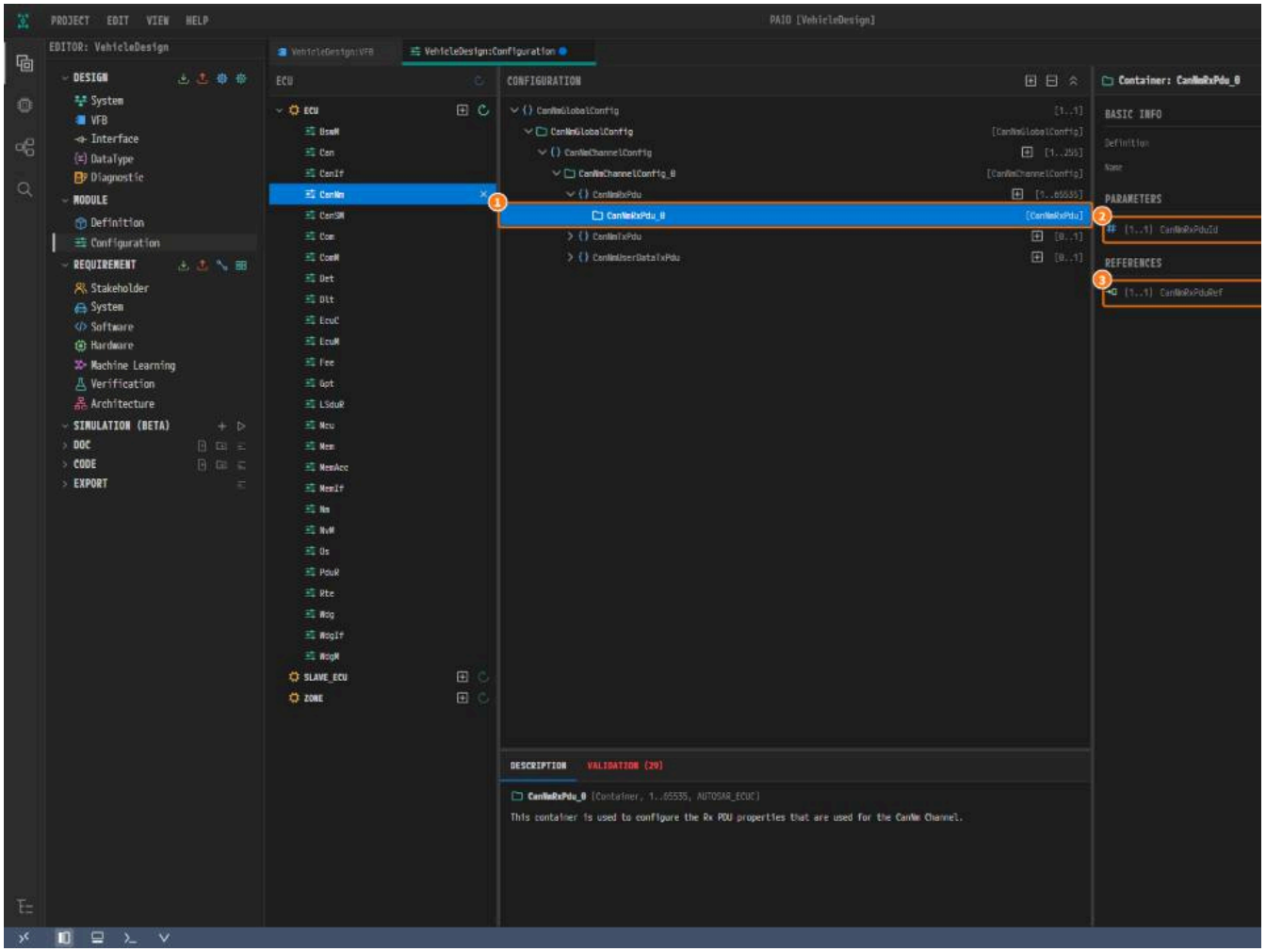


*Figure 9-28. ① `CanNmRxPdu_0`, ② **CanNmRxPduId**, ③ **CanNmRxPduRef**.*

6. Create `CanNmTxPdu` the same way from the table, with `PduNmTx` in **CanNmTxPduRef**. Save with `Ctrl+S`.

LSduR Configuration

The values of `LSduR`, `PduR` and `Com` are in this table.

Module	Container	How you add it	Item	Value
`LSduR`	`LSduRGeneral`	+ on `LSduRGeneral`	**LSduRMetaDataSupport**	**FALSE**
			LSduRZeroCostOperation	**FALSE**
`LSduR`	`LSduRConfig`	+ on `LSduRConfig`		Add only
`LSduR`	`LSduRBswModules_0`	+ on `LSduRBswModules`	**LSduRRxIndication**	**TRUE**
			LSduRTransmit	**TRUE**
			LSduRTxConfirmation	**TRUE**
			LSduRProvideTxBuffer	**FALSE**
			LSduRReleaseRxBuffer	**FALSE**
			LSduRLowerModule	**TRUE**
			LSduRUpperModule	**FALSE**
			LSduRBswModuleRef	`ECU/CanIf` (picked)
`LSduR`	`LSduRBswModules_1`	+ on `LSduRBswModules`	**LSduRRxIndication**	**TRUE**
			LSduRTransmit	**TRUE**
			LSduRTxConfirmation	**TRUE**
			LSduRProvideTxBuffer	**FALSE**
			LSduRReleaseRxBuffer	**FALSE**
			LSduRLowerModule	**FALSE**
			LSduRUpperModule	**TRUE**
			LSduRBswModuleRef	`ECU/CanNm` (picked)
`LSduR`	`LSduRConfig/LSduRPath_0`	+ on `LSduRPath`		Add only
`LSduR`	`LSduRPath_0/LSduRPathSourcePdu`	+ on `LSduRPathSourcePdu`	**LSduRPathSourcePduId**	`0`
			LSduRPathSourcePduRef	`EcuC/EcucConfigSet/EcucPduCollection/`
`LSduR`	`LSduRPath_0/LSduRPathDestinationPdu_0`	+ on `LSduRPathDestinationPdu`	**LSduRPathDestinationPduRef**	`EcuC/EcucConfigSet/EcucPduCollection/`
`LSduR`	`LSduRConfig/LSduRPath_1`	+ on `LSduRPath`		Add only
`LSduR`	`LSduRPath_1/LSduRPathSourcePdu`	+ on `LSduRPathSourcePdu`	**LSduRPathSourcePduId**	`1`
			LSduRPathSourcePduRef	`EcuC/EcucConfigSet/EcucPduCollection/`
`LSduR`	`LSduRPath_1/LSduRPathDestinationPdu_0`	+ on `LSduRPathDestinationPdu`	**LSduRPathDestinationPduRef**	`EcuC/EcucConfigSet/EcucPduCollection/`
`PduR`	`PduRGeneral`	+ on `PduRGeneral`	**PduRZeroCostOperation**	**FALSE**
`PduR`	`PduRRoutingPaths`	+ on `PduRRoutingPaths`	**PduRConfigurationId**	`0`
`Com`	`ComGeneral`	+ on `ComGeneral`	**ComSupportedIPduGroups**	`0`
`Com`	`ComConfig`	+ on `ComConfig`		Add only
`Com`	`ComConfig/ComMainFunctionRx`	+ on `ComMainFunctionRx`	**ComMainRxTimeBase**	`0.01` (filled in)

Module	Container	How you add it	Item	Value
`Com`	`ComConfig/ComMainFunctionTx`	+ on `ComMainFunctionTx`	**ComMainTxTimeBase**	`0.01` (filled in)

> **Caution.** Clicking a **TRUE** or **FALSE** button that is already selected clears the value. Click the same button once more in that case.

1. Click `LSduR` in the module list. In the **CONFIGURATION** tree, click the **+** on the `LSduRGeneral` row.

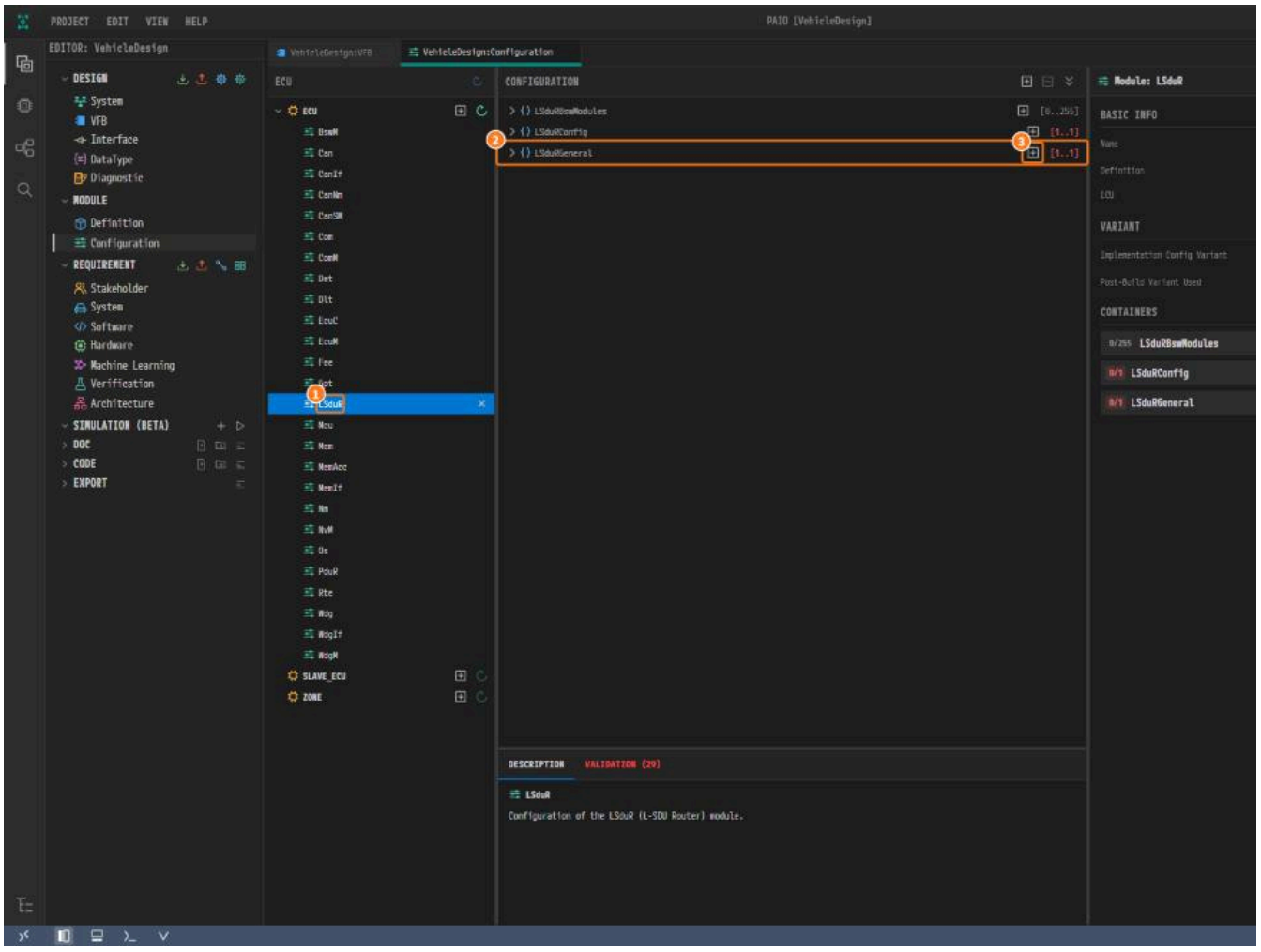


*Figure 9-29. ① `LSduR`, ② `LSduRGeneral`, ③ **+**.*

2. Click the **+** on the `LSduRConfig` row.

3. Click `LSduRGeneral`. Click **FALSE** of **LSduRMetaDataSupport** and of **LSduRZeroCostOperation**.

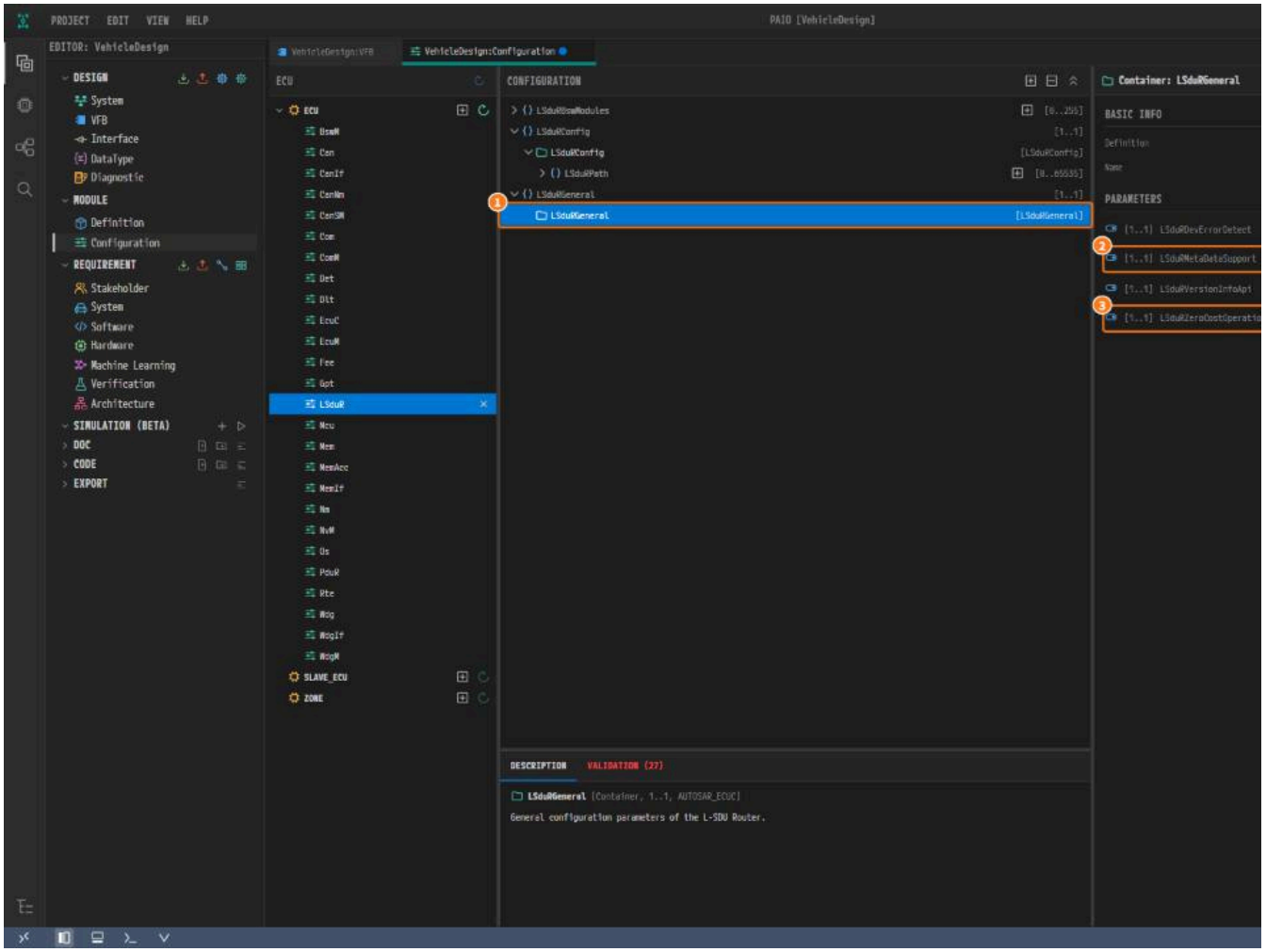


*Figure 9-30. ① `LSduRGeneral`, ② **LSduRMetaDataSupport**, ③ **LSduRZeroCostOperation**.*

4. Click the **+** on the `LSduRBswModules` row. The editor adds `LSduRBswModules_0` and selects it.

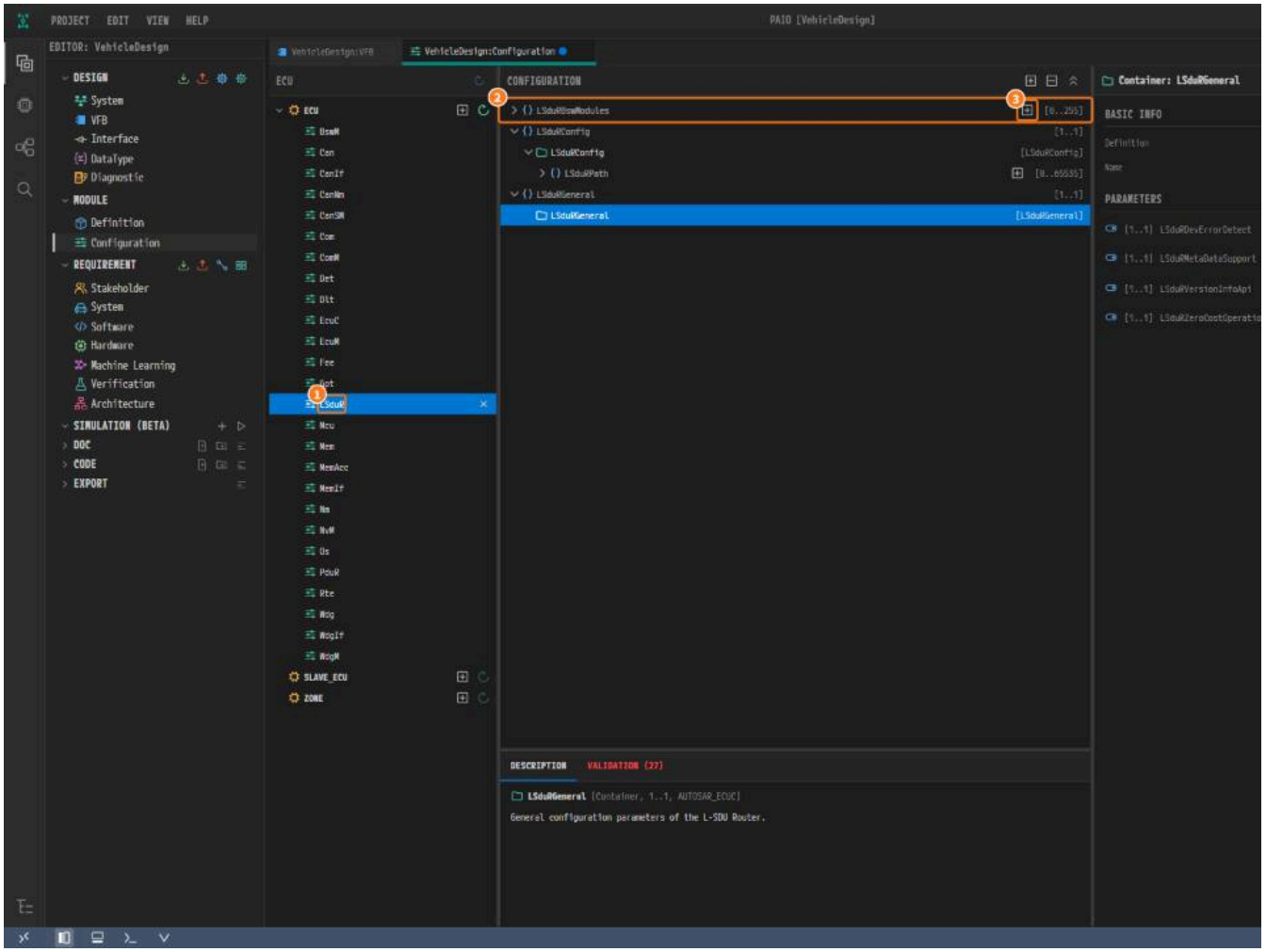


Figure 9-31. ① `LSduR`, ② `LSduRBswModules`, ③ +.

5. In `LSduRBswModules_0`, click **TRUE** of **LSduRRxIndication**, **LSduRTransmit** and **LSduRTxConfirmation**, **FALSE** of **LSduRProvideTxBuffer** and **LSduRRelea**
FALSE of **LSduRUpperModule**.

6. Open the **LSduRBswModuleRef** dropdown and pick `ECU/CanIf`. Pick it from the list: a typed `CanIf` does not resolve, and PRTE stops with `LSDUR_PA`

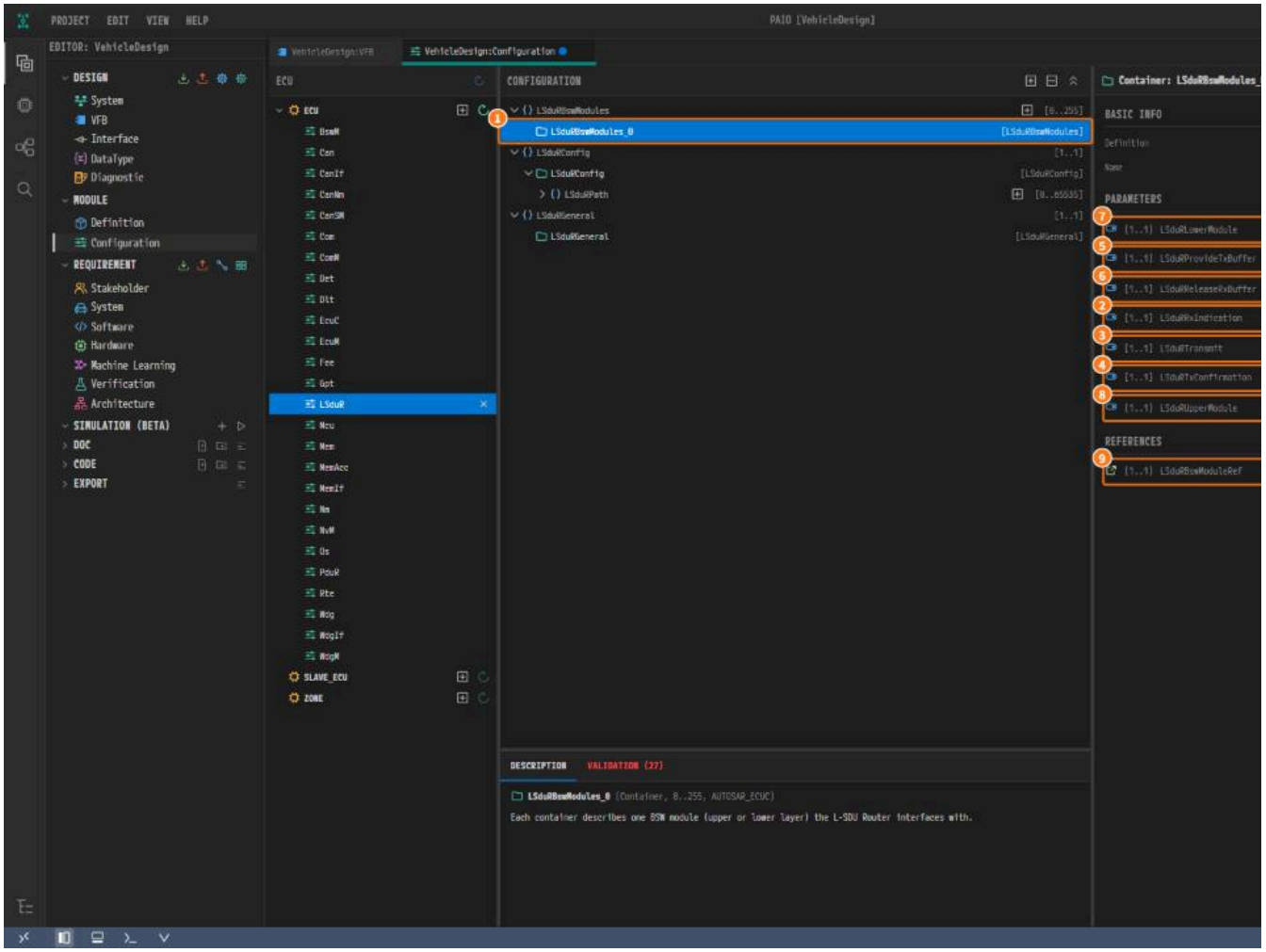


*Figure 9-32. ① `LSduRBswModules_0`, ② **LSduRRxIndication**, ③ **LSduRTransmit**, ④ **LSduRTxConfirmation**, ⑤ **LSduRProvideTxBuffer**, ⑥ **LSduRReleaseRx** **LSduRUpperModule**, ⑨ **LSduRBswModuleRef**.*

7. Click the **+** on the `LSduRBswModules` row again. In `LSduRBswModules_1`, set the same values except **FALSE** of **LSduRLowerModule** and **TRUE** of **LSduR** **LSduRBswModuleRef**.

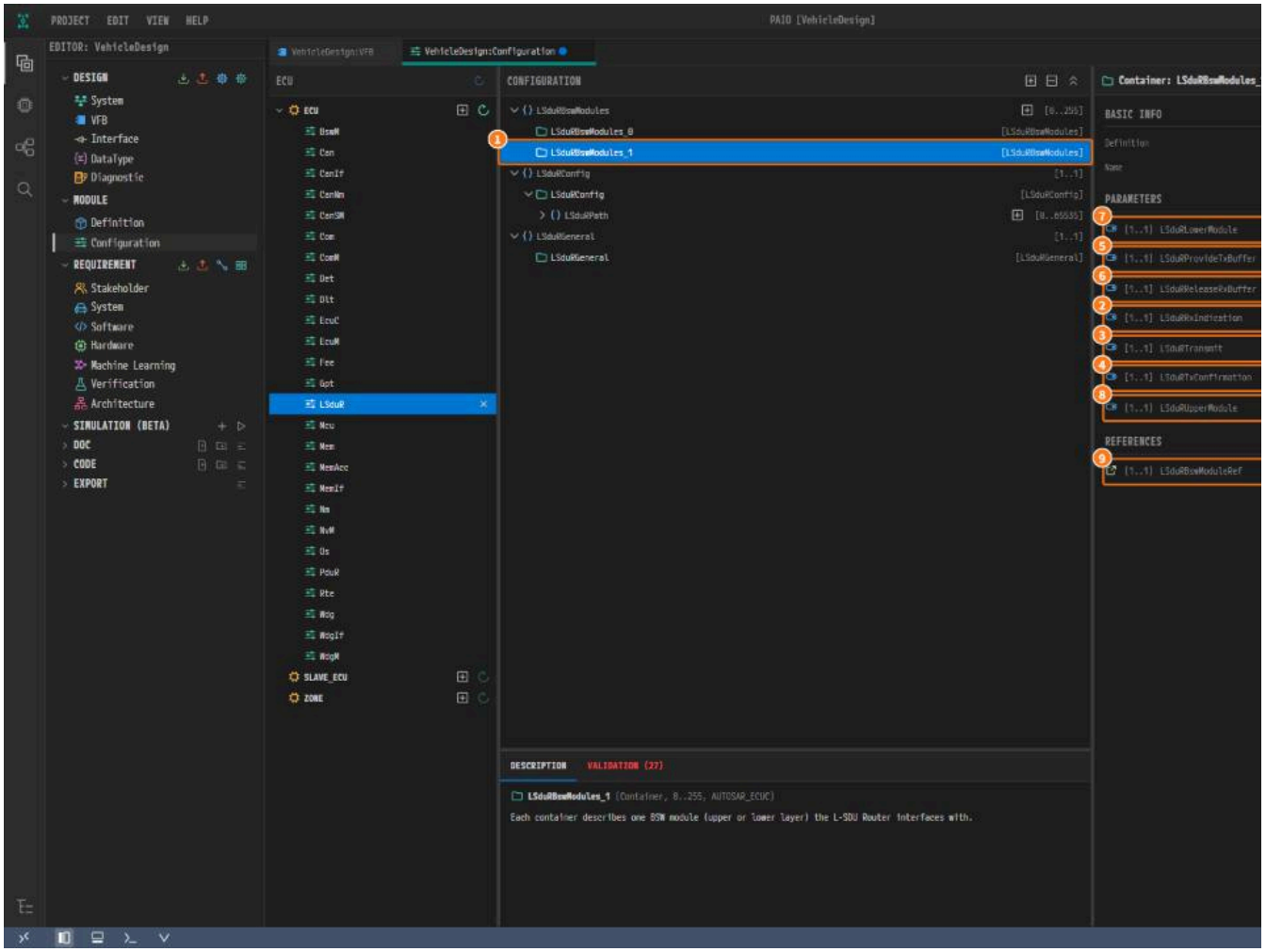


*Figure 9-33. ① `LSduRBswModules_1`, ② **LSduRRxIndication**, ③ **LSduRTransmit**, ④ **LSduRTxConfirmation**, ⑤ **LSduRProvideTxBuffer**, ⑥ **LSduRReleaseRx** **LSduRUpperModule**, ⑨ **LSduRBswModuleRef**.*

8. Under `LSduRConfig`, click the **+** on the `LSduRPath` row. The editor adds `LSduRPath_0`.

9. Under `LSduRPath_0`, click the **+** on the `LSduRPathSourcePdu` row. Type `0` into **LSduRPathSourcePduId** and press the `Tab` key. In **LSduRPathSourceP** `EcuC/EcucConfigSet/EcucPduCollection/PduNmRx`.

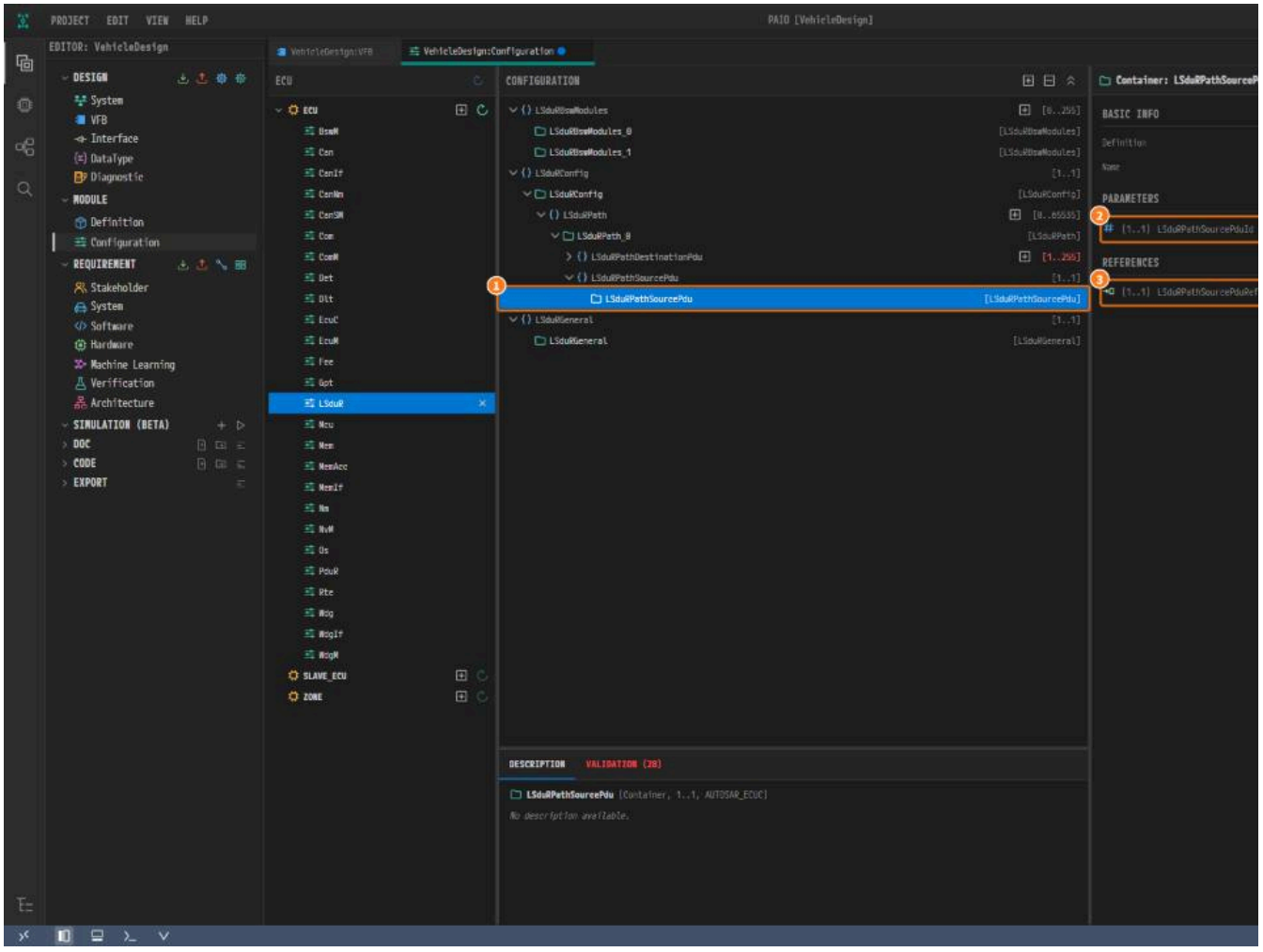


*Figure 9-34. ① `LSduRPathSourcePdu`, ② **LSduRPathSourcePduId**, ③ **LSduRPathSourcePduRef**.*

10. Under `LSduRPath_0`, click the **+** on the `LSduRPathDestinationPdu` row. In **LSduRPathDestinationPduRef** of `LSduRPathDestinationPdu_0`, pick `EcuC`.
11. Create `LSduRPath_1` the same way from the table, with `1` and `PduNmTx`. Save with `Ctrl+S`.

PduR Configuration

1. Click `PduR` in the module list. Click the **+** on the `PduRGeneral` row.

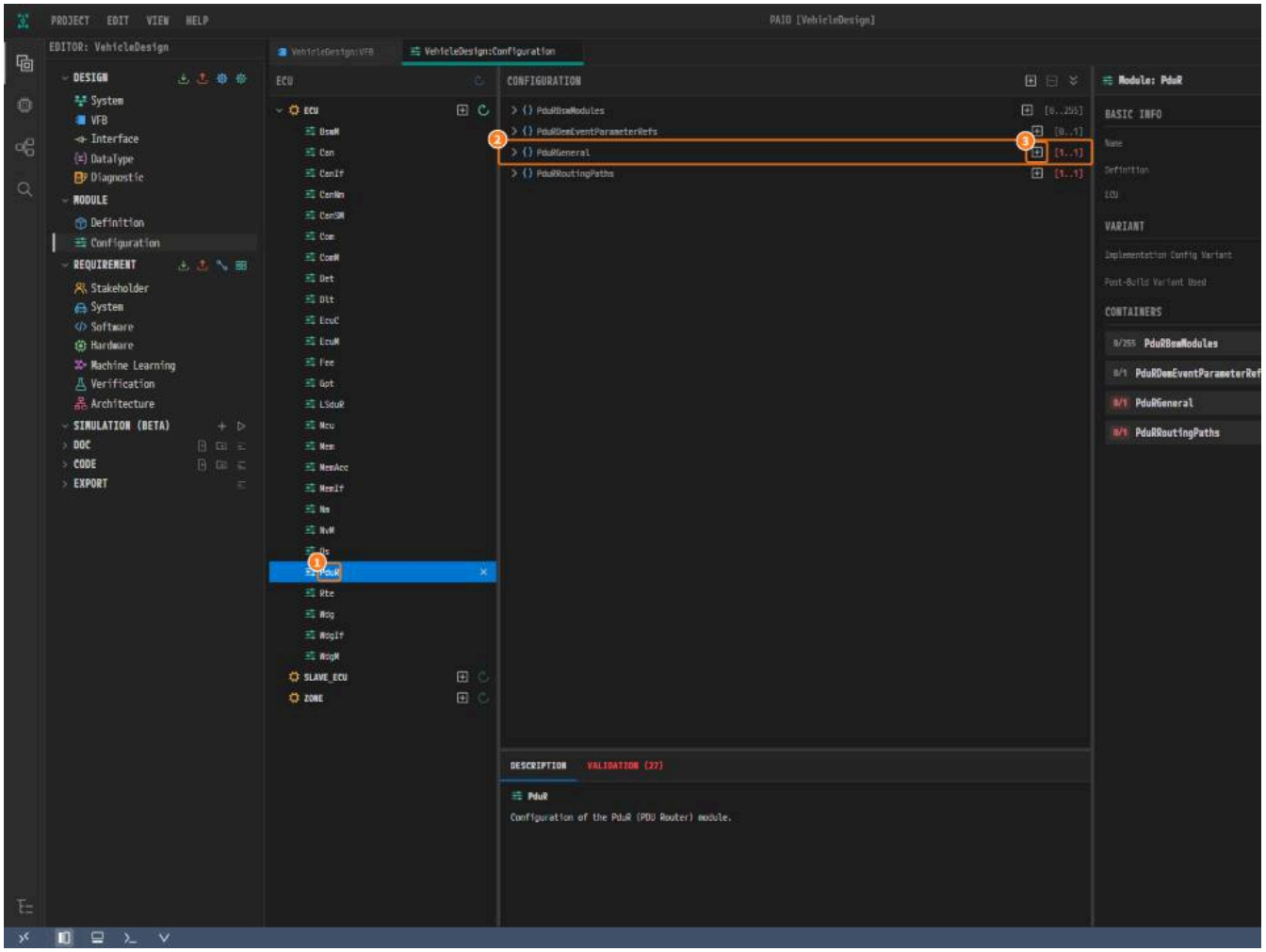


*Figure 9-35. ① `PduR`, ② `PduRGeneral`, ③ **+**.*

2. Click the **+** on the `PduRRoutingPaths` row.

3. Click `PduRGeneral` and click **FALSE** of **PduRZeroCostOperation**.

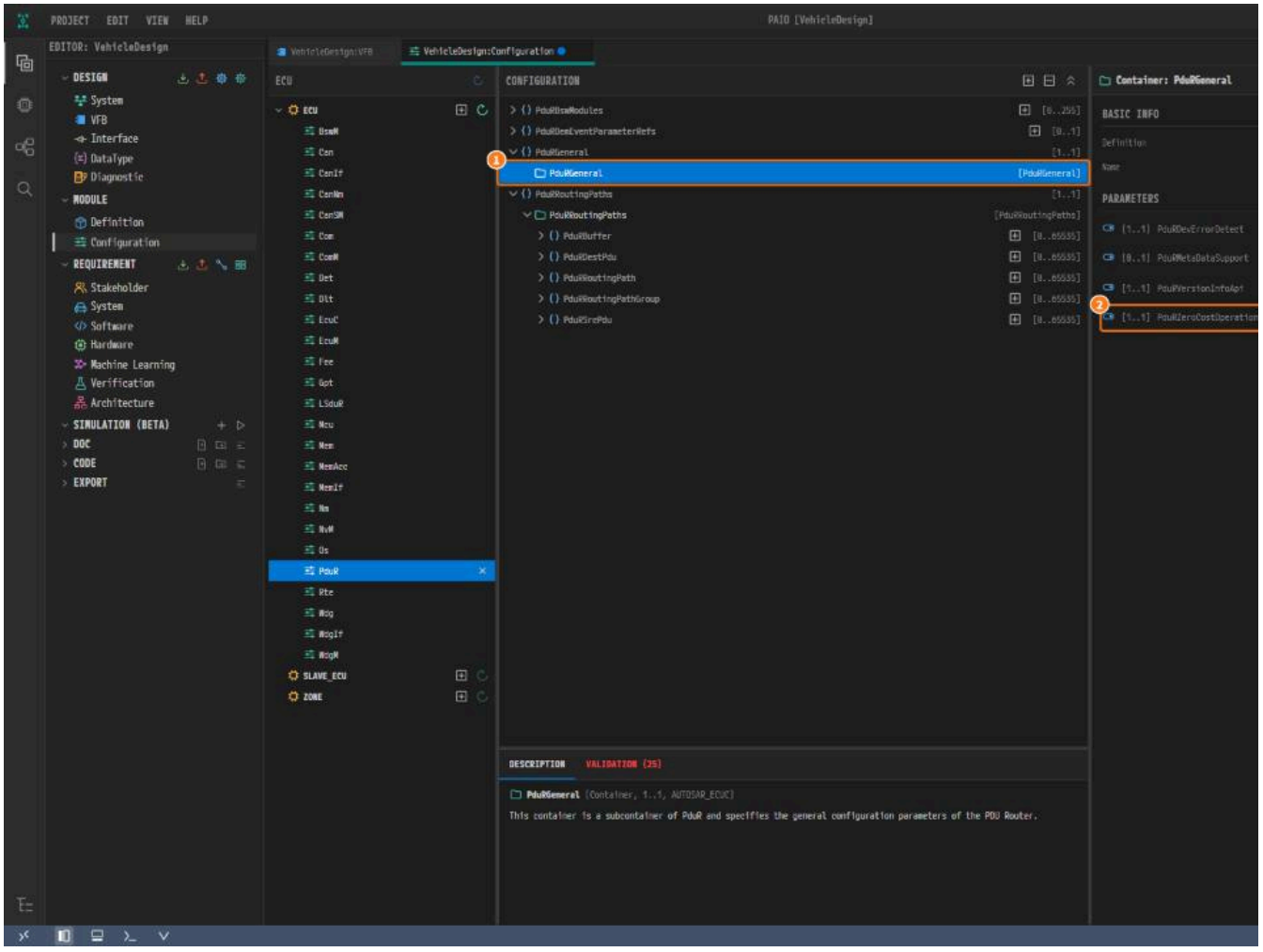


*Figure 9-36. ① `PduRGeneral`, ② **PduRZeroCostOperation**.*

4. Click `PduRRoutingPaths`, type `0` into **PduRConfigurationId** and press the `Tab` key. Save with `Ctrl+S`.

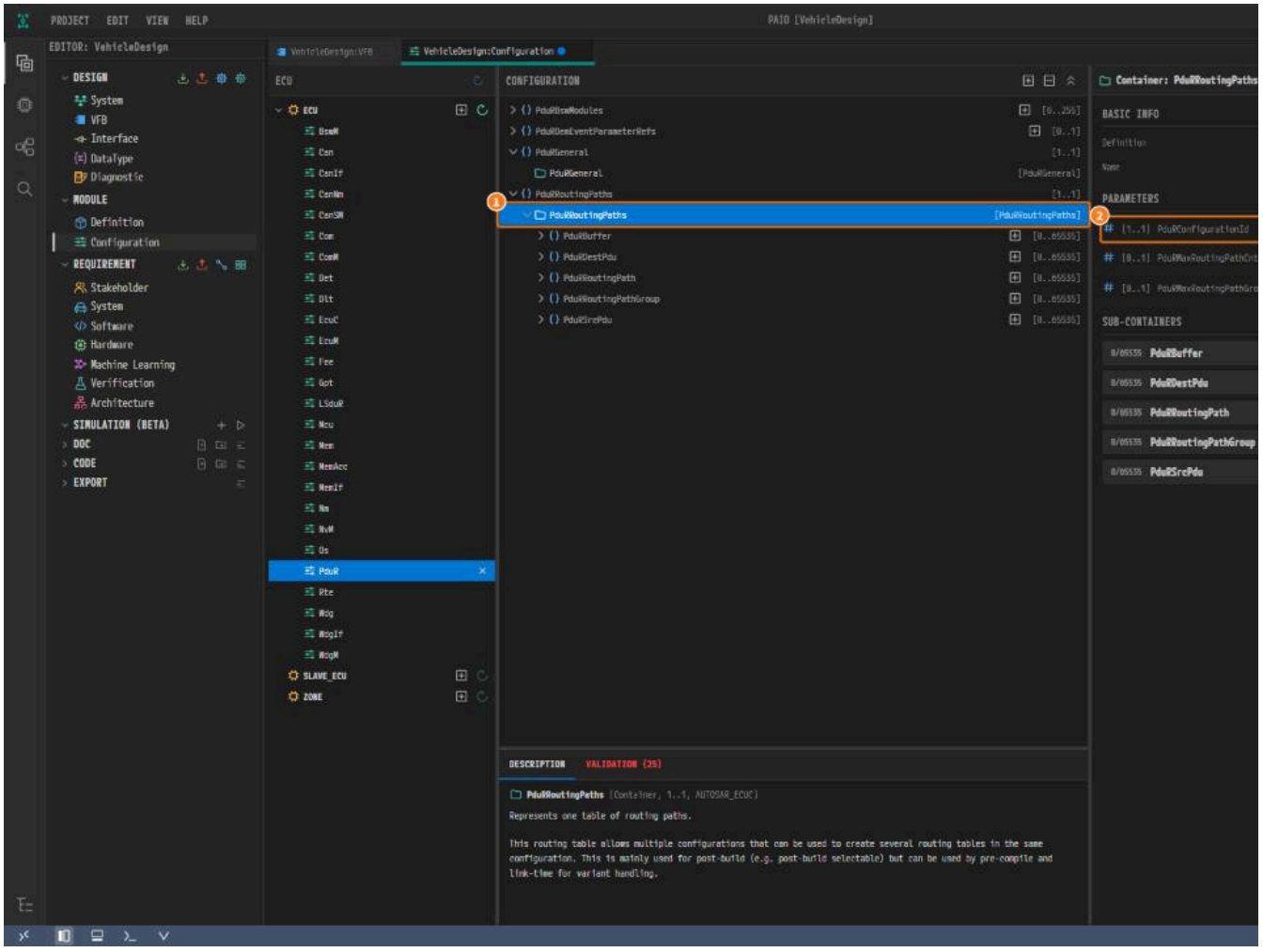


*Figure 9-37. ① `PduRRoutingPaths`, ② **PduRConfigurationId**.*

Com Configuration

1. Click `Com` in the module list. Click the **+** on the `ComGeneral` row.

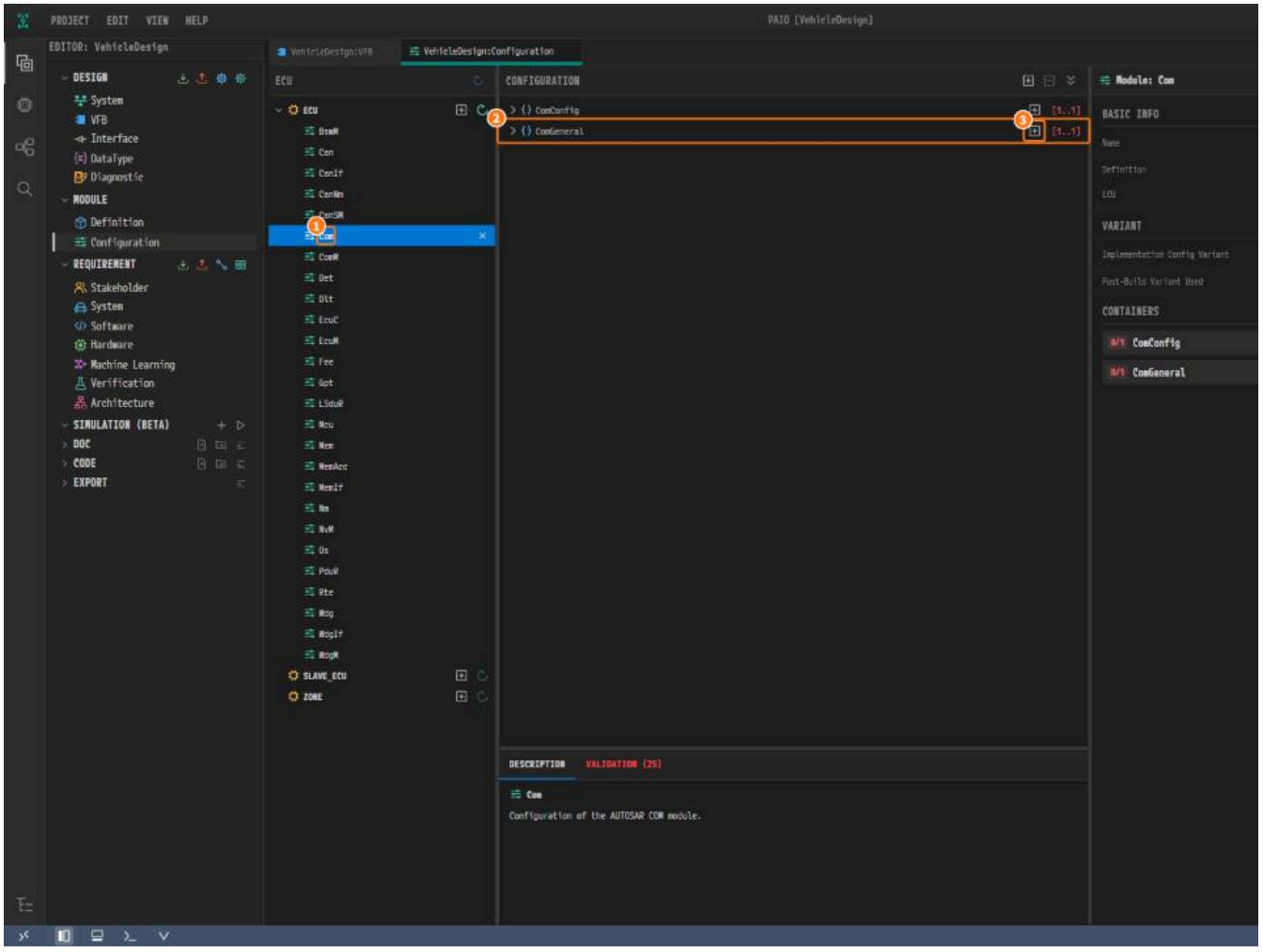


*Figure 9-38. ① `Com`, ② `ComGeneral`, ③ **+**.*

2. Click the **+** on the `ComConfig` row.

3. Click `ComGeneral`, type `0` into **ComSupportedIPduGroups** and press the `Tab` key.

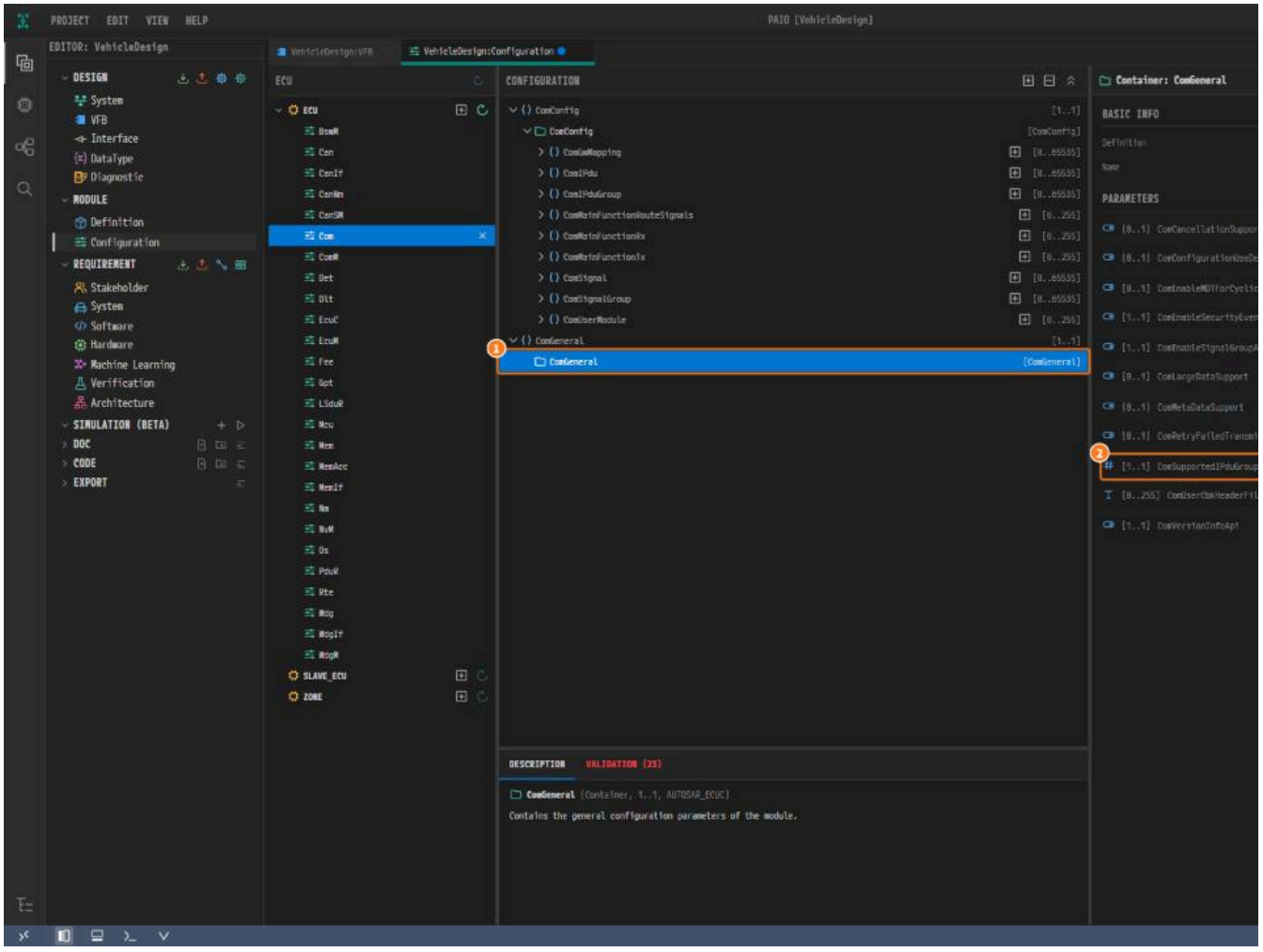


*Figure 9-39. ① `ComGeneral`, ② **ComSupportedIPduGroups**.*

4. Under `ComConfig`, click the **+** on the `ComMainFunctionRx` row.

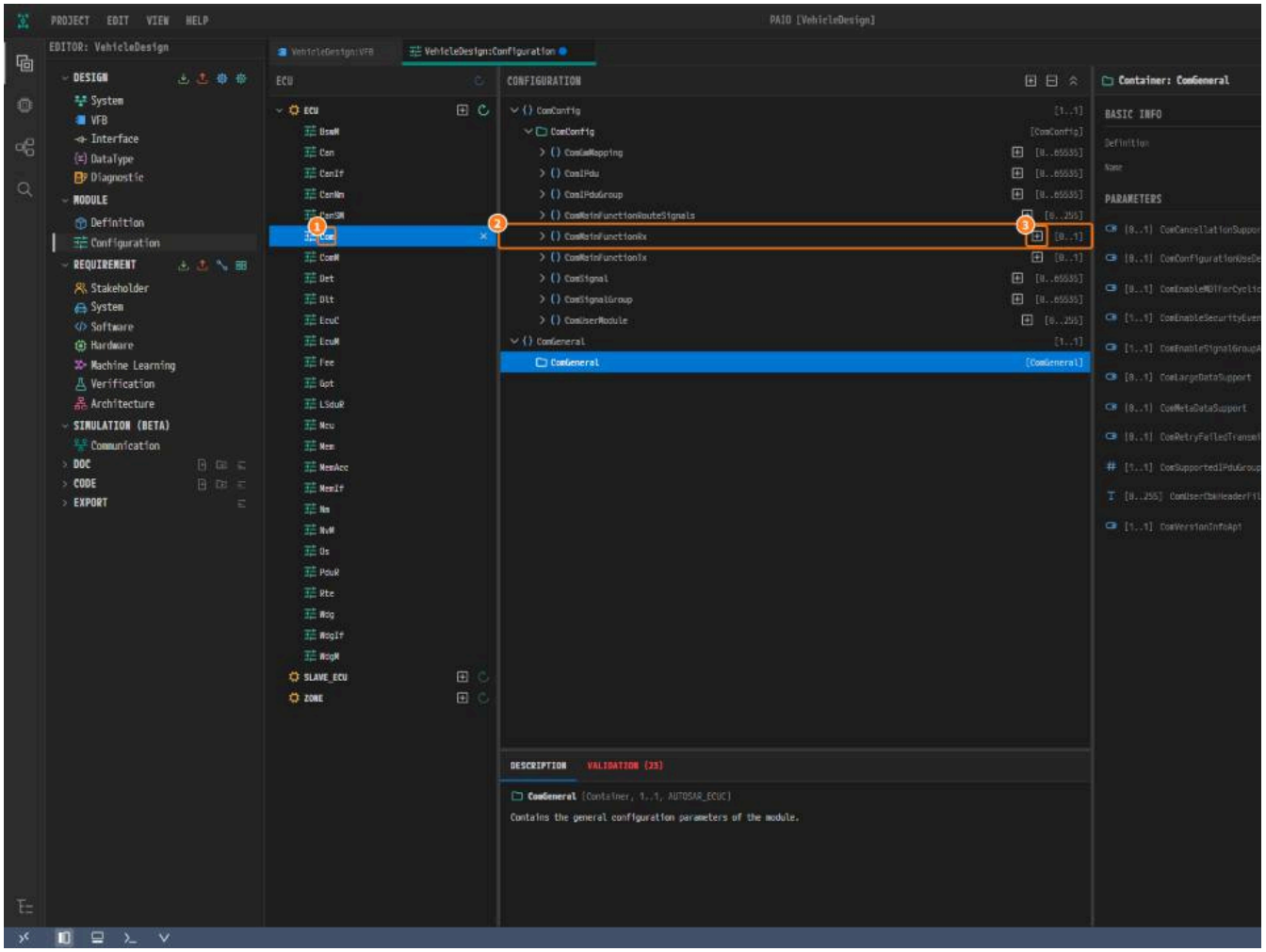


Figure 9-40. ① `Com`, ② `ComMainFunctionRx`, ③ +.

5. Check that **ComMainRxTimeBase** is `0.01`. Keep the name `ComMainFunctionRx`: the Rte mapping of chapter 8 names it.

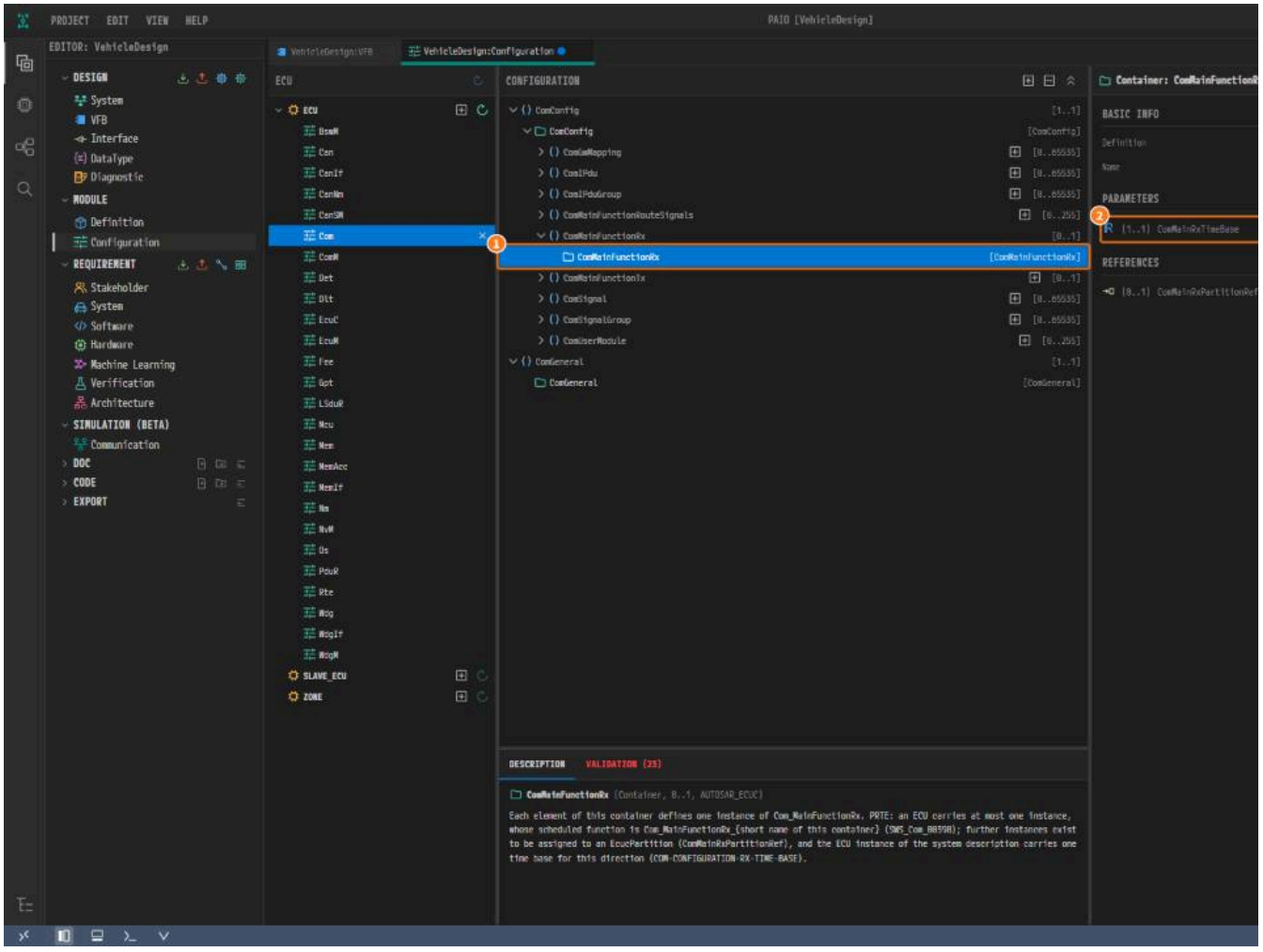


*Figure 9-41. ① `ComMainFunctionRx`, ② **ComMainRxTimeBase**.*

6. Click the **+** on the `ComMainFunctionTx` row the same way and check that **ComMainTxTimeBase** is `0.01`. Save with `Ctrl+S`.

Why Is It Designed This Way?

- **PDUs live in EcuC.** One PDU is used by several modules, so it is defined once in `EcuC` and every module refers to it. `EcuC` therefore comes before `Can`
- **One object per direction.** `CanHardwareObject_0` transmits and `CanHardwareObject_1` receives; `CanIf` reaches them through `CanIfHthCfg_0` and
- **CAN ids in CanIf.** The CAN ids `0x501` and `0x502` of the two PDUs are typed only in `CanIf`.
- **Bus-off reports go to CanSM.** `CAN_SM` in `CanIfDispatchCfg` prepares for `CanSM`, which handles the controller.
- **ComM first.** `CanSM`, `Nm` and `CanNm` all pick `ComMChannel_0`, so the channel must exist before them.
- **Two NM modules.** `Nm` is the layer above the bus; `NM_BUSNM_CANNM` hands its channel to `CanNm`, which carries the NM PDUs.
- **Bus-off values by hand.** Nothing copies the cluster values into `CanSM` in this tutorial, so you type the same numbers yourself.
- **NM only in the ECU configuration.** The System tutorial keeps NM off on `CanCluster` and has no NM frame, so the NM PDUs and their CAN ids `0x501/`
checks them by validation and code generation; how NM behaves on a running bus is not measured.
- **NM PDUs do not pass through PduR.** They go `CanIf` ↔ `LSduR` ↔ `CanNm`, so the NM routes are in `LSduR` and `PduR` routes nothing.
- **One `LSduRBswModules` per neighbour.** `CanIf` is the lower module and `CanNm` the upper one. `LSduR` finds each through **LSduRBswModuleRef**, which
- **One path per NM PDU.** `LSduRPath_0` carries `PduNmRx` and `LSduRPath_1` carries `PduNmTx`. The source and the destination name the same `EcuC` PDU,
- **Two main functions for Com.** `ComMainFunctionRx` and `ComMainFunctionTx` each give `Com` one main function, which the Rte mappings of chapter 8
`RTE_BSWMD_REF_UNRESOLVED`.
- **PduR and Com carry no PDU.** The two SW-Cs talk only inside the ECU, so no application PDU or signal is configured. The modules still belong to the CA

Checklist

- [ ] `EcuC` holds `PduNmTx` and `PduNmRx` with **PduLength** `8`, and `Core0`.
- [ ] `Can` holds `CanGeneral`, `CanController_0` with `CanControllerBaudrateConfig_0`, and the two hardware objects.
- [ ] **CanIfPublicTxBuffering** is **TRUE**.
- [ ] `CanIfTxPduCfg_0` points to `PduNmTx` with `1281`, and `CanIfRxPduCfg_0` to `PduNmRx` with `1282`.

- [ ] **ComMNmVariant** of `ComMNetworkManagement` is `FULL`.
- [ ] `CanSMManagerNetwork_0` holds `2`, `0.1`, `0.2`, `0.1` and points to `ComMChannel_0`.
- [ ] Every **TRUE/FALSE** item of `NmGlobalFeatures`, `NmChannelConfig_0`, `CanNmGlobalConfig` and `CanNmChannelConfig_0` in the tables has a value, exce
- [ ] **CanNmCarWakeUpFilterEnabled** has no value.
- [ ] `CanNmRxPdu_0` points to `PduNmRx` and `CanNmTxPdu` to `PduNmTx`.
- [ ] `LSduR` holds `LSduRGeneral`, `LSduRConfig`, `LSduRBswModules_0` (`ECU/CanIf`) and `LSduRBswModules_1` (`ECU/CanNm`).
- [ ] `LSduRPath_0` names `PduNmRx` and `LSduRPath_1` names `PduNmTx`, each in its source and its destination.
- [ ] `PduR` holds `PduRGeneral` and `PduRRoutingPaths`.
- [ ] `Com` holds `ComGeneral` and `ComConfig` with `ComMainFunctionRx` and `ComMainFunctionTx`.
- [ ] The tab `VehicleDesign:Configuration` carries no modified mark.

Next Step

In Validation and Code Generation you validate the design and generate the code of the ECU.

Validation and Code Generation

What You Will Do

In this chapter you check `No validation issues.` on the **VALIDATION** panel, generate the code of the ECU from the values you entered by hand, and loc

Before You Begin

- Start after finishing Communication Configuration.
- Every module is saved: the tab `VehicleDesign:Configuration` carries no modified mark.

Where	Item	Value
PRTE Code Generation window	**TARGET ECU**	`ECU`

Validating

1. Click the **V** icon in the footer to open the **VALIDATION** panel. It shows `No validation issues.`

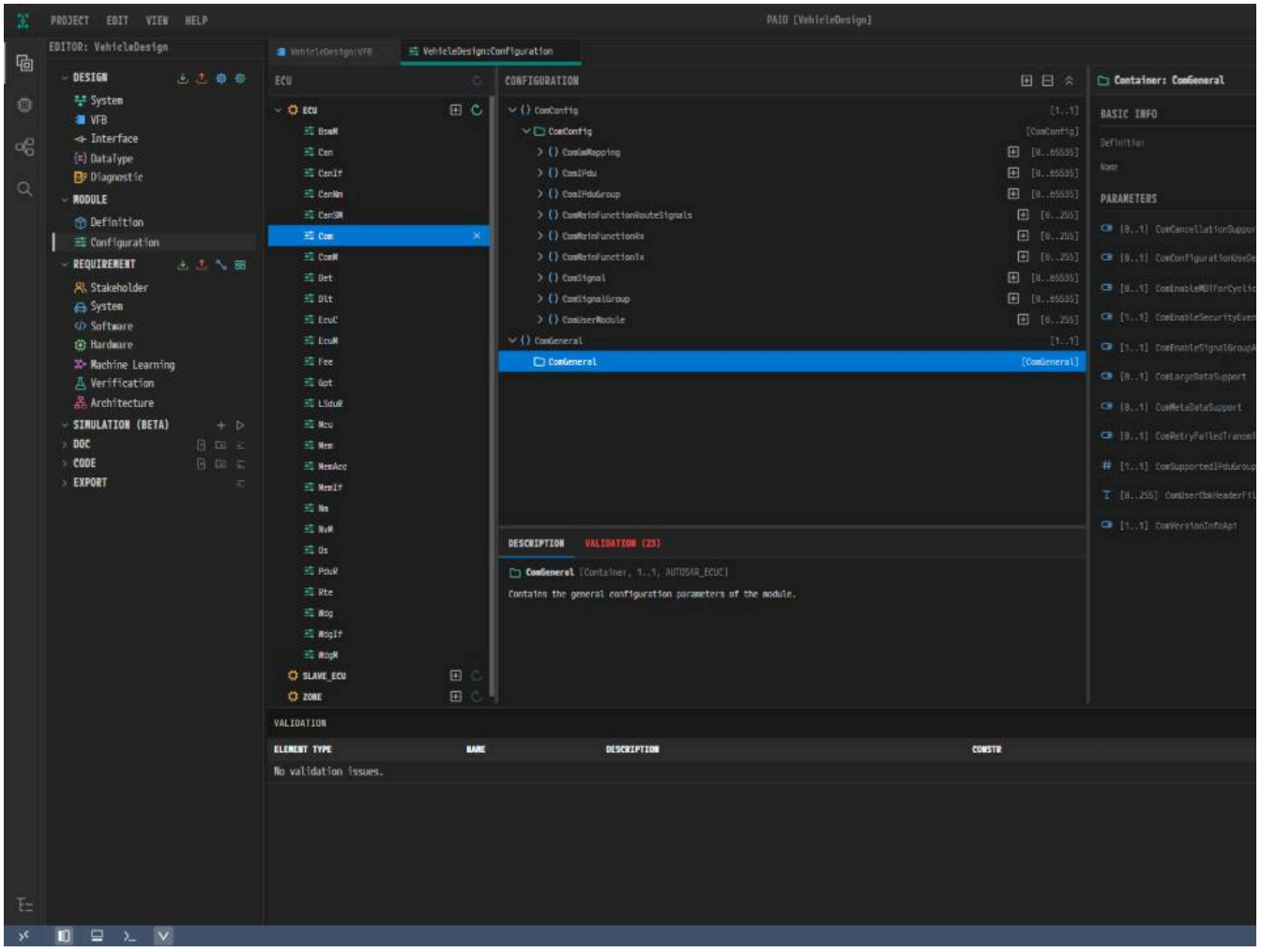


*Figure 10-1. `No validation issues.` The **VALIDATION** tab inside the ECU Configuration editor still shows a count: it lists items such as optional value code can be generated.*

Generating the Code

1. Close the panel. On the **DESIGN** header, click the **PRTE Code Generation** icon.

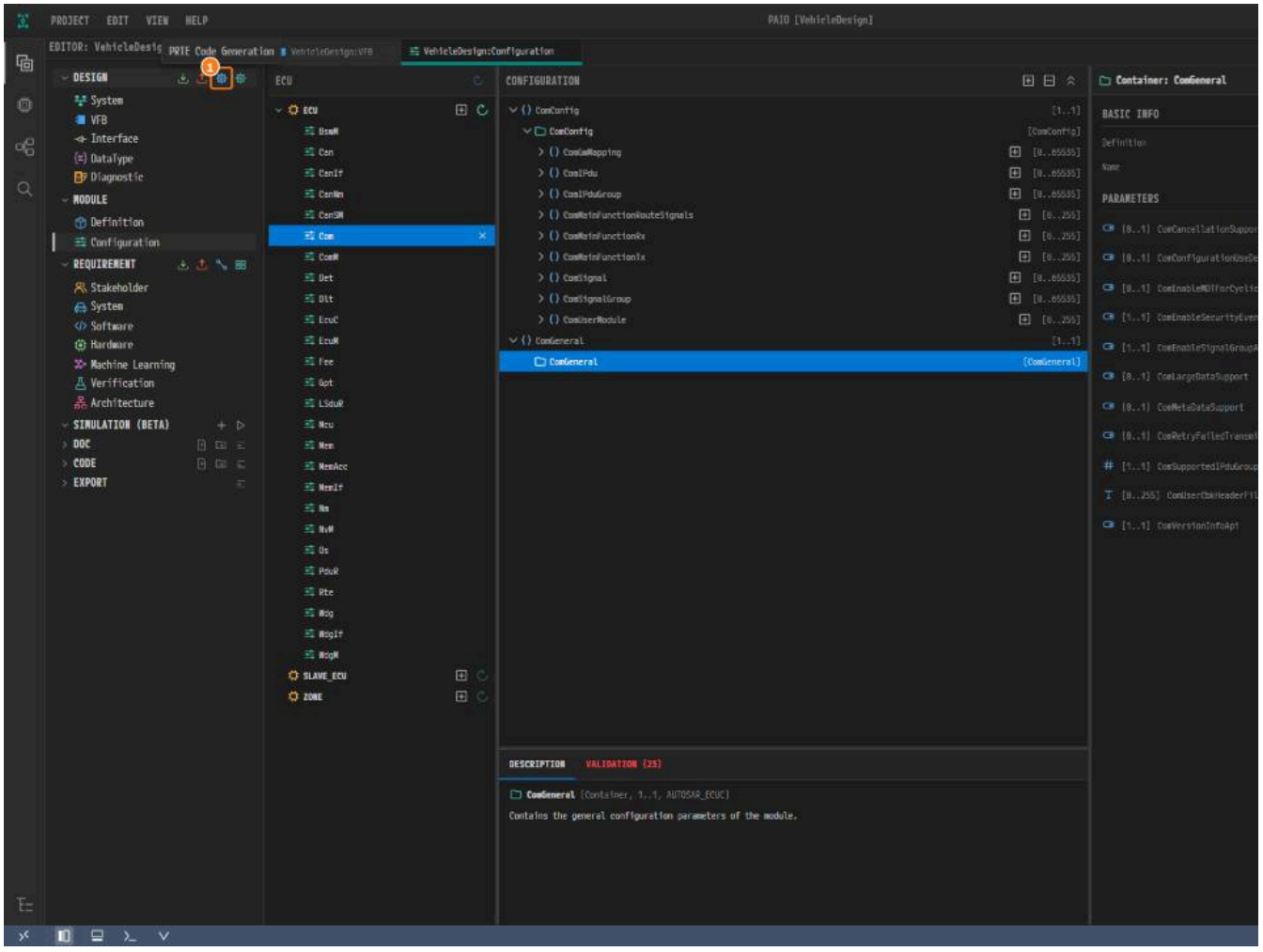


*Figure 10-2. ① **PRTE Code Generation.***

2. Check that **TARGET ECU** reads `ECU` and click **Generate**.

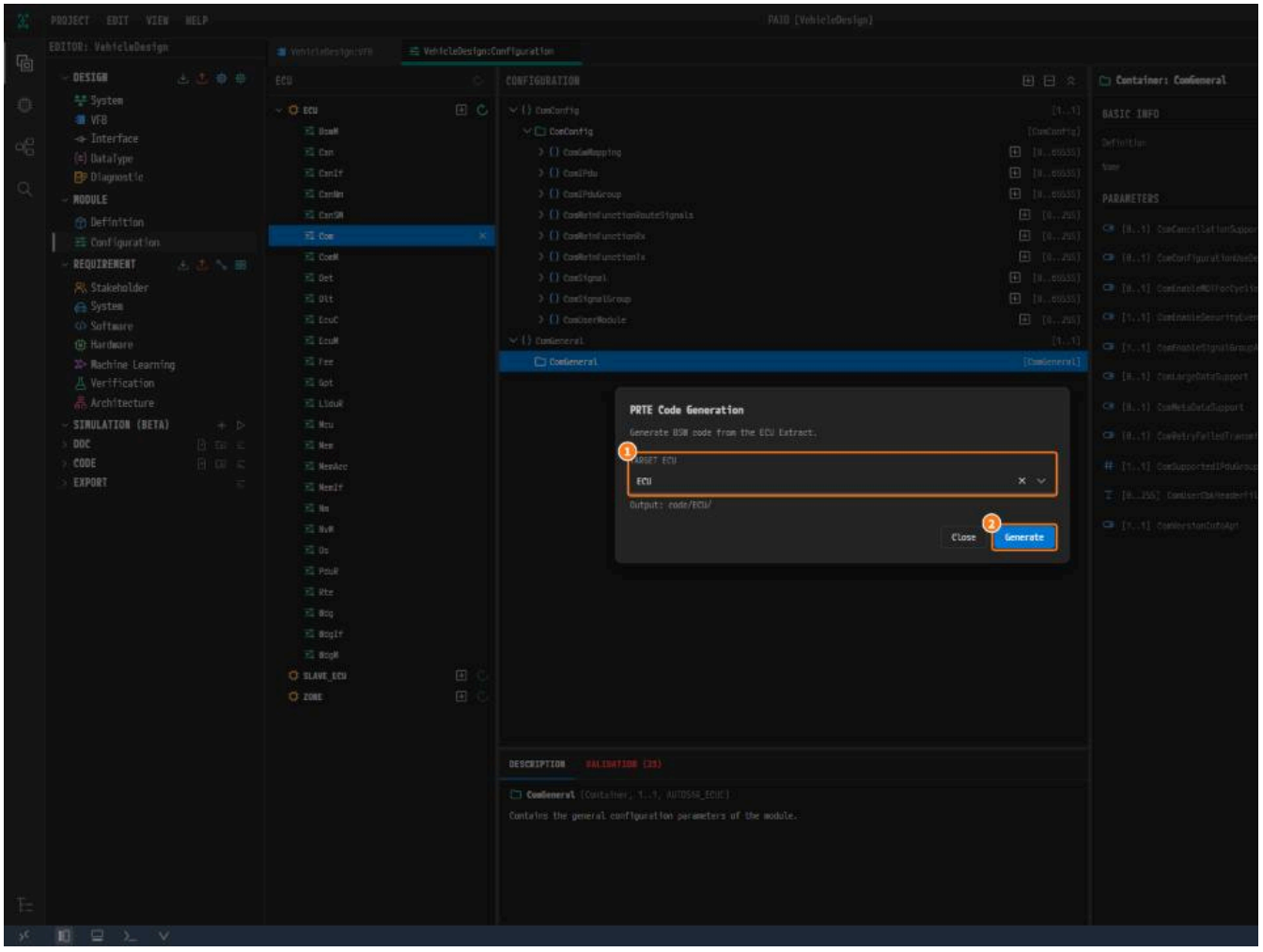


*Figure 10-3. ① **TARGET ECU**, ② **Generate**.*

3. When it ends, the window reads `COMPLETE` and `99 file(s) generated under code/ECU/`, with four `INFO` rows and no warning or error. The `MODULE` `EcuM`, `BswM`, `Rte`, `Os`, `Can`, `CanIf`, `PduR`, `Com`, `ComM`, `CanSM`, `CanNm`, `Nm`, `LSduR`, `Det`. Click **Close**.

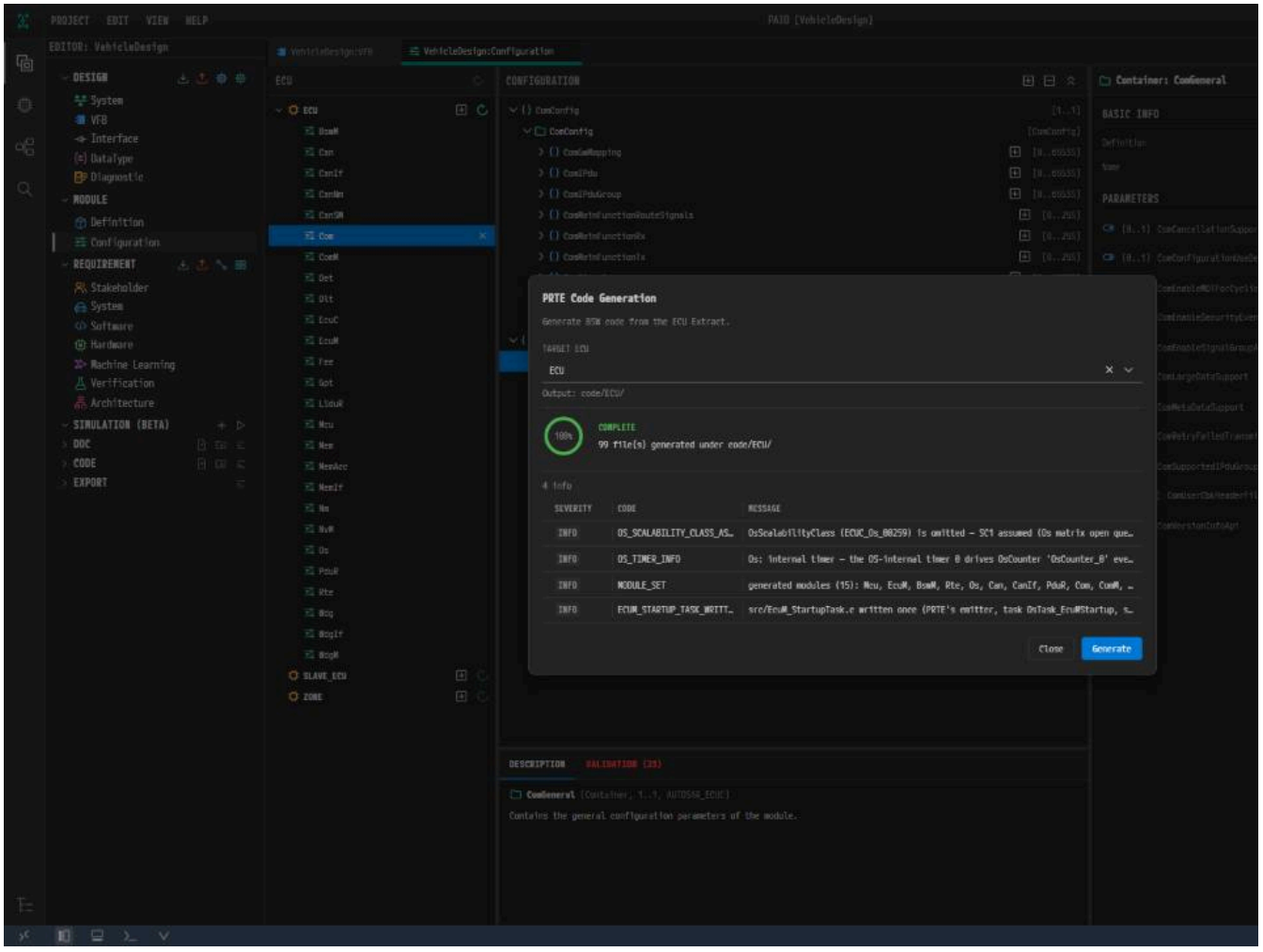


Figure 10-4. The result of the generation.

Looking at the Result

1. Open **CODE** in the **EDITOR** view, then `ECU`.

The folder `code/ECU/` holds:

Path	Content	Who owns it
`gen/`	The generated RTE and BSW configuration	The generator, which rewrites it on eac
`src/`	`SwcSensor.c`, `SwcActuator.c` and the `EcuM` start-up files `EcuM_StartupTask.c` and `EcuM_Callout_Stubs.c`	You. These files are created once and r
`include/`	`SwcSensor.h`, `SwcActuator.h`	You
`CMakeLists.txt`	The build description	You. It is created once

> **Note.** This window may show a **PROBLEMS** count and red marks in the code. The SDK headers such as `Rte.h` are in the PRTE vECU container, not on you reports them as not found, and code that uses their types is marked as well. Check any other kind of mark on its own. This tutorial judges the design b generation. Judging the code itself, by building it, belongs to the vECU.

Why Is It Designed This Way?

- **Every BSW value is yours.** No automatic configuration ran: each value in the generated BSW code is one you entered in chapters 6 to 9 or the defaul exception is what the Rte values file of chapter 6 brought: the six `Rte` references and the SW-C type and instance pairs.
- **The start-up task.** The `ECUM_STARTUP_TASK_WRITTEN` row says `src/EcuM_StartupTask.c` was written for the task `OsTask_EcuMStartup`, which is w
- **The tutorial stops here.** A design that validates and generates is the goal. Writing the runnable bodies and building in the vECU come next, and are y

Checklist

- [ ] The **VALIDATION** panel shows `No validation issues.`
- [ ] The **PRTE Code Generation** window read `COMPLETE` and `99 file(s) generated under code/ECU/`.
- [ ] The `MODULE_SET` row listed 15 modules, and no row was a warning or an error.
- [ ] **CODE** lists `ECU` with `gen`, `include`, `src` and `CMakeLists.txt`.

Next Step

The Appendix collects every value of this design and common problems.

The AUTOSAR AP Tutorial continues with this project: it puts adaptive applications on the node `HPC`. To start the next part from a finished copy of this tu

Appendix

What You Will Do

This appendix is reference material, not a chapter you follow along with. Use it to review your design or to find which value belongs to which chapter.

All Values

Chapters 1 to 3

Chapter	Element	Value
System	Project, ECU node	`VehicleDesign`, `ECU` (from the AUTOSAR System Tutorial)
1	Composition (**NAME**, **AS PROTOTYPE**)	`ECU`, mapped to **ECU [ECU]**
2	Data type	`SpeedType`, **ALIAS**, **BASETYPE** `uint16`
3	Sender-receiver interface	`IfSpeed`, data element `speed` of `SpeedType`
3	Client-server interface	`IfScale`, operation `GetScale`, argument `scale`, `OUT`, `SpeedType`

Chapters 4 and 5

SW-C	Port	Kind	Interface	Setting
`SwcSensor`	`PpSpeed`	**PPort**	`IfSpeed`	**NON-QUEUED**
`SwcSensor`	`RpScale`	**RPort**	`IfScale`	**CLIENT**
`SwcActuator`	`RpSpeed`	**RPort**	`IfSpeed`	**NON-QUEUED**
`SwcActuator`	`PpScale`	**PPort**	`IfScale`	**SERVER**

Runnable	Access point	RTE event
`Sensor_Cyclic`	`Rte_Write_PpSpeed_speed`, `Rte_Call_RpScale_GetScale`	**TimingEvent** `SensorCyclicTiming`, `0.02` `s`
`Actuator_Init`		**InitEvent** `ActuatorInitEvent`
`Actuator_OnData`	`Rte_Read_RpSpeed_speed`	**DataReceivedEvent** `ActuatorOnDataEvent`, `RpSpeed` / `s`
`Actuator_GetScale` (**TRUE** under **CAN BE INVOKED**)		**OperationInvokedEvent** `ActuatorGetScaleEvent`, `PpSca`

Connectors: `Assembly_SwcSensor_PpSpeed_SwcActuator_RpSpeed`, `Assembly_SwcActuator_PpScale_SwcSensor_RpScale`.

Chapter 6

Element	Value
BSW modules	The suggested selection of **PRTE Platform Import**, with the CAN stack `Can`, `CanIf`, `CanSM`, `ComM`, `Com`, `PduR` and `LSduR`, plus `CanNm` and `Nm`, whi
ECU Configuration [ECU] window	Closed with `Esc`, not applied
`EcuM`, `BswM`, `Mcu`, `Det`	Entered by hand from the tables of PRTE Setup; among them **BswMNmEnabled** `TRUE` in `BswMGeneral` and `EcuMDriverInitListOne` with `McuIni` `PduRInit`, `ComInit`, `CanNmInit`, `NmInit`, `ComMInit` and `CanSMInit`
Rte values file	VehicleDesign_Rte.arxml, **All → Use Imported** on `Rte` (`1 replaced`)
`Rte` from the Rte values file	The SW-C type and instance pairs, `BswMInstance/RteEventToTaskMapping_0` (**RteEventRef** to `BswMPollTiming`), `RteBswModuleInstance_0`, `_3`

Chapter 7

`Os` container	Value
`OsOS`	**OsStatus** `EXTENDED`; **OsStackMonitoring**, **OsUseGetServiceId**, **OsUseParameterAccess** TRUE
`OsOS/OsHooks`	The five hooks **FALSE**
`OSDEFAULTAPPMODE` (`OsAppMode`)	No value
`RES_AUTOSAR_ECUM` (`OsResource`)	**OsResourceProperty** `STANDARD`
`OsCounter_0`	**OsCounterMaxAllowedValue** `1000000`, **OsCounterMinCycle** `1`, **OsCounterTicksPerBase** `1`, **OsCounterType** `HARDWARE`, **OsSecondsPerTick** `0`

Every task has **OsTaskActivation** `1`, **OsTaskSchedule** `FULL` and **OsTaskStackSize** `32768`. Every alarm has **OsAlarmCounterRef** `Os/OsCounter_0`.

Task	OsTaskPeriod	OsTaskPriority	Activated by
`OsTask_0`	`0.01`	`2`	`OsAlarm_0`
`OsTask_EcuMStartup`	Empty	`3`	**OsTaskAutostart** in `Os/OSDEFAULTAPPMODE`
`OsTask_2`	`0.02`	`1`	`OsAlarm_1`
`OsTask_3`	`0.01`	`1`	`OsAlarm_2`

Chapter 8

`Rte` container	Value
`RteGeneration`	**RteCodeVendorId** `79`
`RteBswGeneral`	Added, no value
`RteInitializationBehavior_0`	**RteInitializationStrategy** `RTE_INITIALIZATION_STRATEGY_AT_DATA_DECLARATION`, **RteSectionInitializationPolicy** `INIT`
`EcuMInstance/RteModeMachineInstanceConfig_0`	**RteModeMachineQueueLength** `1`, **RteSwcModeManagerRef** `PortInterface/ModeSwitch/EcuM_CurrentMode/current` `SwComponentType/Atomic/Service/EcuM/currentMode`
`RteOsInteraction/RteUsedOsActivation_0`, `_1`, `_2`	**RteExpectedActivationOffset** `0`, **RteExpectedTickDuration** `0.01`, **RteActivationOsAlarmRef** `Os/OsAlarm_0`, `Os/OsAlar`
`RteBswModuleInstance_0` to `_7`	**RteBswImplementationRef** of `BswM`, `EcuM`, `Can`, `CanIf`, `CanSM`, `ComM`, `Com`, `CanNm` in this order; `_1`, `_2` and `_7` added **RteExclusiveAreaImplMechanism** `ALL_INTERRUPT_BLOCKING`. `Nm` has no instance

Every mapping has its mapped-to-task flag **TRUE**. `ActuatorGetScaleEvent` is not mapped.

Event	Task	Alarm	Position
`SensorCyclicTiming`	`Os/OsTask_2`	`Os/OsAlarm_1`	-
`ActuatorInitEvent`	`Os/OsTask_3`	-	`2`
`ActuatorOnDataEvent`	`Os/OsTask_3`	-	`1`
`BswMPollTiming` (Rte values file)	`Os/OsTask_3`	`Os/OsAlarm_2`	`0`
The twelve BSW main function events of `BswM`, `EcuM`, `Can` (five), `CanSM`, `ComM`, `Com` (two), `CanNm`	`Os/OsTask_0`	`Os/OsAlarm_0`	`2` to `13`

Chapter 9

Module	Value
`EcuC`	`EcucConfigSet/EcucPduCollection` (**PduIdTypeEnum** `UINT16`, **PduLengthTypeEnum** `UINT32`) with `PduNmTx` and `PduNmRx` (**PduLength** `8`), `EcucHardware/Core0` (**I**
`Can`	`CanGeneral` (**CanIndex** `0`, **CanMainFunctionModePeriod** `0.01`, **CanTimeoutDuration** `0.001`), `CanConfigSet/CanController_0` (**CanControllerId** `0`, processing `POLL` `Mcu/McuModuleConfiguration/Default/CoreClock`), `CanControllerBaudrateConfig_0` (**CanControllerBaudRate** `500`), `CanHardwareObject_0` (`TRANSMIT`) and `Ca`
`CanIf`	`CanIfPublicCfg` (**CanIfPublicHandleTypeEnum** `UINT16`, **CanIfPublicTxConfirmPollingSupport** and **CanIfPublicTxBuffering TRUE**), `CanIfPrivateCfg` (`LINEAR`), `CanIf` (`CanIf_Config`) with `CanIfInitHohCfg_0`, `CanIfCtrlDrvCfg_0` with `CanIfCtrlCfg_0` (→ `CanController_0`)
`CanIf`	`CanIfHthCfg_0` (→ `CanHardwareObject_0`), `CanIfHrhCfg_0` (→ `CanHardwareObject_1`), `CanIfBufferCfg_0` (**CanIfBufferSize** `1`, → `CanIfHthCfg_0`), `CanIfTxPdu` (`1282` = `0x502`, → `PduNmRx`)
`ComM`	`ComMGeneral`, `ComMConfigSet/ComMChannel_0` (`COMM_BUS_TYPE_CAN`, **ComMChannelId** `0`), `ComMNetworkManagement` (**ComMNmVariant** `FULL`)
`CanSM`	`CanSMGeneral` (`0.01`), `CanSMConfiguration` (`3`, `0.02`), `CanSMManagerNetwork_0` (`2`, `0.1`, `0.2`, `0.1`, **TRUE**, → `ComMChannel_0`), `CanSMController_0` (→ `CanIfCt`
`Nm`	`NmGlobalConfig` with `NmGlobalConstants` (**NmNumberOfChannels** `1`), `NmGlobalFeatures` (eight items **FALSE**), `NmGlobalProperties` (**NmCycletimeMainFunction** `ComMChannel_0`) with `NmBusType/NmStandardBusNmConfig` (`NM_BUSNM_CANNM`)
`CanNm`	`CanNmGlobalConfig` (thirteen items **FALSE**, **CanNmMainFunctionPeriod** `0.01`), `CanNmChannelConfig_0` (**CanNmMsgCycleTime** `0.1`, **CanNmTimeoutTime** `1.0`, **CanNm** **CanNmCarWakeUpFilterEnabled** no value, → `ComMChannel_0`), `CanNmRxPdu_0` (→ `PduNmRx`), `CanNmTxPdu` (→ `PduNmTx`)
`LSduR`	`LSduRGeneral` (**LSduRMetaDataSupport**, **LSduRZeroCostOperation FALSE**), `LSduRConfig`
`LSduR`	`LSduRBswModules_0` (`ECU/CanIf`, **LSduRLowerModule TRUE**), `LSduRBswModules_1` (`ECU/CanNm`, **LSduRUpperModule TRUE**); both **TRUE** on **LSduRRxIndication**, **LSduR**
`LSduR`	`LSduRPath_0` (`0`, source and destination `PduNmRx`), `LSduRPath_1` (`1`, source and destination `PduNmTx`)
`PduR`	`PduRGeneral` (**PduRZeroCostOperation FALSE**), `PduRRoutingPaths` (**PduRConfigurationId** `0`)
`Com`	`ComGeneral` (**ComSupportedIPduGroups** `0`), `ComConfig` with `ComMainFunctionRx` (**ComMainRxTimeBase** `0.01`) and `ComMainFunctionTx` (**ComMainTxTimeBase** `0.`

The full lists of items are in each chapter's tables. Network management lives only in the ECU configuration: the System tutorial keeps NM off on `CanClus` their CAN ids exist only in `EcuC` and `CanIf` (Communication Configuration).

Generated Code

Validation and Code Generation ends with `COMPLETE` and `99 file(s) generated under code/ECU/`, four `INFO` rows and no warning or error. The `MODULI` `Rte`, `Os`, `Can`, `CanIf`, `PduR`, `Com`, `ComM`, `CanSM`, `CanNm`, `Nm`, `LSduR`, `Det`.

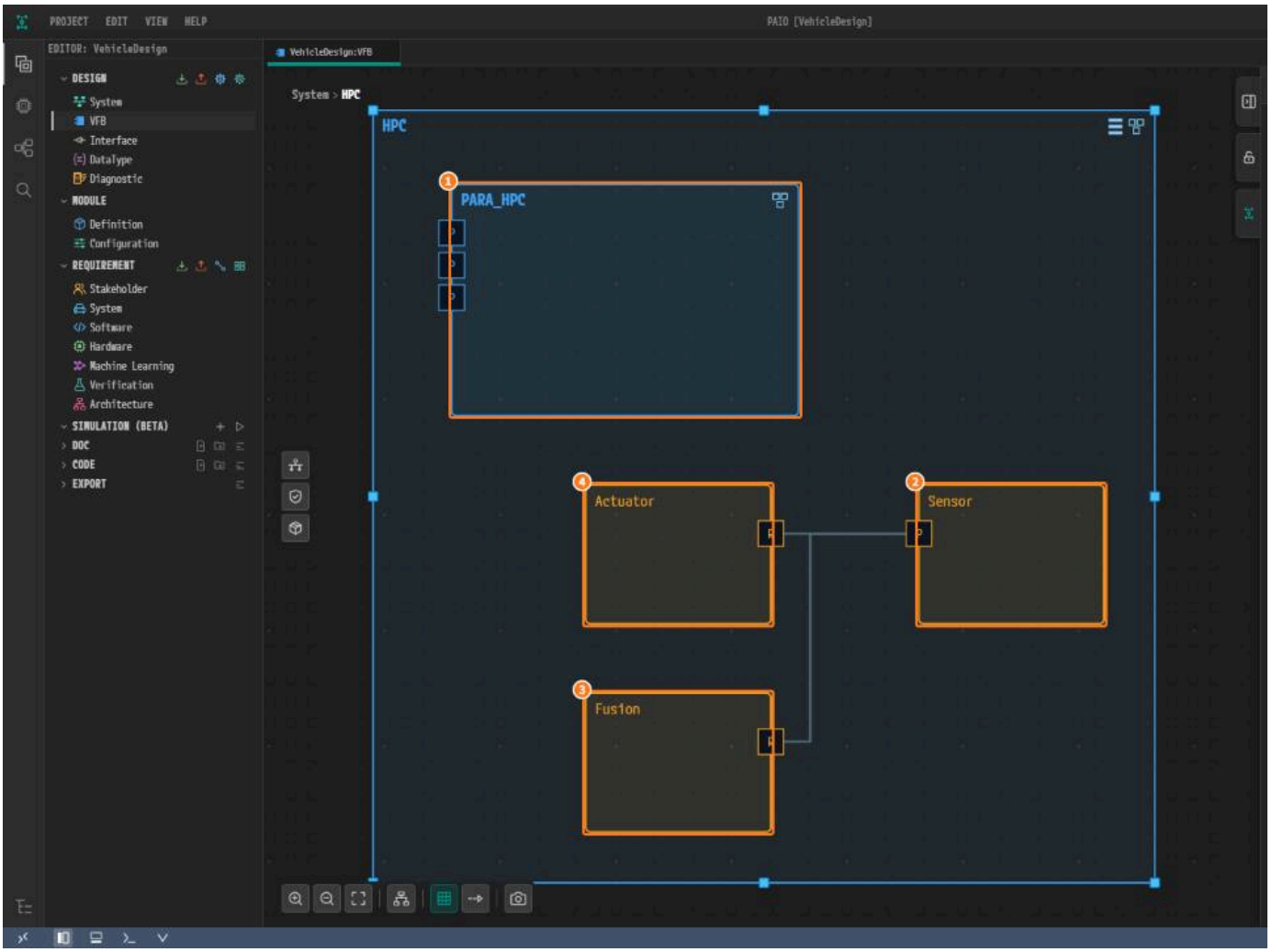
Path under `code/ECU/`	Content	Who owns it
`gen/`	The generated RTE and BSW configuration	The generator, which rewrites it c
`src/`	`SwcSensor.c`, `SwcActuator.c` and the `EcuM` start-up files `EcuM_StartupTask.c` and `EcuM_Callout_Stubs.c`	You. Created once, never overwrit
`include/`	`SwcSensor.h`, `SwcActuator.h`	You
`CMakeLists.txt`	The build description	You. Created once

Common Problems

Symptom	Cause
The ECU icon is missing inside the composition	The composition is not mapped to a node
BSW values you entered were overwritten	**PRTE AUTO CONFIGURATION**, the **PRTE Auto Configuration** icon or **Auto-Generate** was pressed, or the **ECU Configuration** window was applied with `Ctrl+S`
Generation stops with `MANDATORY_MODULE_MISSING`	One of `Os`, `Rte`, `EcuM`, `BswM`, `Mcu` has no configuration
The `EcuM` of PRTE Setup does not find its application mode or resource	The names are not exactly `OSDEFAULTAPPMODE` and `RES_AUTOSAR_ECUM`
`src/EcuM_StartupTask.c` is not written	The start-up task is not named exactly `OsTask_EcuMStartup`
Generation stops with `RTE_EVENT_MAPPING_TASK_FLAG_MISMATCH` or `RTE_BSW_EVENT_MAPPING_TASK_FLAG_MISMATCH`	**RteEventIsMappedToTask** or **RteBswEventIsMappedToTask** is still the `FALSE` that "+" wrot
Generation stops with `RTE_BSW_INSTANCE_MISSING`	`CanNm` has a main function but no `RteBswModuleInstance_7`
Generation stops with `RTE_CS_SERVER_UNMAPPED_NOT_CONCURRENT`	`Actuator_GetScale` is not set to **TRUE**
`EcuC` holds the PDUs but no `Core0`	`EcuC` needs the core `Core0` (**EcucCoreId** `0`) as soon as it holds anything
Generation stops with `CANIF_BUFFERING_GATE_INCONSISTENT`	**CanIfPublicTxBuffering** is not **TRUE**, while the transmit PDU uses `CanIfBufferCfg_0`
A **TRUE** or **FALSE** value is empty and its row has a red border	The button was already selected, and clicking it again cleared the value
ComMNmVariant is not `FULL`	This ECU runs `CanNm`, so it must stay the `FULL` that "+" writes
A reference to `ComMChannel_0` in `CanSM`, `Nm` or `CanNm` has nothing to pick	`ComM` was not configured first
Generation stops on an empty **TRUE**/**FALSE** item of `Nm` or `CanNm`	Many of these items have no default value, so "+" leaves them empty
Generation stops with `CANNM_CARWAKEUP_LAYOUT`	**CanNmCarWakeUpFilterEnabled** still holds the value "+" wrote
Generation stops with `LSDUR_PATH_SOURCE_UNRESOLVED`	**LSduRBswModuleRef** was typed instead of picked from the list
Generated code is marked with a header not found, such as `Rte.h`	The SDK headers are in the vECU, not on your computer, so a window on your computer cannot resolve them

AUTOSAR AP Tutorial

This tutorial puts three Adaptive Applications on the HPC node `HPC` of the project `VehicleDesign` in PAIO, connects them with the SOME/IP service `Sens` design and generates the code of the machine `HPC`. It is the last of three parts on one project: AUTOSAR System → AUTOSAR CP → AUTOSAR AP.



The software you build: `Sensor`, `Fusion` and `Actuator` beside the platform `PARA_HPC` in the composition `HPC`, with the service `Sensor` going from `Sen`

What You Build

`Sensor` provides the service `Sensor` over SOME/IP (service ID `11`), and `Fusion` and `Actuator` consume it. The three processes start in the function grou running, and each one logs over DLT. The eleven platform processes, `EM` and `CM` among them, come from PARA.

Adaptive Application	Role	DLT application ID
`Sensor`	Provides the service `Sensor`	`SENS`
`Fusion`	Consumes the service `Sensor`	`FUSN`
`Actuator`	Consumes the service `Sensor`	`ACTA`

What You Need

- PAIO, installed and able to run.
- The project `VehicleDesign` finished in the AUTOSAR CP Tutorial. To skip the earlier tutorials, download VehicleDesign-cp.zip, unpack it and open the **PROJECT** menu.
- A screen of 1920×1080 or larger. The dialogs are wide.
- A rough idea of Adaptive Applications, service interfaces and SOME/IP. Each chapter shows what to enter on which screen.

Chapters

Chapter	What you do	
1. The Composition HPC and the Machine	Create the composition `HPC`, map it to the node `HPC` and save the machine	VFB, Machine C
2. Configuring the Machine	Set the OS, the cores, the DLT ECU ID and the State Management value	Machine Config
3. Topology — SOME/IP Discovery Port and UDP Endpoints	Add the SD port and three UDP endpoints to `EthernetConnector`	System editor
4. Data Types	Create `ByteArray` and `String`	DataType
5. The Service Interface	Create the service interface `Sensor`	Interface
6. SOME/IP Deployment and Service Instances	Deploy `Sensor` over SOME/IP and create one provider and two consumers	Interface
7. Binding Service Instances to the Connector	Bind the three instances to `EthernetConnector` with their UDP ports	System editor
8. Adaptive Applications and Ports	Create the three Adaptive Applications, their ports and two connectors	VFB
9. PARA Setup	Bring in the platform applications with PARA Platform Import and disable Network Management	Machine Config
10. Executables and Processes	Give each application an executable and a process with its startup states	AA Internal Beh
11. Service Instance Mapping	Map the three ports to the service instances	AA Internal Beh
12. Logging and DLT	Set the DLT application of each process	AA Internal Beh
13. Validation and Code Generation	Validate the design and generate the code of `HPC` with PARA Code Generation	VALIDATION pan
Appendix	Design checklist, value tables, file layout, common problems	—

How to Read This Tutorial

- **Follow the chapters in order.** Each one starts from the result of the previous one.
- **Repetitive work is shown once.** One item is shown as steps; the rest are given in value tables that you enter the same way.
- UI labels are written in **bold** as the screen shows them, and model element names and values in `code`.

Next Step

Start with The Composition HPC and the Machine.

The Composition HPC and the Machine

What You Will Do

In this chapter you create the composition `HPC` in the VFB editor and map it onto the HPC node `HPC`. Then you open the **Machine Configuration** dialog or

Before You Begin

- Start from the project `VehicleDesign` finished in the AUTOSAR CP Tutorial. If you skipped it, download VehicleDesign-cp.zip, unpack it and open the **PROJECT** menu.

Where	Item	Value
Composition panel	**NAME, AS PROTOTYPE**	`HPC`
Composition menu	**ECU/HPC Mapping**	**HPC [HPC]**

Creating the Composition HPC

1. In the **EDITOR** view, click **VFB** under **DESIGN**. The tab `VehicleDesign:VFB` opens with the composition `ECU [ECU]`.
2. Right-click an empty spot outside `ECU [ECU]` and click **Add Composition**.

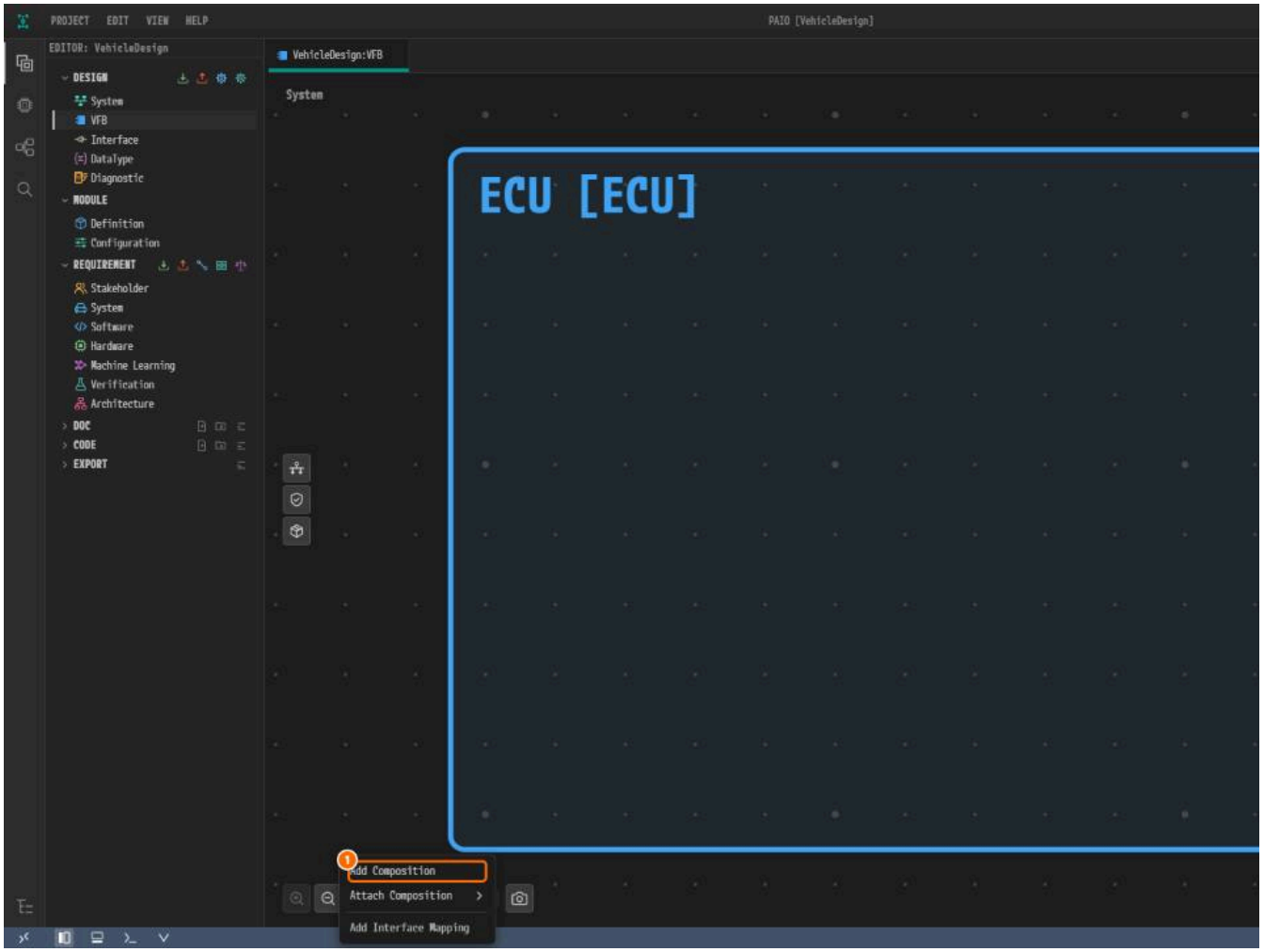


*Figure 1-1. ① **Add Composition**.*

3. Click the new composition. In the panel, enter `HPC` in both **NAME** and **AS PROTOTYPE**, pressing `Tab` after each. **AS PROTOTYPE** does not follow **NAME**, s
4. Press `Ctrl+S`.

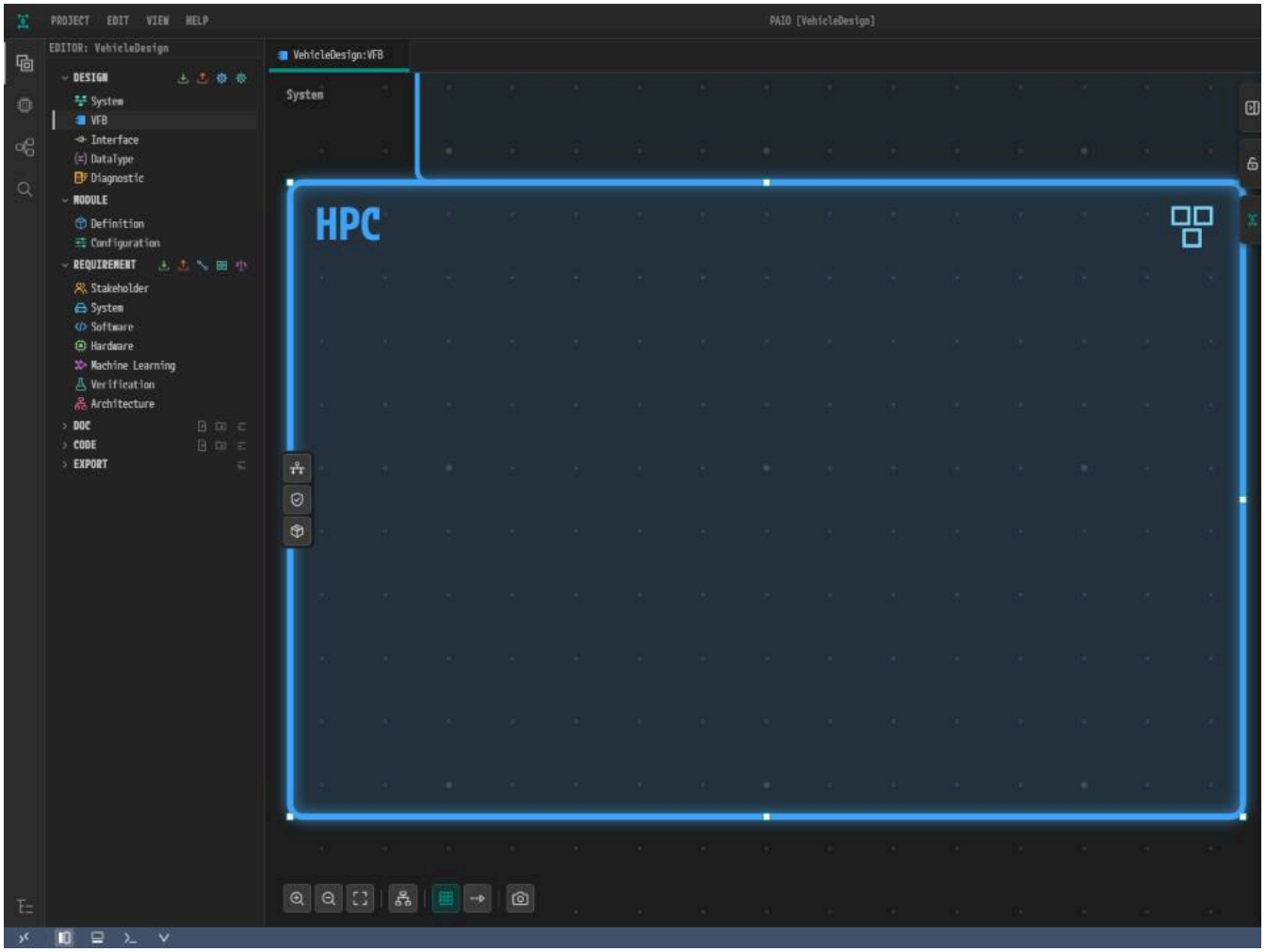


*Figure 1-2. ① **NAME**, ② **AS PROTOTYPE**. The new composition covers `ECU [ECU]` for now.*

Mapping the Composition onto the Node HPC

1. Right-click inside the composition `HPC`. Hover over **ECU/HPC Mapping** and click **HPC [HPC]**. The mapping is saved at once.

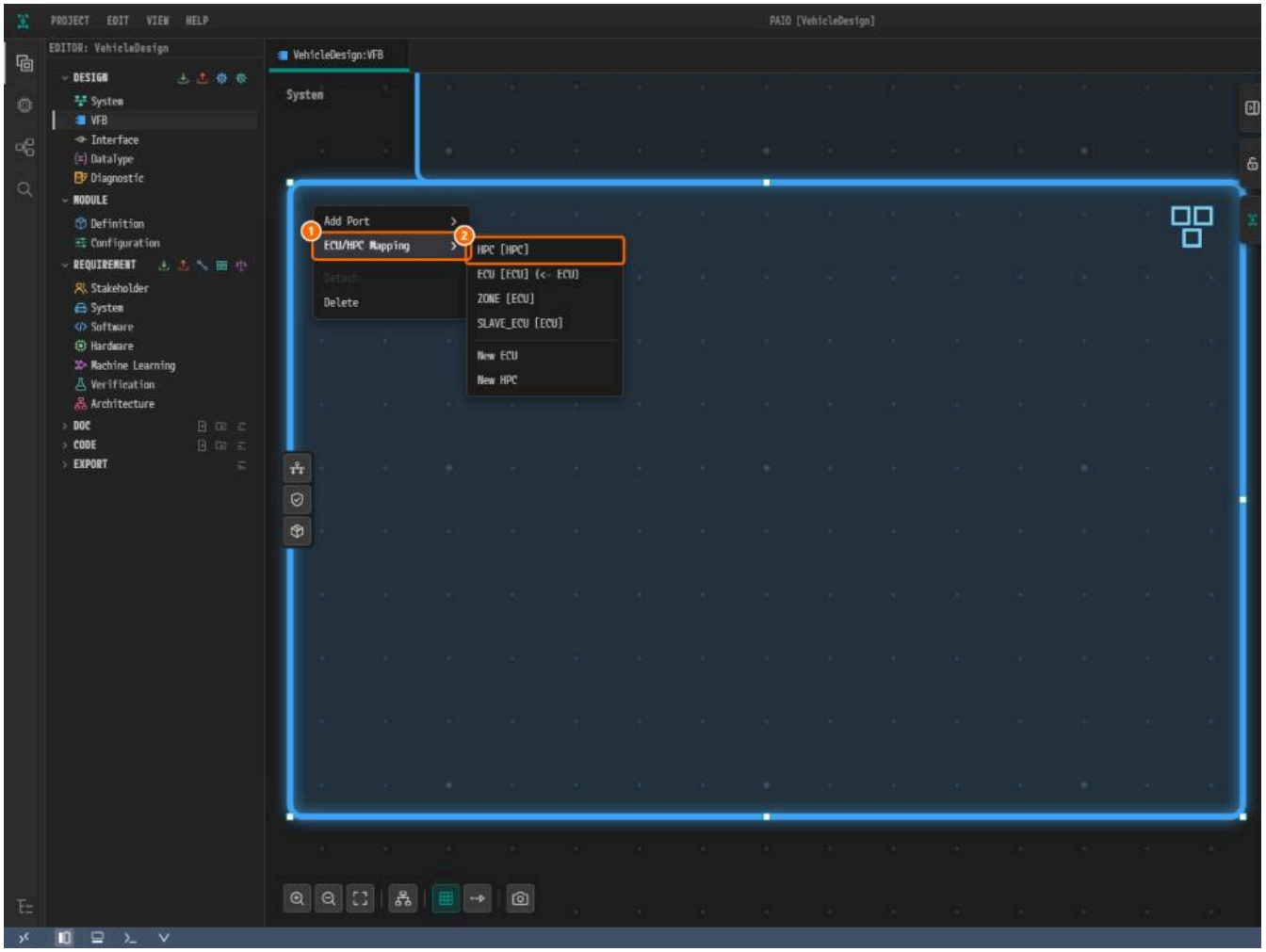


*Figure 1-3. ① **ECU/HPC Mapping**, ② **HPC [HPC]**.*

2. Click the fit button (the third button of the toolbar under the canvas). The composition now reads `HPC [HPC]` and carries a blue machine icon at the

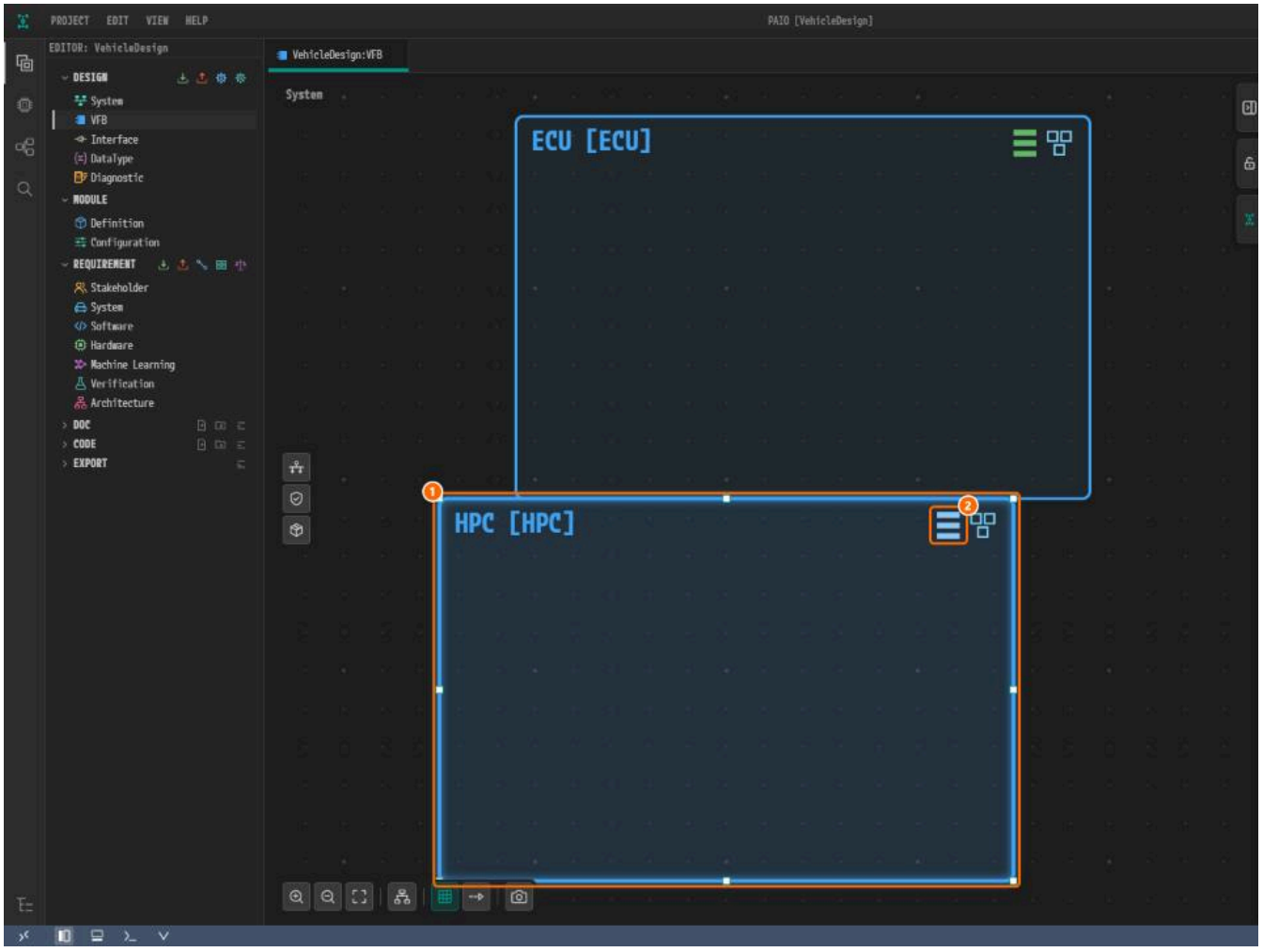


Figure 1-4. ① `HPC [HPC]`, ② the machine icon.

Opening the Machine Configuration Dialog

You change no value in the dialog. Saving it once writes the defaults (function group `MachineFG`, processor `CPU`, eleven modules) into the machine file.

Opening the Dialog

1. Close the tab `VehicleDesign:VFB` with its **×**.
2. In the **EDITOR** view, click **VFB** under **DESIGN** again.
3. Click the machine icon of `HPC [HPC]`. The dialog **Machine Configuration [HPC]** opens.

> **Caution.** In the VFB tab that was open during the mapping, the machine icon does not open the dialog. Close and reopen the tab as in steps 1 and 2.

The MACHINE Tab

The dialog opens on the **MACHINE** tab. **OS** is empty and shows `Name is required`; you fill it in the next chapter.

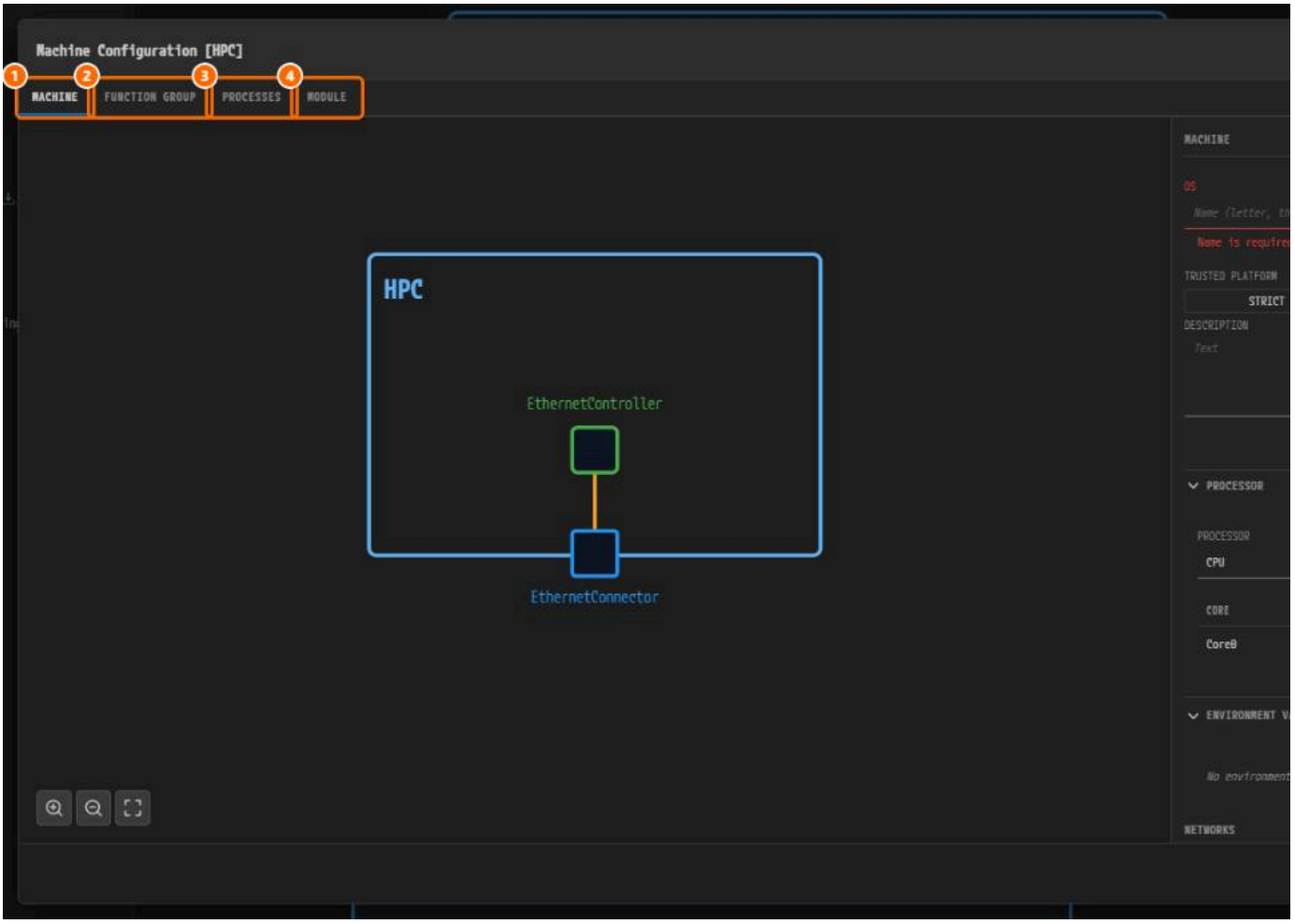


*Figure 1-5. ① **MACHINE**, ② **FUNCTION GROUP**, ③ **PROCESSES**, ④ **MODULE** tabs.*

The FUNCTION GROUP Tab

1. Click the **FUNCTION GROUP** tab. The list holds `MachineFG` with the states `Off`, `Startup`, `Shutdown`, `Restart`, `Verify`.

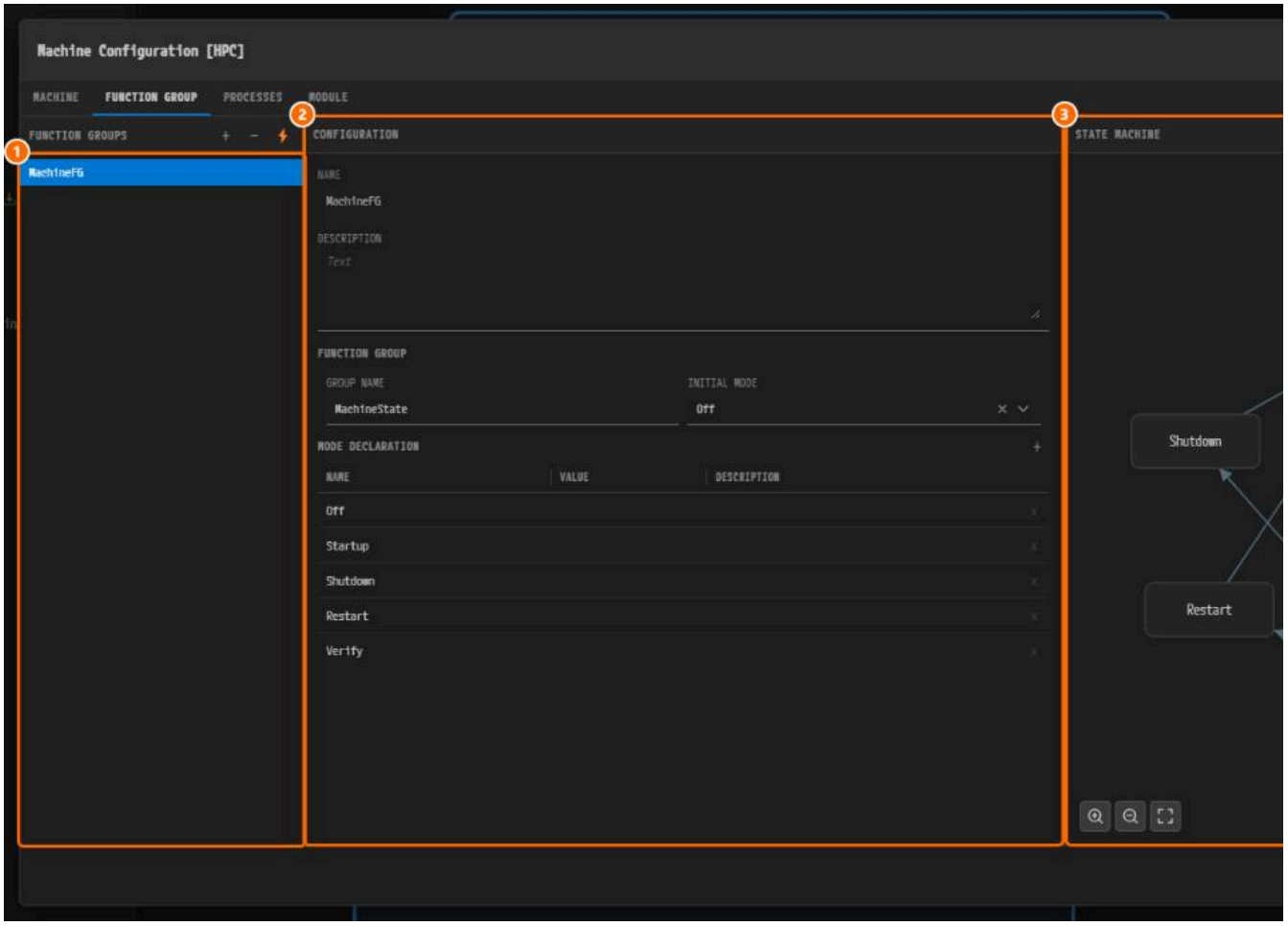


*Figure 1-6. ① **FUNCTION GROUPS** list, ② **CONFIGURATION**, ③ **STATE MACHINE**.*

The PROCESSES Tab

1. Click the **PROCESSES** tab. It reads `No Adaptive Application processes found in this Composition`.

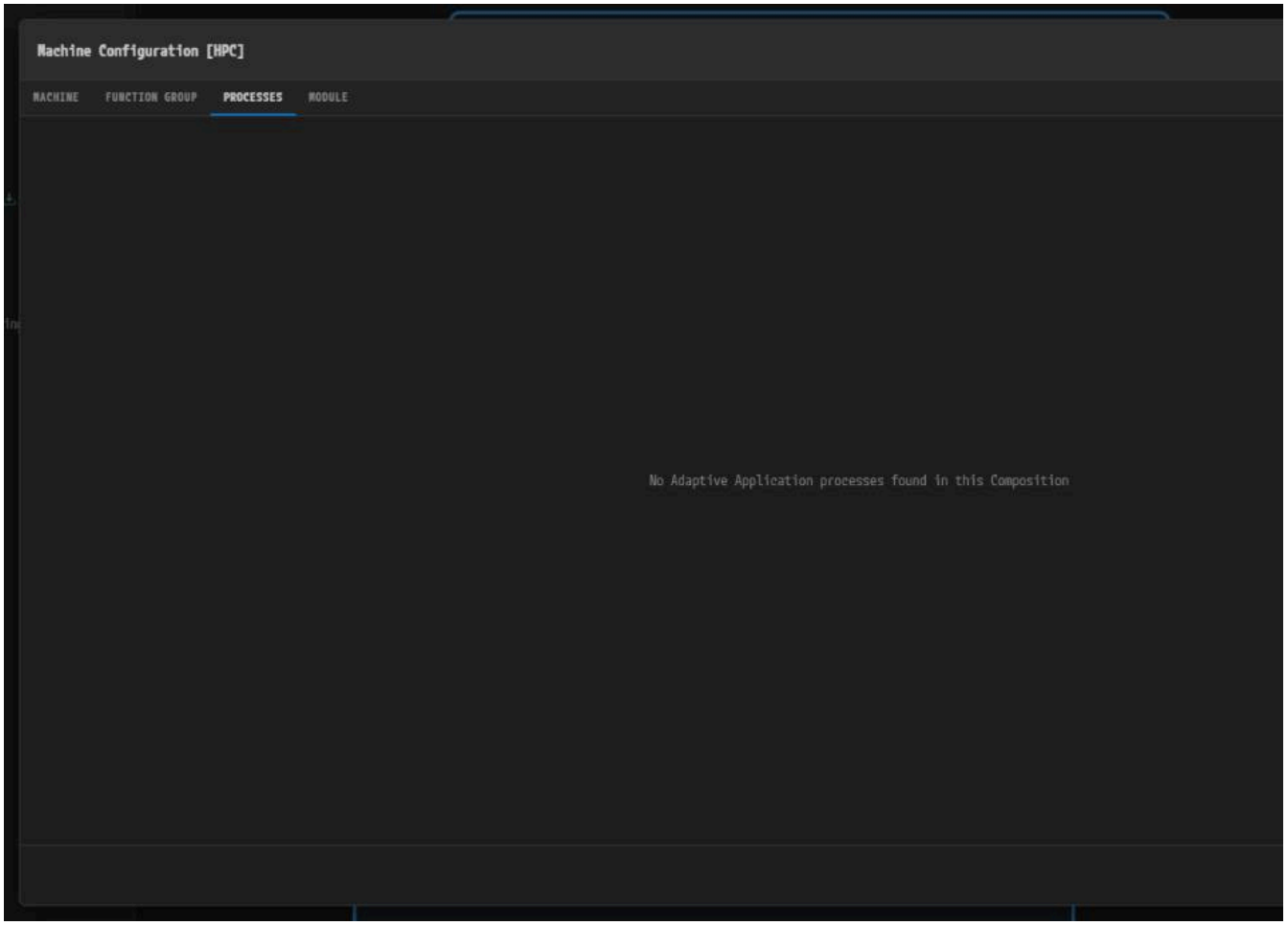


*Figure 1-7. The empty **PROCESSES** tab. Processes appear in Executables and Processes.*

The MODULE Tab

1. Click the **MODULE** tab. The **MODULES** list holds eleven modules, starting with **Log and Trace**.

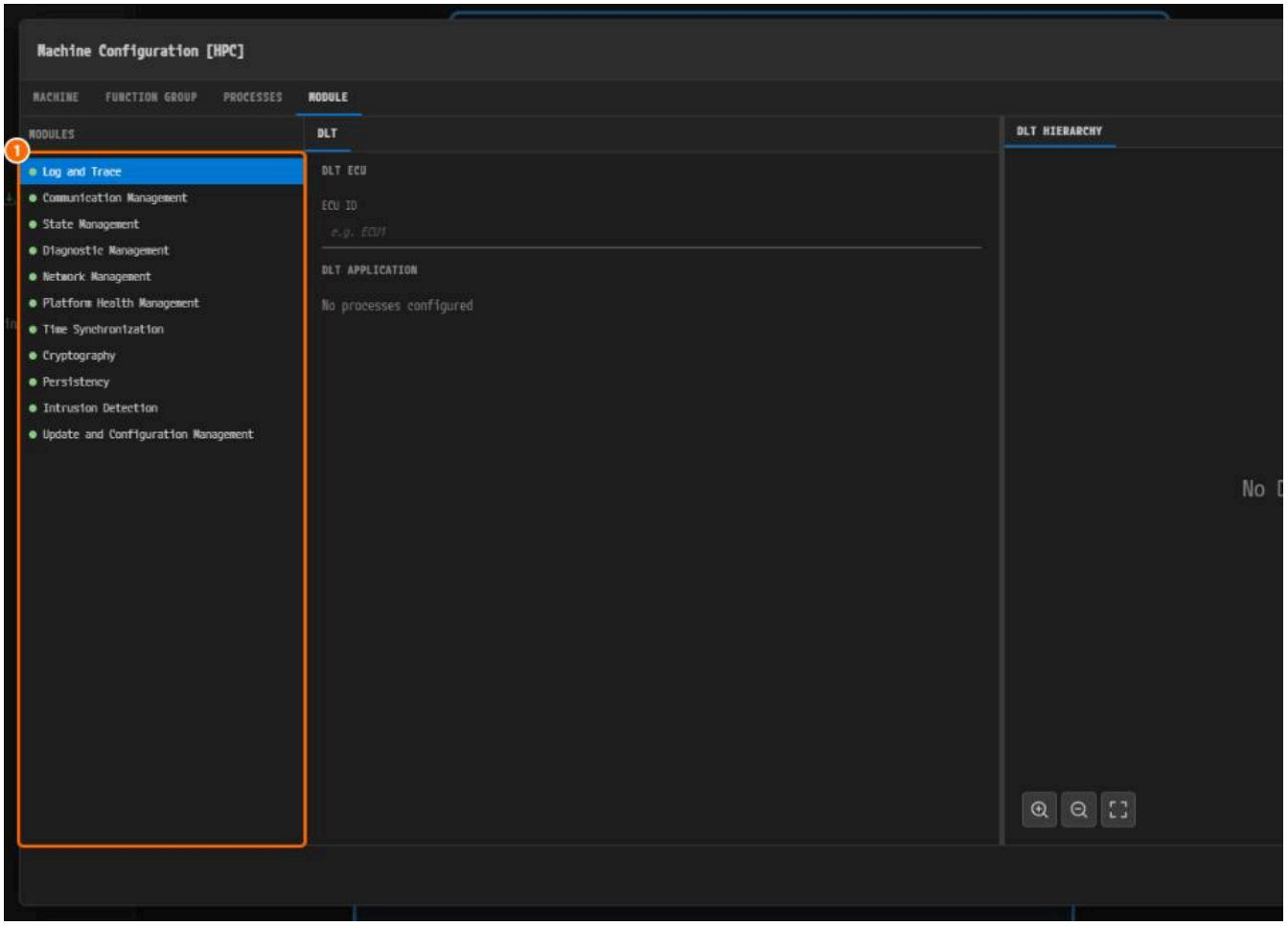


*Figure 1-8. ① The **MODULES** list. A green dot means the module is enabled.*

Saving and Closing

1. Press `Ctrl+S` inside the dialog.
2. Click **Close**, then **PROJECT** > **Save All**.

> **Caution. Close** alone does not create the machine file, and no save prompt appears. Always press `Ctrl+S` inside the dialog.

Viewing the Validation Result

1. Click the **V** icon in the footer (fifth from the left) to open the **VALIDATION** panel. It shows one error.

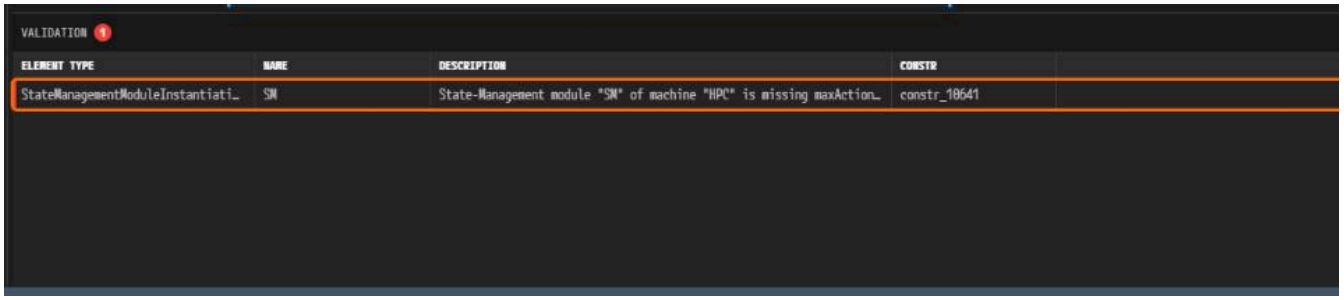


*Figure 1-9. The **VALIDATION** panel with one row.*

ELEMENT TYPE	NAME	CONSTR
`StateManagementModuleInstantiation`	`SM`	`constr_10641`

This error is expected. It disappears when you fill in **Max Action List Duration** in Machine Configuration.

Why Is It Designed This Way?

- **Mapping from the VFB. ECU/HPC Mapping** lists every node, so the same menu maps `ECU` to an ECU node and `HPC` to an HPC node.
- **Saving the dialog unchanged.** It writes the defaults, so later chapters fill values on top of them.

Checklist

- [ ] The VFB root canvas holds `ECU [ECU]` and `HPC [HPC]`, and `HPC [HPC]` carries the machine icon.
- [ ] **NAME** and **AS PROTOTYPE** of `HPC` both read `HPC`.
- [ ] `design/Module/Machine/HPC.machine.io.model` exists in the project folder.
- [ ] The **VALIDATION** panel shows exactly one finding, `constr_10641`.

Next Step

In Machine Configuration you fill in the machine values and clear this error.

Configuring the Machine

What You Will Do

In this chapter you fill in the machine `HPC`: the OS, four cores, the DLT **ECU ID** and the **Max Action List Duration**. The validation error `constr_10641` left by

Before You Begin

- Start after finishing The Composition HPC and the Machine. The **VALIDATION** panel shows one finding, `constr_10641`.

Tab	Item	Value
MACHINE	**OS**	`POSIX`
MACHINE	**PROCESSOR**	`CPU` (default, leave as is)
MACHINE	**CORE** table	+ button three times → `Core1` / `1`, `Core2` / `2`, `Core3` / `3` added automatically
FUNCTION GROUP	`MachineFG`	Do not change (check only)
MODULE > Log and Trace	**ECU ID**	`ECU1`
MODULE > State Management > ACTION ITEMS	**MAX ACTION LIST DURATION**	`1000` (unit `s`)

A field commits its value when you leave it, so press `Tab` after typing.

Checking the Starting State

1. Click the **V** icon in the footer to open the **VALIDATION** panel. It shows one row.

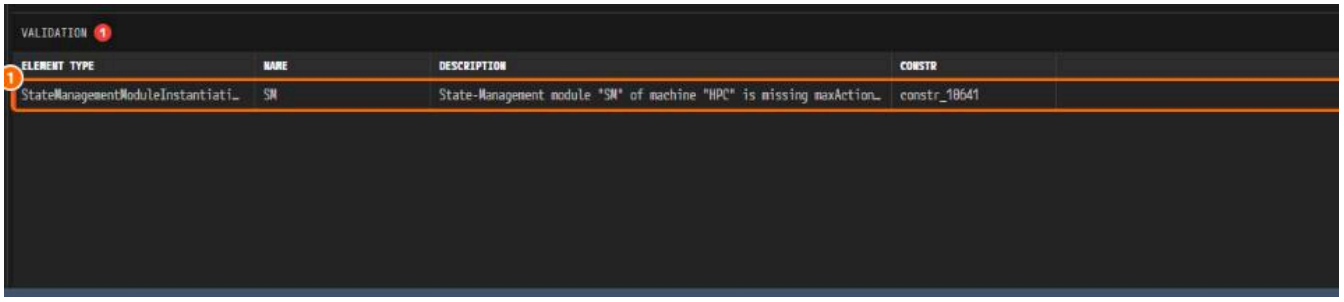


Figure 2-1. ① The row `constr_10641`.

Opening the Dialog

1. Close the **VALIDATION** panel with the **V** icon. If the tab `VehicleDesign:VFB` is open, close it.
2. In the **EDITOR** view, click **VFB** under **DESIGN**.
3. Click the blue machine icon of `HPC [HPC]`. Do not click the green icon of `ECU [ECU]`.

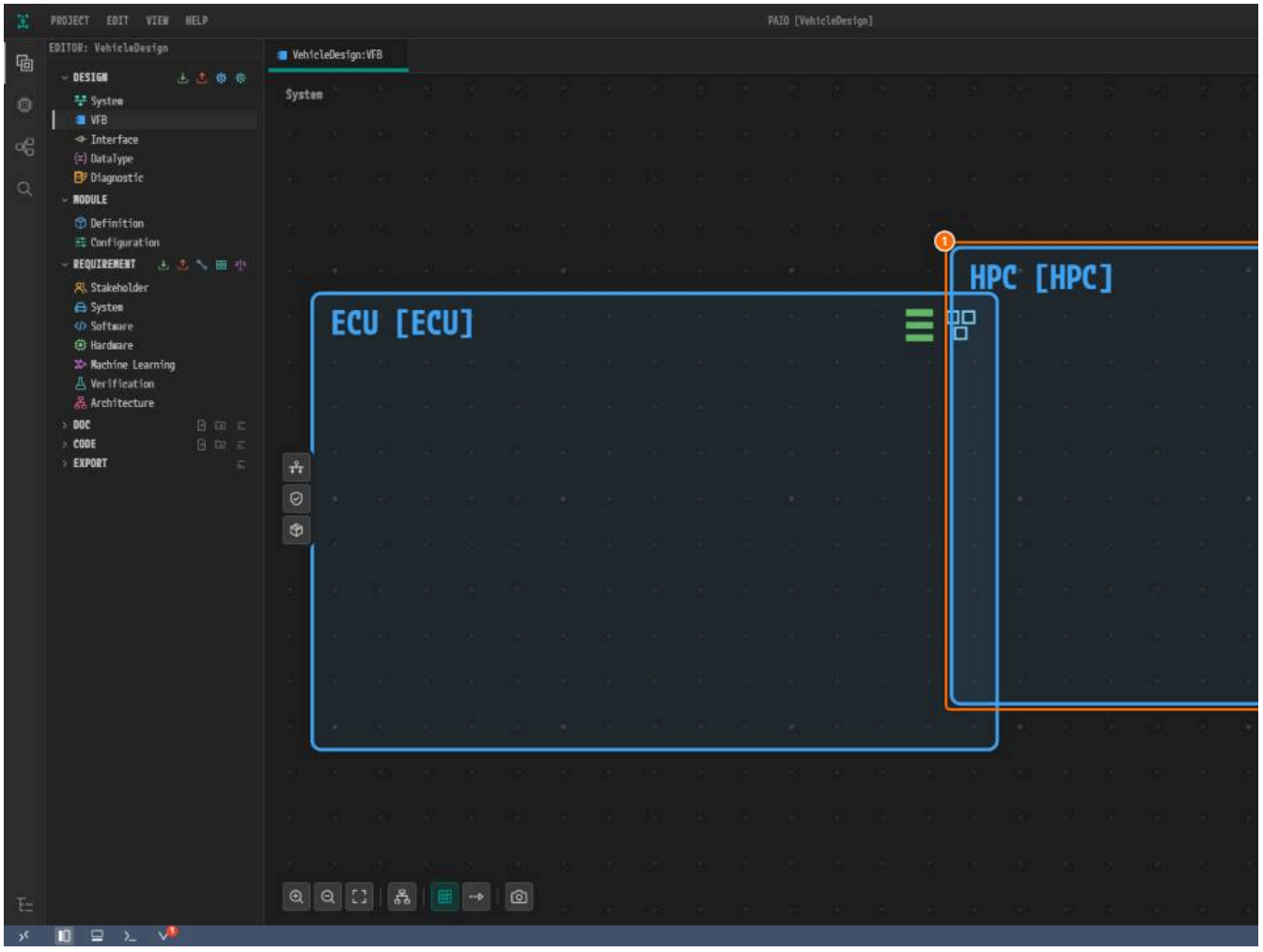


Figure 2-2. ① `HPC [HPC]`, ② the machine icon.

4. The dialog **Machine Configuration [HPC]** opens on the **MACHINE** tab.

MACHINE Tab — OS and Cores

Entering POSIX as the OS

1. Type `POSIX` into **OS** and press `Tab`. The red `Name is required` disappears.
2. Check that **PROCESSOR** reads `CPU`.

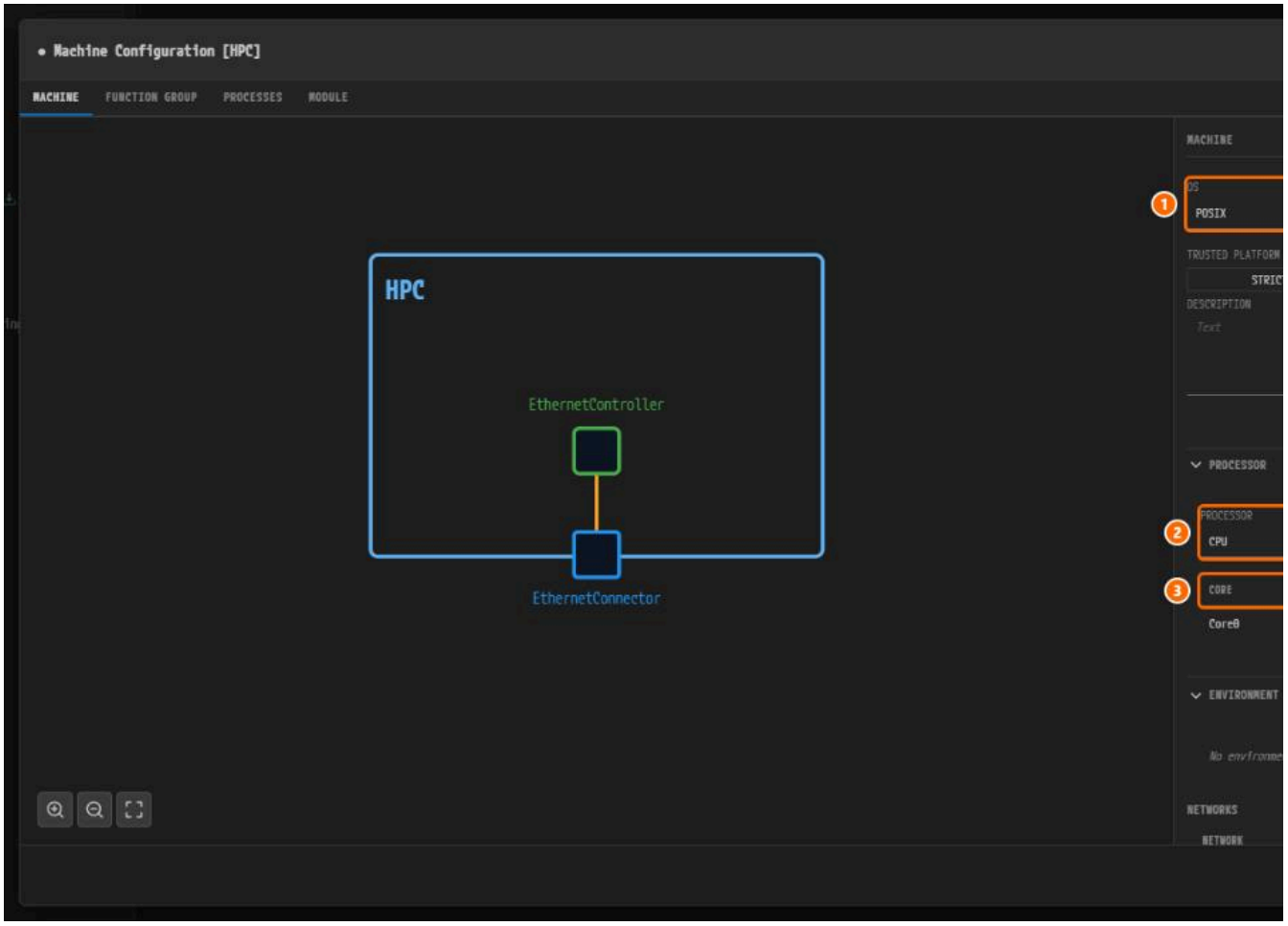


*Figure 2-3. ① **OS**, ② **PROCESSOR**, ③ the **CORE** table with the **+** button.*

> **Caution.** After `Tab`, the focus sits on the **STRICT** button of **TRUSTED PLATFORM**. Do not press Space or Enter; it must stay **NO SUPPORT**.

Adding Cores

1. Click the **+** button (`Add Core`) of the **CORE** table. The row `Core1` / `1` is added.

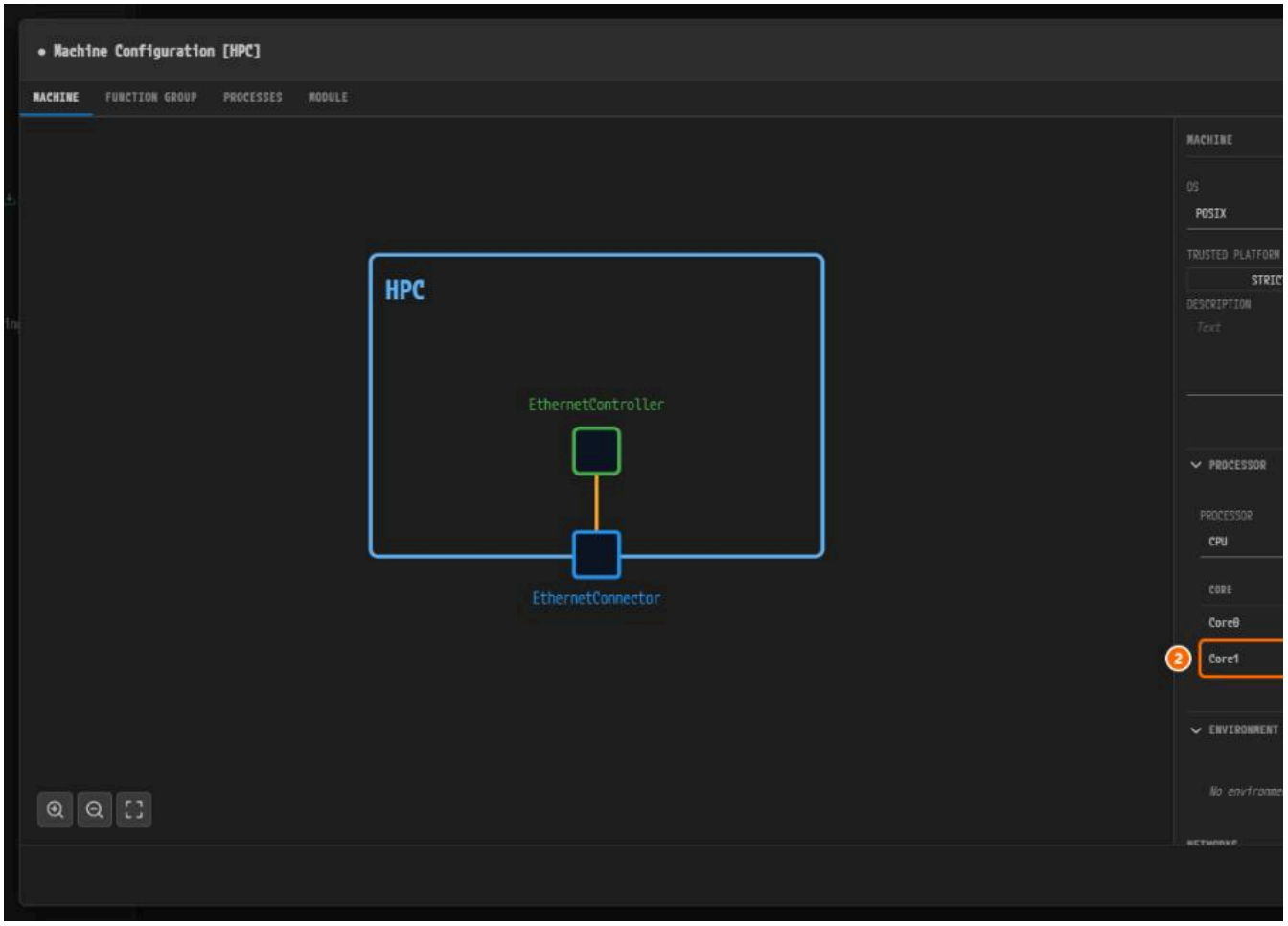


Figure 2-4. ① +, ② the new row `Core1` / `1`.

2. Click **+** two more times. `Core2` / `2` and `Core3` / `3` are added.

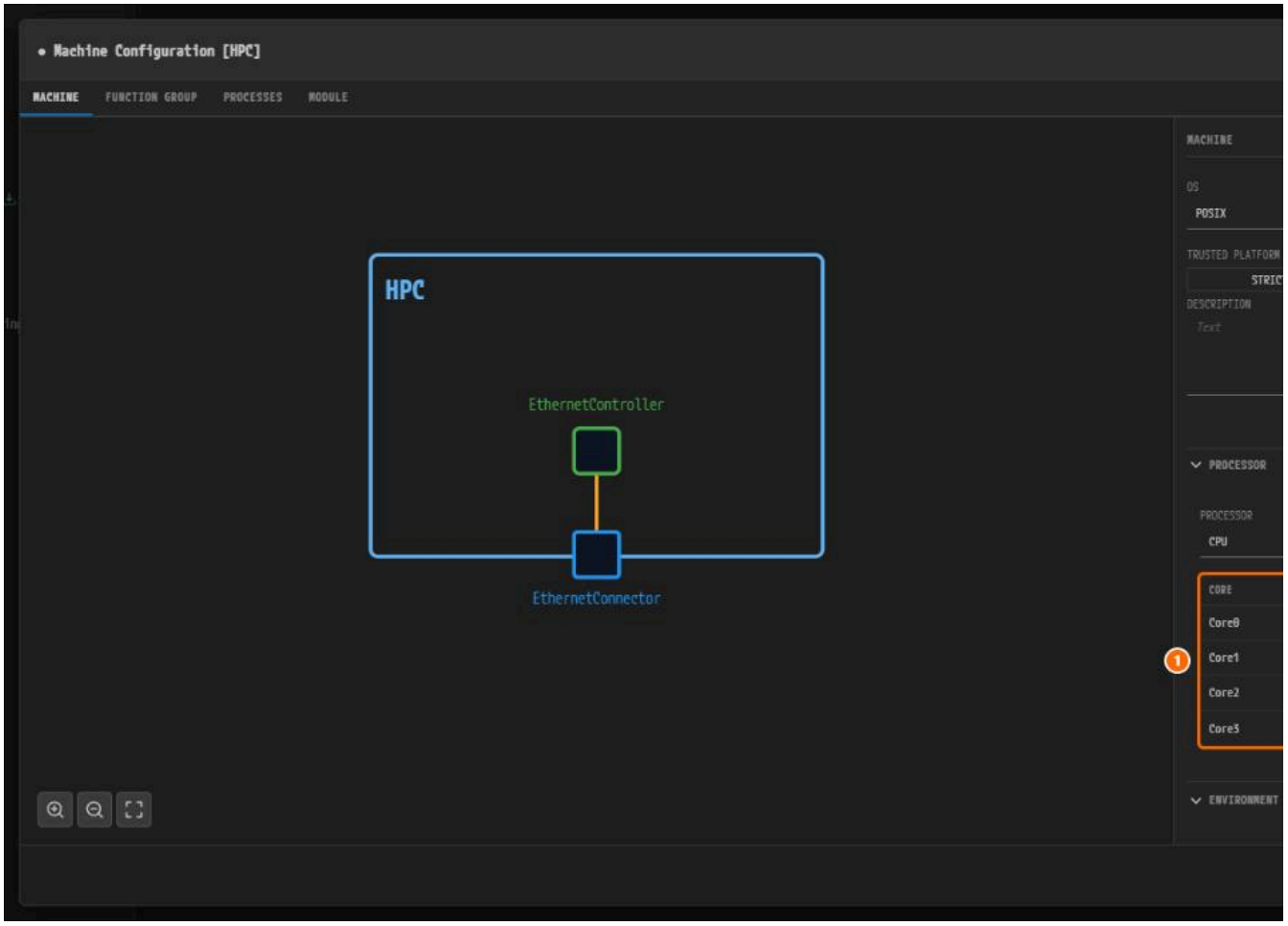


Figure 2-5. ① The four cores `Core0` to `Core3`.

FUNCTION GROUP Tab — Checking MachineFG

1. Click the **FUNCTION GROUP** tab. Check that `MachineFG` is the only entry, with **GROUP NAME** `MachineState` and **INITIAL MODE** `Off`. Change nothing.

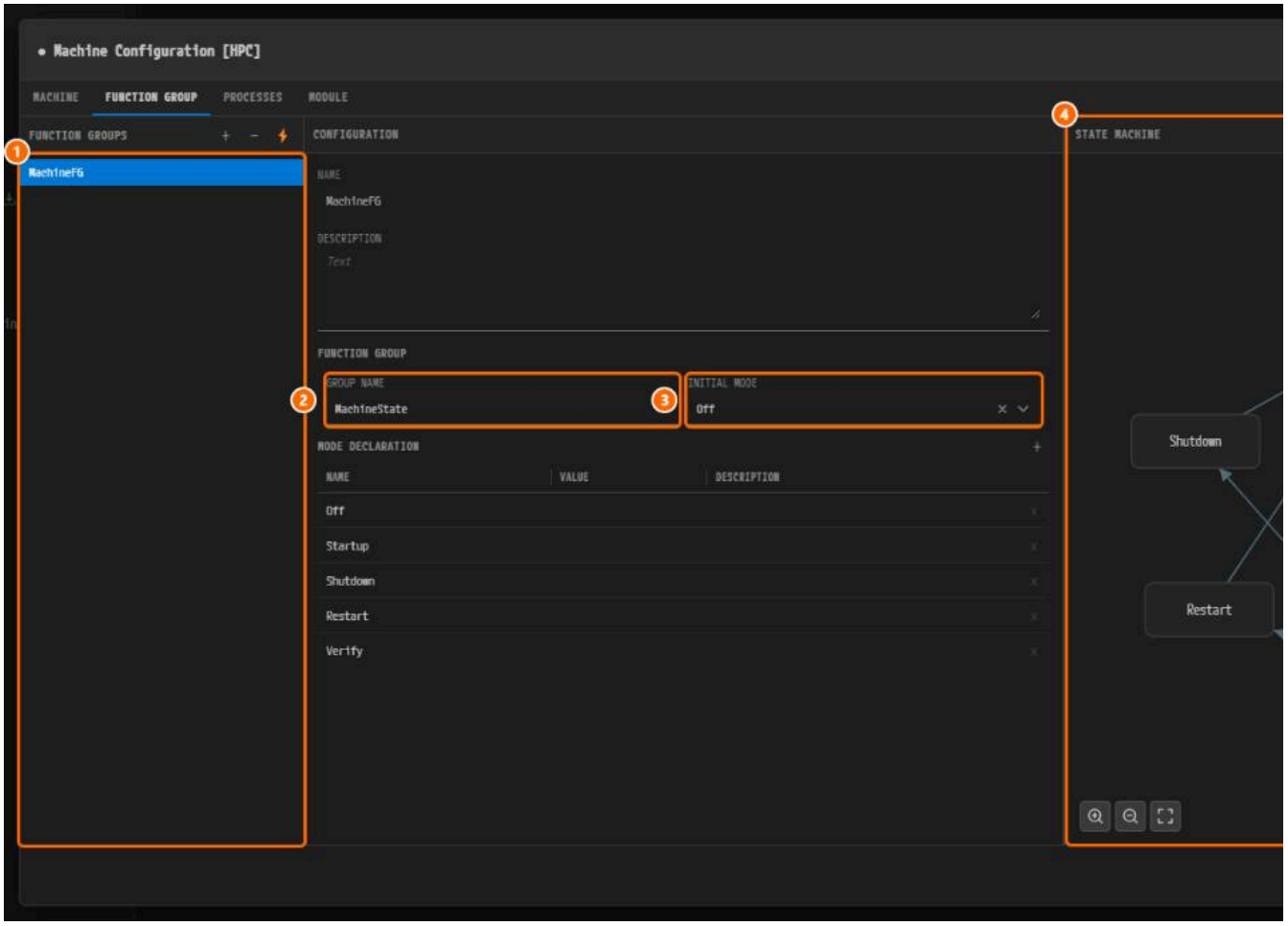


*Figure 2-6. ① `MachineFG`, ② **GROUP NAME**, ③ **INITIAL MODE**, ④ **STATE MACHINE**.*

MODULE Tab — ECU ID of Log and Trace

1. Click the **MODULE** tab. **Log and Trace** is selected.
2. Type `ECU1` into **ECU ID** of the **DLT ECU** section and press `Tab`.

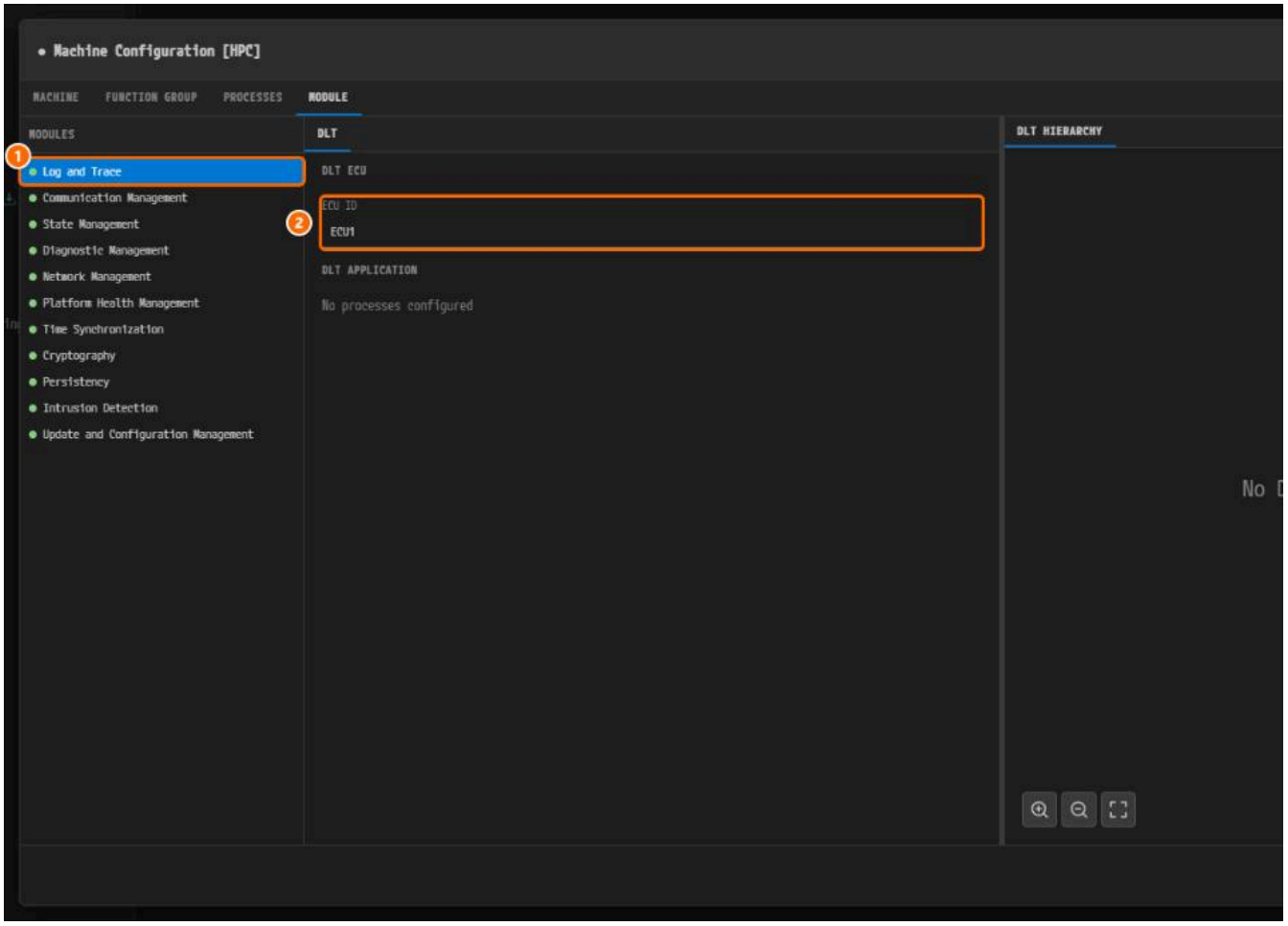


*Figure 2-7. ① **Log and Trace**, ② **ECU ID**.*

MODULE Tab — Max Action List Duration of State Management

1. In the **MODULES** list, click **State Management**, then the **ACTION ITEMS** sub-tab.

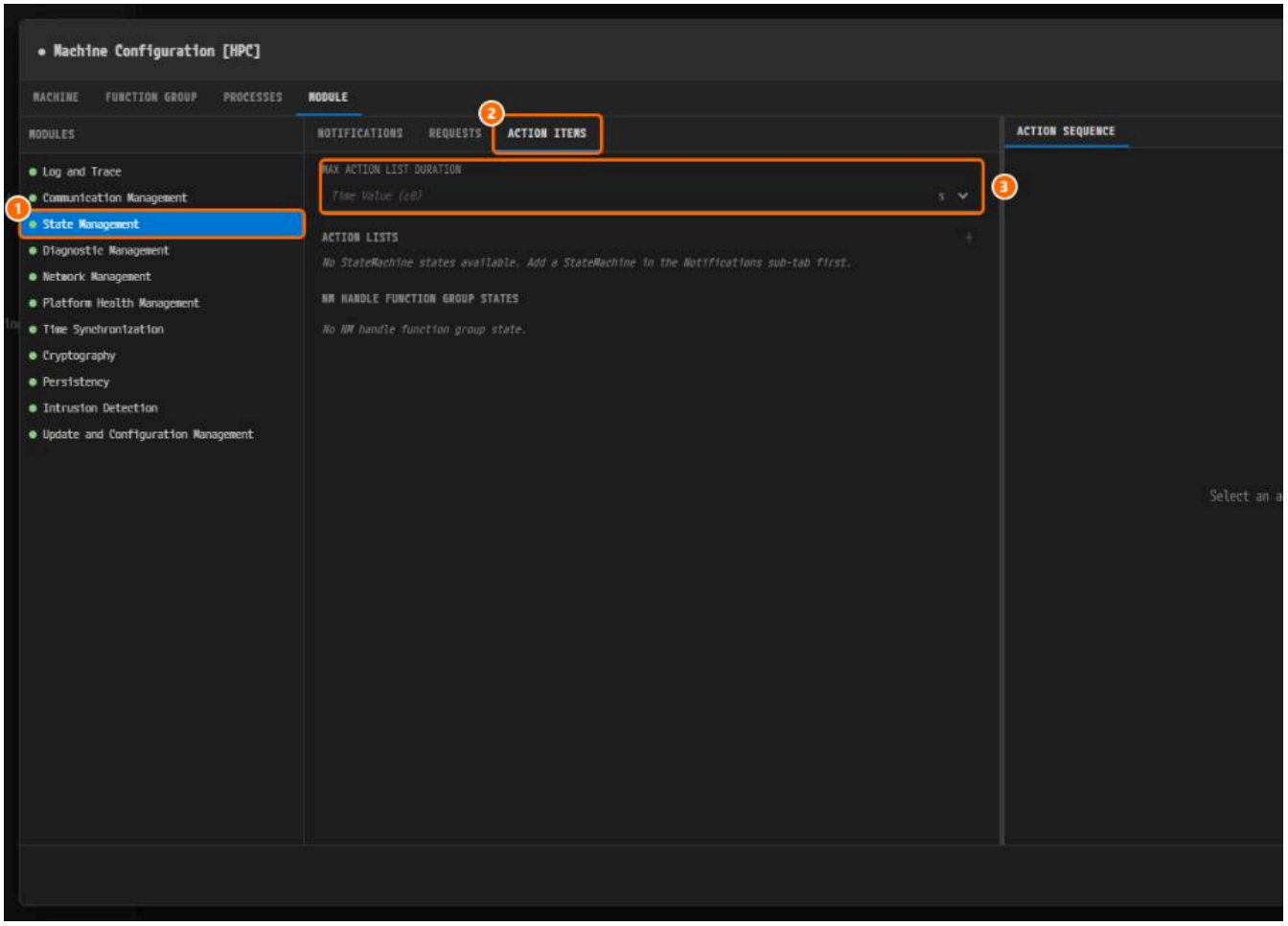


*Figure 2-8. ① **State Management**, ② **ACTION ITEMS**, ③ **MAX ACTION LIST DURATION**.*

2. Type `1000` into **MAX ACTION LIST DURATION** and press `Tab`. Leave the unit at `s`.

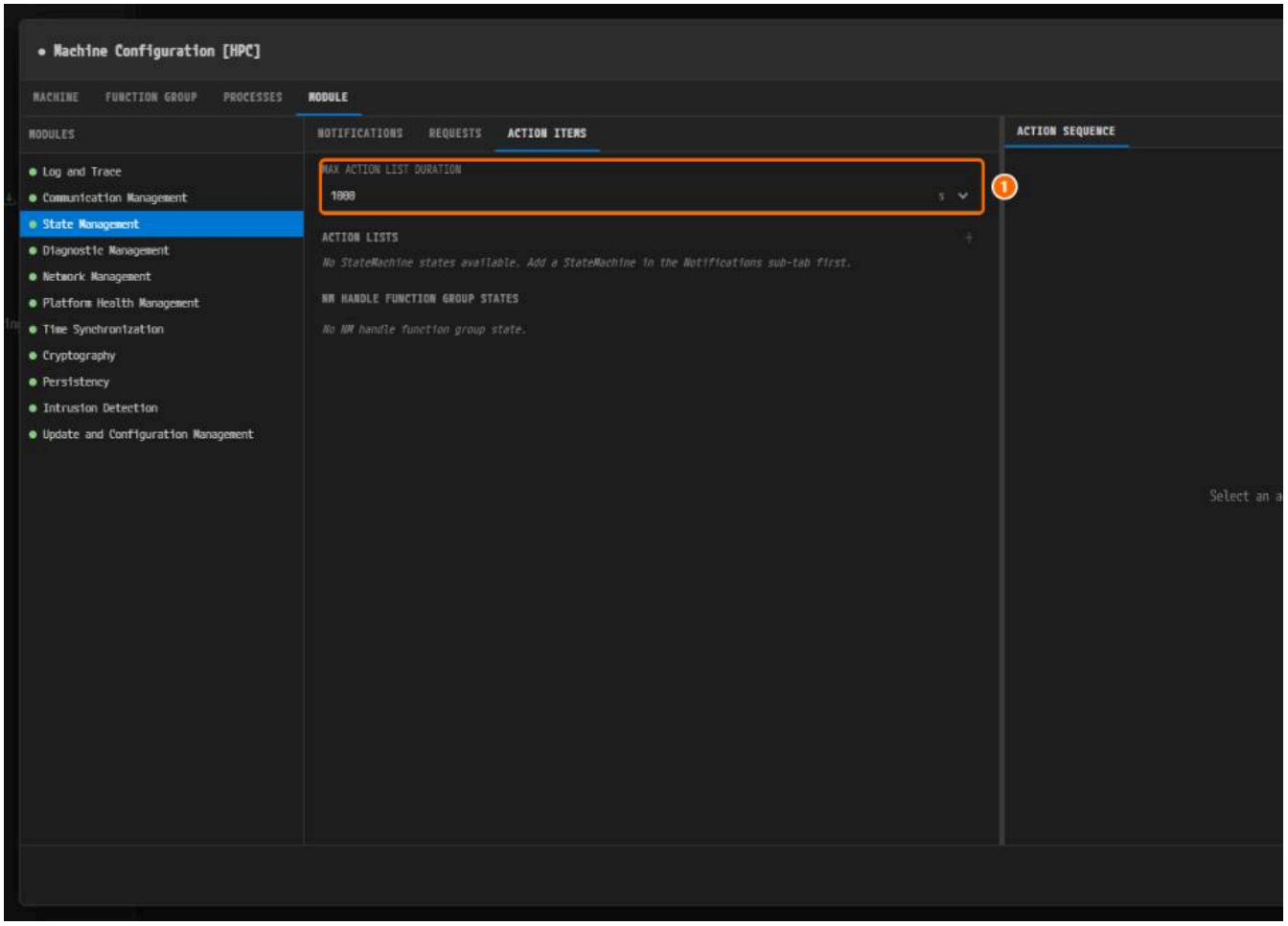


*Figure 2-9. ① **MAX ACTION LIST DURATION** = `1000`, unit `s`.*

Saving and Closing

1. Press `Ctrl+S` inside the dialog.
2. Click **Close**, then **PROJECT** > **Save All**.

> **Caution. Close** alone does not save the machine file. If a save prompt appears, click **Save**.

Viewing the Validation Result

1. Open the **VALIDATION** panel. It reads `No validation issues.`.

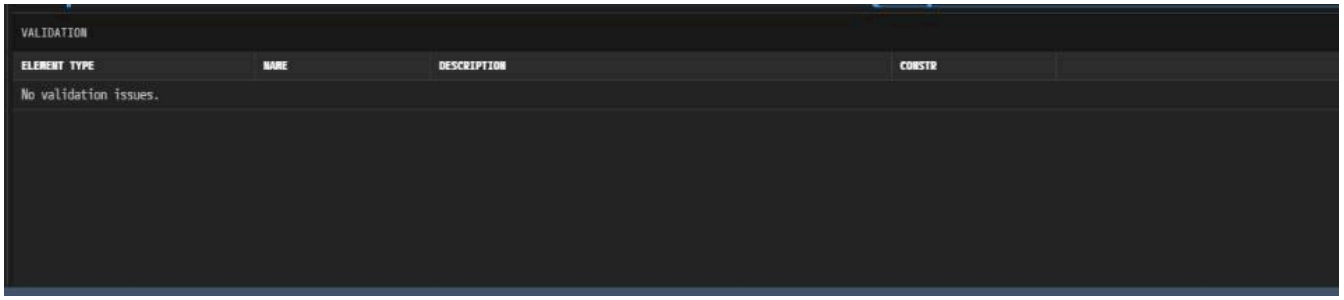


*Figure 2-10. The **VALIDATION** panel after saving.*

> **Note.** The tooltip of the **V** icon keeps the last count. Trust the panel.

Why Is It Designed This Way?

- `POSIX` becomes the name of the `OS-MODULE-INSTANTIATION` in the exported ARXML.
- **The cores** get their names and IDs from the **+** button, so you never line them up by hand.
- `ECU1` is the DLT ECU under which Logging and DLT gathers the DLT applications.

- `Max Action List Duration` is required by PAIO validation (`constr_10641`). Nothing else uses it, so `1000` is fine.
- `MachineFG` stays at its defaults. Executables and Processes binds processes to its states.

Check Your Work

- [ ] On the **MACHINE** tab, **OS** is `POSIX`, **TRUSTED PLATFORM** is **NO SUPPORT**, and the **CORE** table has `Core0` / `0` to `Core3` / `3`.
- [ ] **ECU ID** under **MODULE > Log and Trace** is `ECU1`.
- [ ] **MAX ACTION LIST DURATION** under **MODULE > State Management > ACTION ITEMS** is `1000` with unit `s`.
- [ ] No tab carries a modified marker.
- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In Topology — SOME/IP Discovery Port and UDP Endpoints you add the SOME/IP discovery port and the UDP endpoints to the connector `EthernetConnec`

Topology — SOME/IP Discovery Port and UDP Endpoints

What You Will Do

In this chapter you add what SOME/IP needs to the connector `EthernetConnector` of the node `HPC`: the service discovery port `30490` and three UDP en

Before You Begin

- Start after finishing Configuring the Machine. The **VALIDATION** panel shows `No validation issues.`.
- The connector already has the unicast address `192.168.0.11` and the multicast address `224.0.0.1` from the AUTOSAR System Tutorial.

Where	Item	Value
NETWORK ENDPOINT > MULTICAST CONFIG	**MULTICAST PORT**	`30490`
APPLICATION ENDPOINTS	Three cards, **TYPE** = **UDP**, **PORT NUMBER**	`20001`, `20002`, `20003`

A field commits its value when you leave it, so press `Tab` after typing.

Opening the System Editor

1. In the **EDITOR** view, click **System** under **DESIGN**. The tab `VehicleDesign:System` opens.
2. Click the fit button (the third button of the toolbar under the canvas).

Network Endpoint — the SOME/IP Discovery Port

1. Click the connector `EthernetConnector`, the blue square on the bottom border of the node `HPC`. The **ETHERNET CONNECTOR** panel opens.
2. Under **NETWORK ENDPOINT > MULTICAST CONFIG**, type `30490` into **MULTICAST PORT** and press `Tab`. Leave the other fields as they are.

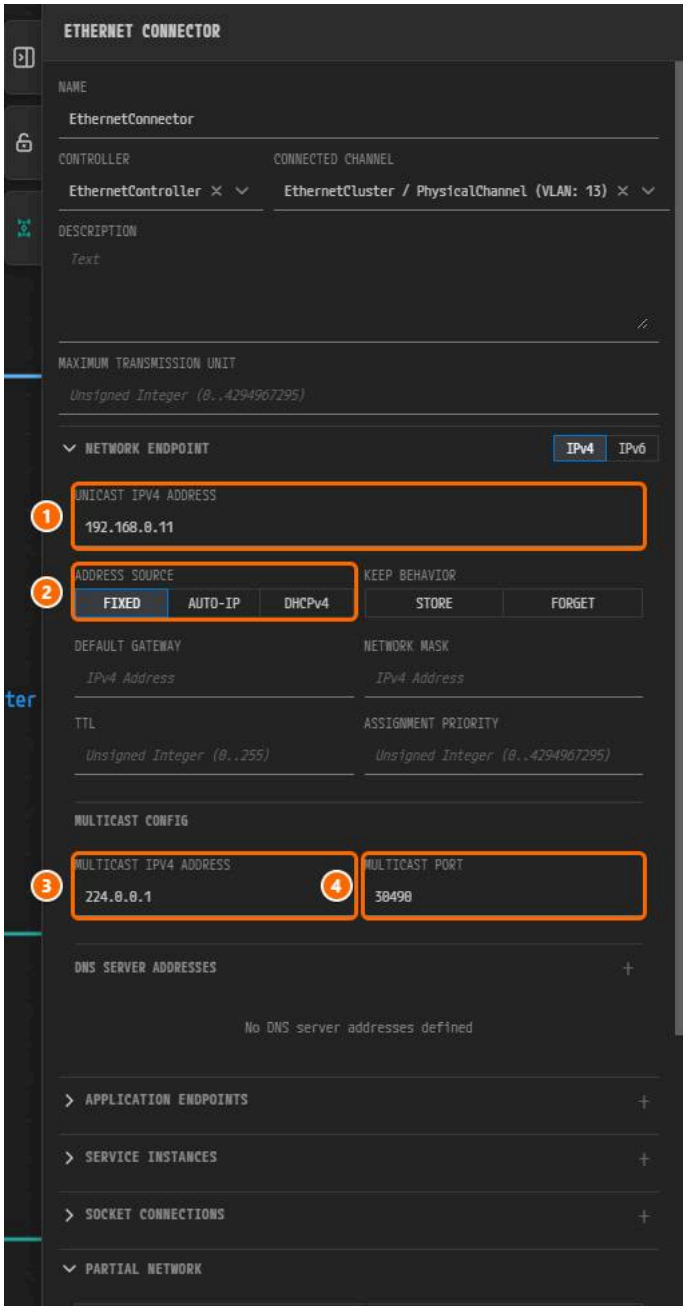


*Figure 3-1. ① **UNICAST IPV4 ADDRESS**, ② **ADDRESS SOURCE**, ③ **MULTICAST IPV4 ADDRESS**, ④ **MULTICAST PORT**.*

Application Endpoints — Three UDP Ports

Follow the port `20001` first.

Adding the First Endpoint

1. Collapse **NETWORK ENDPOINT** and expand **APPLICATION ENDPOINTS** by clicking their headers.
2. Click **+** (`Add Application Endpoint`) at the right of the header. A card `TCP EthernetConnector, 192.168.0.11:0` appears.
3. Click the card to expand it, then click **UDP** under **TYPE**.
4. Type `20001` into **PORT NUMBER** and press `Tab`. The card reads `UDP EthernetConnector, 192.168.0.11:20001`.

Adding the Remaining Endpoints

Create the other two the same way from the table. The name is assigned automatically. Do not create any TCP endpoint.

Order	TYPE	PORT NUMBER	Summary line	Name assigned automatically
1	UDP	`20001`	`UDP EthernetConnector, 192.168.0.11:20001`	`HPC_EthernetConnector_UDP_20001`
2	UDP	`20002`	`UDP EthernetConnector, 192.168.0.11:20002`	`HPC_EthernetConnector_UDP_20002`
3	UDP	`20003`	`UDP EthernetConnector, 192.168.0.11:20003`	`HPC_EthernetConnector_UDP_20003`

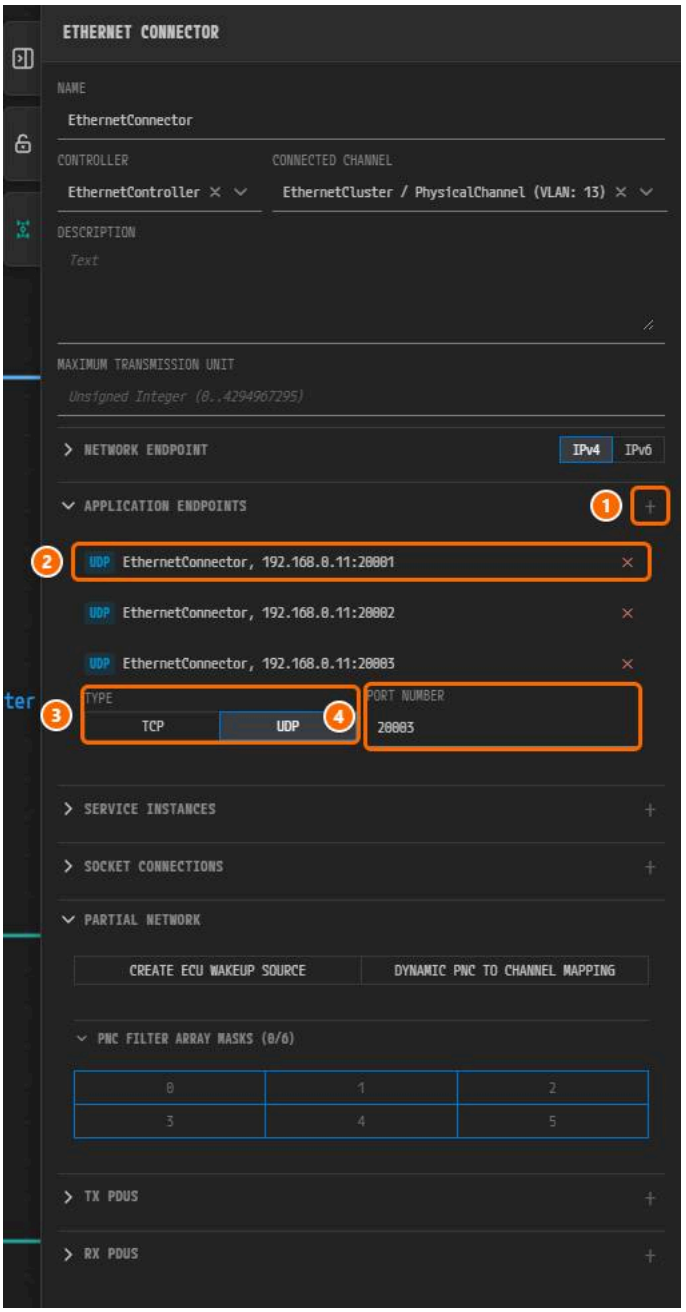


*Figure 3-2. ① **+**, ② the first card, ③ **TYPE** = **UDP**, ④ **PORT NUMBER** = `20003`.*

Saving

1. Click an empty spot on the canvas, then press `Ctrl+S`. The modified marker ● on the tab disappears.

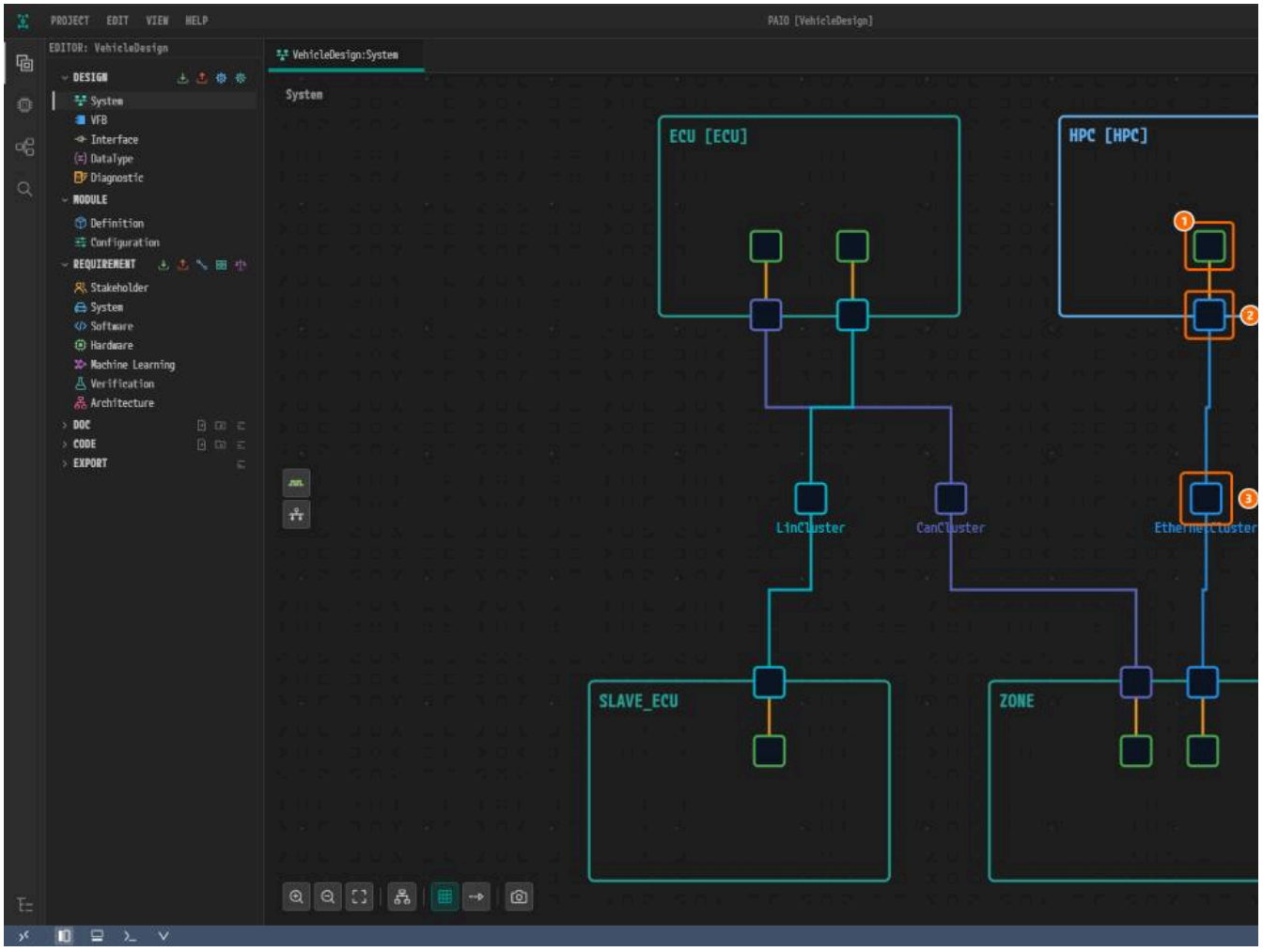


Figure 3-3. ① `EthernetController`, ② `EthernetConnector`, ③ `EthernetCluster`.

Checking the Validation Result and the NM CLUSTER Tab

Checking the Validation Result

1. Open the **VALIDATION** panel with the **V** icon in the footer. It reads `No validation issues.`.

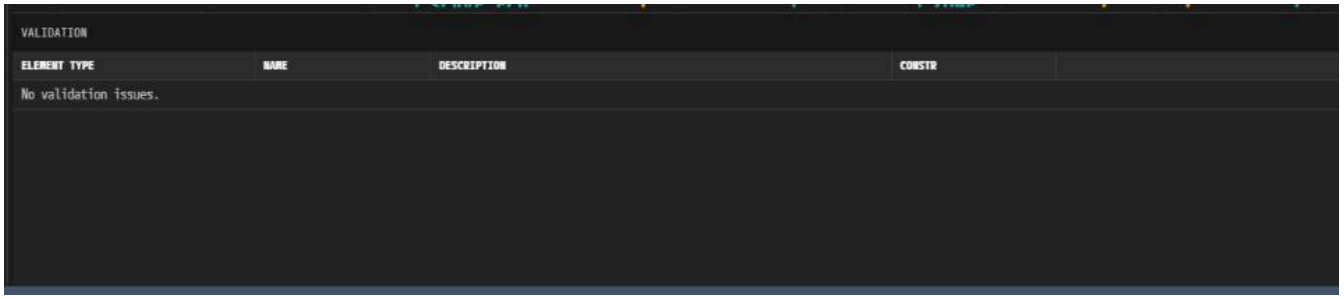


*Figure 3-4. The **VALIDATION** panel after saving.*

Checking the NM CLUSTER Tab in the Dialog

1. Close the **VALIDATION** panel and double-click the cluster `EthernetCluster`. The dialog **Communication Cluster Configuration [EthernetCluster]** open
2. Click the **NM CLUSTER** tab. The hollow dot before its name means NM is off; the tab is a dimmed preview.

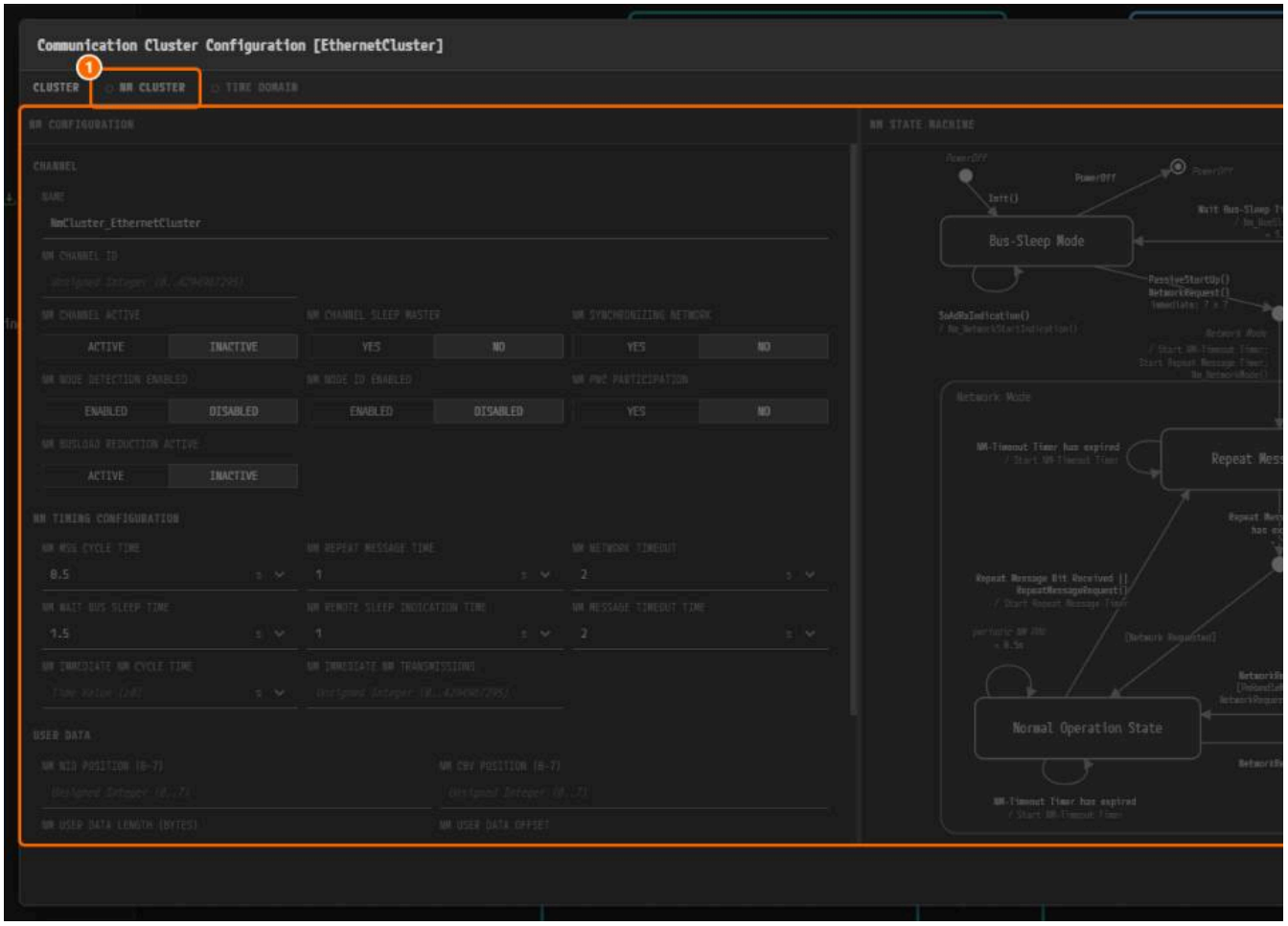


*Figure 3-5. ① **NM CLUSTER** with a hollow dot, ② the dimmed preview.*

3. Click **Close**.

> **Caution.** Do not choose **Enable** in the right-click menu of the tab. This tutorial leaves NM off.

Why Is It Designed This Way?

- **The SD port comes from the connector.** The export builds `SOMEIP-SERVICE-DISCOVERY-PORT 30490` from **MULTICAST IPV4 ADDRESS** `224.0.0.1` and
- **Three UDP ports, no TCP.** In Binding Service Instances to the Connector, `Actuator_Sensor` uses `20001`, `Fusion_Sensor` `20002` and `Provider_Sens`
- **NM stays off.** This tutorial does not use Network Management.

Check

- [ ] **MULTICAST PORT** of `EthernetConnector` is `30490`.
- [ ] **APPLICATION ENDPOINTS** holds the three UDP cards `20001`, `20002`, `20003`, and no TCP card.
- [ ] The tab `VehicleDesign:System` carries no modified marker ●.
- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In Data Types you create the C++ implementation data types `ByteArray` and `String`.

Data Types

What You Will Do

In this chapter you create the data types `ByteArray` and `String` in the DataType editor. In Service Interface the event `SensorData` uses `ByteArray`.

Before You Begin

- Start after finishing Topology — SOME/IP Discovery Port and UDP Endpoints. The **VALIDATION** panel shows `No validation issues.`.
- You may close the tab `VehicleDesign:System`.

Type	CATEGORY	ELEMENT TYPE	MAX SIZE	C++ NAMESPACE
`ByteArray`	**DYNAMIC_ARRAY**	`uint8`	`1`	`aa::type`
`String`	**STRING**	(no field)	`256`	`aa::type`

Leave **DESCRIPTION** empty. An input field commits its value only when you leave it, so press `Tab` after typing.

Opening the DataType Editor

1. In the **EDITOR** view, click **DataType** under **DESIGN**. The tab `VehicleDesign:DataType` opens.
2. The **DATATYPE** list starts with 13 built-in types in italics, 11 `[PRIMITIVE]` and 2 `[POINTER]`, followed by the 53 types from the AUTOSAR CP part. Leav built-in type: its fields cannot be edited.

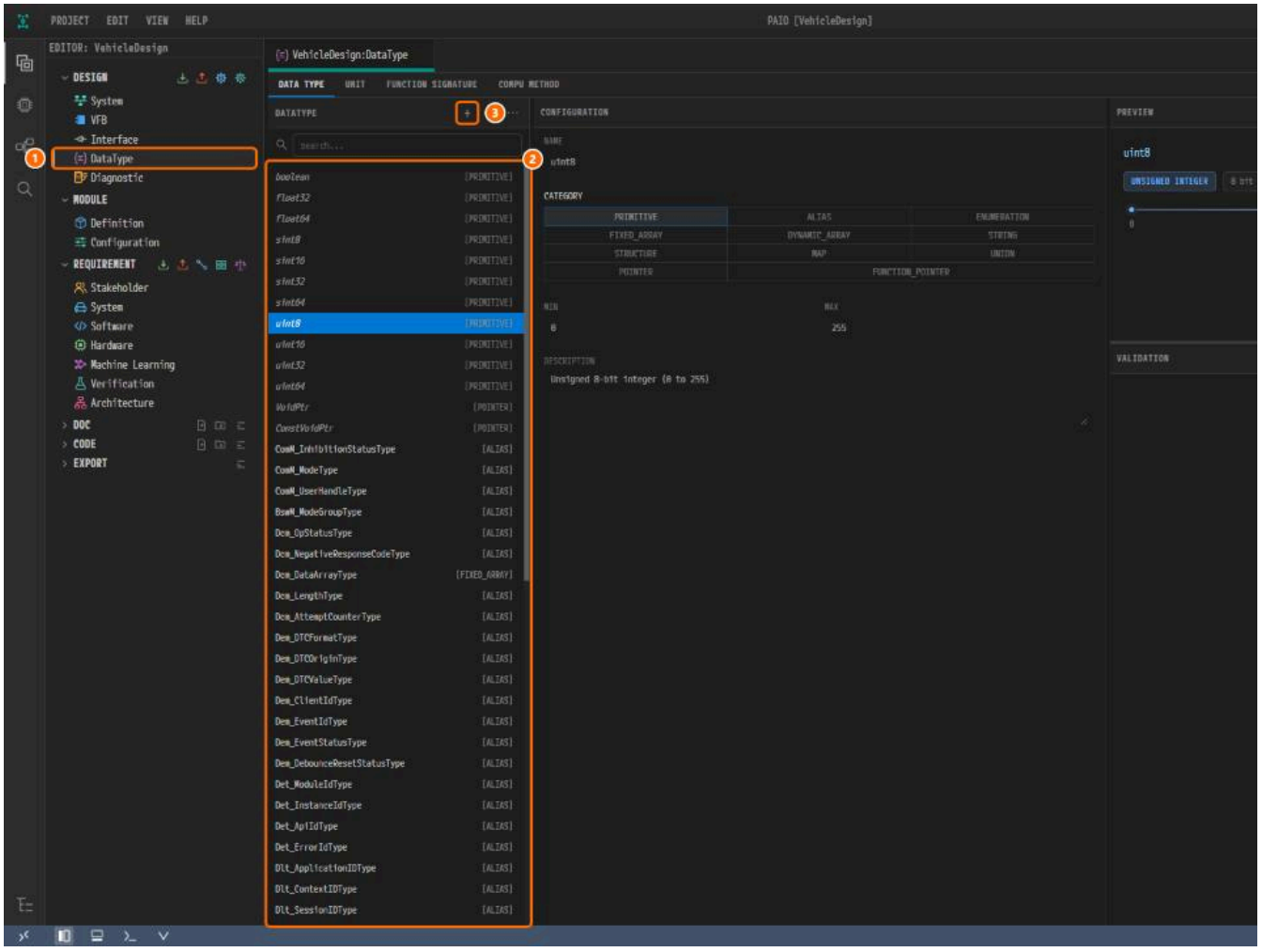


*Figure 4-1. ① **DataType** under **DESIGN**, ② the **DATATYPE** list (`uint8` selected), ③ the **+** button.*

Creating ByteArray — DYNAMIC_ARRAY

1. Click **+** in the list header. A new entry `DataType66` appears at the bottom of the list and is selected. Scroll down to see it.

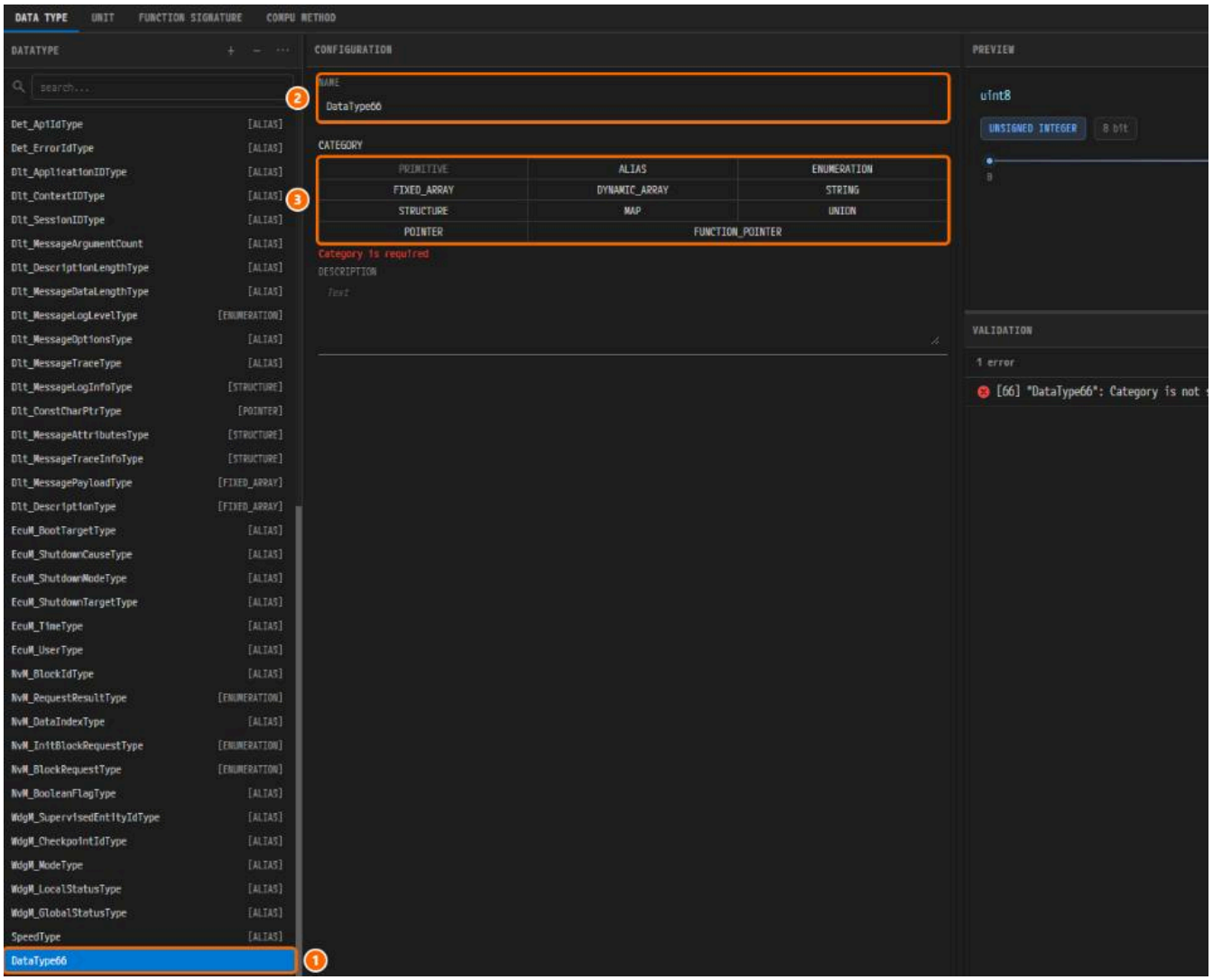


*Figure 4-2. ① `DataType66`, ② **NAME**, ③ **CATEGORY**.*

2. Change **NAME** to `ByteArray` and press `Tab`.
3. Click **DYNAMIC_ARRAY** in **CATEGORY**. The fields **C++ NAMESPACE**, **ELEMENT TYPE** and **MAX SIZE** appear. The editor shows an error until you pick the eler
4. Click the **ELEMENT TYPE** field, scroll the list and click `uint8`.

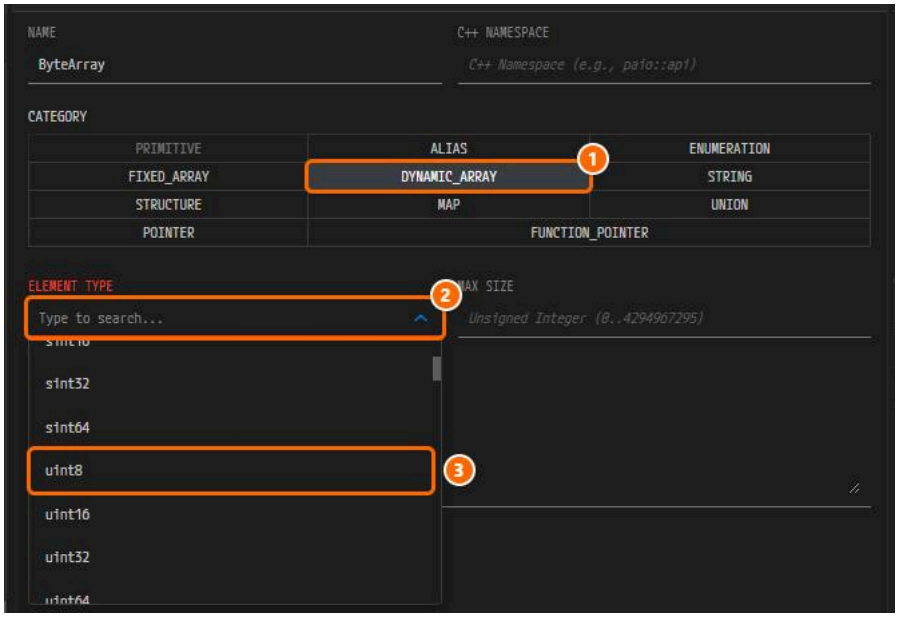


*Figure 4-3. ① **DYNAMIC_ARRAY**, ② **ELEMENT TYPE**, ③ `uint8`.*

5. Type `1` in **MAX SIZE** and `aa::type` in **C++ NAMESPACE**, pressing `Tab` after each. The row now reads `ByteArray [DYNAMIC_ARRAY]`.

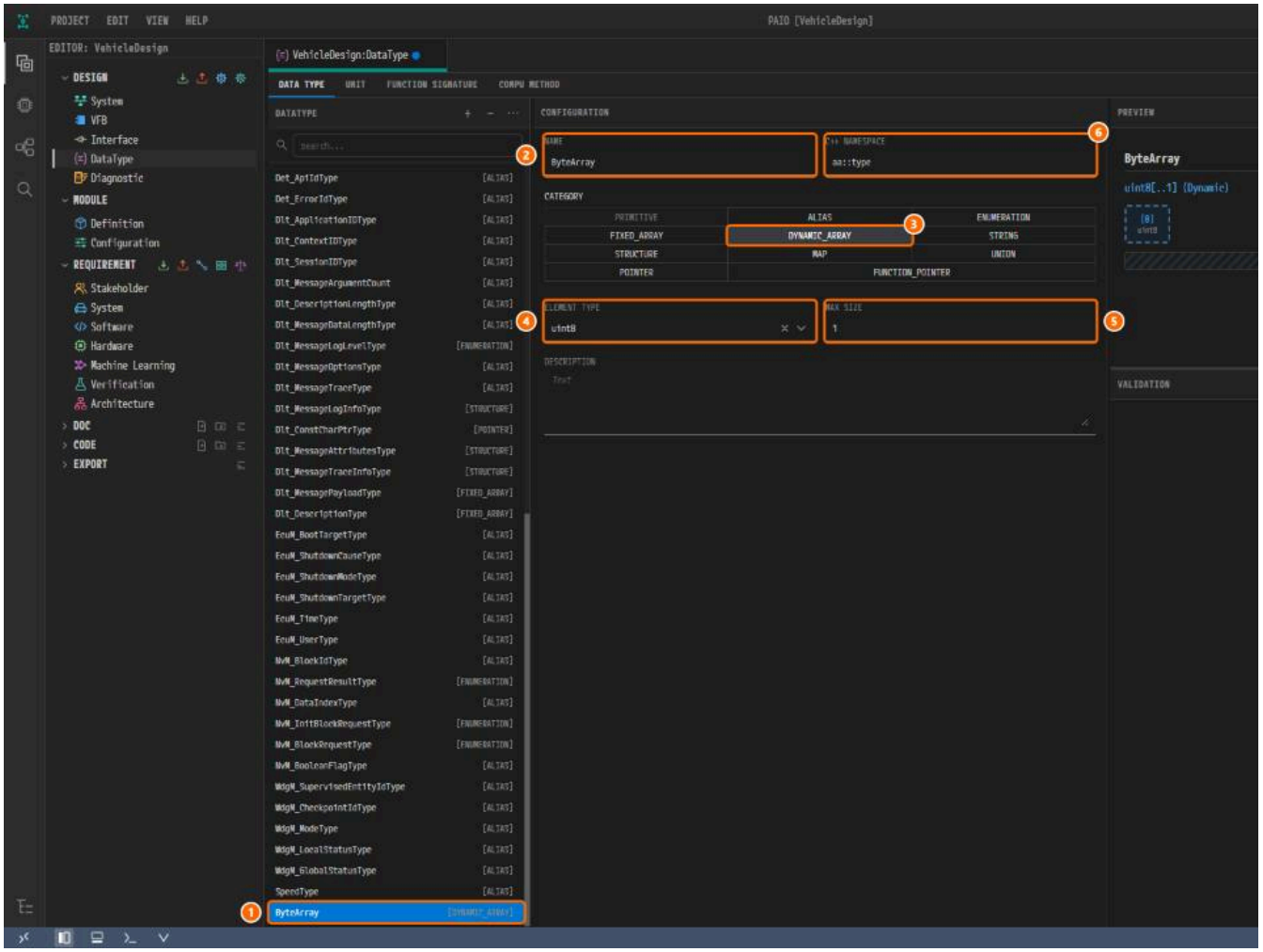


*Figure 4-4. ① `ByteArray [DYNAMIC_ARRAY]`, ② **NAME**, ③ **CATEGORY**, ④ **ELEMENT TYPE**, ⑤ **MAX SIZE**, ⑥ **C++ NAMESPACE**.*

Creating String — STRING

1. Click **+** again. Change **NAME** of the new entry `DataType67` to `String` and press `Tab`.

2. Click **STRING** in **CATEGORY**. This category has no **ELEMENT TYPE** field.

3. Type `256` in **MAX SIZE** and `aa::type` in **C++ NAMESPACE**, pressing `Tab` after each. If **MAX SIZE** stays empty, the default `255` is applied instead.

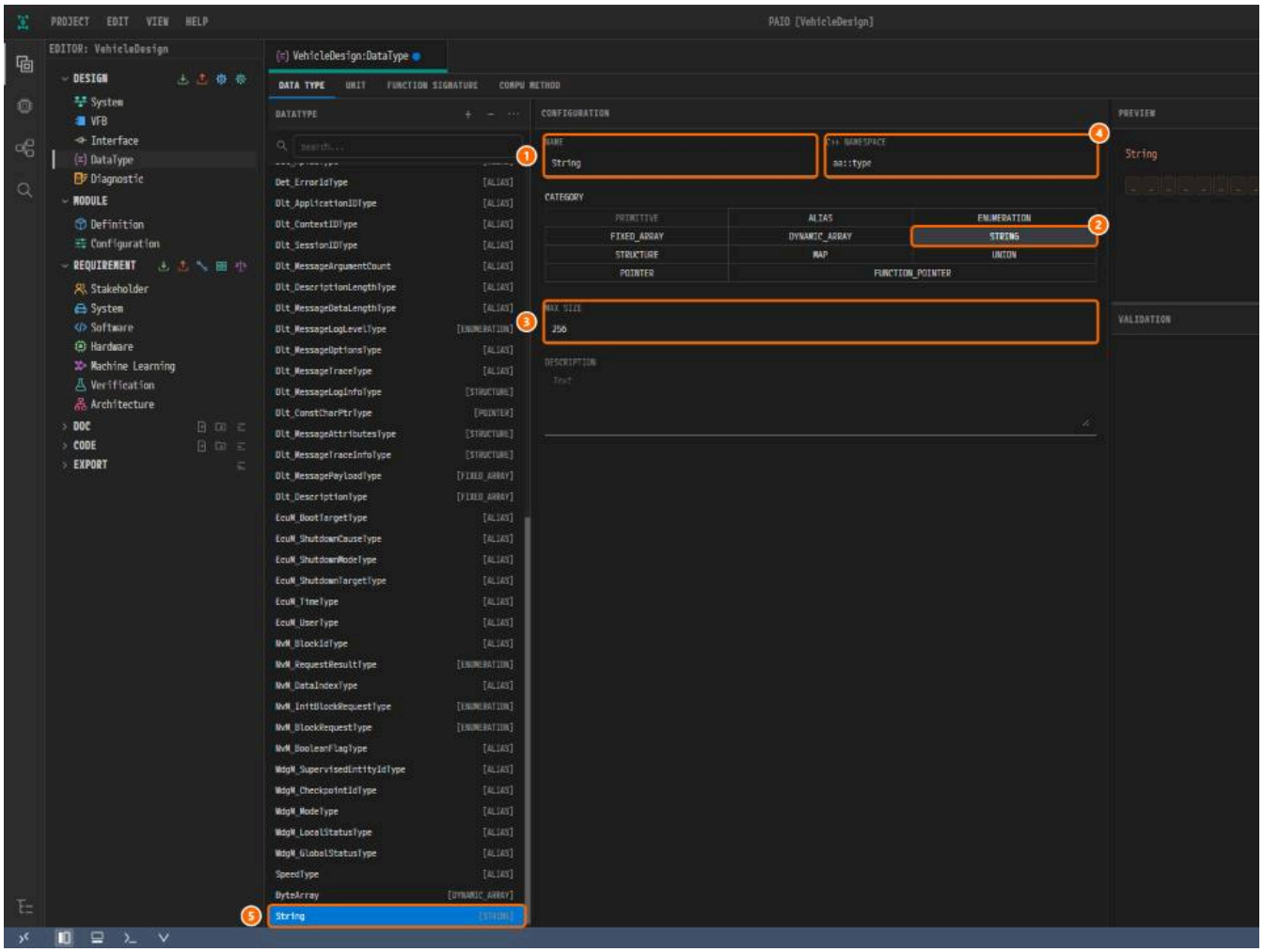


*Figure 4-5. ① **NAME**, ② **CATEGORY**, ③ **MAX SIZE**, ④ **C++ NAMESPACE**, ⑤ `String [STRING]`.*

Saving

1. Press `Ctrl+S`. The modified mark ● on the tab `VehicleDesign:DataType` disappears.

2. Click `ByteArray` in the list and check that its values were kept.

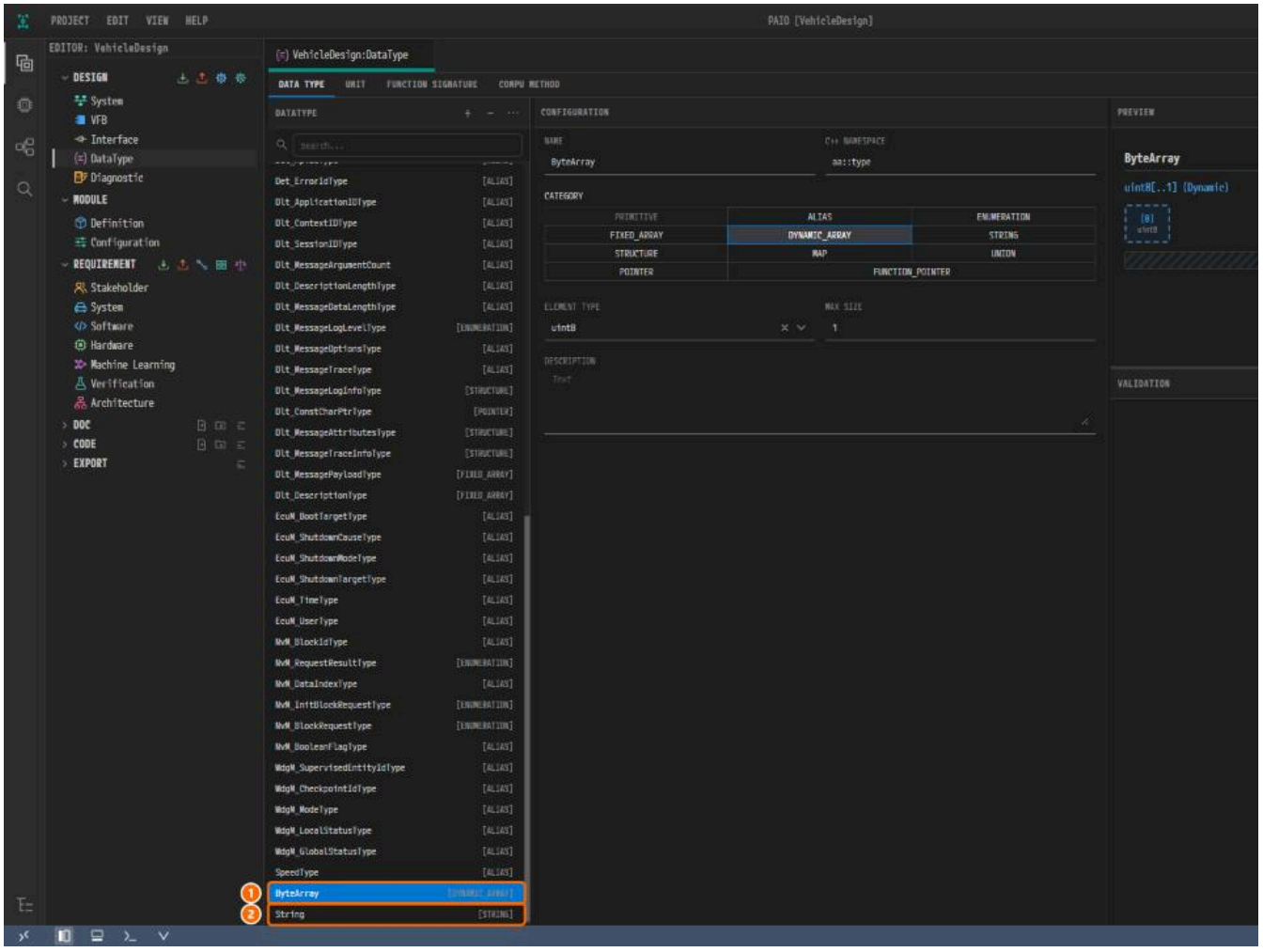


Figure 4-6. ① `ByteArray [DYNAMIC_ARRAY]`, ② `String [STRING]`.

Viewing the Validation Result

1. Click the **V** icon in the footer to open the **VALIDATION** panel. It shows `No validation issues.`.

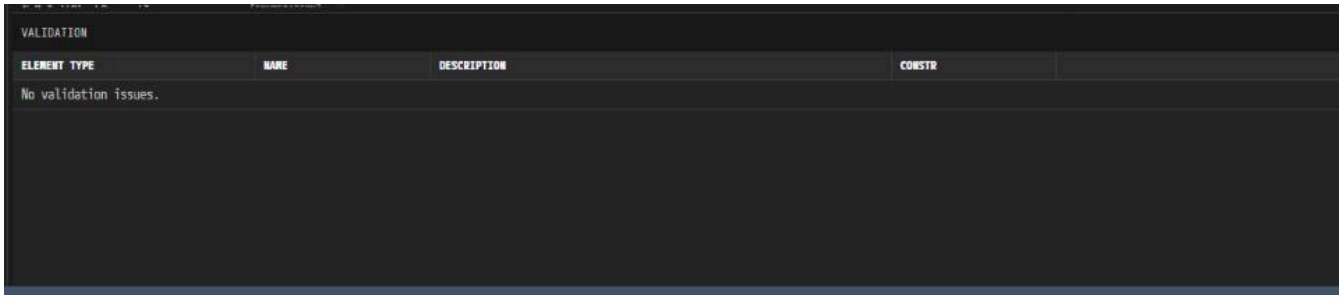


*Figure 4-7. The **VALIDATION** panel after saving.*

Why Is It Designed This Way?

- `ByteArray` **comes first.** The type list of the Interface editor shows only the types that exist in the DataType editor.
- **The built-in types are not entered.** The exporter writes them in the Machine Extract of Validation and Code Generation, under their C++ names such
- `String` **is not used by any interface.** It shows how a **STRING** type is entered, where **MAX SIZE** is the maximum text length.
- **The namespace** `aa::type`**.** It becomes the `NAMESPACES` of the export. Without it, no namespace is written.

Checklist

- [ ] The list ends with `ByteArray [DYNAMIC_ARRAY]` and `String [STRING]`, 68 entries in all.
- [ ] `ByteArray` has **ELEMENT TYPE** `uint8`, **MAX SIZE** `1` and **C++ NAMESPACE** `aa::type`.

- [ ] `String` has **MAX SIZE** `256` and **C++ NAMESPACE** `aa::type`.
- [ ] The tab carries no modified mark and the **VALIDATION** panel shows `No validation issues.`.

Next Step

In Service Interface you create the service interface `Sensor` and give its event the type `ByteArray`.

The Service Interface

What You Will Do

In this chapter you create the service interface `Sensor` with one event and one method. Later chapters give it a SOME/IP deployment and use it on the p

Before You Begin

- Start after finishing Data Types.
- You may close the tab `VehicleDesign:DataType`.

Element	NAME	Other values
Service interface	`Sensor`	**C++ NAMESPACE** `sensor::service`, **MAJOR VERSION** `1`, **MINOR VERSION** `0` (defaults)
Event	`SensorData`	**TYPE** `ByteArray`
Method	`Calculation`	**METHOD TYPE REQUEST AND RESPONSE** (default)

Create the five arguments of `Calculation` in this order.

#	DIRECTION	TYPE	NAME
1	**IN** (default)	`float32`	`front`
2	**IN** (default)	`float32`	`rear`
3	**IN** (default)	`float32`	`left`
4	**IN** (default)	`float32`	`right`
5	**OUT**	`float32`	`result`

Leave every **DESCRIPTION** empty. Press `Enter` after typing a name and `Tab` after typing the namespace.

Opening the Interface Editor

1. In the **EDITOR** view, click **Interface** under **DESIGN**. The tab `VehicleDesign:Interface` opens with the **SERVICE** tab active. The **INTERFACES** list is empt

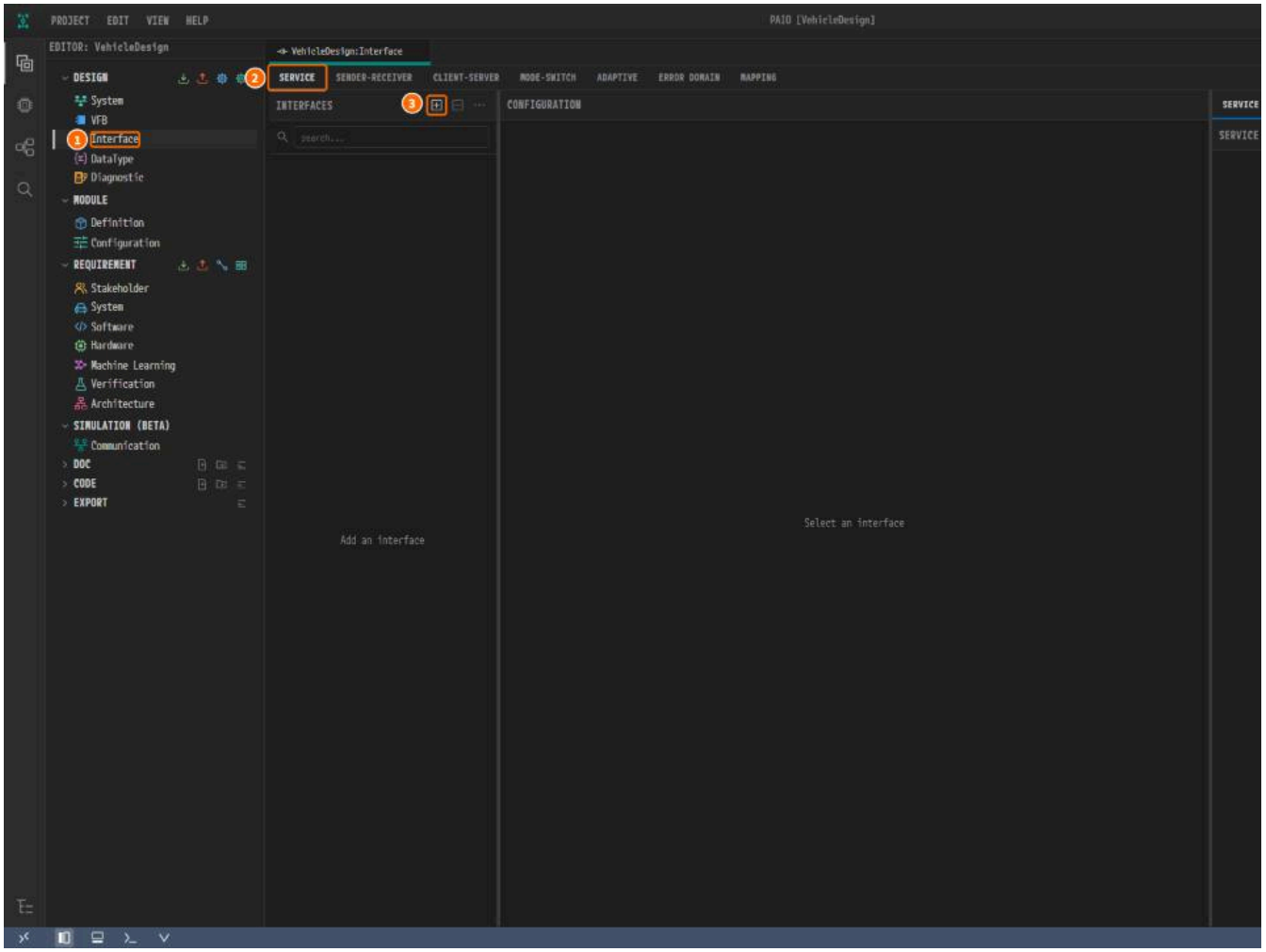


*Figure 5-1. ① **Interface** under **DESIGN**, ② **SERVICE**, ③ the **+** button.*

Creating the Service Interface Sensor

1. Click **+** in the **INTERFACES** header. A new interface `ServiceInterface_0` appears and is selected.

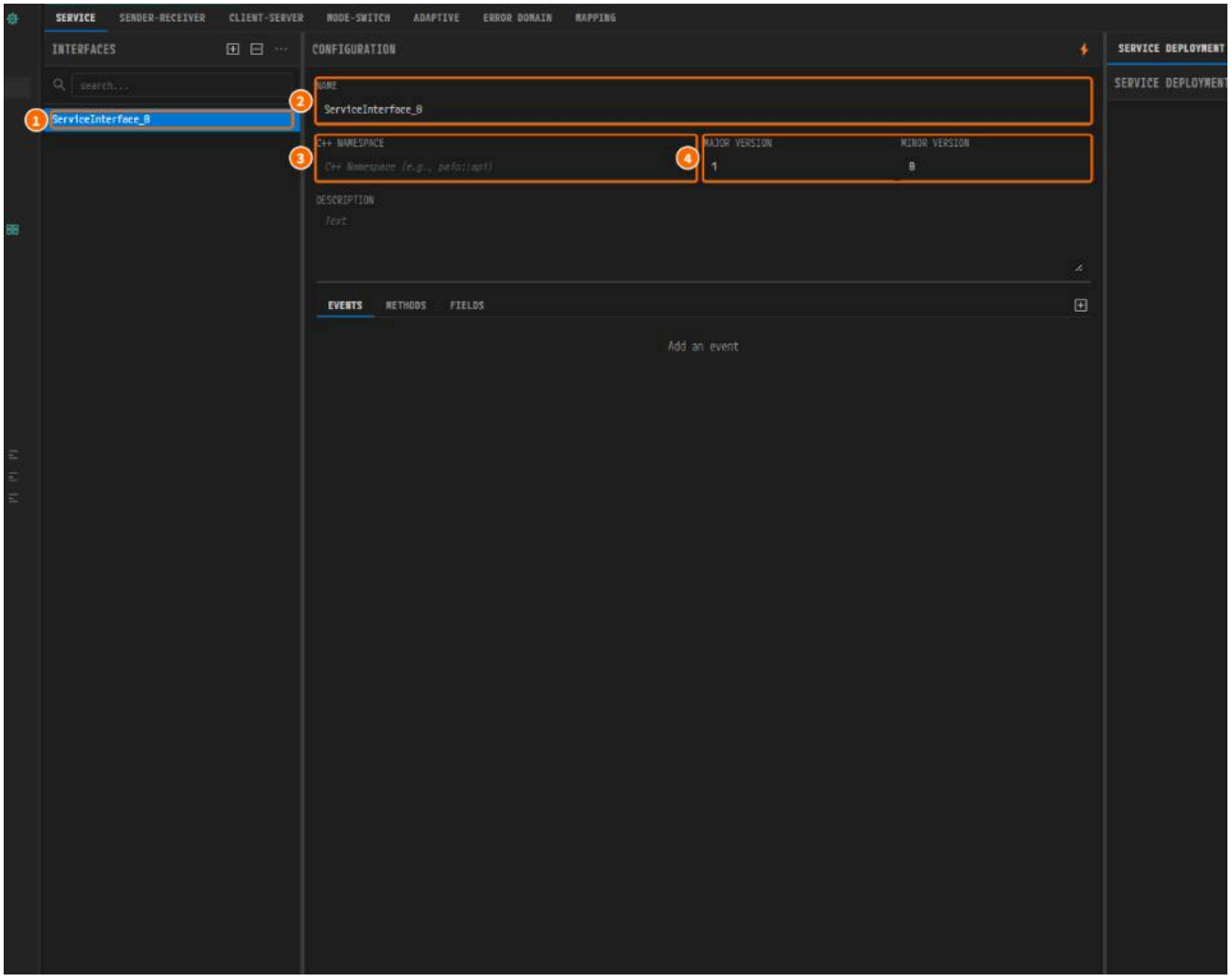


*Figure 5-2. ① `ServiceInterface_0`, ② **NAME**, ③ **C++ NAMESPACE**, ④ **MAJOR VERSION** and **MINOR VERSION**.*

2. Change **NAME** to `Sensor` and press `Enter`.

3. Type `sensor::service` in **C++ NAMESPACE** and press `Tab`. Leave the versions `1` and `0`.

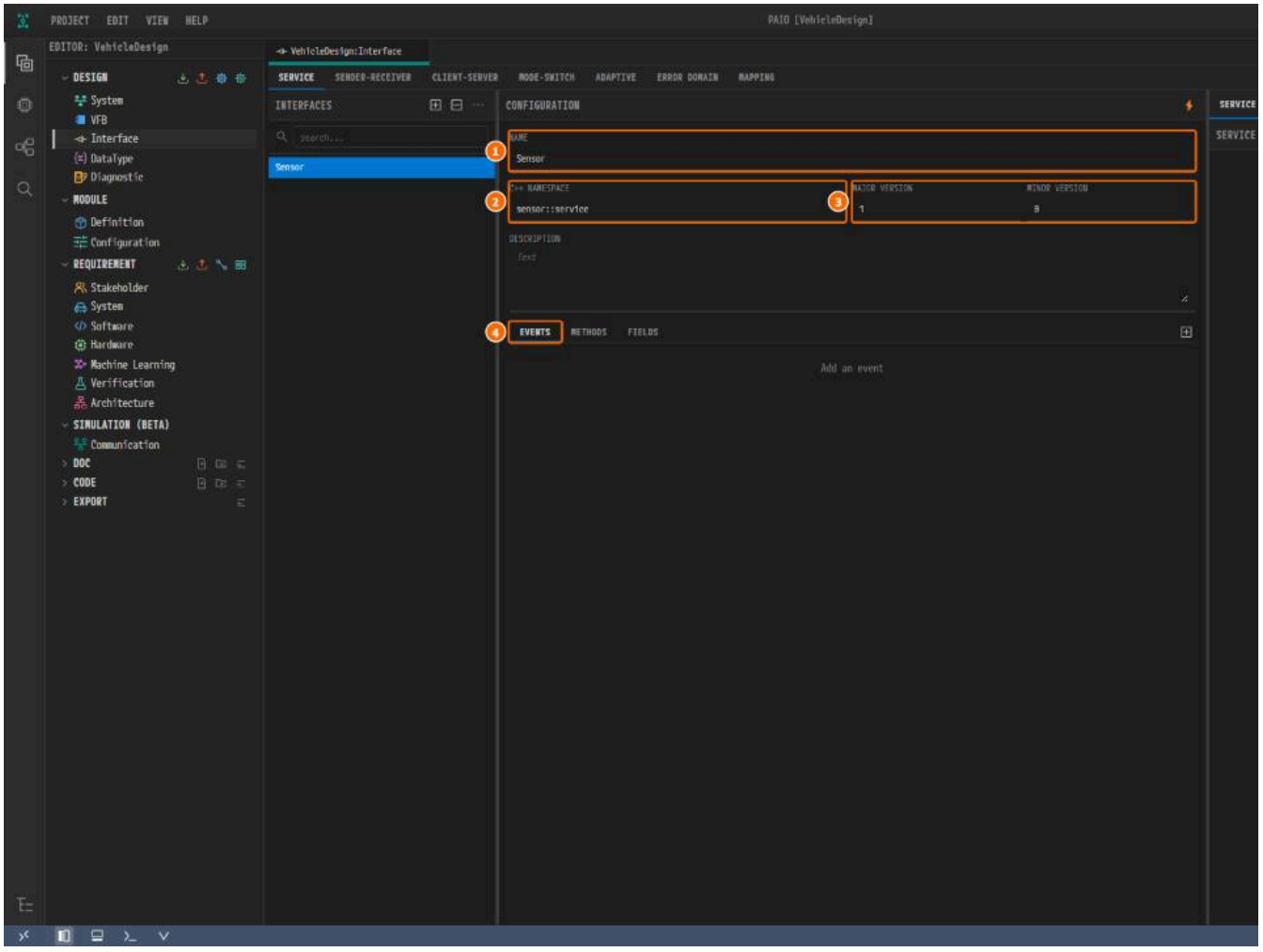


*Figure 5-3. ① **NAME**, ② **C++ NAMESPACE**, ③ **MAJOR VERSION** and **MINOR VERSION**, ④ **EVENTS**.*

Adding the Event SensorData

1. With the **EVENTS** subtab active, click the **+** at the right end of the subtab row. An expanded row `Event_0` of type `uint8` appears.
2. Change **NAME** to `SensorData` and press `Enter`.
3. Click the ⌄ of **TYPE**. Scroll to `ByteArray`, below `uint64` and `BswM_ModeGroupType`, and click it.

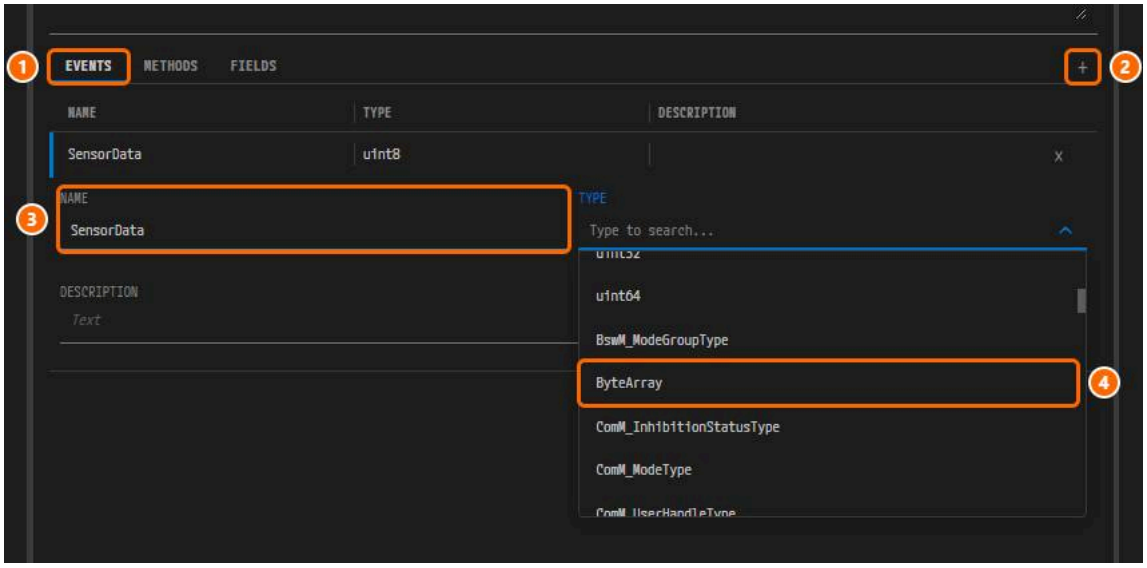


*Figure 5-4. ① **EVENTS**, ② **+**, ③ **NAME**, ④ `ByteArray`.*

4. The row reads `SensorData` `ByteArray`. The right panel switches to **DATA TYPE** and shows `ByteArray`. Change nothing there.

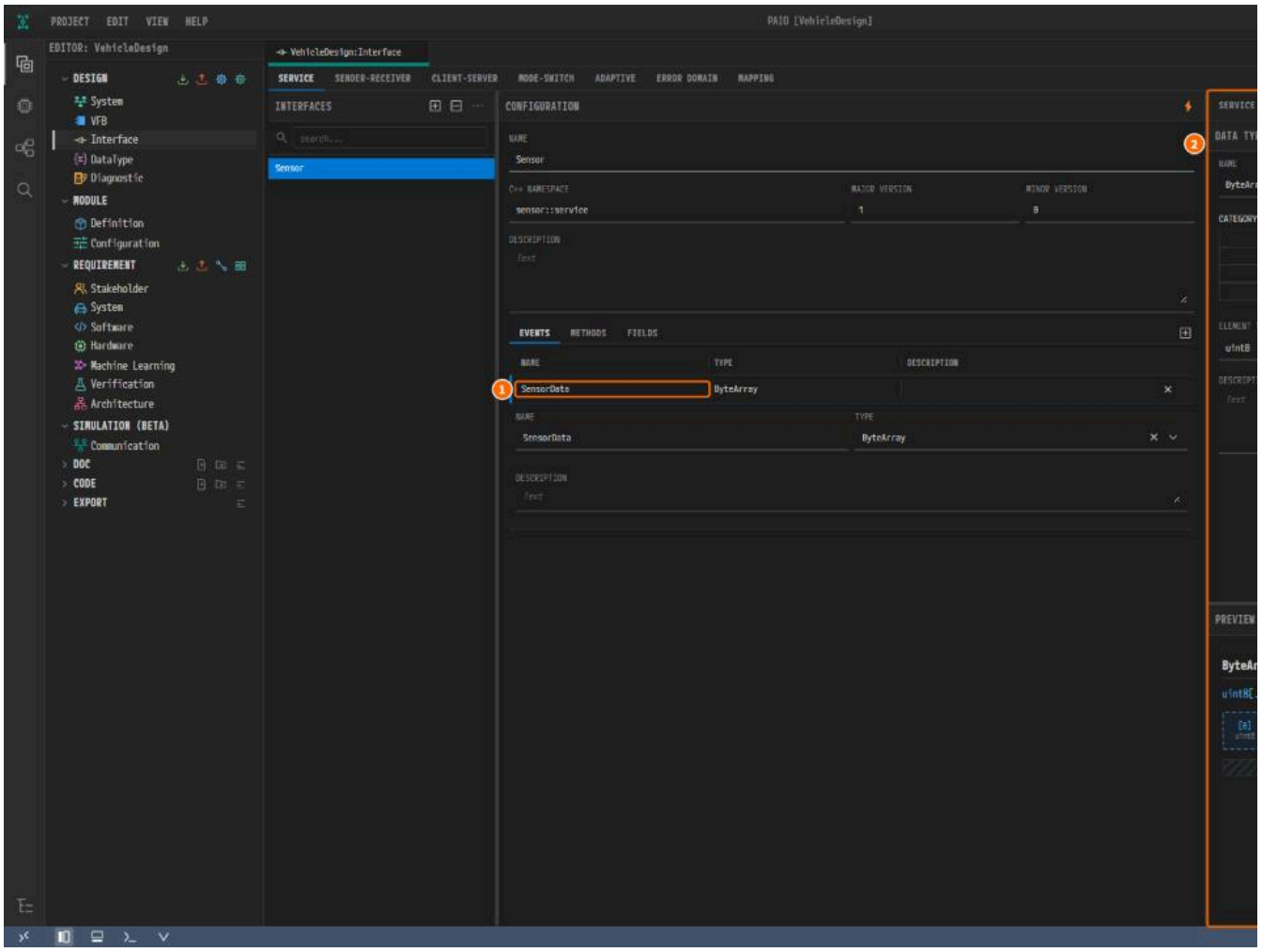


*Figure 5-5. ① The `SensorData` row, ② the **DATA TYPE** panel.*

Adding the Method Calculation

1. Click the **METHODS** subtab, then the **+** at the right end of the subtab row. An expanded row `Method_0` appears.
2. Change **NAME** to `Calculation` and press `Enter`. Leave **METHOD TYPE** at **REQUEST AND RESPONSE**.

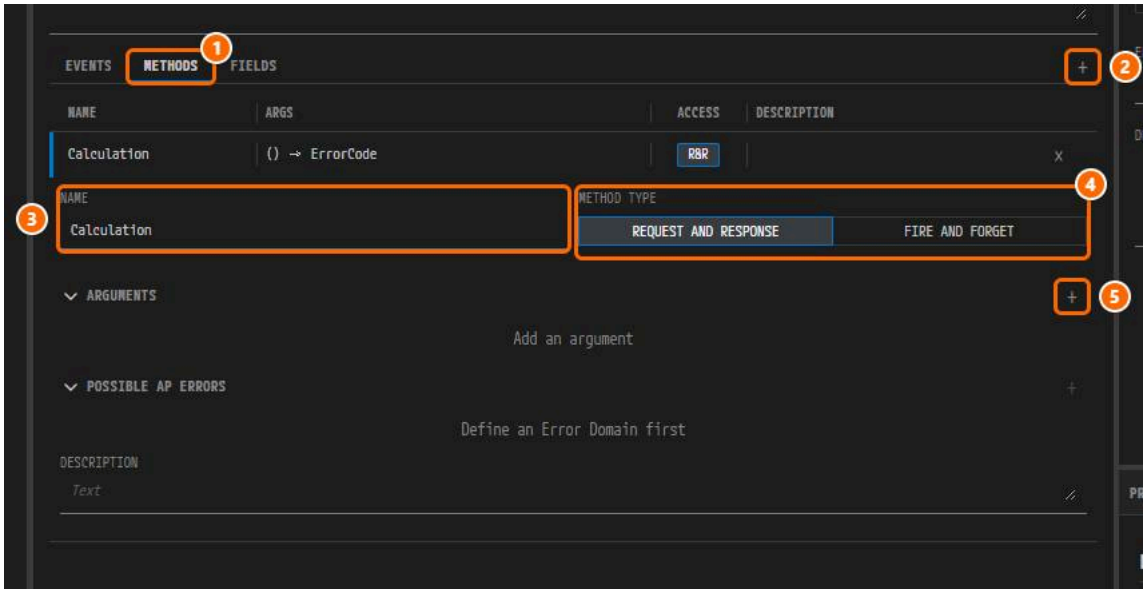


*Figure 5-6. ① **METHODS**, ② **+**, ③ **NAME**, ④ **METHOD TYPE**, ⑤ the **+** of **ARGUMENTS**.*

3. Click the **+** of **ARGUMENTS**. Row `1` appears with `IN`, `uint8` and `argument_0`.

4. Change its **NAME** to `front` and press `Enter`. Click the ⌄ of **TYPE** and click `float32`.

5. Add `rear`, `left` and `right` the same way.

6. Add row 5. Click the ⌄ of its **DIRECTION** and click `OUT`. While the list is open, the cell shows the hint `Type`.

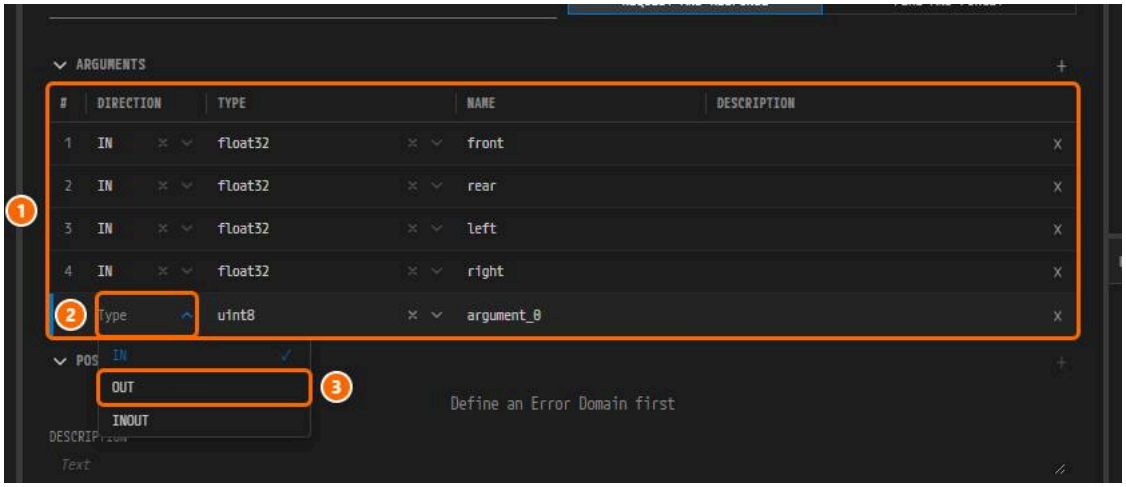


*Figure 5-7. ① Rows 1 to 4, ② **DIRECTION** of row 5, ③ `OUT`.*

7. Set **NAME** of row 5 to `result` and **TYPE** to `float32`. **ARGS** in the method row reads `(float32 front, float32 rear, float32 left, float32 rig`

Saving

1. Click an empty spot in the tab bar, then press `Ctrl+S`. The modified mark ● on the tab disappears.

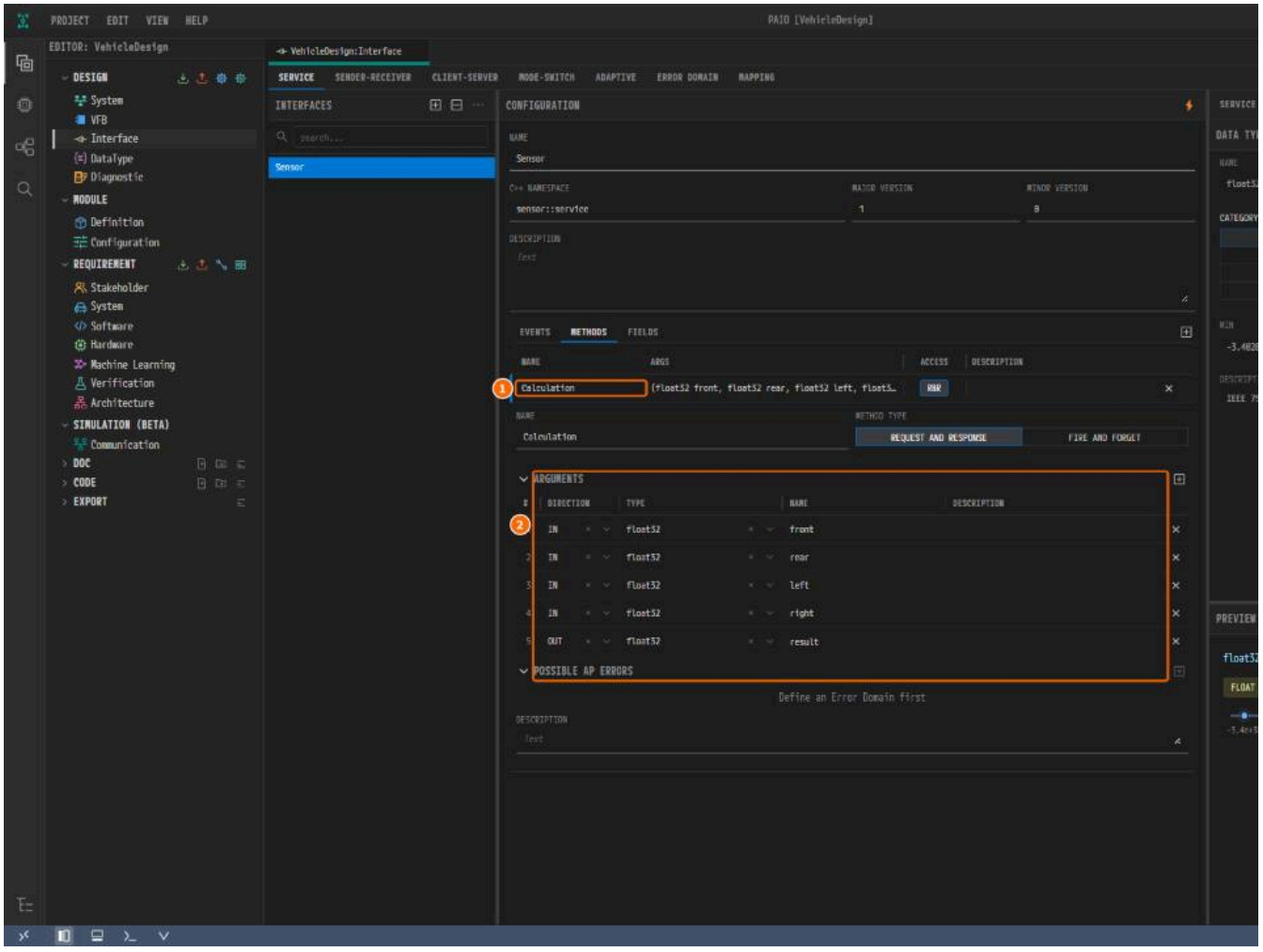


Figure 5-8. ① The `Calculation` row, ② the five arguments.

Viewing the Validation Result

1. Click the **V** icon in the footer to open the **VALIDATION** panel. It shows `No validation issues.`.

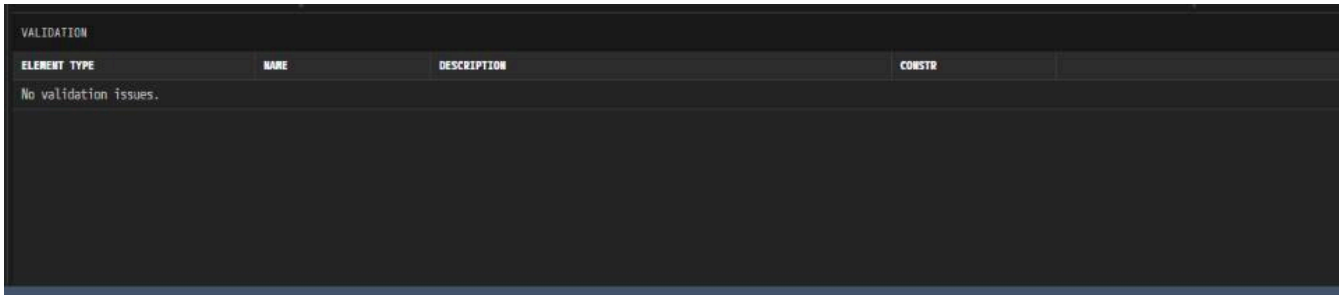


*Figure 5-9. The **VALIDATION** panel after saving.*

Why Is It Designed This Way?

- **One service interface.** `Sensor` is the only `SERVICE-INTERFACE` in the Machine Extract of Validation and Code Generation.
- **The namespace `sensor::service`.** It becomes the `NAMESPACES` of the export.
- **Defaults are kept.** The version `1` / `0` and **REQUEST AND RESPONSE** are already the values the design needs.
- **`float32` in the editor.** It comes out under its C++ name `float`. The argument order must match the table.
- **No deployment yet.** The **SERVICE DEPLOYMENT** panel stays empty until SOME/IP Deployment and Service Instances.

Checklist

- [ ] The **INTERFACES** list holds only `Sensor`, with **C++ NAMESPACE** `sensor::service`.

- [ ] **EVENTS** holds `SensorData` `ByteArray`.
- [ ] **METHODS** holds `Calculation` with the five arguments of the table.
- [ ] The tab carries no modified mark and the **VALIDATION** panel shows `No validation issues.`.

Next Step

In SOME/IP Deployment and Service Instances you create the SOME/IP deployment (service ID `11`) and the service instances of `Sensor`.

SOME/IP Deployment and Service Instances

What You Will Do

In this chapter you give the interface `Sensor` a SOME/IP deployment and three service instances: the provider `Provider_Sensor` and the consumers `Ac`

Before You Begin

- Start after finishing The Service Interface.
- Keep the tab `VehicleDesign:Interface` open. If you closed it, click **Interface** under **DESIGN** in the **EDITOR** view.

Element	Default name → NAME to type	Other values
SOME/IP deployment	`Sensor` (unchanged)	**SERVICE ID** `11`, **MAJOR VERSION** `1` / **MINOR VERSION** `0` (defaults)
Event deployment	`SensorData` (automatic)	**EVENT ID** `1`, **TRANSPORT UDP** (all defaults)
Method deployment	`Calculation` (automatic)	**METHOD ID** `1`, **TRANSPORT UDP** (all defaults)
Event group	`EG_SensorData` → `Eventgroup_SensorData`	**EVENTGROUP ID** `1`, **EVENTS** `SensorData` (all defaults)
Provider	`Provider_Sensor_1` → `Provider_Sensor`	**INSTANCE ID** `1` (default)
Consumer 1	`Consumer_Sensor_1` → `Actuator_Sensor`	**REQUIRED INSTANCE ID** `ANY` (default)
Consumer 2	`Consumer_Sensor_2` → `Fusion_Sensor`	**REQUIRED INSTANCE ID** `ANY` (default)

The SD (Service Discovery) timing goes into **SD SERVER CONFIG** of the provider and **SD CLIENT CONFIG** of the consumers. Keep every other SD field at its de

Field	Default	Value to enter	Provider	Consumer
INITIAL DELAY MIN	`10 ms`	`10 us`	enter	enter
INITIAL DELAY MAX	`100 ms`	`100 us`	enter	enter
REPETITION BASE DELAY	`200 ms`	`200 us`	enter	enter
OFFER CYCLIC DELAY	`500 ms`	`1 ms`	enter	no field

For a time field, pick the unit to the right of the value first, then type the number. Changing the unit only converts the displayed number.

> **Note.** After each SD value, press `Tab` and click `Sensor` in the **INTERFACES** list once, so the panel is redrawn. Earlier PAIO builds lost SD values edited one

Creating the SOME/IP Deployment

1. In the **SERVICE** tab, click `Sensor` in the **INTERFACES** list. If the right panel shows **DATA TYPE**, click **SERVICE DEPLOYMENT**.

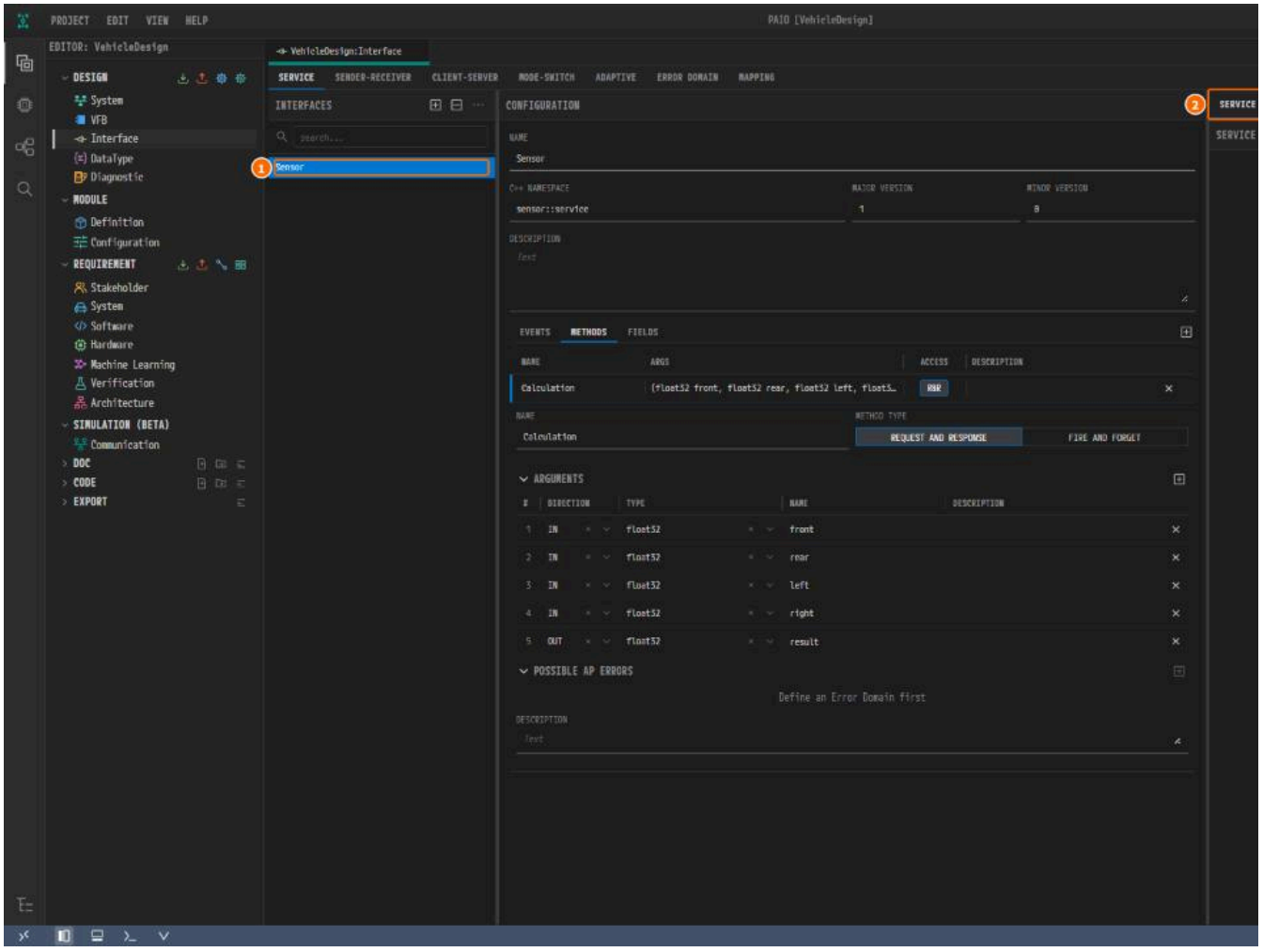


*Figure 6-1. ① `Sensor`, ② **SERVICE DEPLOYMENT**, ③ the **+** button.*

2. Click **+** in the **SERVICE DEPLOYMENT** header.

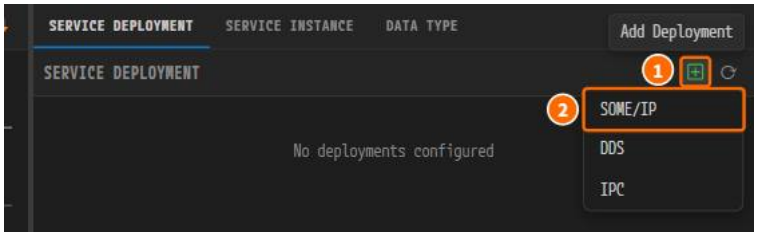


*Figure 6-2. ① **+**, ② **SOME/IP**.*

3. Click **SOME/IP**. A collapsed card `SOME/IP` `Sensor` appears. The deployments of `SensorData` and `Calculation` are created with it.



Figure 6-3. ① The card `SOME/IP` `Sensor`. The service ID is still the default `0x0001`.

4. Click the card header to expand it. Change **SERVICE ID** to `11`, with **DEC** active, and press `Tab`.

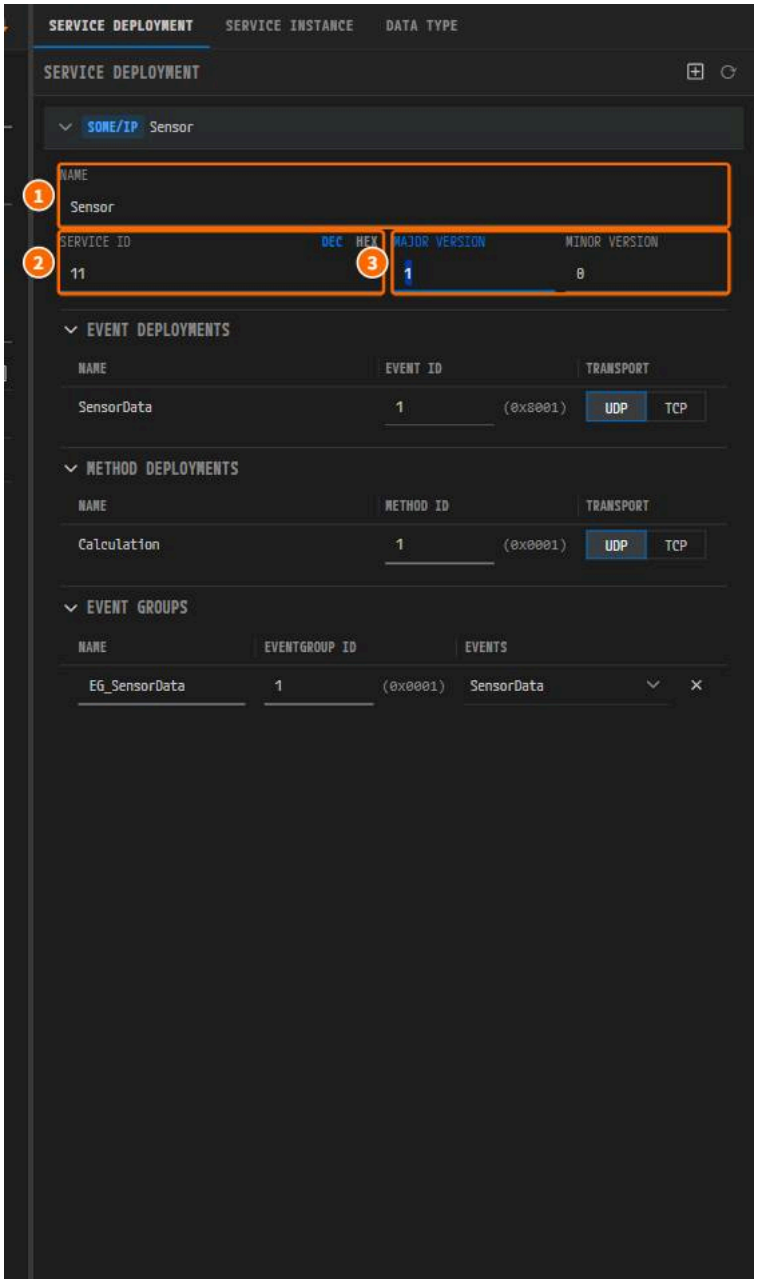


*Figure 6-4. ① **NAME**, ② **SERVICE ID**, ③ **MAJOR VERSION** and **MINOR VERSION**.*

5. Check that **EVENT DEPLOYMENTS** holds `SensorData` and **METHOD DEPLOYMENTS** holds `Calculation`, both with ID `1` and **UDP**. Leave them.

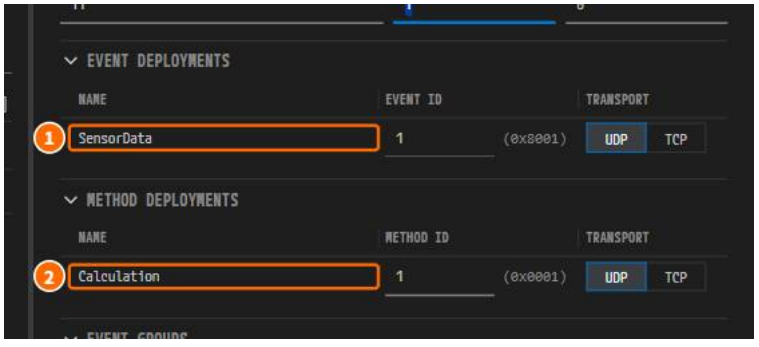


Figure 6-5. ① `SensorData`, ② `Calculation`. `(0x8001)` only shows how SOME/IP transmits the event; the event ID is `1`.

6. In **EVENT GROUPS**, change **NAME** of `EG_SensorData` to `Eventgroup_SensorData` and press `Tab`.



*Figure 6-6. ① **NAME**, ② **EVENTGROUP ID**, ③ **EVENTS**. The cell cuts off the last letter of the name.*

Adding the Service Instances

1. Hover over the card header and click the **+** that appears at its right end.



*Figure 6-7. ① **+**, ② **Provider** and **Consumer**.*

2. Click **Provider**. Then add two consumers: click **+** and **Consumer**, twice.

3. Click the card header to collapse it. The summary lists `Provider_Sensor_1`, `Consumer_Sensor_1` and `Consumer_Sensor_2`.



Figure 6-8. ① `Provider_Sensor_1`, ② `Consumer_Sensor_1`, ③ `Consumer_Sensor_2`.

Configuring the Provider Provider_Sensor

1. Click `Provider_Sensor_1` in the summary. The right panel switches to **SERVICE INSTANCE** with the provider card expanded.

2. Change **NAME** to `Provider_Sensor` and press `Tab`. Leave **INSTANCE ID** `1`.

3. Expand **SD SERVER CONFIG** and enter the four values of the table. If the card collapses after you click `Sensor`, expand it and **SD SERVER CONFIG** agai

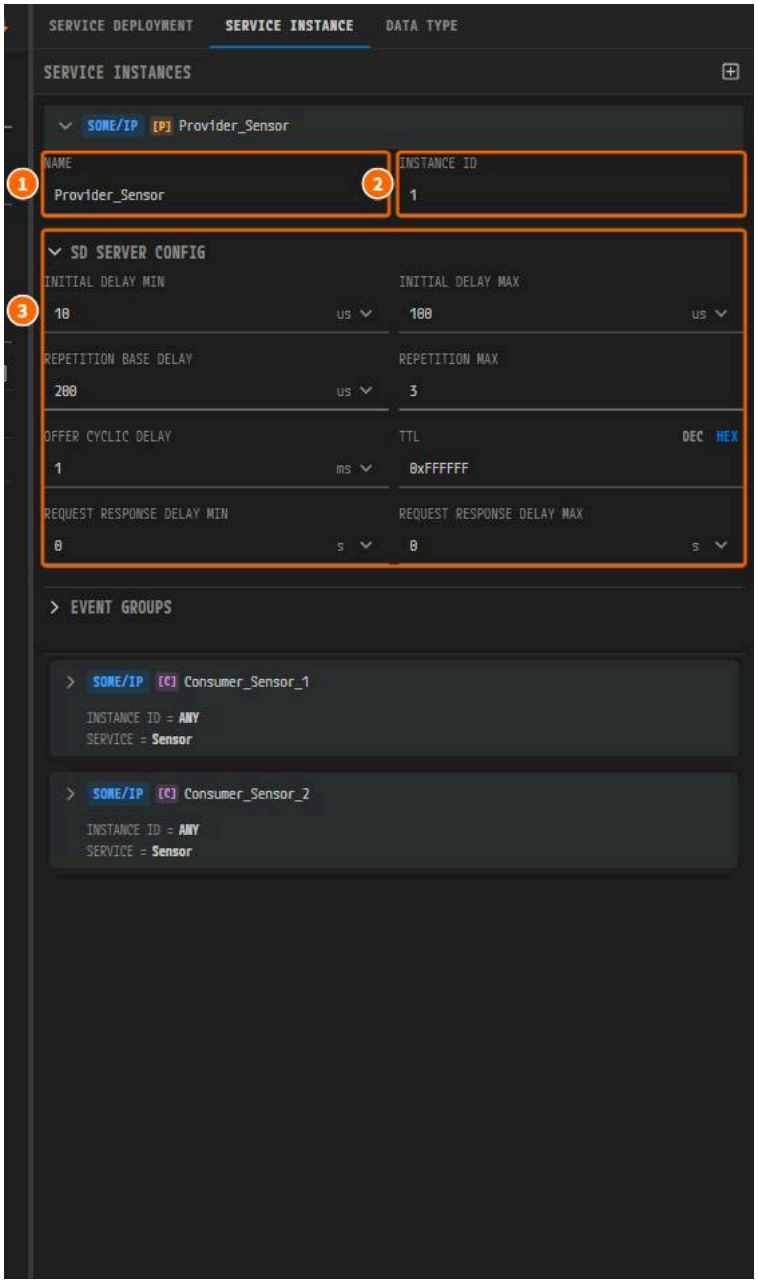


*Figure 6-9. ① **NAME**, ② **INSTANCE ID**, ③ **SD SERVER CONFIG**.*

4. Expand **EVENT GROUPS** and the row `Eventgroup_SensorData`. Leave everything at its default, **UNICAST**.

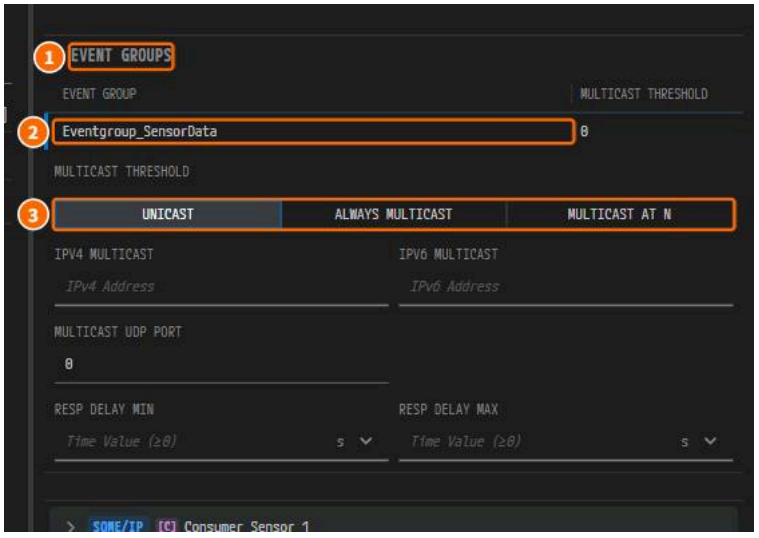


*Figure 6-10. ① **EVENT GROUPS**, ② `Eventgroup_SensorData`, ③ **MULTICAST THRESHOLD**.*

Configuring the Consumers Actuator_Sensor and Fusion_Sensor

1. Expand the card `Consumer_Sensor_1`. Change **NAME** to `Actuator_Sensor` and press `Tab`. Leave **REQUIRED INSTANCE ID** `ANY`.
2. Expand **SD CLIENT CONFIG** and enter the three values of the table the same way.

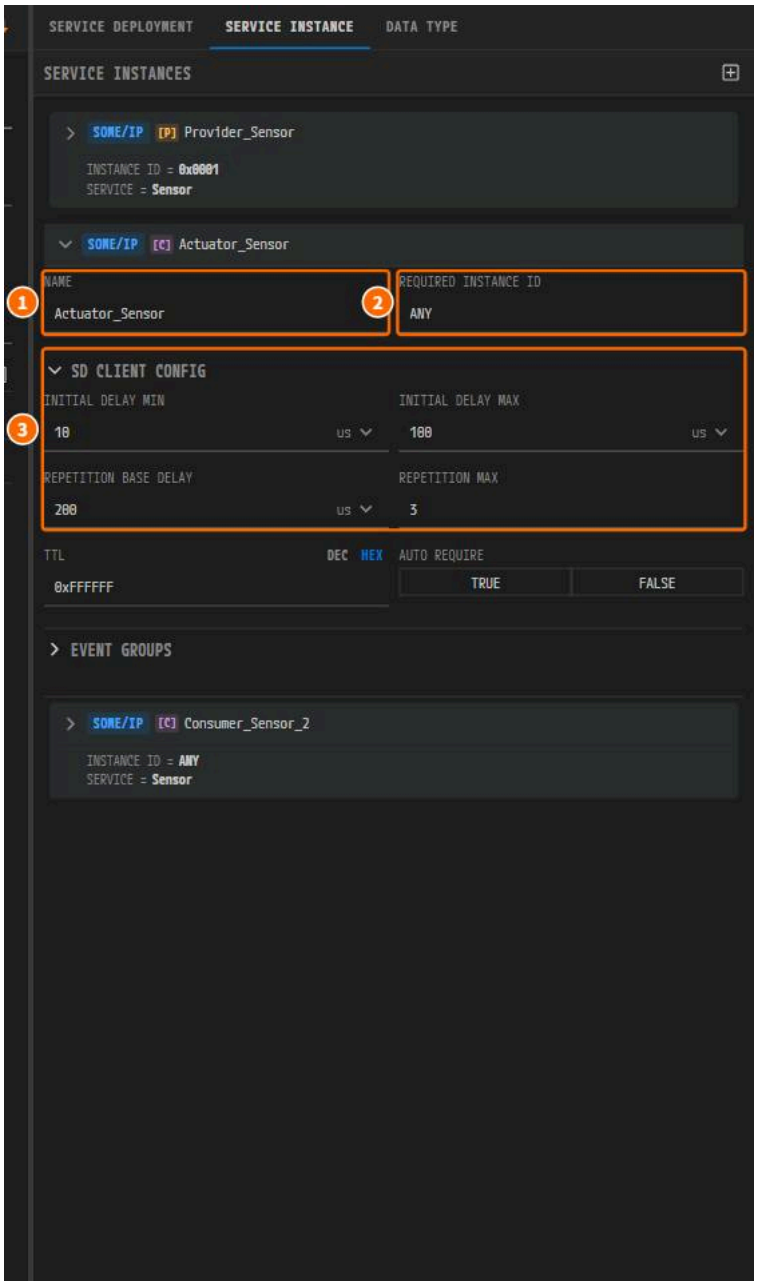


*Figure 6-11. ① **NAME**, ② **REQUIRED INSTANCE ID**, ③ **SD CLIENT CONFIG**.*

3. Configure `Consumer_Sensor_2` the same way with the name `Fusion_Sensor`.

Saving

1. Click an empty spot in the tab bar and press `Ctrl+S`. The modified mark ● disappears and `design/SERVICE.io.model` is created.
2. Collapse the expanded cards.

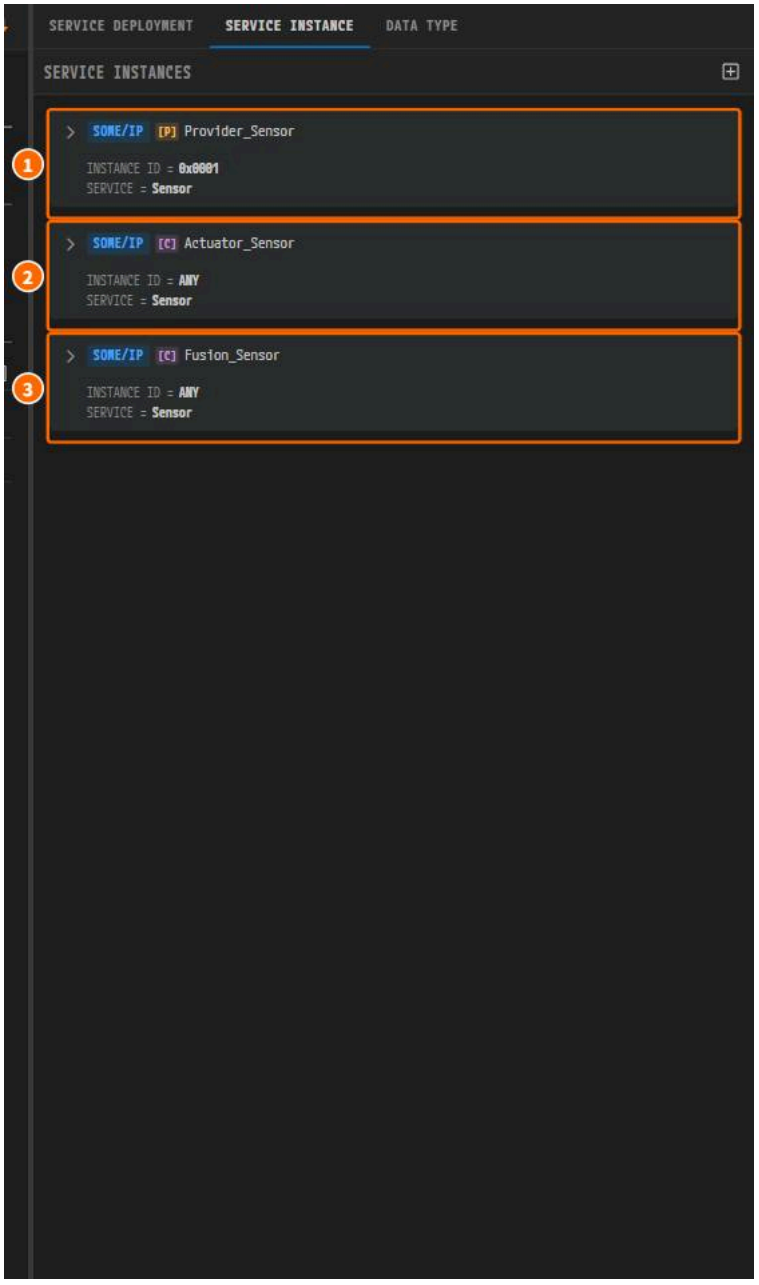


Figure 6-12. ① `Provider_Sensor`, ② `Actuator_Sensor`, ③ `Fusion_Sensor`.

The save also derives the Classic Platform signals in `SIGNAL.io.model` and the interfaces `Sensor_SR` and `Sensor_CS` automatically. Leave them.

Viewing the Validation Result

1. Click the **V** icon in the footer to open the **VALIDATION** panel. It shows `No validation issues.`.

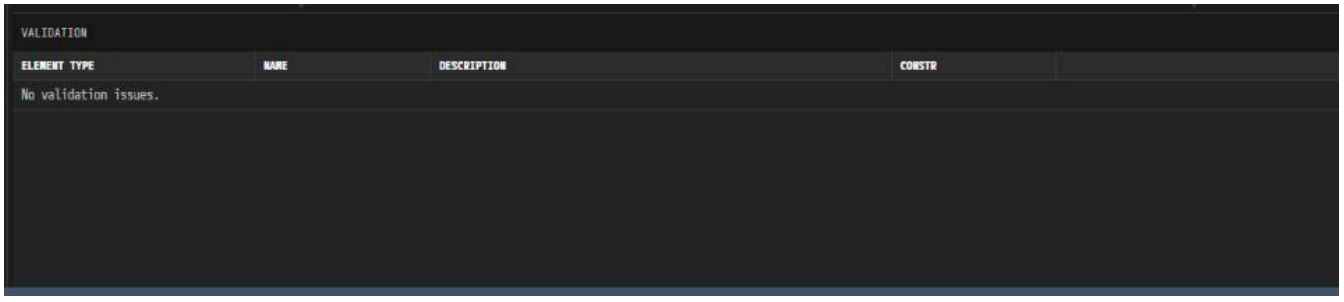


*Figure 6-13. The **VALIDATION** panel after saving.*

Why Is It Designed This Way?

- **Only the service ID is entered.** The editor names the deployment after the interface and creates the event and method deployments with the neede
- **The event group name.** The exporter of Validation and Code Generation looks up the events of a group by the rule `Eventgroup_{event}`.
- **The instance names.** They become the `SHORT-NAME`s of the instances in the export.
- **Short SD timing.** The entered values are much shorter than the defaults.
- **No machine binding yet.** It is created in Binding Service Instances to the Connector.

Checklist

- [ ] The card `SOME/IP` `Sensor` has **SERVICE ID** `11` and the event group `Eventgroup_SensorData`.
- [ ] **SERVICE INSTANCE** shows `[P]` `Provider_Sensor`, `[C]` `Actuator_Sensor` and `[C]` `Fusion_Sensor`.
- [ ] The SD values of each card match the table.
- [ ] The tab carries no modified mark and the **VALIDATION** panel shows `No validation issues.`.

Next Step

In Binding Service Instances to the Connector you bind the three instances to the Ethernet connector and assign their UDP ports.

Binding Service Instances to the Connector

What You Will Do

In this chapter you bind the three service instances from SOME/IP Deployment and Service Instances to the Ethernet connector `EthernetConnector` an SOME/IP Discovery Port and UDP Endpoints.

Before You Begin

- Start after finishing SOME/IP Deployment and Service Instances.
- The **VALIDATION** panel shows `No validation issues.`.

Instance	Role	UDP PORT	TCP PORT
`Provider_Sensor`	Provider (badge **P**)	`20003`	leave empty
`Actuator_Sensor`	Consumer (badge **C**)	`20001`	leave empty
`Fusion_Sensor`	Consumer (badge **C**)	`20002`	leave empty

Opening the SERVICE INSTANCES of the Connector EthernetConnector

1. In the **EDITOR** view, click **System** under **DESIGN**. The tab `VehicleDesign:System` opens.
2. Click the connector `EthernetConnector` on the border of the node `HPC [HPC]`. The **ETHERNET CONNECTOR** panel opens on the right.
3. Click the **SERVICE INSTANCES** header to expand it. It shows `No service instances defined`.

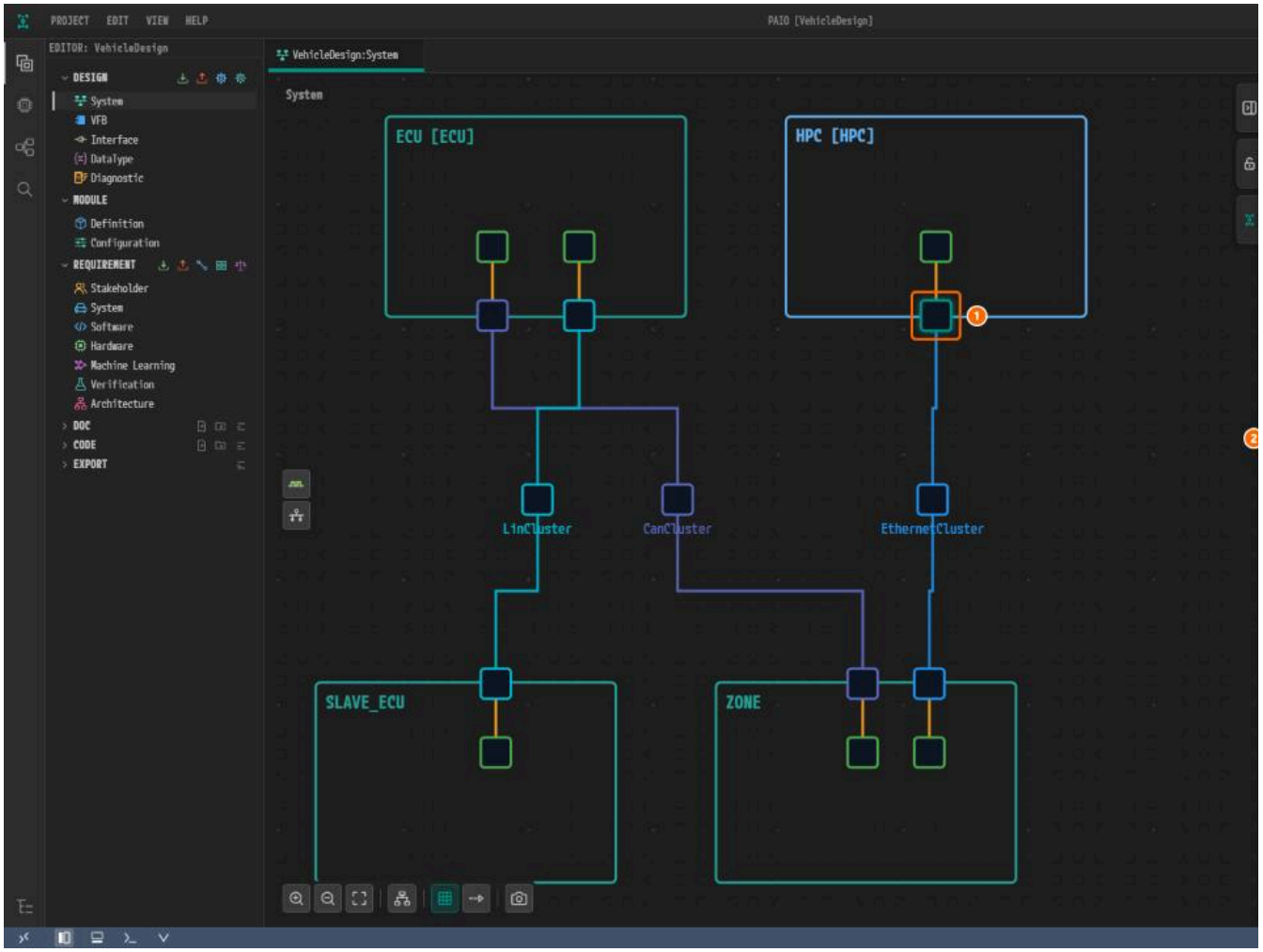


*Figure 7-1. ① The connector `EthernetConnector`, ② **SERVICE INSTANCES**, ③ the **+** button.*

Binding the Three Service Instances

1. Click the **+** button in the **SERVICE INSTANCES** header. The modal **Select Service Instance** opens with three rows.
2. Check all three rows and click **Bind**.

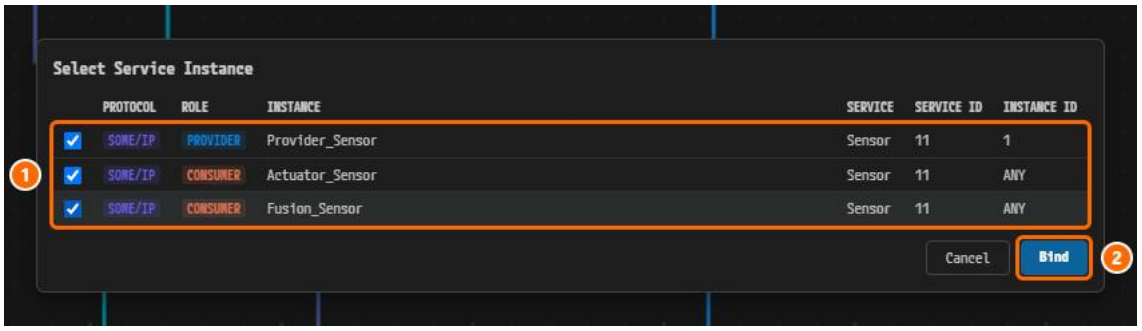


*Figure 7-2. ① The three rows, all checked, ② **Bind**.*

3. Three cards appear. The **V** icon in the footer shows a red badge `3`: one error for each binding without a UDP port. Each port you pick removes one.

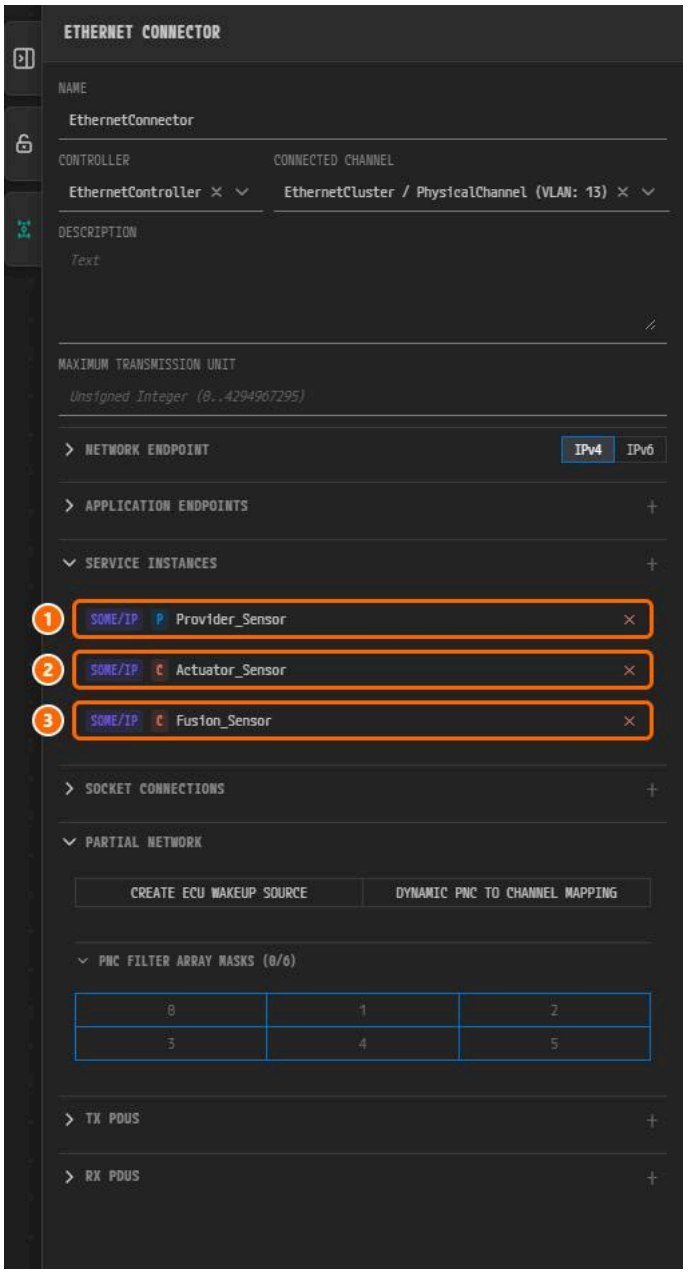


Figure 7-3. ① `Provider_Sensor`, ② `Actuator_Sensor`, ③ `Fusion_Sensor`. No card has a port yet.

Assigning the UDP Ports

Follow `Provider_Sensor` first.

1. Click the card `Provider_Sensor` to expand it. Click the ▾ button to the right of **UDP PORT**.

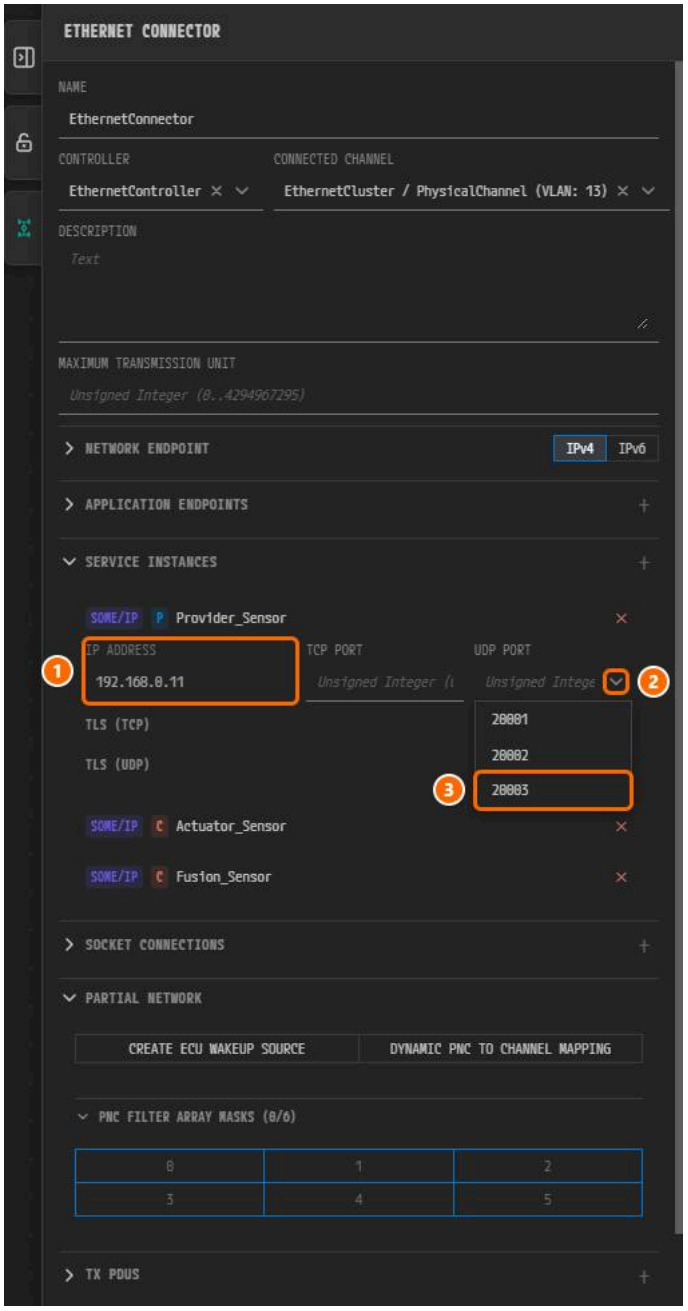


*Figure 7-4. ① **IP ADDRESS** (read-only), ② the ▾ button, ③ `20003`.*

2. Click `20003`. Leave **TCP PORT** and the rows **TLS (TCP)** and **TLS (UDP)** empty.
3. Click the connector `EthernetConnector` again and expand the card. Its summary line now shows `UDP:20003`.

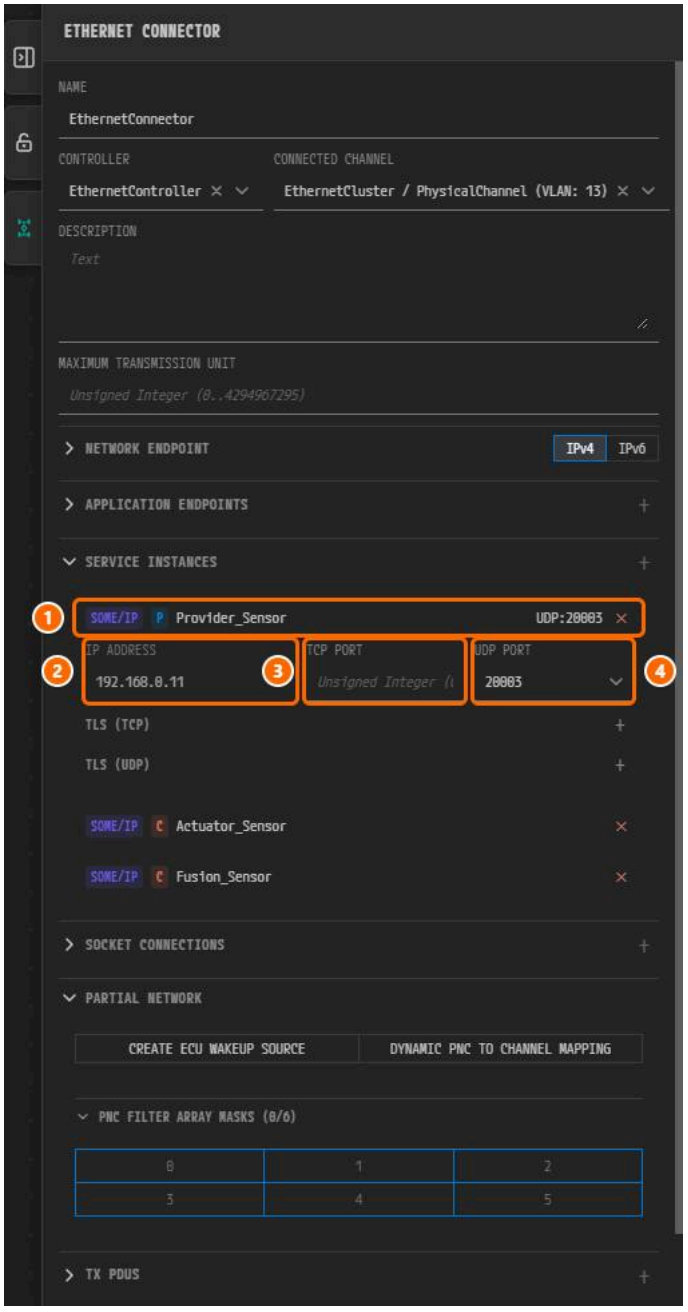


*Figure 7-5. ① `UDP:20003`, ② **IP ADDRESS**, ③ **TCP PORT** (empty), ④ **UDP PORT** = `20003`.*

4. Give `Actuator_Sensor` the port `20001` and `Fusion_Sensor` the port `20002` the same way. The badge on the **V** icon disappears.

5. Click the connector again and expand **APPLICATION ENDPOINTS**. It still holds only the three endpoints from chapter 3.

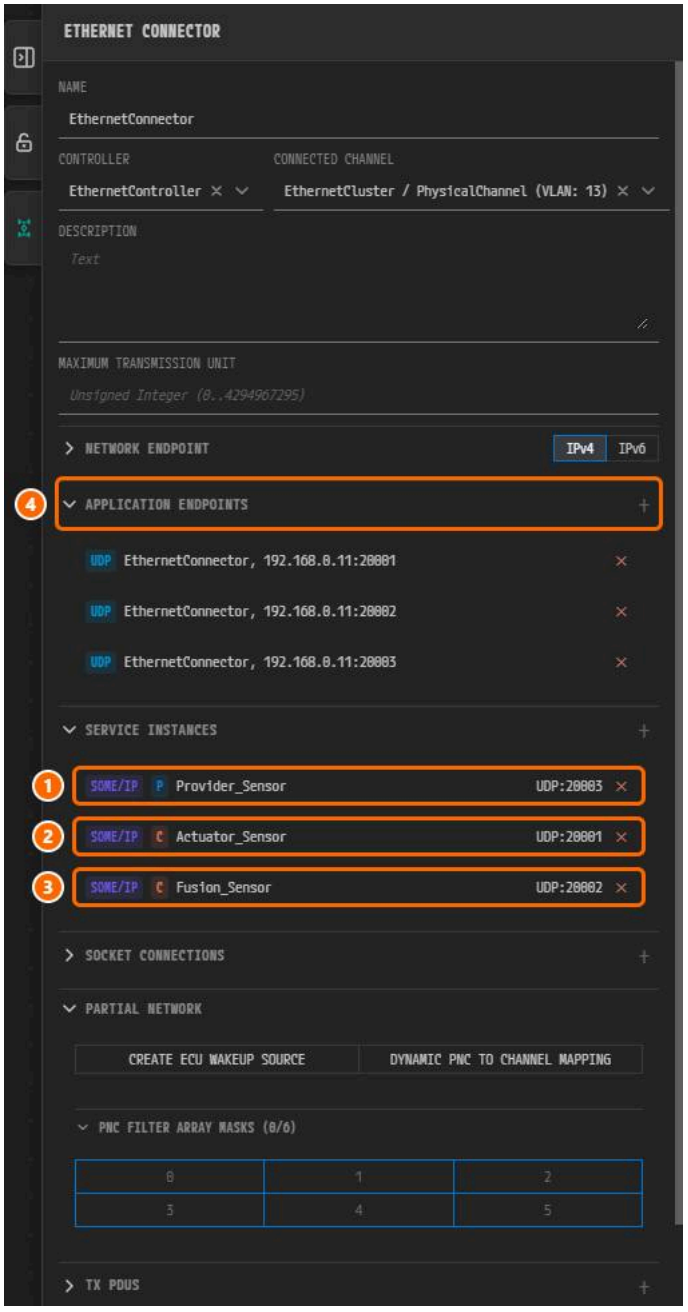


*Figure 7-6. ① `Provider_Sensor` `UDP:20003`, ② `Actuator_Sensor` `UDP:20001`, ③ `Fusion_Sensor` `UDP:20002`, ④ **APPLICATION ENDPOINTS**, unchange*

Saving

1. Click an empty spot on the canvas and press `Ctrl+S`. The modified mark ● on the tab `VehicleDesign:System` disappears.
2. Click the connector `EthernetConnector` again and check the three cards.

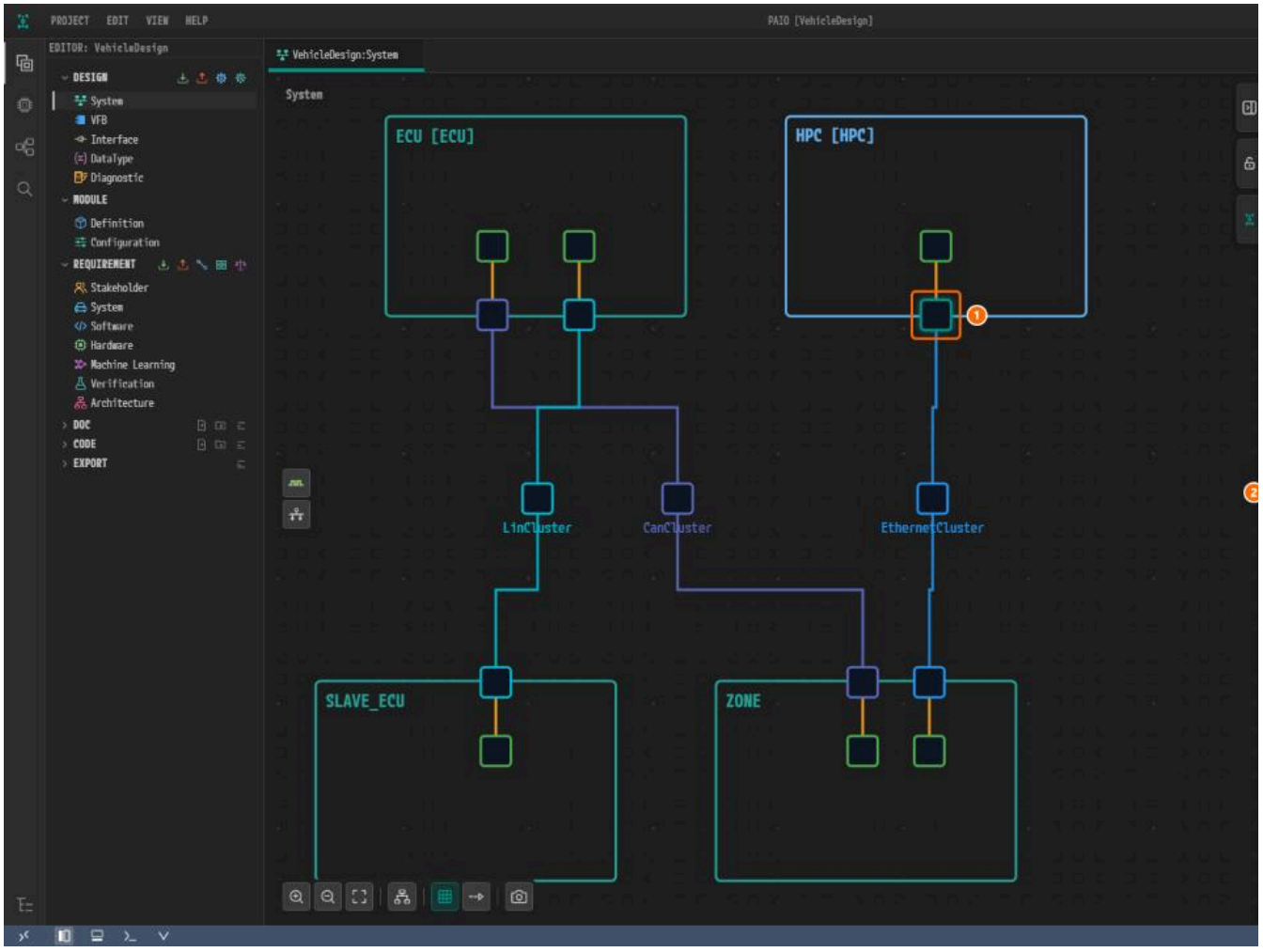


Figure 7-7. ① `EthernetConnector`, ② the three cards with their ports.

Viewing the Validation Result

1. Open the **VALIDATION** panel. If it is closed, click the **V** icon in the footer. It shows `No validation issues.`.

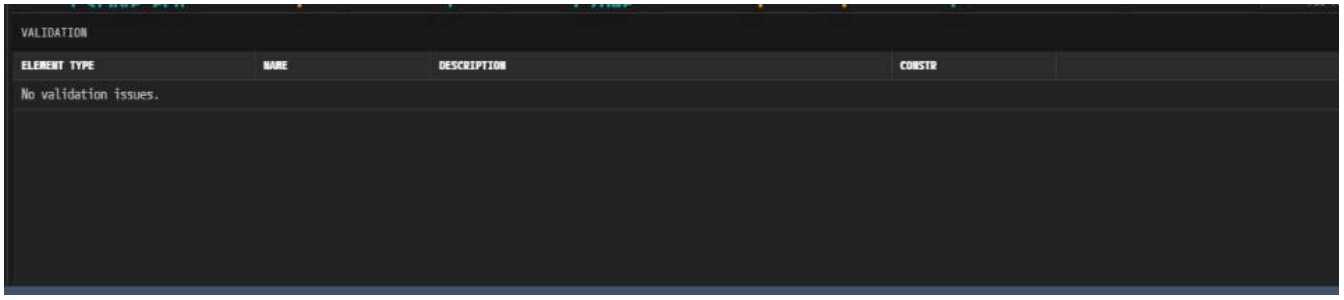


*Figure 7-8. The **VALIDATION** panel after saving.*

Why Is It Designed This Way?

- **One binding, one mapping.** In the Machine Extract of Validation and Code Generation each binding becomes a `SOMEIP-SERVICE-INSTANCE-TO-MACHI`
- **Ports from the list.** Picking from the list references the existing endpoint, so no endpoint is added.
- **No TCP port.** The event `SensorData` and the method `Calculation` use **UDP**, so only UDP ports are needed.

Checklist

- ☐ **SERVICE INSTANCES** of `EthernetConnector` shows `Provider_Sensor` `UDP:20003`, `Actuator_Sensor` `UDP:20001` and `Fusion_Sensor` `UDP:20002`.
- ☐ **APPLICATION ENDPOINTS** holds only the three endpoints from chapter 3.
- ☐ The tab `VehicleDesign:System` has no modified mark ●.

- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In Adaptive Applications and Ports you create the three Adaptive Applications and their ports.

Adaptive Applications and Ports

What You Will Do

In this chapter you create three Adaptive Applications inside the composition `HPC`, give `Sensor` a P port and `Fusion` and `Actuator` an R port with the s
Interface, and connect the P port to both R ports.

Before You Begin

- Start after finishing Binding Service Instances to the Connector.
- The composition `HPC` from The Composition HPC and the Machine is still empty.

Adaptive Application	NAME = AS PROTOTYPE	Port	Port NAME	PORT INTERFACE
Sensor	`Sensor`	one **PPort**	`Sensor`	`Service: Sensor`
Fusion	`Fusion`	one **RPort**	`Sensor`	`Service: Sensor`
Actuator	`Actuator`	one **RPort**	`Sensor`	`Service: Sensor`

The two connectors are named `Assembly_Sensor_Fusion` and `Assembly_Sensor_Actuator`.

Drilling into the Composition HPC

1. In the **EDITOR** view, click **VFB** under **DESIGN**. The tab `VehicleDesign:VFB` opens.
2. Double-click the node `HPC [HPC]`. The breadcrumb changes to `System > HPC` and an empty frame `HPC` appears.
3. Click the frame title `HPC`. Drag the bottom-right handle to widen the frame to about 900×520 so the three nodes fit.

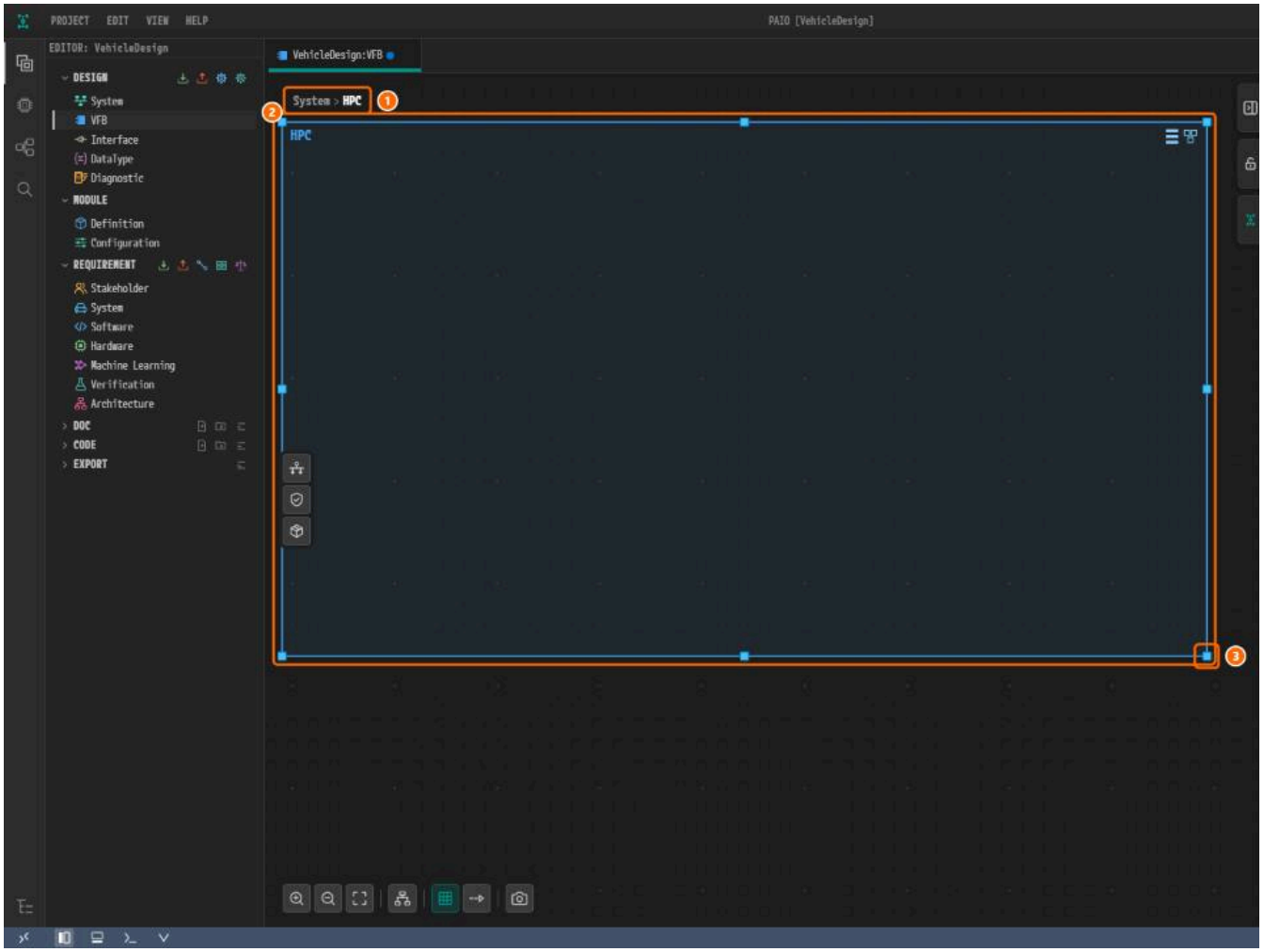


Figure 8-1. ① The breadcrumb `System > HPC`, ② the frame `HPC`, ③ the resize handle.

Creating the Adaptive Application Sensor

1. Right-click an empty spot inside the frame. Hover over **Add SWC** and click **Adaptive Application**.

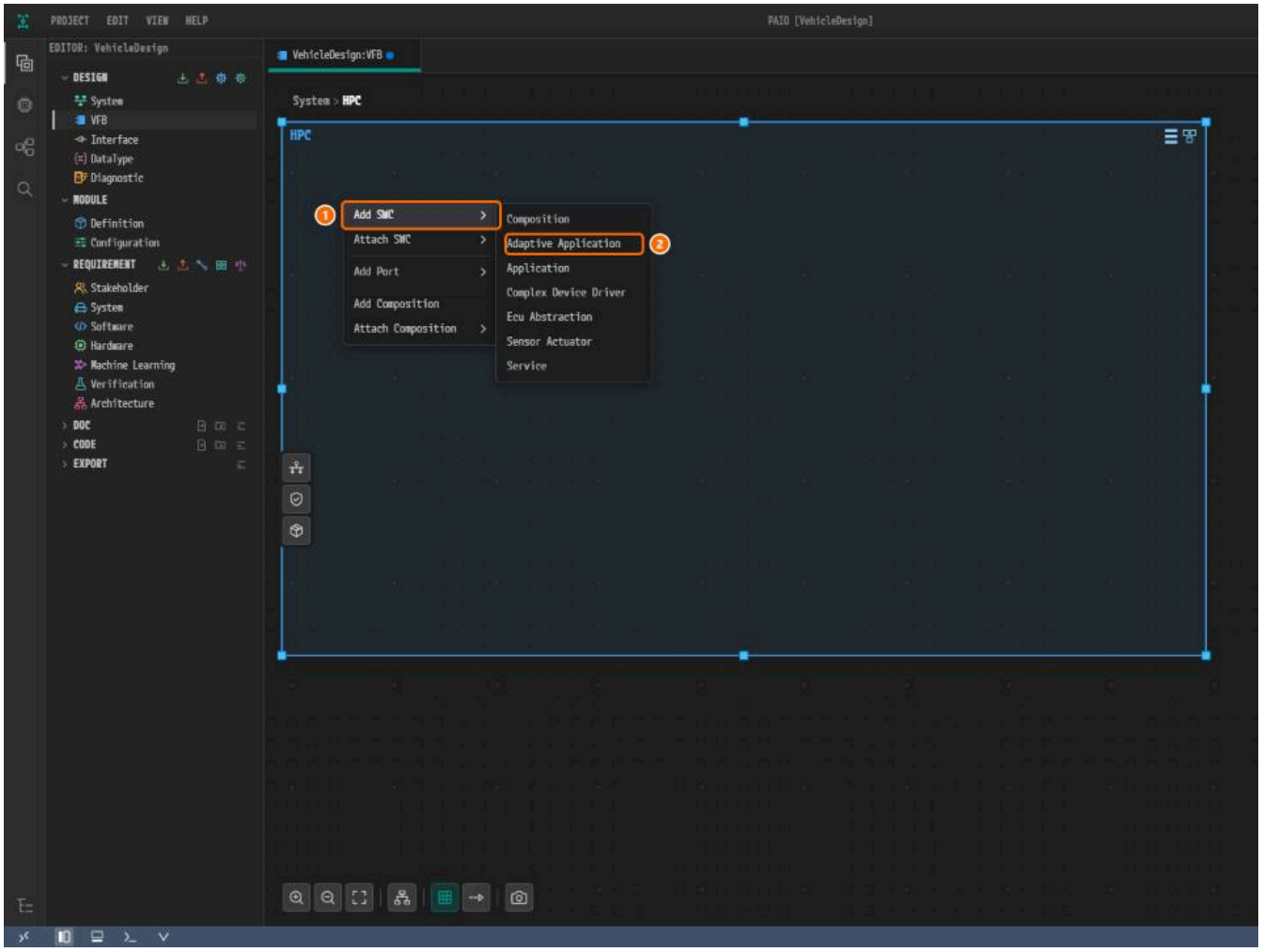


*Figure 8-2. ① **Add SWC**, ② **Adaptive Application**.*

2. Click the header of the new node `AdaptiveApp_0`. The **ADAPTIVE APPLICATION** panel opens.

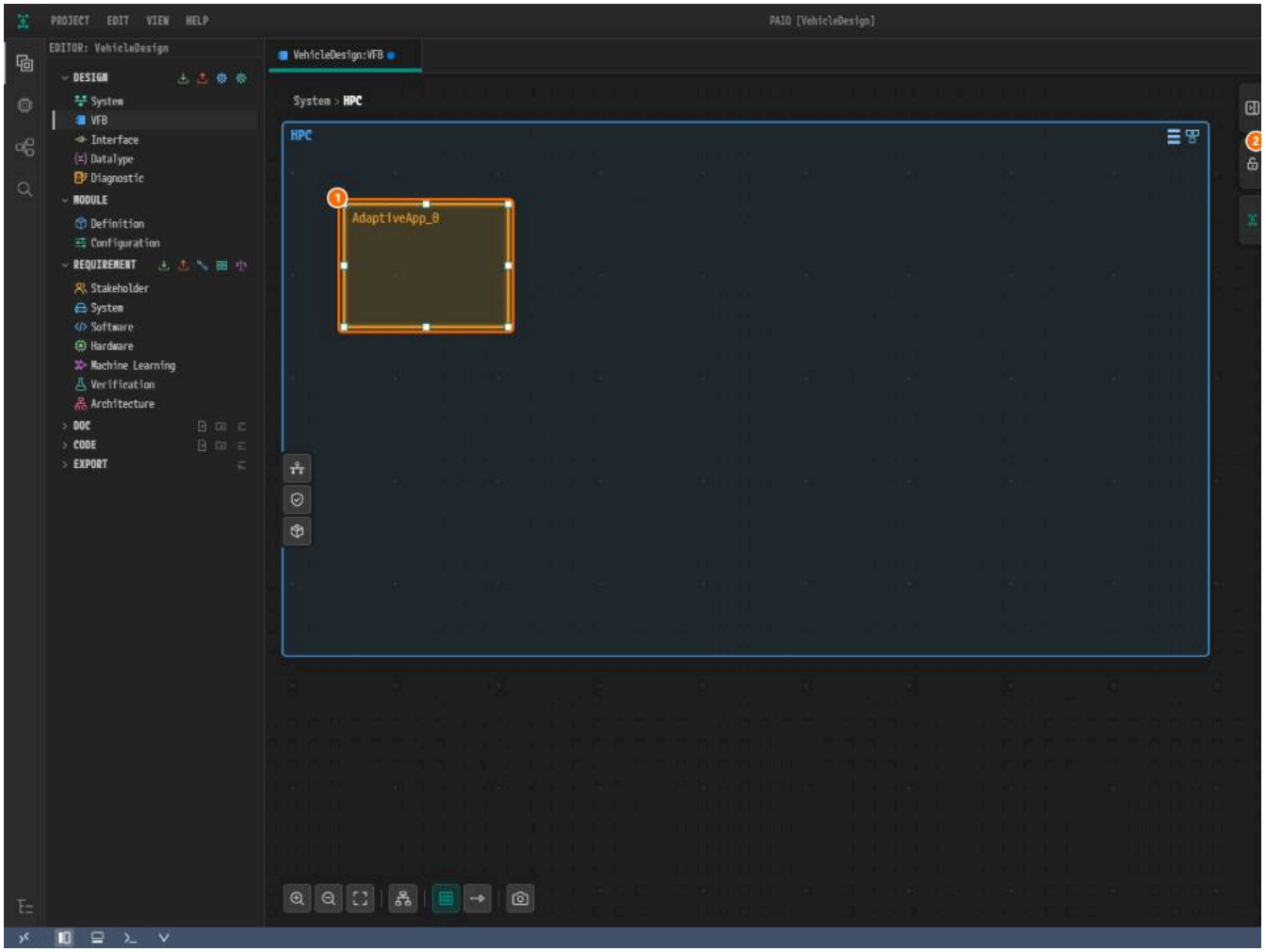


*Figure 8-3. ① The node `AdaptiveApp_0`, ② **NAME**, ③ **AS PROTOTYPE**.*

3. Enter `Sensor` in **NAME** and press `Tab`. Enter `Sensor` in **AS PROTOTYPE** too and press `Tab`. **AS PROTOTYPE** does not follow **NAME**.

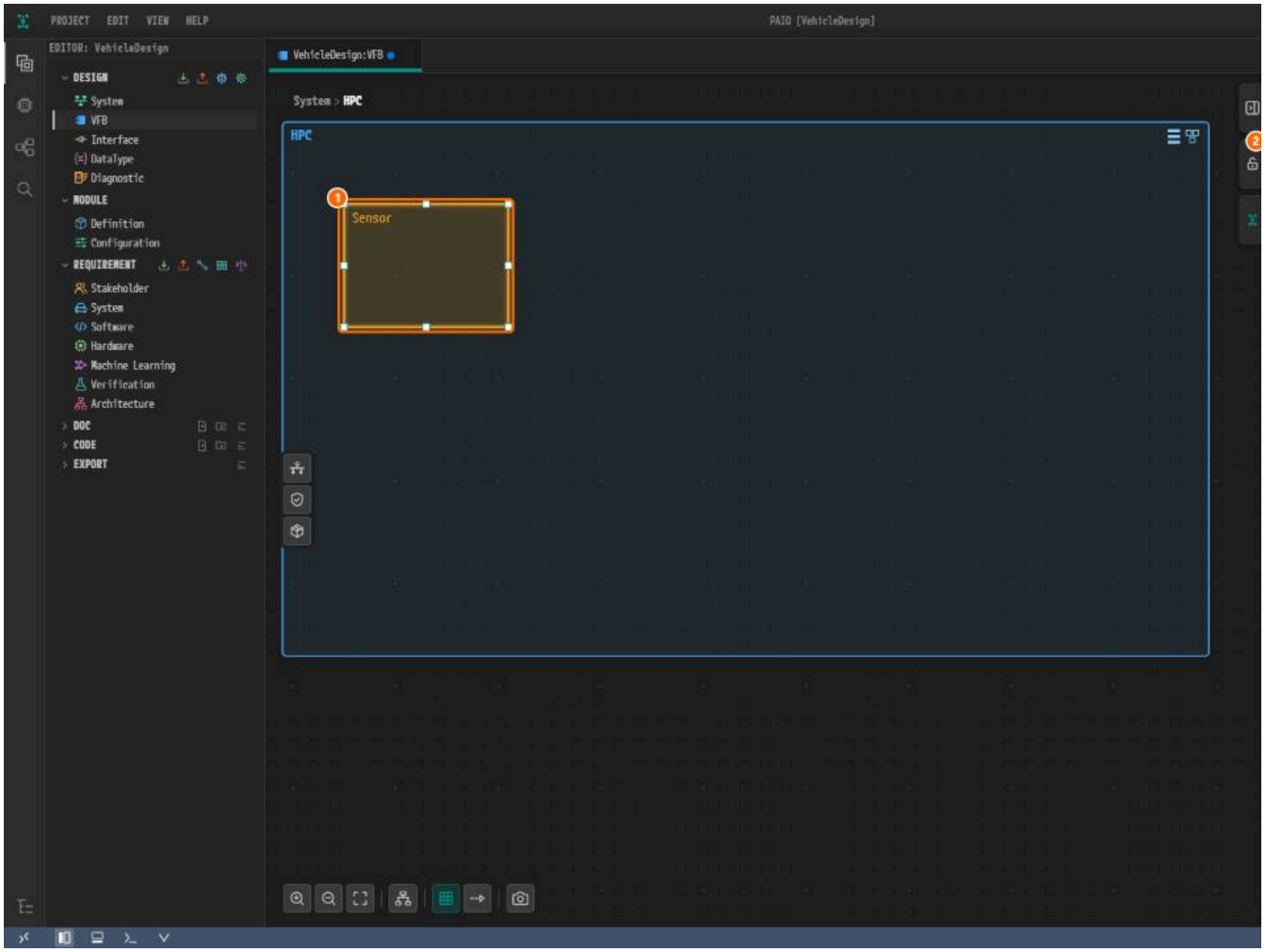


*Figure 8-4. ① The node `Sensor`, ② **NAME**, ③ **AS PROTOTYPE**.*

Creating Fusion and Actuator

Create `Fusion` and `Actuator` the same way from the table and put them to the right of `Sensor`.

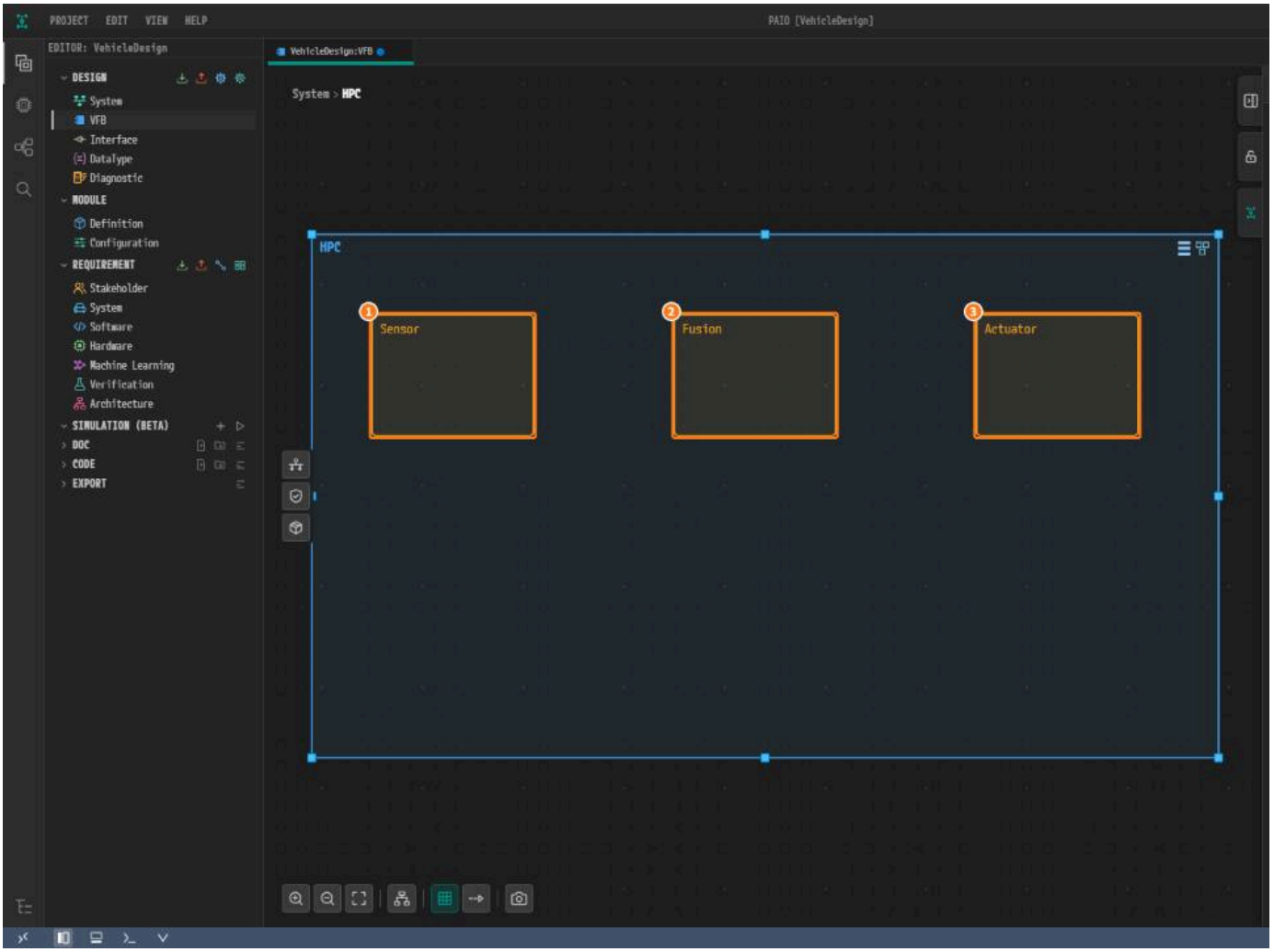


Figure 8-5. ① `Sensor`, ② `Fusion`, ③ `Actuator`.

Attaching the P Port to Sensor

1. Right-click near the right edge of `Sensor`. Hover over **Add Port** and click **PPort**.

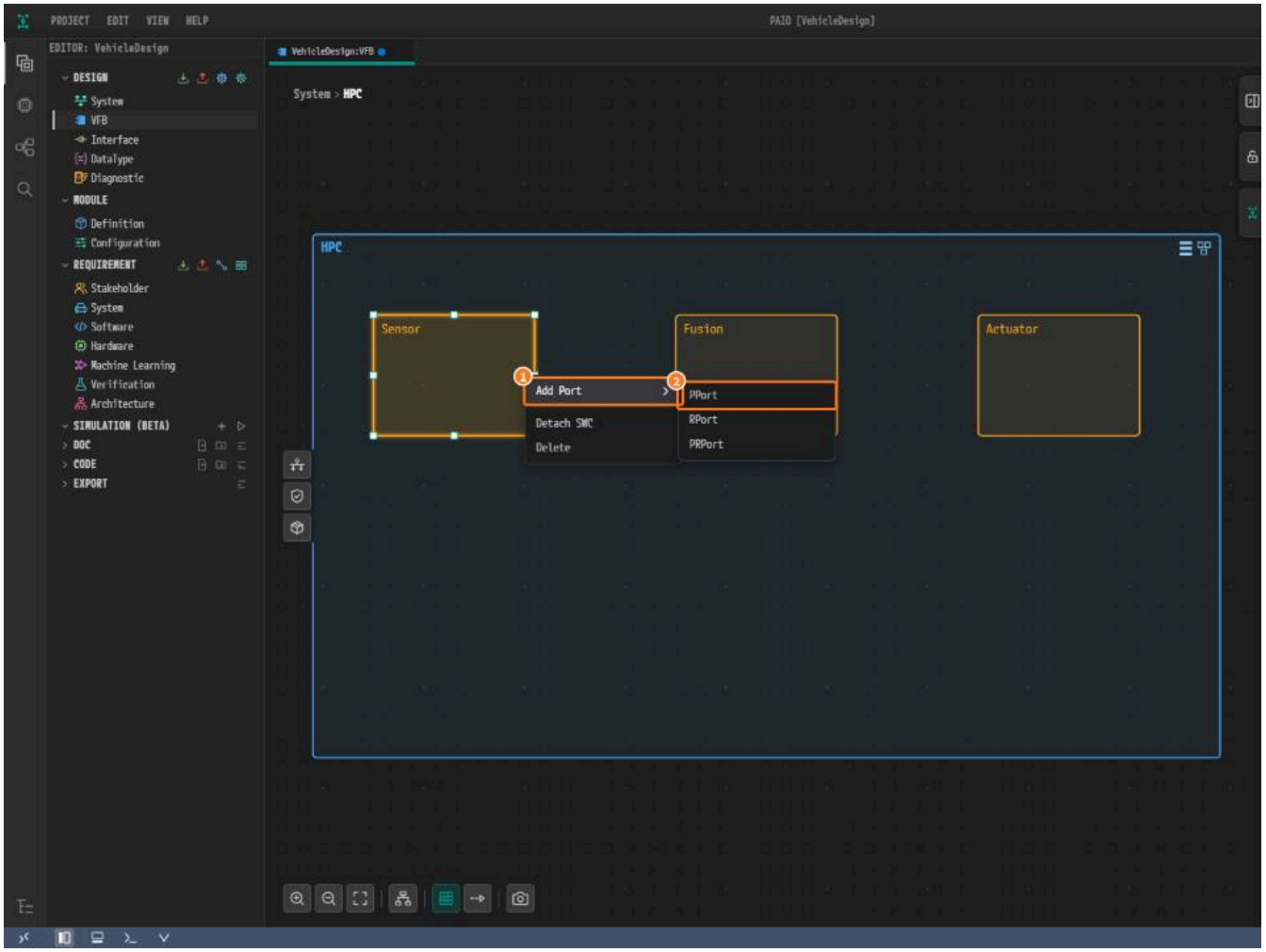


*Figure 8-6. ① **Add Port**, ② **PPort**.*

2. Click the new `P` port. Enter `Sensor` in **NAME** and press `Tab`.

3. Click **PORT INTERFACE** and click `Service: Sensor`, the only entry. Until you do, the **V** icon in the footer shows a red badge `1`.

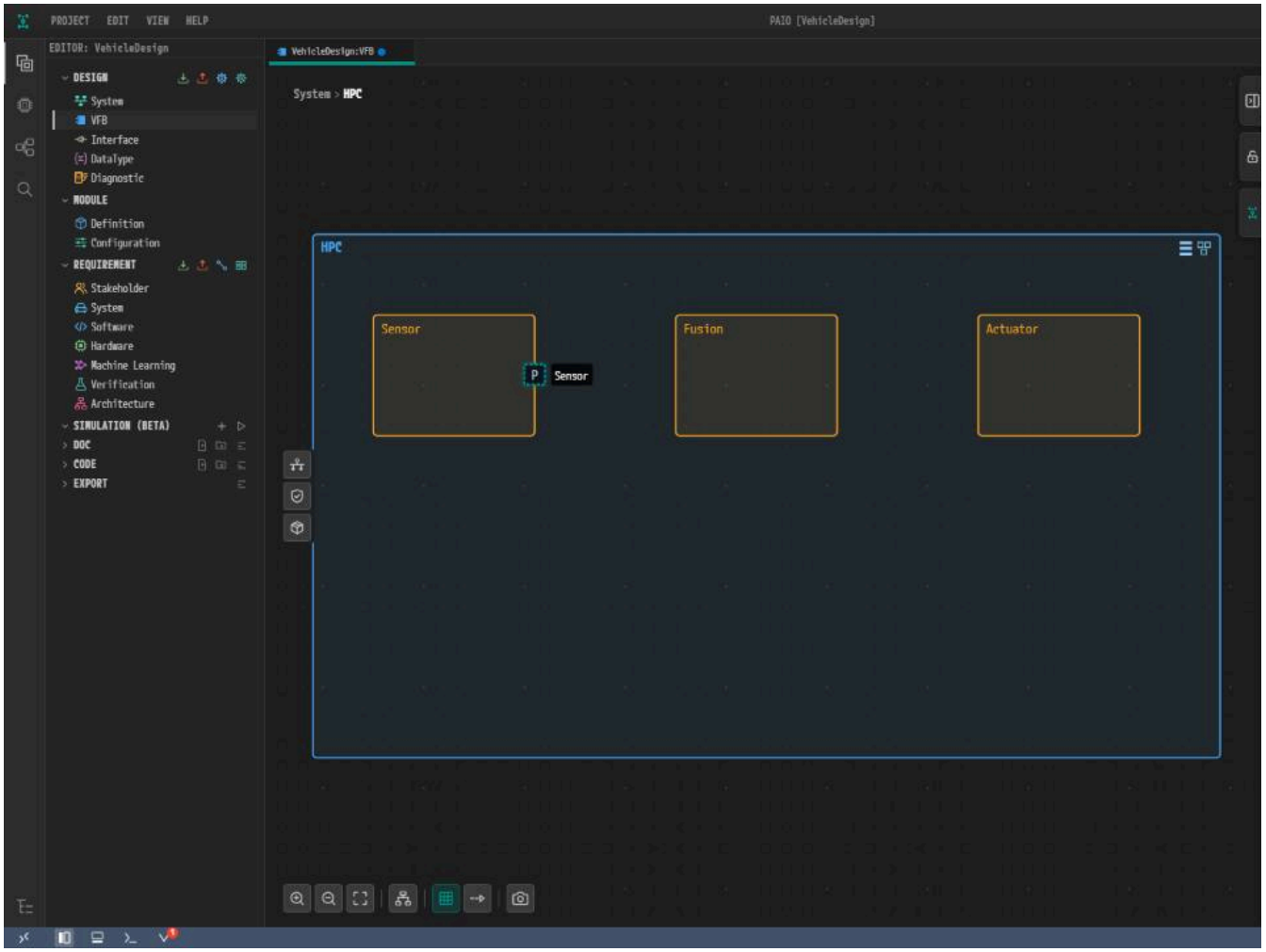


*Figure 8-7. ① **NAME**, ② **PORT INTERFACE**, ③ `Service: Sensor`.*

4. The section **SERVICE COMSPEC** appears. Leave it as it is.

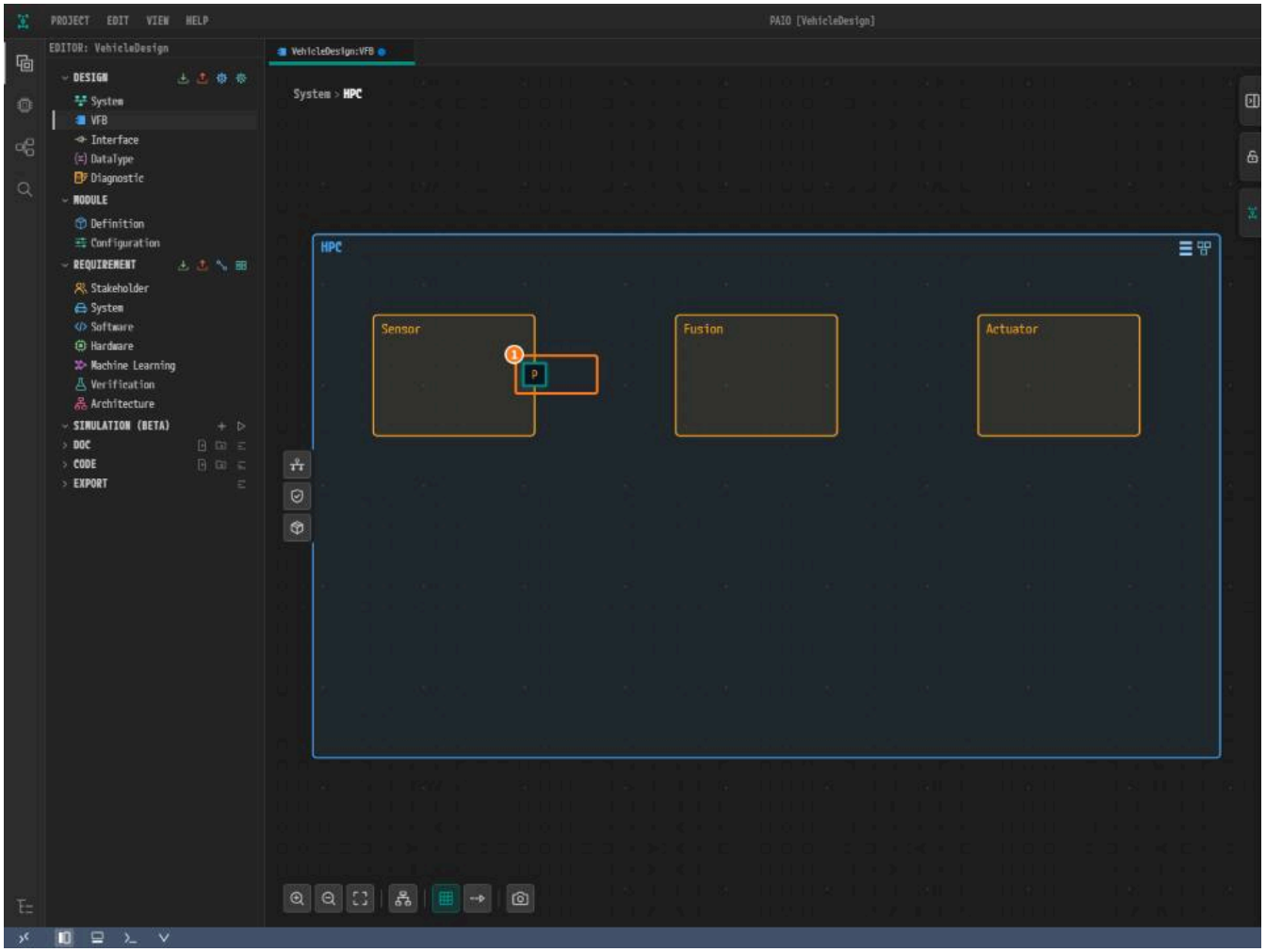


*Figure 8-8. ① The `P` port, ② **PORT INTERFACE** = `Service: Sensor`.*

Attaching the R Ports to Fusion and Actuator

1. Right-click near the left edge of `Fusion`. Click **Add Port** > **RPort**. Set **NAME** and **PORT INTERFACE** as for `Sensor`.
2. Do the same for `Actuator`.

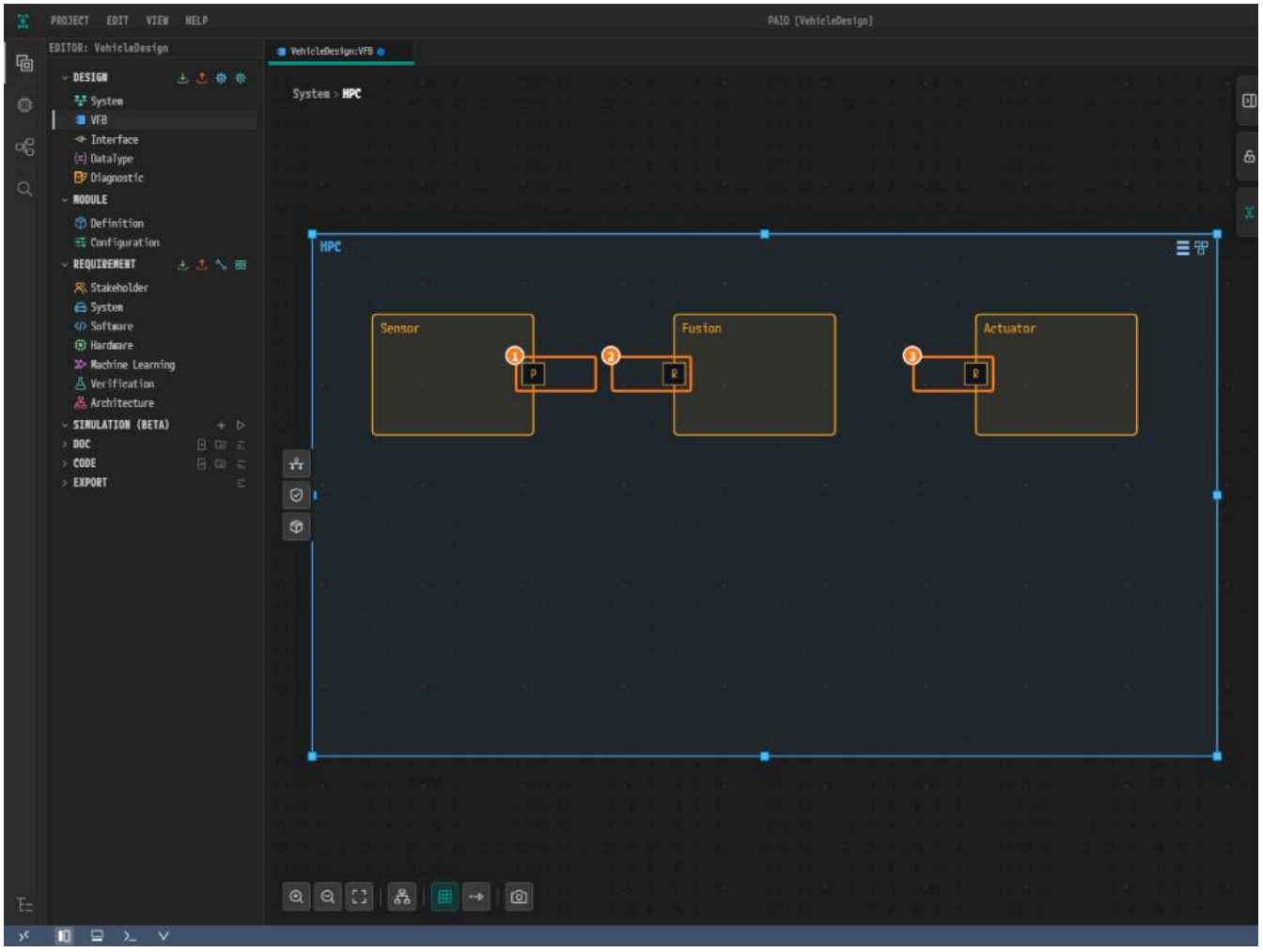


Figure 8-9. ① `P` on `Sensor`, ② `R` on `Fusion`, ③ `R` on `Actuator`.

Connecting the Ports

1. Right-click the `P` port of `Sensor`. Hover over **Connect to** and click `[A] Fusion / Sensor`.

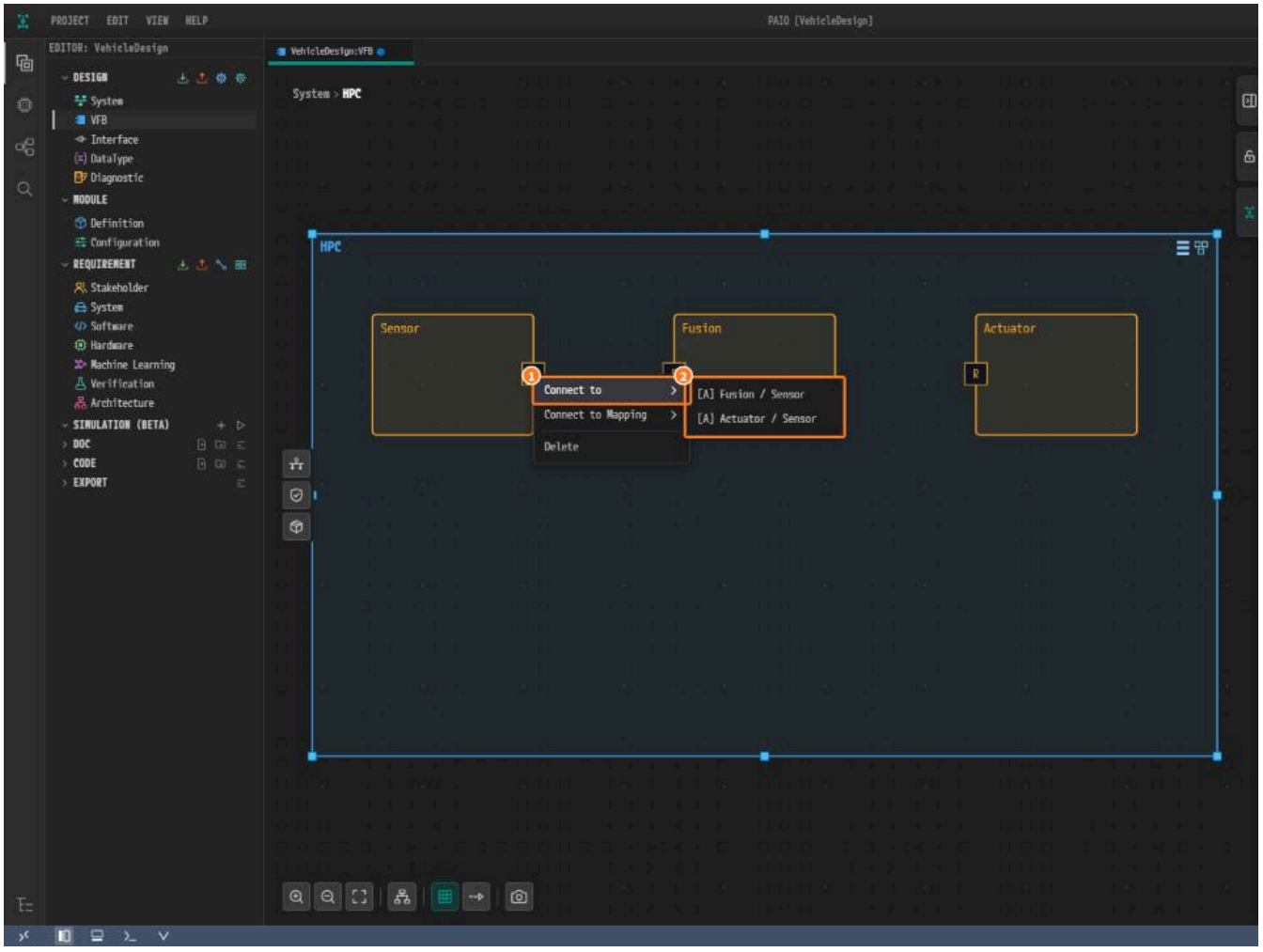


*Figure 8-10. ① **Connect to**, ② the candidates.*

2. Open the same menu again and click `[A] Actuator / Sensor`.

3. Click the line to `Fusion` near the R port of `Fusion`. In the **ASSEMBLY** panel, change **NAME** from `Assembly_Sensor_Sensor_Fusion_Sensor` to `Assembl`

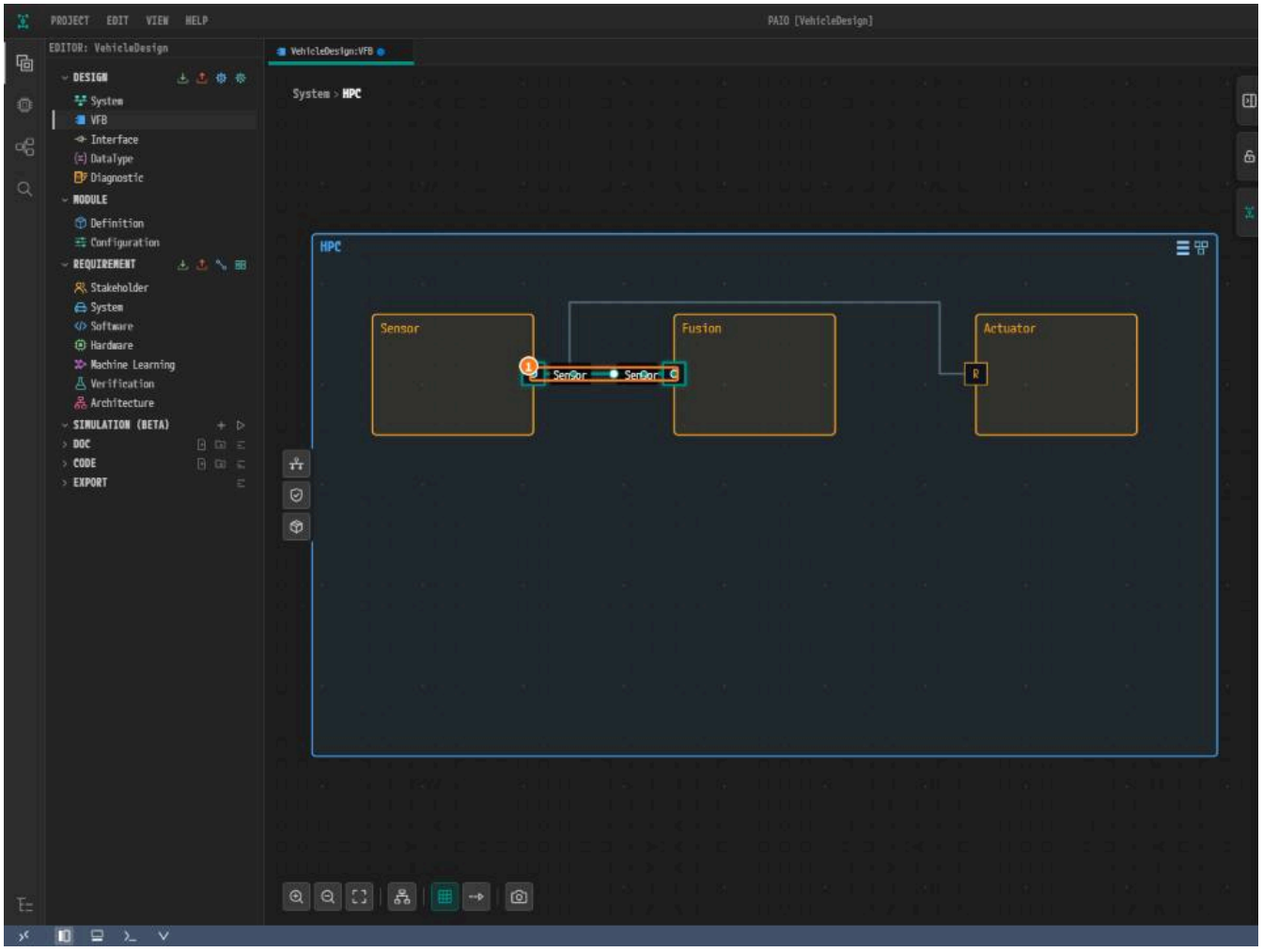


*Figure 8-11. ① The clicked line, ② **NAME** = `Assembly_Sensor_Fusion`.*

4. Rename the line to `Actuator` from `Assembly_Sensor_Sensor_Actuator_Sensor` to `Assembly_Sensor_Actuator` the same way.

Tidying with Auto Layout and Saving

1. Click **Auto Layout** in the toolbar under the canvas. Ports may move to the other side.
2. Click an empty spot on the canvas and press `Ctrl+S`.

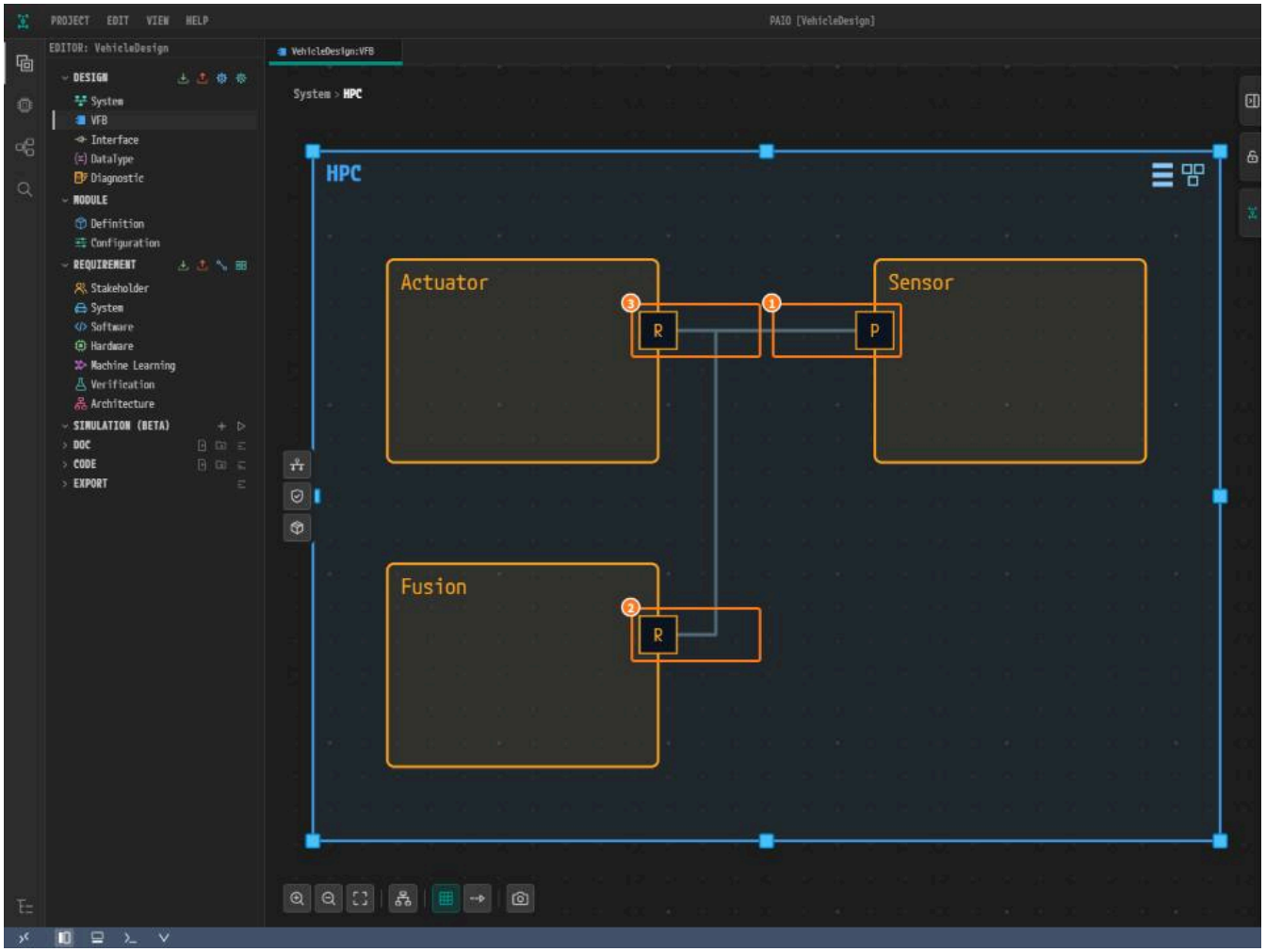


Figure 8-12. ① The P port of `Sensor`, ② the R port of `Fusion`, ③ the R port of `Actuator`. Auto Layout decides the positions, so your screen may dif

Viewing the Validation Result

1. Open the **VALIDATION** panel. If it is closed, click the **V** icon in the footer. It shows `No validation issues.`.

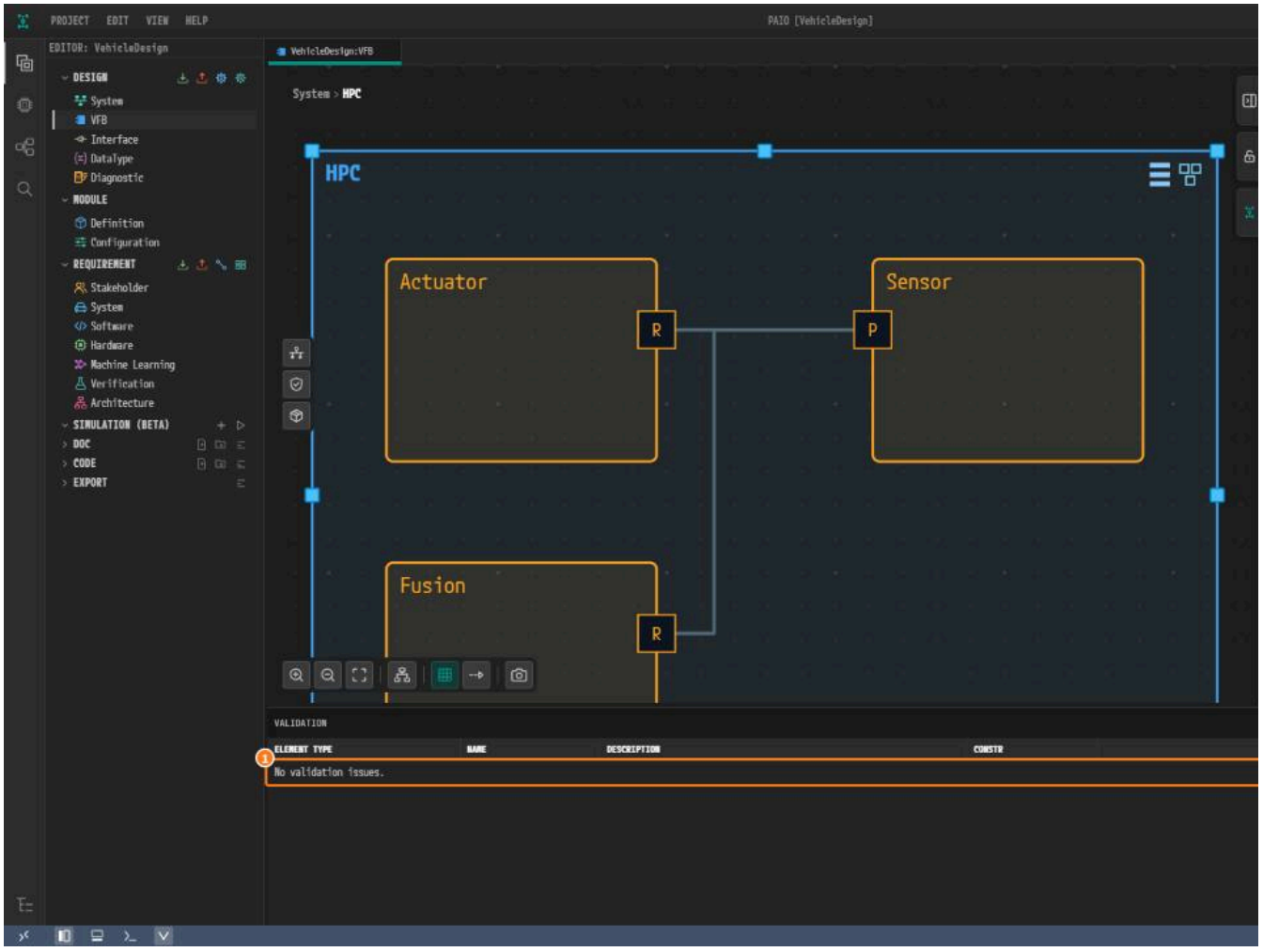


*Figure 8-13. The **VALIDATION** panel after saving.*

Why Is It Designed This Way?

- **Prototype name = type name.** Each prototype in `HPC` is named like its type, so **AS PROTOTYPE** gets the same value as **NAME**.
- **All three ports are named `Sensor`**, after their interface.
- **One provider, two consumers.** `Sensor` fans out to `Fusion` and `Actuator`; it is not a chain through `Fusion`.
- **Short connector names.** The default name also carries the port names. This design names a connector after the two prototypes only.
- **Only your own applications.** The platform applications such as `EM` and `CM` come from PARA in the next chapter; you do not draw them.
- **Layout is not design.** Positions are stored only in `HPC.vfb.layout` and are not exported.

Checklist

- [ ] The frame `HPC` holds `Sensor`, `Fusion` and `Actuator`, each with the same **NAME** and **AS PROTOTYPE**.
- [ ] `Sensor` has one `P` port, `Fusion` and `Actuator` one `R` port each, all named `Sensor` with **PORT INTERFACE** `Service: Sensor`.
- [ ] The two lines are named `Assembly_Sensor_Fusion` and `Assembly_Sensor_Actuator`.
- [ ] The tab `VehicleDesign:VFB` has no modified mark ●.
- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In PARA Setup you bring the platform applications of PARA, among them `EM` and `CM`, into the machine.

PARA Setup

What You Will Do

In this chapter you bring the platform of PARA, the PopcornSAR AUTOSAR Adaptive Platform, into the machine `HPC` with **PARA Platform Import**. It adds the yours, among them `EM` (execution management) and `CM` (communication management). Then you disable the **Network Management** module, which this

Before You Begin

- Start after finishing Adaptive Applications and Ports. The tab `VehicleDesign:VFB` is saved.
- PARA comes with PAIO. You do not need a file.

Where	Item	Value
Conflict Resolution window	**ECU ID** (`ECU1` / `HPC`)	**Keep Project**
Conflict Resolution window	**Max Action List Duration** (`1000` / `5`)	**Keep Project**
Machine Configuration [HPC], MODULE	**Network Management**	**Disable**

Importing the PARA Platform

1. In the tab `VehicleDesign:VFB`, go inside the composition `HPC` (`System > HPC`). Click the machine icon at the top right of the frame. **Machine Confi**

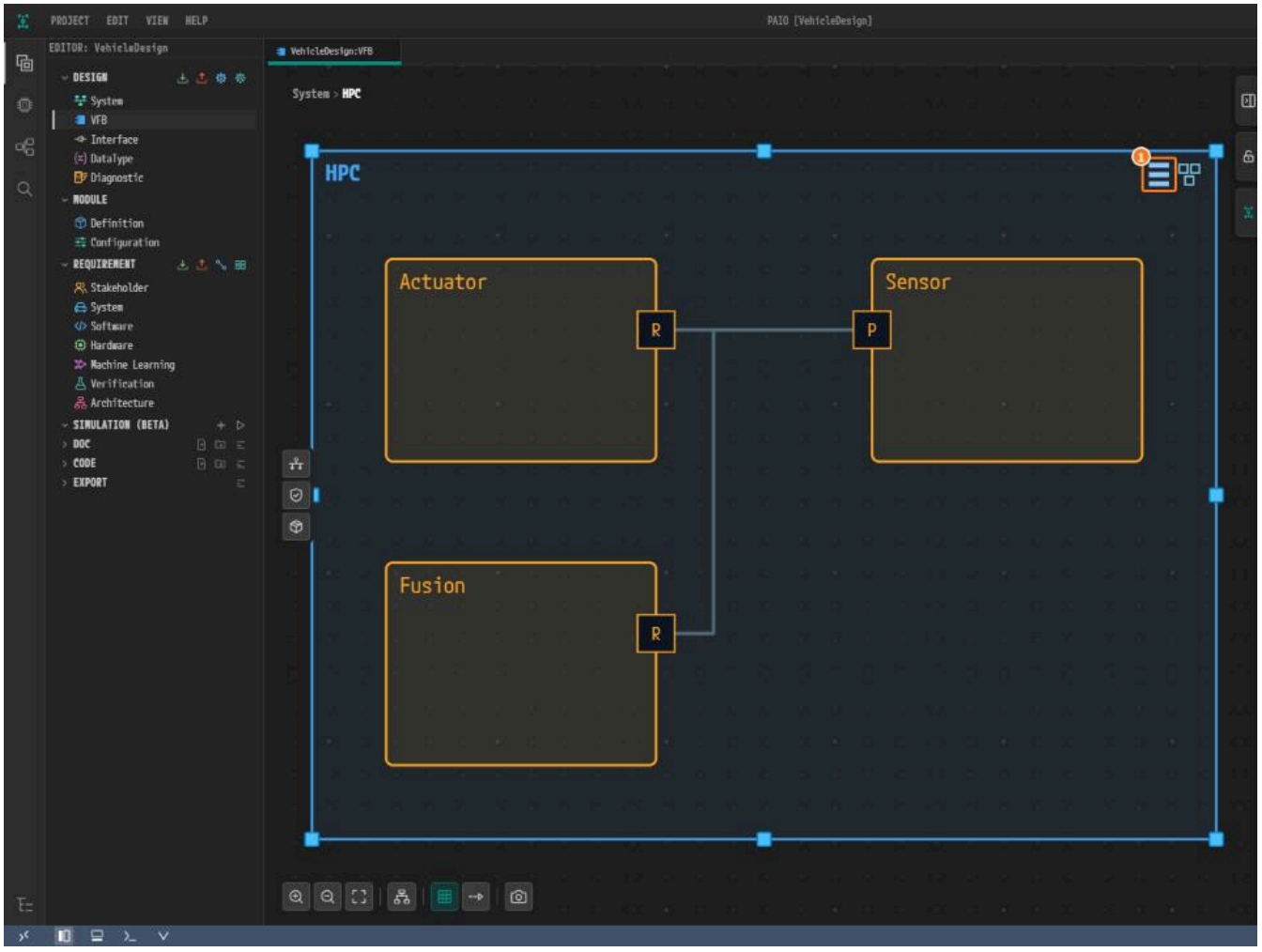


Figure 9-1. ① The machine icon.

2. Click **PARA Platform Import** at the top right of the window.

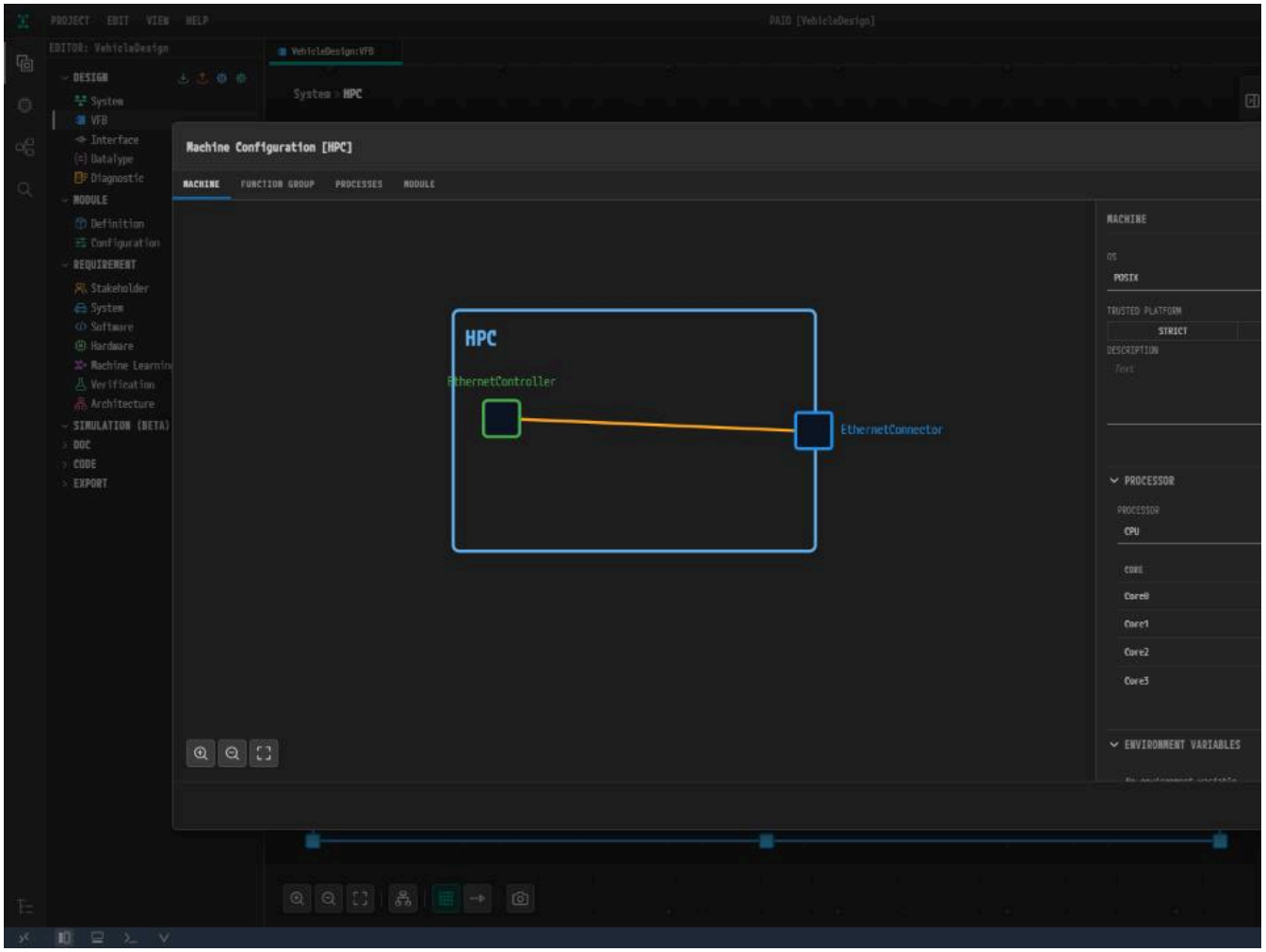


*Figure 9-2. ① **PARA Platform Import**.*

3. Click **Import** in the confirmation.

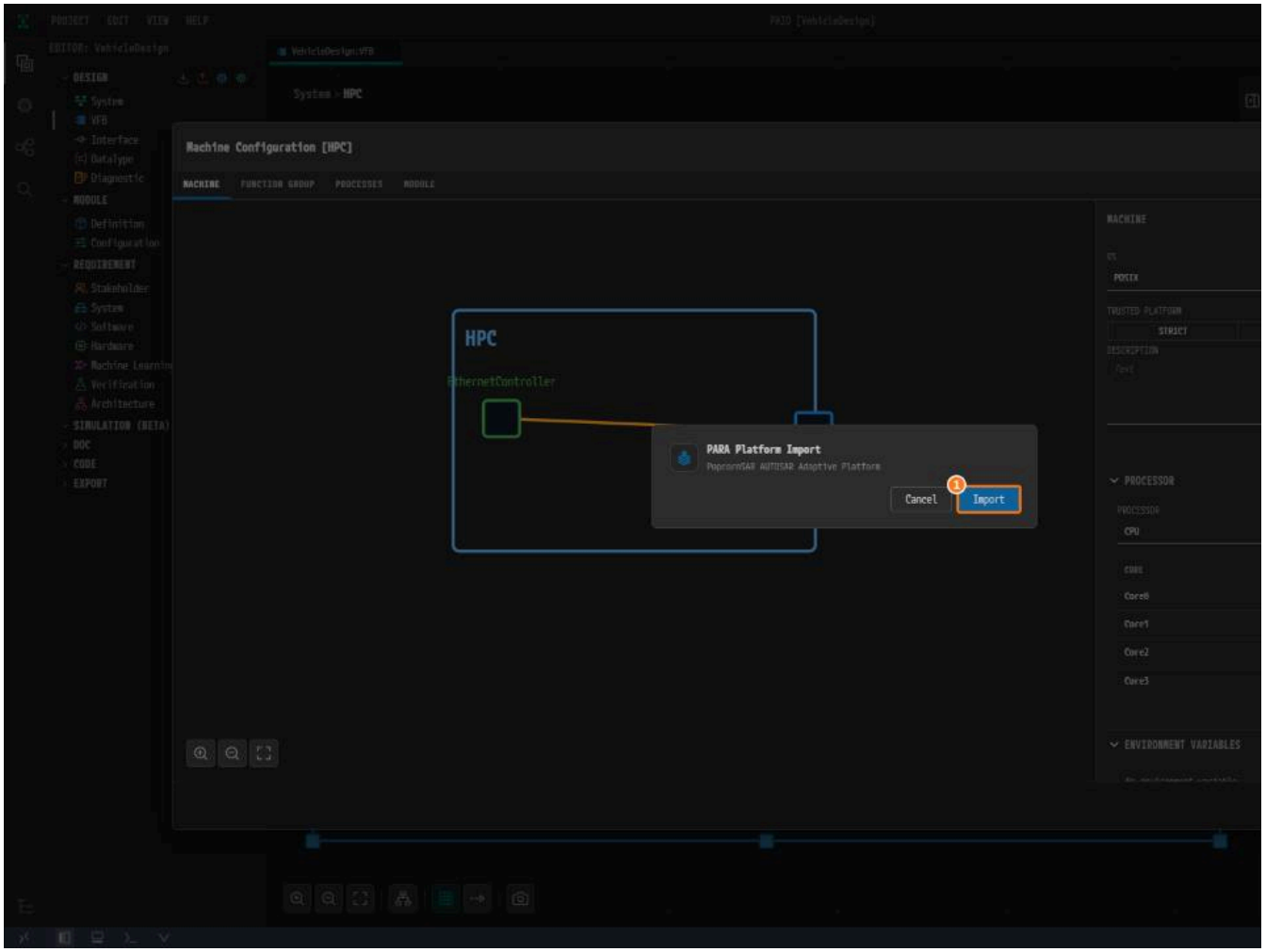


*Figure 9-3. ① **Import**.*

4. The **Conflict Resolution** window opens: the machine `HPC` differs from PARA's in two values you entered in Configuring the Machine. Click **Keep Projec Duration**. When the button at the bottom right reads `Resolve 1/1`, click it.

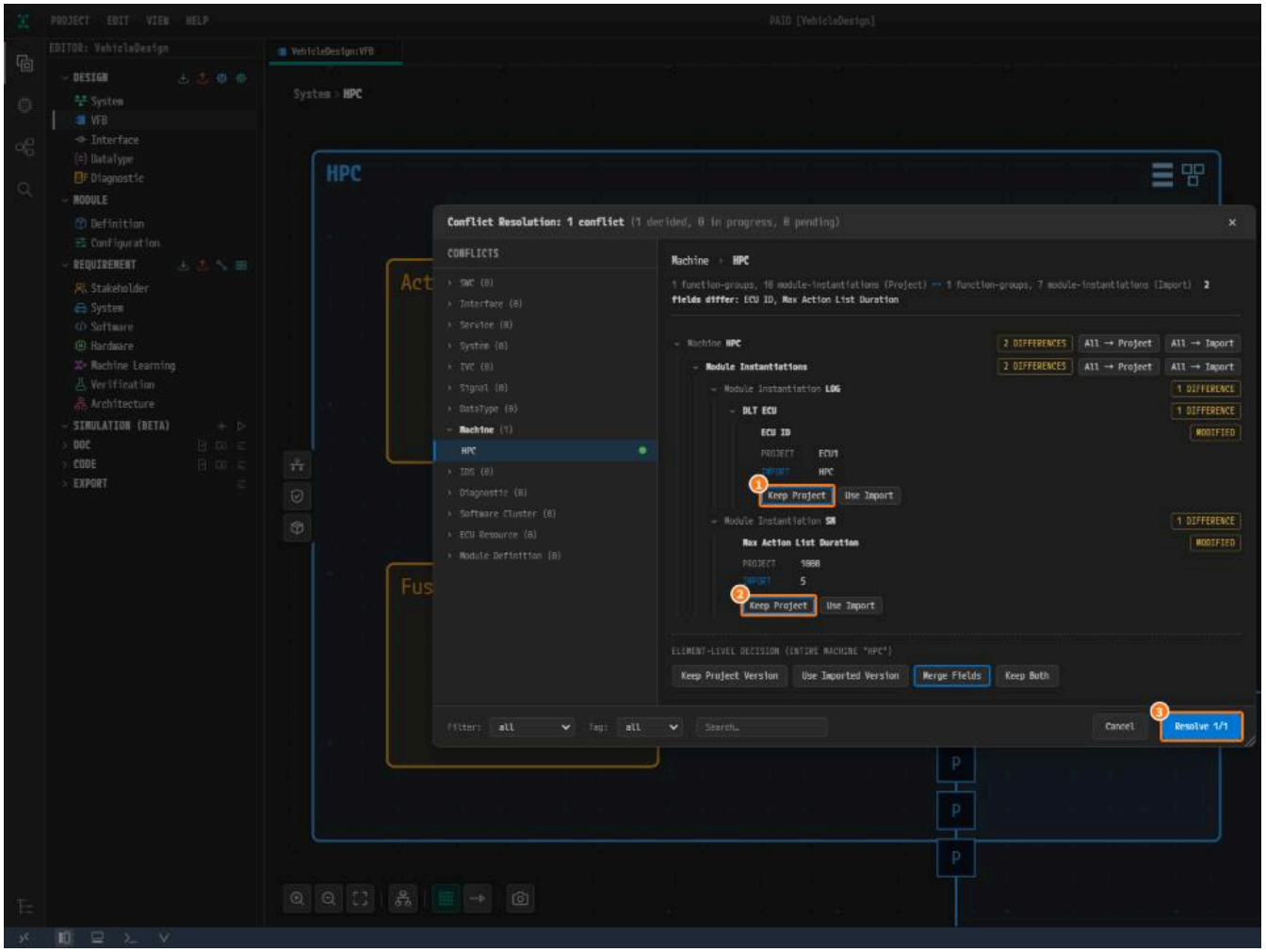


*Figure 9-4. ① **Keep Project** of **ECU ID**, ② **Keep Project** of **Max Action List Duration**, ③ `Resolve 1/1`.*

5. The **Machine Configuration [HPC]** window comes back and `Import complete` appears at the bottom right. Click the tab **PROCESSES.** The list holds th `TBM`, `DM`, `IDSM`, `NM`, `UCM`, `VUCM` and `CSM`.

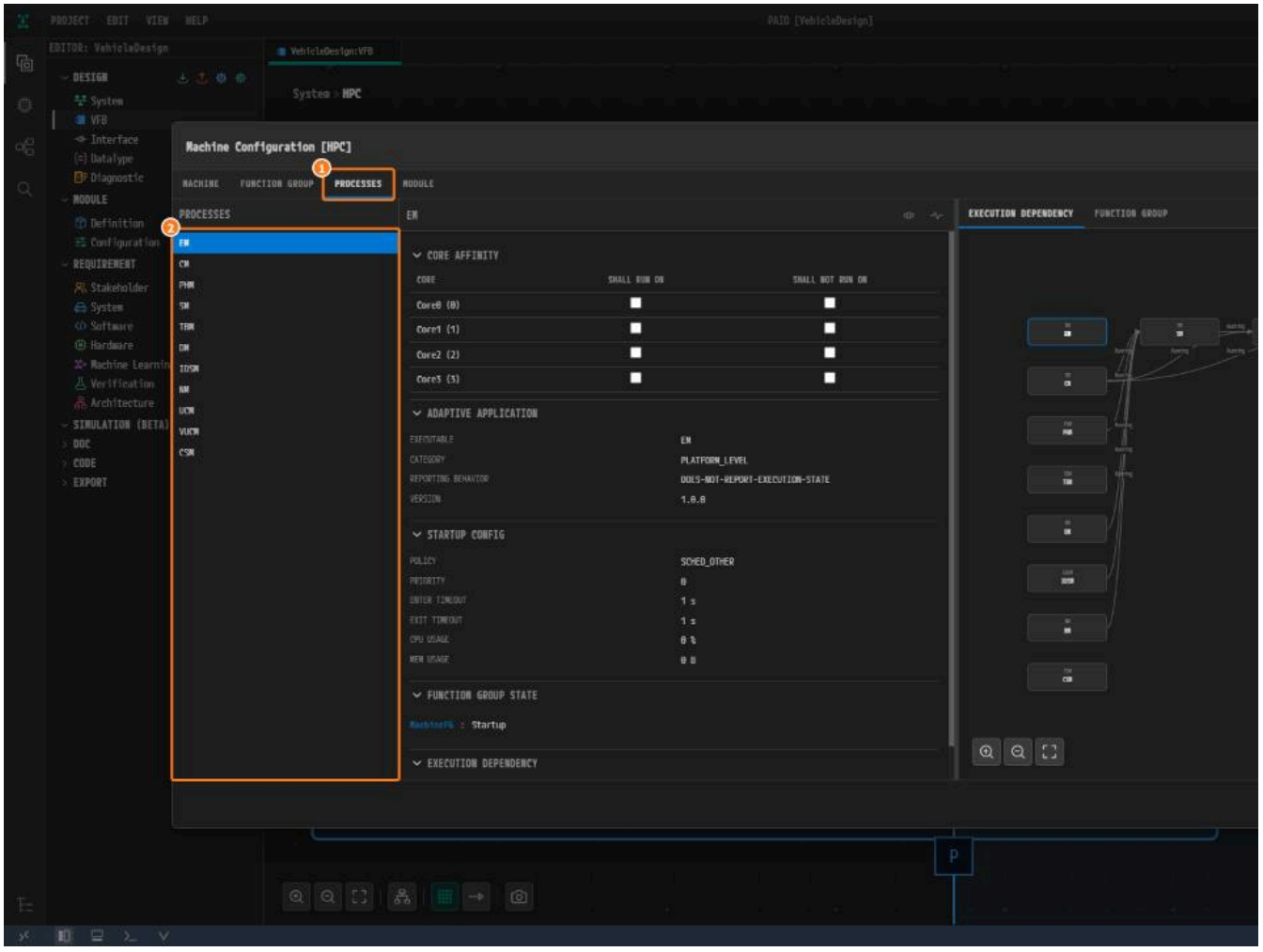


*Figure 9-5. ① **PROCESSES**, ② the eleven platform processes.*

Disabling Network Management

1. Click the tab **MODULE**. Right-click **Network Management** and click **Disable**.

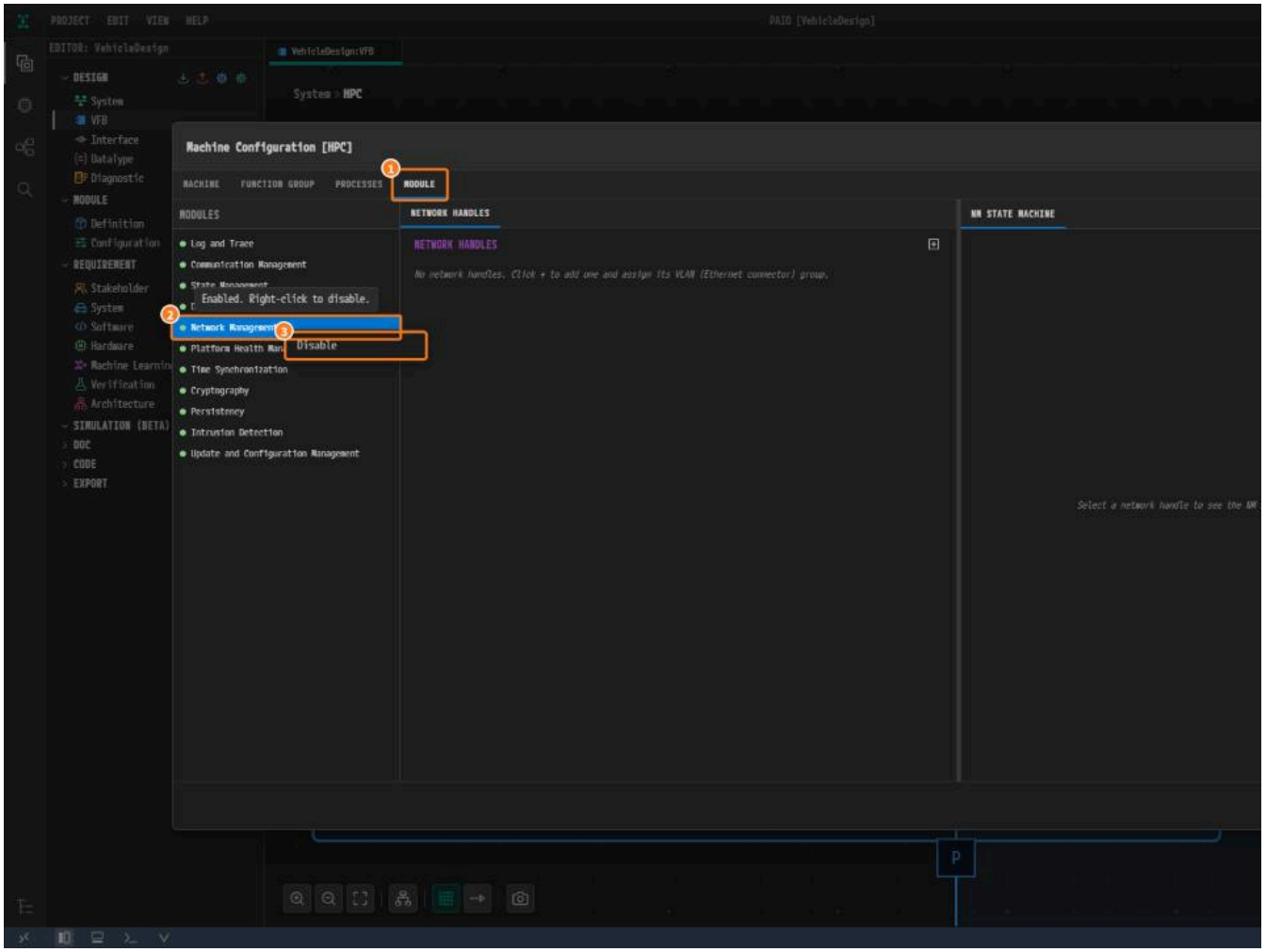


*Figure 9-6. ① **MODULE**, ② **Network Management**, ③ **Disable**.*

2. The dot in front of **Network Management** turns hollow and its panel is dimmed. Press `Ctrl+S` in the window, then click **Close**.

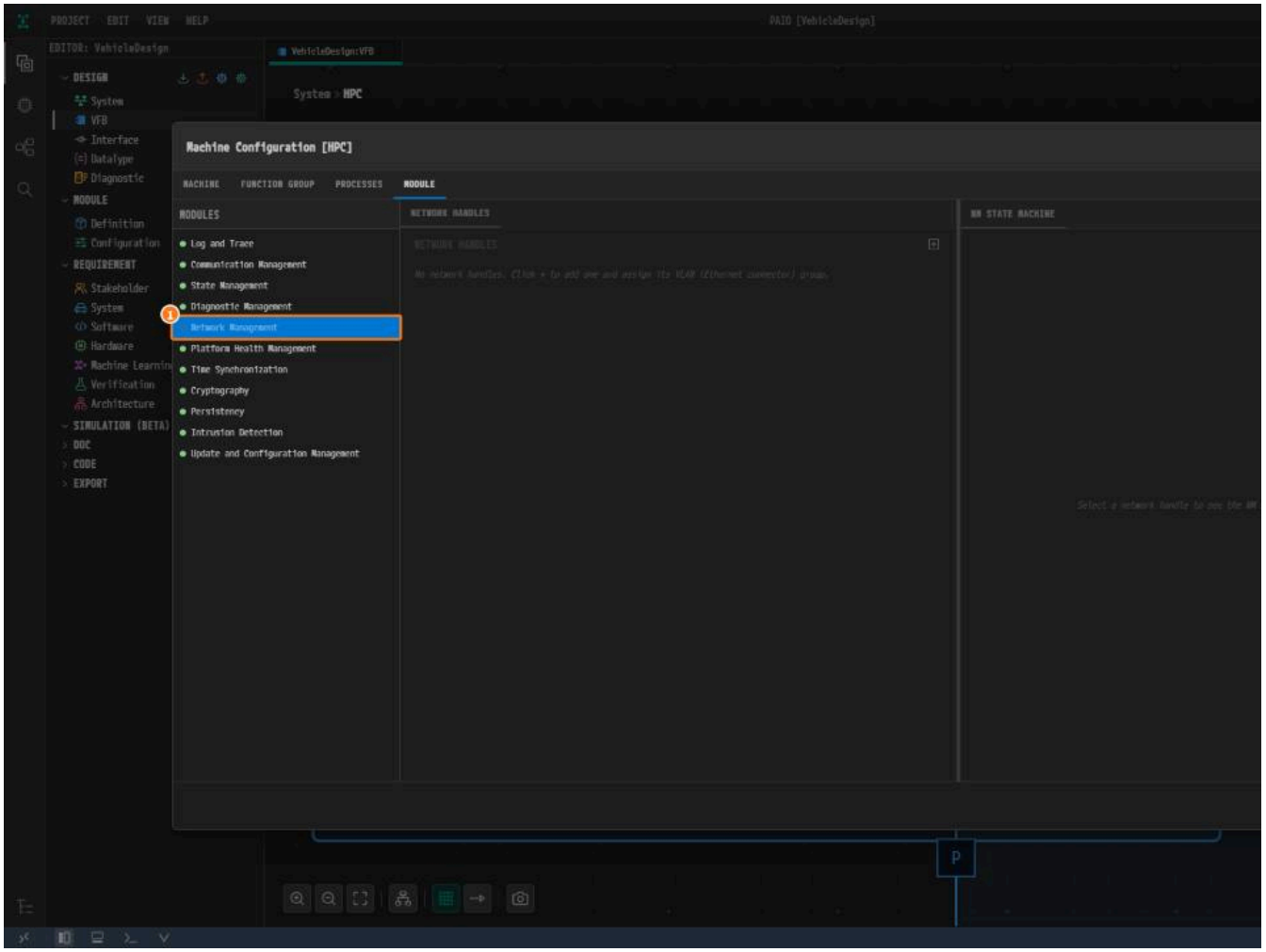


*Figure 9-7. ① **Network Management**, disabled.*

Tidying with Auto Layout and Saving

1. The import also put the composition `PARA_HPC` on the canvas, partly outside the frame. Click **Auto Layout** in the toolbar under the canvas.

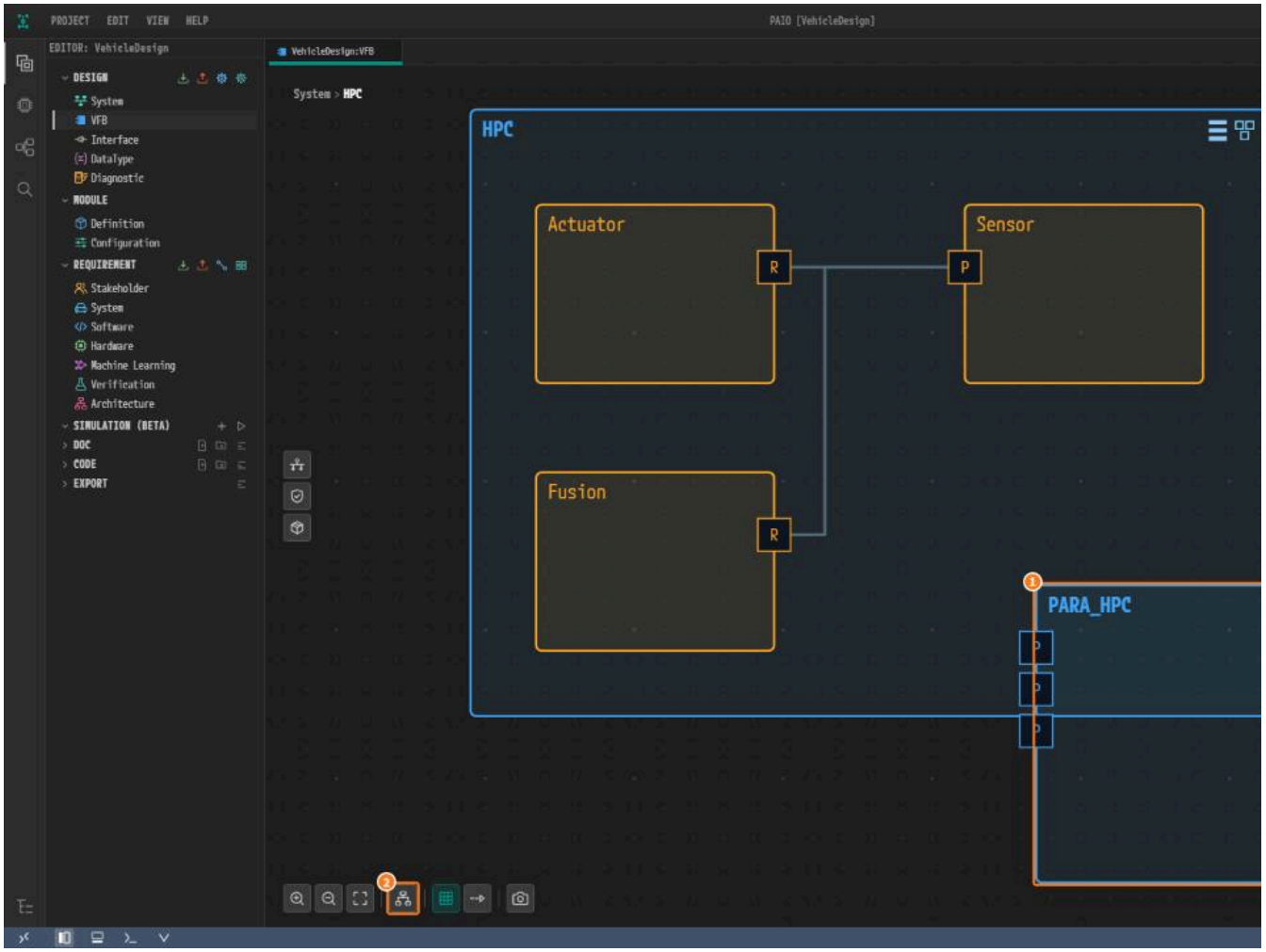


*Figure 9-8. ① `PARA_HPC`, ② **Auto Layout**.*

2. Click an empty spot on the canvas and press `Ctrl+S`.

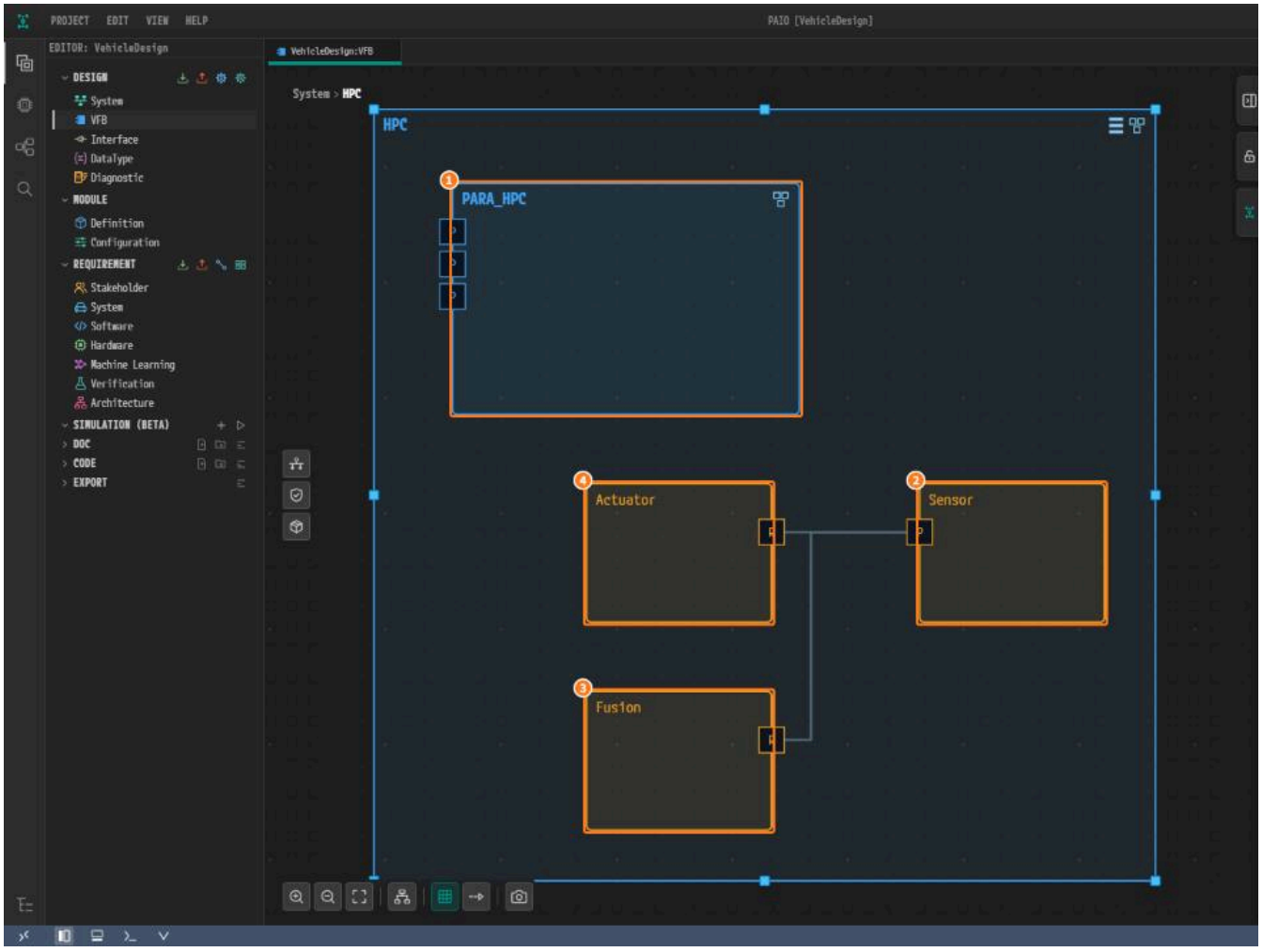


Figure 9-9. ① `PARA_HPC`, ② `Sensor`, ③ `Fusion`, ④ `Actuator`. Auto Layout decides the positions, so your screen may differ.

> **Note.** `PARA_HPC` holds the eleven platform applications; leave its contents as they are. The import also adds the interfaces and data types of the platf now list many entries you did not create.

Viewing the Validation Result

1. Open the **VALIDATION** panel. If it is closed, click the **V** icon in the footer. It shows `No validation issues.`.

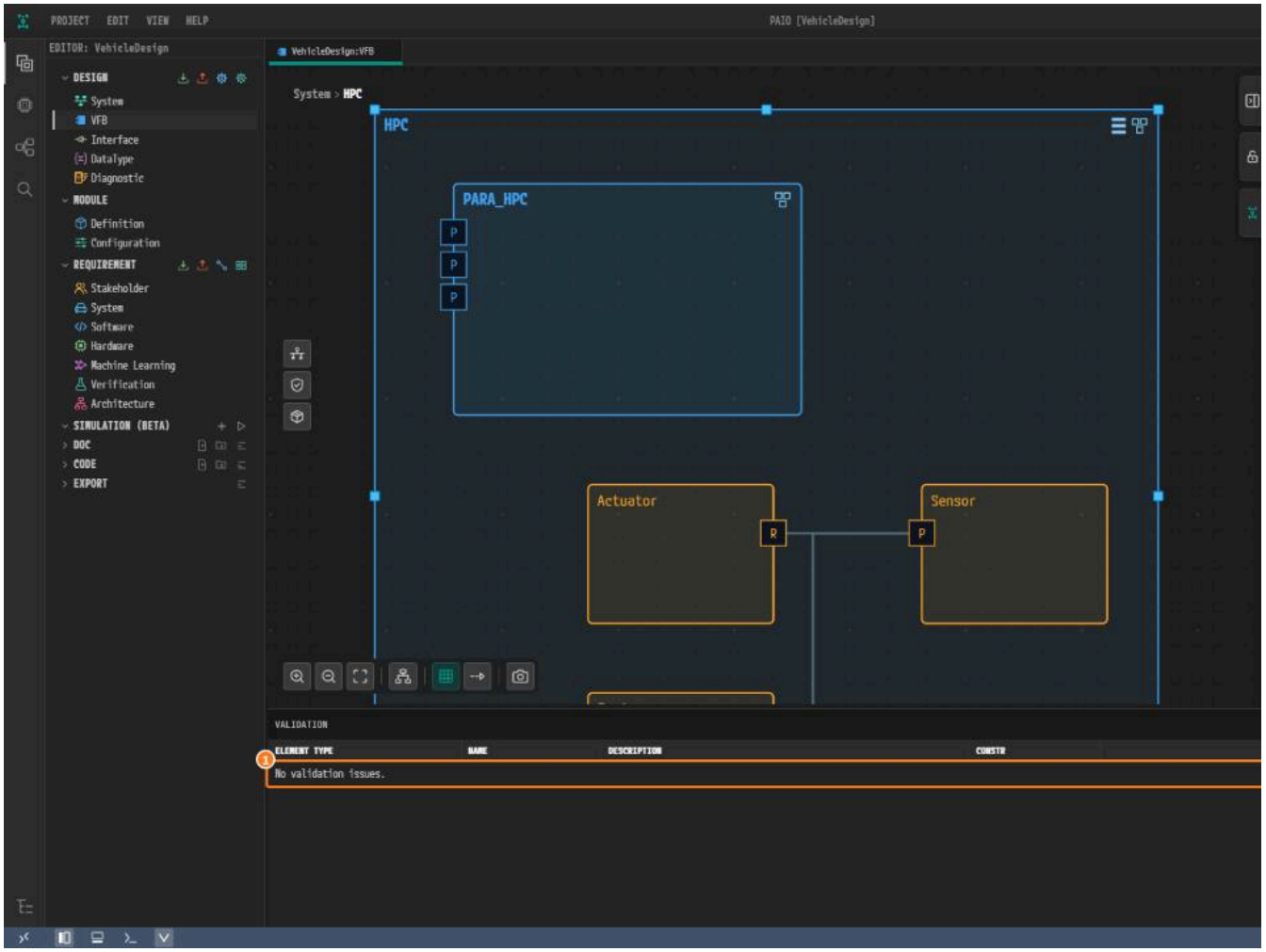


*Figure 9-10. The **VALIDATION** panel after saving.*

Why Is It Designed This Way?

- **Here, not earlier.** The interface list of a port in chapter 8 offers one entry, because the interfaces of PARA are not in the project yet.
- **Here, not later.** In the next chapter `Sensor`, `Fusion` and `Actuator` wait for the process `CM`, which comes from this import.
- **The platform is imported, not drawn.** A hand-made `EM` or `CM` would stand beside PARA's own and validation would report the duplicates.
- **Keep Project for both values.** `ECU1` and `1000` are this machine's own values from chapter 2.
- **Network Management is disabled.** PARA brings it enabled, and code generation then requires a network handle, which this design does not have. A d exported nor validated.

Checklist

- [ ] The **PROCESSES** tab of **Machine Configuration [HPC]** lists the eleven platform processes.
- [ ] **Network Management** in the **MODULE** tab is disabled, and all other modules are enabled.
- [ ] **ECU ID** under **MODULE > Log and Trace** is still `ECU1`.
- [ ] The frame `HPC` holds `PARA_HPC`, `Sensor`, `Fusion` and `Actuator`, and the tab has no modified mark ●.
- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In Executables and Processes you create a Process for each of your three Adaptive Applications.

Executables and Processes

What You Will Do

In this chapter you add a Process to each of the three Adaptive Applications `Sensor`, `Fusion` and `Actuator`. Each Process gets startup timeouts, a stat process. Then you check the processes on the machine.

Before You Begin

- Start after finishing PARA Setup. The process `CM` comes from there.
- The machine `HPC` has the function group `MachineFG` from Machine Configuration.

Adaptive Application	Process NAME	ENTER TIMEOUT / EXIT TIMEOUT	FUNCTION GROUP STATE	EXECUTION DEPENDENCY
`Sensor`	`Sensor`	`5` / `5`	`MachineFG`, `Startup`	`CM`, `Running`
`Fusion`	`Fusion`	`5` / `5`	`MachineFG`, `Startup`	`CM`, `Running`
`Actuator`	`Actuator`	`5` / `5`	`MachineFG`, `Startup`	`CM`, `Running`

Leave the **EXECUTABLE** section, the timeout unit `s`, **SCHEDULING POLICY** `SCHED_OTHER` and **SCHEDULING PRIORITY** `0` as they are for all three.

Creating the Process of Sensor

1. In the **EDITOR** view, click **VFB** under **DESIGN** and double-click the node `HPC [HPC]`. Skip this if the breadcrumb already reads `System > HPC`.
2. Double-click the node `Sensor`.

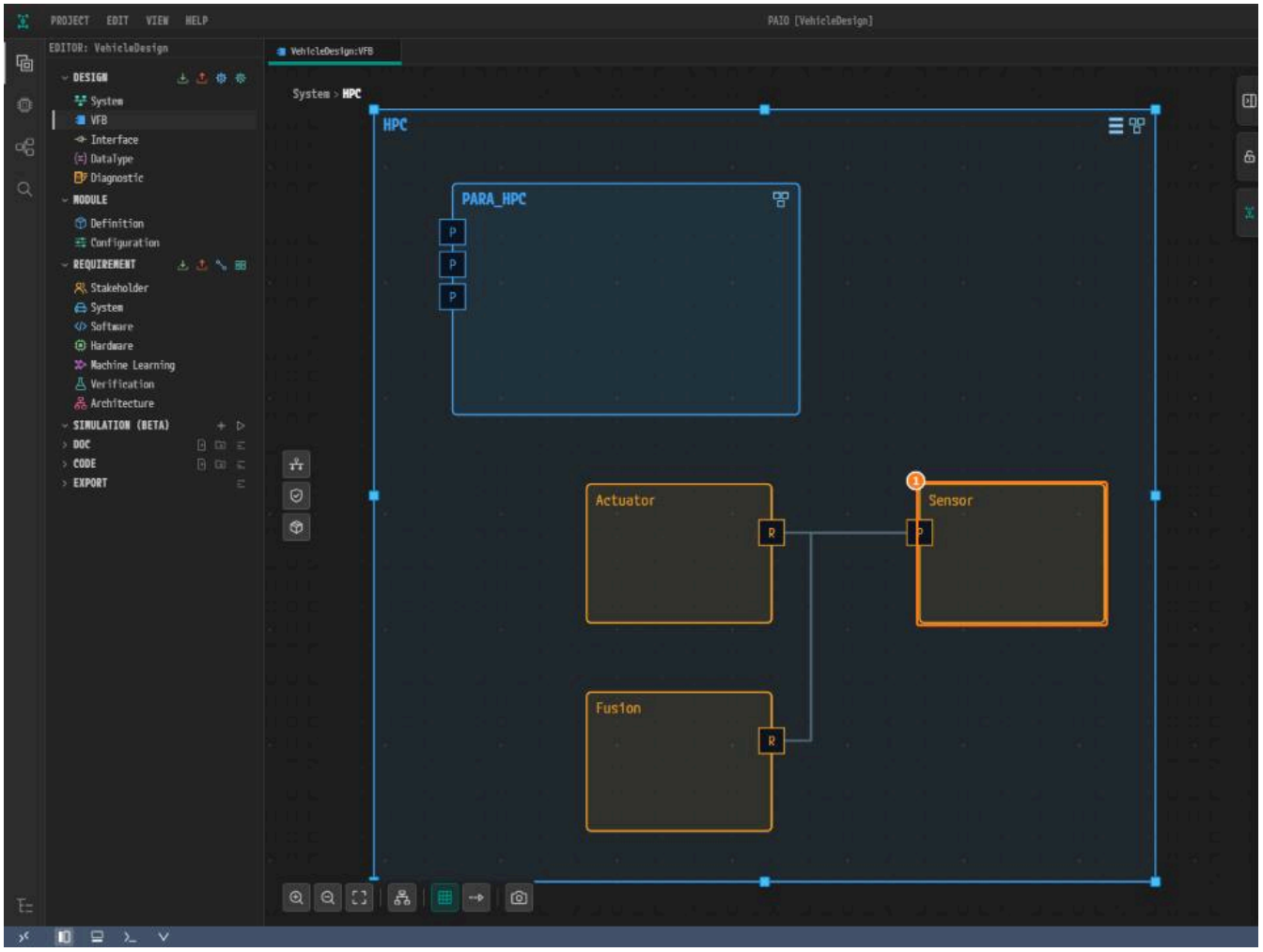


Figure 10-1. ① The node `Sensor`.

3. The dialog **AA Internal Behavior [Sensor]** opens. Click the tab **EXECUTION**.

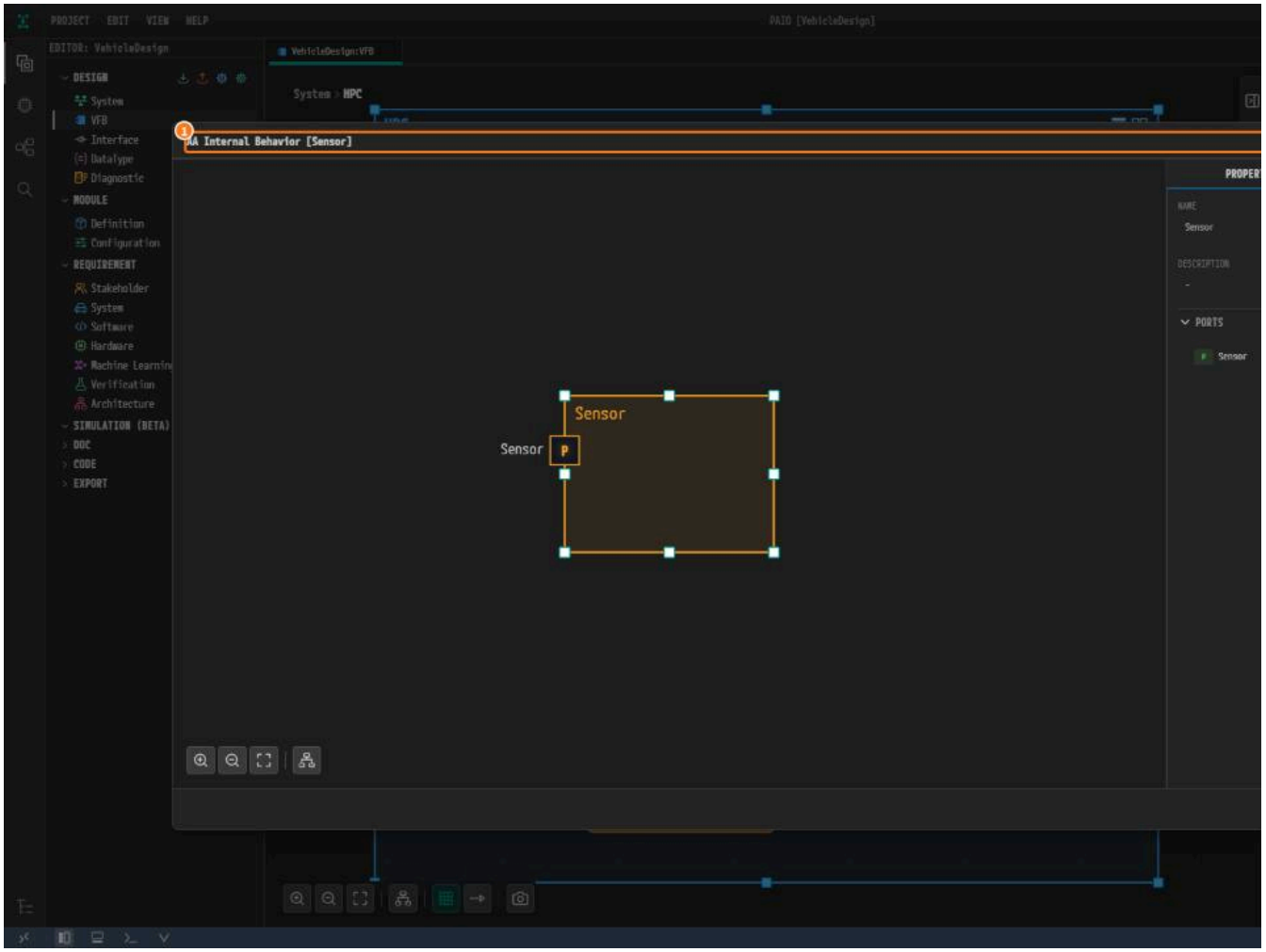


*Figure 10-2. ① The dialog title, ② **EXECUTION**.*

4. The **EXECUTABLE** section is already filled in. Leave it as it is. Click the **+** button of **PROCESS**.

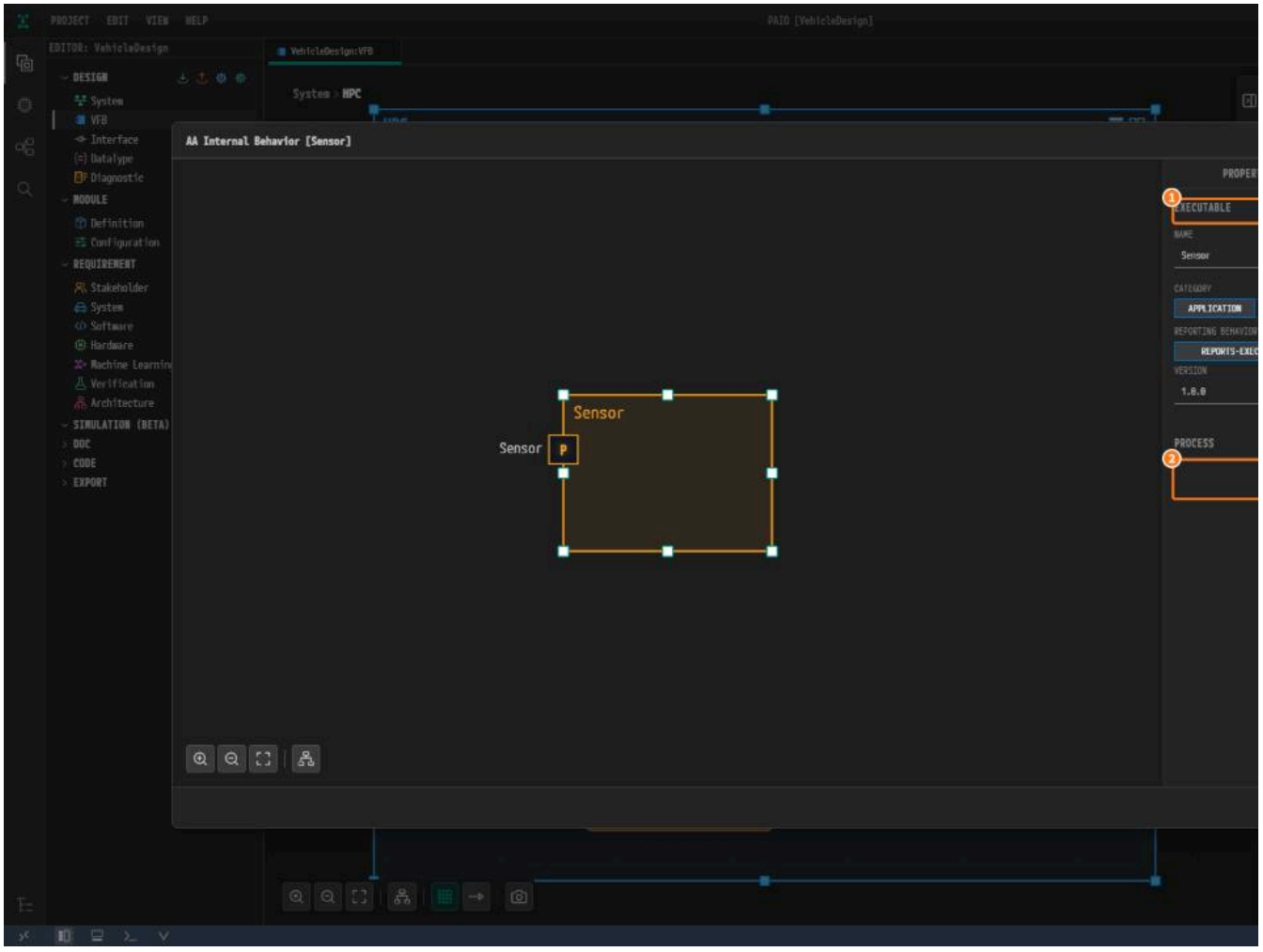


*Figure 10-3. ① **EXECUTABLE**, ② `No processes defined`, ③ the **+** button.*

5. Click the new card `Sensor_0` to expand it. Enter `Sensor` in **NAME**, `5` in **ENTER TIMEOUT** and `5` in **EXIT TIMEOUT**, pressing `Tab` after each.

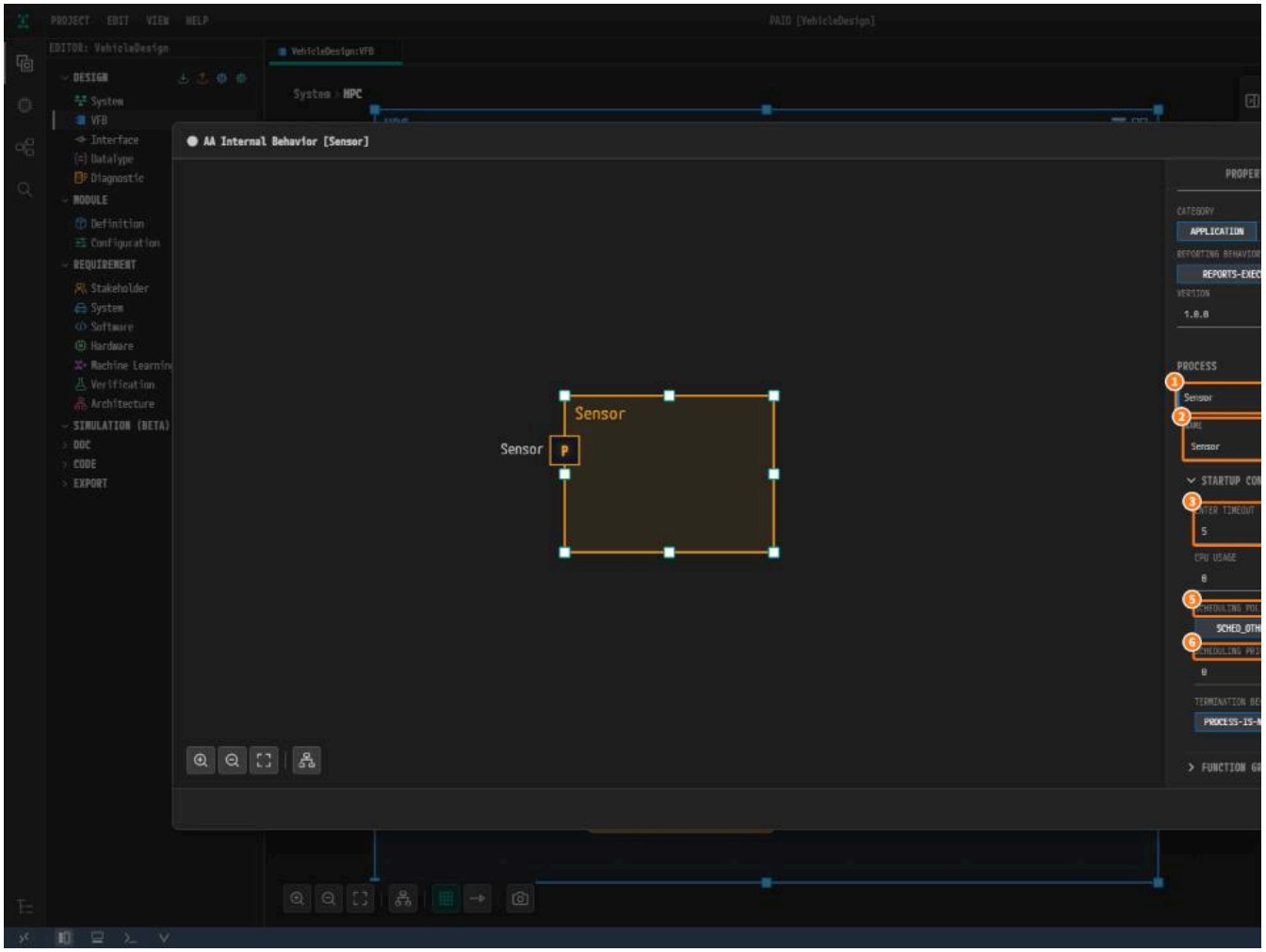


*Figure 10-4. ① The card header, ② **NAME**, ③ **ENTER TIMEOUT** = `5`, ④ **EXIT TIMEOUT** = `5`, ⑤ **SCHEDULING POLICY**, ⑥ **SCHEDULING PRIORITY**.*

6. Click the **+** button of **FUNCTION GROUP STATE**, which is red while the list is empty. A row `MachineFG`, `Off` appears. Click the cell `Off` under **FUNCTION**

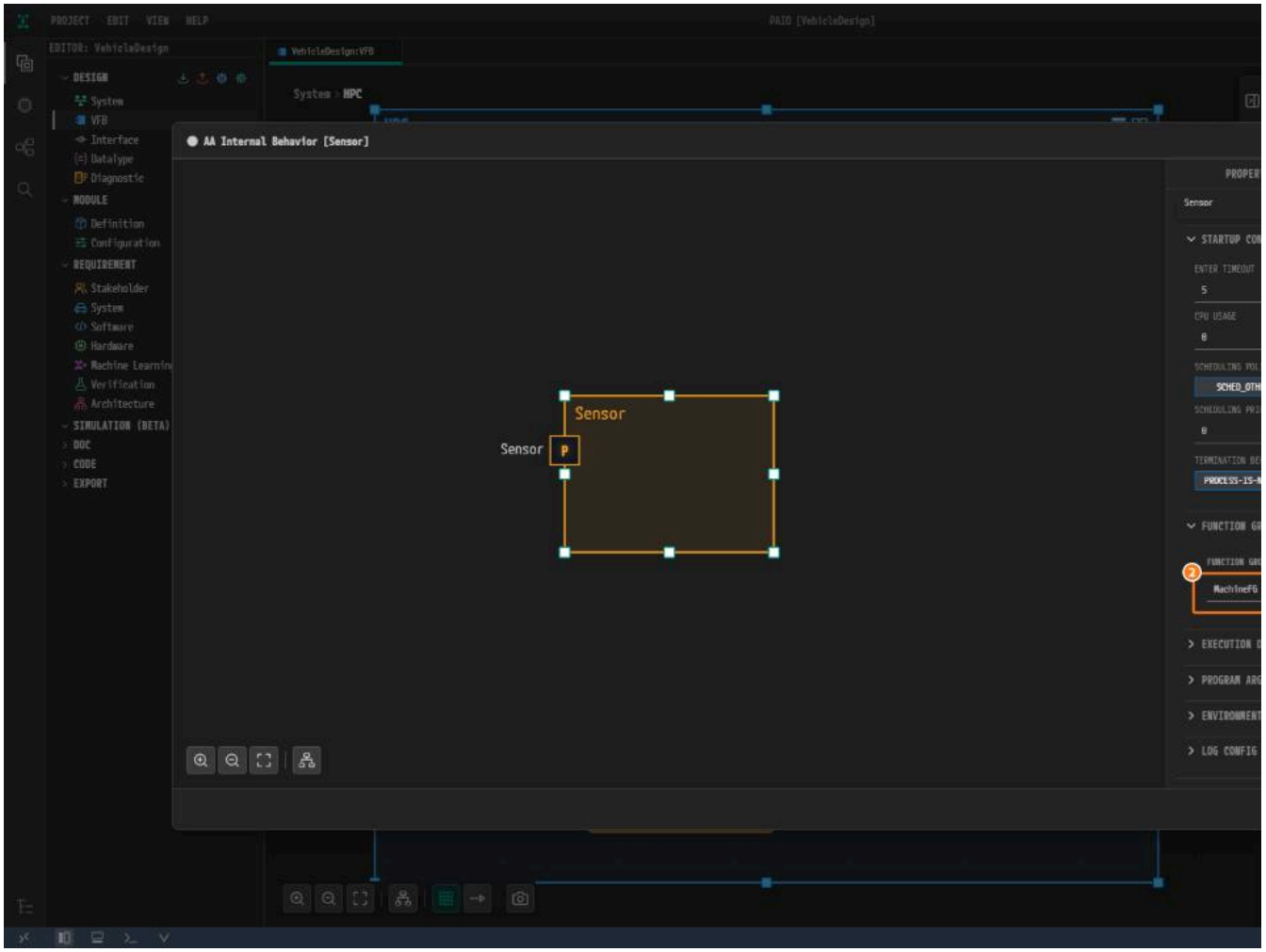


Figure 10-5. ① The + button, ② the new row, ③ `Startup`.

7. Click the + button of **EXECUTION DEPENDENCY**. A row `EM`, `Running` appears. Click the cell under **PROCESS** and click `CM`. Leave **PROCESS STATE** at `Runn`

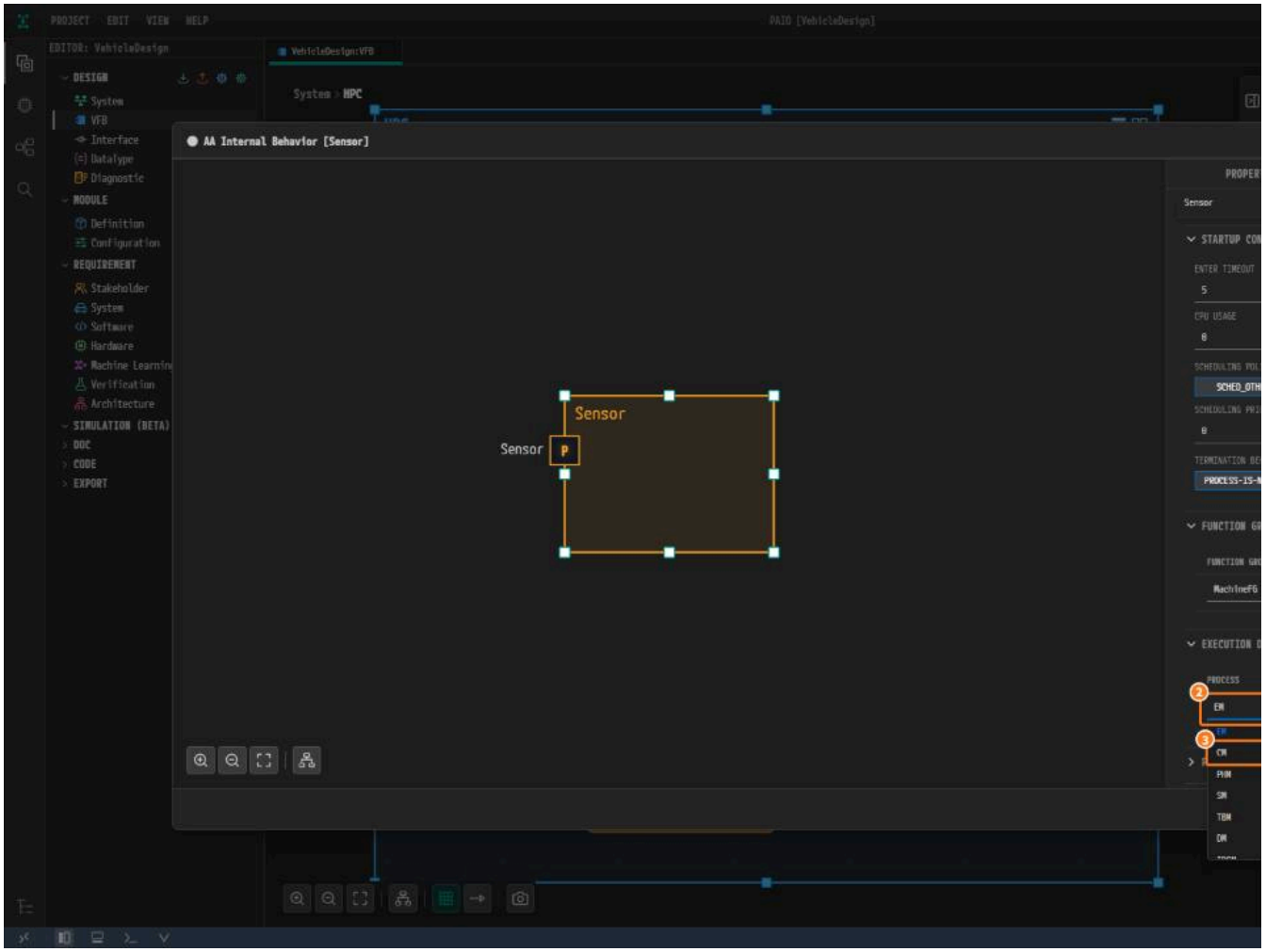


*Figure 10-6. ① The **+** button, ② **PROCESS**, ③ `CM`, ④ **PROCESS STATE** = `Running`.*

8. Press `Ctrl+S` in the dialog, then click **Close**.

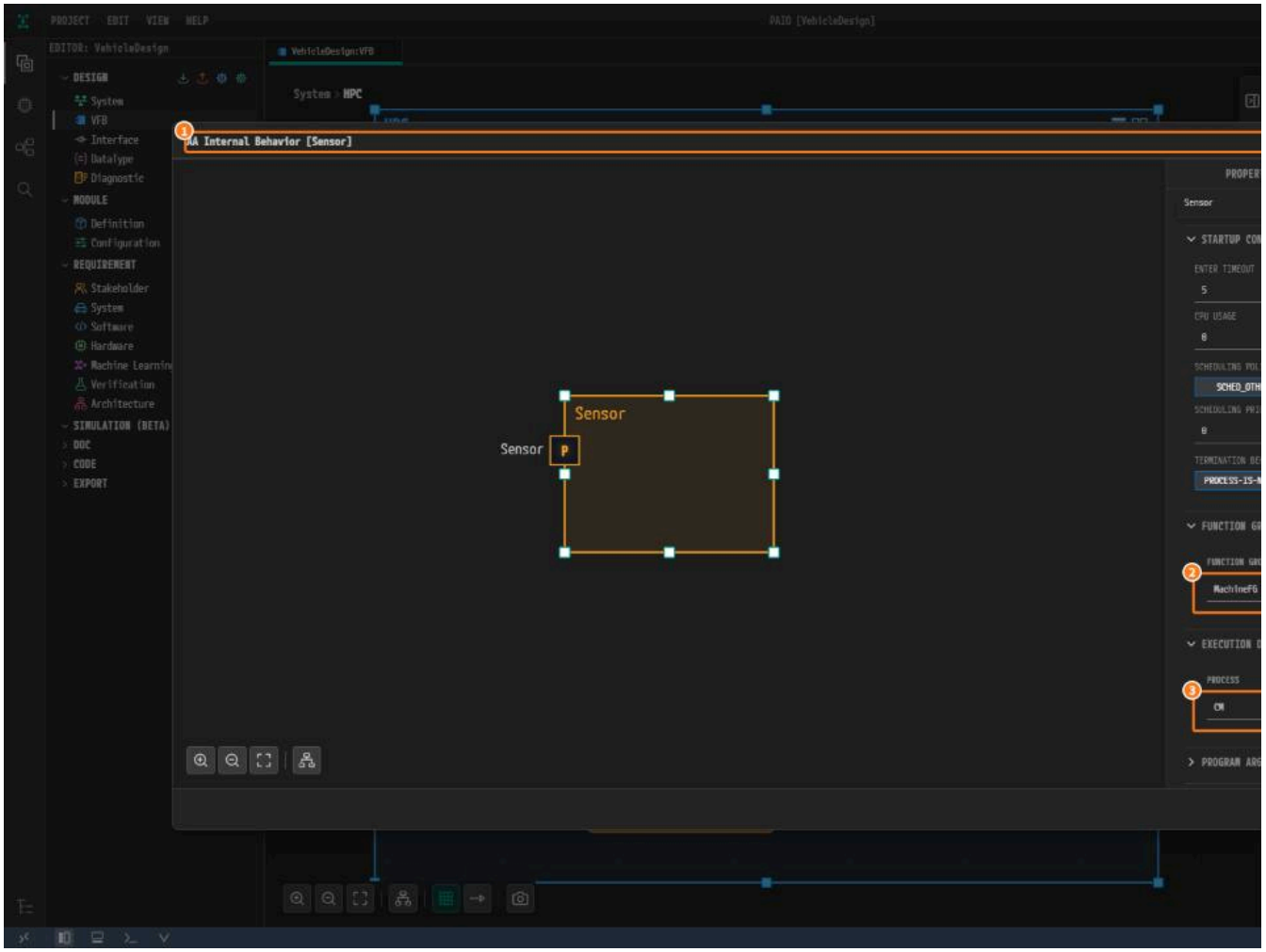


Figure 10-7. ① The title without ●, ② `MachineFG`, `Startup`, ③ `CM`, `Running`.

Creating the Processes of Fusion and Actuator

Create the other two the same way from the table, and finish each with `Ctrl+S` and **Close**.

> **Caution.** In `Fusion` and `Actuator` a new **EXECUTION DEPENDENCY** row starts with the process `Sensor`. Change **PROCESS** to `CM` before saving.

Checking in the PROCESSES Tab of the Machine

1. Click the machine icon at the top right of the frame `HPC`. **Machine Configuration [HPC]** opens.
2. Click the tab **PROCESSES**. The list holds `Sensor`, `Fusion` and `Actuator`, followed by the eleven platform processes.
3. Click `Sensor`. The **EXECUTION DEPENDENCY** graph on the right shows `Running` arrows from `CM` to `Sensor`, `Fusion` and `Actuator`.

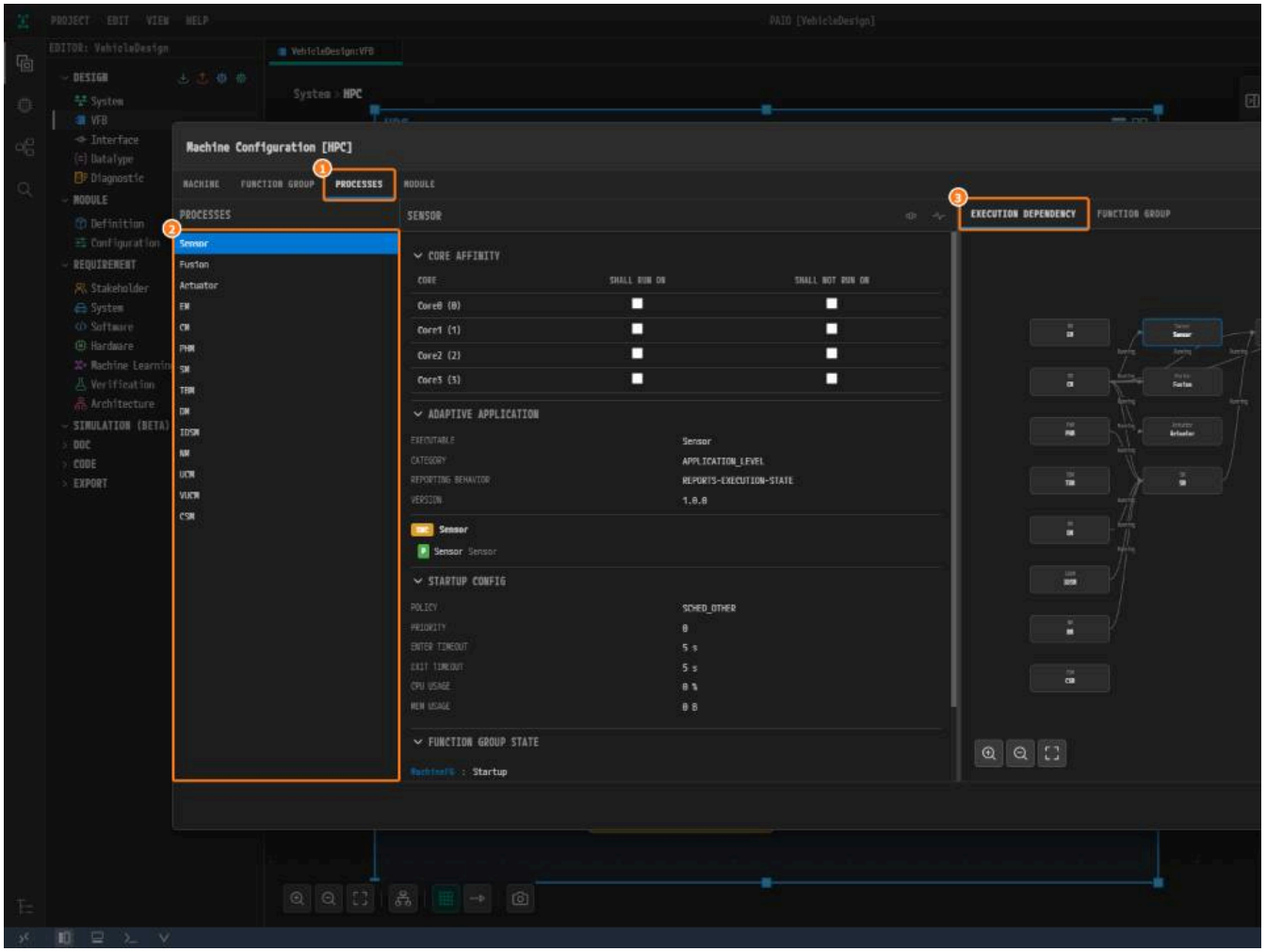


*Figure 10-8. ① **PROCESSES**, ② the fourteen processes, ③ the **EXECUTION DEPENDENCY** graph.*

4. Click **Close**.

Viewing the Validation Result

1. Open the **VALIDATION** panel. If it is closed, click the **V** icon in the footer. It shows `No validation issues.`.

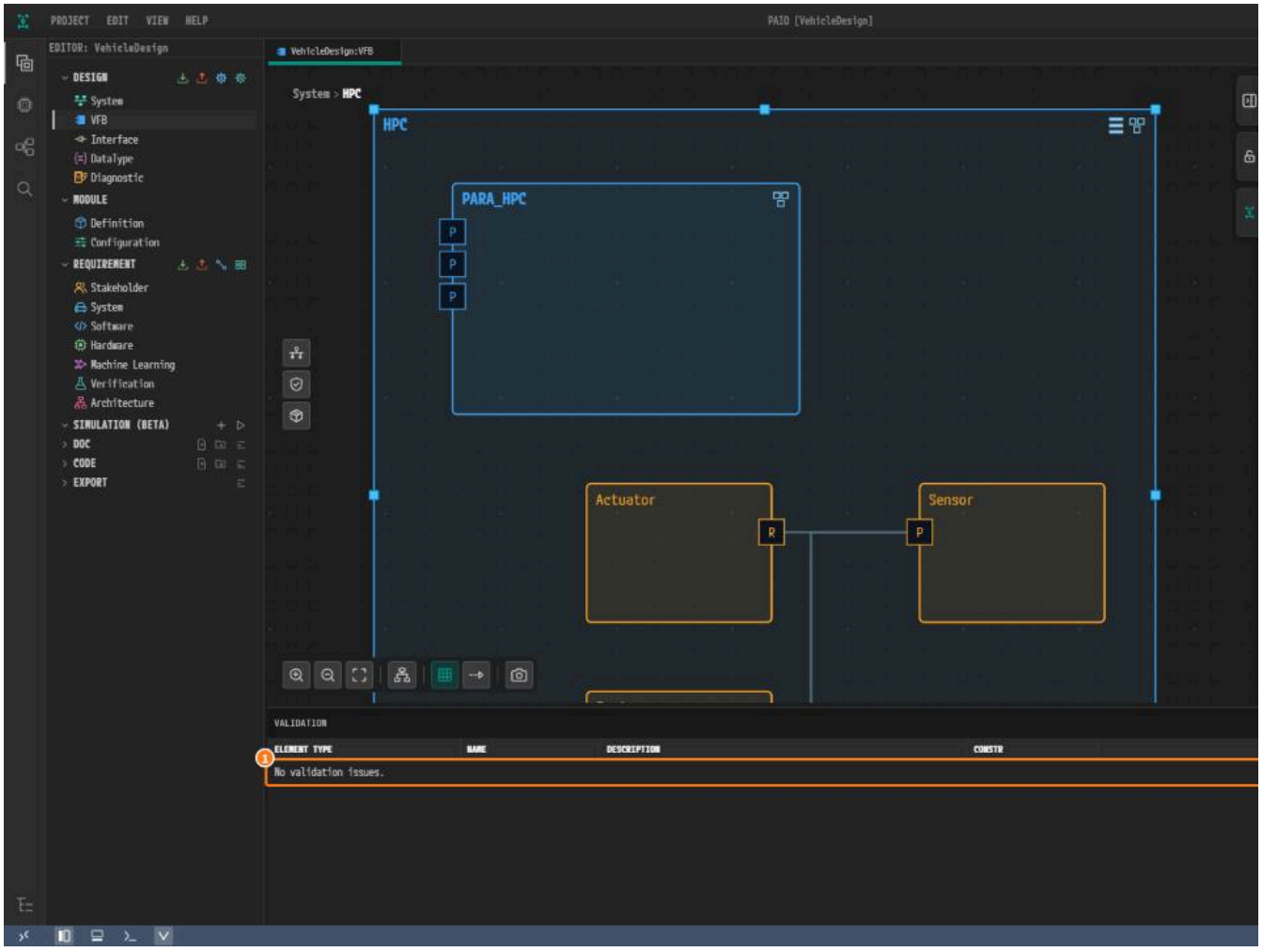


*Figure 10-9. The **VALIDATION** panel after all three dialogs are saved.*

Why Is It Designed This Way?

- **The Executable keeps its defaults.** The defaults already give the values this design exports.
- `CM` **comes first.** `Sensor`, `Fusion` and `Actuator` start only when `CM` is `Running`.
- **Every process needs a function group state.** Validation rejects a process without one.
- **The platform processes stay as PARA gives them.** Their timeouts, states and dependencies come with the import.

Checklist

- [ ] Each of the three Adaptive Applications has one process card with its own name and the timeouts `5` / `5`.
- [ ] Each has one **FUNCTION GROUP STATE** row `MachineFG`, `Startup`. No row reads `Off`.
- [ ] Each has the dependency `CM`, `Running`.
- [ ] The **PROCESSES** tab of **Machine Configuration [HPC]** lists fourteen processes.
- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In Service Instance Mapping you map the ports of `Sensor`, `Fusion` and `Actuator` to the SOME/IP service instances.

Service Instance Mapping

What You Will Do

In this chapter you map the port of `Sensor` to the provider instance `Provider_Sensor`, and the ports of `Fusion` and `Actuator` to the consumer instan the two consumers you enter a **CLIENT ID**.

Before You Begin

- Start after finishing Executables and Processes. The instances from SOME/IP Deployment and Service Instances are bound to the connector as in Bi
- Only the ports of your three applications are mapped. Leave the applications in `PARA_HPC` as they are.

	`Sensor`	`Fusion`	`Actuator`
PORT	`Sensor` (`P`)	`Sensor` (`R`)	`Sensor` (`R`)
SERVICE INSTANCE	`Provider_Sensor (SOME/IP)`	`Fusion_Sensor (SOME/IP)`	`Actuator_Sensor (SOME/IP)`
CLIENT ID	no field	`2` → `0x2`	`1` → `0x1`
Mapping box	`Sensor/AA/Sensor`	`Fusion/AA/Sensor`	`Actuator/AA/Sensor`
Read-back **ENDPOINT**	`192.168.0.11:20003`	`192.168.0.11:20002`	`192.168.0.11:20001`

Mapping the Port of Sensor to Provider_Sensor

Opening the AA Internal Behavior Dialog

1. In the **EDITOR** view, click **VFB** under **DESIGN**, then double-click the node `HPC [HPC]`.

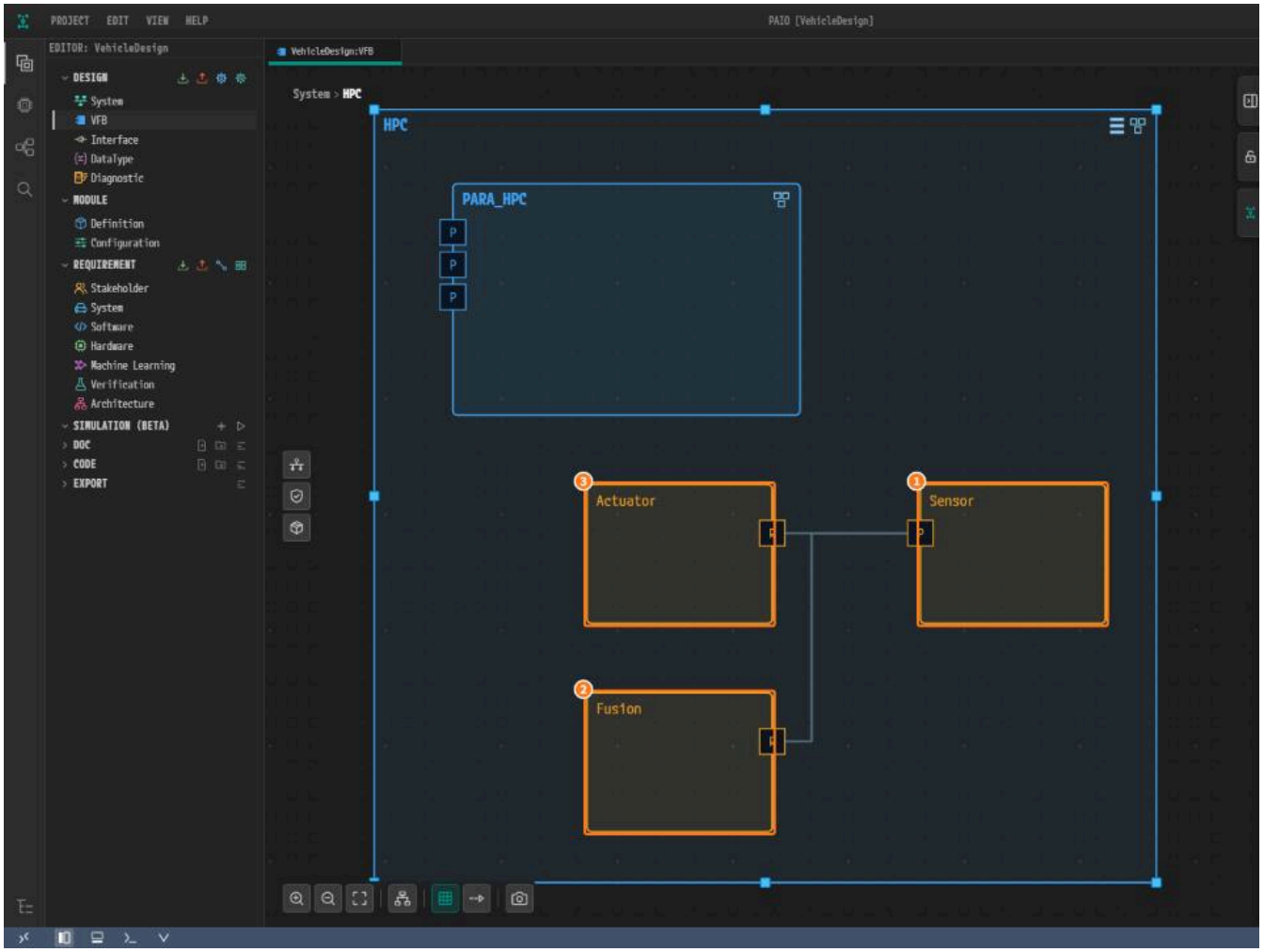


Figure 11-1. ① `Sensor`, ② `Fusion`, ③ `Actuator`. Your node arrangement may differ.

2. Double-click the node `Sensor`. The dialog **AA Internal Behavior [Sensor]** opens.

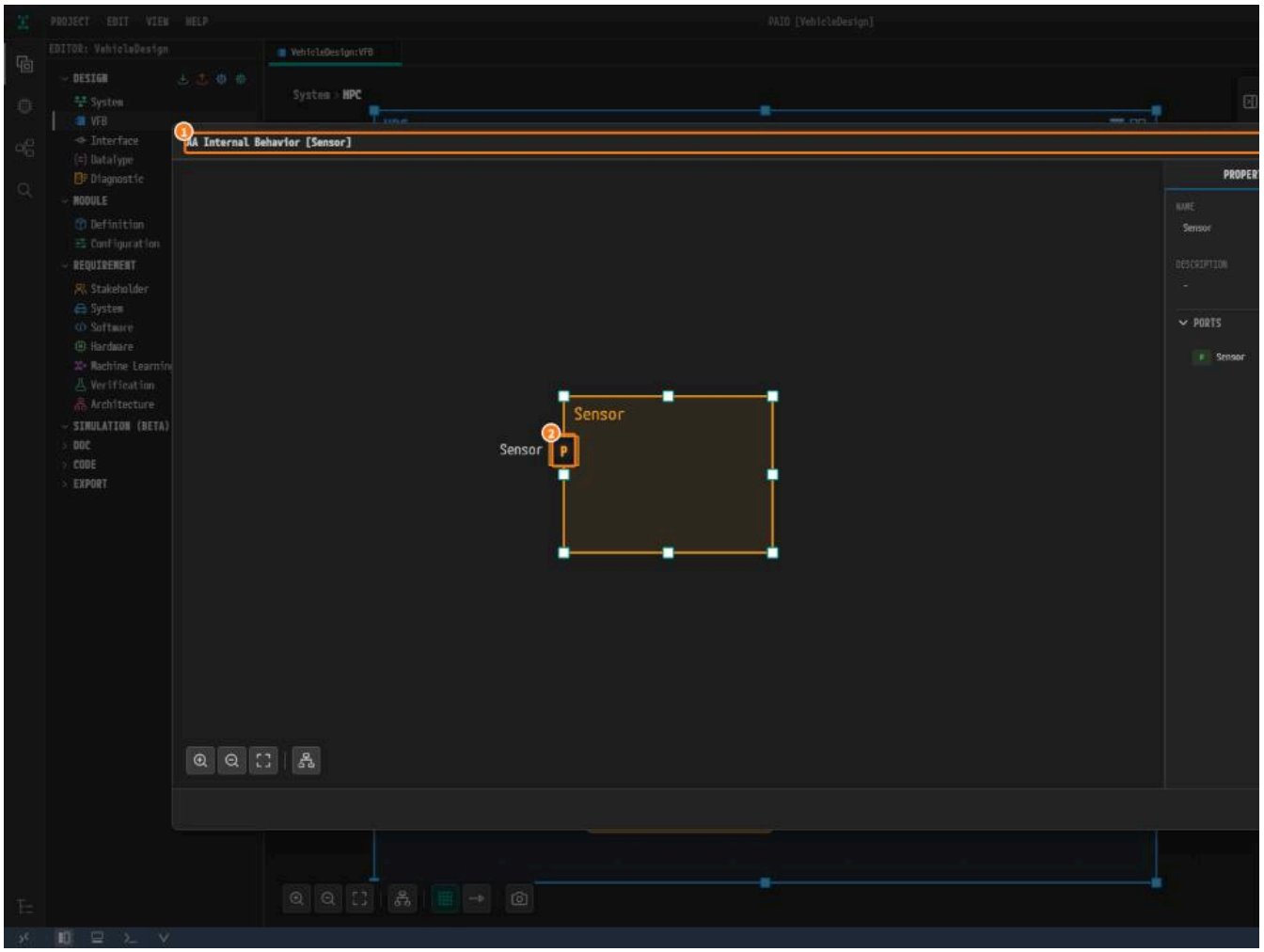


Figure 11-2. ① The dialog title, ② the `P` port `Sensor`.

Creating the Mapping Box from the Add Instance Menu

1. Right-click an empty spot of the `Sensor` body.
2. Hover over **Add Instance**.
3. Hover over **ara::com**.

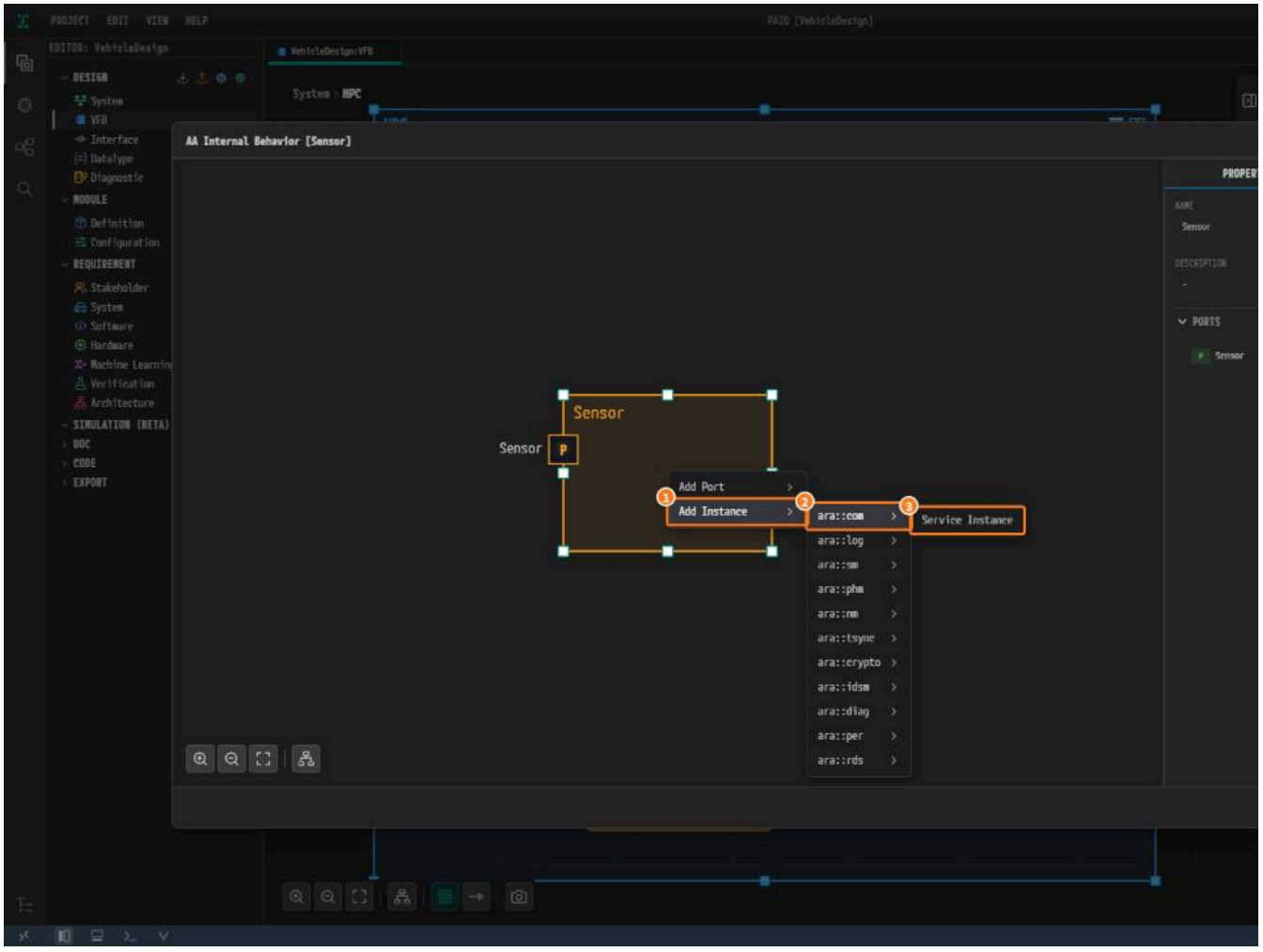


*Figure 11-3. ① **Add Instance**, ② **ara::com**, ③ **Service Instance**.*

4. Click **Service Instance**. A mapping box `Sensor/AA/` appears and the panel changes to **SERVICE INSTANCE MAPPING**. **PORT** and **SERVICE INSTANCE** are r

Picking the PORT

1. Click the **PORT** field.

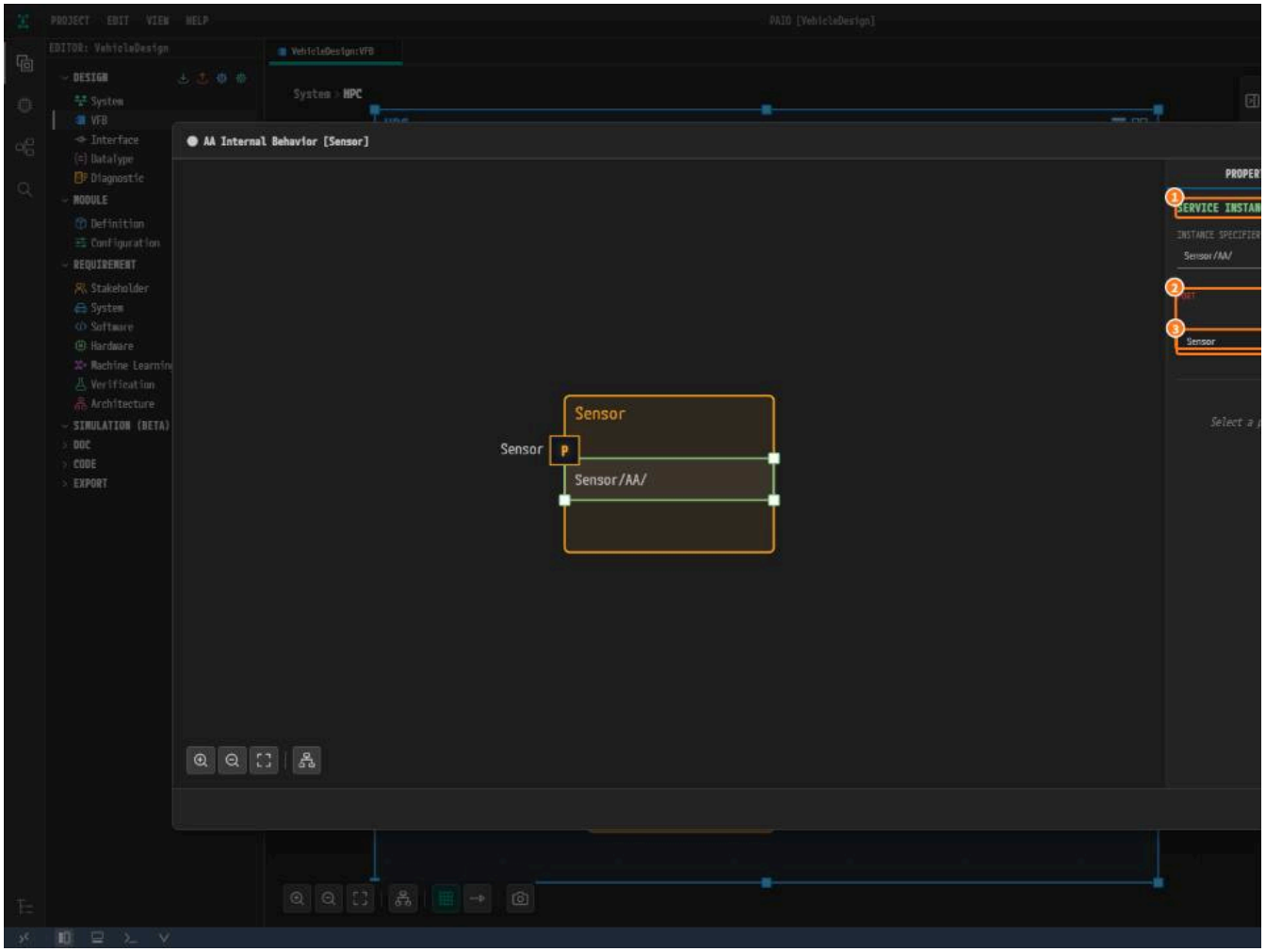


*Figure 11-4. ① **SERVICE INSTANCE MAPPING**, ② **PORT**, ③ `Sensor`.*

2. Click `Sensor`. The box becomes `Sensor/AA/Sensor`.

> **Caution.** Pick **PORT** first. The **SERVICE INSTANCE** list then shows only matching instances: providers on a `P` port, consumers on an `R` port.

Picking the SERVICE INSTANCE

1. Click the **SERVICE INSTANCE** field.

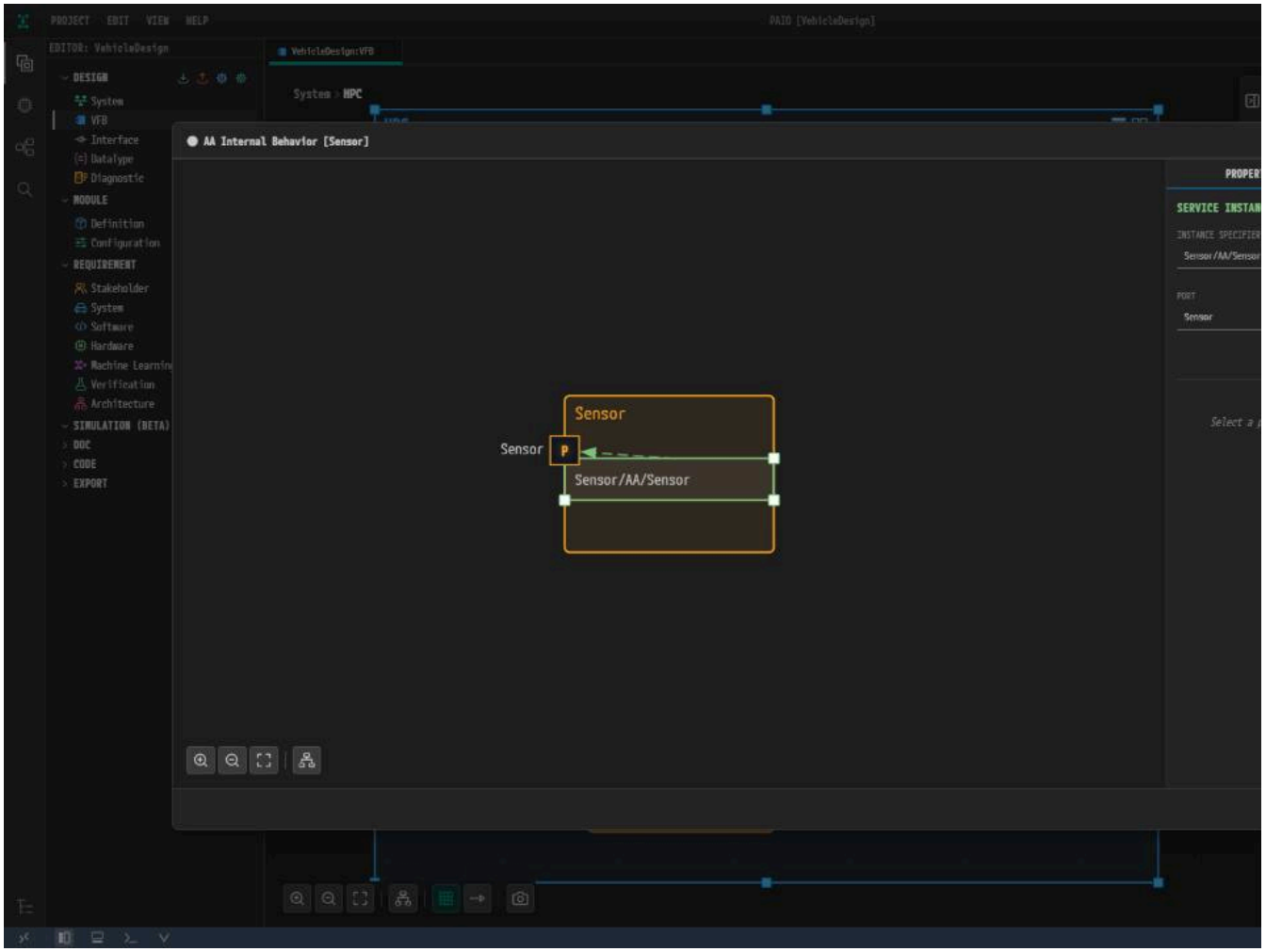


*Figure 11-5. ① **SERVICE INSTANCE**, ② `Provider_Sensor (SOME/IP)`.*

2. Click `Provider_Sensor (SOME/IP)`. The read-only block **REFERENCED INSTANCE** appears. A provider has no **CLIENT ID** field.

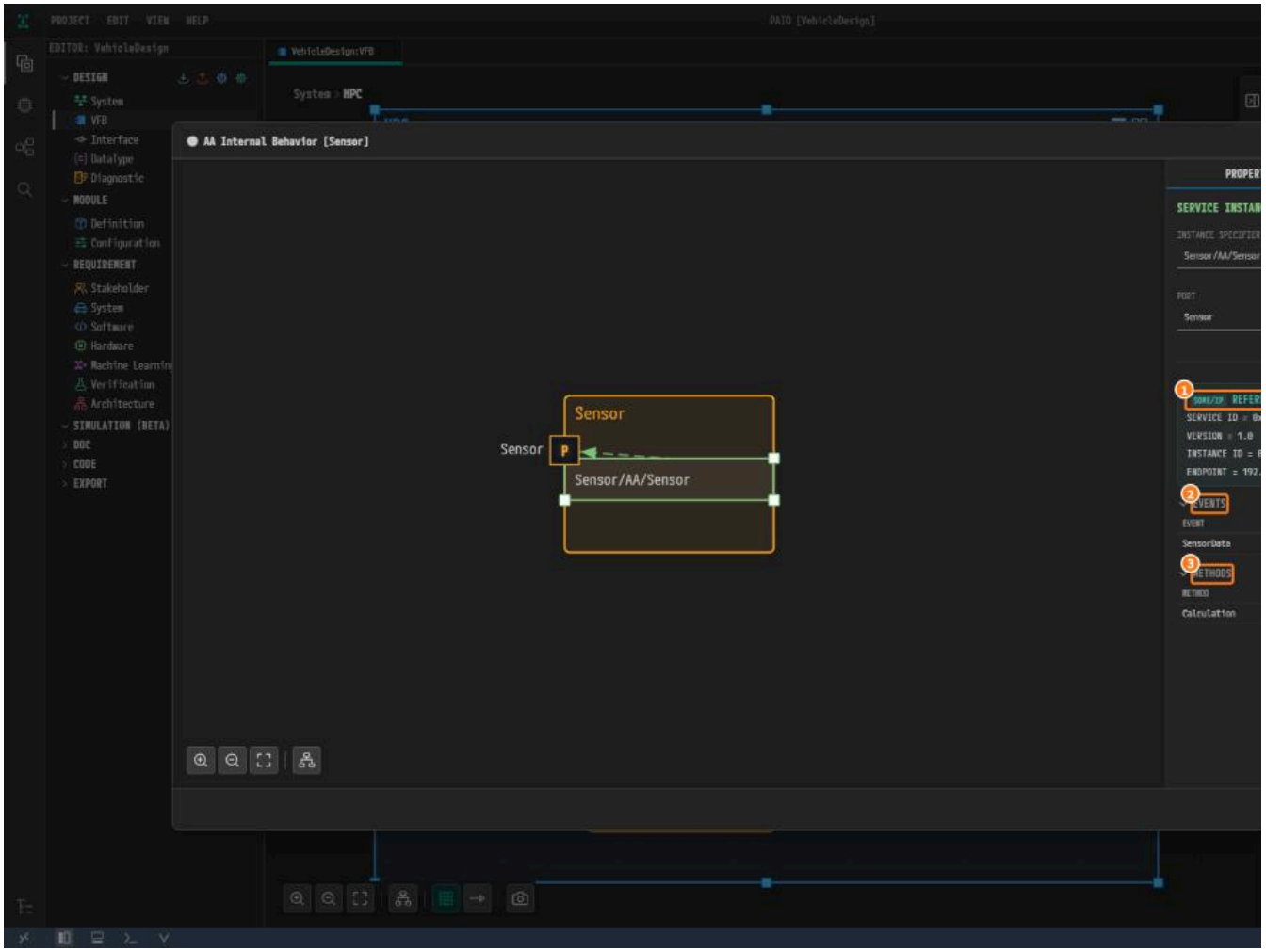


*Figure 11-6. ① **REFERENCED INSTANCE** (`ENDPOINT = 192.168.0.11:20003`), ② **EVENTS**, ③ **METHODS**.*

Saving and Closing

1. Press `Ctrl+S` inside the dialog.
2. Click **Close**.

Mapping the Port of Fusion to Fusion_Sensor

`Fusion` is a consumer. The steps are the same as for `Sensor`, plus the **CLIENT ID**.

1. Double-click the node `Fusion`.
2. Right-click the body and click **Add Instance** > **ara::com** > **Service Instance**.
3. In **PORT**, pick `Sensor`.
4. In **SERVICE INSTANCE**, pick `Fusion_Sensor (SOME/IP)`.
5. Check that **ENDPOINT** reads `192.168.0.11:20002`.
6. Type `2` in **CLIENT ID** and press `Tab`. The field shows `0x2`.

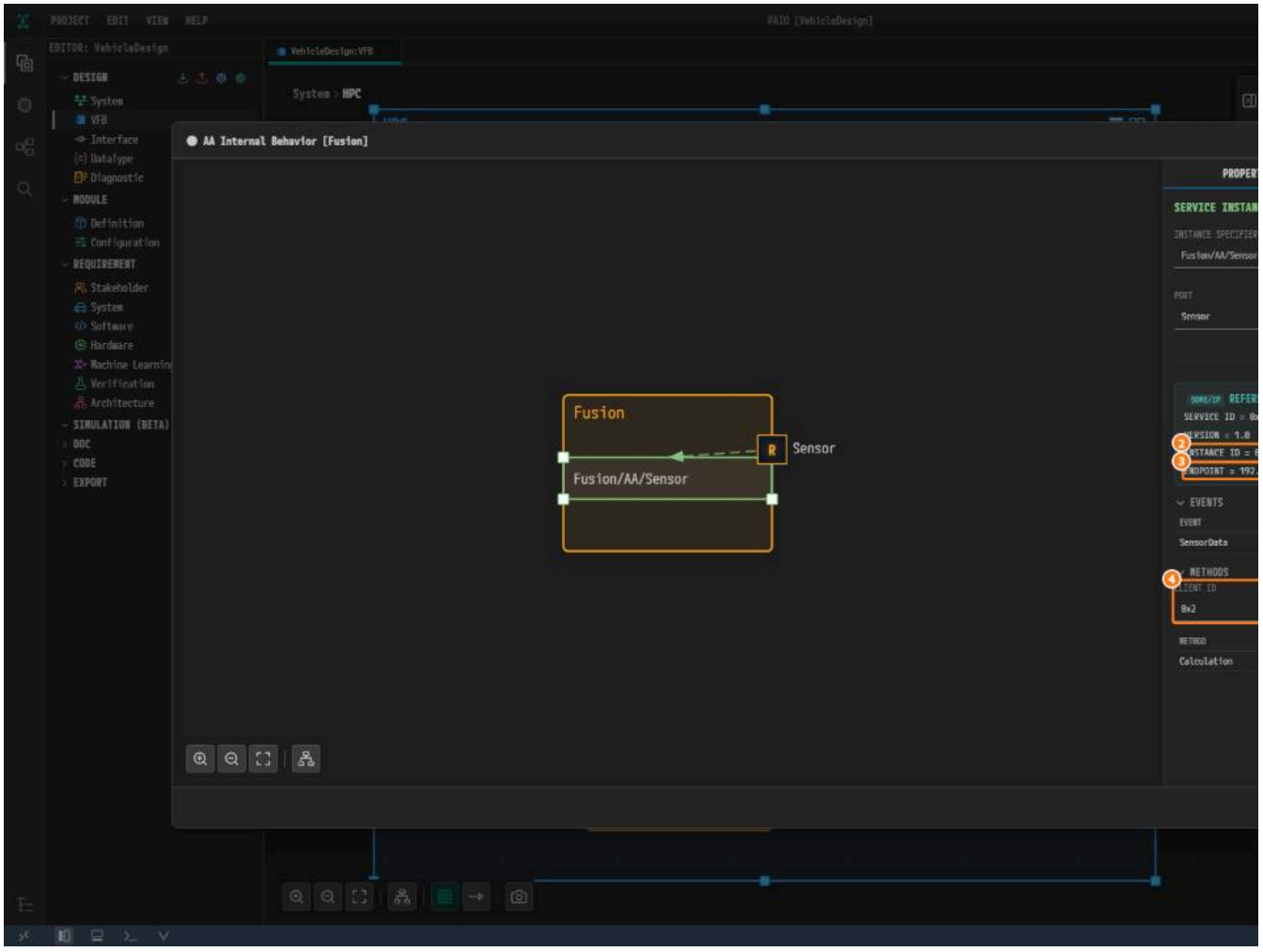


*Figure 11-7. ① **SERVICE INSTANCE**, ② `INSTANCE ID = 0xFFFF`, ③ `ENDPOINT = 192.168.0.11:20002`, ④ **CLIENT ID** `0x2`.*

7. Press `Ctrl+S`, then click **Close**.

Mapping the Port of Actuator to Actuator_Sensor

Map `Actuator` the same way: **SERVICE INSTANCE** `Actuator_Sensor (SOME/IP)`, **CLIENT ID** `1`.

> **Caution.** `Fusion` and `Actuator` see the same two items. Check the **ENDPOINT**: `Actuator` must read `192.168.0.11:20001`.

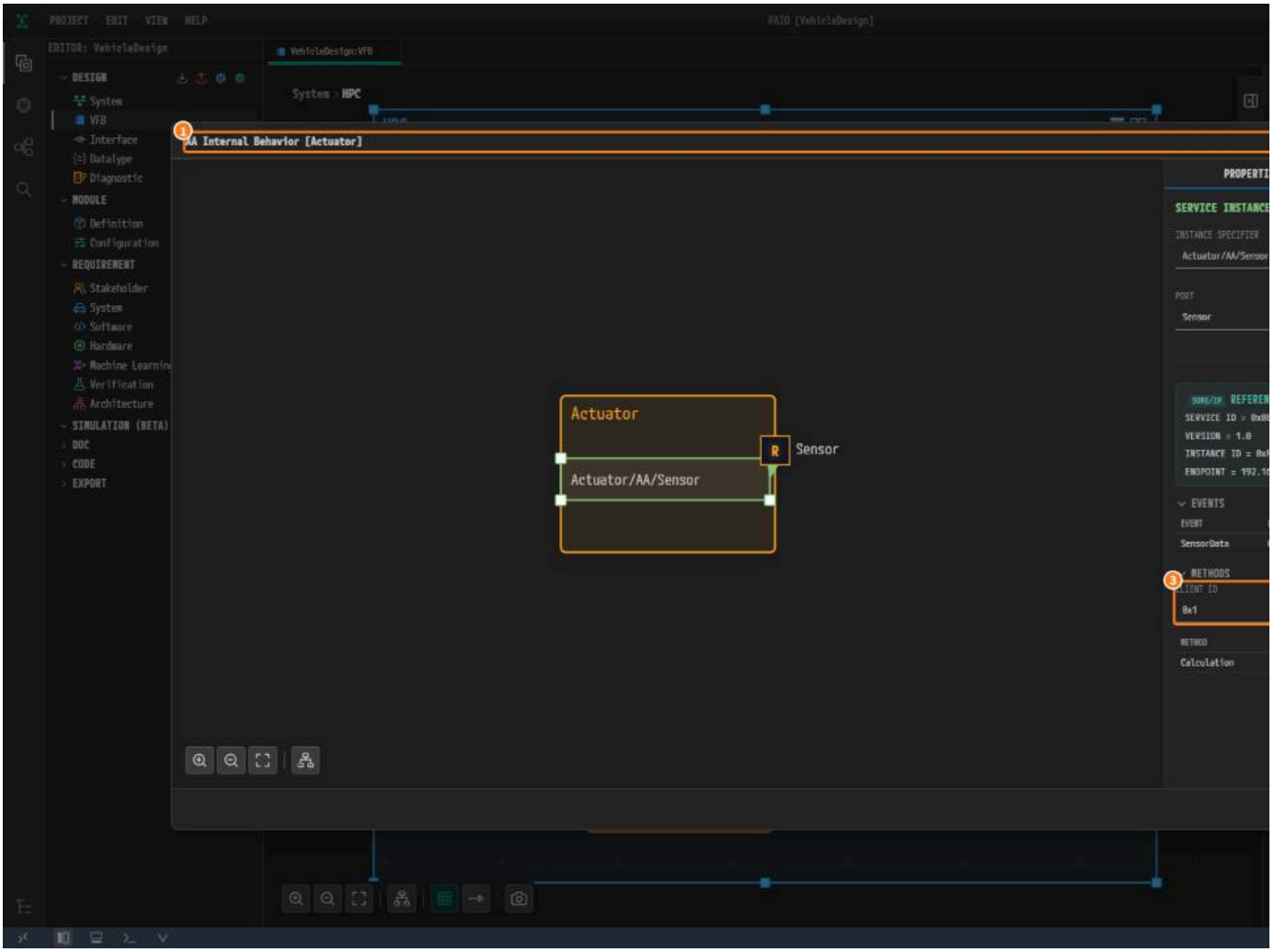


*Figure 11-8. ① The title without the modified mark, ② **SERVICE INSTANCE**, ③ **CLIENT ID** `0x1`.*

The mapping boxes are visible only inside the dialogs; the `HPC` canvas looks unchanged.

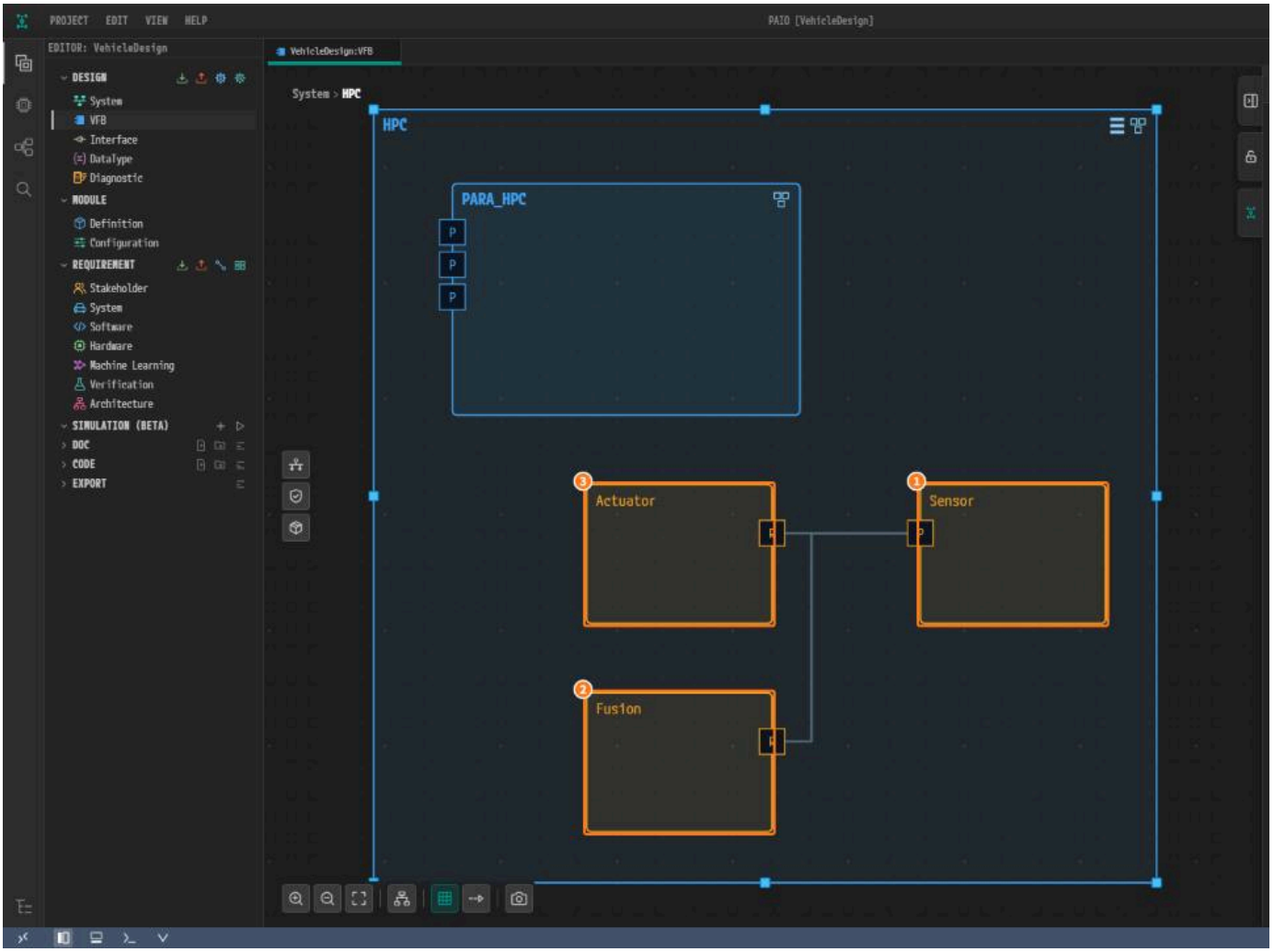


Figure 11-9. ① `Sensor`, ② `Fusion`, ③ `Actuator`.

Viewing the Validation Result

1. Look at the **VALIDATION** panel. If it is closed, click the **V** icon in the footer. It reads `No validation issues.`.

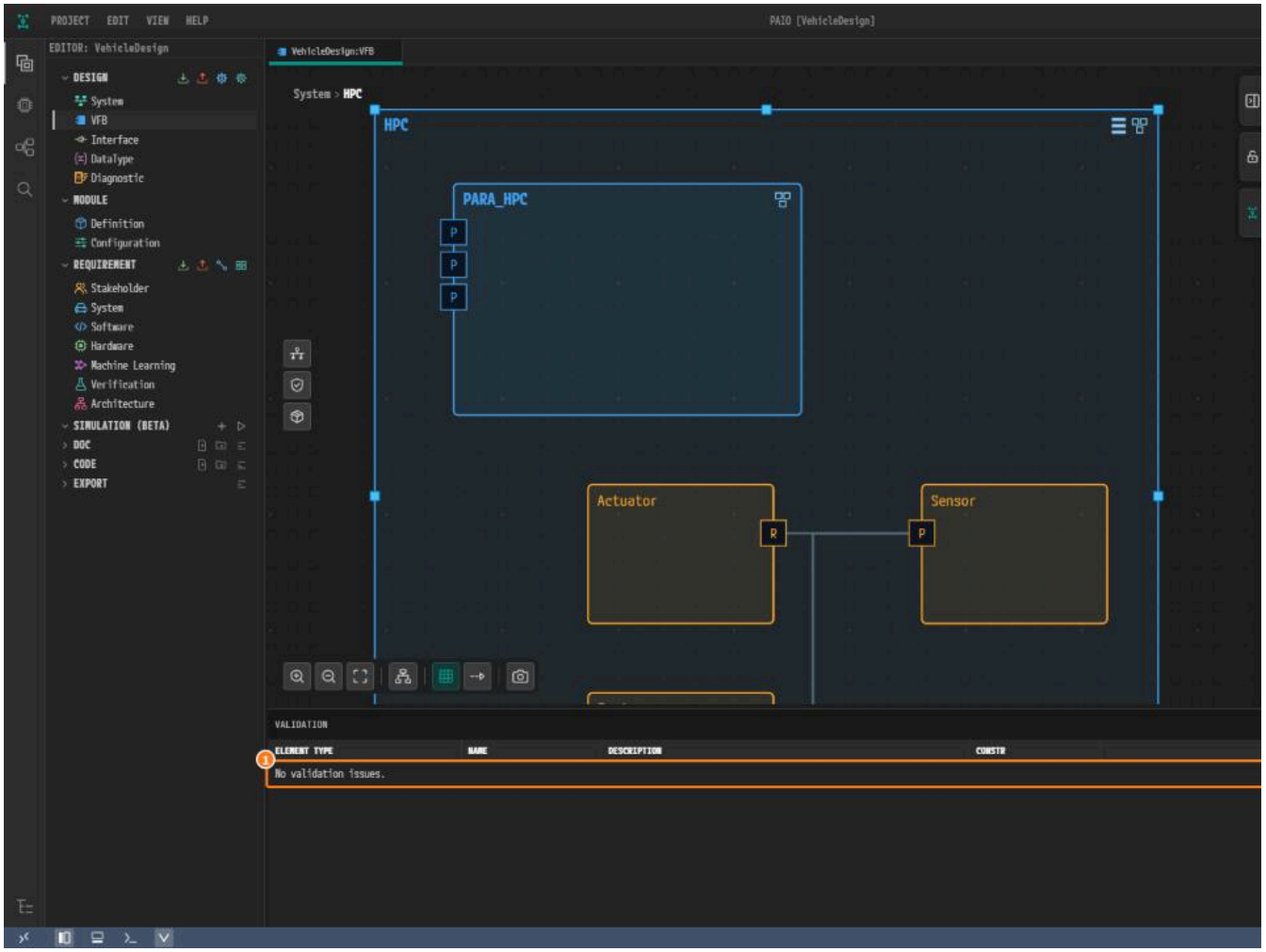


Figure 11-10. `No validation issues.`.

Why Is It Designed This Way?

- **Only your own ports are mapped.** This design offers no platform service of PARA on the network, so those ports stay unmapped.
- **CLIENT ID on consumers only.** A provider mapping has no **CLIENT ID** field.
- **The read-back block only shows earlier values.** Service ID, Instance ID and endpoint are changed in the chapters that set them, not here.

Checklist

- [ ] The three dialogs hold the boxes `Sensor/AA/Sensor`, `Fusion/AA/Sensor` and `Actuator/AA/Sensor`.
- [ ] The **ENDPOINT** is `…:20003` for `Sensor`, `…:20002` for `Fusion` and `…:20001` for `Actuator`.
- [ ] The **CLIENT ID** is `0x2` for `Fusion` and `0x1` for `Actuator`.
- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In Logging and DLT you enter the log settings of your three processes.

Logging and DLT

What You Will Do

In this chapter you fill in the **LOG CONFIG** of the processes `Sensor`, `Fusion` and `Actuator`: a DLT (Diagnostic Log and Trace) application ID, a default log l
the result in the machine's **Log and Trace** table.

Before You Begin

- Start after finishing Service Instance Mapping.
- The log settings of the platform processes stay as PARA brought them.
- Leave **LOG MODE** at `CONSOLE` and the context **LOG LEVEL** at `INFO` everywhere.

Process	APPLICATION ID	APPLICATION DESCRIPTION	DEFAULT LOG LEVEL	CONTEXT ID	CONTEXT DESCRIPTION
`Sensor`	`SENS`	`Sensor application`	`VERBOSE`	`SENS`	`Sensor context`
`Fusion`	`FUSN`	`Fusion application`	`VERBOSE`	`FUSN`	`Fusion context`
`Actuator`	`ACTA`	`Actuator application`	`VERBOSE`	`ACTA`	`Actuator context`

Entering the LOG CONFIG and DLT Context of Sensor

Expanding LOG CONFIG

1. In the **EDITOR** view, click **VFB** under **DESIGN**, then double-click the node `HPC [HPC]`.

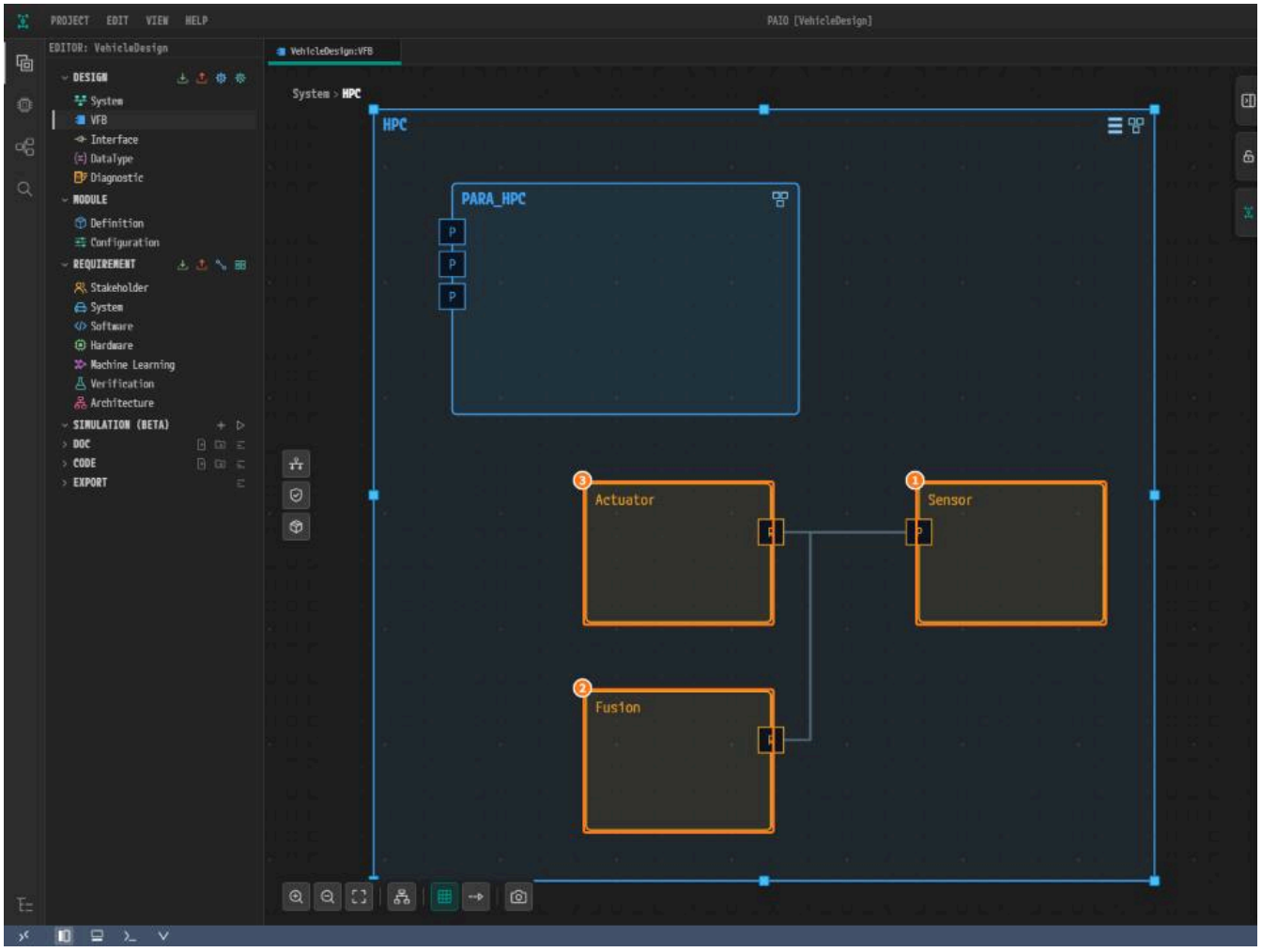


Figure 12-1. ① `Sensor`, ② `Fusion`, ③ `Actuator`. Your node arrangement may differ.

2. Double-click the node `Sensor`.

3. Click the **EXECUTION** tab. If the process card `Sensor` is collapsed, click its header.

4. Click the **LOG CONFIG** header to expand it.

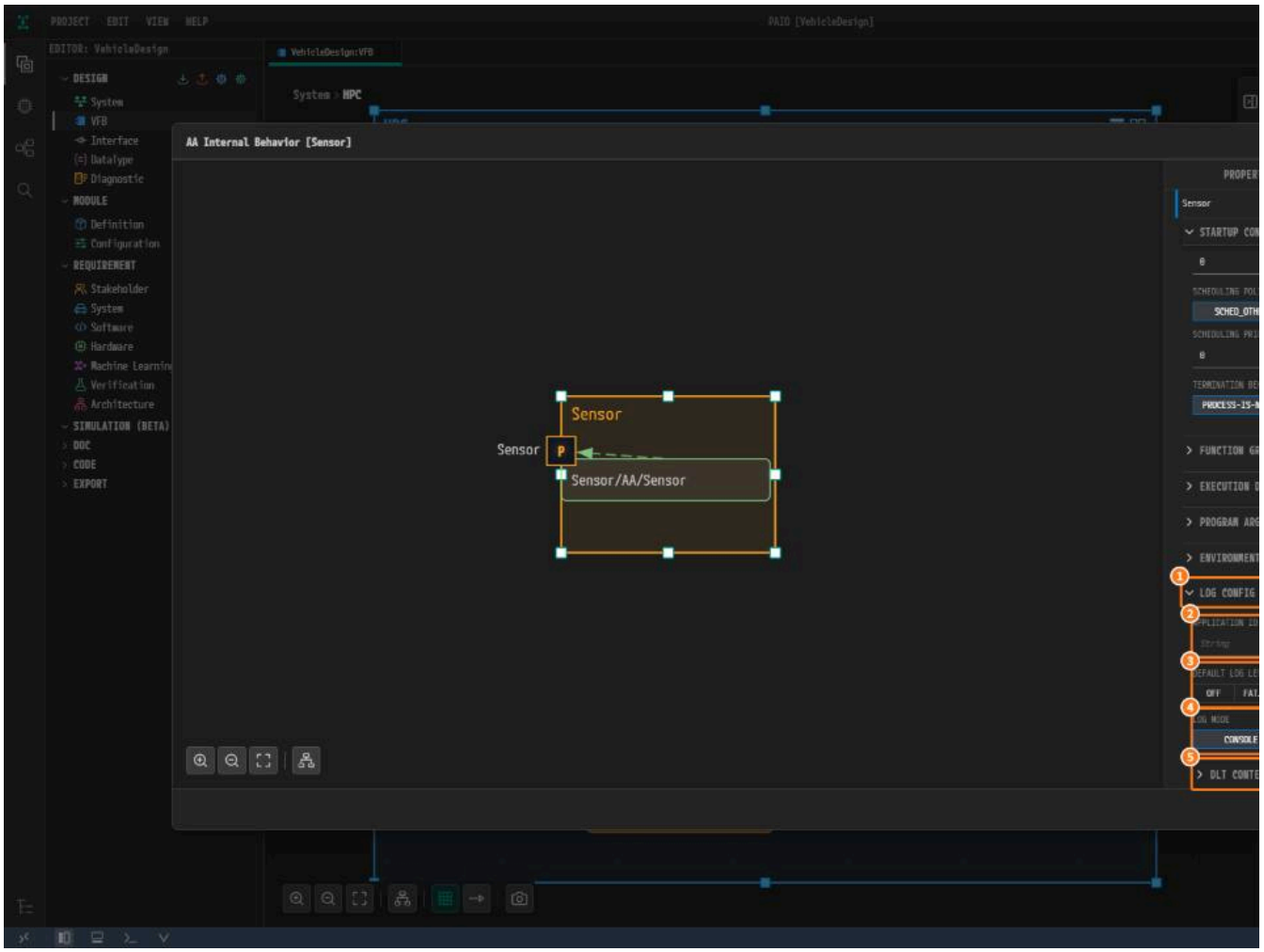


*Figure 12-2. ① **LOG CONFIG**, ② **APPLICATION ID**, ③ **DEFAULT LOG LEVEL**, ④ **LOG MODE**, ⑤ **DLT CONTEXTS**.*

> **Note.** The level buttons are narrow, so `VERBOSE` shows as `VER…`.

Entering APPLICATION ID and DEFAULT LOG LEVEL

1. Type `SENS` in **APPLICATION ID** and press `Tab`.

2. Type `Sensor application` in **APPLICATION DESCRIPTION** and press `Tab`.

3. In **DEFAULT LOG LEVEL**, click `VERBOSE`.

4. Leave **LOG MODE** at `CONSOLE`.

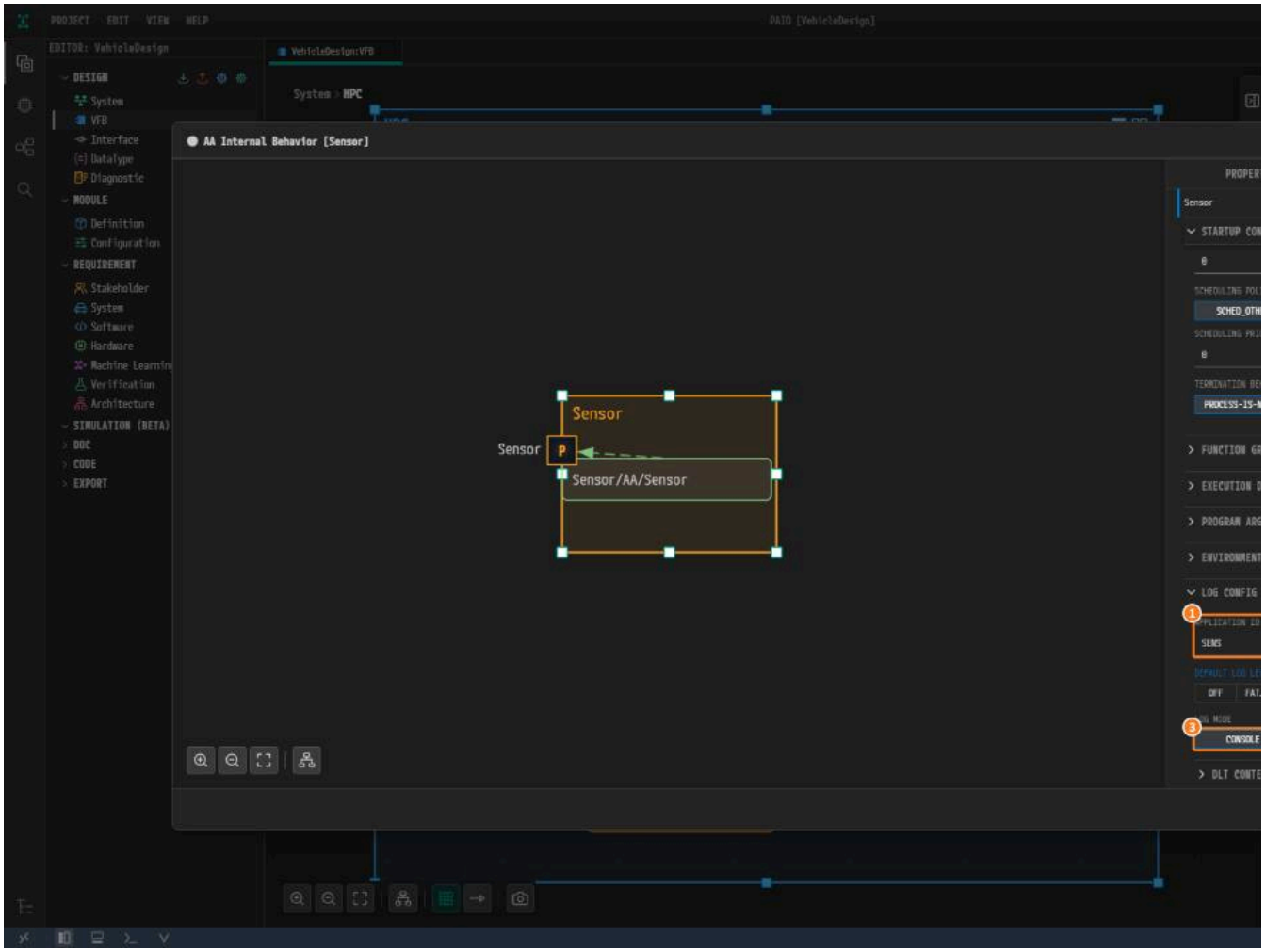


*Figure 12-3. ① **APPLICATION ID** `SENS`, ② `VERBOSE`, ③ `CONSOLE`.*

Adding a DLT Context

1. Click the **DLT CONTEXTS** header to expand it.
2. Click the **+** button at its right. A collapsed row `Context_0 : INFO` appears.

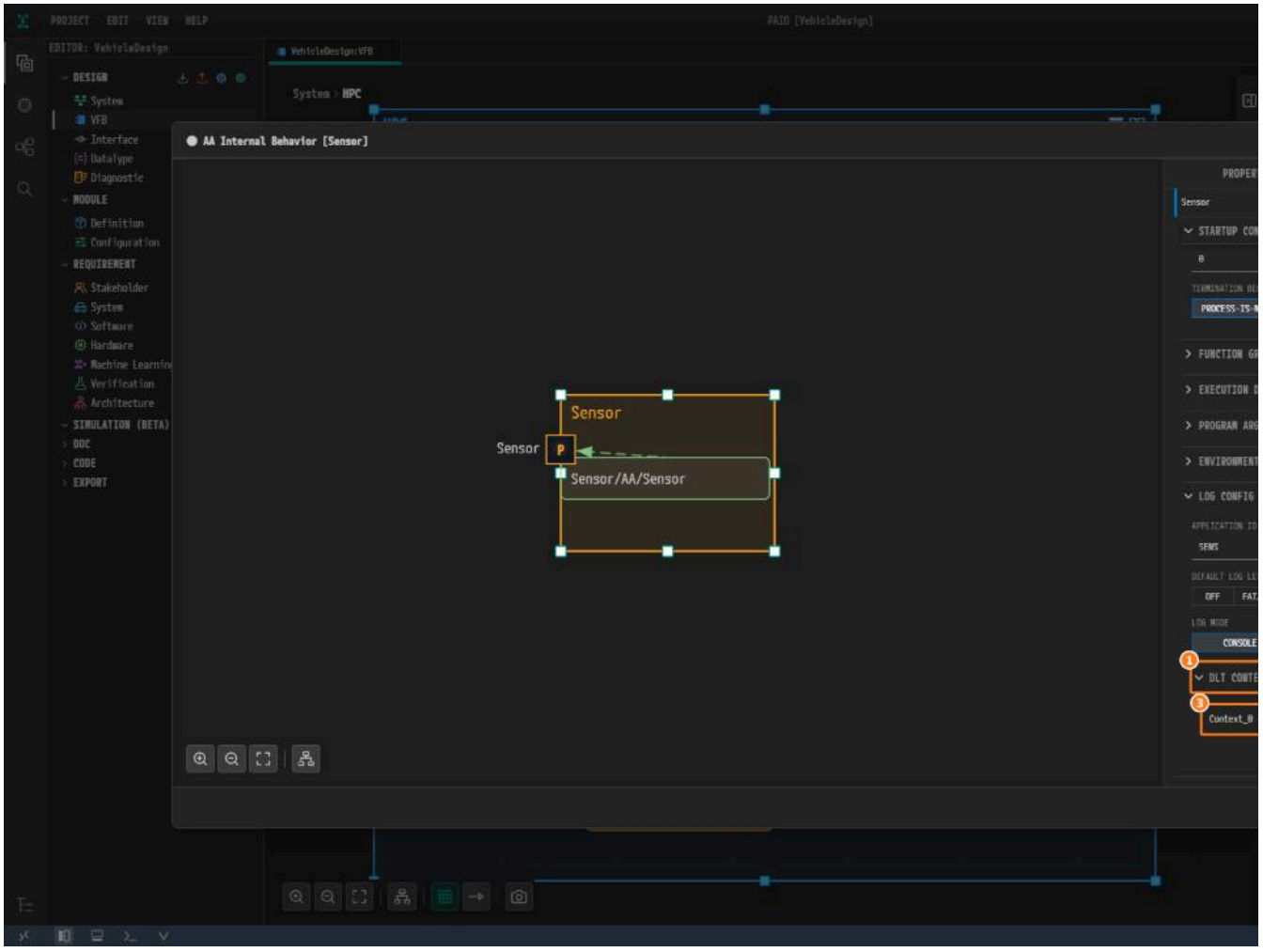


*Figure 12-4. ① **DLT CONTEXTS**, ② **+**, ③ `Context_0 : INFO`.*

3. Click the row header `Context_0 : INFO` to expand it.

4. Replace `Context_0` in **CONTEXT ID** with `SENS` and press `Tab`. The row header becomes `SENS : INFO`.

5. Type `Sensor context` in **CONTEXT DESCRIPTION** and press `Tab`.

6. Leave **LOG LEVEL** at `INFO`.

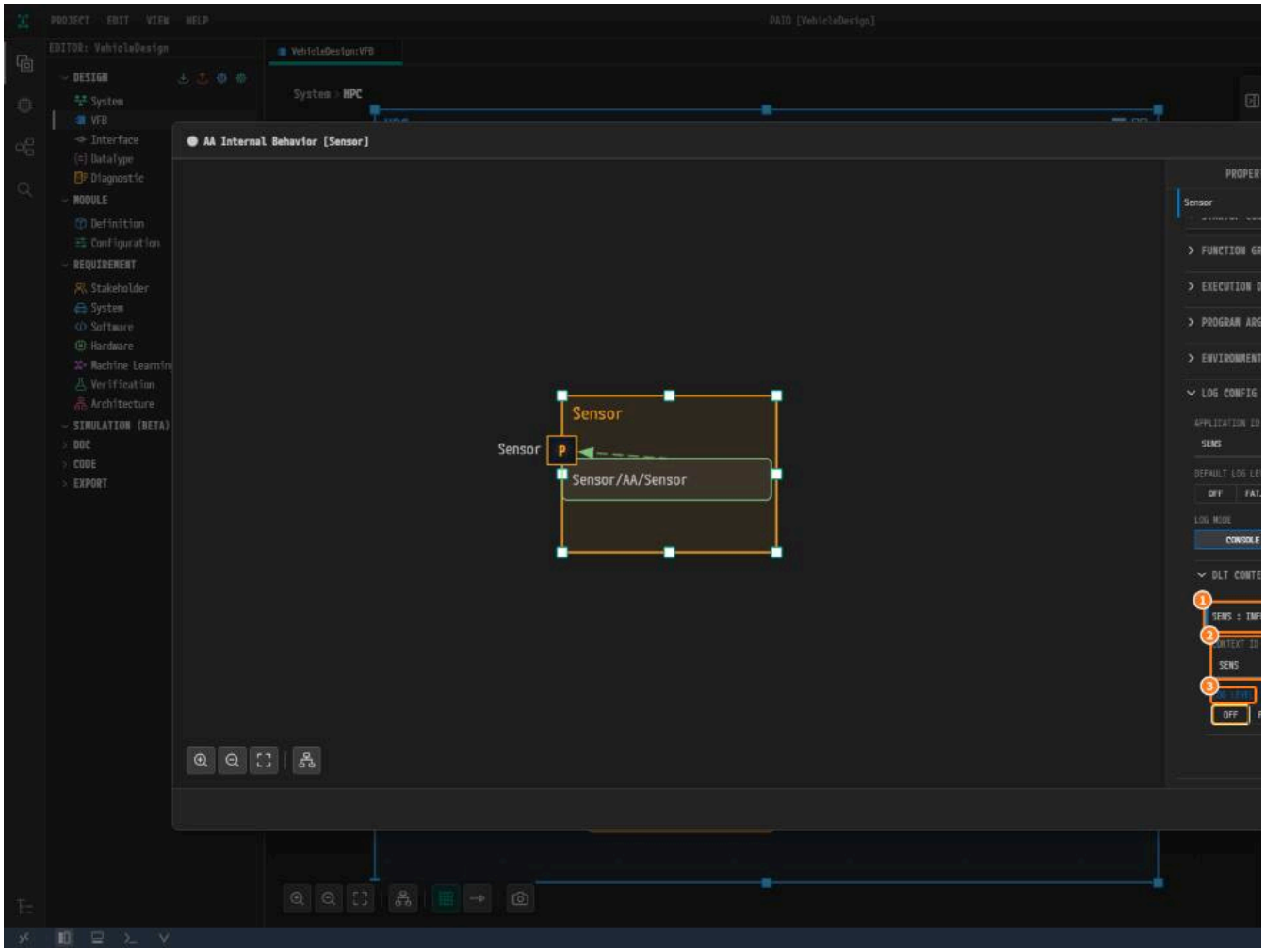


*Figure 12-5. ① `SENS : INFO`, ② **CONTEXT ID** `SENS`, ③ **LOG LEVEL** `INFO`.*

Saving and Closing

1. Press `Ctrl+S` inside the dialog.

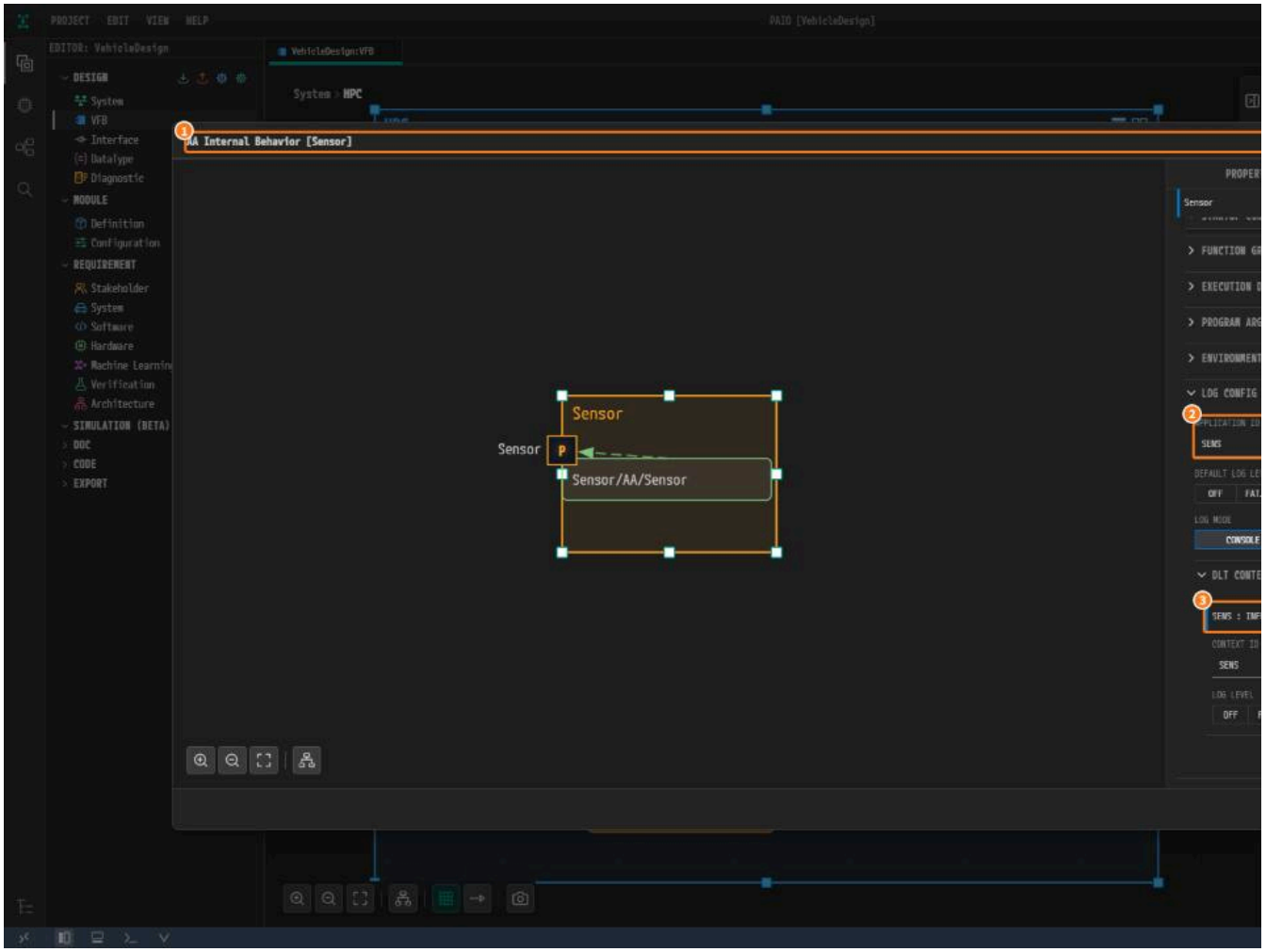


Figure 12-6. ① The title without the modified mark, ② `SENS`, ③ `SENS : INFO`.

2. Click **Close**.

Entering Fusion and Actuator the Same Way

Enter `Fusion` and `Actuator` the same way from the table, each with one context. Finish each with `Ctrl+S` and **Close**.

Checking the Machine's Log and Trace Table

1. Click the machine icon at the top right of the frame `HPC`. **Machine Configuration [HPC]** opens.
2. Click the **MODULE** tab, then **Log and Trace**.
3. Check that **ECU ID** is `ECU1` and the **DLT APPLICATION** table has fourteen rows: `SENS`, `FUSN`, `ACTA` with `VERBOSE`, and the eleven platform processes, w

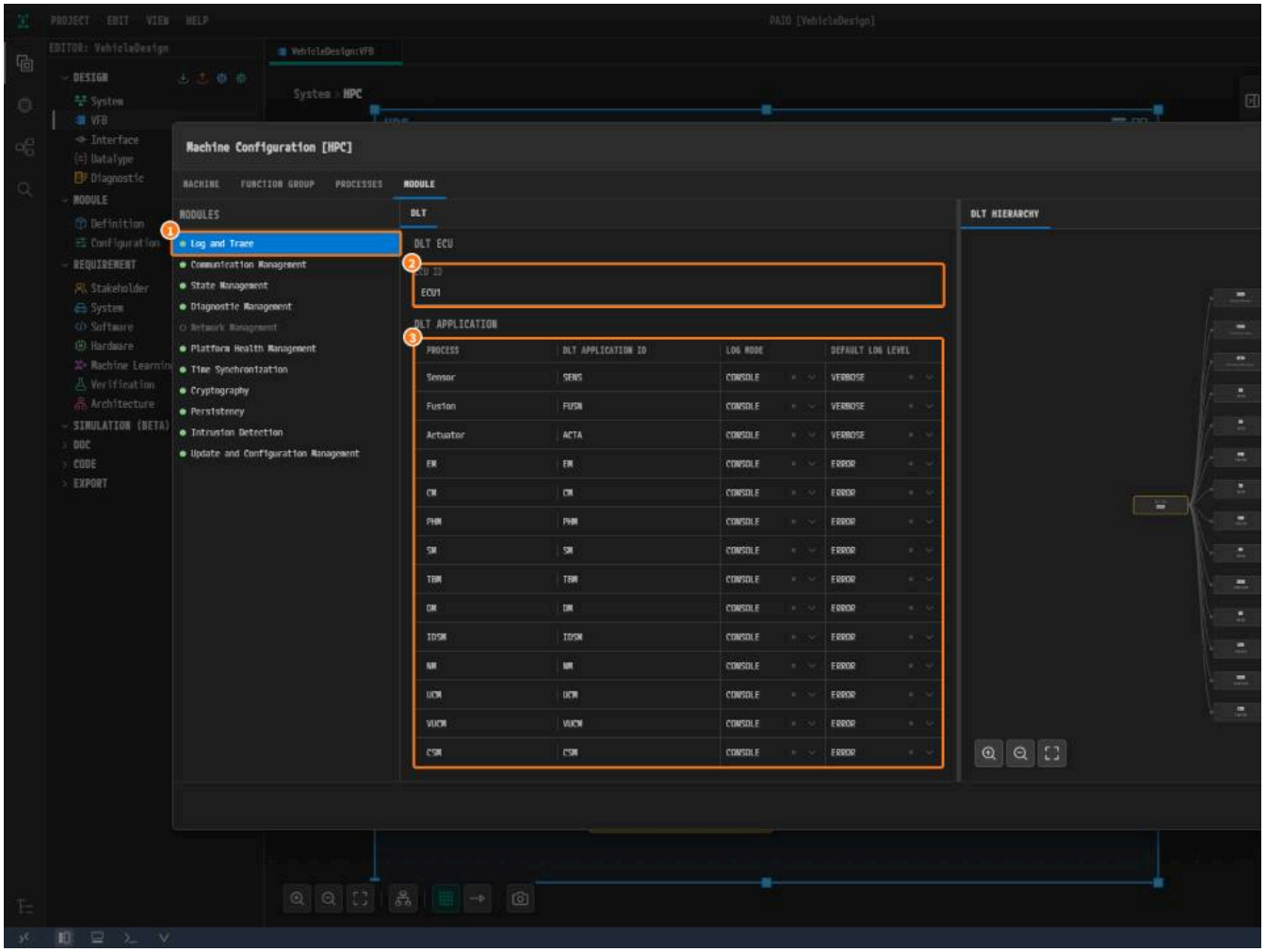


*Figure 12-7. ① **Log and Trace**, ② **ECU ID** `ECU1`, ③ the fourteen rows.*

4. Click the `Sensor` row. It shows `No LogAndTrace ports`, which is expected.

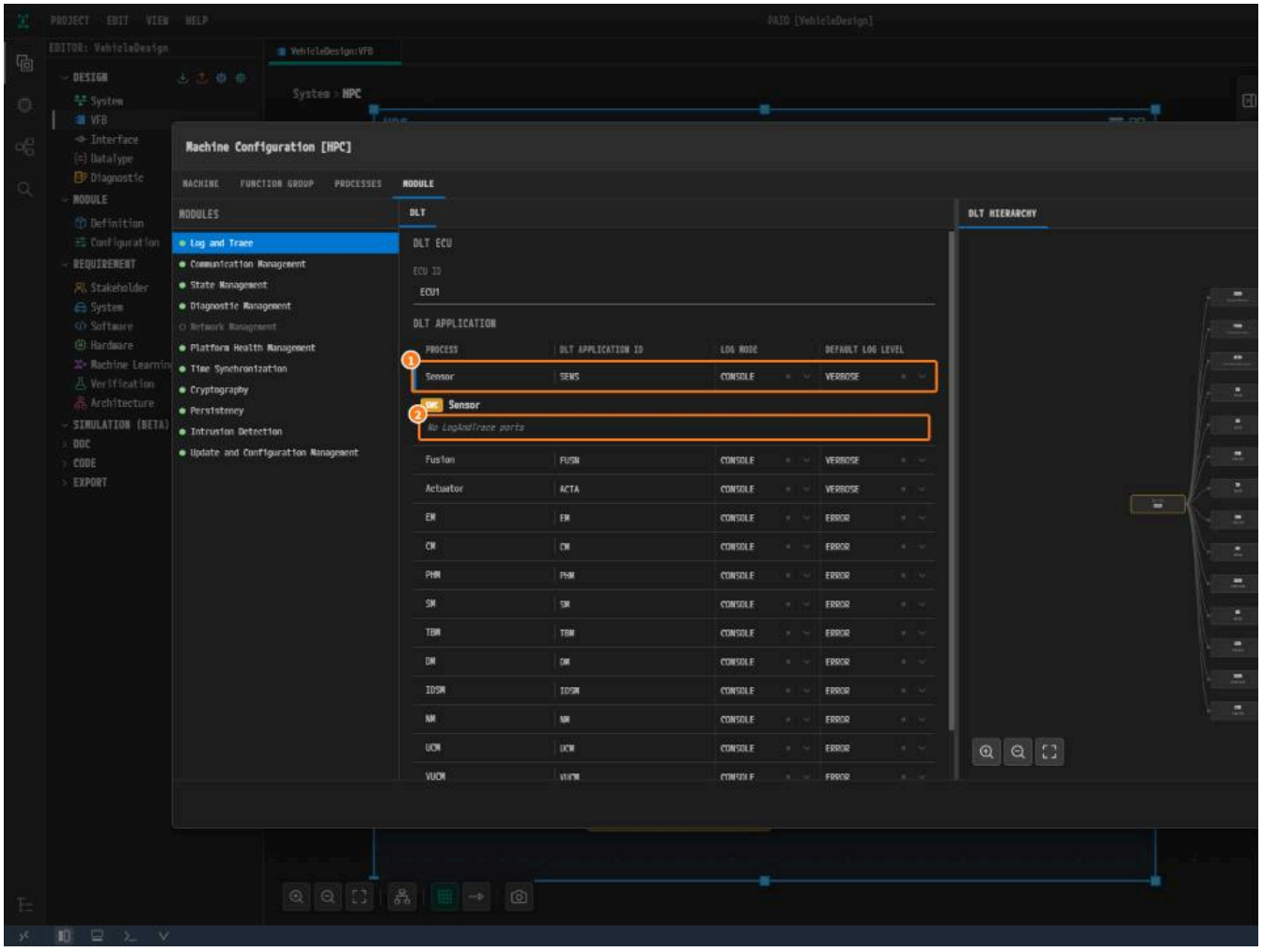


Figure 12-8. ① The `Sensor` row, ② `No LogAndTrace ports`.

5. Click **Close**.

Viewing the Validation Result

1. Look at the **VALIDATION** panel. If it is closed, click the **V** icon in the footer. It reads `No validation issues.`.

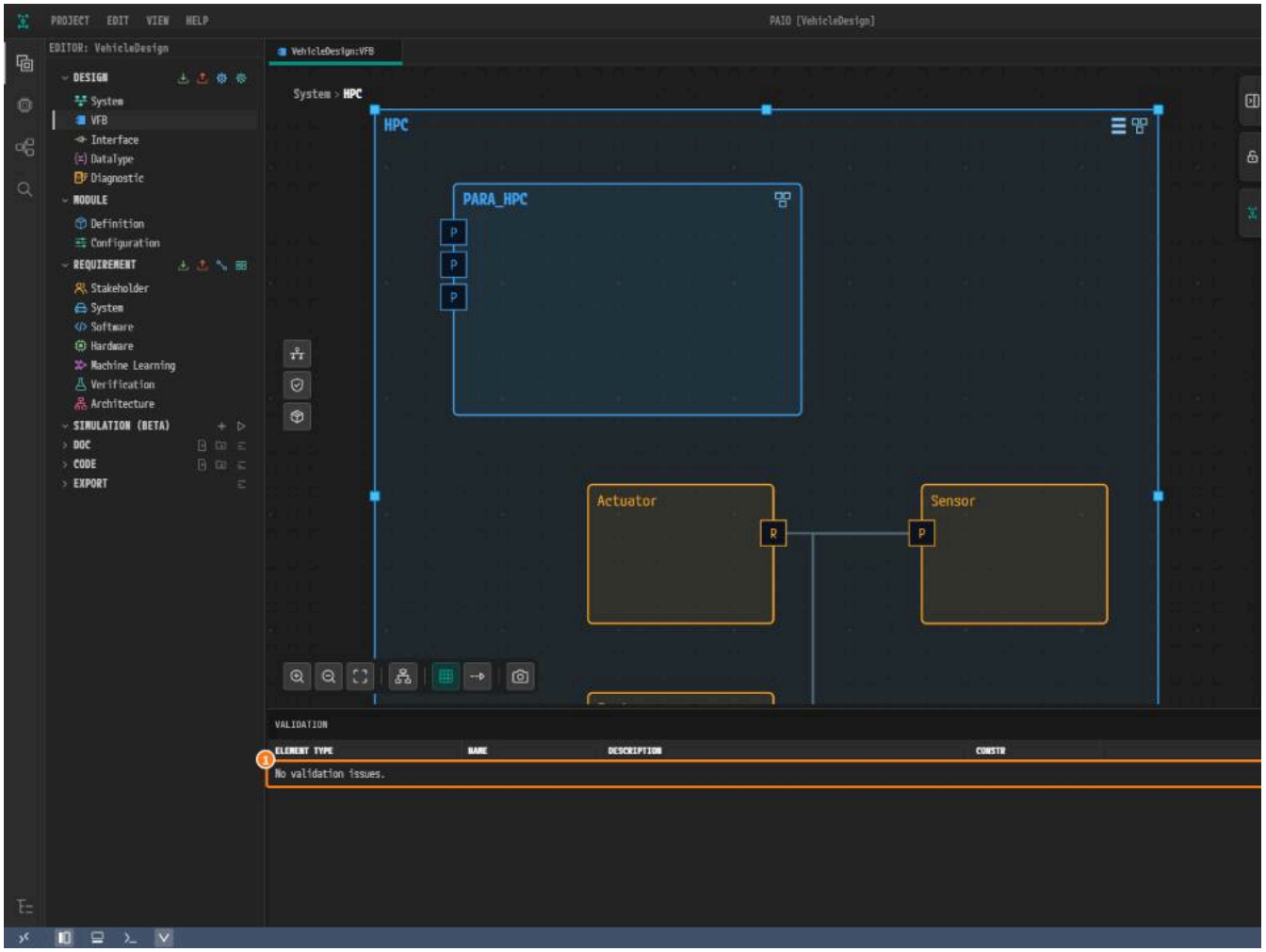


Figure 12-9. `No validation issues.`.

Why Is It Designed This Way?

- **Descriptions are required.** PAIO requires a description on every DLT application and context (`constr_5297`, `constr_5299`).
- **Unique IDs.** The application IDs must differ within the machine. When two collide, the **VALIDATION** panel reports both, and Validation and Code Genera
- **No logging ports.** This design has no **ara::log** port, so `No LogAndTrace ports` is expected.

Checklist

- [ ] `Sensor`, `Fusion`, `Actuator` have **APPLICATION ID** `SENS`, `FUSN`, `ACTA`, `VERBOSE`, and one context each.
- [ ] The **Log and Trace** table has fourteen rows.
- [ ] The **VALIDATION** panel shows `No validation issues.`.

Next Step

In Validation and Code Generation you validate the design and generate the code of `HPC`.

Validation and Code Generation

What You Will Do

In this chapter you check `No validation issues.` on the **VALIDATION** panel, generate the Adaptive Platform code of the machine `HPC` with **PARA Code G**

Before You Begin

- Start after finishing Logging and DLT.
- Every dialog is saved and closed, and no tab carries the modified mark `•`.

Where	Item	Value
PARA Code Generation window	**TARGET HPC**	`HPC`

Validating

1. Click the **V** icon in the footer to open the **VALIDATION** panel. It shows `No validation issues.`

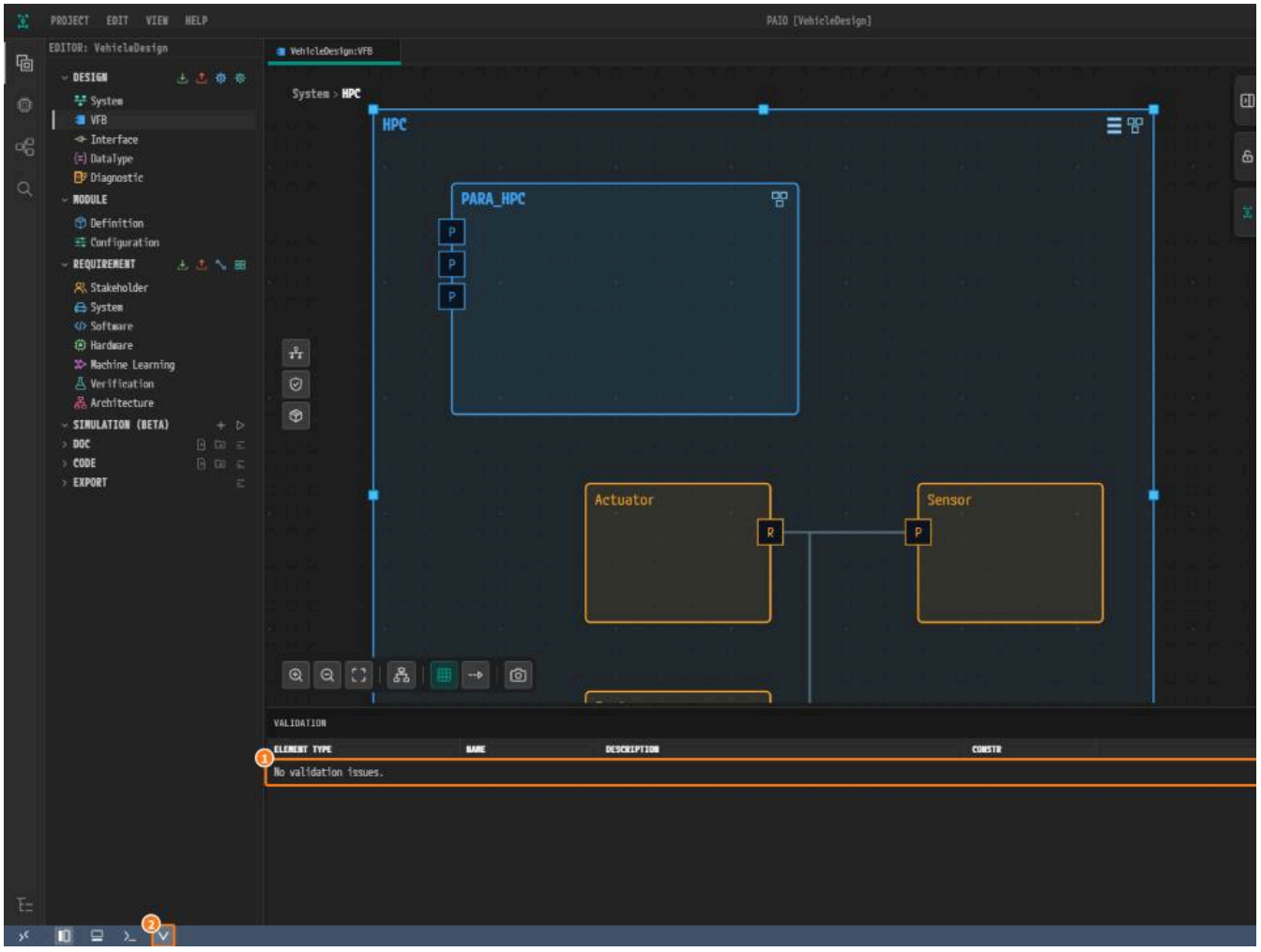


*Figure 13-1. ① `No validation issues.`, ② the **V** icon without a badge.*

> **Note.** If the panel lists rows, click a row: PAIO opens the editor or dialog that holds the value and marks the field. Fix the value, press `Ctrl+S` and look a messages.

Generating the Code

1. Close the panel with the **V** icon. On the **DESIGN** header, click the **PARA Code Generation** icon, the last of the four icons, right after **PRTE Code Generati**

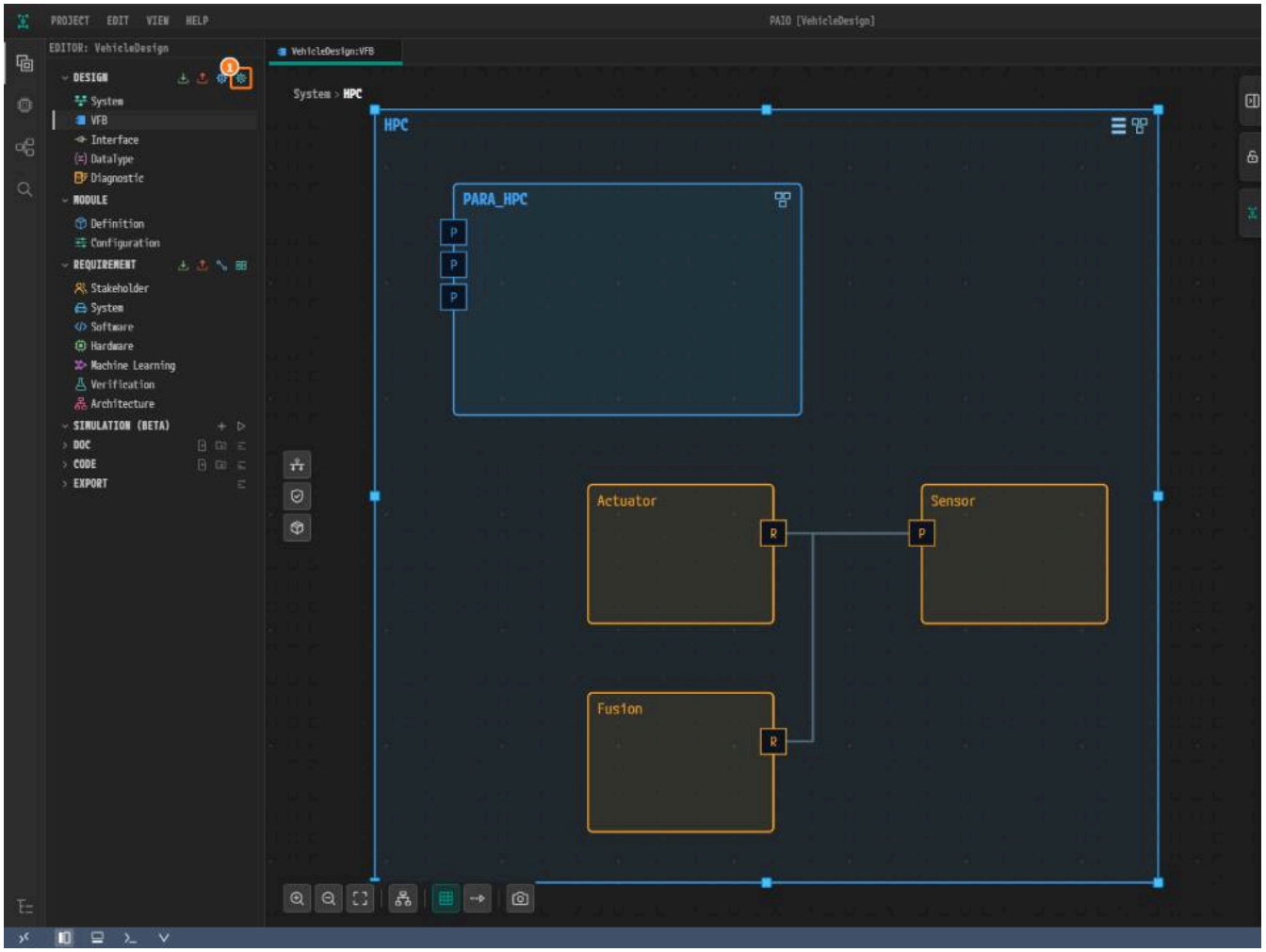


*Figure 13-2. ① **PARA Code Generation.***

2. Check that **TARGET HPC** reads `HPC` and click **Generate.** The line under the field reads `Output: code/<Machine>/`.

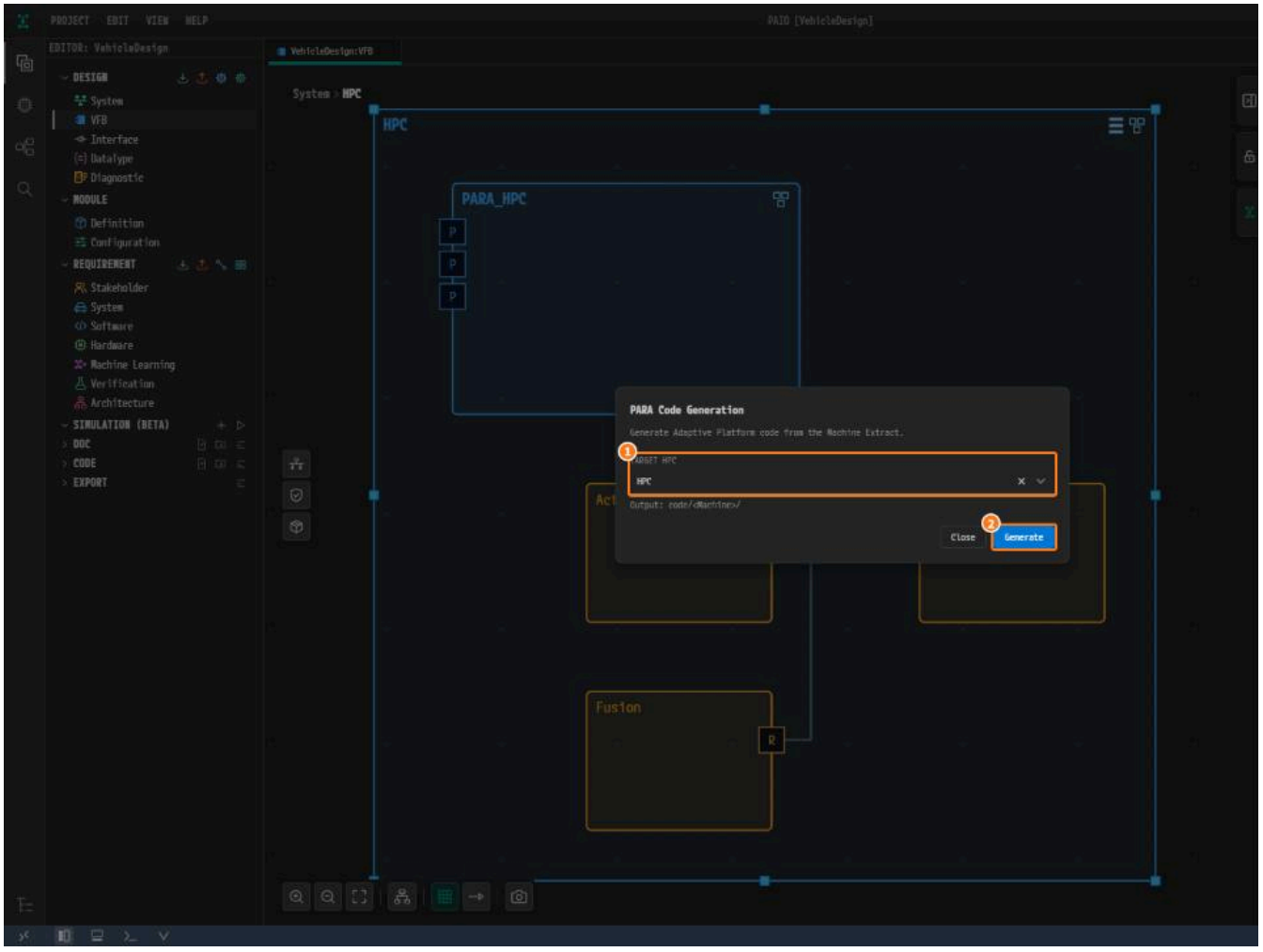


*Figure 13-3. ① **TARGET HPC**, ② **Generate**.*

3. When it ends, the window reads `COMPLETE` and `Generated 85 file(s) for HPC`. The table under it reads `7 warning(s), 1 info` and has no error.

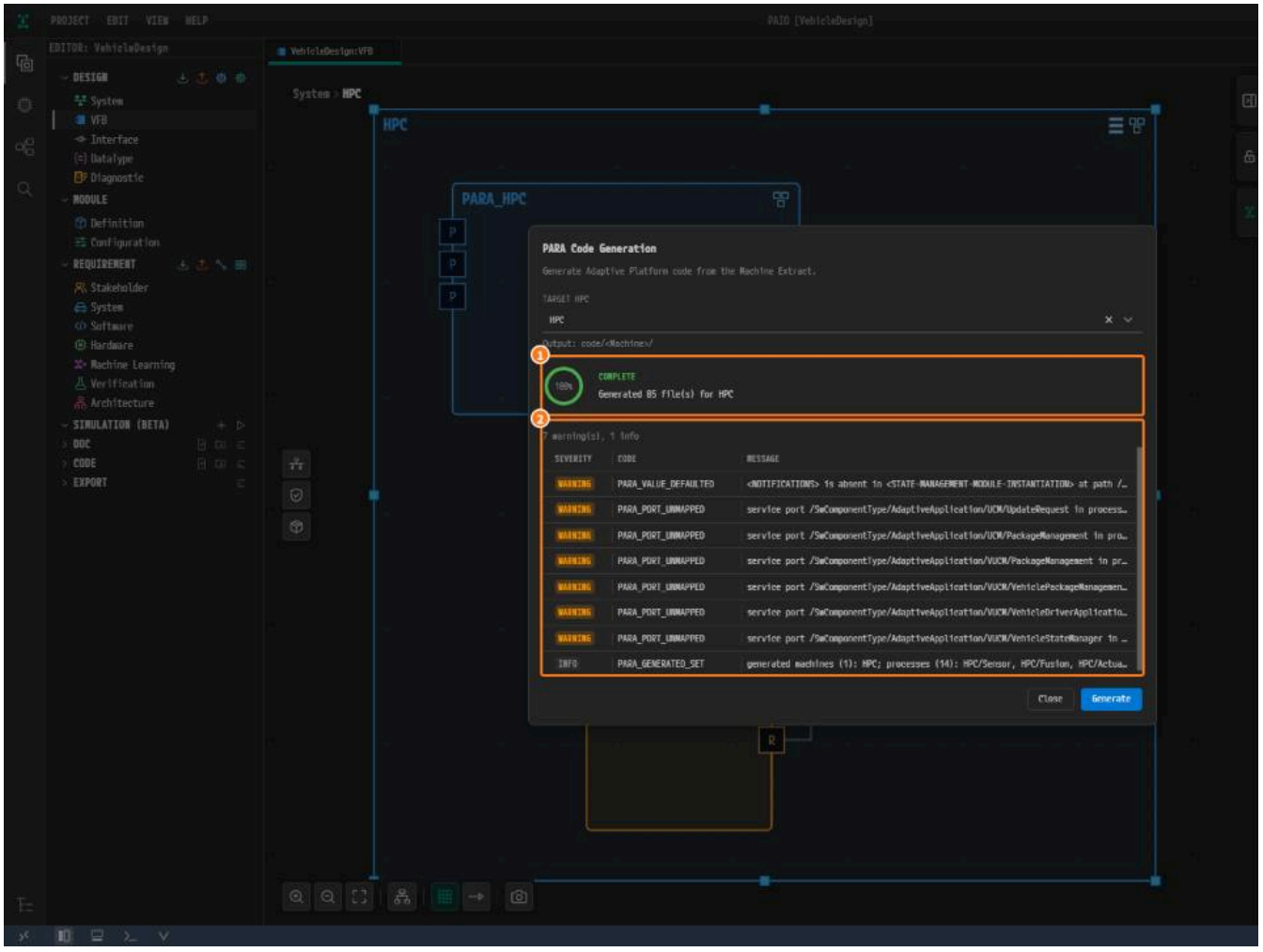


Figure 13-4. ① The result, ② the diagnostics table.

CODE	What it says
`PARA_VALUE_DEFAULTED`	The State Management module of `HPC` has no **NOTIFICATIONS**, so it defines no state machine and no SM manifest is written
`PARA_PORT_UNMAPPED` (six rows)	Two service ports of `UCM` and four of `VUCM` have no service instance mapping, so no code is generated for those ports
`PARA_GENERATED_SET` (info)	One machine `HPC` and fourteen processes were generated: your three and the eleven platform processes

The warnings name platform functions this design does not use. They do not stop the generation.

Looking at the Result

1. In the **EDITOR** view, open **CODE** and then `HPC`.
2. Open **EXPORT** and then `HPC`.

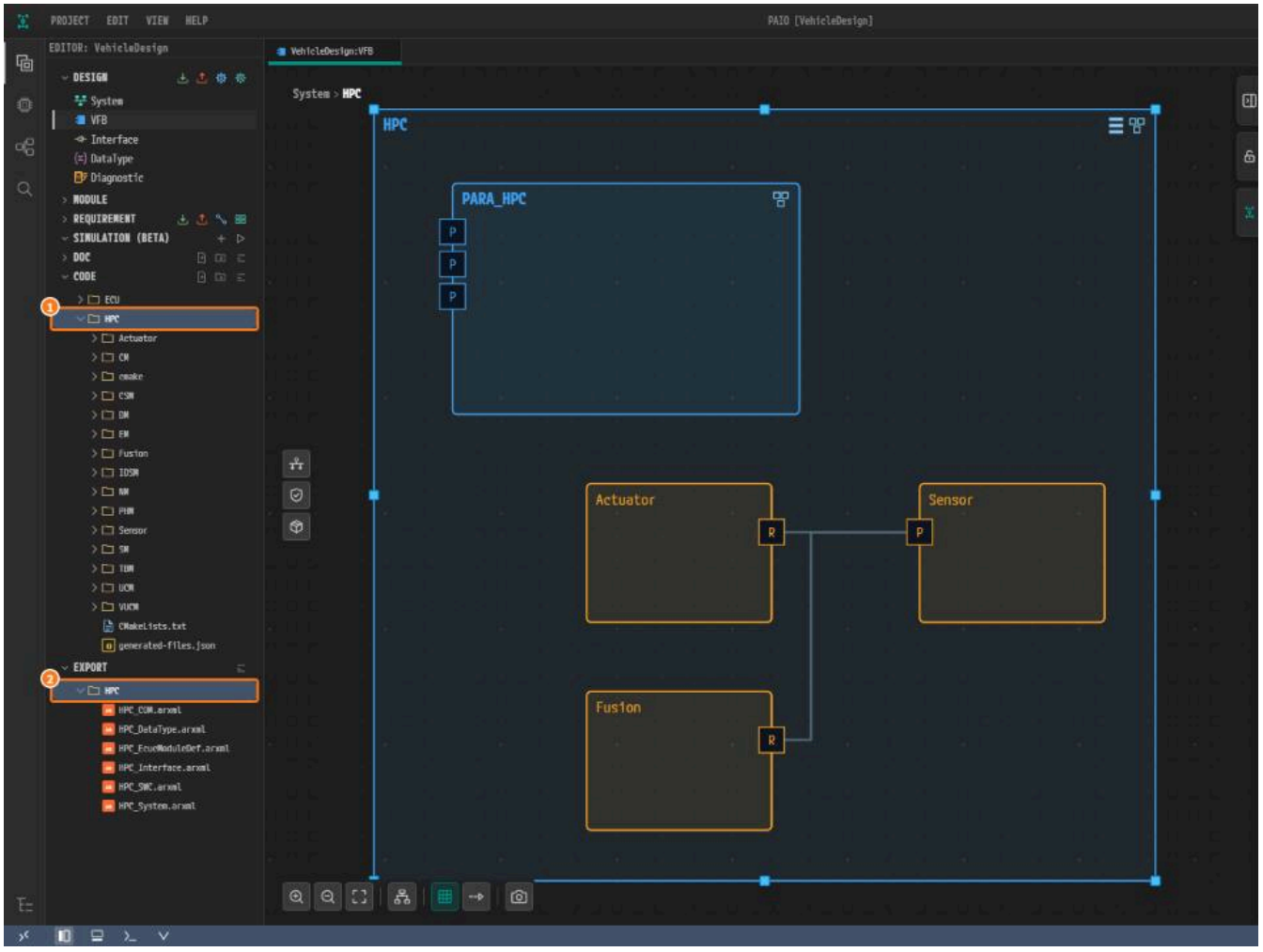


Figure 13-5. ① `code/HPC`, ② `export/HPC`.

The generation wrote two folders.

Folder	Content
`export/HPC/`	The Machine Extract of the node `HPC`: `HPC_COM.arxml`, `HPC_DataType.arxml`, `HPC_EcucModuleDef.arxml`, `HPC_Interface.arxml`, `HPC_SWC.arxml`, `HPC_Sy` needs
`code/HPC/`	The code of the machine `HPC`, generated from that extract

The folder `code/HPC/` holds:

Path	Content
`CMakeLists.txt`, `cmake/`	The build of the whole machine
`Sensor/`, `Fusion/`, `Actuator/`	One folder per application: `CMakeLists.txt`, `src/main.cpp` and the application class, the headers of the service `Ser` `manifest/exec/`
`EM/`	The platform process `EM`: its execution manifest, and the manifests of the machine `HPC` and of the function group `M`
`CM/`	The platform process `CM`: its execution manifest, and the SOME/IP manifests of the three service instances `Provider`
`PHM/`, `SM/`, `TBM/`, `DM/`, `IDSM/`, `NM/`, `UCM/`, `VUCM/`, `CSM/`	The other platform processes: an execution manifest each. `IDSM/` also holds the IdsM manifest, `UCM/` and `VUCM/` app
`generated-files.json`	The list of every other file the generator wrote

> **Note.** A second **Generate** rewrites the files that `generated-files.json` lists and keeps the files you added. The **ECU** folder under **CODE** is the code of

Why Is It Designed This Way?

- **Validation is the gate. Generate** validates the design first, and an error stops the generation before anything is written.
- **The extract is the input.** The generator reads only `export/HPC/`, the Machine Extract written in the same run, not the whole design.
- **One machine, one folder.** The output folder is named after the machine, `code/HPC/`, and every process of the machine has its own folder in it.
- **The tutorial stops here.** A design that validates and generates is the goal. Writing the application logic and building and running the machine in the I work.

Checklist

- [ ] The **VALIDATION** panel shows `No validation issues.`
- [ ] The **PARA Code Generation** window read `COMPLETE` and `Generated 85 file(s) for HPC`, with no error row.
- [ ] **CODE** lists `HPC` with the fourteen process folders, `cmake`, `CMakeLists.txt` and `generated-files.json`.
- [ ] **EXPORT** lists `HPC` with the six `HPC_*.arxml` files.

Next Step

The main part of the tutorial ends here. The Appendix collects every value of this design and common problems.

Appendix

What You Will Do

This appendix is reference material, not a chapter you follow along with. Use it to review your design or to find which value belongs to which chapter.

Before You Begin

- If a value here differs from a chapter, follow the chapter.
- The finished design is the state after Validation and Code Generation.

Checklist of the Finished Design

The Machine `HPC`

- [ ] The VFB root canvas has the composition `HPC [HPC]`, and clicking the machine icon at the right of its header opens the **Machine Configuration [HPC]**
- [ ] In the **MACHINE** tab, **OS** is `POSIX`, **PROCESSOR** is `CPU`, the **CORE** table has four rows from `Core0` / `0` to `Core3` / `3` and **TRUSTED PLATFORM** is **NO SUPPOI**
- [ ] The **FUNCTION GROUP** tab has one function group `MachineFG` with the modes `Off`, `Startup`, `Shutdown`, `Restart` and `Verify`.
- [ ] **ECU ID** under **MODULE > Log and Trace** is `ECU1`, and **MAX ACTION LIST DURATION** under **MODULE > State Management > ACTION ITEMS** is `1000` with the u
- [ ] The **PROCESSES** tab lists `Sensor`, `Fusion`, `Actuator` and the eleven platform processes, and the **EXECUTION DEPENDENCY** graph has `Running` arrows
- [ ] **Network Management** in the **MODULE** tab is disabled.

Topology

- [ ] The **CONTROLLER** of the connector `EthernetConnector` is `EthernetController` and its **CONNECTED CHANNEL** is `EthernetCluster / PhysicalChanr`
- [ ] **NETWORK ENDPOINT** is `192.168.0.11` / **FIXED**, with the multicast `224.0.0.1` / `30490`.
- [ ] **APPLICATION ENDPOINTS** has only the three cards `UDP EthernetConnector, 192.168.0.11:20001`, `:20002` and `:20003`.
- [ ] **SERVICE INSTANCES** has `Provider_Sensor` `UDP:20003`, `Actuator_Sensor` `UDP:20001` and `Fusion_Sensor` `UDP:20002`.
- [ ] In **Communication Cluster Configuration [EthernetCluster]**, the tabs **NM CLUSTER** and **TIME DOMAIN** have a hollow dot in front of their names.

Data Types, Interface and SOME/IP

- [ ] The list in the DataType editor holds `ByteArray [DYNAMIC_ARRAY]` and `String [STRING]` besides the 13 built-in types and the types of the CP tuto
- [ ] `Sensor` in the **SERVICE** tab of the Interface editor has the event `SensorData` `ByteArray` and the method `Calculation` (five arguments).
- [ ] The summary of the card `SOME/IP` `Sensor` in **SERVICE DEPLOYMENT** is `SERVICE ID = 0x000B` `VERSION 1.0` `EVENTS = 1` `METHODS = 1`, with the prov consumers `Actuator_Sensor` and `Fusion_Sensor` `0xFFFF (ANY)`.

Adaptive Applications

- [ ] The composition `HPC` contains `PARA_HPC`, `Sensor`, `Fusion` and `Actuator`, and each of the three applications has the same **NAME** and **AS PROTOTYPE**
- [ ] All three ports are named `Sensor` with **PORT INTERFACE** `Service: Sensor`, and the connectors `Assembly_Sensor_Fusion` and `Assembly_Sensor_Act`
- [ ] The **EXECUTION** tab of the **AA Internal Behavior** dialog of each of the three applications has one process card, and no row has the state `Off`.
- [ ] The mapping boxes of `Sensor`, `Fusion` and `Actuator` are `Sensor/AA/Sensor`, `Fusion/AA/Sensor` and `Actuator/AA/Sensor`, and **CLIENT ID** is `0x2` fo
- [ ] The **APPLICATION ID** values in **LOG CONFIG** are `SENS`, `FUSN` and `ACTA`, with the DLT contexts `SENS : INFO`, `FUSN : INFO` and `ACTA : INFO`.

Validation and Code Generation

- [ ] The **VALIDATION** panel shows `No validation issues.`, and there is no badge next to the panel title or on the footer **V** icon.
- [ ] **PARA Code Generation** for `HPC` ends with `COMPLETE` and no error, `export/HPC` holds the six `HPC_*.arxml` files and `code/HPC` the generated code.
- [ ] No tab or dialog carries the modified mark `•`.

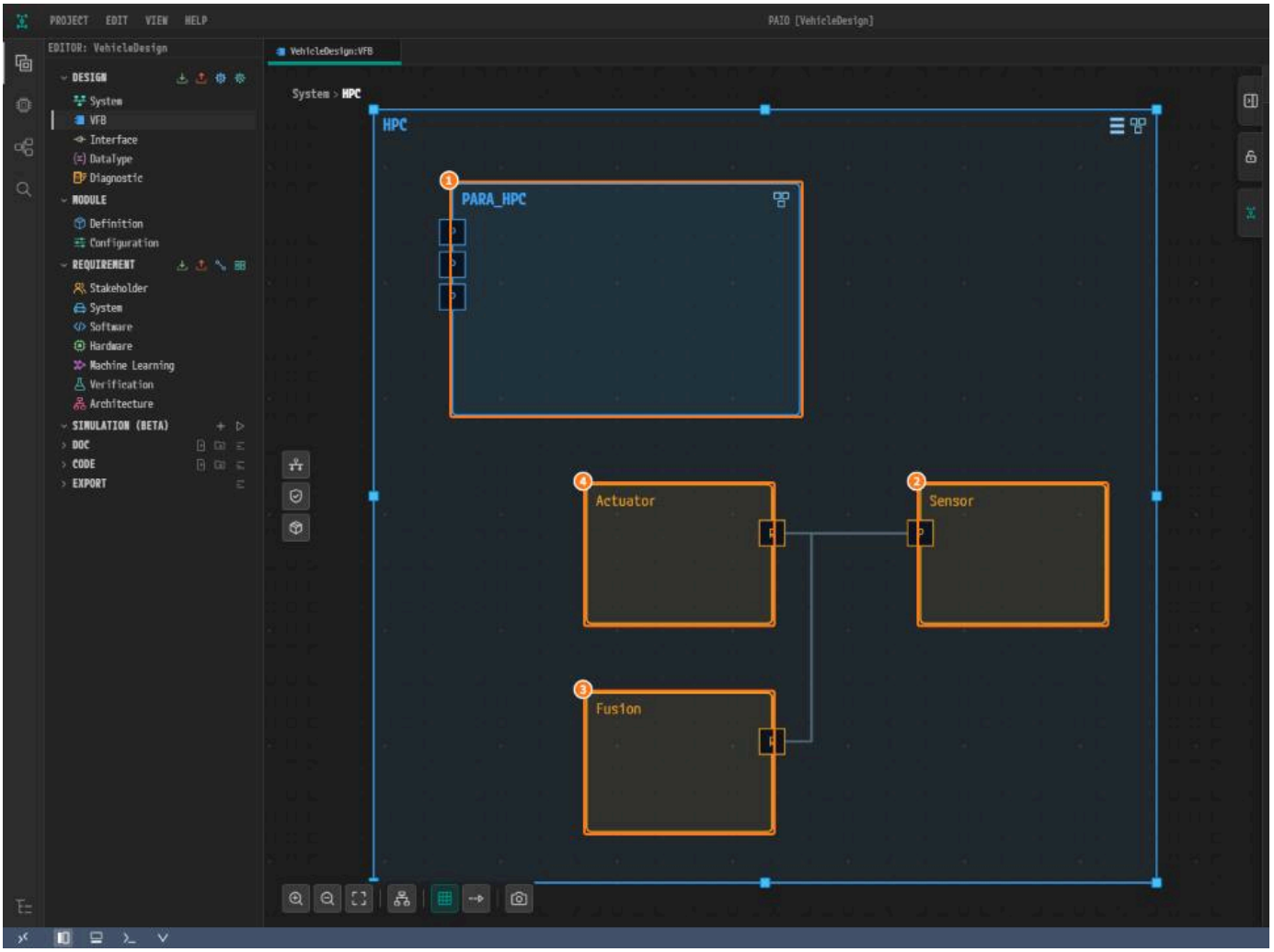


*Figure 14-1. The finished composition `HPC` (Figure 9-9 in PARA Setup). ① `PARA_HPC`, ② `Sensor`, ③ `Fusion`, ④ `Actuator`. Positions depend on **Auto Layou***

All Value Tables

Project and Machine (Chapters 1 and 2)

Item	Value	Chapter
Project name	`VehicleDesign`	AUTOSAR System tutorial
HPC node name in the System editor	`HPC`	AUTOSAR System tutorial
Composition **NAME** / **AS PROTOTYPE**	`HPC` / `HPC`	1
Mapping	Composition `HPC` > **ECU/HPC Mapping** > **HPC [HPC]**	1
OS	`POSIX`	2
TRUSTED PLATFORM	**NO SUPPORT** (default)	1
PROCESSOR	`CPU` (default)	1
CORE / **CORE ID**	`Core0` / `0`, `Core1` / `1`, `Core2` / `2`, `Core3` / `3`	2
Function group	`MachineFG` (default, unchanged)	1
GROUP NAME / **INITIAL MODE**	`MachineState` / `Off`	1
Mode declarations	`Off`, `Startup`, `Shutdown`, `Restart`, `Verify`	1
Log and Trace > **ECU ID**	`ECU1`	2
State Management > **MAX ACTION LIST DURATION**	`1000` `s`	2

Topology (Chapter 3)

The rows marked *System* were entered in the AUTOSAR System Tutorial; chapter 3 adds only the multicast port and the application endpoints.

Element	Item	Value
Ethernet controller	**NAME**	`EthernetController` (*System*)
Ethernet connector	**NAME** / **CONTROLLER**	`EthernetConnector` / `EthernetController` (*System*)
Communication cluster	**NAME**	`EthernetCluster` (*System*)
Physical channel	**CHANNEL NAME** / **VLAN ID**	`PhysicalChannel` / `13` (*System*)
Network endpoint	**UNICAST IPV4 ADDRESS** / **ADDRESS SOURCE**	`192.168.0.11` / **FIXED** (*System*)
Multicast	**MULTICAST IPV4 ADDRESS** / **MULTICAST PORT**	`224.0.0.1` (*System*) / `30490`
Application endpoints	**TYPE UDP**, **PORT NUMBER**	`20001`, `20002`, `20003`
Endpoint names (automatic)	—	`HPC_EthernetConnector_UDP_20001`, `HPC_EthernetConnector_UDP`
Communication Cluster Configuration [EthernetCluster]	**NM CLUSTER**, **TIME DOMAIN** tabs	Disabled (default, left as is)

Data Types (Chapter 4)

Type	CATEGORY	ELEMENT TYPE	MAX SIZE	C++ NAMESPACE
`ByteArray`	**DYNAMIC_ARRAY**	`uint8`	`1`	`aa::type`
`String`	**STRING**	—	`256`	`aa::type`

The 11 primitives and the pointer types `VoidPtr` and `ConstVoidPtr` are built in; you do not enter them.

Service Interface (Chapter 5)

Element	Value
Service interface	`Sensor`, **C++ NAMESPACE** `sensor::service`, **MAJOR VERSION** `1`, **MINOR VERSION** `0`
Event	`SensorData`, **TYPE** `ByteArray`
Method	`Calculation`, **METHOD TYPE REQUEST AND RESPONSE**
Arguments 1 to 4	**IN** `float32` `front`, `rear`, `left`, `right`
Argument 5	**OUT** `float32` `result`
Fields	None

SOME/IP Deployment and Service Discovery (Chapter 6)

Element	Value
Deployment	`Sensor`, **SERVICE ID** `11` (`0x000B`), **MAJOR VERSION** `1` / **MINOR VERSION** `0`
Event deployment	`SensorData`, **EVENT ID** `1`, **TRANSPORT UDP**
Method deployment	`Calculation`, **METHOD ID** `1`, **TRANSPORT UDP**
Event group	`Eventgroup_SensorData`, **EVENTGROUP ID** `1`, **EVENTS** `SensorData`
Provider	`Provider_Sensor`, **INSTANCE ID** `1`
Consumers	`Actuator_Sensor`, `Fusion_Sensor`, **REQUIRED INSTANCE ID** `ANY` (model value `65535`)
Provider event group	**MULTICAST THRESHOLD UNICAST** (`0`)

SD field	Default	Entered value	Provider (SD SERVER CONFIG)	Consumers (SD CLIENT CONFIG)
INITIAL DELAY MIN	`10 ms`	`10 us`	Entered	Entered
INITIAL DELAY MAX	`100 ms`	`100 us`	Entered	Entered
REPETITION BASE DELAY	`200 ms`	`200 us`	Entered	Entered
REPETITION MAX	`3`	Unchanged	Default	Default
OFFER CYCLIC DELAY	`500 ms`	`1 ms`	Entered	No field
TTL	`0xFFFFFF`	Unchanged	Default	Default
REQUEST RESPONSE DELAY MIN / MAX	`0 s` / `0 s`	Unchanged	Default	No field

Connector Binding (Chapter 7)

Instance	Role	UDP PORT	TCP PORT
`Provider_Sensor`	Provider (**P**)	`20003`	Left empty
`Actuator_Sensor`	Consumer (**C**)	`20001`	Left empty
`Fusion_Sensor`	Consumer (**C**)	`20002`	Left empty

All three instances are bound to the connector `EthernetConnector`, and **IP ADDRESS** is filled in automatically with `192.168.0.11`.

Adaptive Applications, Ports and Connectors (Chapter 8)

Adaptive Application	NAME = AS PROTOTYPE	Port	Port NAME	PORT INTERFACE
Sensor	`Sensor`	**PPort**	`Sensor`	`Service: Sensor`
Fusion	`Fusion`	**RPort**	`Sensor`	`Service: Sensor`
Actuator	`Actuator`	**RPort**	`Sensor`	`Service: Sensor`

Connector	Provider side	Requester side
`Assembly_Sensor_Fusion`	`Sensor` / `Sensor`	`Fusion` / `Sensor`
`Assembly_Sensor_Actuator`	`Sensor` / `Sensor`	`Actuator` / `Sensor`

PARA Setup (Chapter 9)

Item	Value
Import	**PARA Platform Import** in **Machine Configuration [HPC]**
Conflicts **ECU ID**, **Max Action List Duration**	**Keep Project** (`ECU1`, `1000`)
Imported composition	`PARA_HPC`
Platform processes	`EM`, `CM`, `PHM`, `SM`, `TBM`, `DM`, `IDSM`, `NM`, `UCM`, `VUCM`, `CSM`
MODULE > Network Management	Disabled

Executables and Processes (Chapter 10)

All three Executables keep their prefilled values.

Label	Value
NAME	The Adaptive Application name
ROOT SWC PROTOTYPE	`AA`
CATEGORY	`APPLICATION` (`APPLICATION_LEVEL` in the file)
BUILD TYPE	Empty
REPORTING BEHAVIOR	`REPORTS-EXECUTION-STATE`
VERSION	`1.0.0`
MINIMUM TIMER GRANULARITY	Empty

Process NAME	ENTER TIMEOUT / EXIT TIMEOUT	FUNCTION GROUP STATE	EXECUTION DEPENDENCY
`Sensor`	`5` / `5` `s`	`MachineFG`, `Startup`	`CM`, `Running`
`Fusion`	`5` / `5` `s`	`MachineFG`, `Startup`	`CM`, `Running`
`Actuator`	`5` / `5` `s`	`MachineFG`, `Startup`	`CM`, `Running`

All other process values keep their defaults. The values of the platform processes stay as PARA gives them.

Service Instance Mapping (Chapter 11)

	`Sensor`	`Fusion`	`Actuator`
PORT	`Sensor` (`P`)	`Sensor` (`R`)	`Sensor` (`R`)
SERVICE INSTANCE	`Provider_Sensor (SOME/IP)`	`Fusion_Sensor (SOME/IP)`	`Actuator_Sensor (SOME/IP)`
CLIENT ID	No field	`2` → `0x2`	`1` → `0x1`
INSTANCE SPECIFIER	`Sensor/AA/Sensor`	`Fusion/AA/Sensor`	`Actuator/AA/Sensor`
PROCESS (automatic)	`Sensor`	`Fusion`	`Actuator`
Read-back **INSTANCE ID**	`0x0001`	`0xFFFF`	`0xFFFF`
Read-back **ENDPOINT**	`192.168.0.11:20003`	`192.168.0.11:20002`	`192.168.0.11:20001`
Saved name (automatic)	`Sensor_AA_Sensor`	`Fusion_AA_Sensor`	`Actuator_AA_Sensor`

The ports of the platform applications are not mapped. The read-back **SERVICE ID** and **VERSION** are `0x000B` and `1.0` for all three.

Logging and DLT (Chapter 12)

Process	APPLICATION ID	DEFAULT LOG LEVEL	LOG MODE	DLT CONTEXTS (CONTEXT ID : LOG LEVEL)
`Sensor`	`SENS`	`VERBOSE`	`CONSOLE`	`SENS : INFO`
`Fusion`	`FUSN`	`VERBOSE`	`CONSOLE`	`FUSN : INFO`
`Actuator`	`ACTA`	`VERBOSE`	`CONSOLE`	`ACTA : INFO`

APPLICATION DESCRIPTION is `Sensor application`, `Fusion application` and `Actuator application`, **CONTEXT DESCRIPTION** `Sensor context`, `Fusion` settings of the platform processes stay as PARA gives them.

Project File Layout

The files this tutorial creates or changes, with the chapters that write them. Files of the System and CP tutorials are left out.

```
VehicleDesign/
├── VehicleDesign.io.prj
├── design/
│   ├── DATATYPE.io.model                    Chapters 4, 9
│   ├── INTERFACE.io.model                   Chapters 5, 6, 9
│   ├── IVC.io.model                         Chapters 3, 7
│   ├── SERVICE.io.model                     Chapter 6
│   ├── SIGNAL.io.model                      Chapter 6
│   ├── SYSTEM.io.model                      Chapter 1
│   ├── VFB.io.model                         Chapters 1, 8, 9
│   ├── Module/
│   │   └── Machine/
│   │       └── HPC.machine.io.model         Chapters 1, 2, 9
│   └── SWC/
│       ├── AdaptiveApplication/
│       │   ├── Actuator.swc.io.model        Chapters 8, 10 to 12
│       │   ├── Fusion.swc.io.model          Chapters 8, 10 to 12
│       │   ├── Sensor.swc.io.model          Chapters 8, 10 to 12
│       │   └── HPC/                         Chapter 9  the eleven platform applications
│       ├── Composition/
│       │   ├── HPC.swc.io.model             Chapters 1, 8, 9
│       │   ├── PARA_HPC.swc.io.model        Chapter 9
│       │   └── System.swc.io.model          Chapter 1
│       └── Layout/
│           ├── HPC.vfb.layout               Chapters 8, 9
│           ├── PARA_HPC.vfb.layout          Chapter 9
│           └── System.vfb.layout            Chapter 1
├── export/
│   └── HPC/                                 Chapter 13
│       ├── HPC_COM.arxml
│       ├── HPC_DataType.arxml
│       ├── HPC_EcucModuleDef.arxml
│       ├── HPC_Interface.arxml
│       ├── HPC_SWC.arxml
│       └── HPC_System.arxml
└── code/
    └── HPC/                                 Chapter 13
        ├── CMakeLists.txt
        ├── generated-files.json
        ├── cmake/
        ├── Actuator/  Fusion/  Sensor/
        ├── CM/  CSM/  DM/  EM/  IDSM/  NM/  PHM/  SM/  TBM/  UCM/  VUCM/
        └── ...
```

- `HPC.machine.io.model` takes its name from the HPC node `HPC` in the System editor.
- A `*.vfb.layout` file holds only node and port positions. It is not exported.
- The folders `doc`, `workflow` and `import` are not used. `code/ECU` is the code of the CP tutorial.
- `SIGNAL.io.model`, `Sensor_SR` and `Sensor_CS` are derived automatically from the SOME/IP deployment in SOME/IP Deployment and Service Instanc

Common Problems

Symptom or validation message	Cause	Fix
`State-Management module "SM" of machine "HPC" is missing maxActionListDuration.` (`constr_10641`)	Expected at the end of chapter 1	Enter `1000` in **MAX ACTION LIST DURATION** and press (the dialog
`The DLT application-id "SENS" truncates to "SENS" and collides with "SENS" (Sensor).` and its paired row `… (Fusion).`	Two processes have the same **APPLICATION ID**	Make the **APPLICATION ID** values differ (`FUSN` and so c press `Ctrl+S`
`The process "Sensor" declares no functionGroupState; at least one is required.`	No **FUNCTION GROUP STATE** row	Add a row with **+**, choose `Startup`, press `Ctrl+S`
`"Sensor" of "Sensor": referenced port interface "SensorIf" not found.`	The port points to a missing interface	Choose `Service: Sensor` in **PORT INTERFACE**, press (and reopen the VFB tab
Clicking the machine icon does not open the **Machine Configuration** dialog	The VFB tab was open during the mapping	Close the VFB tab and open **VFB** again
Values entered in a dialog are not saved	Only **Close** was pressed	Press `Ctrl+S` in the dialog, then **Close**
Several SD fields were changed but only the last one remains	Consecutive changes reset the earlier fields	After each value press `Tab` and click `Sensor` in **INTEF** again
The execution dependency of `Fusion` or `Actuator` is entered as `Sensor`, `Running`	A new row defaults to the first process	Choose `CM` in **PROCESS**
`CM` is missing from the **PROCESS** list of **EXECUTION DEPENDENCY**	The PARA platform is not imported	Run **PARA Platform Import** and save
Code generation stops with the error `PARA_REQUIRED_VALUE_MISSING` (`<NM-NETWORK-HANDLE>`)	The **Network Management** module is enabled	Disable **Network Management** in the **MODULE** tab and `Ctrl+S`
A prototype name stays `Composition_0` or `AdaptiveApp_0_0`	**AS PROTOTYPE** does not follow **NAME**	Enter the name in both fields, then check the connec (next row)
The connectors are named `Assembly_Sensor_Sensor_Fusion_Sensor` and `Assembly_Sensor_Sensor_Actuator_Sensor`	Default connector names	Rename them to `Assembly_Sensor_Fusion` and `Assembly_Sensor_Actuator` in **ASSEMBLY**, press `Ctr`

Checklist

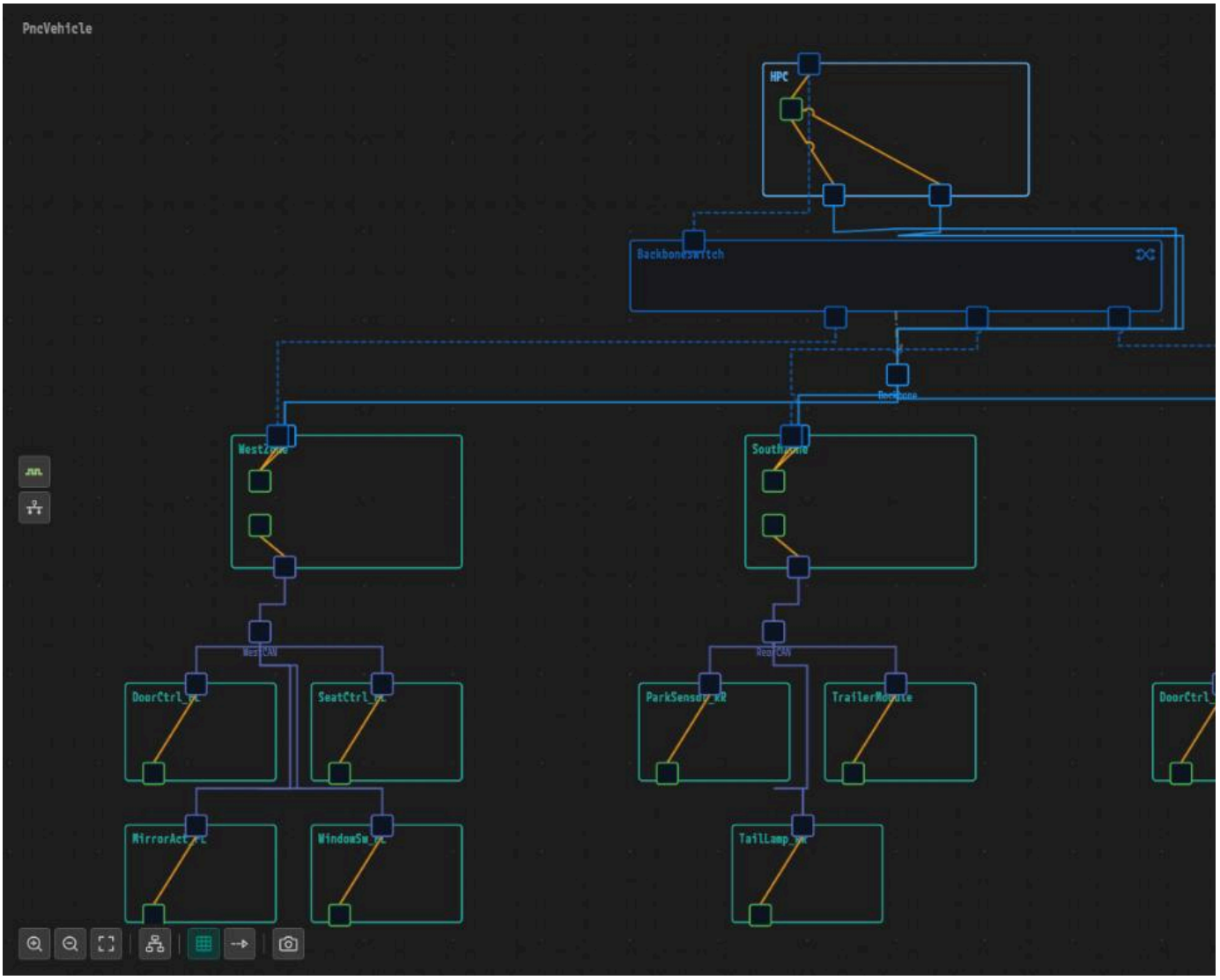
- [ ] Every item of "Checklist of the Finished Design" matches.
- [ ] The values in "All Value Tables" match the values on the screens of your design.
- [ ] The files under `design/` match "Project File Layout".

Next Step

To change the design, fix the value in its chapter and repeat Validation and Code Generation; to start over, return to the AUTOSAR AP Tutorial.

PNC Design Tutorial

This tutorial designs three PNCs (Partial Network Clusters) for a zonal vehicle in PAIO, from the topology to the ARXML export.



The vehicle you build: the HPC (AUTOSAR Adaptive Platform) with its backbone switch at the top, three zone ECUs in the middle, and ten end ECUs on thre

What You Build

Three PNCs express one vehicle requirement: "when a door opens, wake only the Comfort network; the park assist network comes alive only in reverse ge attached."

PNC	Identifier	Wakeup trigger
`PNC_Comfort`	`17`	Door unlock, door open
`PNC_ParkAssist`	`18`	Reverse gear. The HPC requests it.
`PNC_Trailer`	`19`	Trailer attached

What You Need

- PAIO **0.6.0** (AUTOSAR R25-11).
- No prepared project. Chapter 2 starts from a PAIO window with no project open.
- A screen of 1920×1080 or larger. The PNC dialog and the Machine Configuration dialog are wide.
- An active license and a server connection, only for Chapter 12 (ARXML export).

- A rough idea of what a PNC is. Each chapter shows what to enter on which screen.

Chapters

Chapter	What you do
1. Example and Design Goals	Derive three PNCs from the vehicle scenario and identify the layers a PNC spans
2. Topology Design	Create the project and place the nodes, clusters, backbone switch, and coupling ports
3. PNC Definitions and Connector PNC Properties	Define the three PNCs, enter gateway, wakeup, and filter mask values on the connectors, and fill in the NM cluster and NM node values
4. Signals, PDUs, Frames, and I-PDU Groups	Create and map the frames, I-PDUs, signals, and NM-PDUs, then create the eight per-PNC Rx/Tx I-PDU Groups
5. Interfaces and SOME/IP Services	Create the service interfaces and the SOME/IP deployment, and bind the instances to the connectors
6. VFB and VFC (PortGroup)	Create one composition and its atomic SWCs per node, connect them, and group the ports of each function into a PortGroup (= VFC)
7. PNC Dialog — Per-PNC Bindings	Bind the VFCs, I-PDU Groups, physical channels, service instances, and wakeup frames to each PNC
8. Reviewing PDU GROUPS, MATRIX, and NM VECTOR	Cross-check the design in the three viewer tabs
9. AP Machine — NM Handles and FunctionGroups	Add three FunctionGroups and the State Management values to the HPC, and map three NM network handles to FunctionGroup states
10. PARA Platform Import and PNC Request Ports	Import PARA and give the application AAs `ara::nm` ports and handle mappings
11. System Editor and Validation	Check the SYSTEM panels and fix what the VALIDATION panel reports
12. ARXML Export	Export the design and confirm the PNC mappings in the result
Appendix	Design checklist, project file structure, model fragments, and common problems

How to Read This Tutorial

- **Follow the chapters in order.** Each one starts from the result of the previous one. You go back once: after Chapter 4 you return to Chapter 3 to fill in that step.
- **Repetitive work is shown once.** One item (of 42 signals, 14 compositions, 33 PortGroups and so on) is shown as steps; the rest are given in value tabl
- **"Before You Begin"** states what the model must hold at the start, and any screenshot whose item count differs from your project.
- UI labels are written in **bold** as the screen shows them, and model element names and values in `code`.

Next Step

Start with Example and Design Goals.

Example and Design Goals

What You Will Do

This is a reading chapter with no screen work. It derives the three PNCs (Partial Network Clusters) of the example from a vehicle requirement and shows w

Before You Begin

- There is nothing to prepare. Model work starts in the next chapter, Topology Design.

Design Goals

The tutorial models this vehicle-level requirement as AUTOSAR PNCs.

> When a door opens, wake only the Comfort network; the park assist network comes alive only in reverse gear; and the trailer network runs only while a

The example vehicle uses a zonal architecture. One HPC runs on the AUTOSAR Adaptive Platform, three zone ECUs below it run on the Classic Platform, anc zone. A backbone Ethernet links the HPC and the zone ECUs.

The finished model has zero findings in the **VALIDATION** panel (The System Editor and Validation). Its exported ARXML passes the AUTOSAR R25-11 schema that come from the export itself (ARXML Export).

What a PNC Is

A PNC is **a set of functions that wake up together and go to sleep together**. It keeps unneeded ECUs and their communication asleep while the vehicle is defined by four things.

Element	Meaning
Identifier (PNC identifier)	The bit number this PNC occupies in the PNC bit vector inside the NM PDU
Participating physical channels	The CAN channels and Ethernet VLANs that wake up together when this PNC turns on
Communication unit	The I-PDU Group (Rx/Tx pair) this PNC switches on and off
Functional unit	The bundle of SWC ports this PNC controls. AUTOSAR calls it a PortGroup; this tutorial calls it a VFC (Virtual Function Cluster).

Group functions by **vehicle situation (wakeup trigger)**; the ECUs and buses that take part in them give the PNCs and their channels.

Deriving Three PNCs from the Vehicle Scenario

The requirement holds three vehicle situations. This table is the design specification of the whole tutorial.

PNC	Identifier	Vehicle situation (wakeup trigger)	Participating ECUs	Related buses	
`PNC_Comfort`	`17`	Door unlock, door open (`DoorStatus` frame)	`DoorCtrl_FL`, `DoorCtrl_FR`, `SeatCtrl_FL`, `SeatCtrl_FR`, `MirrorAct_FL`, `MirrorAct_FR`, `WindowSw_FL`, `WestZone`, `EastZone`, `ComfortApp` on the HPC	`WestCAN`, `EastCAN`, `VLAN_Body`	Door, left ar
`PNC_ParkAssist`	`18`	Reverse gear. When the `ParkAssistFG` state on the HPC becomes `Active`	`ParkSensor_RR`, `SouthZone`, `ParkAssistApp` on the HPC	`RearCAN`, `VLAN_Rear`	South HPC o throu
`PNC_Trailer`	`19`	Trailer attached (`TrailerStatus` frame)	`TrailerModule`, `TailLamp_RR`, `SouthZone`, `VehicleStateApp` on the HPC	`RearCAN`, `VLAN_Rear`	Trailer attach

A frame trigger such as the one of `PNC_Comfort` becomes a wakeup property on the connector (PNC Definition and Connector PNC Properties). The Func an NM network handle on the AP machine (The AP Machine — NM Handles and FunctionGroups).



*Figure 1-1. The **MATRIX** tab of the PNC dialog in the finished example. Rows are the PNCs, columns the physical channels; the marks match the "Related bu GROUPS, MATRIX and NM VECTOR.*

Why the PNC Identifiers Are 17, 18 and 19

A PNC identifier is **a bit number inside the NM PDU**, so it must fall within the PNC bit vector. This example follows the standard CanNm layout: byte 0 is the Identifier), and bytes 2 to 7 the PNC bit vector.

So **PNC VECTOR OFFSET** is `2` and **PNC VECTOR LENGTH** is `6`, and the usable identifiers run from bit `16` (2 × 8) to bit `63` ((2 + 6) × 8 − 1). AUTOSAR specifies t reports an identifier outside it as an error.

An early draft used `8`, `9` and `10`, which violated `constr_3040`. In PNC Definition and Connector PNC Properties you produce this error and fix it. You chec Reviewing PDU GROUPS, MATRIX and NM VECTOR.

The Five Layers a PNC Spans

AUTOSAR records a PNC across several elements, and PAIO edits each in its own editor.

AUTOSAR element	Role	PAIO edit
`PncMapping` (SystemMapping)	PNC identifier, participating physical channels, I-PDU Group, VFC, service instances, wakeup frames	PNC dialog of the System editor
PNC properties of `CommunicationConnector`	`pncGatewayType`, wakeup CAN ID and mask, PNC filter array mask	Properties panel of the System edito
`ISignalIPduGroup`	The communication unit a PNC switches on and off (Rx/Tx, container and leaf)	Signal editor and PNC dialog
`PortGroup` (= VFC)	The functional unit a PNC controls — a bundle of SWC ports	VFB editor for atomic SWCs; **SWC HIE** compositions and adaptive applicati
`NmNetworkHandle` and `NmHandleToFunctionGroupStateMapping`	The channel through which the AP machine (HPC) requests and releases a PNC, plus the link to FunctionGroup states	**Network Management** on the **MODUL** Configuration dialog
`CanNmCluster`, `UdpNmCluster`	The PNC bit vector inside the NM PDU (offset and length), and the NM nodes	NM configuration of the System edit the PNC dialog

The service instances the PNC mapping references are created in Chapter 5. The two "NM" rows differ: `NmNetworkHandle` is the software channel for a PN `UdpNmCluster` define the NM PDU on the bus.

Checklist

- [ ] You can name the identifier, trigger and related buses of each of the three PNCs.
- [ ] You can show why `17`, `18` and `19` are valid identifiers and `8`, `9` and `10` are not.
- [ ] You know which editors record a PNC: System editor (with the PNC dialog), Signal, Interface, VFB and Machine Configuration.

Next Step

In Topology Design you create the project `PNC` and draw the vehicle of this chapter in the System editor.

Topology Design

What You Will Do

In this chapter you create the project `PNC` and draw the vehicle topology in the System editor: the HPC, three zone ECUs, ten end ECUs, three CAN FD clu
a switch inside the HPC. You build a few elements step by step and add the rest from the tables.

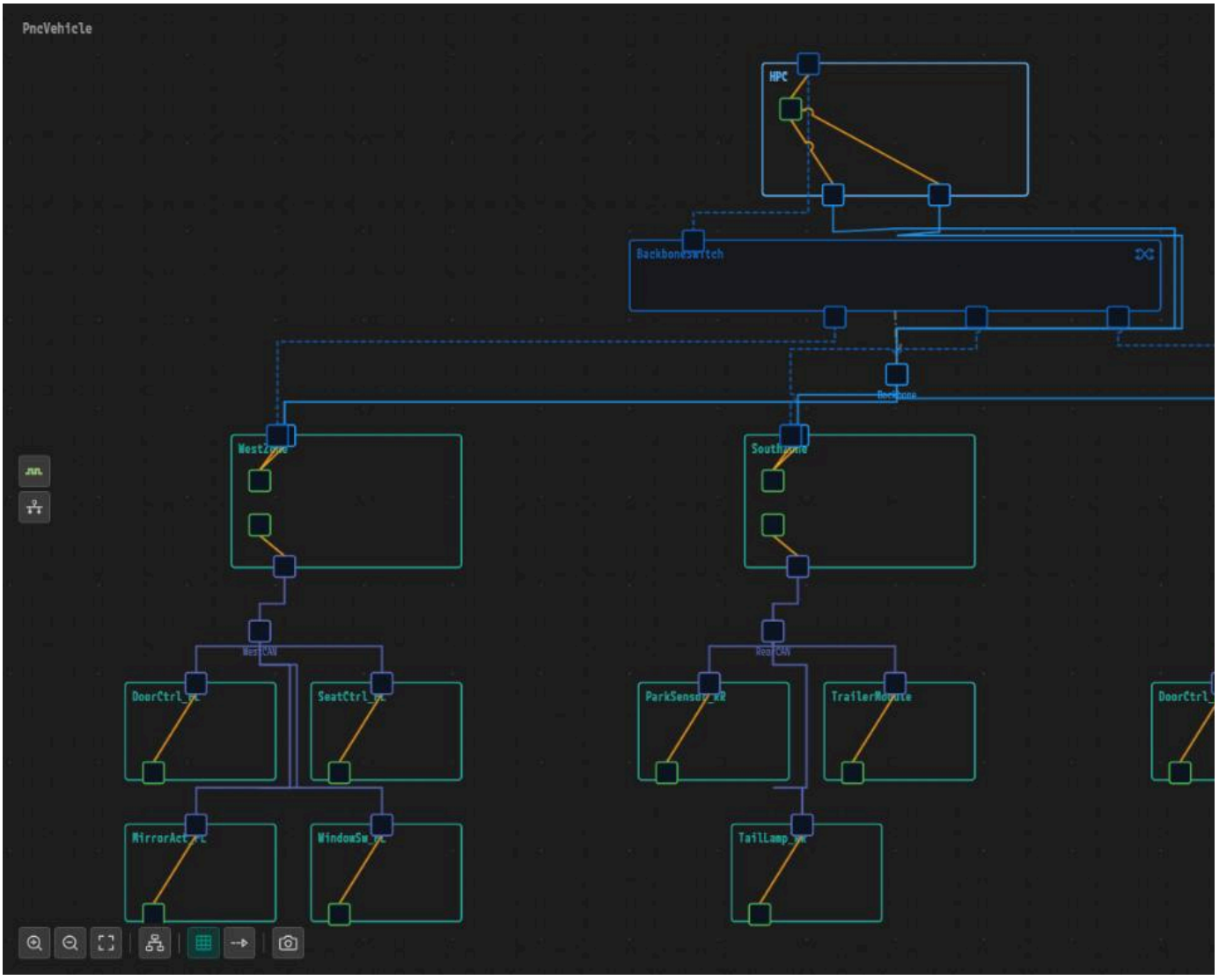


Figure 2-1. The finished topology. `PncVehicle` at the upper left is the root name of the finished example project.

Before You Begin

- Start PAIO in a window with no project open: the **EDITOR** view shows `No project open`. If PAIO opens an earlier project, choose **PROJECT** > **New Wind**
- The full System editor (formerly the IVC editor) is described in IVC Editor.

You build these elements step by step. The rest come from the tables in Adding the Remaining Nodes and Channels.

Element	Name	Notes
HPC	`HPC`	Ethernet controller `HPC_Eth`, connector `HPC_Eth_Body_Conn`
Zone ECU	`WestZone`	CAN FD controller `WestZone_CanFd`, Ethernet controller `WestZone_Eth`
End ECU	`DoorCtrl_FL`	CAN FD controller `DoorCtrl_FL_CanFd`
CAN cluster	`WestCAN`	500 kbps, channel `WestCAN_Channel`
Ethernet cluster	`Backbone`	Channels `VLAN_Body` (VLAN ID 100) and `VLAN_Rear` (VLAN ID 200)
Ethernet switch	`BackboneSwitch`	Embedded in the HPC, cluster `Backbone`
Coupling ports	`HPC_Eth_Port`, `WestZone_Eth_Port`, `Port_0`, `Port_1`	Physical connection between a switch port and a controller port

Creating the New Project

1. Click **PROJECT** > **New Project...**.

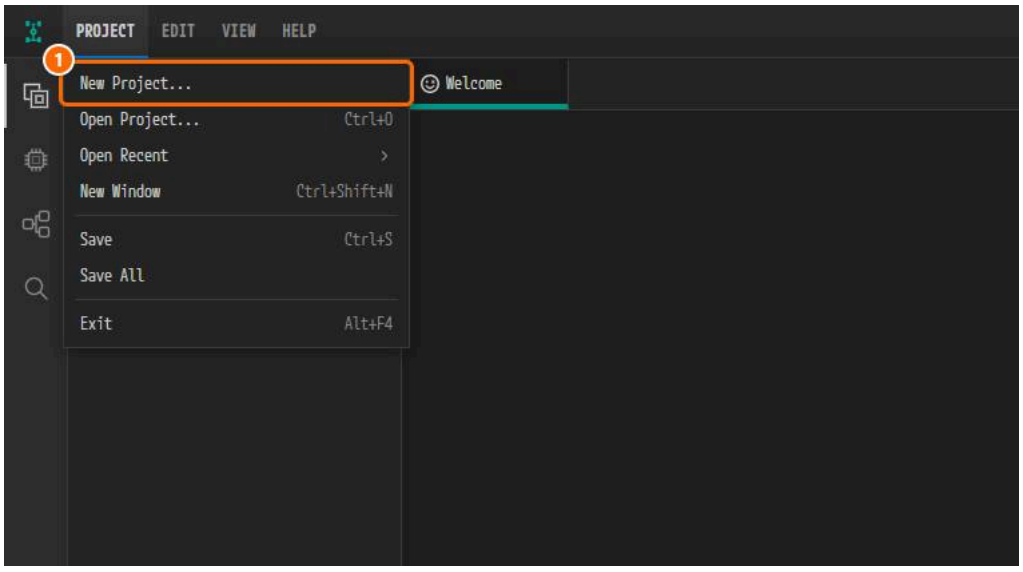


*Figure 2-2. ① **New Project...**.*

2. Enter `PNC` in **Name**. Set **Location** to a folder of your own, by typing it or with **Browse...**. Click **Create**.

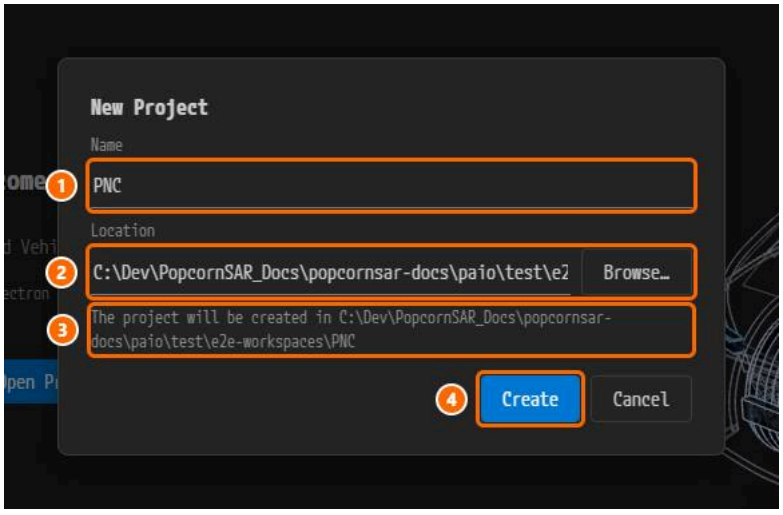


*Figure 2-3. ① **Name**, ② **Location** and **Browse...**, ③ the new project folder, ④ **Create**. The folder is only an example.*

3. Check that the title bar reads `PAIO [PNC]`.

Opening the System Editor

1. In the **EDITOR** view, click **System** under **DESIGN**.

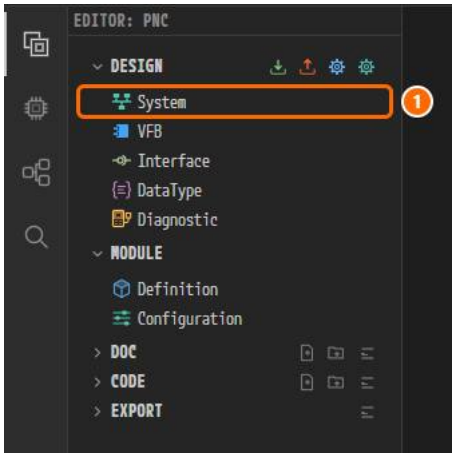


*Figure 2-4. ① **System**.*

2. The tab **PNC:System** opens with an empty canvas.

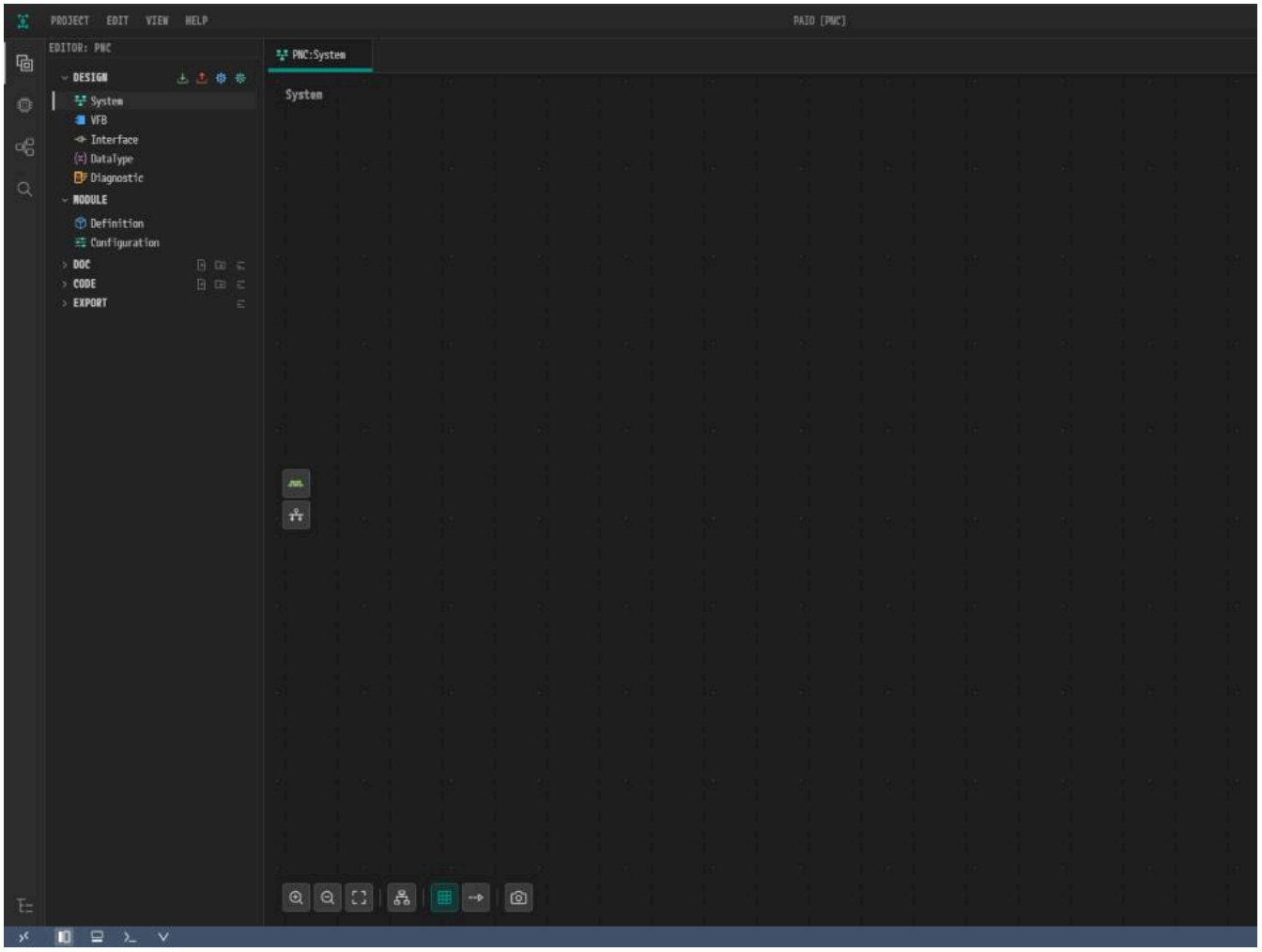


*Figure 2-5. The empty System editor. The toolbar with **Auto Layout** is at the lower left of the canvas. The root name of a new project is `System`.*

Adding the HPC

1. Right-click an empty area of the canvas and click **Add HPC**.



*Figure 2-6. ① **Add HPC.***

2. Click the new node `HPC_0`. Enter `HPC` in **NAME** and press `Tab`.

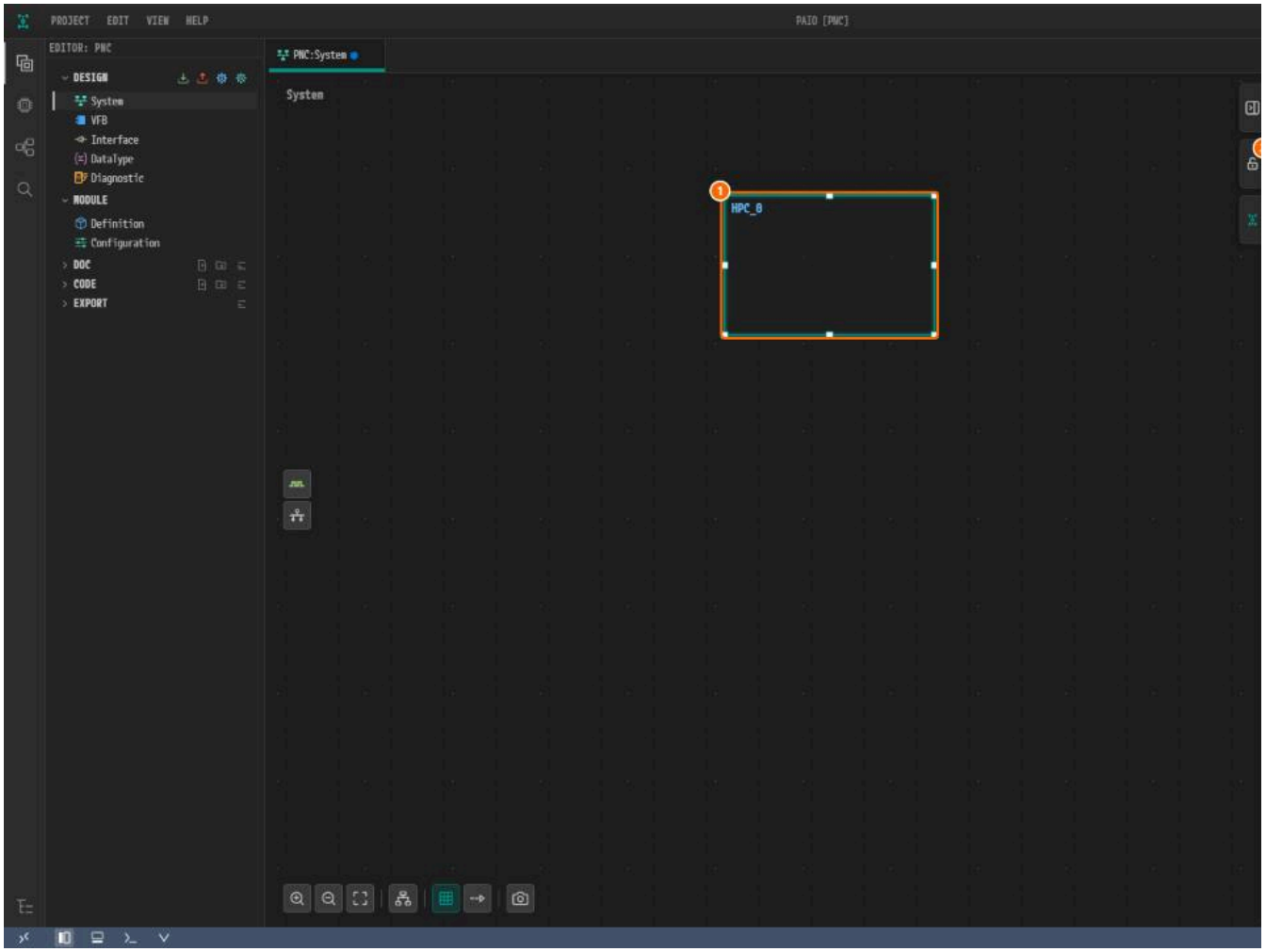


*Figure 2-7. ① The HPC node, ② **NAME.***

Adding the Zone ECU and the End ECU

1. Right-click an empty area of the canvas and click **Add ECU**. Rename the new node to `WestZone`.
2. Add another ECU the same way and rename it to `DoorCtrl_FL`.

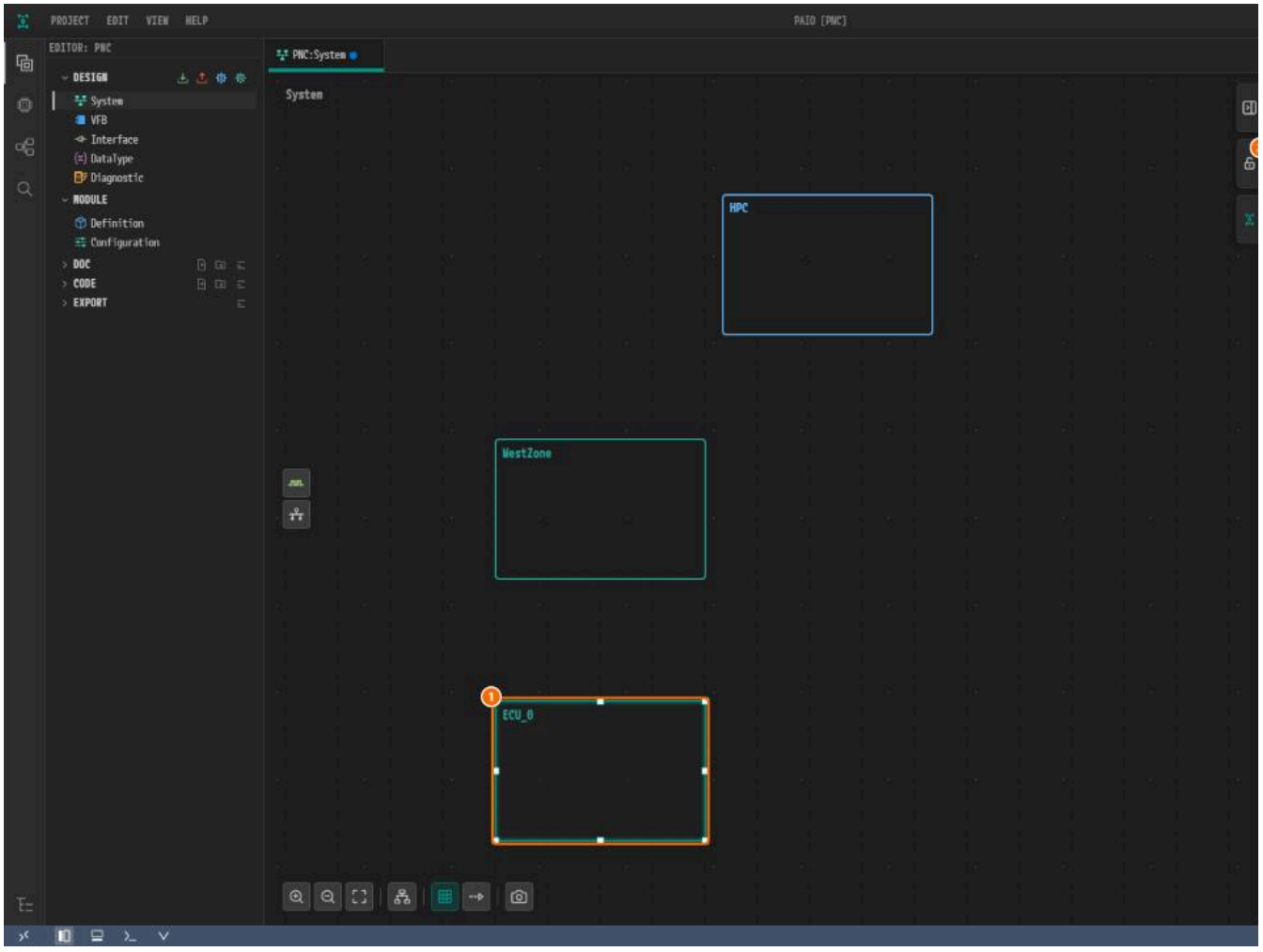


*Figure 2-8. ① The new ECU, ② **NAME**.*

Adding the CAN FD Controller and Connector

1. Right-click an empty area inside `WestZone` and choose **Add Controller** ▸ **CAN**.

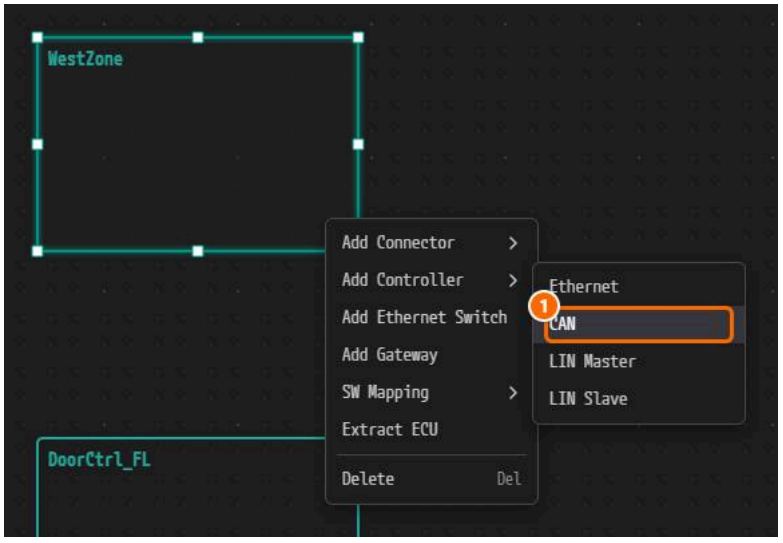


*Figure 2-9. ① **CAN**.*

> **Caution.** Right-click inside the node, away from its header and the connectors on its edge. Otherwise the menu of that element opens.

2. Click the new controller (the green rectangle). Change **NAME** to `WestZone_CanFd` and click **CAN FD** in the **CAN CONTROLLER CONFIGURATION** header. L

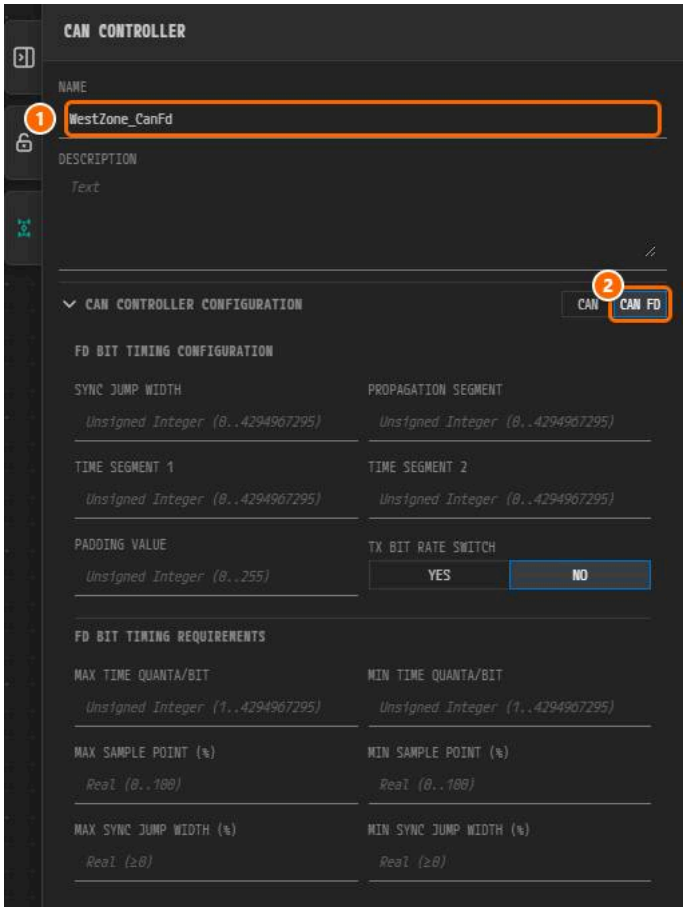


*Figure 2-10. ① **NAME**, ② **CAN FD**.*

3. Right-click inside `WestZone` and choose **Add Connector ▸ CAN**. Click the new connector on the node edge.

4. Change **NAME** to `WestZone_CanFd_Conn` and pick `WestZone_CanFd` in **CONTROLLER**. An orange line joins the controller and the connector.

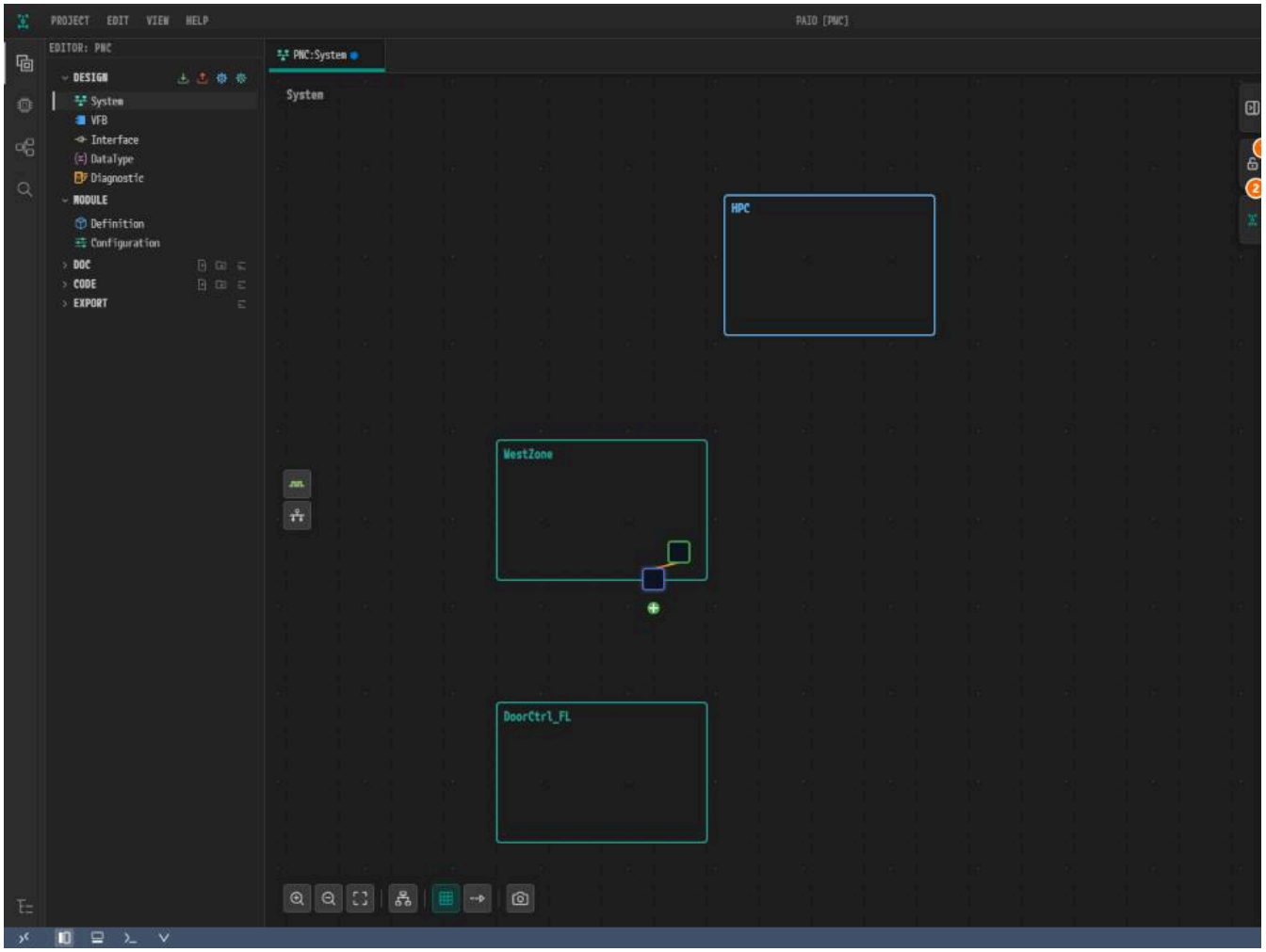


*Figure 2-11. ① **NAME**, ② **CONTROLLER**.*

5. On `DoorCtrl_FL`, create the controller `DoorCtrl_FL_CanFd` (**CAN FD**) and the connector `DoorCtrl_FL_CAN_Conn` the same way.

Creating the CAN Cluster WestCAN

1. Click the green **[+]** badge below `WestZone_CanFd_Conn` and click **Create CommunicationCluster [Can]**.

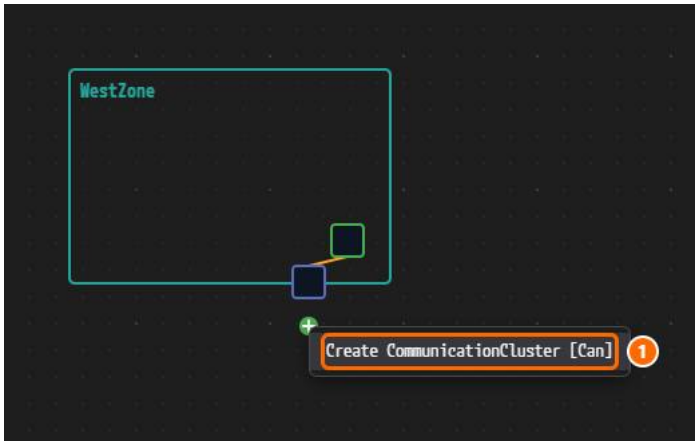


*Figure 2-12. ① **Create CommunicationCluster [Can]**.*

2. Click the new cluster (the purple rectangle). Change **NAME** to `WestCAN`, click `500` under **BAUDRATE (ARBITRATION, KBPS)**, and change **CHANNEL NAME**

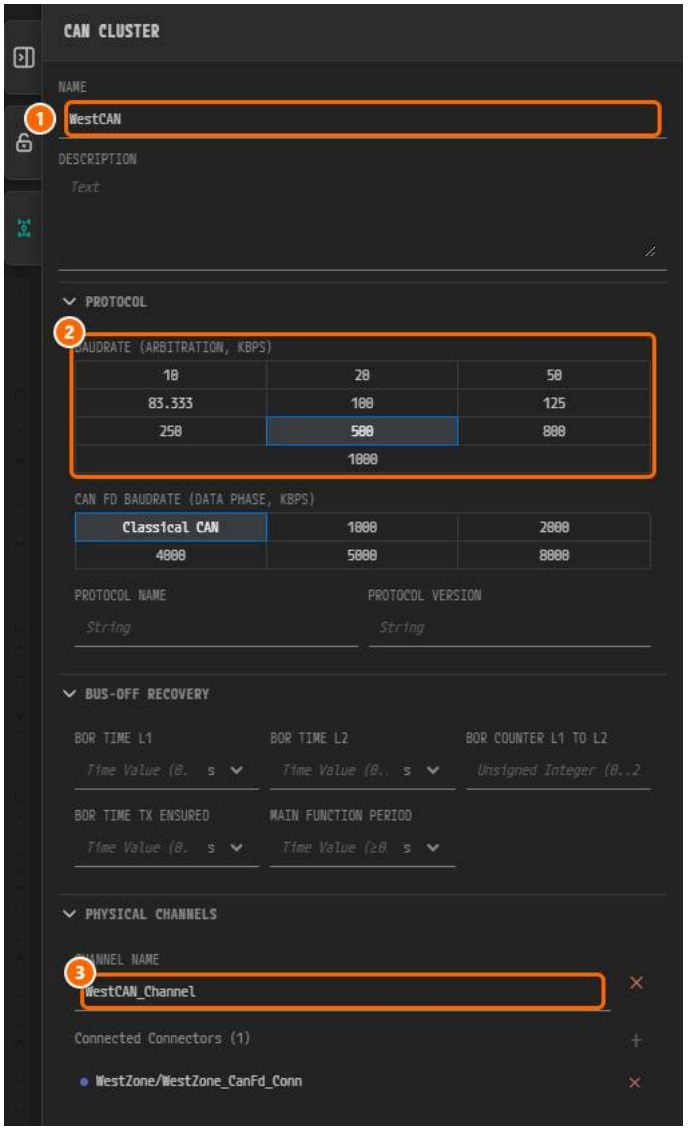


*Figure 2-13. ① **NAME**, ② `500`, ③ **CHANNEL NAME**. Leave **CAN FD BAUDRATE (DATA PHASE, KBPS)** at **Classical CAN**.*

3. Click the **[+]** badge below `DoorCtrl_FL_CAN_Conn` and choose **Connect to Channel...** ▸ **WestCAN** ▸ **WestCAN_Channel**.

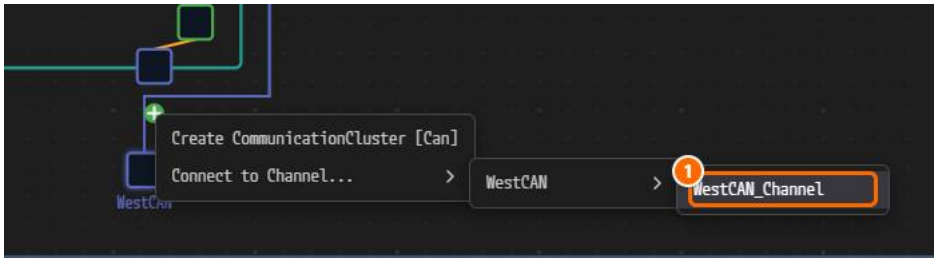


Figure 2-14. ① `WestCAN_Channel`.

Creating the Ethernet Cluster Backbone and Its VLAN Channels

1. Inside `HPC`, add an Ethernet controller with **Add Controller** ▸ **Ethernet** and name it `HPC_Eth`.
2. Inside `HPC`, add a connector with **Add Connector** ▸ **Ethernet**. Name it `HPC_Eth_Body_Conn` and set **CONTROLLER** to `HPC_Eth`.
3. Click the **[+]** badge of `HPC_Eth_Body_Conn` and choose **Create CommunicationCluster [Ethernet]**.
4. Click the new cluster and change **NAME** to `Backbone`. On the first channel under **PHYSICAL CHANNELS**, set **CHANNEL NAME** to `VLAN_Body` and **VLAN ID**
5. Click **+** at the right of **PHYSICAL CHANNELS**. Set the second channel to `VLAN_Rear` with **VLAN ID** `200`.

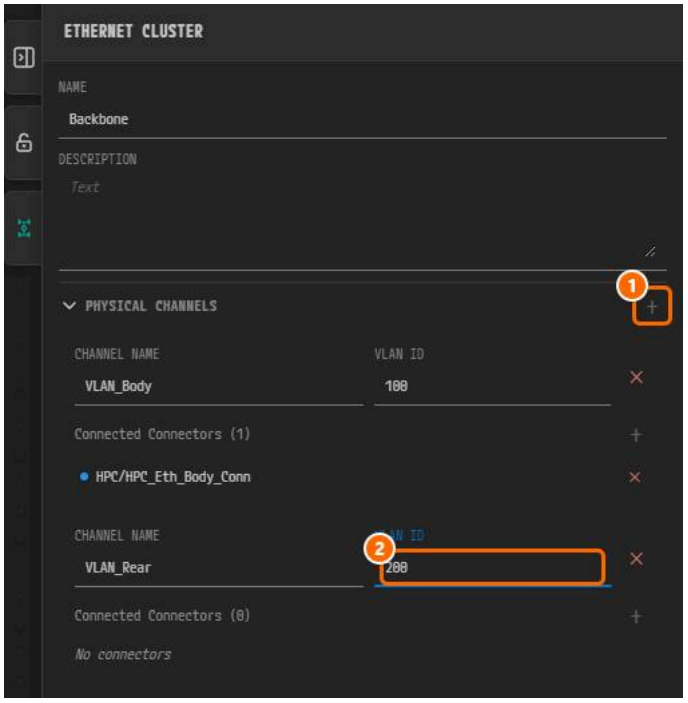


*Figure 2-15. ① **+** (Add Channel), ② **VLAN ID** of the second channel.*

6. Inside `WestZone`, add the Ethernet controller `WestZone_Eth` and the Ethernet connector `WestZone_Eth_Conn` with **CONTROLLER** `WestZone_Eth`.
7. Click the **[+]** badge of `WestZone_Eth_Conn` and choose **Connect to Channel...** ▸ **Backbone** ▸ **VLAN_Body (VLAN: 100)**.

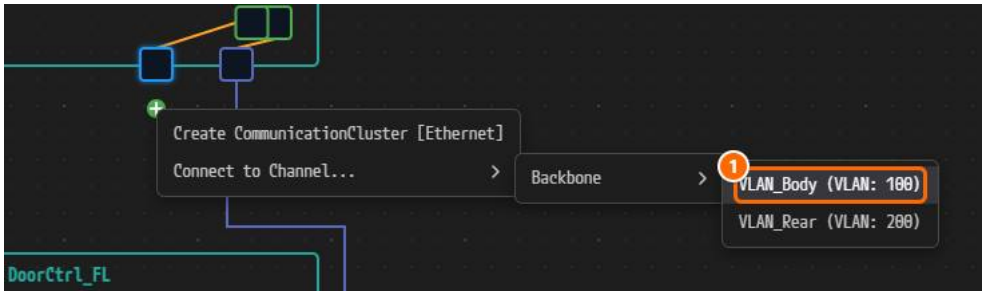


*Figure 2-16. ① **VLAN_Body (VLAN: 100)**.*

Adding the Ethernet Switch

1. Right-click an empty area inside `HPC` and click **Add Ethernet Switch**.

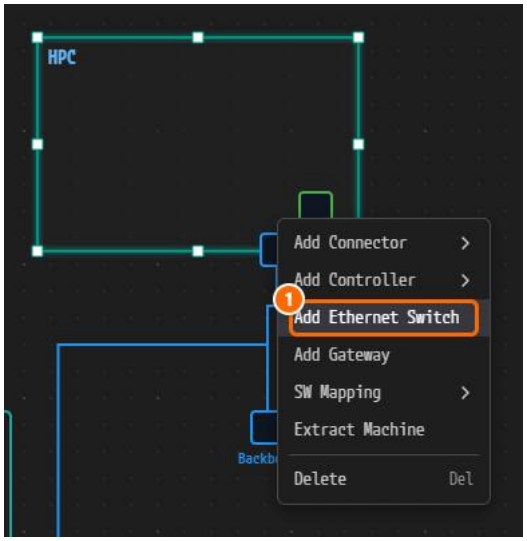


*Figure 2-17. ① **Add Ethernet Switch.***

2. Click the new switch `EthSwitch_0`. Change **NAME** to `BackboneSwitch` and pick `Backbone` in **COMMUNICATION CLUSTER**. Leave **COUPLING TYPE** at **SWI**

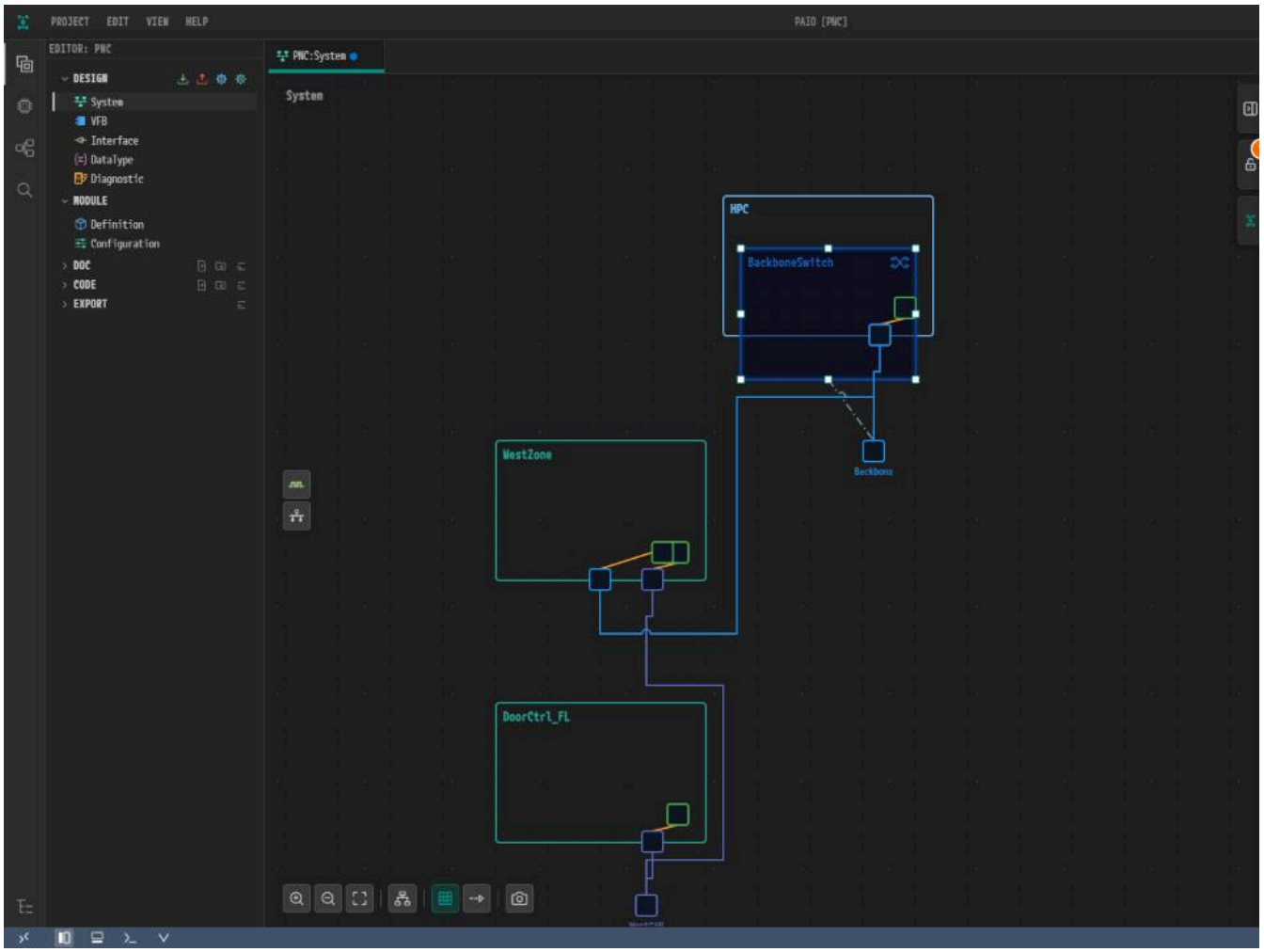


*Figure 2-18. ① **NAME**, ② **COMMUNICATION CLUSTER**.*

The new switch overlaps the HPC edge until you run **Auto Layout**.

Auto Layout and Saving

1. Click **Auto Layout** in the toolbar at the lower left of the canvas.

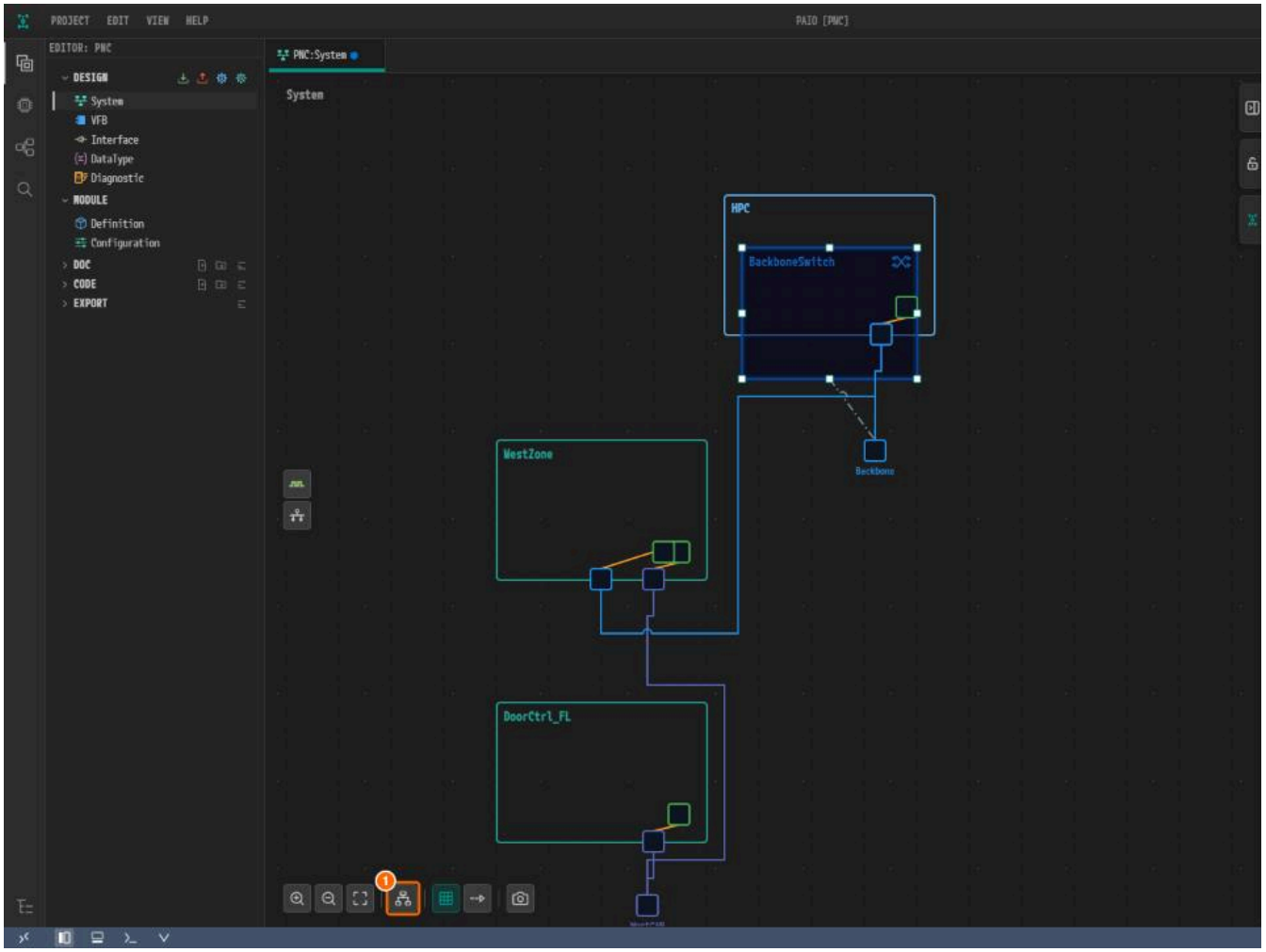


*Figure 2-19. ① **Auto Layout.***

2. Save with `Ctrl+S`.

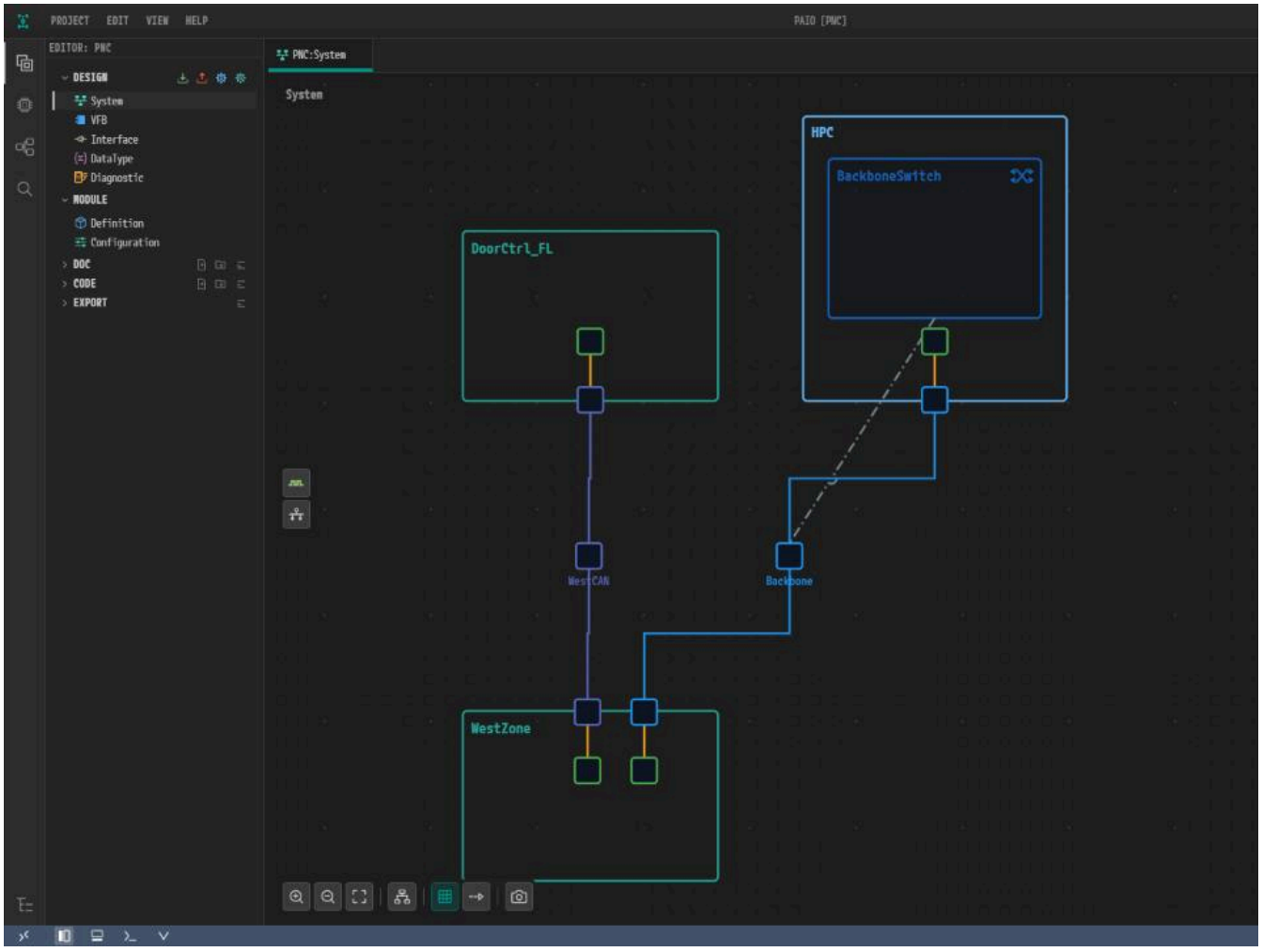


*Figure 2-20. After **Auto Layout**. `BackboneSwitch` now sits inside the HPC.*

Auto Layout does not order the nodes from top to bottom, so `DoorCtrl_FL` may end up above `WestZone`. Drag nodes if you like.

Creating and Connecting the Switch Ports

A coupling port joins a switch port to an Ethernet controller. **Connect to Port...** lists only unconnected ports in the same Ethernet cluster, so connect eac

1. Right-click the controller `HPC_Eth` inside `HPC` and click **Add Coupling Port**.

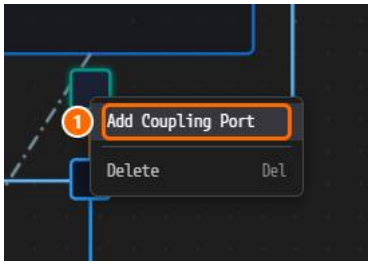


*Figure 2-21. ① **Add Coupling Port**.*

2. Click the new port on the node edge and change **NAME** to `HPC_Eth_Port`.
3. Create the port `WestZone_Eth_Port` on `WestZone_Eth` the same way.
4. Right-click `BackboneSwitch` and click **Add CouplingPort**.

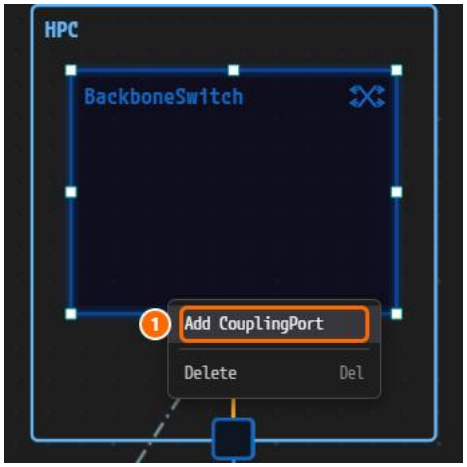


*Figure 2-22. ① **Add CouplingPort**.*

5. Click the new port. Change **NAME** to `Port_0`, and set **PORT ROLE** to **HOST-PORT**, **PHYSICAL LAYER TYPE** to **1000BASE-T1** and **NEGOTIATION** to **AUTO**. With `constr_5251`.

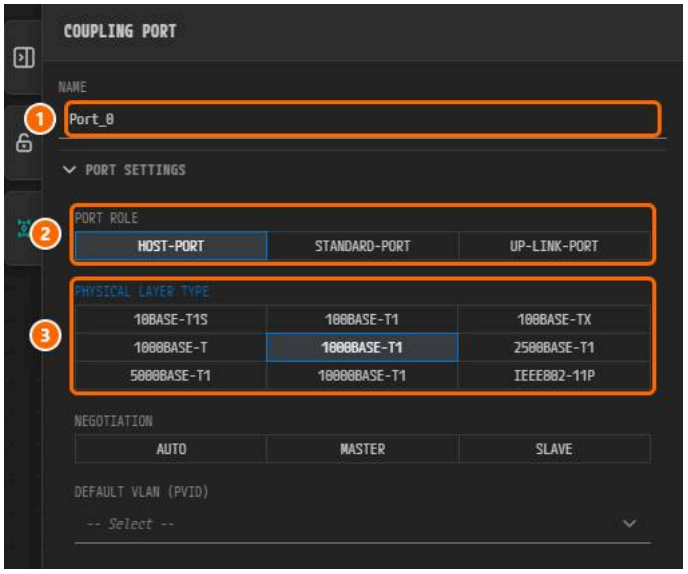


*Figure 2-23. ① **NAME**, ② **PORT ROLE**, ③ **PHYSICAL LAYER TYPE**. **NEGOTIATION** is below ③.*

6. Right-click a different spot on `BackboneSwitch` and add `Port_1` the same way, with **PORT ROLE STANDARD-PORT**.

> **Caution.** A new port attaches to the edge nearest the click. Two ports made at the same spot overlap, and the first one can no longer be clicked.

7. Right-click `Port_0` and choose **Connect to Port...** ▸ **HPC_Eth / HPC_Eth_Port**. Connect `Port_1` to **WestZone_Eth / WestZone_Eth_Port** the same wa



*Figure 2-24. ① **WestZone_Eth / WestZone_Eth_Port**.*

8. Save with `Ctrl+S`.

Adding the Remaining Nodes and Channels

Add the remaining elements with the same steps, using the names in the tables.

- Nodes: **Add ECU** on the canvas, then **NAME**.

- Controllers and connectors: **Add Controller** and **Add Connector** inside the node, then **NAME** and the connector's **CONTROLLER**. Turn on **CAN FD** on eve second connector in `HPC` whose **CONTROLLER** is also `HPC_Eth`.
- Clusters: **[+]** on the first connector → **Create CommunicationCluster [Can]**, then **NAME**, `500` and **CHANNEL NAME**.
- Channel connections: **[+]** on the connector → **Connect to Channel...** ▸ **cluster** ▸ **channel**.

Node Table

Node	Kind	Controllers	Connectors	Channel to connect
`HPC`	HPC	`HPC_Eth` (Ethernet)	`HPC_Eth_Body_Conn`, `HPC_Eth_Rear_Conn`	`VLAN_Body`, `VLAN_Rear`
`WestZone`	ECU (zone)	`WestZone_Eth` (Ethernet), `WestZone_CanFd` (CAN FD)	`WestZone_Eth_Conn`, `WestZone_CanFd_Conn`	`VLAN_Body`, `WestCAN_Chann`
`SouthZone`	ECU (zone)	`SouthZone_Eth` (Ethernet), `SouthZone_CanFd` (CAN FD)	`SouthZone_Eth_Conn`, `SouthZone_CanFd_Conn`	`VLAN_Rear`, `RearCAN_Chann`
`EastZone`	ECU (zone)	`EastZone_Eth` (Ethernet), `EastZone_CanFd` (CAN FD)	`EastZone_Eth_Conn`, `EastZone_CanFd_Conn`	`VLAN_Body`, `EastCAN_Chann`
`DoorCtrl_FL`	ECU	`DoorCtrl_FL_CanFd` (CAN FD)	`DoorCtrl_FL_CAN_Conn`	`WestCAN_Channel`
`SeatCtrl_FL`	ECU	`SeatCtrl_FL_CanFd` (CAN FD)	`SeatCtrl_FL_CAN_Conn`	`WestCAN_Channel`
`MirrorAct_FL`	ECU	`MirrorAct_FL_CanFd` (CAN FD)	`MirrorAct_FL_CAN_Conn`	`WestCAN_Channel`
`WindowSw_FL`	ECU	`WindowSw_FL_CanFd` (CAN FD)	`WindowSw_FL_CAN_Conn`	`WestCAN_Channel`
`ParkSensor_RR`	ECU	`ParkSensor_RR_CanFd` (CAN FD)	`ParkSensor_RR_CAN_Conn`	`RearCAN_Channel`
`TrailerModule`	ECU	`TrailerModule_CanFd` (CAN FD)	`TrailerModule_CAN_Conn`	`RearCAN_Channel`
`TailLamp_RR`	ECU	`TailLamp_RR_CanFd` (CAN FD)	`TailLamp_RR_CAN_Conn`	`RearCAN_Channel`
`DoorCtrl_FR`	ECU	`DoorCtrl_FR_CanFd` (CAN FD)	`DoorCtrl_FR_CAN_Conn`	`EastCAN_Channel`
`SeatCtrl_FR`	ECU	`SeatCtrl_FR_CanFd` (CAN FD)	`SeatCtrl_FR_CAN_Conn`	`EastCAN_Channel`
`MirrorAct_FR`	ECU	`MirrorAct_FR_CanFd` (CAN FD)	`MirrorAct_FR_CAN_Conn`	`EastCAN_Channel`

Cluster Table

Cluster	Kind	BAUDRATE (ARBITRATION, KBPS)	Channel (VLAN ID)	Connected connectors
`Backbone`	Ethernet	—	`VLAN_Body` (100)	`HPC_Eth_Body_Conn`, `WestZone_Eth_Conn`, `EastZone_Eth_Conn`
			`VLAN_Rear` (200)	`HPC_Eth_Rear_Conn`, `SouthZone_Eth_Conn`
`WestCAN`	CAN	500	`WestCAN_Channel`	`WestZone_CanFd_Conn`, `DoorCtrl_FL_CAN_Conn`, `SeatCtrl_FL_CAN_Conn`, `MirrorAct_FL`
`RearCAN`	CAN	500	`RearCAN_Channel`	`SouthZone_CanFd_Conn`, `ParkSensor_RR_CAN_Conn`, `TrailerModule_CAN_Conn`, `TailLam`
`EastCAN`	CAN	500	`EastCAN_Channel`	`EastZone_CanFd_Conn`, `DoorCtrl_FR_CAN_Conn`, `SeatCtrl_FR_CAN_Conn`, `MirrorAct_FR`

Switch Port Table

Every port uses **PHYSICAL LAYER TYPE 1000BASE-T1** and **NEGOTIATION AUTO**.

Port	Role	Connected to	VLAN
`Port_0`	HOST	`HPC/HPC_Eth_Port`	`VLAN_Body`, `VLAN_Rear`
`Port_1`	STD	`WestZone/WestZone_Eth_Port`	`VLAN_Body`
`Port_2`	STD	`SouthZone/SouthZone_Eth_Port`	`VLAN_Rear`
`Port_3`	STD	`EastZone/EastZone_Eth_Port`	`VLAN_Body`

Chapter 3 attaches PNCs to these ports, so create all of them. Details are in sections 6.4.3, 6.4.4 and 6.5.4 of IVC Editor.

1. Create `SouthZone_Eth_Port` and `EastZone_Eth_Port` on `SouthZone_Eth` and `EastZone_Eth` as in Creating and Connecting the Switch Ports.
2. Add `Port_2` and `Port_3` to `BackboneSwitch` as **STANDARD-PORT**, each at a different spot on the edge.
3. Connect each switch port to its target from the table with **Connect to Port...**.
4. Double-click `BackboneSwitch` to open the workbench. Set **DEFAULT VLAN (PVID)** and **VLAN MEMBERSHIP** of each port as in the table. Only `Port_2` ha `VLAN_Body`.

Run **Auto Layout** and save with `Ctrl+S`.

Reviewing the Finished Example

Your layout and root name may differ from the figures; the nodes, clusters and connections must match.

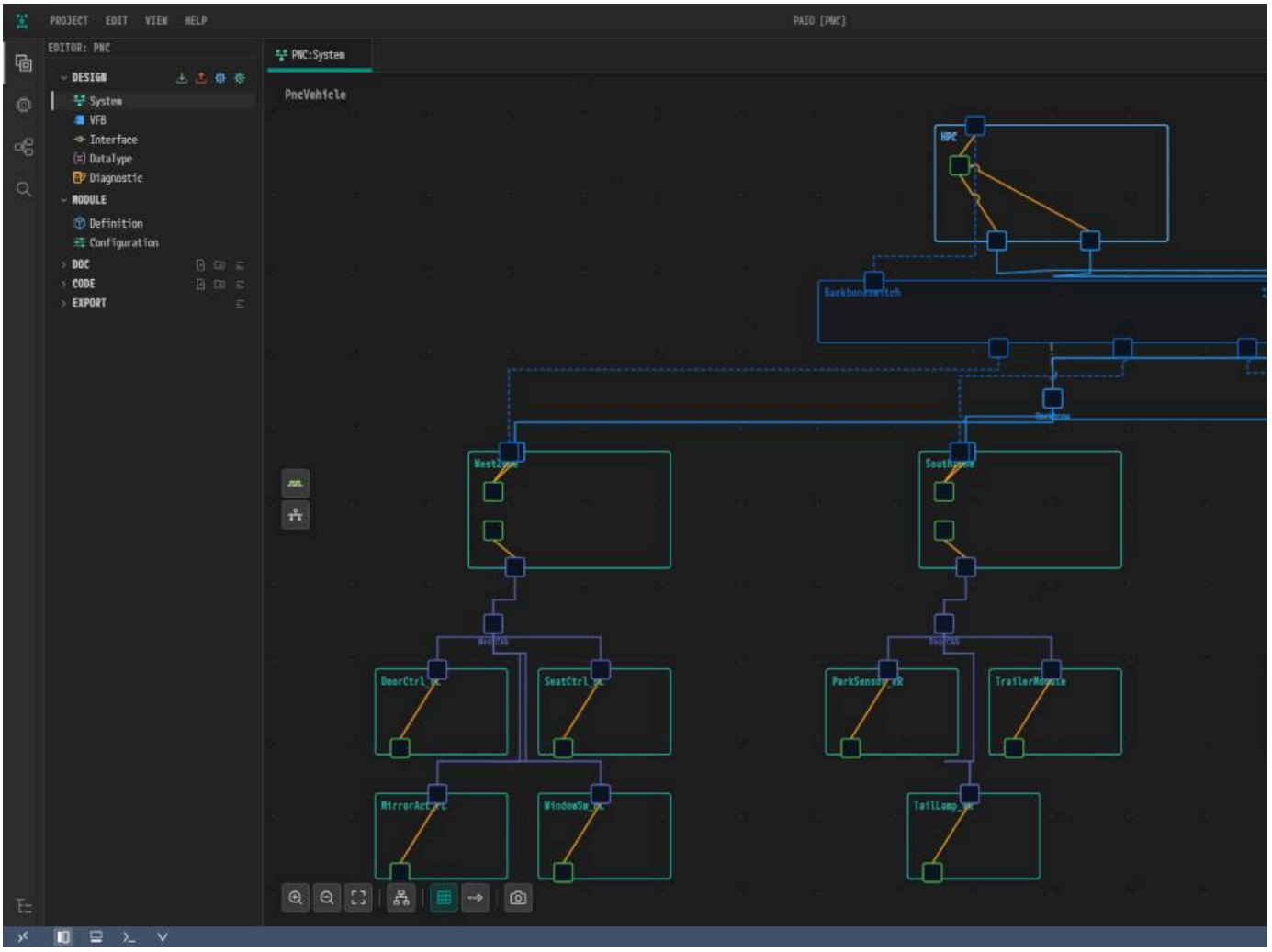


*Figure 2-25. The finished example after **Fit to View**. The blue rectangle in the center is `Backbone`, the purple ones are `WestCAN`, `RearCAN` and `EastCAN`. B*

Hover over a cluster to see its name, kind and baud rate.

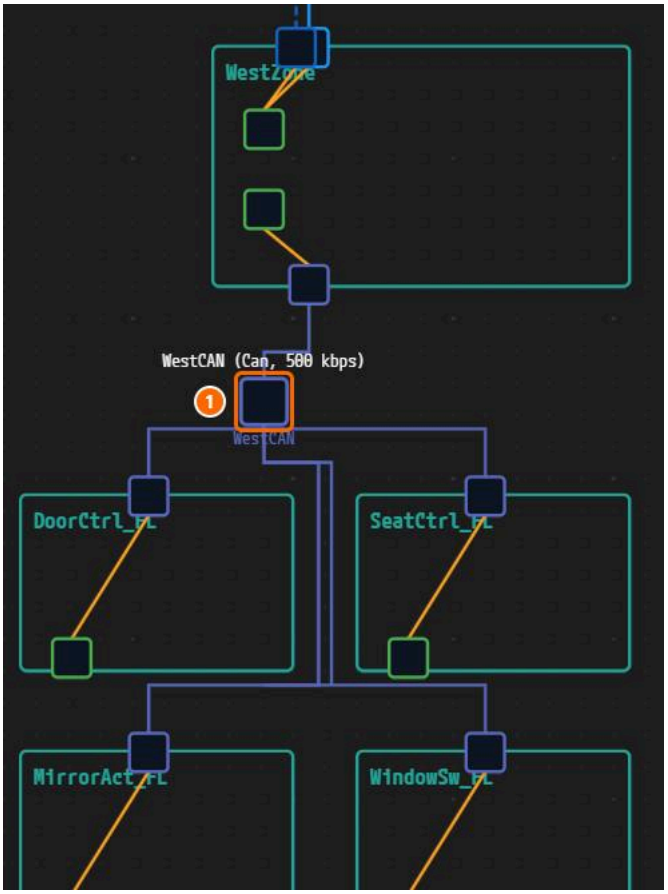


Figure 2-26. ① The `WestCAN` cluster, shown as `WestCAN (Can, 500 kbps)`.

Double-click `BackboneSwitch` to check the workbench, then click **Close**.

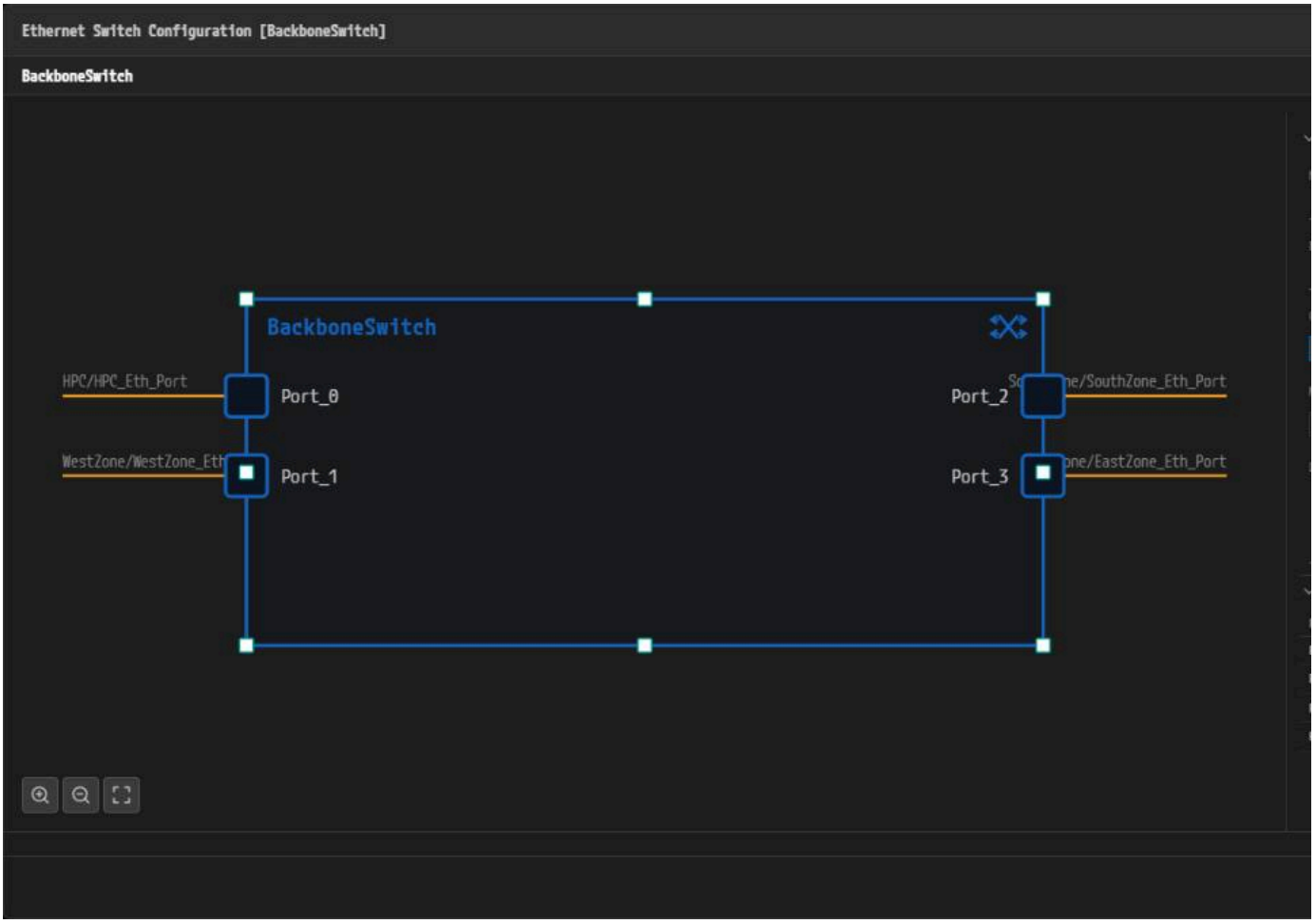


*Figure 2-27. The **Ethernet Switch Configuration [BackboneSwitch]** workbench. In **PORTS (4)**, only `Port_0` is HOST and only `Port_2` uses VLAN 200.*

Why Split the Backbone into Two VLANs?

- `constr_5093`. A physical channel may have at most one connector whose PNC gateway type is active.
- `constr_3781`. Every PNC needs a top-level coordinator: an ECU connected to two or more of its channels, with all those connections active.
- **The HPC is an AP machine** and cannot be a PNC gateway. With a single backbone channel, only one of the three zones could be active.
- **Two VLANs.** With `VLAN_Body` (West, East) and `VLAN_Rear` (South), WestZone coordinates `PNC_Comfort`, SouthZone coordinates `PNC_ParkAssist` an coordinator with a passive gateway on `VLAN_Body`. You enter the gateways in PNC Definitions and Connector PNC Properties.

Checklist

- [ ] The canvas holds `HPC`, `WestZone`, `SouthZone`, `EastZone` and ten end ECUs.
- [ ] Every CAN controller is **CAN FD**, and every CAN cluster is `500` with the connectors of the cluster table.
- [ ] `Backbone` has `VLAN_Body` (100) and `VLAN_Rear` (200).
- [ ] Each `<node>_Eth_Port` is connected to its switch port, and every switch port has **NEGOTIATION AUTO**.
- [ ] The **PORTS (4)** table of the workbench matches the switch port table.
- [ ] The tab shows no modified mark.

Next Step

In PNC Definitions and Connector PNC Properties you define the three PNCs and set their properties on these connectors.

PNC Catalog, Connector PNC Attributes, and NM Configuration

What You Will Do

In this chapter you declare the three PNCs (Partial Network Clusters) in the PNC dialog and fix the validation error that an invalid identifier causes. Then y attributes of the connectors, nodes, and backbone switch ports, and fill in the NM values that carry the PNC bits.

Before You Begin

- Finish Topology Design so that all nodes, clusters, connectors, and switch ports exist.
- In the **EDITOR** view, click **System** under **DESIGN** to open the System editor (tab **PNC:System**).
- **TX-NM-PDU-REFS** and **RX-NM-PDU-REFS** of the NM nodes need the 18 NM-PDUs of Signals, PDUs, Frames, and I-PDU Groups. Fill them in after that cha
- The screenshots come from the finished example, so the root name on the canvas reads `PncVehicle`. In your project it may read `System`.

PNC IDENTIFIER	NAME	Position in the filter mask
`17`	`PNC_Comfort`	Byte 0, bit 1 → `0x02`
`18`	`PNC_ParkAssist`	Byte 0, bit 2 → `0x04`
`19`	`PNC_Trailer`	Byte 0, bit 3 → `0x08`

Opening the PNC Dialog

1. In the icon strip on the left edge of the canvas, click the lower icon, **Partial Network (PNC)**.

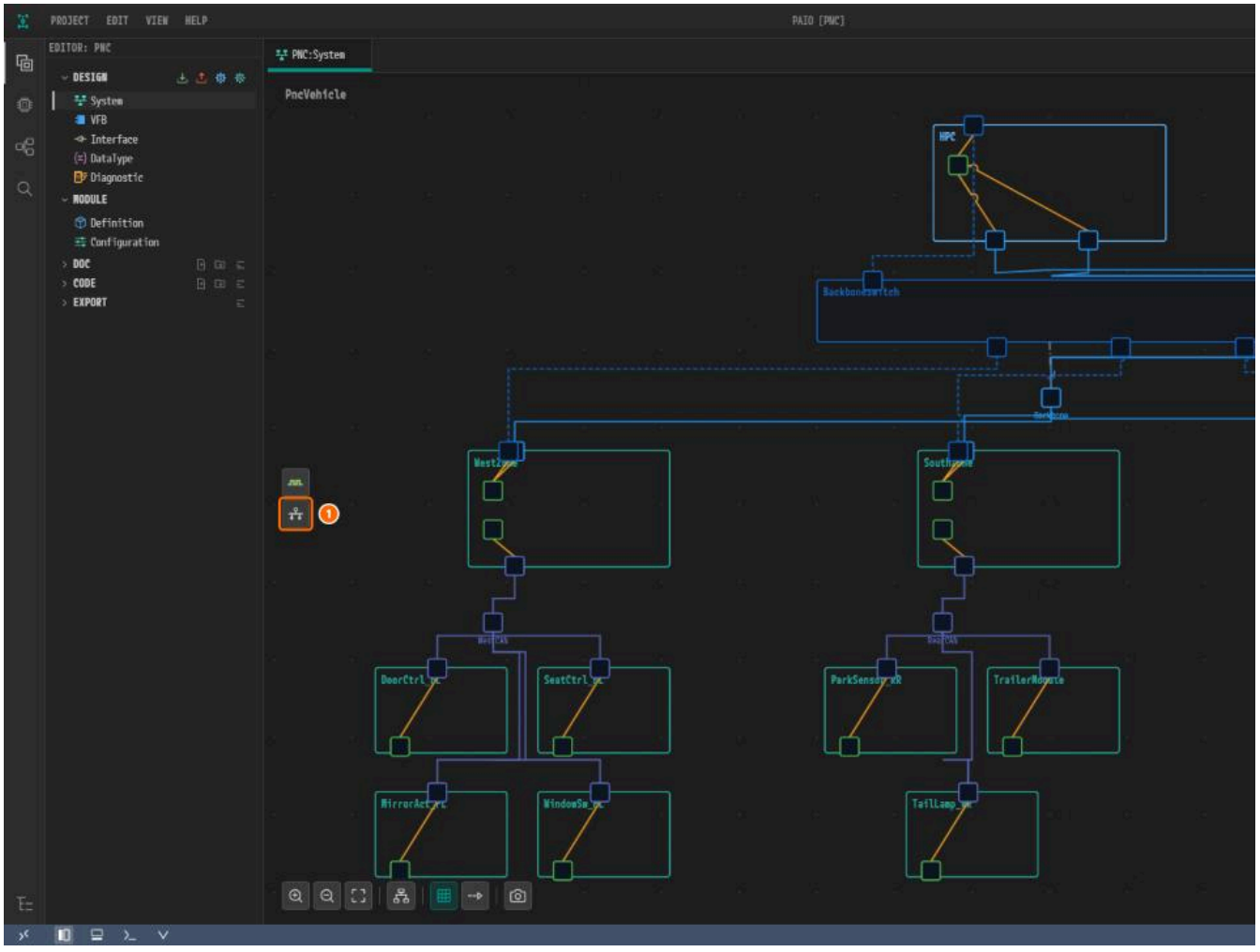


*Figure 3-1. ① The PNC button. The icon above it is **Signal**.*



*Figure 3-2. ① The **Partial Network (PNC)** tooltip.*

2. The **Partial Network (PNC) Configuration** dialog opens with an empty **PARTIAL NETWORKS** list.

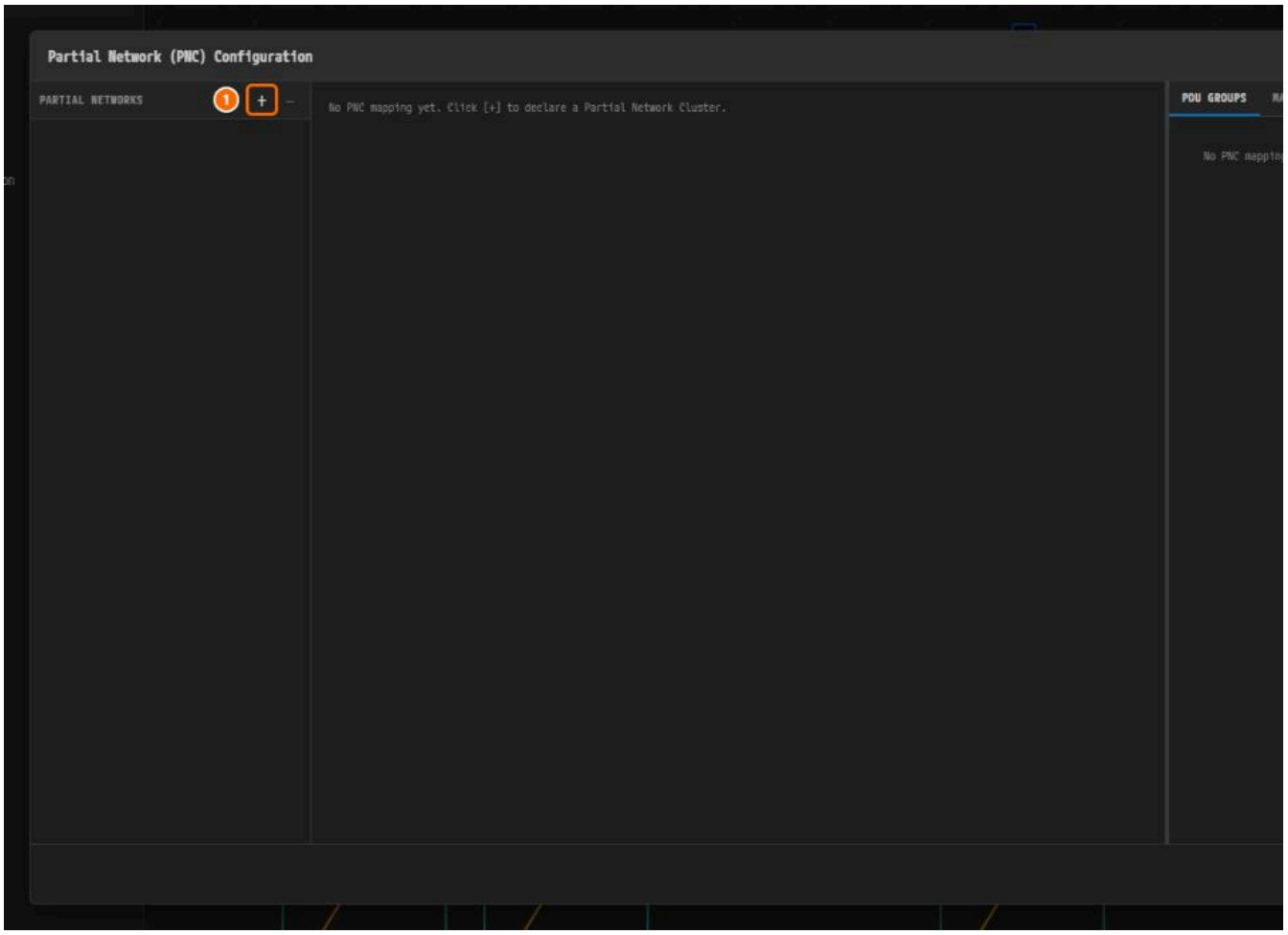


*Figure 3-3. ① The **[+]** button.*

Adding the First PNC

1. Click **[+]** next to **PARTIAL NETWORKS**. The entry **PNC 0 Pnc_0** appears.

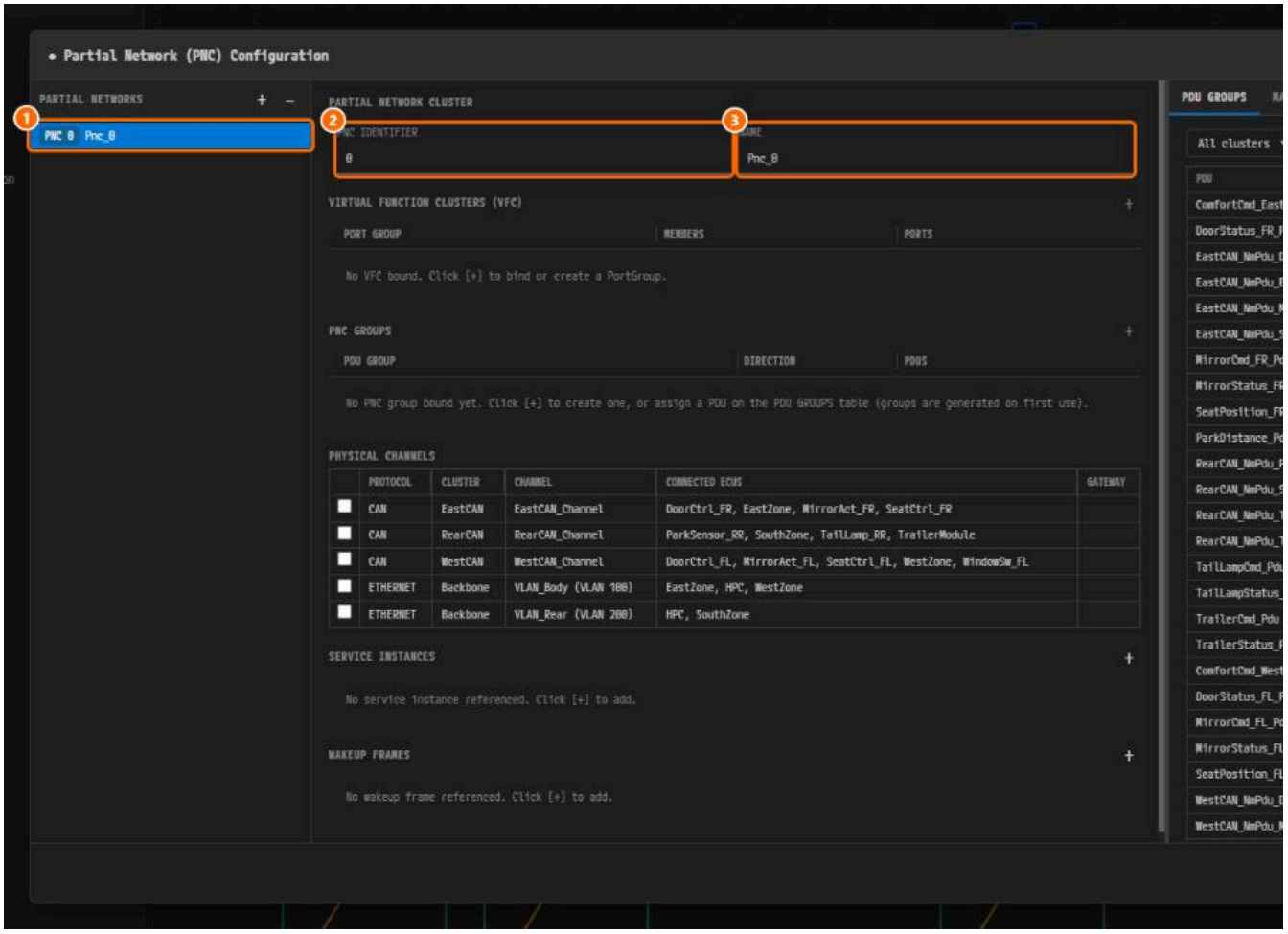


*Figure 3-4. ① **PNC 0 Pnc_0**, ② **PNC IDENTIFIER**, ③ **NAME**. You fill in the lists below them in PNC Dialog — Per-PNC Bindings.*

2. Enter `PNC_Comfort` in **NAME** and press `Enter`. Leave **PNC IDENTIFIER** at `0` on purpose.

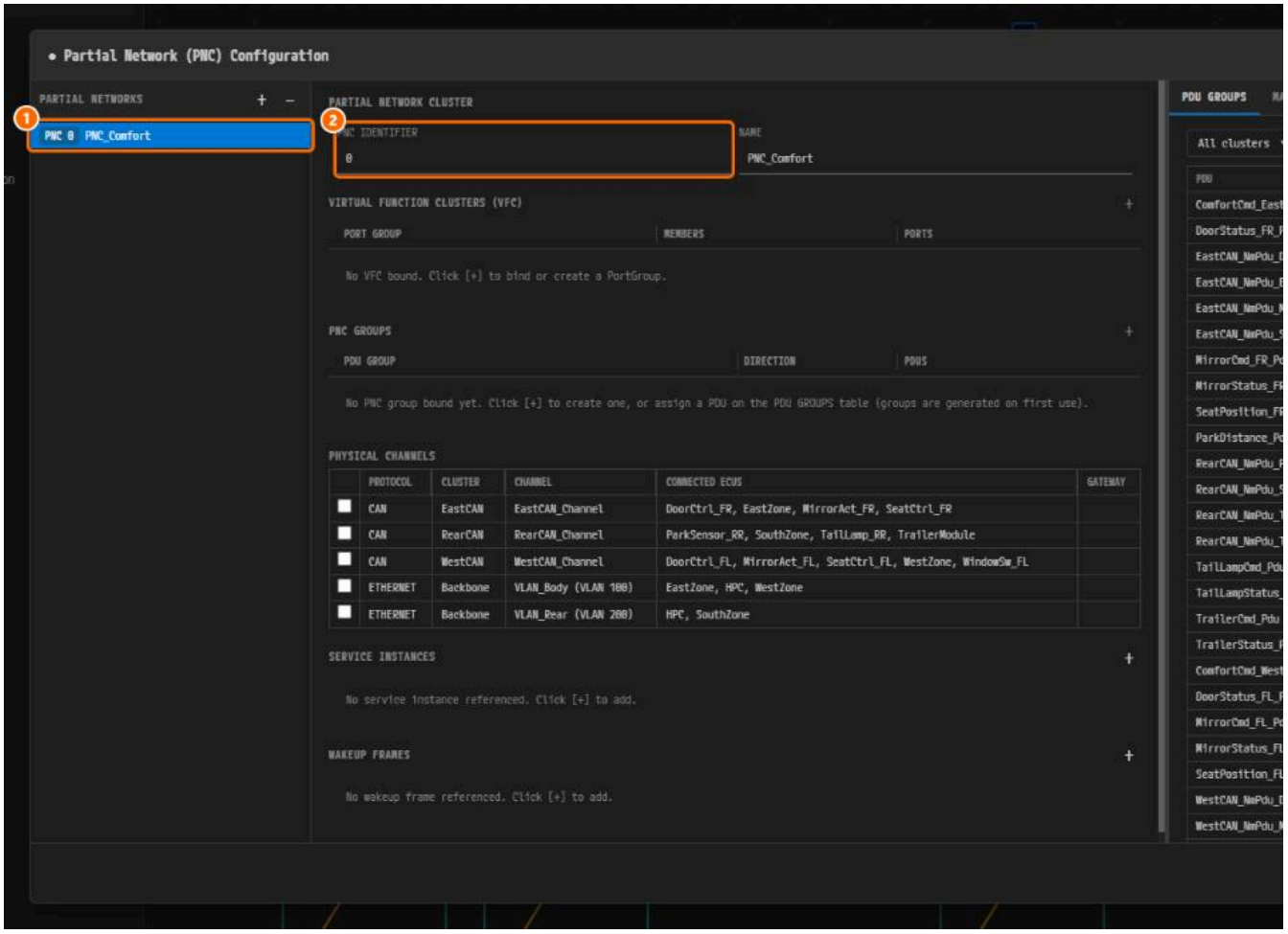


*Figure 3-5. ① **PNC 0 PNC_Comfort**, ② **PNC IDENTIFIER** left at `0`.*

3. Press `Ctrl+S` and click **Close**.

Checking and Fixing the Validation Error

Saving runs validation over the whole project. The identifier `0` is invalid, so it reports two errors.

1. Click the validation icon (**V**) in the status bar. It carries a red badge `2`. The **VALIDATION** panel opens.

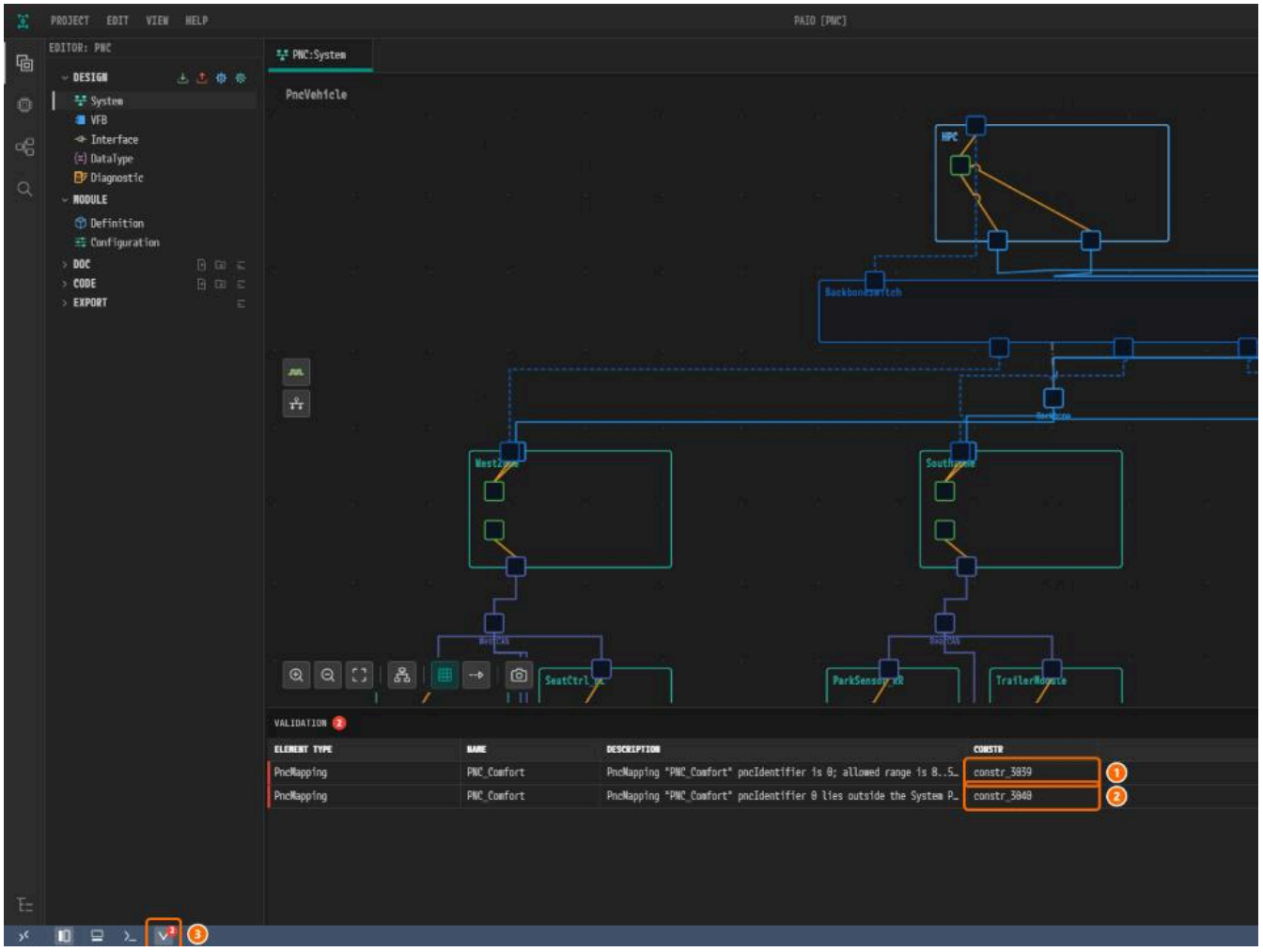


Figure 3-6. ① `constr_3039`, ② `constr_3040`, ③ the validation icon.



Figure 3-7. ① `constr_3039`: the identifier must be in `8`..`511`. ② `constr_3040`: it must lie in the PNC vector bits `16`..`63`.

2. Click the `constr_3040` row. The PNC dialog opens with the faulty entry marked in red.

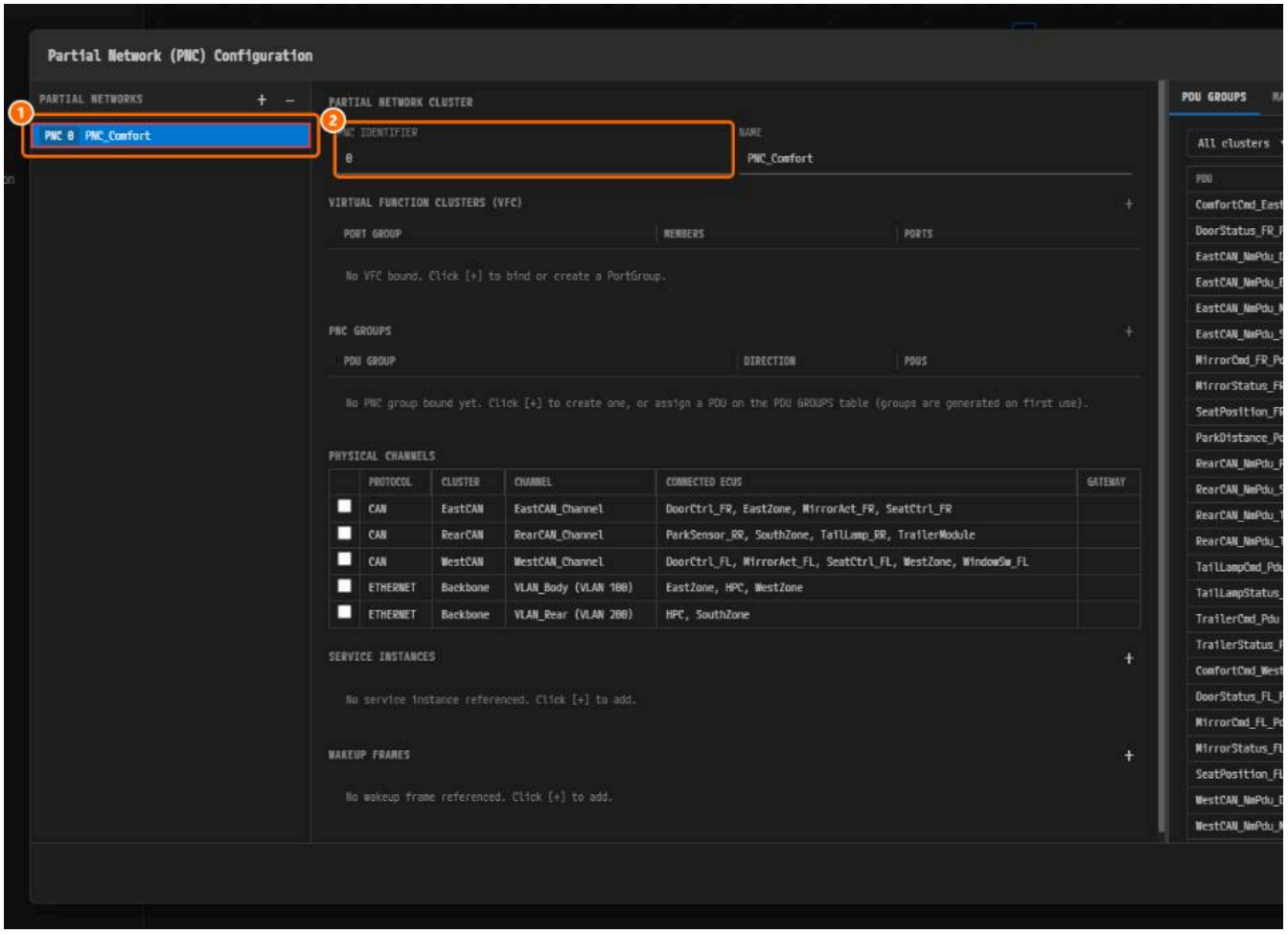


*Figure 3-8. ① The faulty PNC entry, ② **PNC IDENTIFIER**.*

3. Enter `17` in **PNC IDENTIFIER** and press `Enter`.

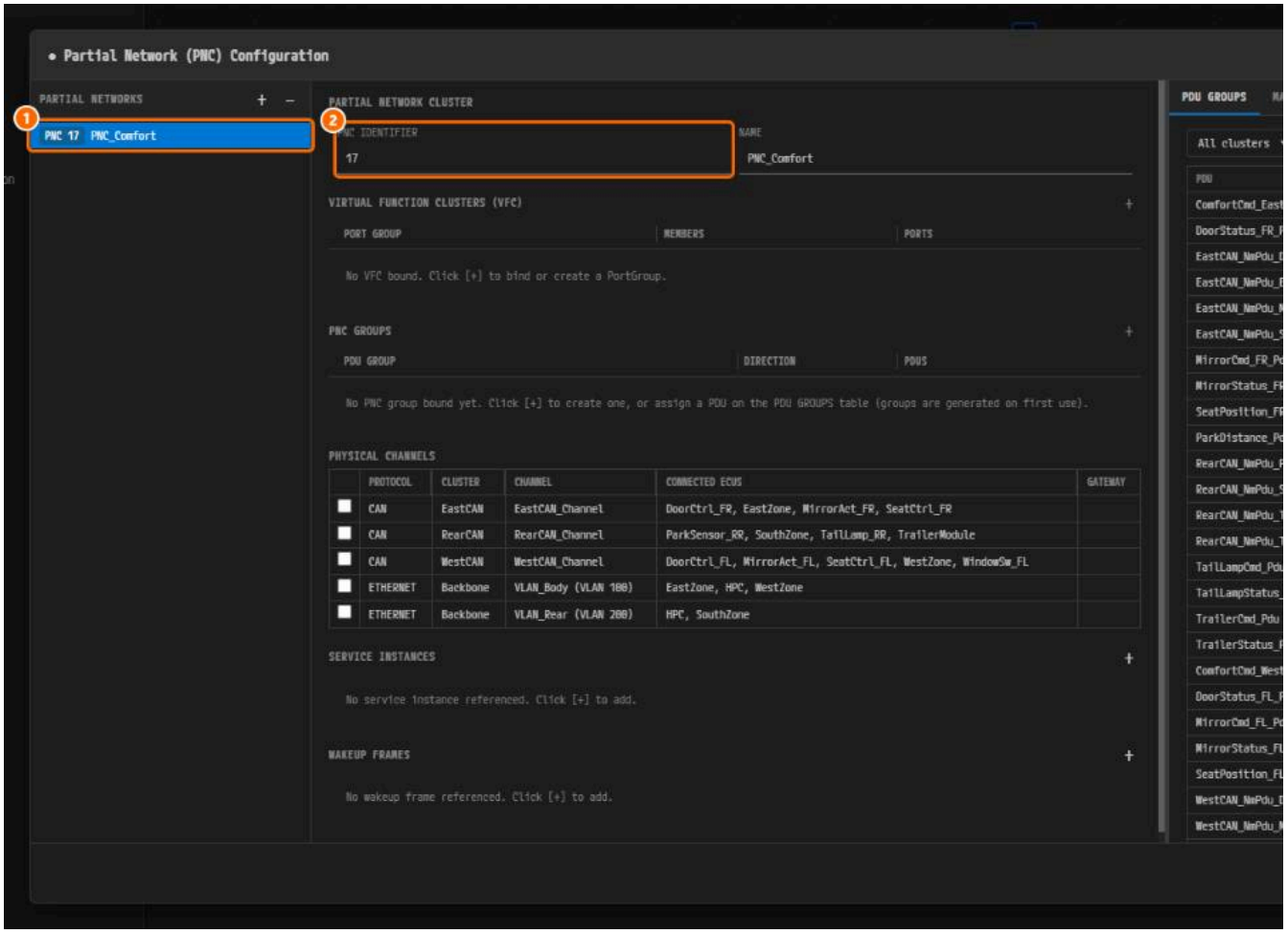


*Figure 3-9. ① **PNC 17 PNC_Comfort**, ② **PNC IDENTIFIER** = `17`.*

4. Press `Ctrl+S` and click **Close**. The **VALIDATION** panel shows "No validation issues."

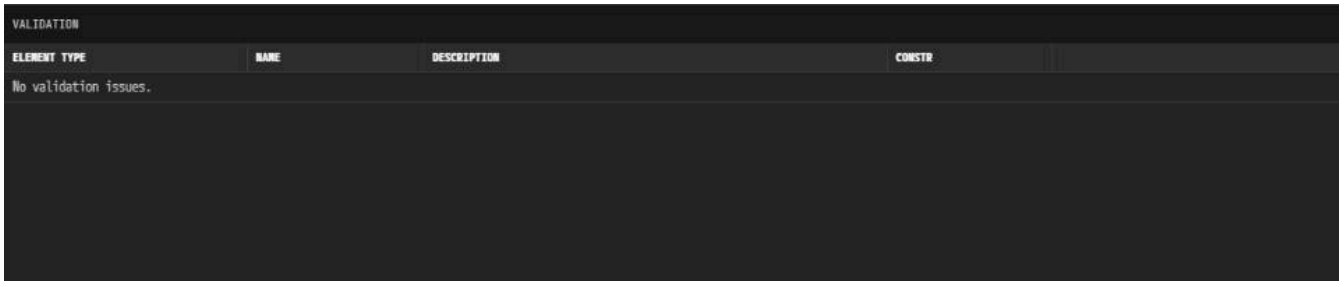


Figure 3-10. "No validation issues."

Adding the Remaining PNCs

1. Click the PNC button (Figure 3-1) to reopen the dialog.
2. Click **[+]**. Enter **PNC IDENTIFIER** `18` and **NAME** `PNC_ParkAssist`, each followed by `Enter`.
3. Add `19` `PNC_Trailer` the same way, and press `Ctrl+S`.

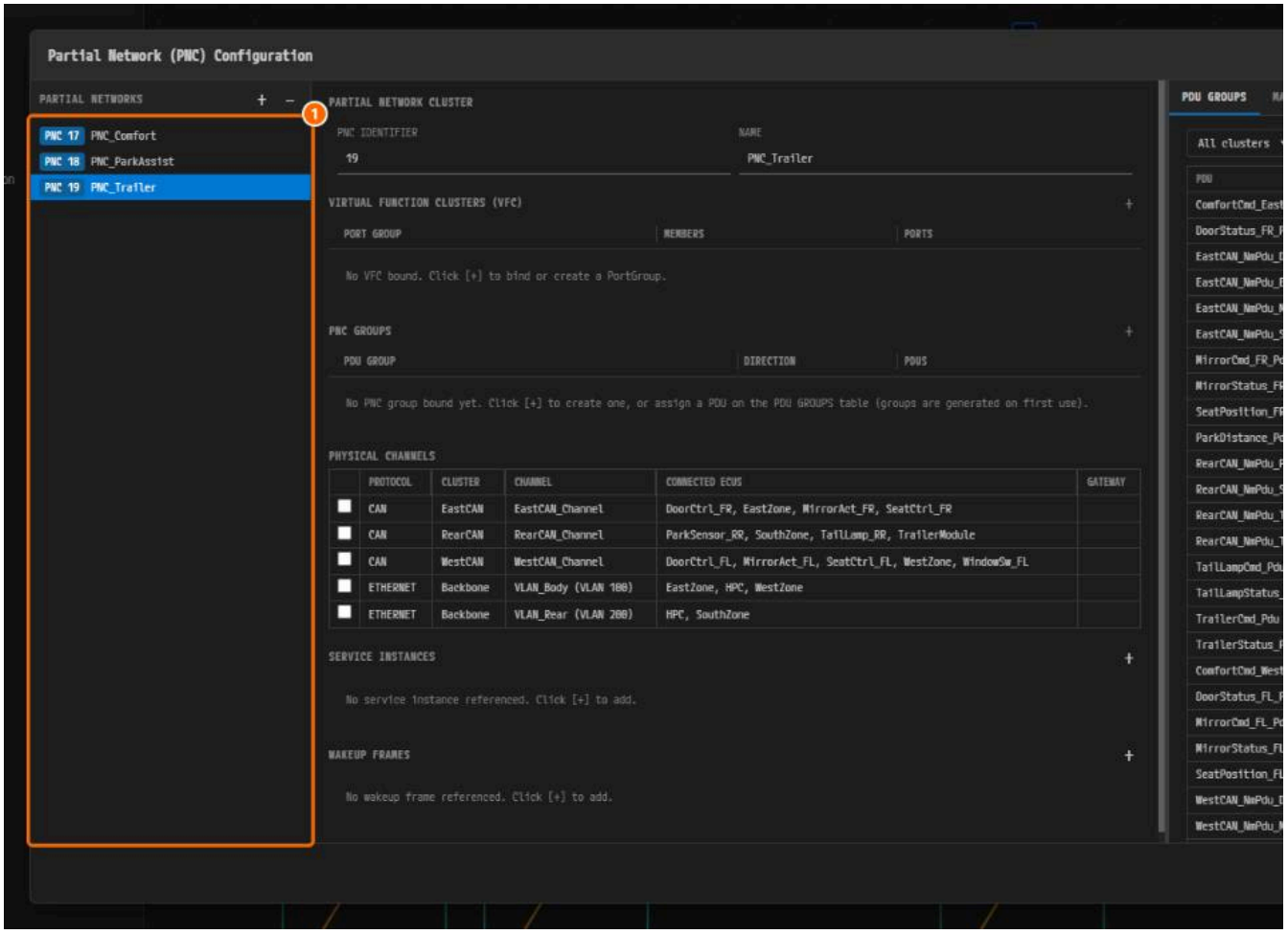


*Figure 3-11. ① **PNC 17 PNC_Comfort, PNC 18 PNC_ParkAssist, PNC 19 PNC_Trailer.***

4. Click **Close**. Validation still reports no errors.

Setting the PNC Bit Vector

1. Reopen the PNC dialog and click the **NM VECTOR** tab.

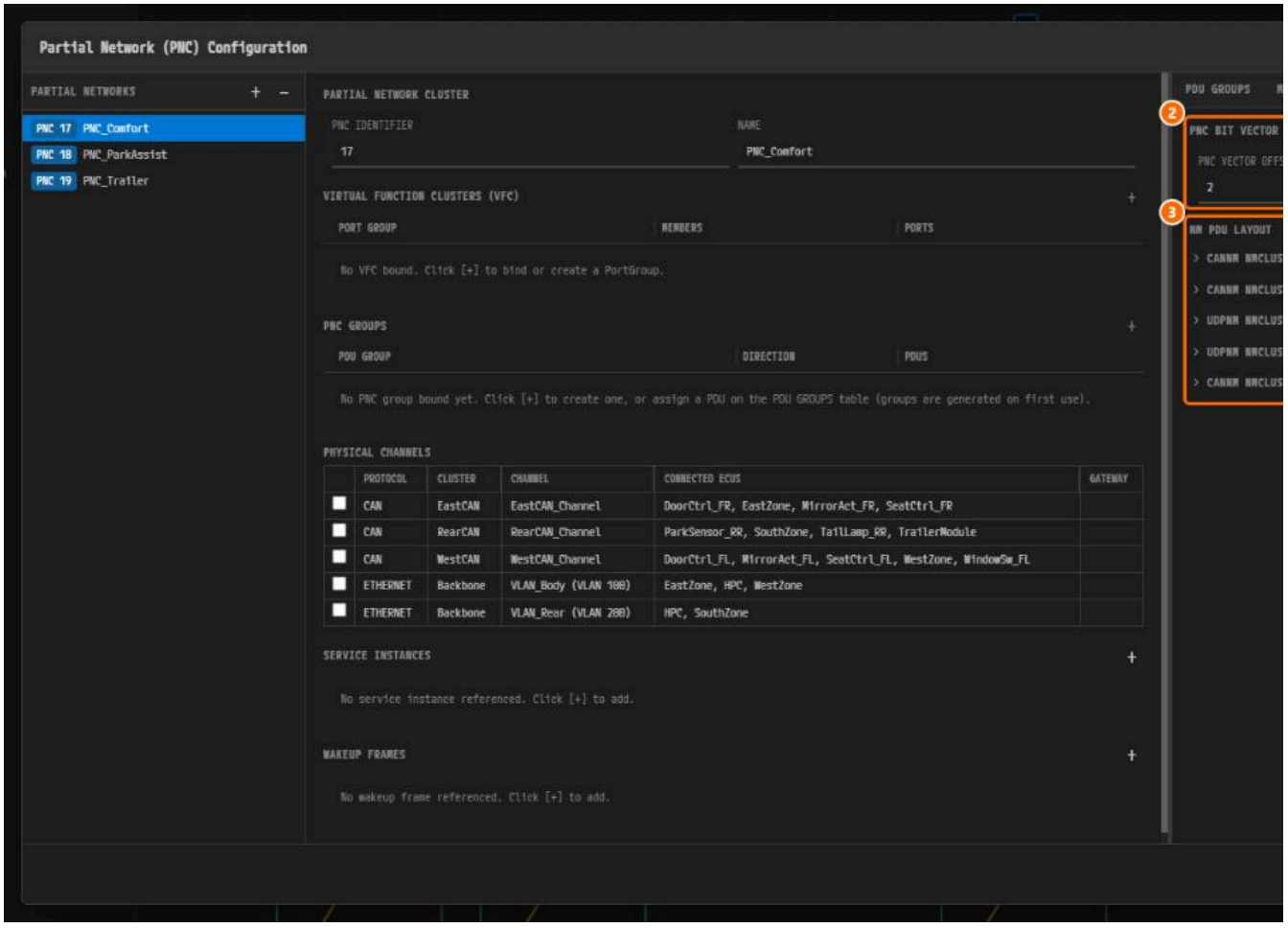


*Figure 3-12. ① **NM VECTOR**, ② **PNC BIT VECTOR**, ③ **NM PDU LAYOUT**.*

2. Check that **PNC VECTOR OFFSET** is `2` and **PNC VECTOR LENGTH** is `6`. Enter a value that differs or is empty.



*Figure 3-13. ① **PNC VECTOR OFFSET** = `2`, ② **PNC VECTOR LENGTH** = `6`.*

3. In **NM PDU LAYOUT**, expand `CANNM NMCLUSTER_WESTCAN (WESTCAN)`. In your project the cluster headers appear only after you turn NM on in NM Conf

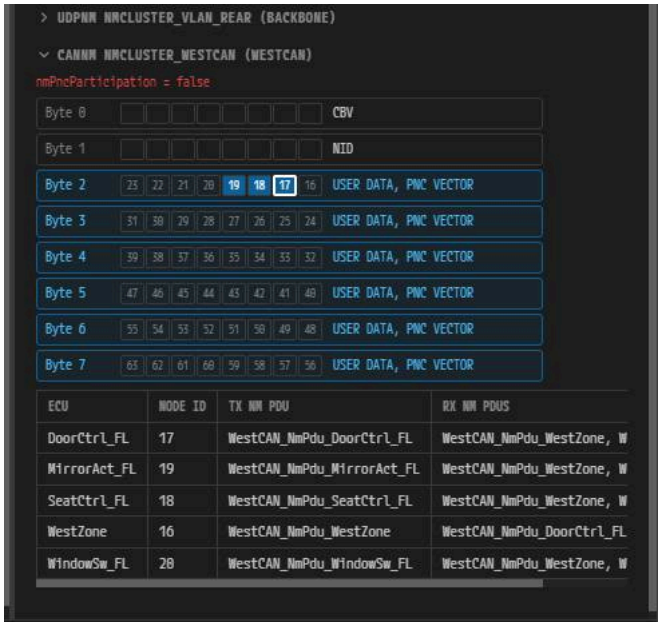


*Figure 3-14. Byte 0 = **CBV**, Byte 1 = **NID**, Byte 2 to 7 = **USER DATA, PNC VECTOR**. Bits `17`, `18`, `19` in Byte 2 are filled.*

4. (Optional) Change **PNC VECTOR LENGTH** to `4`. The **PNC VECTOR** shrinks to Byte 2 to 5.

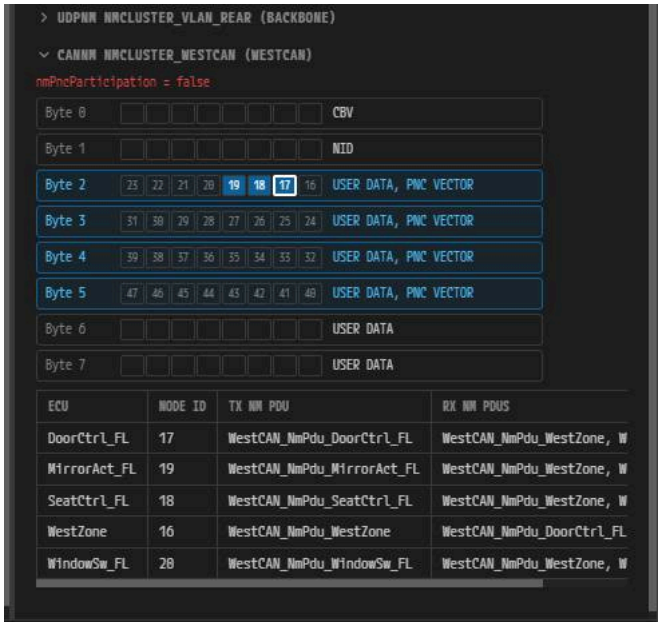


*Figure 3-15. **PNC VECTOR LENGTH** = `4`.*

5. Set **PNC VECTOR LENGTH** back to `6`, press `Ctrl+S`, and click **Close**.

Entering PNC Attributes on Connectors

The gateway role, the wakeup, and the filter mask belong to a connector, not to an ECU. Follow `WestZone_CanFd_Conn`, then enter the others from Conne

Entering WestZone_CanFd_Conn

1. Click the connector `WestZone_CanFd_Conn` on the lower edge of `WestZone`. If the properties panel is collapsed, click the top button on the right ed

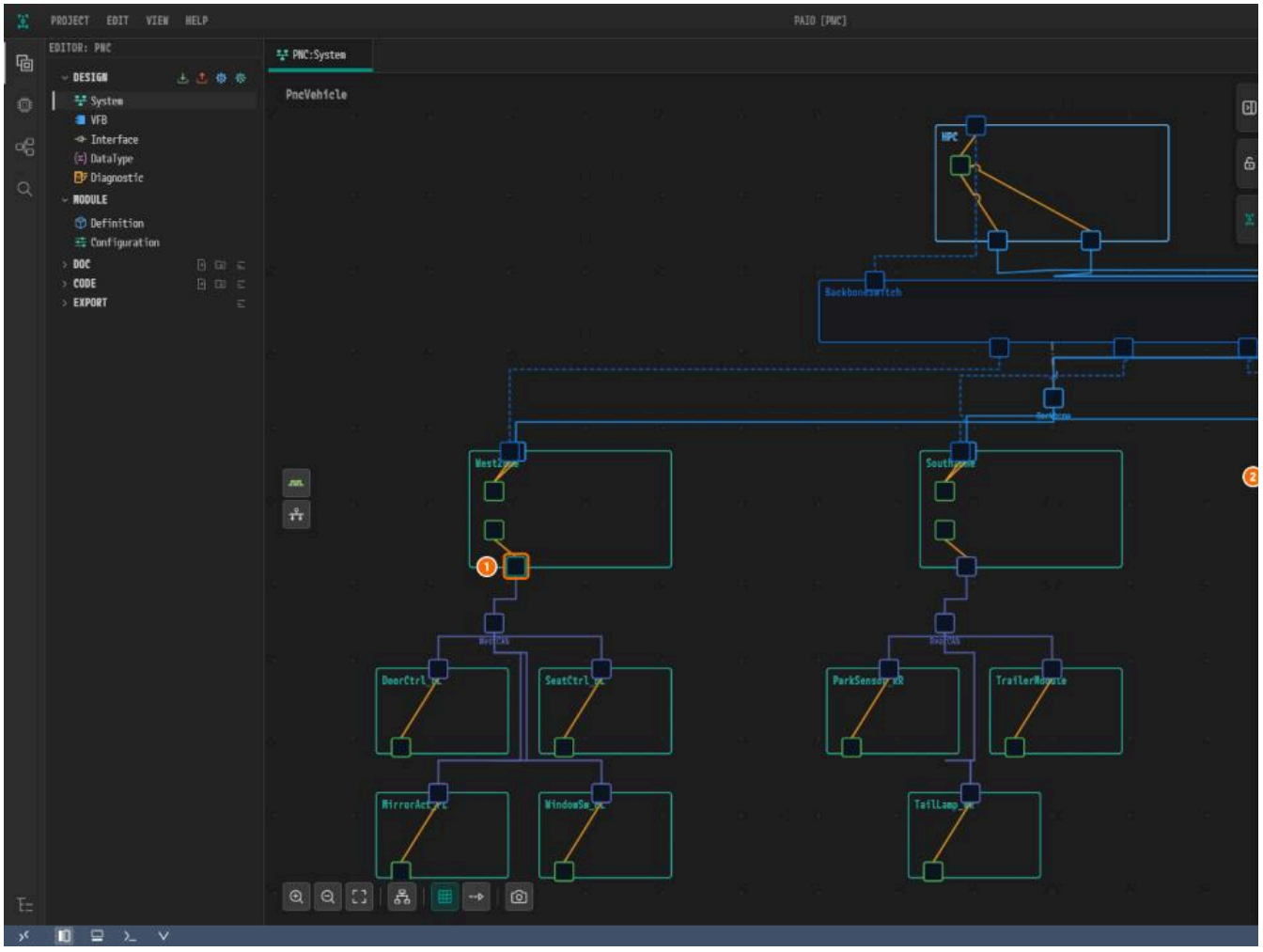


*Figure 3-16. ① `WestZone_CanFd_Conn`, ② the **PARTIAL NETWORK** section.*

2. In **PNC GATEWAY TYPE**, click **ACTIVE**.

3. Turn on **CREATE ECU WAKEUP SOURCE**.

4. Enter `500` in **PNC WAKEUP CAN ID**. The field is in **HEX**, so it shows `0x500`.

5. Enter `700` in **PNC WAKEUP CAN ID MASK** and `8` in **PNC WAKEUP DLC**.

6. Click Byte 0 of **PNC FILTER ARRAY MASKS**, enter `02`, and press `Enter`. Empty cells are saved as `0x00`.



*Figure 3-17. ① **ACTIVE**, ② **CREATE ECU WAKEUP SOURCE**, ③ `0x500`, ④ `0x700`, ⑤ `8`, ⑥ Byte 0 = `0x02`.*

7. Leave the other fields empty or off, and press `Ctrl+S`.

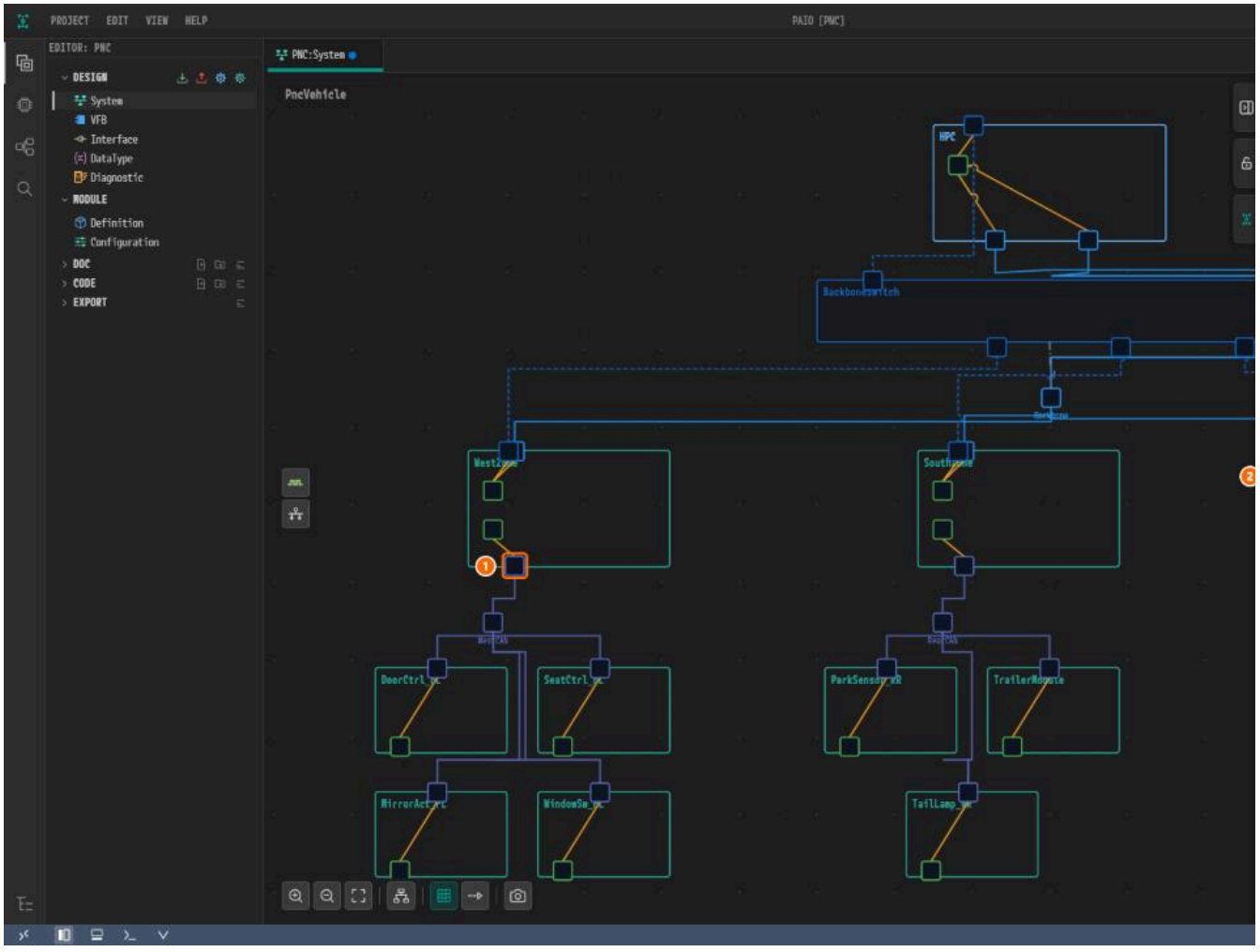


*Figure 3-18. ① The connector, ② **PARTIAL NETWORK**, before saving.*

Calculating the Filter Mask

The filter mask marks which PNC bits pass this connector.

```
byteIndex = quotient of pncId / 8 - pncVectorOffset    # 17 / 8 = 2, 2 - 2 = 0 → Byte 0
bit       = remainder of pncId / 8                     # 17 mod 8 = 1
value     = 2 ^ bit                                    # 2 ^ 1 = 0x02
```

For several PNCs, OR the values in the same byte: `0x04` | `0x08` = `0x0C`. The array has `pncVectorLength` (6) bytes (`constr_3686`).

Ethernet Connectors and HPC Connectors

An Ethernet connector has no wakeup CAN ID fields. An `HPC` connector has no **PNC GATEWAY TYPE** either, because only CP ECUs have it; enter only the filt

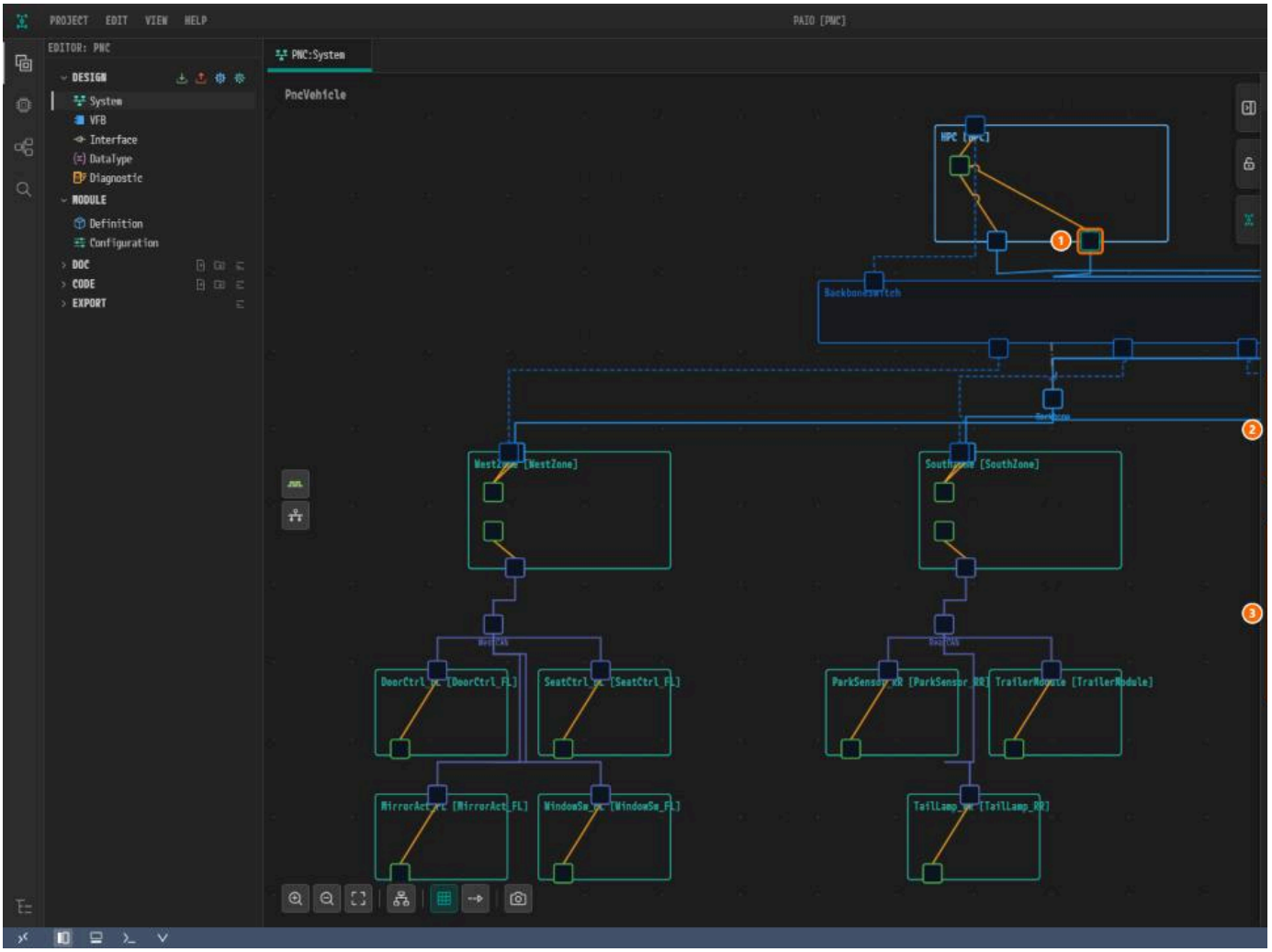


*Figure 3-19. `HPC_Eth_Rear_Conn` in the finished example. ① The connector, ② **SERVICE INSTANCES**, ③ **PARTIAL NETWORK**.*

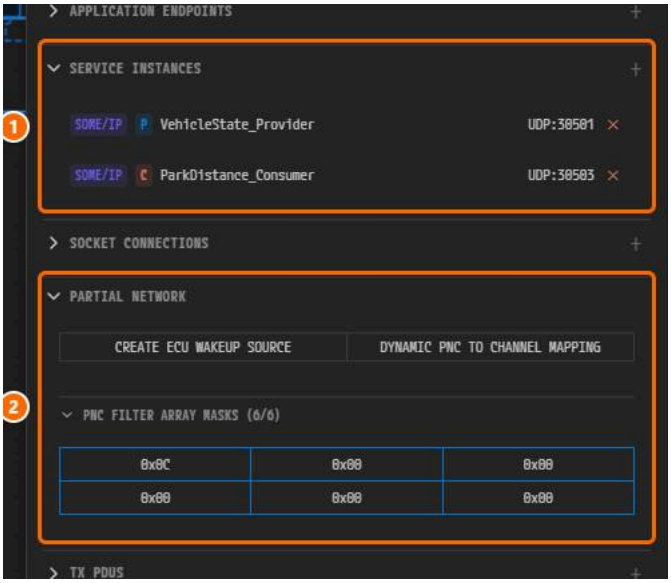


*Figure 3-20. ① **SERVICE INSTANCES**, made in Interfaces and SOME/IP Services. ② **PARTIAL NETWORK**: masks `0x0C 0x00 0x00 0x00 0x00 0x00`, no gateway*

VLAN_Body has only one active gateway, so `EastZone_Eth_Conn` is **PASSIVE**.

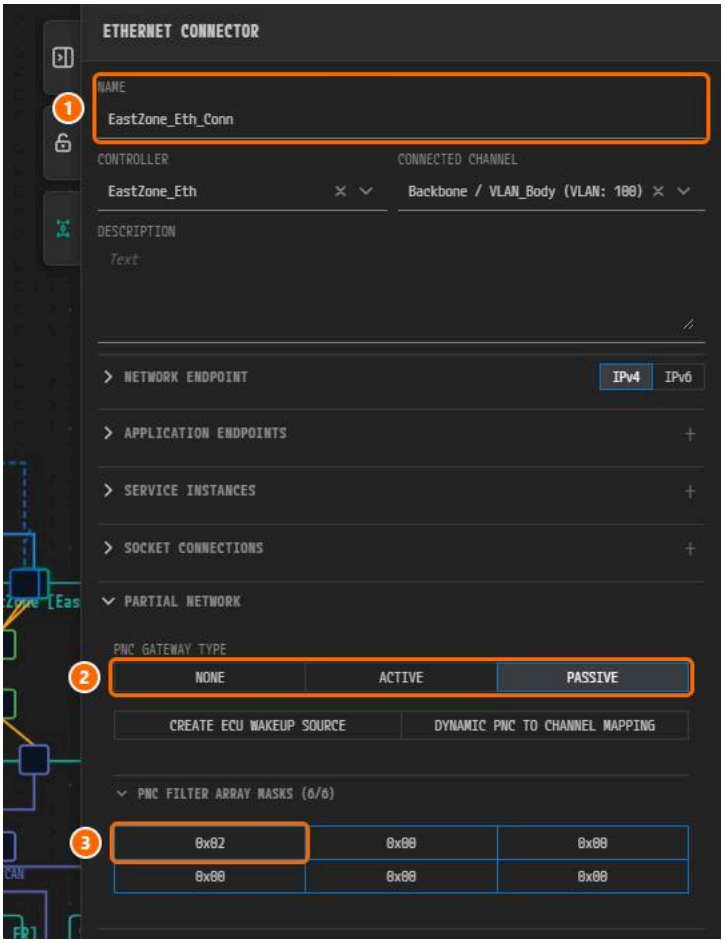


*Figure 3-21. ① `EastZone_Eth_Conn`, ② **PASSIVE**, ③ Byte 0 = `0x02`.*

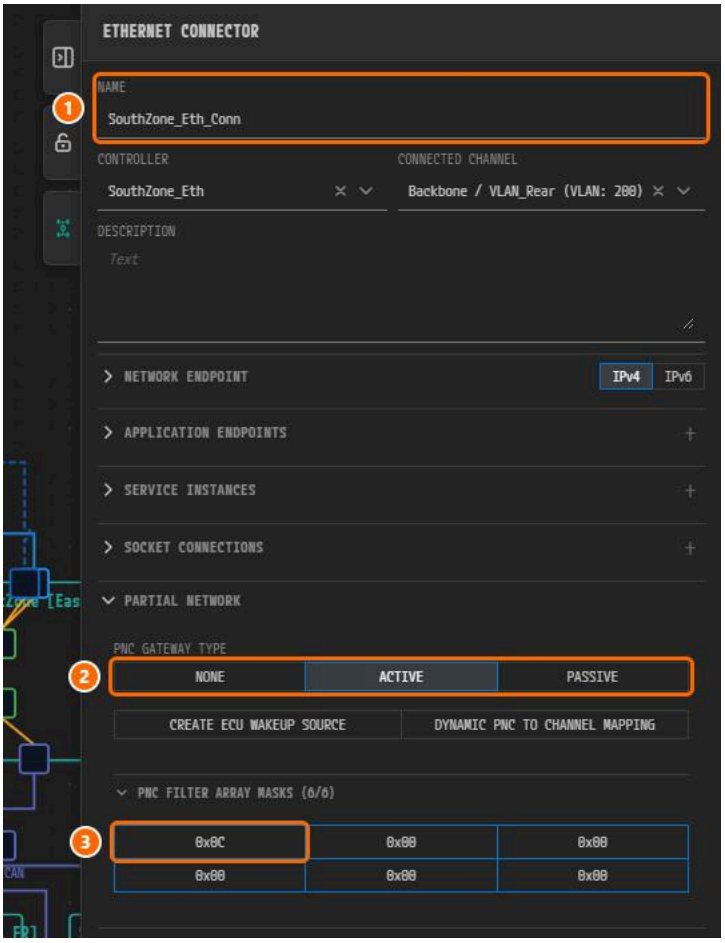


*Figure 3-22. ① `SouthZone_Eth_Conn`, ② **ACTIVE**, ③ Byte 0 = `0x0C`.*

> **Tip.** The expanded properties panel covers the `EastZone` side of the canvas. Collapse it or drag the canvas before you click a connector there.

Connector Values in Full

Only Byte 0 is listed; Byte 1 to 5 are `0x00`. "—" means empty or off.

Connector	Channel	PNC GATEWAY TYPE	CREATE ECU WAKEUP SOURCE	WAKEUP CAN ID / MASK / DLC	Byte 0	
`WestZone_CanFd_Conn`	WestCAN	**ACTIVE**	on	`0x500` / `0x700` / `8`	`0x02`	Th
`EastZone_CanFd_Conn`	EastCAN	**ACTIVE**	on	`0x500` / `0x700` / `8`	`0x02`	Th
`SouthZone_CanFd_Conn`	RearCAN	**ACTIVE**	on	`0x500` / `0x700` / `8`	`0x0C`	Th
`WestZone_Eth_Conn`	VLAN_Body	**ACTIVE**	—	(no such fields)	`0x02`	To
`EastZone_Eth_Conn`	VLAN_Body	**PASSIVE**	—	(no such fields)	`0x02`	W((c
`SouthZone_Eth_Conn`	VLAN_Rear	**ACTIVE**	—	(no such fields)	`0x0C`	To
`_CAN_Conn` of `DoorCtrl_FL`, `SeatCtrl_FL`, `MirrorAct_FL`, `WindowSw_FL`	WestCAN	**NONE**	on	—	`0x02`	No
`_CAN_Conn` of `DoorCtrl_FR`, `SeatCtrl_FR`, `MirrorAct_FR`	EastCAN	**NONE**	on	—	`0x02`	No
`ParkSensor_RR_CAN_Conn`	RearCAN	**NONE**	on	—	`0x04`	Or
`TrailerModule_CAN_Conn`, `TailLamp_RR_CAN_Conn`	RearCAN	**NONE**	on	—	`0x08`	Or
`HPC_Eth_Body_Conn`	VLAN_Body	(no such field)	—	(no such fields)	`0x02`	Ar
`HPC_Eth_Rear_Conn`	VLAN_Rear	(no such field)	—	(no such fields)	`0x0C`	Ar

Press `Ctrl+S` when all connectors are done.

Entering the PARTIAL NETWORK Values of a Node

A node also has a **PARTIAL NETWORK** section. It holds the NM request and the sleep timers.

1. Click the header of the `WestZone` node (not a connector).
2. In **PARTIAL NETWORK**, turn on **PNC NM REQUEST**. Leave **PNC SYNCHRONOUS WAKEUP** and **CHANNEL SYNCHRONOUS WAKEUP** off.
3. Enter `0.5` in **PNC PREPARE SLEEP TIMER** and `2` in **PN RESET TIME**. The default unit is **s**.

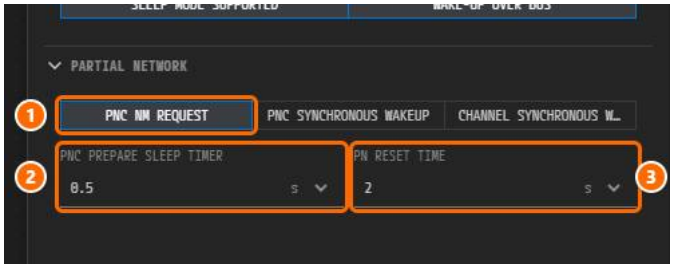


*Figure 3-23. ① **PNC NM REQUEST** on, ② `0.5` s, ③ `2` s.*

4. Enter the other nodes from the table and press `Ctrl+S`.

Node	PNC NM REQUEST	PNC SYNCHRONOUS WAKEUP	PNC PREPARE SLEEP TIMER	PN RESET TIME / PN RESET TIMER
`WestZone`, `SouthZone`, `EastZone`	on	off	`0.5` s	`2` s (**PN RESET TIME**)
The 10 end ECUs	on	off	`0.5` s	`2` s (**PN RESET TIME**)
`HPC`	(no such field)	(no such field)	`0.5` s	`2` s (**PN RESET TIMER**)

`HPC` is an AP machine, so it has only the two timer fields.



*Figure 3-24. ① **PNC PREPARE SLEEP TIMER** = `0.5` s, ② **PN RESET TIMER** = `2` s.*

Assigning PNCs to Switch Ports

PNC MAPPINGS on a switch port names the PNCs the port is used for.

1. Double-click the `BackboneSwitch` node. The **Ethernet Switch Configuration [BackboneSwitch]** workbench opens.
2. Click `Port_1` in the switch diagram.
3. Open **PNC MAPPINGS**, check `PNC_Comfort (PNC 17)`, and press `Esc`. If the list is empty, close and reopen the workbench.

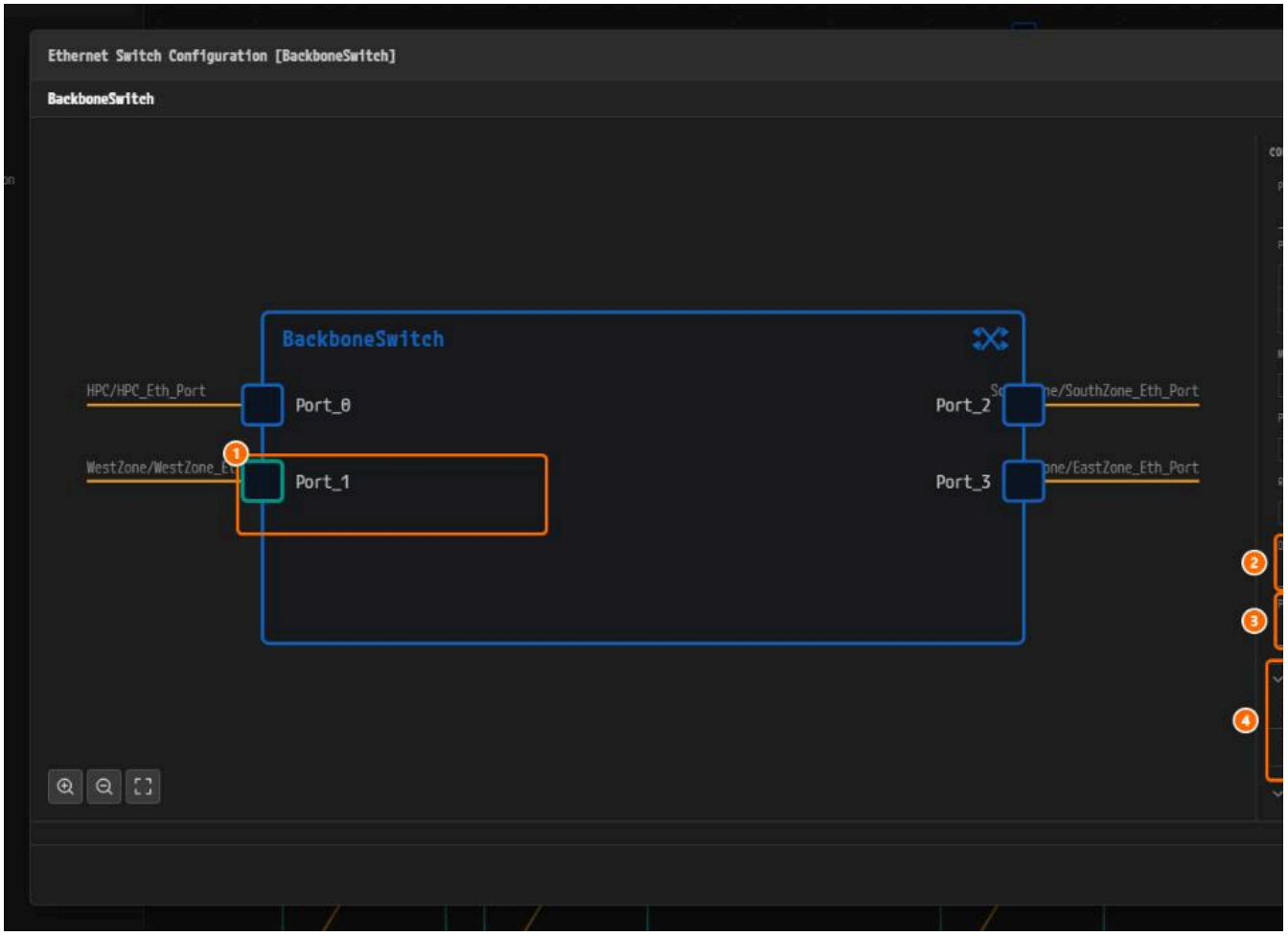


*Figure 3-25. ① `Port_1`, ② **DEFAULT VLAN (PVID)**, ③ **PNC MAPPINGS**, ④ **VLAN MEMBERSHIP**.*



*Figure 3-26. ① **DEFAULT VLAN (PVID)**, ② **PNC MAPPINGS**, ③ **VLAN MEMBERSHIP**.*

4. Assign the other ports from the table.

Port	Connects to	VLAN	PNC MAPPINGS
`Port_0`	HPC	`VLAN_Body`, `VLAN_Rear`	`PNC_Comfort`, `PNC_ParkAssist`, `PNC_Trailer`
`Port_1`	WestZone	`VLAN_Body`	`PNC_Comfort`
`Port_2`	SouthZone	`VLAN_Rear`	`PNC_ParkAssist`, `PNC_Trailer`
`Port_3`	EastZone	`VLAN_Body`	`PNC_Comfort`

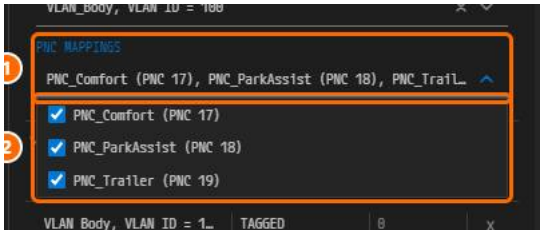


Figure 3-27. ① The closed field, ② all three PNCs checked on `Port_0`.

5. Press `Ctrl+S` in the workbench and click **Close**.

6. Press `Ctrl+S` in the System editor.

> **Note.** The finished example also assigns the PNCs on the Ethernet controller ports: all three on `HPC_Eth_Port`, `PNC_Comfort` on `WestZone_Eth_Port` a `PNC_Trailer` on `SouthZone_Eth_Port`.

NM Configuration

The PNC bits travel in NM PDUs. So each NM cluster must have **NM PNC PARTICIPATION** = **YES** and its mandatory timing values, and each NM node needs an NM clusters and NM nodes from the topology. You turn NM on per cluster and fill in values. Follow `NmCluster_WestCAN` and `WestZone`, then the NM Value

Finding the Missing Mandatory NM Attributes

When the mandatory attributes of an NM cluster are empty, validation reports four errors.

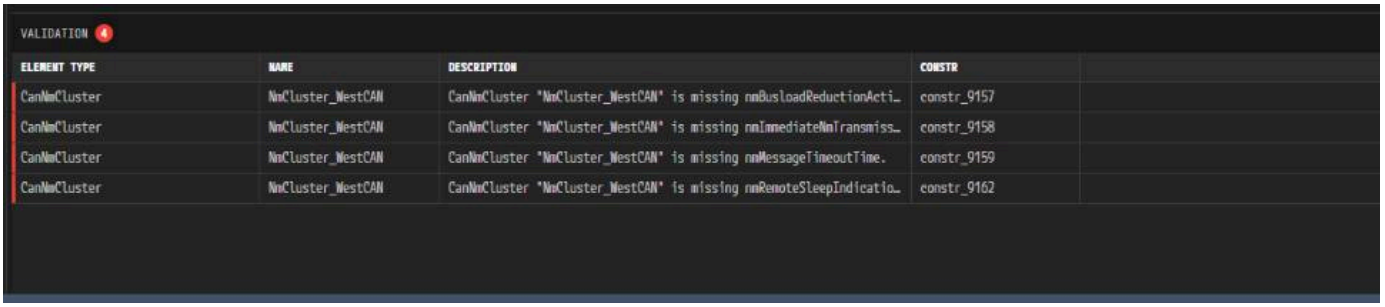


Figure 3-28. `constr_9157`, `constr_9158`, `constr_9159`, `constr_9162` on `NmCluster_WestCAN`.

They are **NM BUSLOAD REDUCTION ACTIVE**, **NM IMMEDIATE NM TRANSMISSIONS**, **NM MESSAGE TIMEOUT TIME**, and **NM REMOTE SLEEP INDICATION TIME**. In your see these errors.

Filling In NmCluster_WestCAN

1. Double-click the `WestCAN` cluster node below `WestZone`.

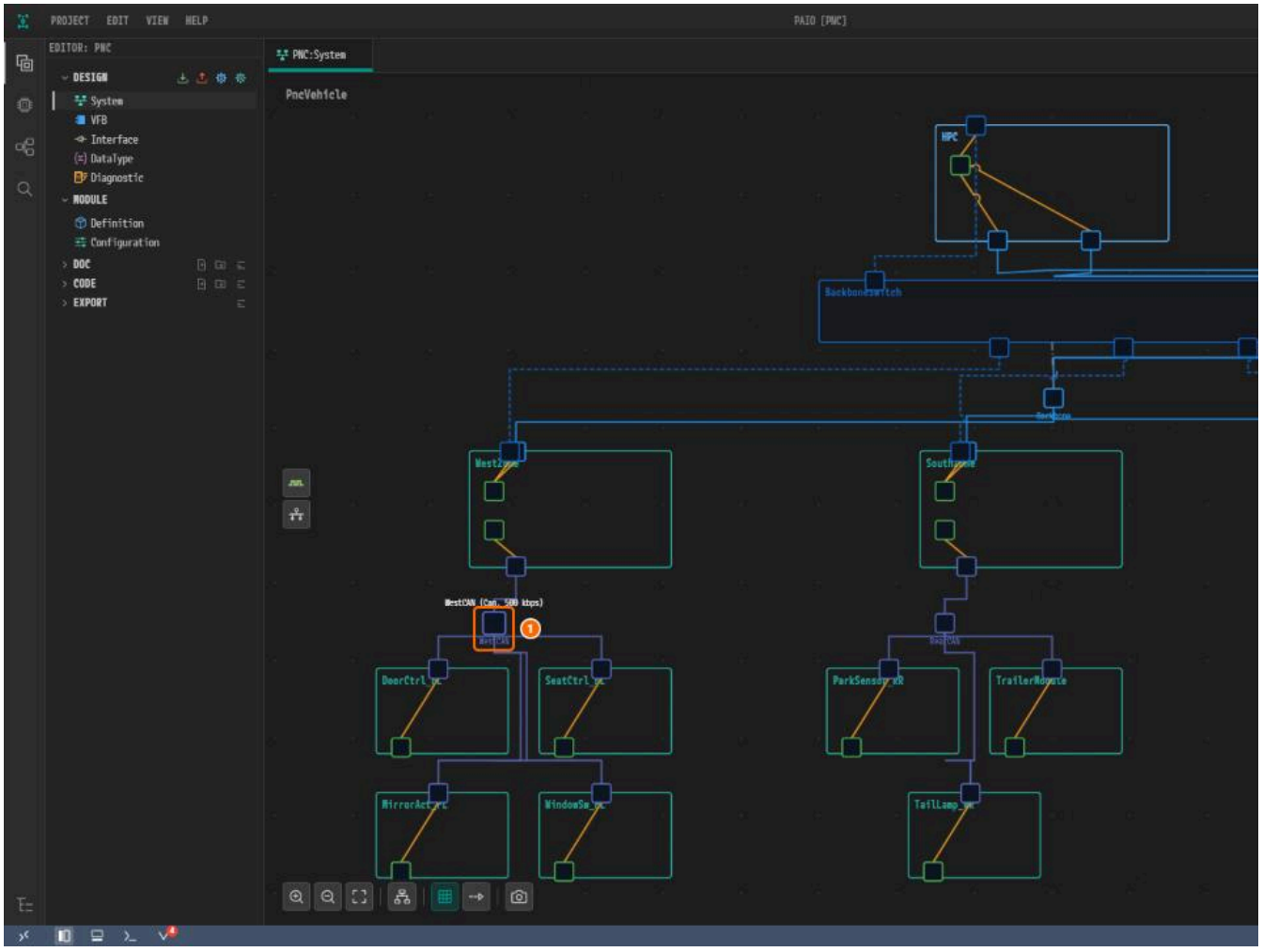


Figure 3-29. ① The `WestCAN` cluster node.

2. In **Communication Cluster Configuration [WestCAN]**, click the **CAN NM CLUSTER** tab. If the tab reads **NM CLUSTER** with a hollow dot, right-click it and c

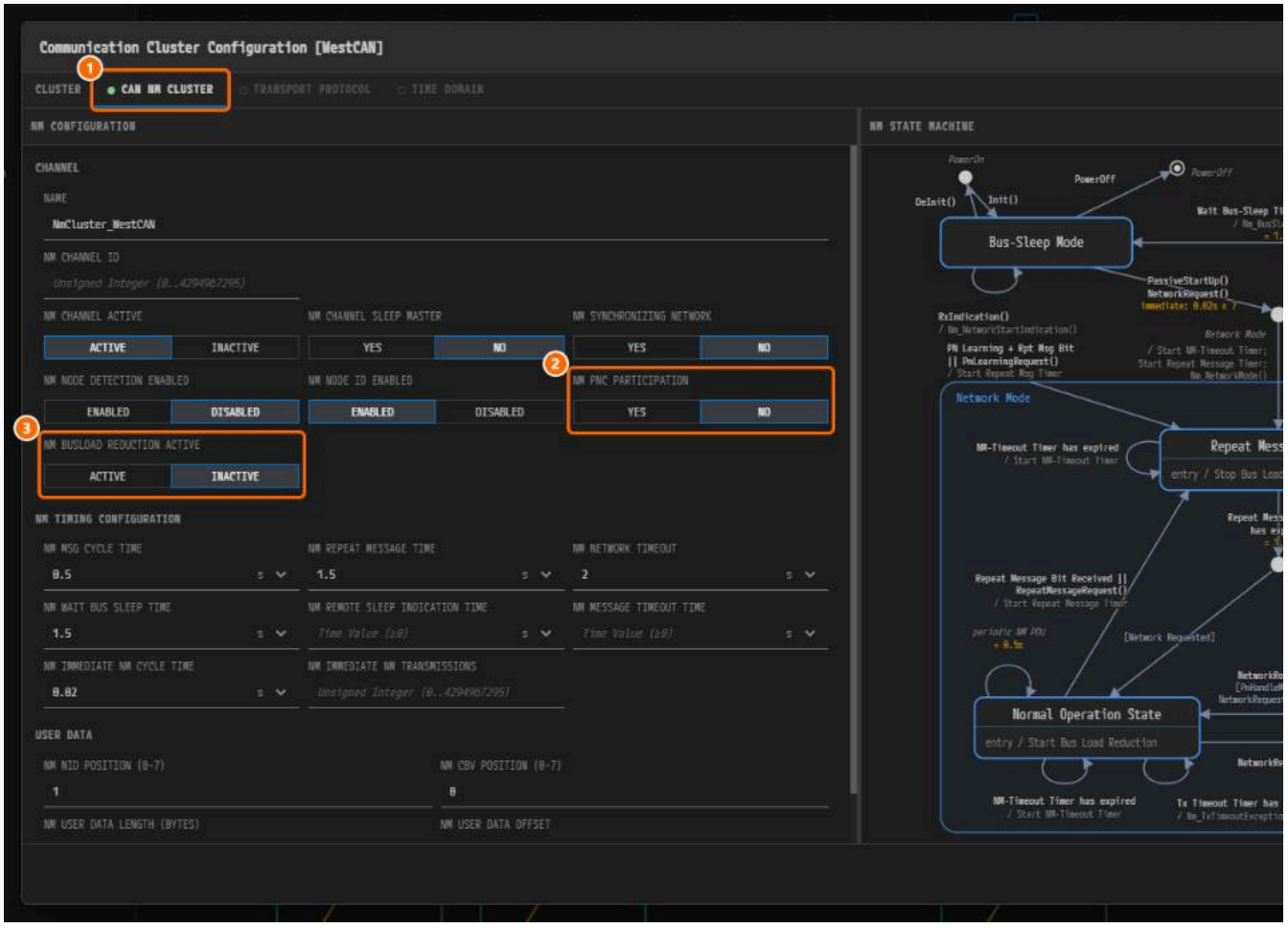


*Figure 3-30. ① **CAN NM CLUSTER**, ② **NM PNC PARTICIPATION**, ③ **NM BUSLOAD REDUCTION ACTIVE**.*

3. Set **NM PNC PARTICIPATION** to **YES**.

4. Click **NM BUSLOAD REDUCTION ACTIVE** to **ACTIVE**, then back to **INACTIVE**.

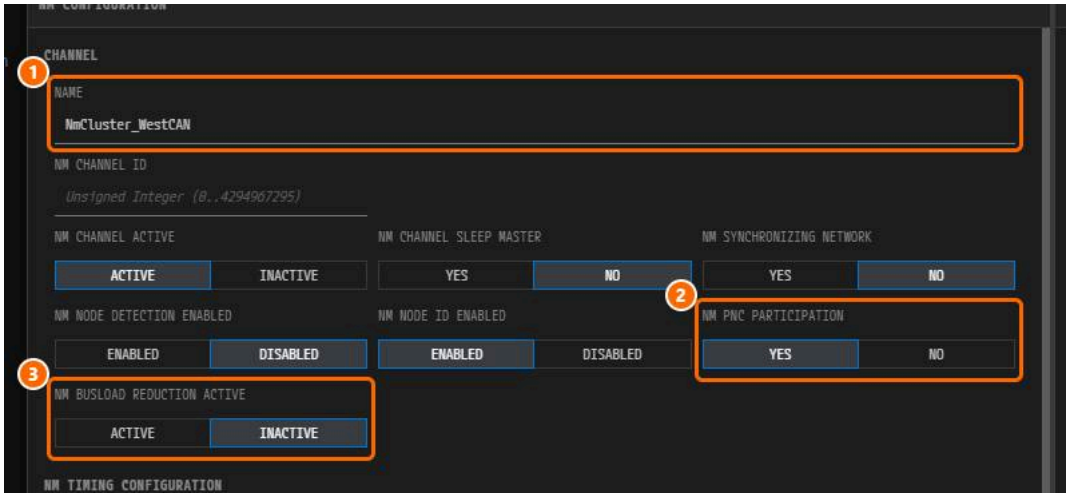


*Figure 3-31. ① `NmCluster_WestCAN`, ② **YES**, ③ **INACTIVE**. Leave **NM CHANNEL ID** empty.*

> **Caution.** A two-state button with no value looks as if one side were pressed. Click the other side once and click back to store the value.

5. In **NM TIMING CONFIGURATION**, enter the values below. The default unit is **s**.

Field	Value
NM MSG CYCLE TIME	`0.5`
NM REPEAT MESSAGE TIME	`1.5`
NM WAIT BUS SLEEP TIME	`1.5`
NM NETWORK TIMEOUT	`2`
NM MESSAGE TIMEOUT TIME	`0.1`
NM REMOTE SLEEP INDICATION TIME	`1`
NM IMMEDIATE NM CYCLE TIME	`0.02`
NM IMMEDIATE NM TRANSMISSIONS	`5`

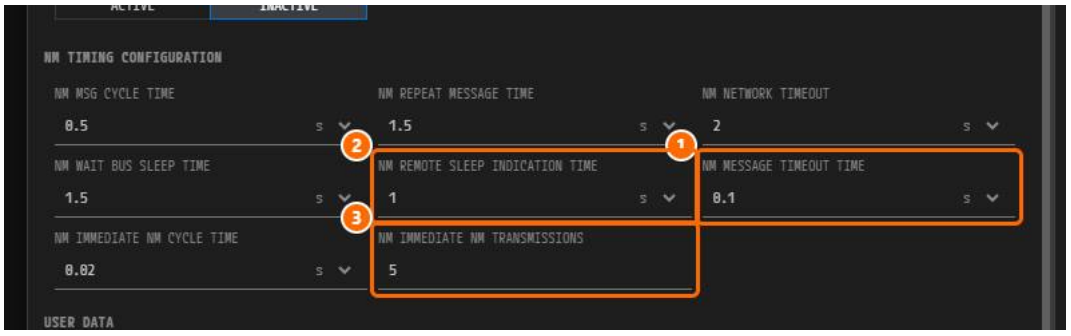


*Figure 3-32. ① **NM MESSAGE TIMEOUT TIME** = `0.1` s, ② **NM REMOTE SLEEP INDICATION TIME** = `1` s, ③ **NM IMMEDIATE NM TRANSMISSIONS** = `5`.*

> **Caution.** The defaults of **NM REPEAT MESSAGE TIME** (`1.0`), **NM MESSAGE TIMEOUT TIME** (`2`), and **NM IMMEDIATE NM TRANSMISSIONS** (`0`) differ from empty. Change them.

6. In **USER DATA** at the end of the form, enter **NM NID POSITION (0-7)** `1`, **NM CBV POSITION (0-7)** `0`, **NM USER DATA OFFSET** `2`, and **NM USER DATA LENGTH (**

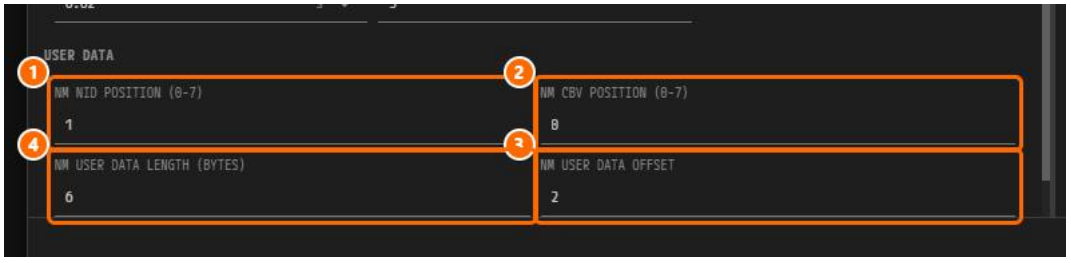


Figure 3-33. ① `1`, ② `0`, ③ `2`, ④ `6`. The last two match the PNC bit vector.

7. Press `Ctrl+S` and click **Close**.

Filling In the NM Node Values of WestZone

1. Double-click the `WestZone` node. In **ECU Configuration [WestZone]**, click the **NETWORK MANAGEMENT** tab.

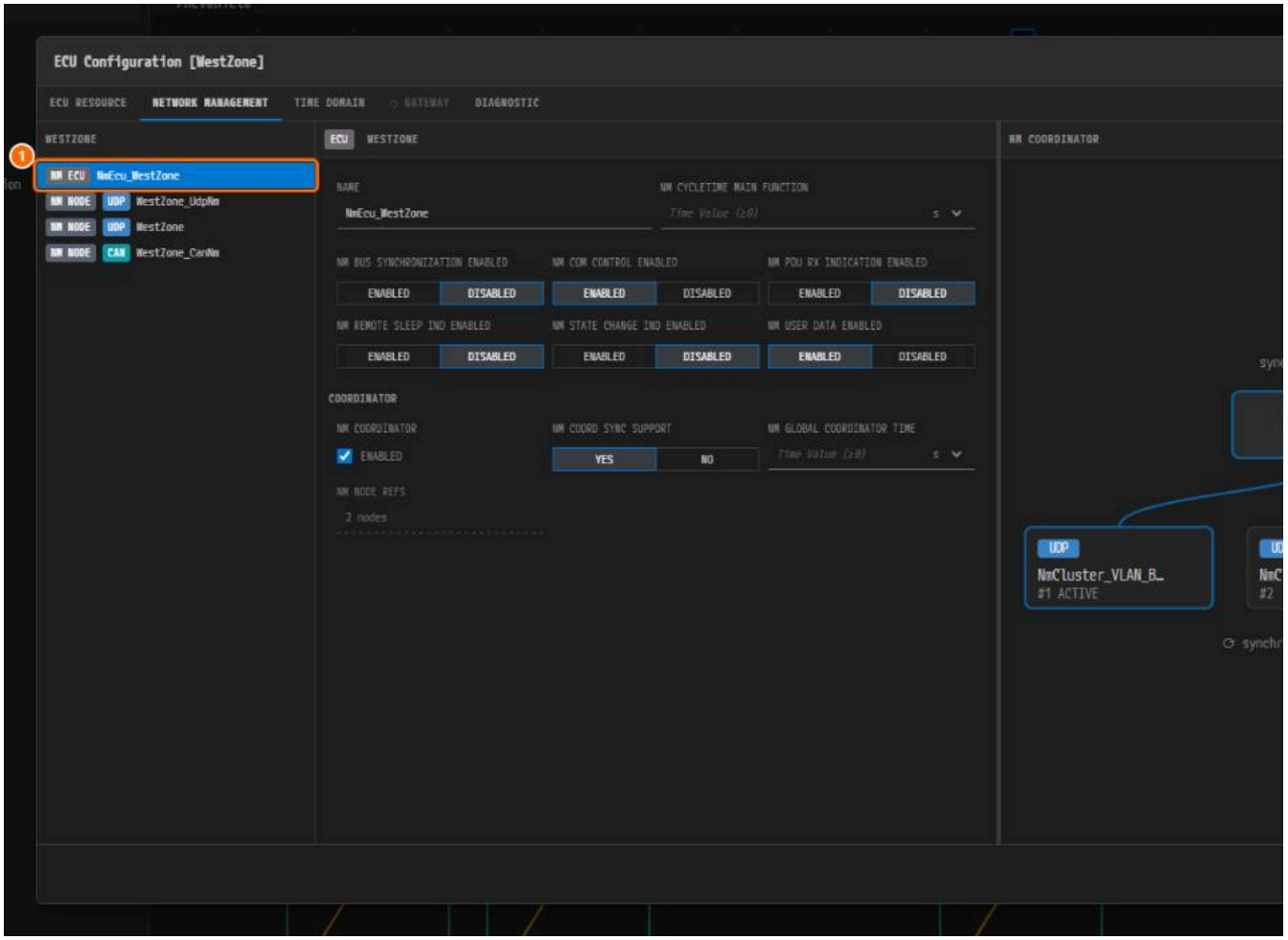


*Figure 3-34. ① The list of one **NM ECU** and the **NM NODE** entries with **CAN** or **UDP** badges.*

2. Select the NM node with the **CAN** badge. Enter `WestZone_CanNm` in **NAME**, `16` in **NM NODE ID**, and click **ACTIVE** in **NM COORDINATOR ROLE**.

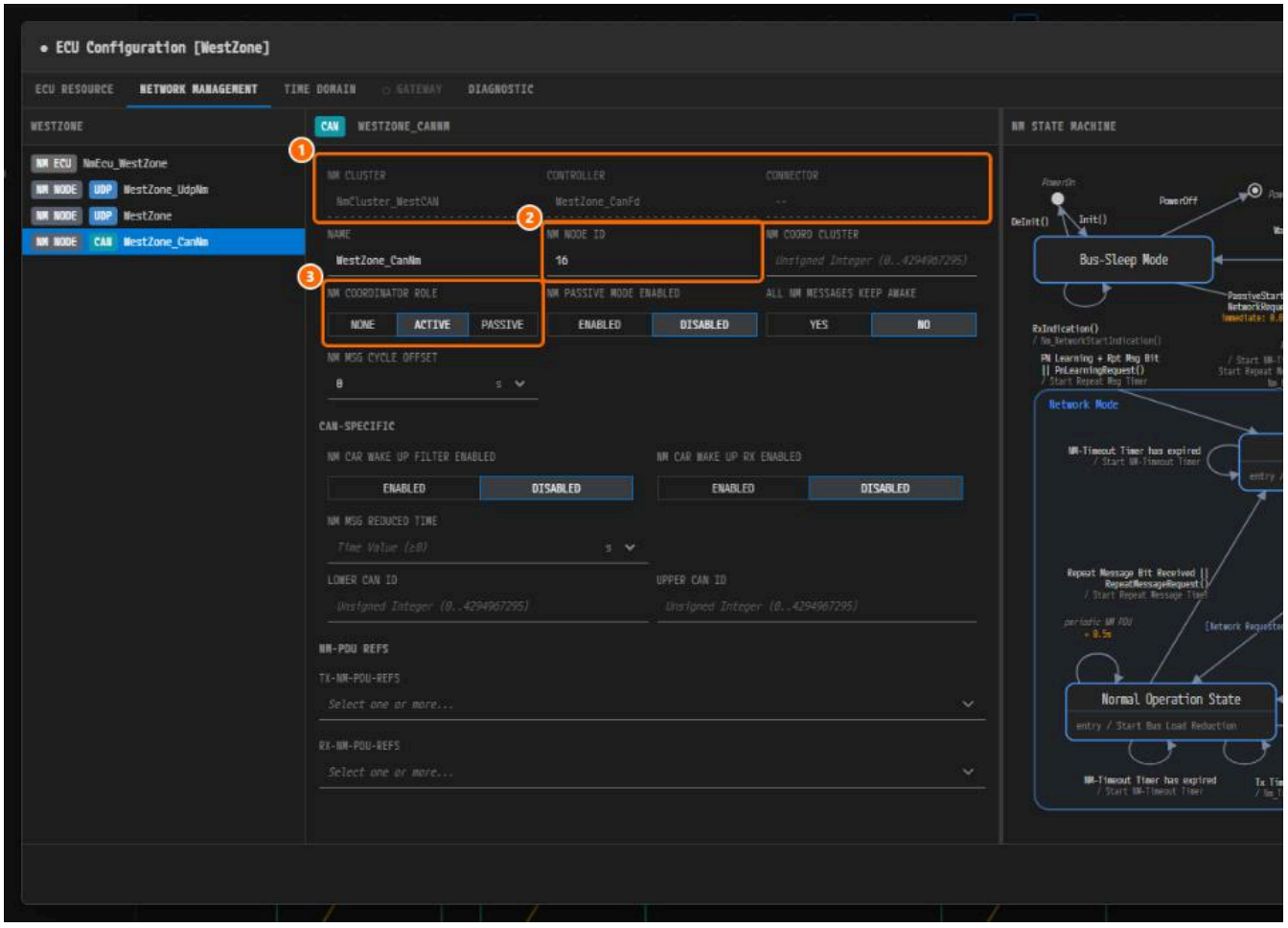


*Figure 3-35. ① Read-only rows, ② **NM NODE ID** = `16`, ③ **NM COORDINATOR ROLE** = **ACTIVE**.*

> **Caution.** Steps 3 and 4 need the NM-PDUs. Skip them now and come back after Creating the NM-PDUs.

3. Open **TX-NM-PDU-REFS**, check only `WestCAN_NmPdu_WestZone`, and press `Esc`.

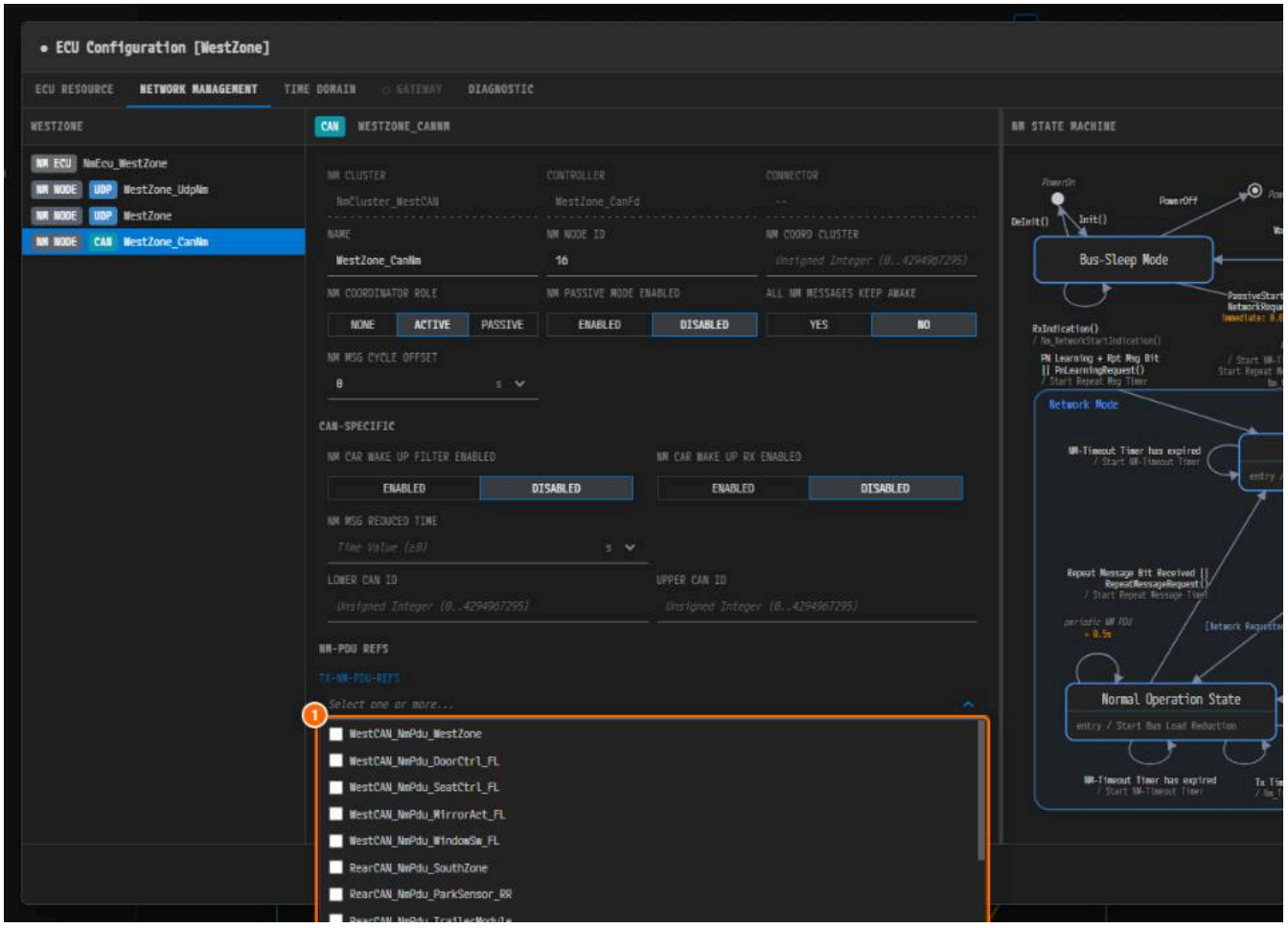


*Figure 3-36. ① The **TX-NM-PDU-REFS** list. Pick by the bus prefix `WestCAN_`.*

4. In **RX-NM-PDU-REFS**, check `WestCAN_NmPdu_DoorCtrl_FL`, `WestCAN_NmPdu_SeatCtrl_FL`, `WestCAN_NmPdu_MirrorAct_FL`, and `WestCAN_NmPdu_Wind`

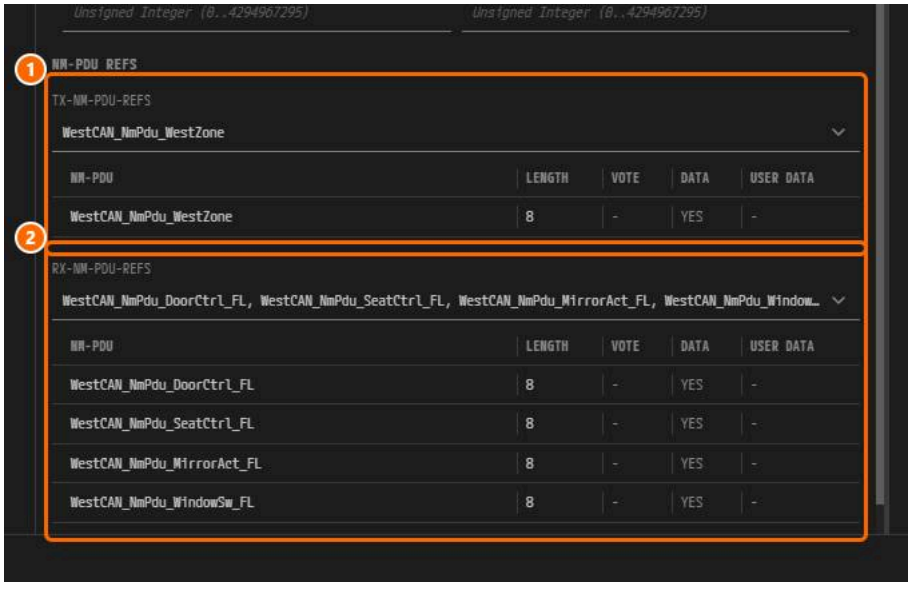


*Figure 3-37. ① One **TX-NM-PDU-REFS** (at most one, `constr_9388`), ② four **RX-NM-PDU-REFS**.*

5. Press `Ctrl+S` and click **Close**.

6. The **VALIDATION** panel no longer shows `constr_9157`, `constr_9158`, `constr_9159`, or `constr_9162`.

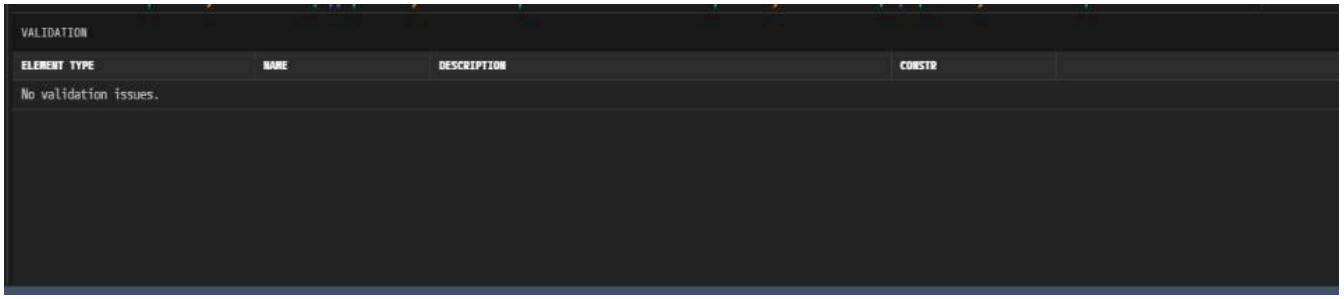


Figure 3-38. "No validation issues."

> **Caution.** `Ctrl+S` in the **Communication Cluster Configuration** dialog rewrites `IVC.io.model`. The VLAN configuration of `Backbone` is lost and a fe
> — Differences Between Your Own Project and the Finished Example.

Differences on the Ethernet (UdpNm) Side

1. Double-click the `Backbone` cluster node and click **UDP NM CLUSTER** (**Enable** it if needed). Enter the same values as for CAN, but leave **NM IMMEDIATE TRANSMISSIONS** empty.

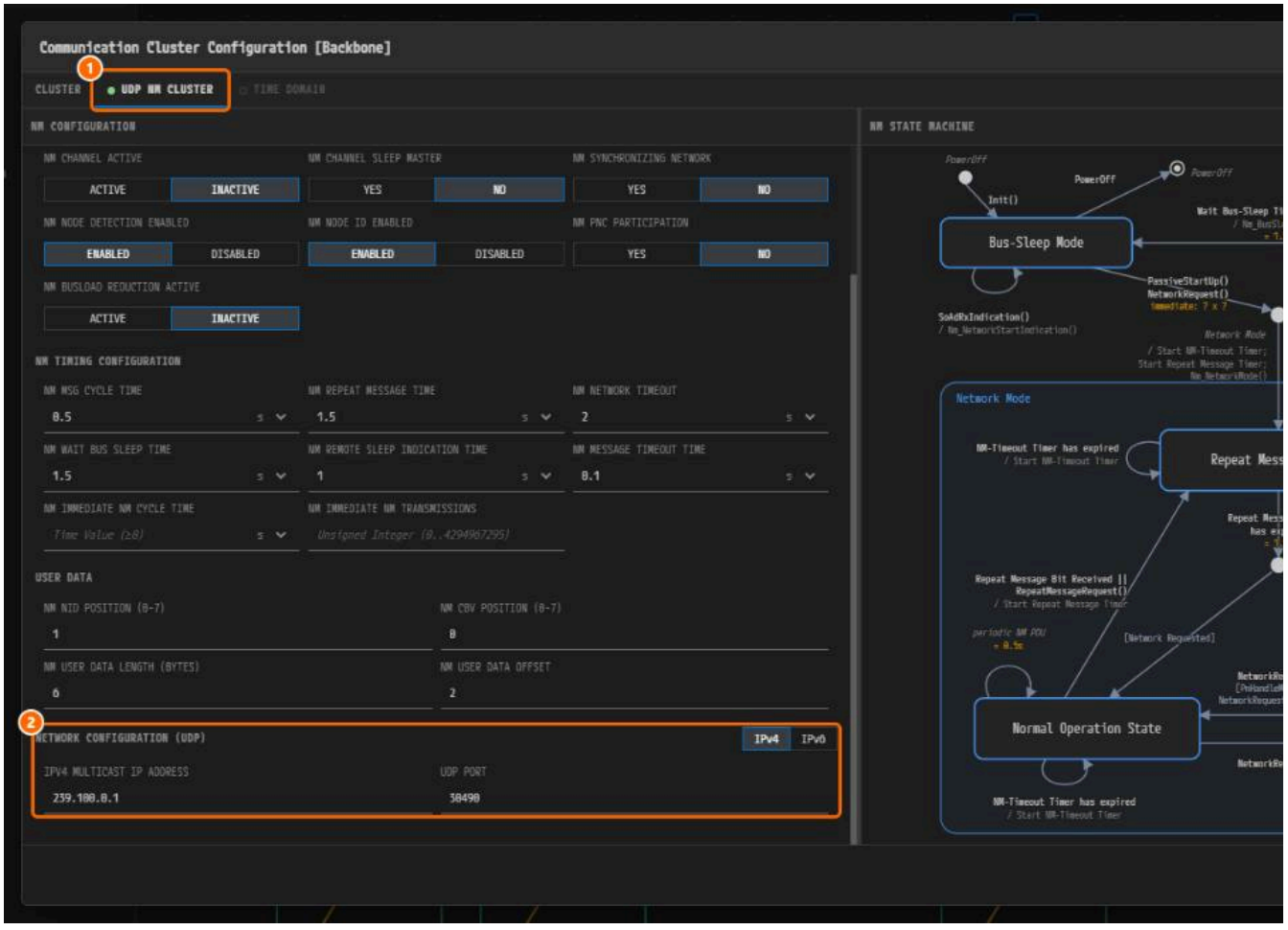


*Figure 3-39. ① **UDP NM CLUSTER**, ② **NETWORK CONFIGURATION (UDP)** at the end of the form.*

2. In **NETWORK CONFIGURATION (UDP)**, enter **IPV4 MULTICAST IP ADDRESS** `239.100.0.1` and **UDP PORT** `30490`.

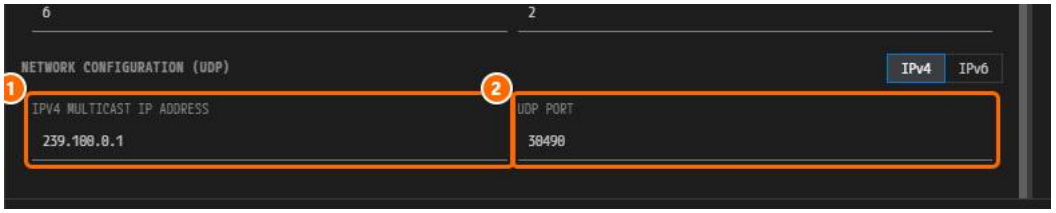


Figure 3-40. ① `239.100.0.1`, ② `30490`.

3. Fill in the UDP NM nodes from the ECU dialogs (`HPC` opens as **HPC Configuration [HPC]**). Enter **NM MSG CYCLE OFFSET** even if it is `0` (`constr_5223`).

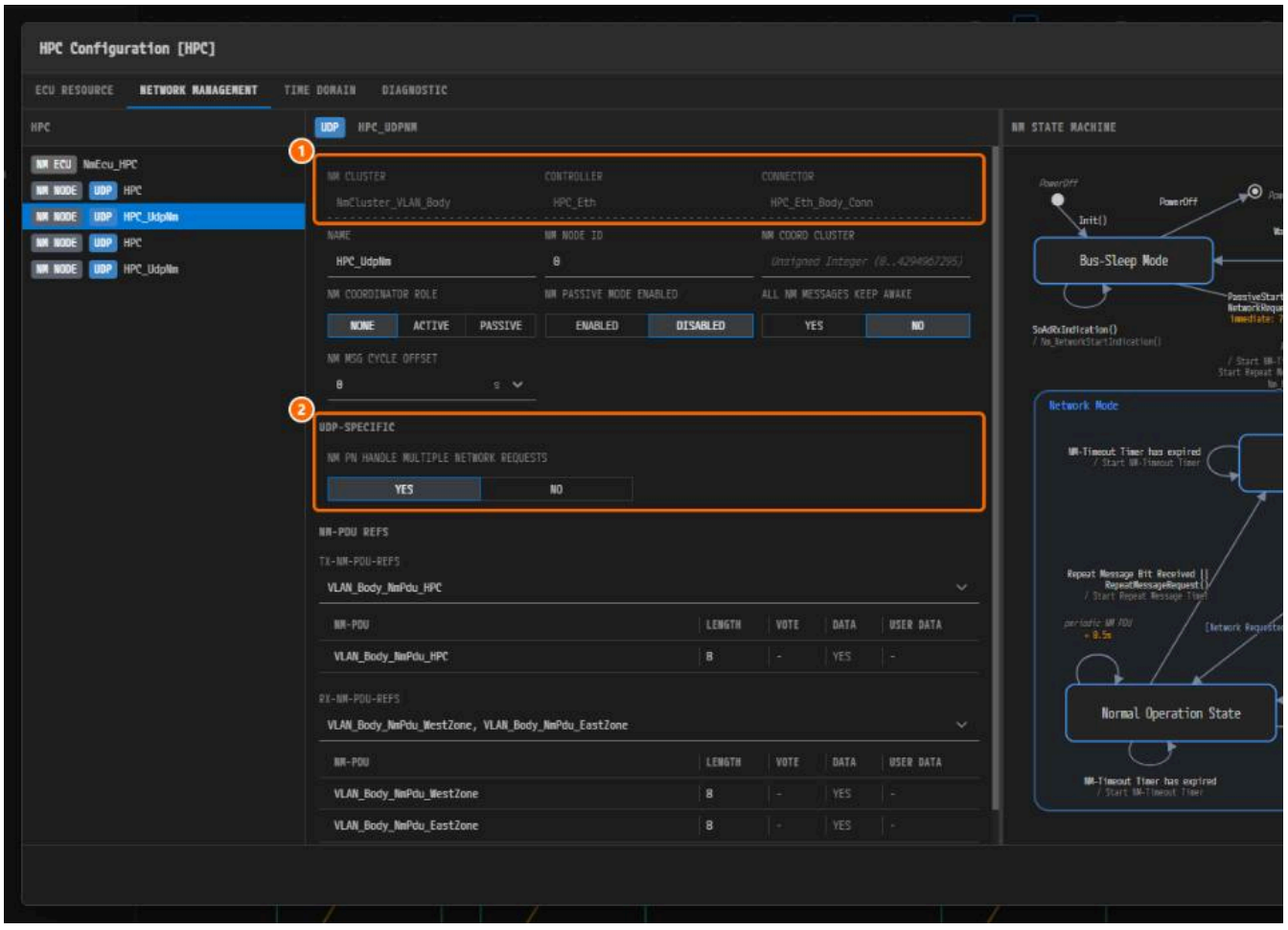


*Figure 3-41. ① Read-only rows; **CONNECTOR** is `HPC_Eth_Body_Conn`. ② **UDP-SPECIFIC: NM PN HANDLE MULTIPLE NETWORK REQUESTS = YES.***

> **Caution.** Double-clicking `Backbone` always opens `NmCluster_VLAN_Body`. `NmCluster_VLAN_Rear` cannot be edited on screen.

NM Values in Full

Every time value is in seconds (**s**).

NM cluster	Kind	Communication cluster	NM PNC PARTICIPATION	MSG CYCLE / REPEAT MESSAGE / WAIT BUS SLEEP / NETWORK TIMEOUT	REMOTE SLEEP IND / MESSAGE TIMEOUT	NID POS / CBV POS / USER DATA OFFSET / LENGTH
`NmCluster_WestCAN`	CanNm	`WestCAN`	**YES**	`0.5` / `1.5` / `1.5` / `2`	`1` / `0.1`	`1` / `0` / `2` / `6`
`NmCluster_RearCAN`	CanNm	`RearCAN`	**YES**	same	same	same
`NmCluster_EastCAN`	CanNm	`EastCAN`	**YES**	same	same	same
`NmCluster_VLAN_Body`	UdpNm	`Backbone`	**YES**	same	same	same
`NmCluster_VLAN_Rear`	UdpNm	`Backbone`	**YES**	same	same	same

Skip `NmCluster_VLAN_Rear`; it cannot be opened on screen.

All clusters: **NM NODE ID ENABLED** = **ENABLED**, **NM CHANNEL ID** empty. CanNm: **NM CHANNEL ACTIVE** = **ACTIVE**, **NM CHANNEL SLEEP MASTER** = **NO**, **NM NODE DE DETECTION ENABLED** = **ENABLED**.

Rename each NM node as in the table. The gateway node takes the first **NM NODE ID**: `16` on CAN, `0` on Ethernet.

NM cluster	NM node	NM NODE ID	NM COORDINATOR ROLE	NM MSG CYCLE OFFSET	TX-NM-PDU-REFS
`NmCluster_WestCAN`	`WestZone_CanNm`	`16`	**ACTIVE**	`0`	`WestCAN_NmPdu_WestZone`
`NmCluster_WestCAN`	`DoorCtrl_FL_CanNm`	`17`	**NONE**	`0.05`	`WestCAN_NmPdu_DoorCtrl_FL`
`NmCluster_WestCAN`	`SeatCtrl_FL_CanNm`	`18`	**NONE**	`0.10`	`WestCAN_NmPdu_SeatCtrl_FL`
`NmCluster_WestCAN`	`MirrorAct_FL_CanNm`	`19`	**NONE**	`0.15`	`WestCAN_NmPdu_MirrorAct_FL`
`NmCluster_WestCAN`	`WindowSw_FL_CanNm`	`20`	**NONE**	`0.20`	`WestCAN_NmPdu_WindowSw_FL`
`NmCluster_RearCAN`	`SouthZone_CanNm`	`16`	**ACTIVE**	`0`	`RearCAN_NmPdu_SouthZone`
`NmCluster_RearCAN`	`ParkSensor_RR_CanNm`	`17`	**NONE**	`0.05`	`RearCAN_NmPdu_ParkSensor_RR`
`NmCluster_RearCAN`	`TrailerModule_CanNm`	`18`	**NONE**	`0.10`	`RearCAN_NmPdu_TrailerModule`
`NmCluster_RearCAN`	`TailLamp_RR_CanNm`	`19`	**NONE**	`0.15`	`RearCAN_NmPdu_TailLamp_RR`
`NmCluster_EastCAN`	`EastZone_CanNm`	`16`	**ACTIVE**	`0`	`EastCAN_NmPdu_EastZone`
`NmCluster_EastCAN`	`DoorCtrl_FR_CanNm`	`17`	**NONE**	`0.05`	`EastCAN_NmPdu_DoorCtrl_FR`
`NmCluster_EastCAN`	`SeatCtrl_FR_CanNm`	`18`	**NONE**	`0.10`	`EastCAN_NmPdu_SeatCtrl_FR`
`NmCluster_EastCAN`	`MirrorAct_FR_CanNm`	`19`	**NONE**	`0.15`	`EastCAN_NmPdu_MirrorAct_FR`
`NmCluster_VLAN_Body`	`HPC_UdpNm`	`0`	**NONE**	`0`	`VLAN_Body_NmPdu_HPC`
`NmCluster_VLAN_Body`	`WestZone_UdpNm`	`1`	**ACTIVE**	`0`	`VLAN_Body_NmPdu_WestZone`
`NmCluster_VLAN_Body`	`EastZone_UdpNm`	`2`	**PASSIVE**	`0`	`VLAN_Body_NmPdu_EastZone`
`NmCluster_VLAN_Rear`	`HPC_UdpNm`	`0`	**NONE**	`0`	`VLAN_Rear_NmPdu_HPC`
`NmCluster_VLAN_Rear`	`SouthZone_UdpNm`	`1`	**ACTIVE**	`0`	`VLAN_Rear_NmPdu_SouthZone`

Enter **NM NODE ID**, **NM COORDINATOR ROLE**, and **NM MSG CYCLE OFFSET** now. After Chapter 4, fill in **TX-NM-PDU-REFS** from the table and, in **RX-NM-PDU-REF**

Why Is It Designed This Way?

- **PNC 17, 18, 19.** The identifier is the absolute bit in the NM PDU. Offset `2` and length `6` give bits `16`..`63`; 17 to 19 all fit in one byte.
- **Wakeup ID and mask.** A frame wakes the bus when (ID AND `0x700`) = `0x500`. The NM frames (`0x510` + node ID) fall in that block.
- **Gateway on the connector.** One zone ECU can play different roles on its CAN and Ethernet connectors.
- **One active gateway per VLAN.** See Topology Design.
- **Two places.** Connectors say how an ECU handles a PNC; **PHYSICAL CHANNELS** and **MATRIX** in PNC Dialog — Per-PNC Bindings say which channels a PNC

Checklist

- [ ] The PNC dialog lists **PNC 17 PNC_Comfort**, **PNC 18 PNC_ParkAssist**, and **PNC 19 PNC_Trailer**.
- [ ] **PNC VECTOR OFFSET** is `2` and **PNC VECTOR LENGTH** is `6`.
- [ ] `WestZone_CanFd_Conn` is **ACTIVE** with `0x500` / `0x700` / `8` and Byte 0 = `0x02`; only `EastZone_Eth_Conn` is **PASSIVE**.
- [ ] `Port_0` of `BackboneSwitch` has all three PNCs.
- [ ] `NmCluster_WestCAN` has **NM PNC PARTICIPATION** = **YES**; `WestZone_CanNm` has **NM NODE ID** `16`.
- [ ] (After Chapter 4) `WestZone_CanNm` has one TX and four RX NM-PDU references.
- [ ] The tab **PNC:System** carries no modified mark.

Next Step

In Signals, PDUs, Frames, and I-PDU Groups you create the signals, PDUs, and I-PDU Groups, then return to NM Configuration to fill in the NM-PDU reference

Signals, PDUs, Frames and I-PDU Groups

What You Will Do

What a PNC starts and stops is an I-PDU group, a group of I-Signal I-PDUs. In this chapter you create CAN frames, I-Signal I-PDUs, signals and NM-PDUs in t PDU groups in the PNC dialog and put PDUs into them.

Before You Begin

- Start after finishing PNC Definitions and Connector PNC Properties. The PNCs 17, 18 and 19 exist.
- You build one set step by step: the frame `WestCAN_DoorStatus_FL`, its PDU `DoorStatus_FL_Pdu`, three signals, the NM-PDU `WestCAN_NmPdu_WestZc` create the rest from the tables.
- The procedure figures show a project in which the other frames already exist, so their item counts differ from yours. The labels and values are the sa
- The Signal editor cannot create NM frames. A project you build yourself ends with 16 application frames instead of 29 (see Creating NM-PDUs).

Item	Finished example
CAN frames	29 (16 application + 13 NM)
I-Signal I-PDUs	16
NM-PDUs	18
Signals	42
I-PDU groups	`PNC_Comfort` 4, `PNC_ParkAssist` 2, `PNC_Trailer` 2

Opening the Signal Editor

1. Under **DESIGN** in the **EDITOR** view, click **System**. In the icon strip on the left edge of the canvas, click the upper icon **Signal**.

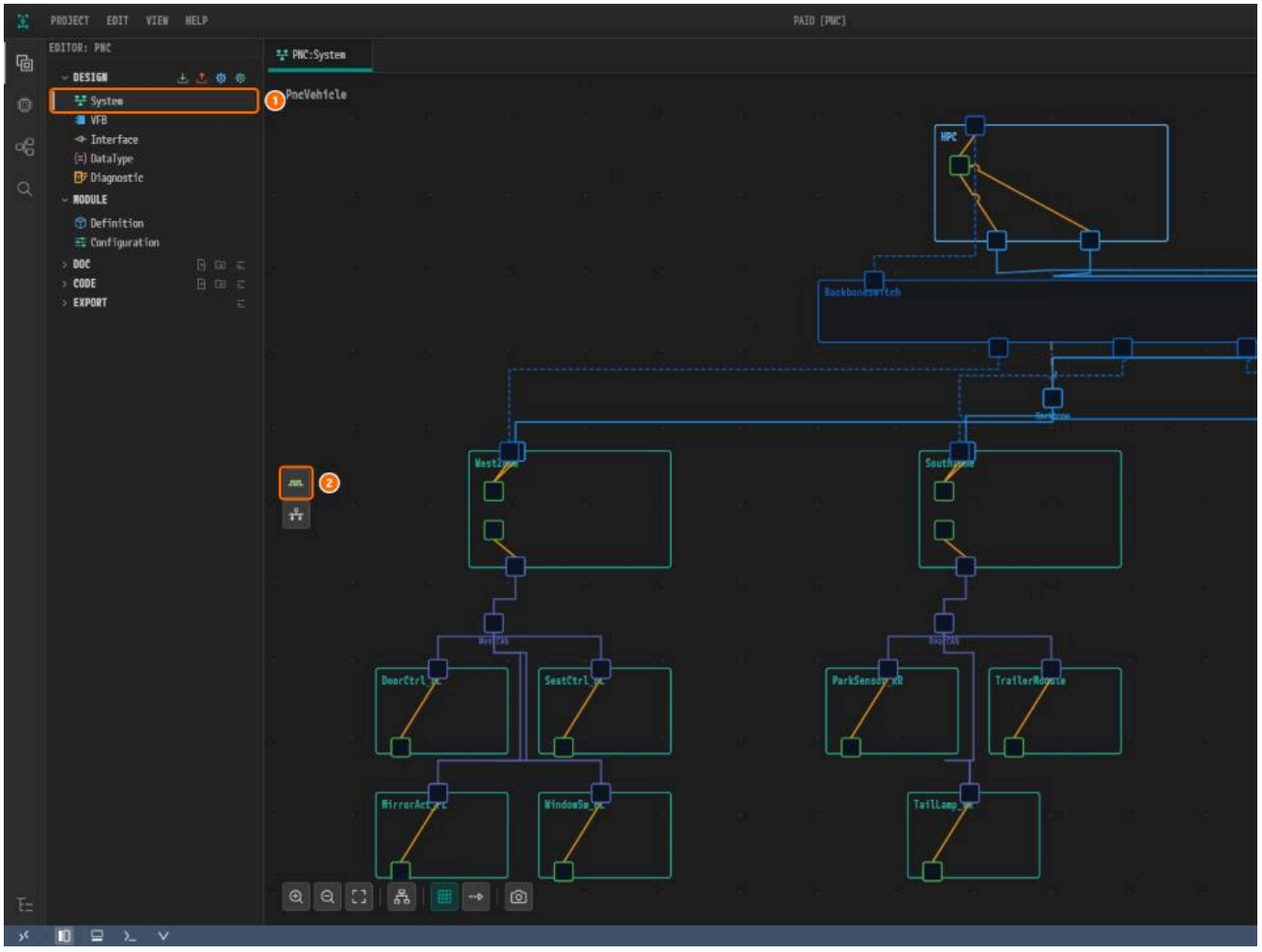


*Figure 4-1. ① **System**, ② **Signal**. The icon below it is **Partial Network (PNC)**.*

2. The tab **PNC:Signal** opens with the tabs **CAN**, **LIN**, **PDU** and **SIGNAL** on top. **CAN** is selected.

The list is on the left, **CONFIGURATION** and the **SIGNALS** table in the center, and the tabs **PREVIEW** and **PDU REFERENCES** on the right.

Creating a CAN Frame and Its PDU

1. On the **CAN** tab, click **+** (**Add**) on the **CAN FRAMES** header.

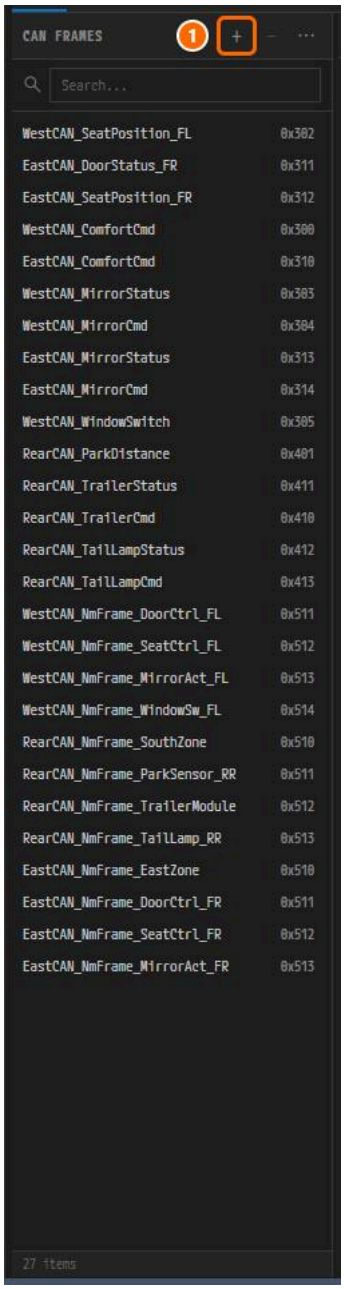


*Figure 4-2. ① **+**.*

2. `NewFrame` appears and is selected. Its I-Signal I-PDU `NewFrame_Pdu` is created with it. Click the **PDU REFERENCES** tab to see it.

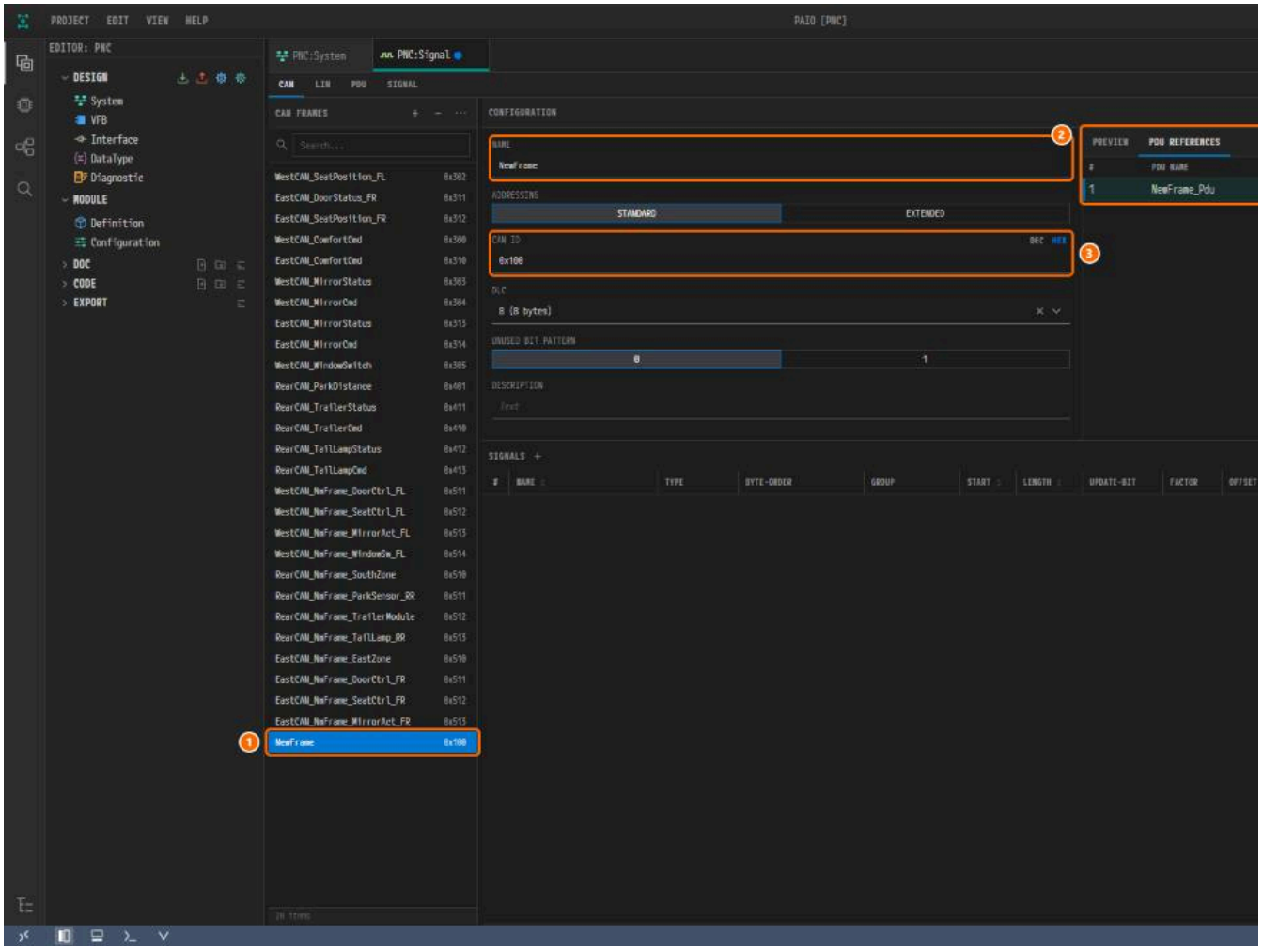


*Figure 4-3. ① `NewFrame`, ② **NAME**, ③ **CAN ID**, ④ `NewFrame_Pdu`.*

3. Enter `WestCAN_DoorStatus_FL` in **NAME** and `0x301` in **CAN ID**, pressing `Enter` after each. Leave **ADDRESSING** **STANDARD**, **DLC** `8 (8 bytes)` and **UNUS** name.

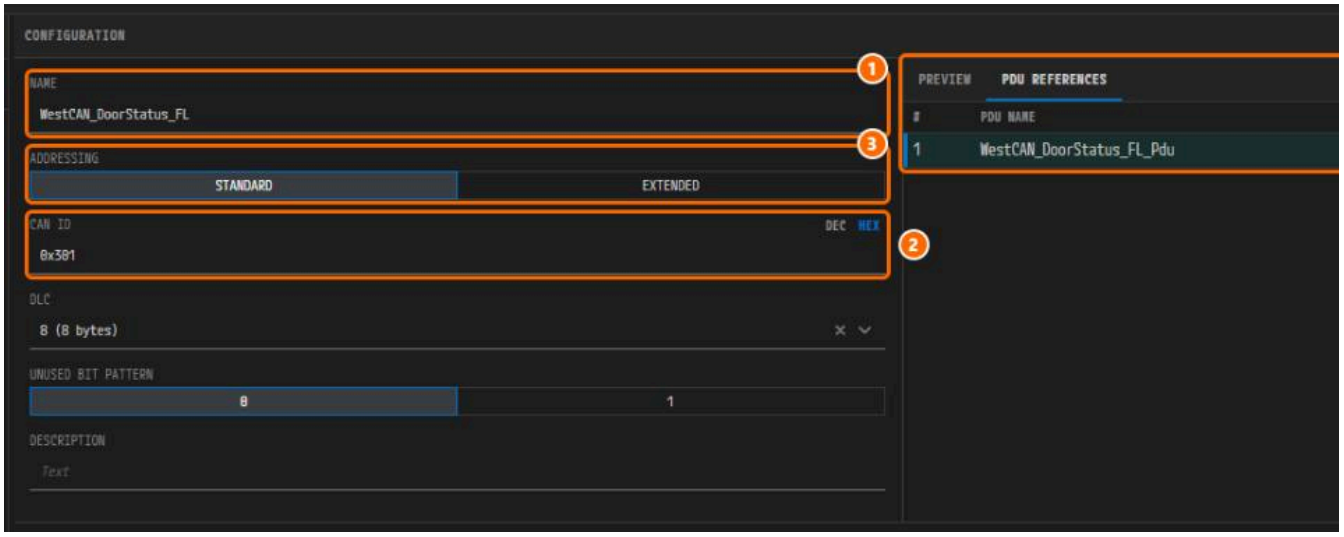


*Figure 4-4. ① **NAME**, ② **CAN ID**, ③ **ADDRESSING**, ④ **PDU REFERENCES** with `WestCAN_DoorStatus_FL_Pdu`.*

DEC | **HEX** next to the **CAN ID** label changes only the notation.

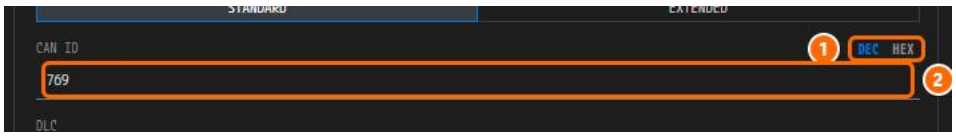


*Figure 4-5. ① **DEC | HEX**, ② `769` in **DEC**.*

4. Click the **PDU** tab, then click `WestCAN_DoorStatus_FL_Pdu` (**IS** tag).

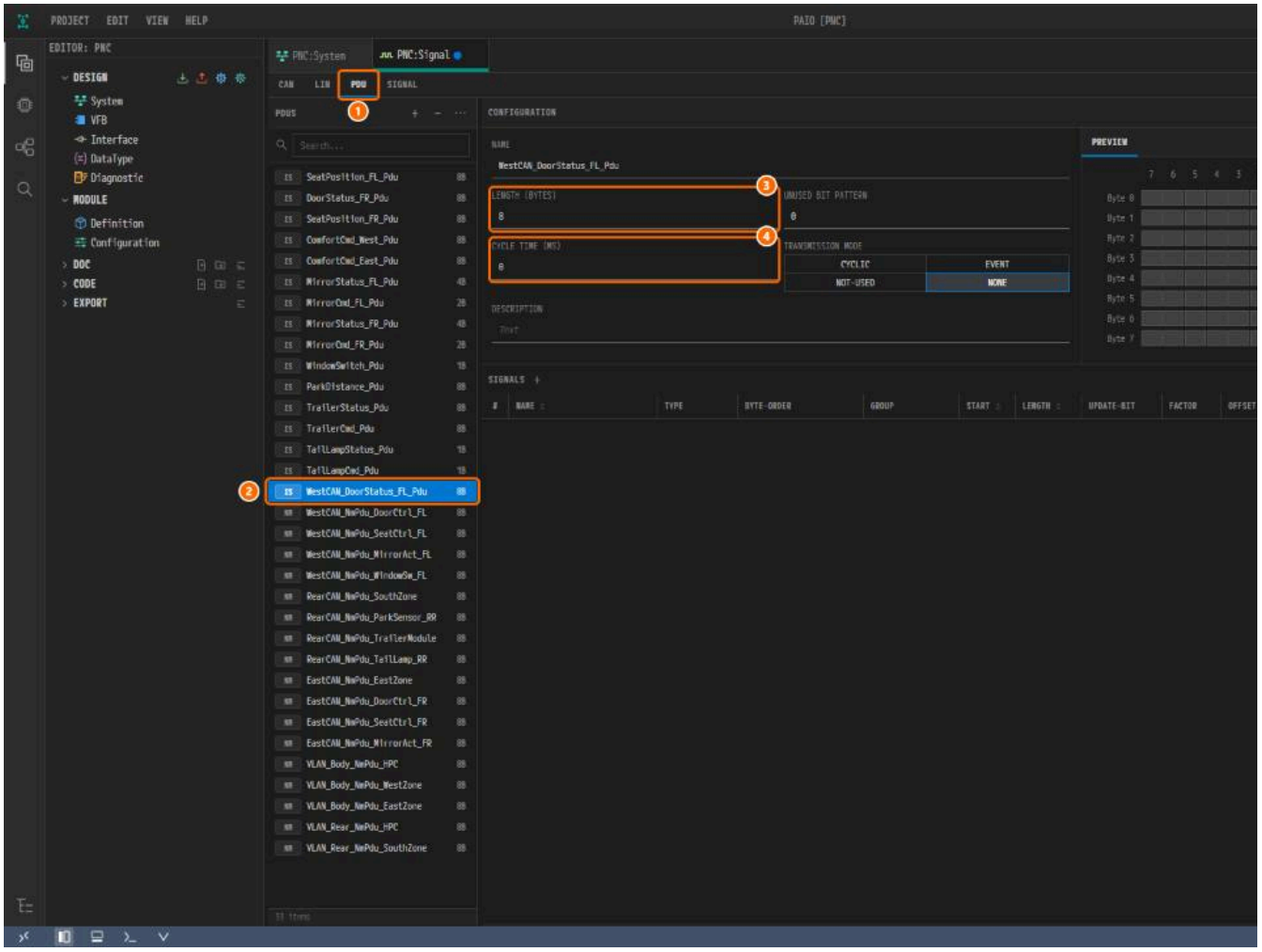


*Figure 4-6. ① **PDU**, ② `WestCAN_DoorStatus_FL_Pdu`, ③ **LENGTH (BYTES)**, ④ **CYCLE TIME (MS)**.*

5. Enter `DoorStatus_FL_Pdu` in **NAME** and `50` in **CYCLE TIME (MS)**. Leave **LENGTH (BYTES)** at `8`. Select **CYCLIC** in **TRANSMISSION MODE**.

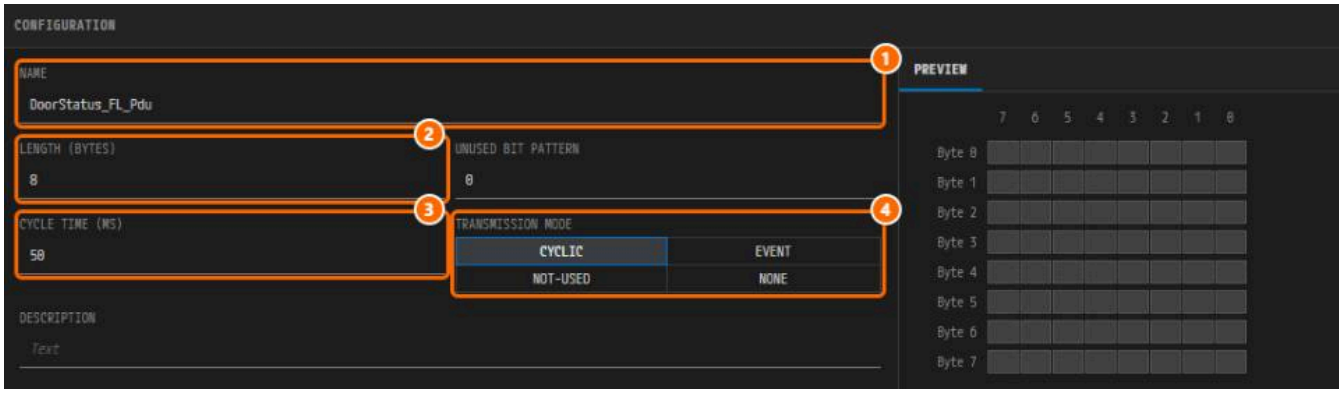


*Figure 4-7. ① **NAME**, ② **LENGTH (BYTES)**, ③ **CYCLE TIME (MS)**, ④ **TRANSMISSION MODE**.*

6. Save with `Ctrl+S`.

Creating Signals and Mapping Them to a PDU

1. On the **CAN** tab, click `WestCAN_DoorStatus_FL`.
2. Click **+** (**Add Signal**) next to the **SIGNALS** title. The signal `WestCAN_DoorStatus_FL_Sig` is created and mapped to the frame's PDU.



*Figure 4-8. ① **+**, ② the new row.*

3. Click a cell, type the value and press `Enter`. Set **NAME** `DoorOpen_FL`, **LENGTH** `1` and **MAX** `0`. Edit **LENGTH** before **START**, or the bits may overlap.

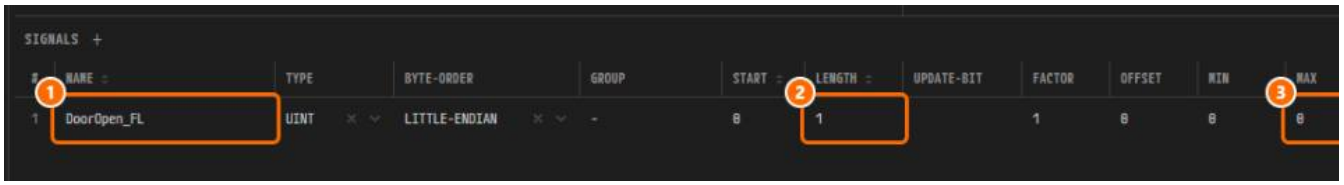


*Figure 4-9. ① **NAME**, ② **LENGTH**, ③ **MAX**.*

4. Add the other two signals the same way: click **+**, then edit the new row. Click the **PREVIEW** tab to check the layout.

NAME	LENGTH	START	MAX
`DoorOpen_FL`	`1`	`0` (automatic)	`0`
`DoorLock_FL`	`2`	`1` (automatic)	`0`
`WindowPos_FL`	`8`	`8` (typed in)	`0`

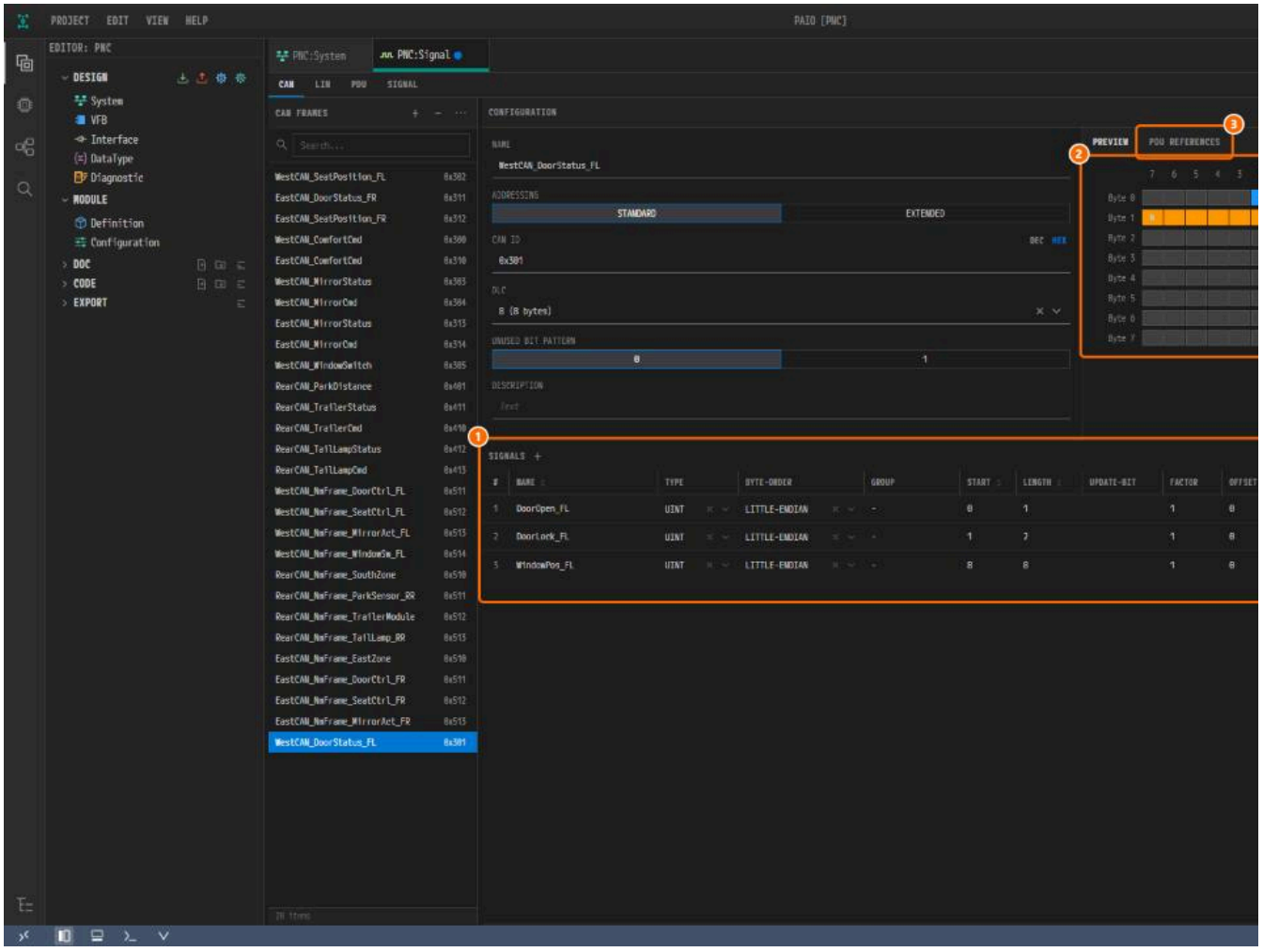


*Figure 4-10. ① The three rows, ② **PREVIEW**, ③ **PDU REFERENCES** with **SIGNALS** `3`.*

5. Save with `Ctrl+S`.

> **Caution.** A new signal gets **MAX** `255` and a **START** right after the last signal, not at the first free bit. Set **MAX** to `0` yourself, and type **START** `8` for `Wind`

Creating NM-PDUs

1. On the **PDU** tab, click **+** (**Add**) on the **PDUS** header. Hover over **Network Mgmt** and click **NM-PDU**.

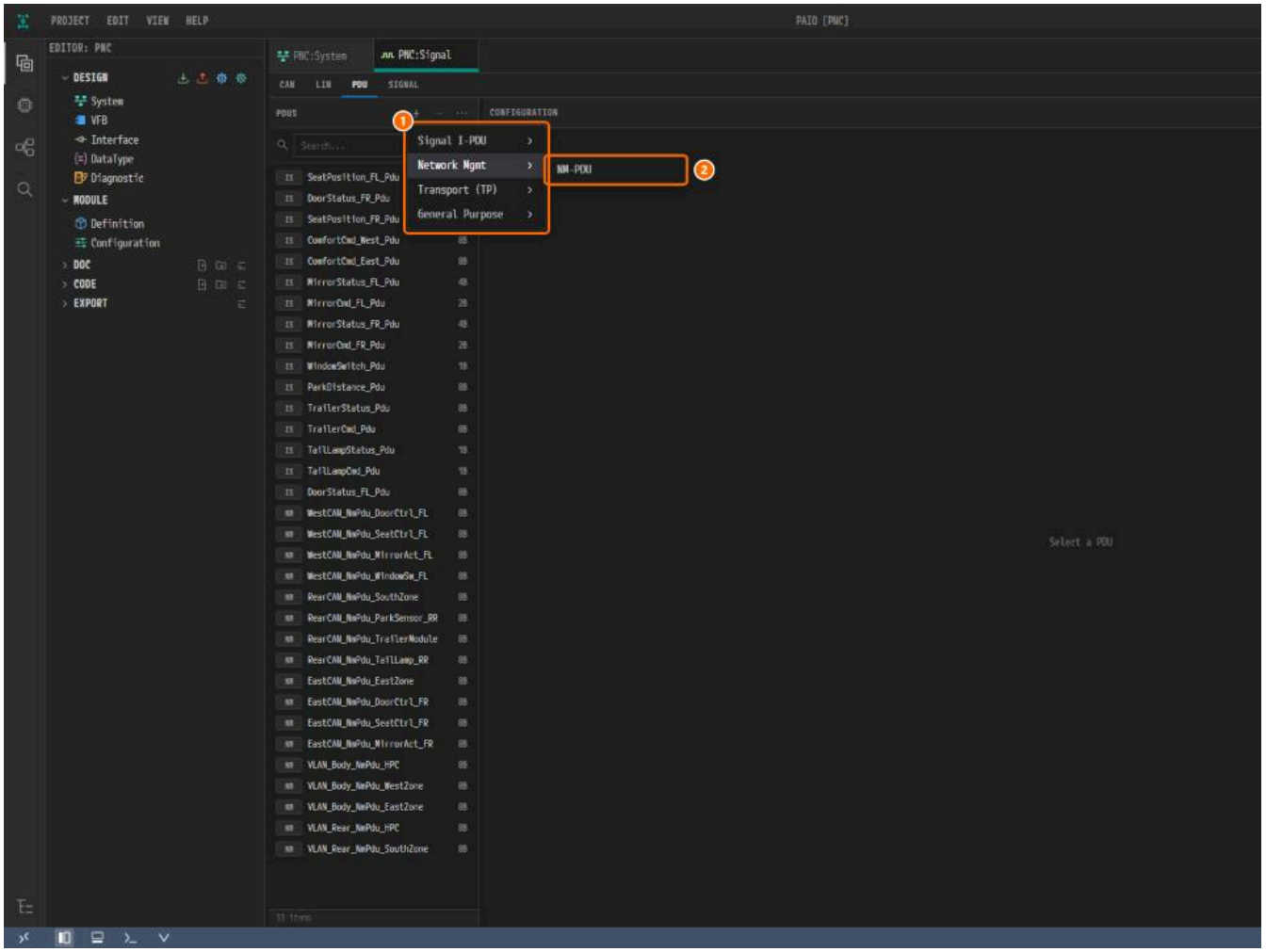


*Figure 4-11. ① The menu groups, ② **NM-PDU**.*

2. `NewNmPdu` is created and selected.

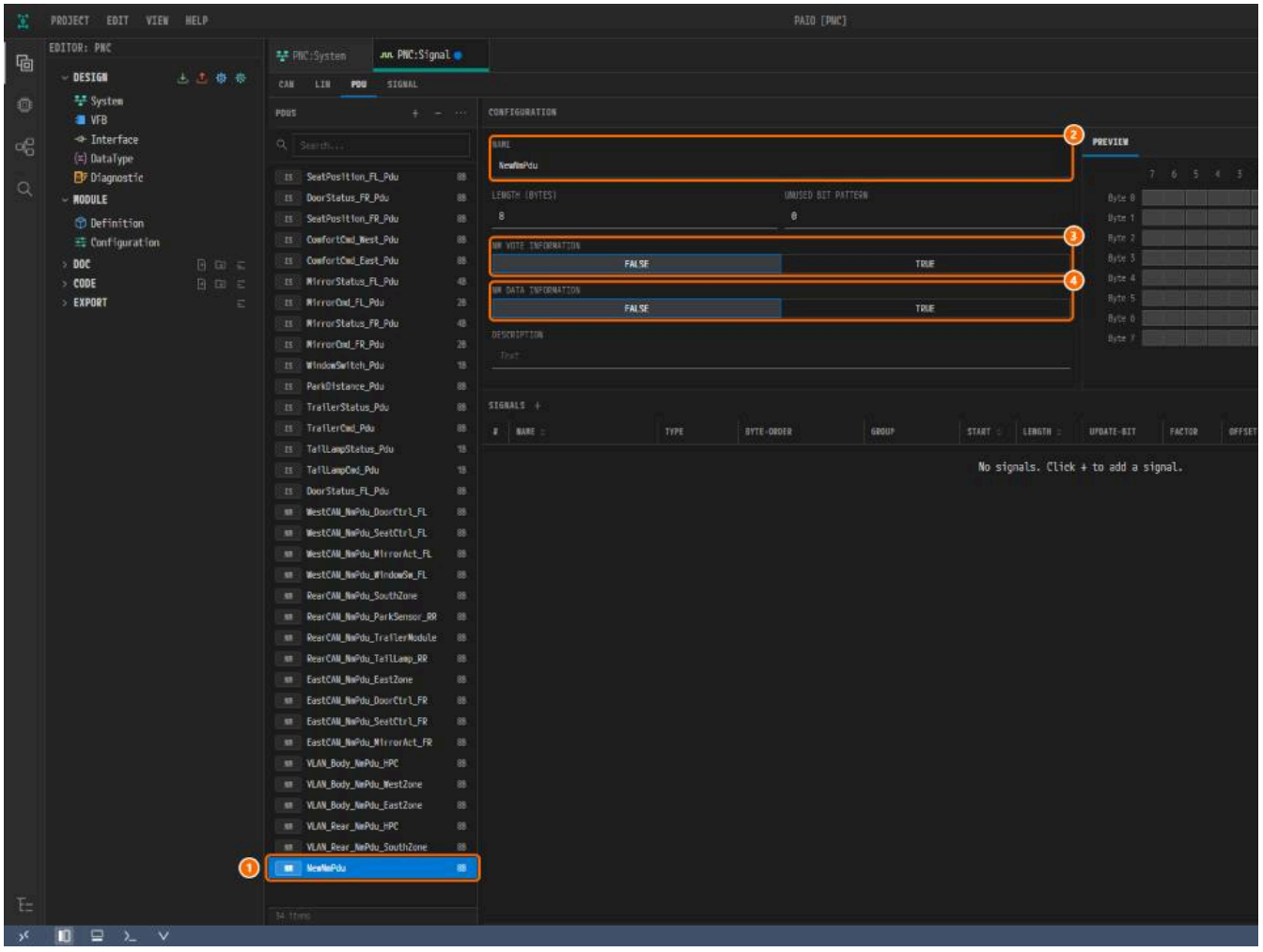


*Figure 4-12. ① `NewNmPdu`, ② **NAME**, ③ **NM VOTE INFORMATION**, ④ **NM DATA INFORMATION**.*

3. Enter `WestCAN_NmPdu_WestZone` in **NAME** and set **NM DATA INFORMATION** to **TRUE**. Leave **LENGTH (BYTES)** `8` and **NM VOTE INFORMATION FALSE**.



*Figure 4-13. ① **NAME**, ② **NM DATA INFORMATION**, ③ **LENGTH (BYTES)**.*

4. Create the other 17 NM-PDUs from the table with the same settings, and save with `Ctrl+S`.

> **Note.** With these 18 saved, you can do the step deferred in NM Settings. After this chapter, go back and fill **TX-NM-PDU-REFS** and **RX-NM-PDU-REFS**

Cluster	NM-PDU
WestCAN	`WestCAN_NmPdu_WestZone`, `WestCAN_NmPdu_DoorCtrl_FL`, `WestCAN_NmPdu_SeatCtrl_FL`, `WestCAN_NmPdu_MirrorAct_FL`, `WestCAN_NmPdu_WindowSw_FL`
RearCAN	`RearCAN_NmPdu_SouthZone`, `RearCAN_NmPdu_ParkSensor_RR`, `RearCAN_NmPdu_TrailerModule`, `RearCAN_NmPdu_TailLamp_RR`
EastCAN	`EastCAN_NmPdu_EastZone`, `EastCAN_NmPdu_DoorCtrl_FR`, `EastCAN_NmPdu_SeatCtrl_FR`, `EastCAN_NmPdu_MirrorAct_FR`
VLAN_Body	`VLAN_Body_NmPdu_HPC`, `VLAN_Body_NmPdu_WestZone`, `VLAN_Body_NmPdu_EastZone`
VLAN_Rear	`VLAN_Rear_NmPdu_HPC`, `VLAN_Rear_NmPdu_SouthZone`

> **Caution.** The Signal editor cannot create NM frames: the **PDU REFERENCES** list of a CAN frame never offers NM-PDUs. The 13 NM frames of the example c import. The I-PDU groups do not need them, so continue without them.

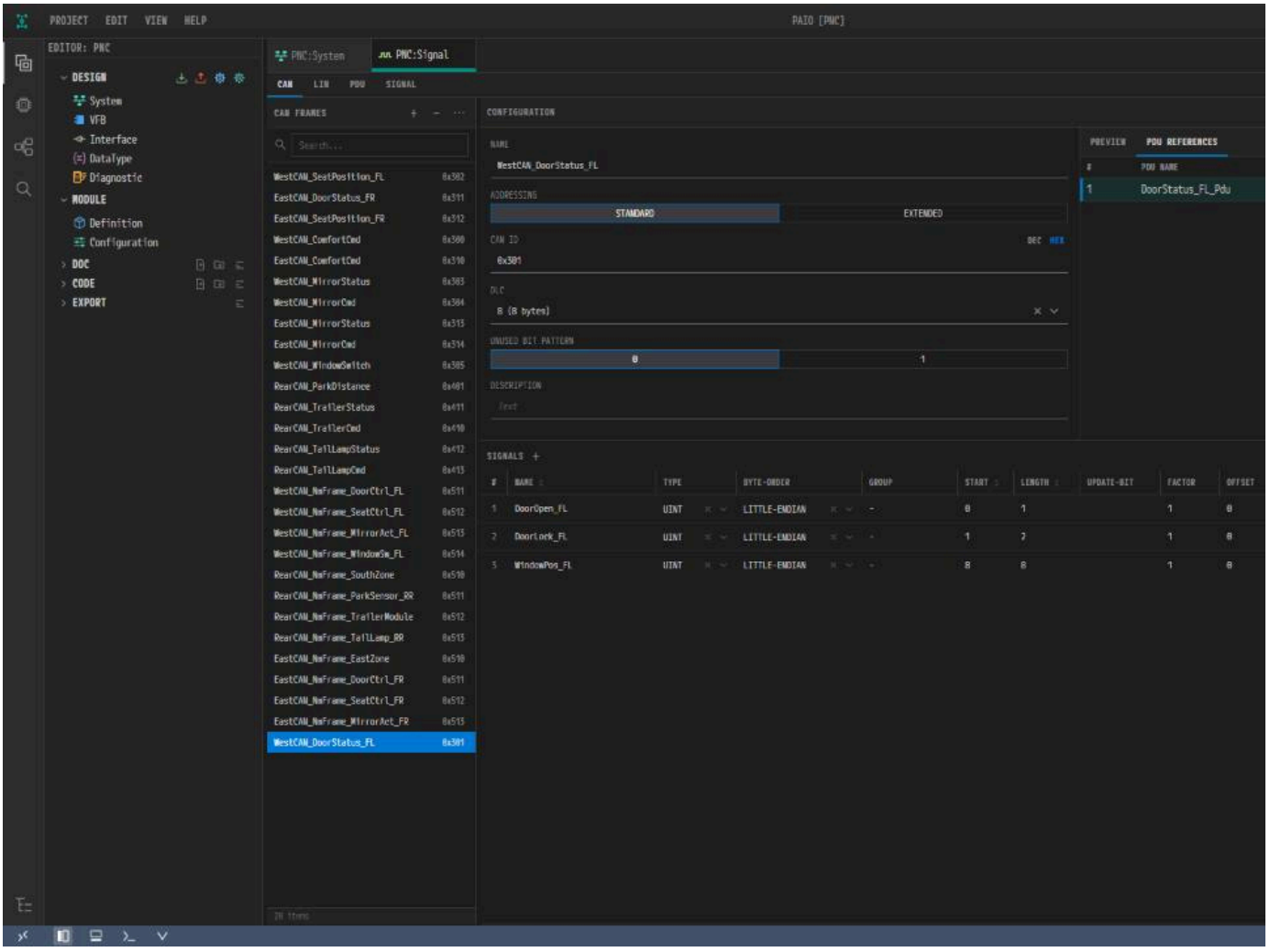


*Figure 4-14. ① The combo box on the **PDU REFERENCES** tab, ② its candidates: I-Signal I-PDUs only, no NM-PDU.*

Remaining Frames, PDUs and Signals

Create the other 15 frames from the table the same way: frame name and **CAN ID**, then the PDU name, length and cycle time, then the signals.

- Frames: **ADDRESSING** **STANDARD**, **DLC** `8 (8 bytes)` and **UNUSED BIT PATTERN** `0`, even when the PDU is shorter.
- PDUs: **TRANSMISSION MODE** **CYCLIC**. The default is **NONE**, so set it on each PDU.
- Signals: **MAX** `0`. Leave the other fields at their defaults.

Frame	CAN ID	PDU	PDU length (B)	Cycle time (ms)	Signals (START, LENGTH)
WestCAN_ComfortCmd	0x300	ComfortCmd_West_Pdu	8	20	LockCmd_FL (0, 2), WindowCmd_FL (2, 4), SeatMoveCmd_FL (8, 4), Se
WestCAN_DoorStatus_FL	0x301	DoorStatus_FL_Pdu	8	50	DoorOpen_FL (0, 1), DoorLock_FL (1, 2), WindowPos_FL (8, 8)
WestCAN_SeatPosition_FL	0x302	SeatPosition_FL_Pdu	8	100	SeatPos_FL (0, 8), SeatHeat_FL (8, 2)
WestCAN_MirrorStatus	0x303	MirrorStatus_FL_Pdu	4	100	MirrorFoldSts_FL (0, 2), MirrorTiltX_FL (8, 8), MirrorTiltY_FL
WestCAN_MirrorCmd	0x304	MirrorCmd_FL_Pdu	2	100	MirrorFoldCmd_FL (0, 2), MirrorMoveCmd_FL (2, 4)
WestCAN_WindowSwitch	0x305	WindowSwitch_Pdu	1	100	WindowSwUp (0, 1), WindowSwDown (1, 1), WindowSwAuto (2, 1)
EastCAN_ComfortCmd	0x310	ComfortCmd_East_Pdu	8	20	LockCmd_FR (0, 2), WindowCmd_FR (2, 4), SeatMoveCmd_FR (8, 4), Se
EastCAN_DoorStatus_FR	0x311	DoorStatus_FR_Pdu	8	50	DoorOpen_FR (0, 1), DoorLock_FR (1, 2), WindowPos_FR (8, 8)
EastCAN_SeatPosition_FR	0x312	SeatPosition_FR_Pdu	8	100	SeatPos_FR (0, 8), SeatHeat_FR (8, 2)
EastCAN_MirrorStatus	0x313	MirrorStatus_FR_Pdu	4	100	MirrorFoldSts_FR (0, 2), MirrorTiltX_FR (8, 8), MirrorTiltY_FR
EastCAN_MirrorCmd	0x314	MirrorCmd_FR_Pdu	2	100	MirrorFoldCmd_FR (0, 2), MirrorMoveCmd_FR (2, 4)
RearCAN_ParkDistance	0x401	ParkDistance_Pdu	8	50	ParkDist_1 (0, 8), ParkDist_2 (8, 8), ParkDist_3 (16, 8), ParkDis
RearCAN_TrailerCmd	0x410	TrailerCmd_Pdu	8	50	TrailerLampCmd (0, 8)
RearCAN_TrailerStatus	0x411	TrailerStatus_Pdu	8	100	TrailerAttached (0, 1), TrailerLampFault (1, 4), TrailerBrakeLa
RearCAN_TailLampStatus	0x412	TailLampStatus_Pdu	1	100	TailLampSts (0, 4)
RearCAN_TailLampCmd	0x413	TailLampCmd_Pdu	1	100	TailLampCmd (0, 4)

Save with `Ctrl+S`.

Looking at a CAN Frame

The figures from here on show the finished example.

1. On the **CAN** tab, click `WestCAN_DoorStatus_FL`, then the **PREVIEW** tab.

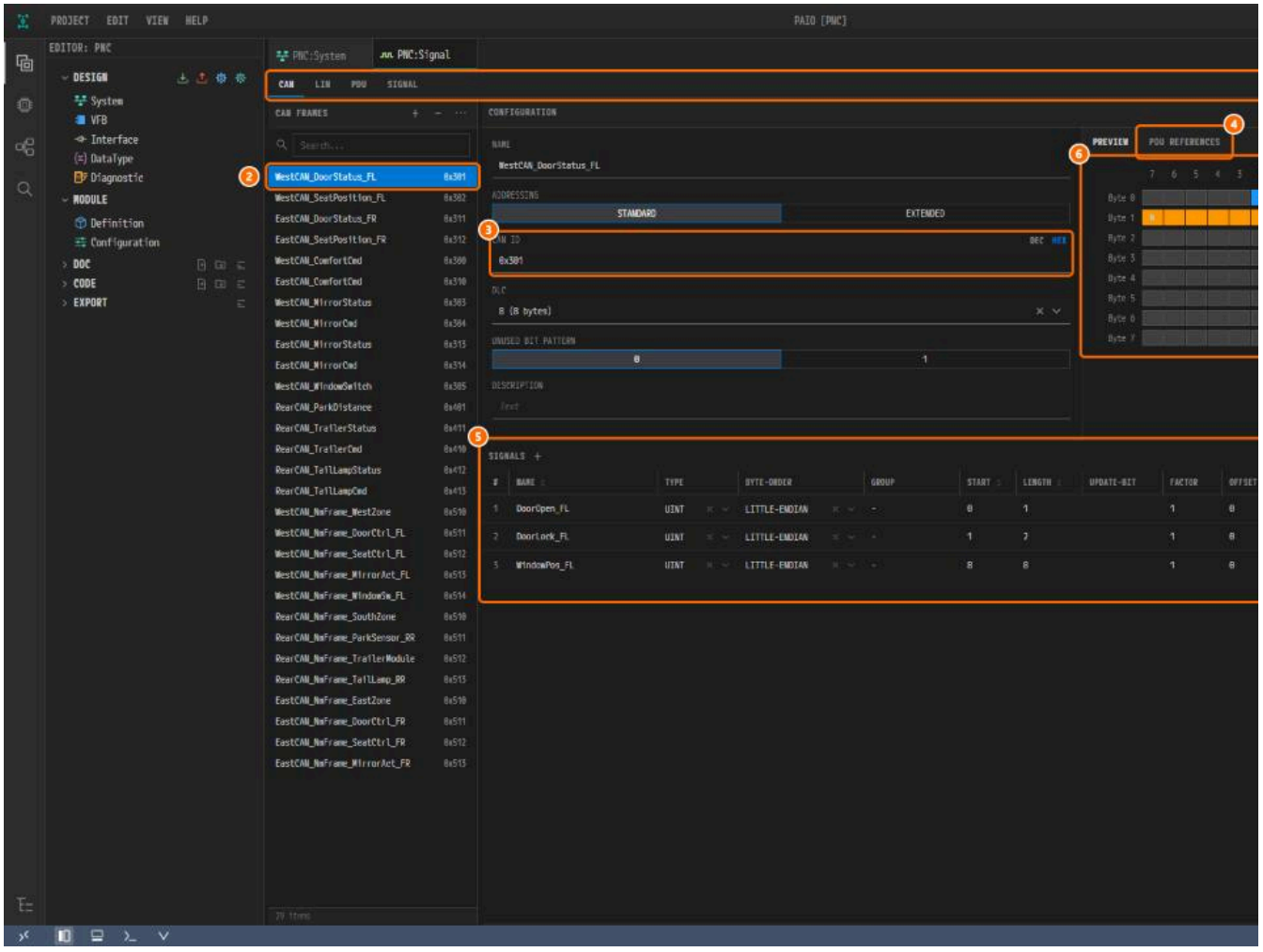


*Figure 4-15. ① Protocol tabs, ② `WestCAN_DoorStatus_FL`, ③ **CAN ID**, ④ **PDU REFERENCES**, ⑤ **SIGNALS**, ⑥ **PREVIEW**.*

2. Click the **PDU REFERENCES** tab. It lists `DoorStatus_FL_Pdu` with **LENGTH** `8` and **SIGNALS** `3`.

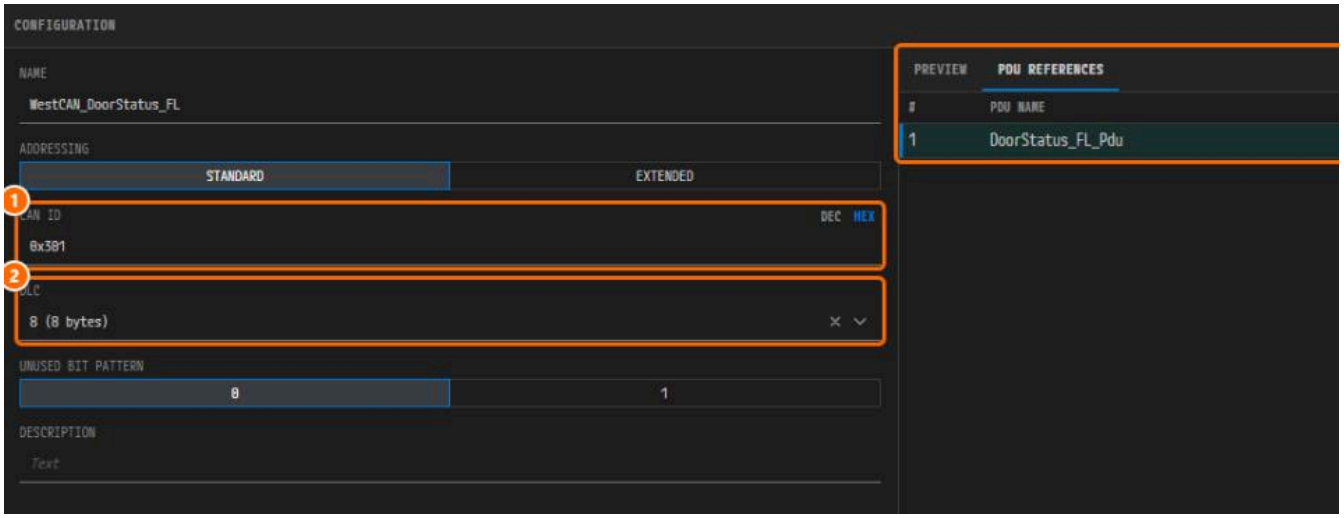


*Figure 4-16. ① **CAN ID** `0x301`, ② **DLC**, ③ **PDU REFERENCES**.*

Looking at a PDU

1. Click the **PDU** tab, then click `DoorStatus_FL_Pdu`. The list holds 34 PDUs: 16 with the **IS** tag and 18 with the **NM** tag.

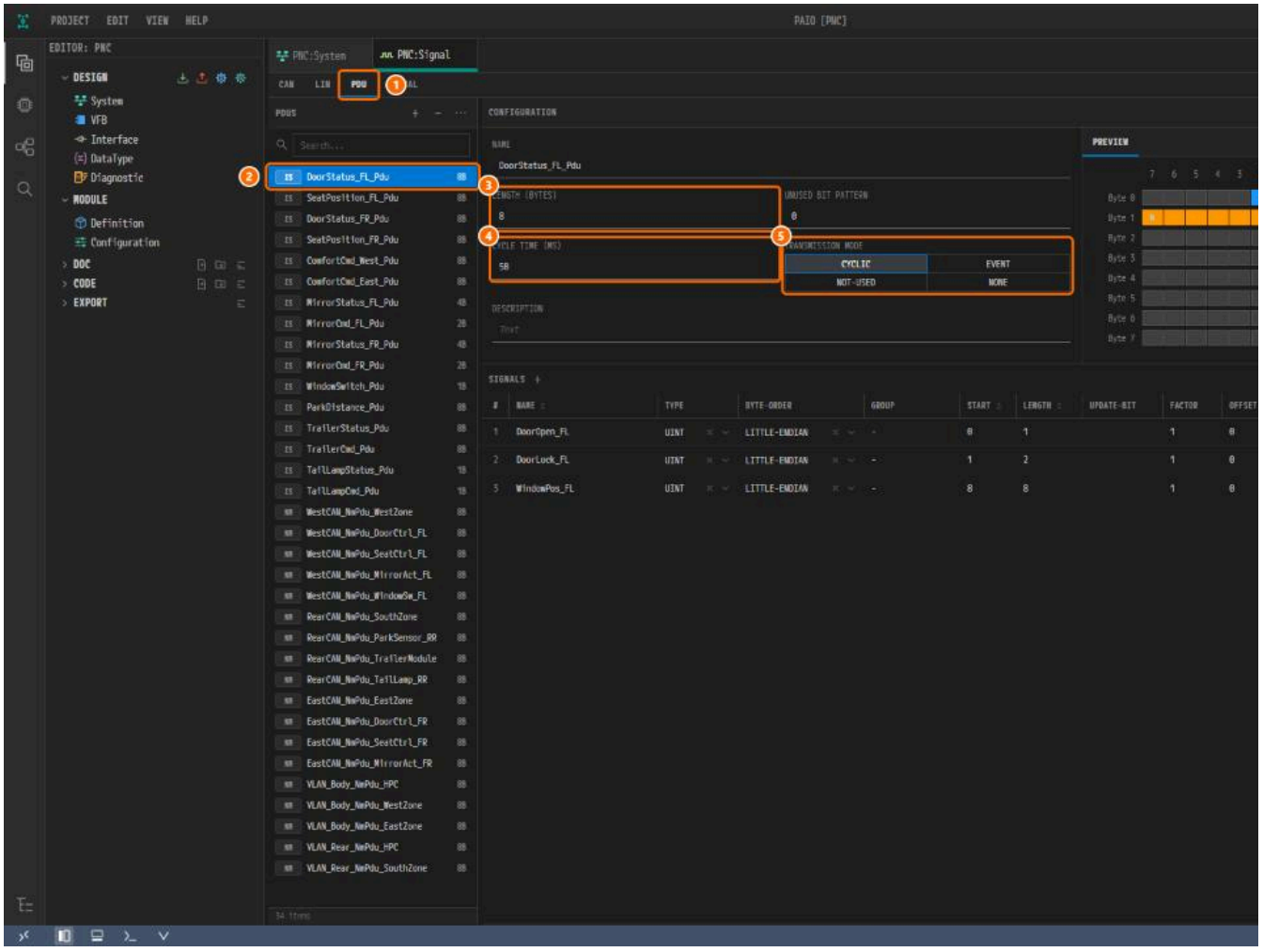


*Figure 4-17. ① **PDU**, ② `DoorStatus_FL_Pdu`, ③ **LENGTH (BYTES)**, ④ **CYCLE TIME (MS)**, ⑤ **TRANSMISSION MODE**.*

2. Click `WestCAN_NmPdu_WestZone`. An NM-PDU has no signals.

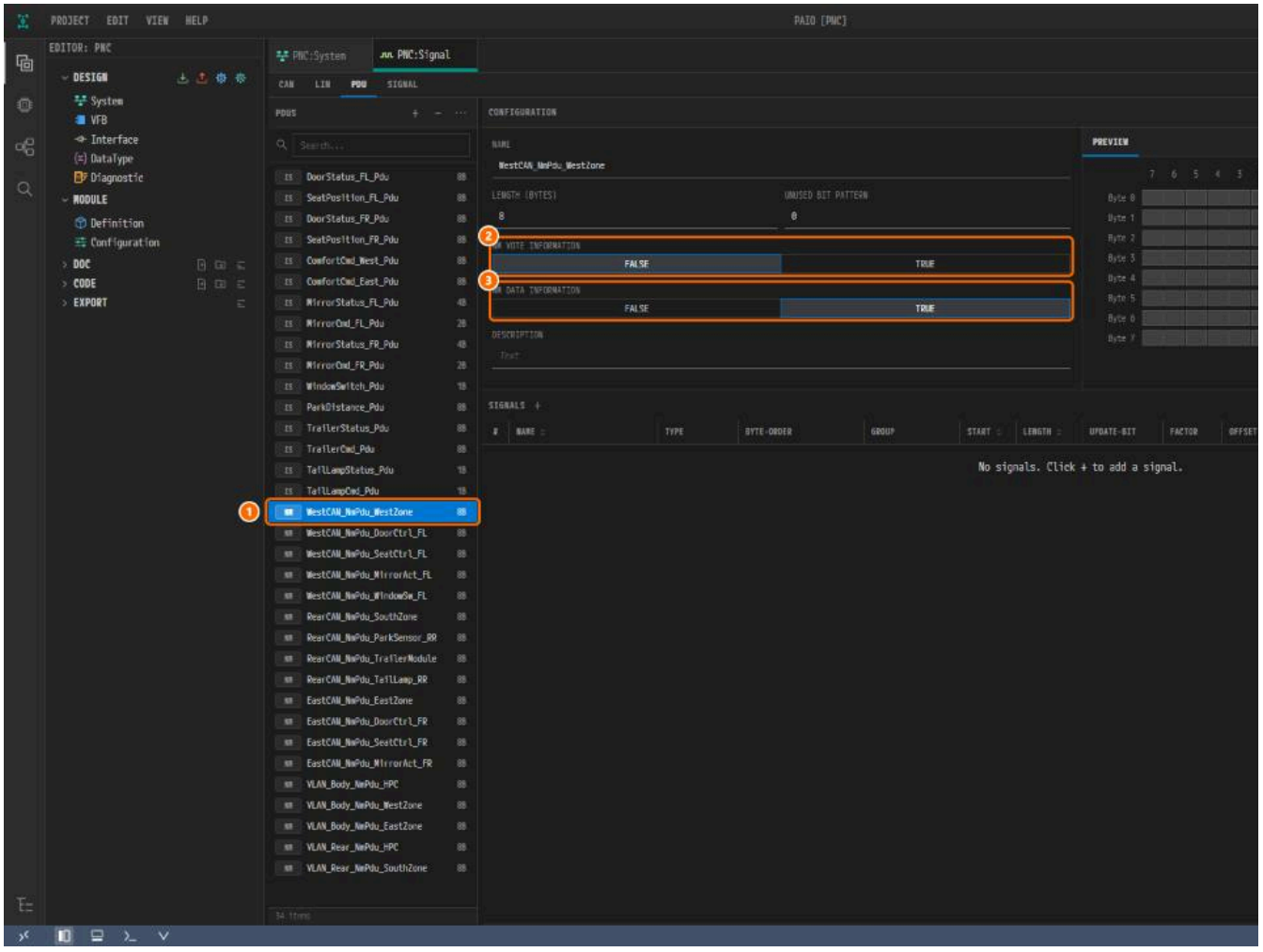


*Figure 4-18. ① `WestCAN_NmPdu_WestZone`, ② **NM VOTE INFORMATION**, ③ **NM DATA INFORMATION**.*

Looking at a Signal

1. Click the **SIGNAL** tab (42 signals), then click `DoorOpen_FL`.

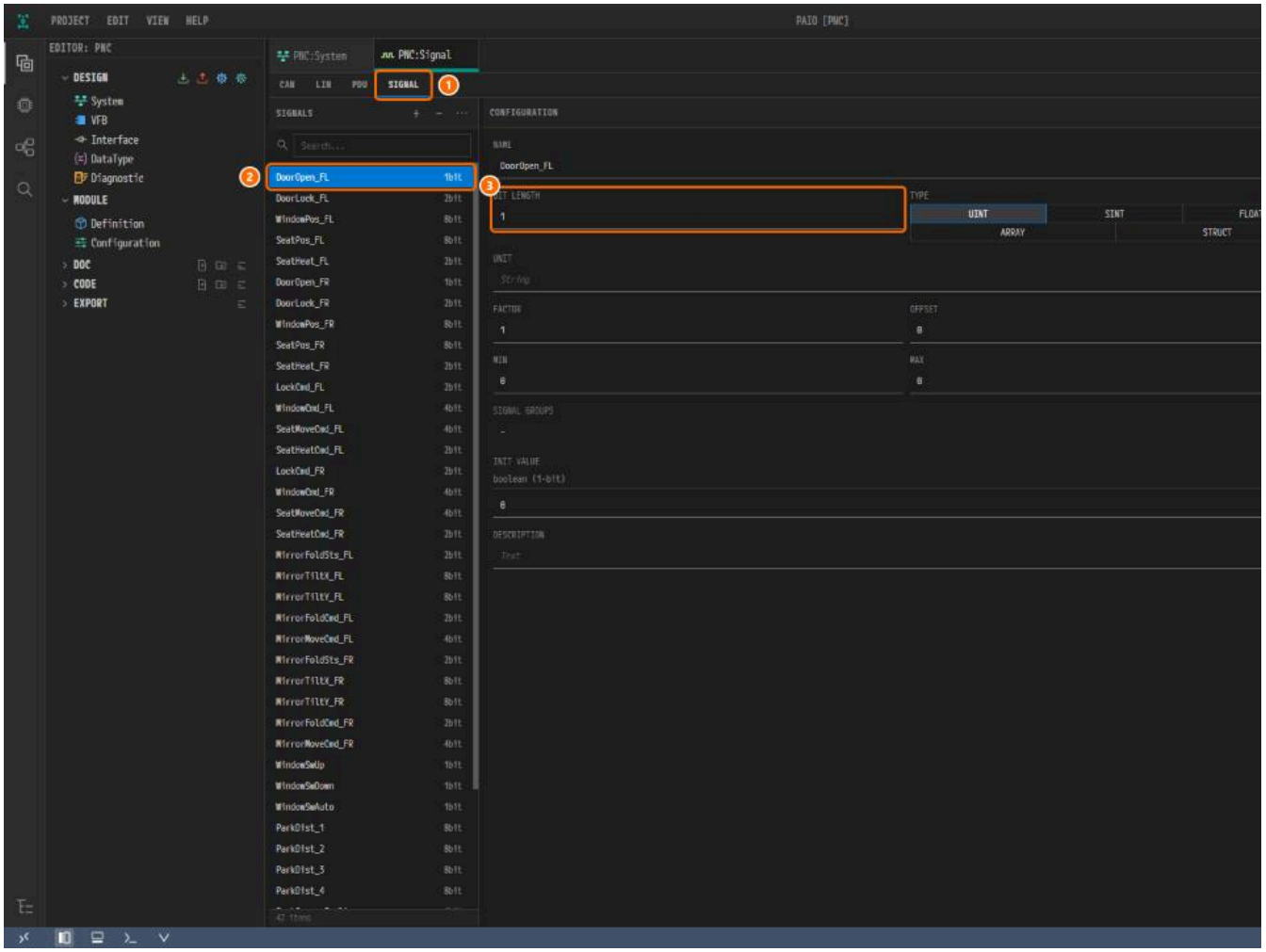


*Figure 4-19. ① **SIGNAL**, ② `DoorOpen_FL`, ③ **BIT LENGTH**.*

Looking at an NM Frame

Only the example project has NM frames.

1. On the **CAN** tab, click `WestCAN_NmFrame_WestZone` (`0x510`).

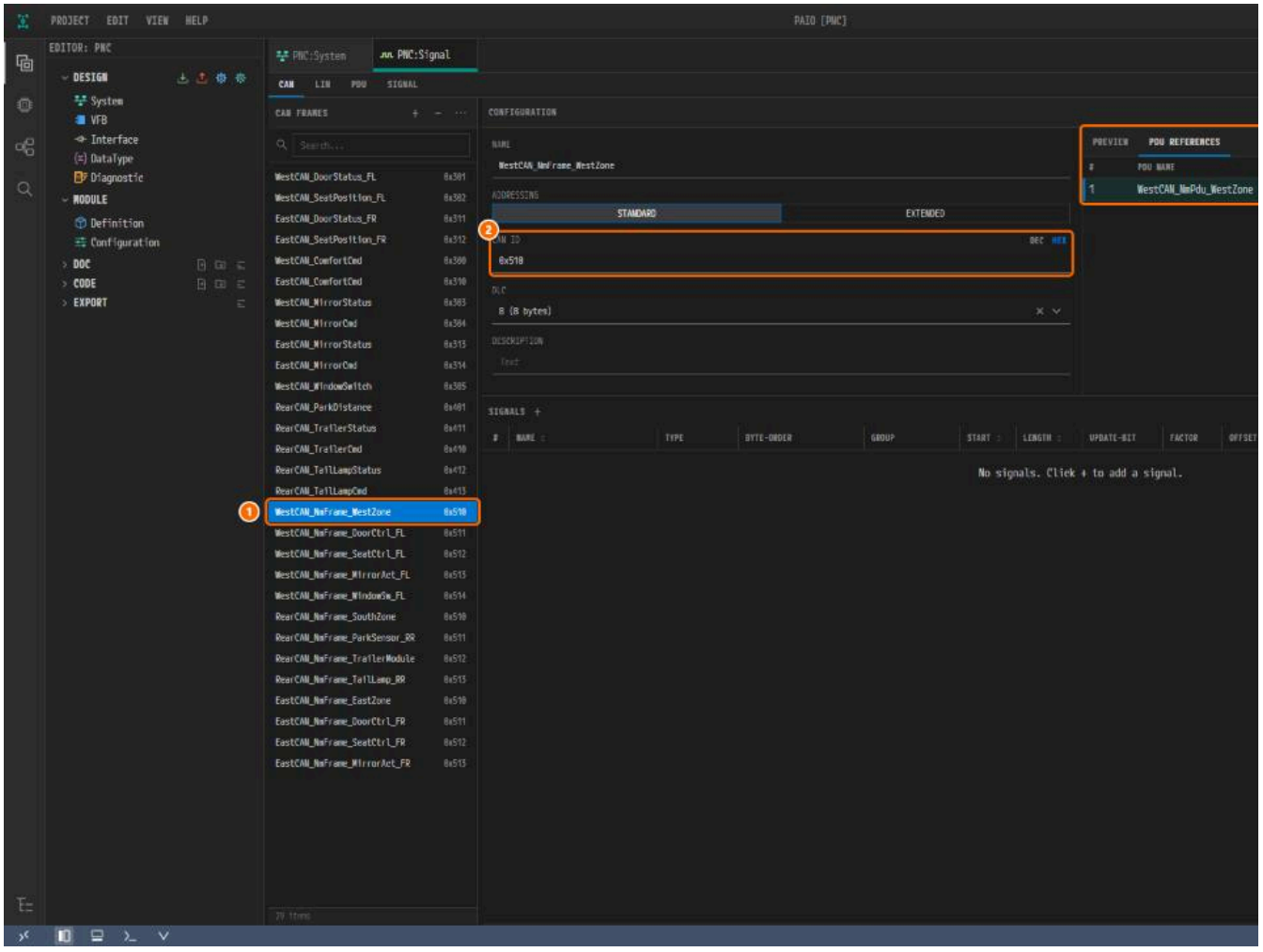


*Figure 4-20. ① `WestCAN_NmFrame_WestZone`, ② **CAN ID**, ③ `WestCAN_NmPdu_WestZone` on **PDU REFERENCES**.*

Each NM frame `{cluster}_NmFrame_{node}` carries `{cluster}_NmPdu_{node}`. The IDs start at `0x510` in each cluster.

Cluster	`0x510`	`0x511`	`0x512`	`0x513`	`0x514`
WestCAN	`WestZone`	`DoorCtrl_FL`	`SeatCtrl_FL`	`MirrorAct_FL`	`WindowSw_FL`
RearCAN	`SouthZone`	`ParkSensor_RR`	`TrailerModule`	`TailLamp_RR`	—
EastCAN	`EastZone`	`DoorCtrl_FR`	`SeatCtrl_FR`	`MirrorAct_FR`	—

Practice: Adding a Signal

The practice signal is not part of the example. You discard it at the end.

1. On the **CAN** tab, click `WestCAN_DoorStatus_FL`, then the **PREVIEW** tab. Click **+** (**Add Signal**).



*Figure 4-21. ① **+**.*

2. A fourth row `WestCAN_DoorStatus_FL_Sig` appears at **START** `16`.

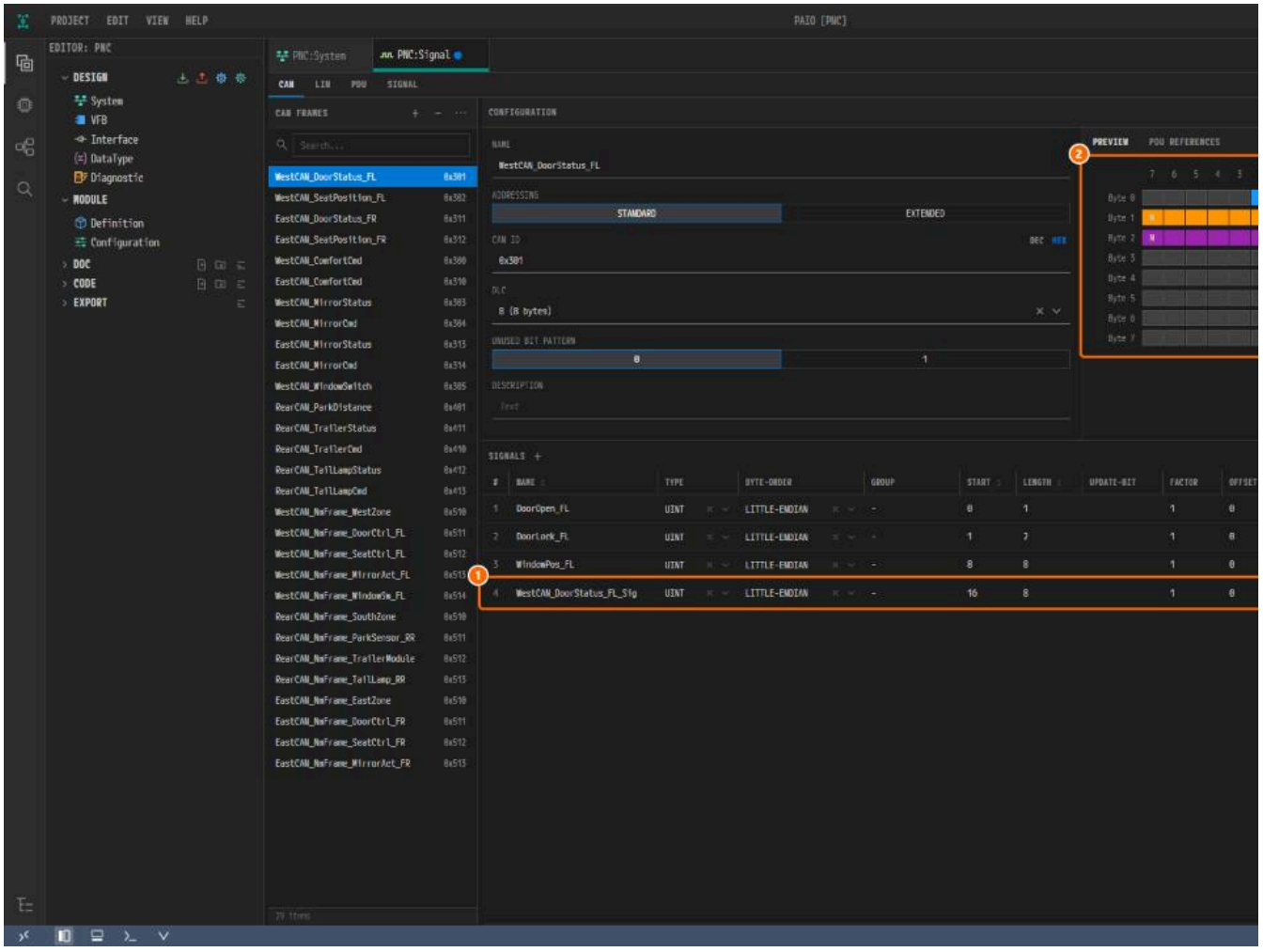


*Figure 4-22. ① The new row, ② Byte 2 in **PREVIEW**.*

3. Click its **NAME** cell, type `ChildLock_FL` and press `Enter`.

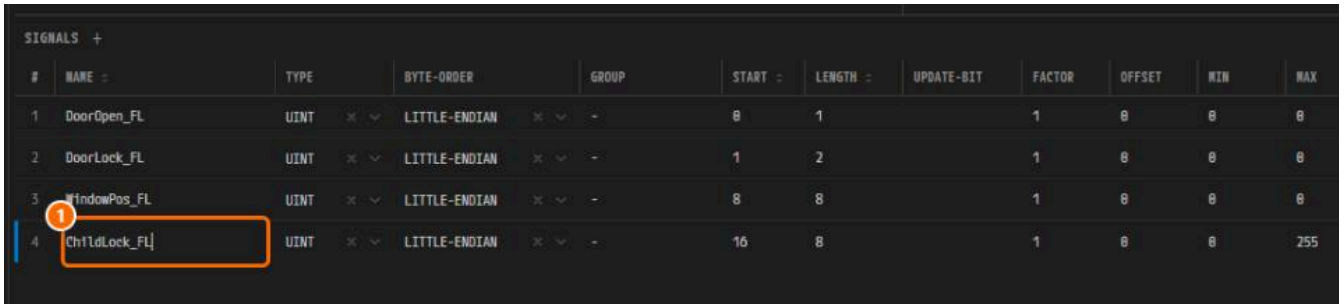


*Figure 4-23. ① **NAME**.*

4. Set **LENGTH** to `2`. Only bits 1–0 of Byte 2 are colored.

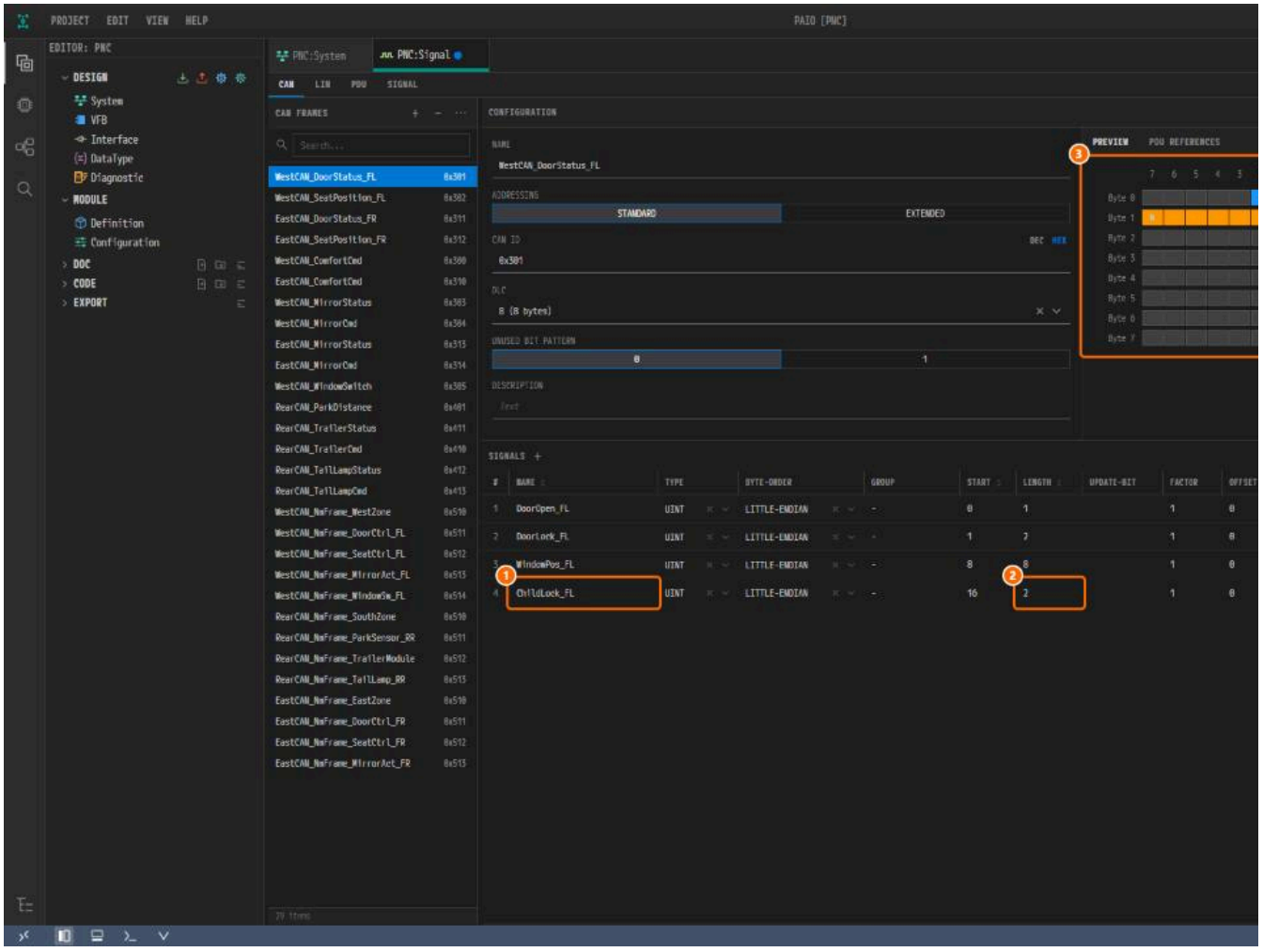


*Figure 4-24. ① **NAME**, ② **LENGTH**, ③ bits 1–0 of Byte 2.*

5. Click **×** (**Remove Signal**) at the right end of the row.
6. Close the **PNC:Signal** tab with its **×** and click **Don't Save**.

> **Caution.** Close the tab without saving before you open the PNC dialog. Saving the PNC dialog also saves a modified Signal editor that is still open.

Creating I-PDU Groups

Creating the Group

1. Open the System editor and click the lower icon **Partial Network (PNC)** on the left edge of the canvas (Figure 3-1).
2. In **PARTIAL NETWORKS**, confirm that **PNC 17 PNC_Comfort** is selected.

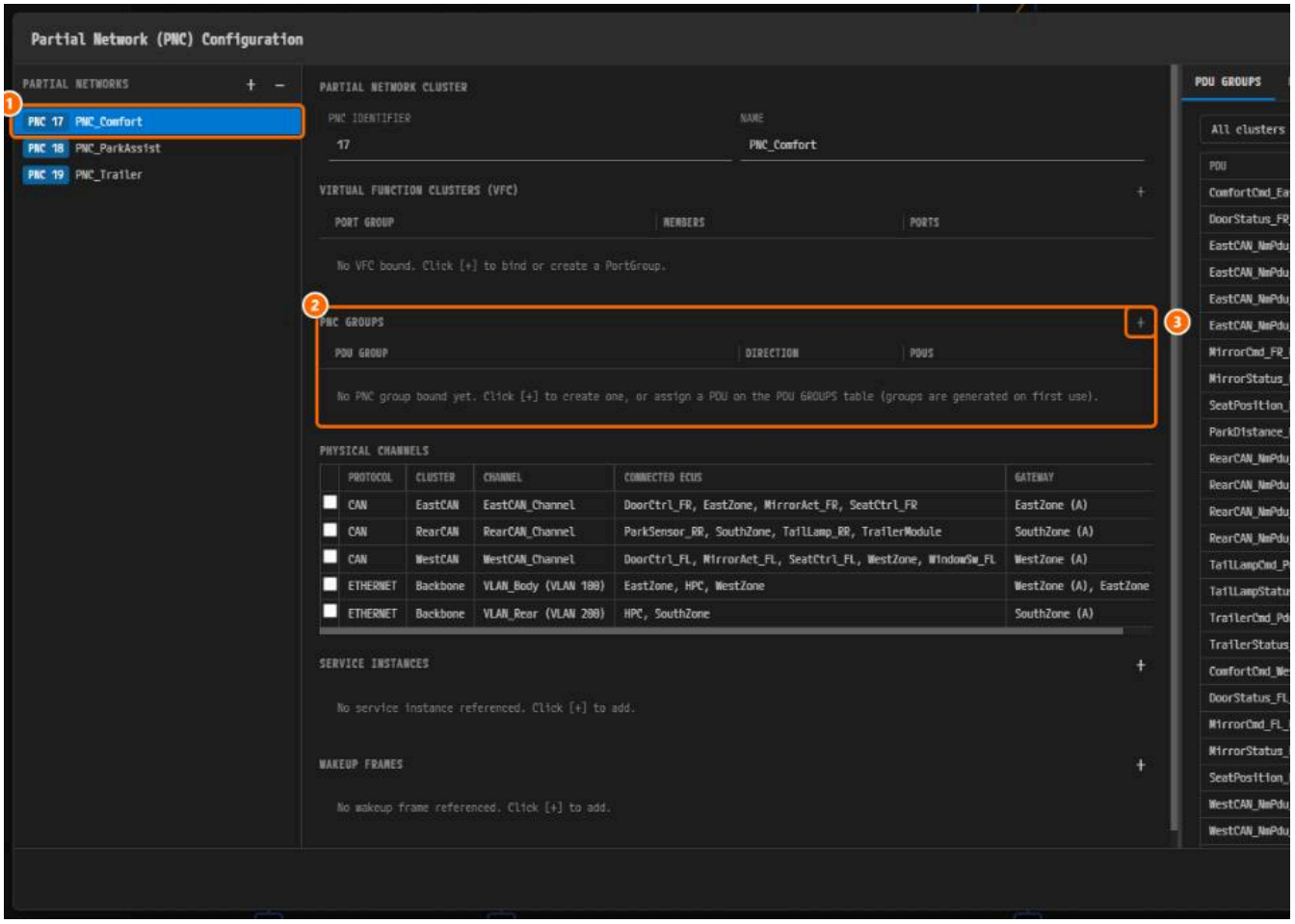


*Figure 4-25. ① **PNC 17 PNC_Comfort**, ② **PNC GROUPS**, ③ **+**.*

3. Click **+** on the **PNC GROUPS** header, then **New Rx group...**.

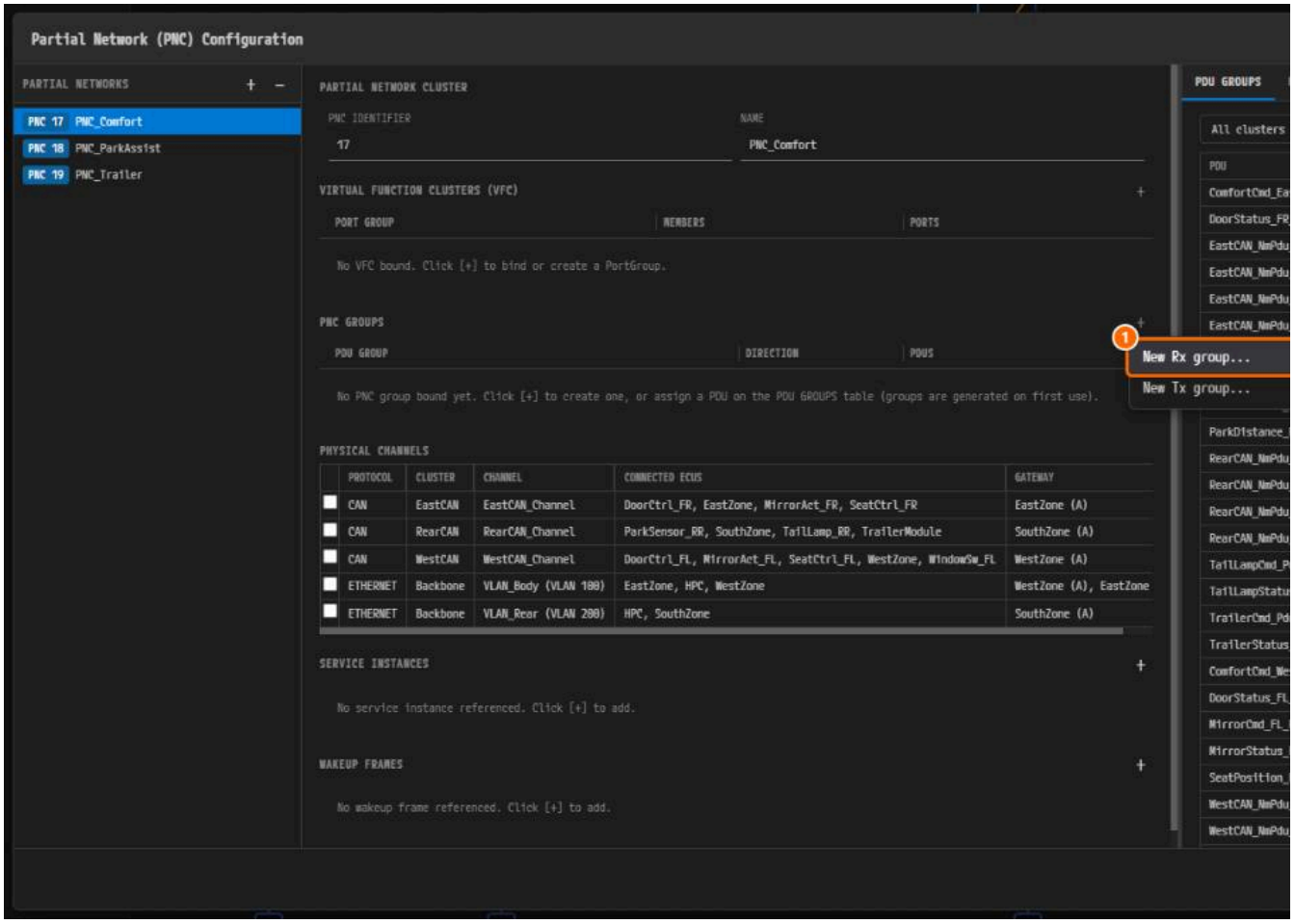


*Figure 4-26. ① **New Rx group.... New Tx group...** creates a transmit group.*

4. Enter `WestCAN_PNC_Comfort_Rx` and click **OK**. PAIO does not enforce the Naming Convention, so type the name exactly.

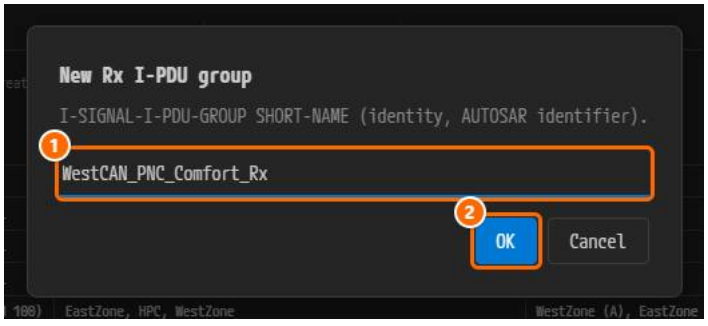


*Figure 4-27. ① Name field, ② **OK**.*

5. The group row appears, expanded. It is bound to the PNC selected in **PARTIAL NETWORKS**.

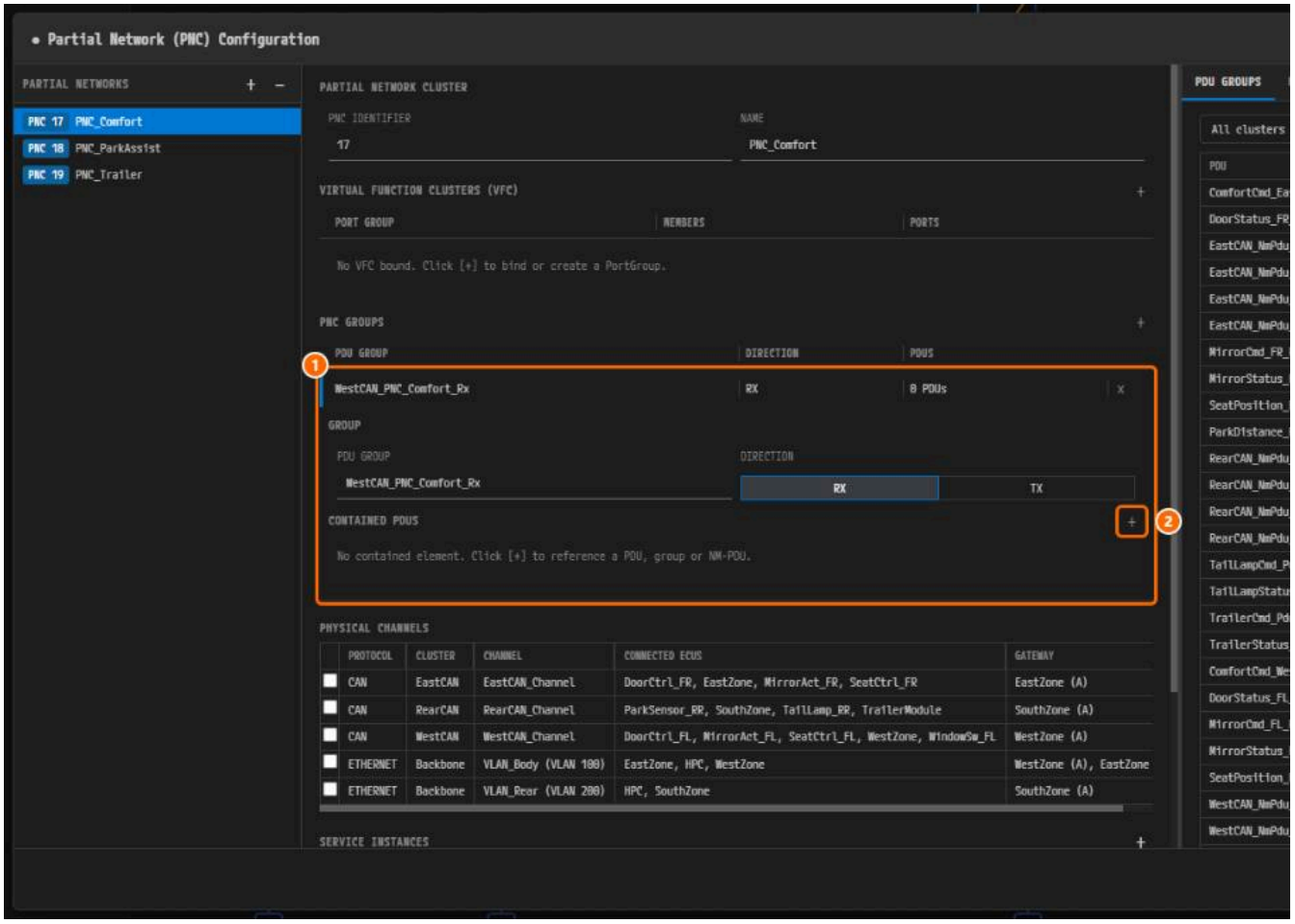


*Figure 4-28. ① The new group, ② **+** of **CONTAINED PDUS**.*

Putting PDUs into the Group

1. Click **+** (**Add contained element**) on the **CONTAINED PDUS** header. **KIND** is **I-SIGNAL-I-PDU**.
2. Click the **REFERENCE** cell and type `DoorStatus`.

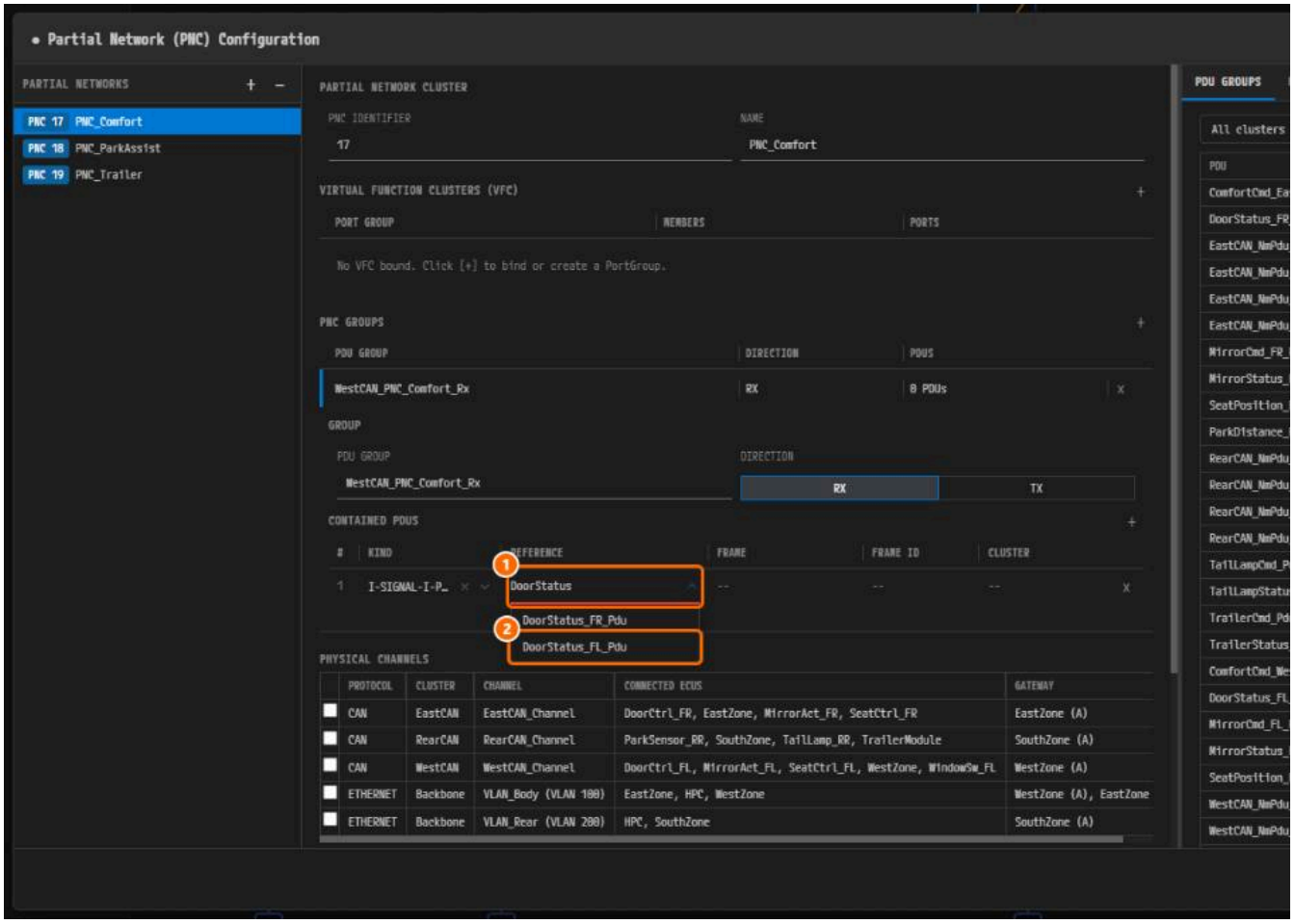


*Figure 4-29. ① **REFERENCE**, ② `DoorStatus_FL_Pdu`.*

3. Click `DoorStatus_FL_Pdu`. **FRAME**, **FRAME ID** and **CLUSTER** fill in.

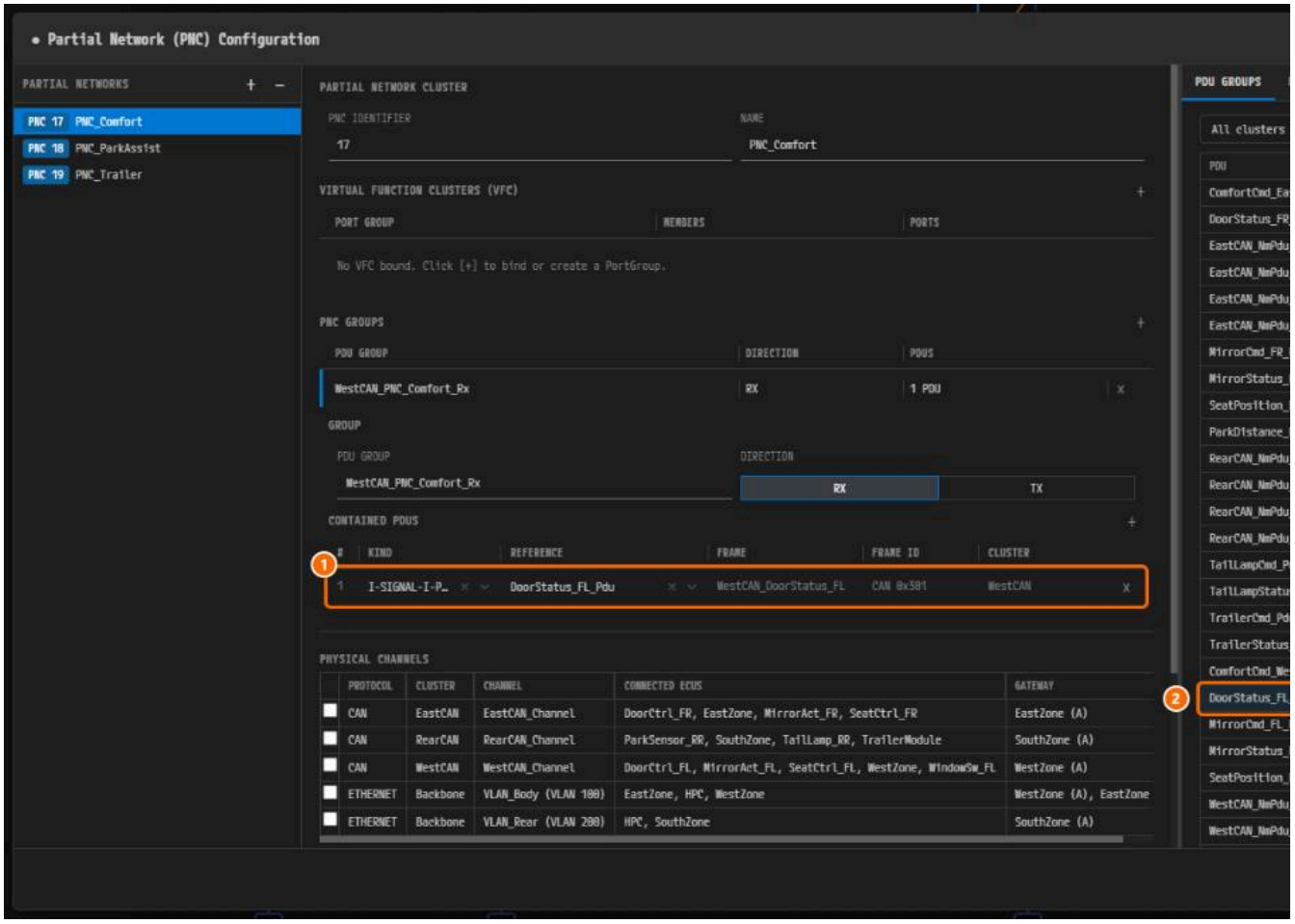


*Figure 4-30. ① The **CONTAINED PDUS** row, ② the same PDU in the **PDU GROUPS** table.*

4. Save with `Ctrl+S`.

> **Caution.** The **REFERENCE** list also offers NM-PDUs (`*_NmPdu_*`). Do not put them into a PNC group. The **×** on a group row deletes the group itself.

Creating the Remaining Groups

- Add `SeatPosition_FL_Pdu`, `MirrorStatus_FL_Pdu` and `WindowSwitch_Pdu` to `WestCAN_PNC_Comfort_Rx`.
- Select the PNC in **PARTIAL NETWORKS** before you create its groups.
- `RearCAN_PNC_ParkAssist_Tx` stays empty.

PNC	Group	DIRECTION	CONTAINED PDUS
PNC 17 `PNC_Comfort`	`WestCAN_PNC_Comfort_Rx`	**RX**	`DoorStatus_FL_Pdu`, `SeatPosition_FL_Pdu`, `MirrorStatus_FL_Pdu`, `WindowSwitch_Pdu`
PNC 17 `PNC_Comfort`	`WestCAN_PNC_Comfort_Tx`	**TX**	`ComfortCmd_West_Pdu`, `MirrorCmd_FL_Pdu`
PNC 17 `PNC_Comfort`	`EastCAN_PNC_Comfort_Rx`	**RX**	`DoorStatus_FR_Pdu`, `SeatPosition_FR_Pdu`, `MirrorStatus_FR_Pdu`
PNC 17 `PNC_Comfort`	`EastCAN_PNC_Comfort_Tx`	**TX**	`ComfortCmd_East_Pdu`, `MirrorCmd_FR_Pdu`
PNC 18 `PNC_ParkAssist`	`RearCAN_PNC_ParkAssist_Rx`	**RX**	`ParkDistance_Pdu`
PNC 18 `PNC_ParkAssist`	`RearCAN_PNC_ParkAssist_Tx`	**TX**	(none)
PNC 19 `PNC_Trailer`	`RearCAN_PNC_Trailer_Rx`	**RX**	`TrailerStatus_Pdu`, `TailLampStatus_Pdu`
PNC 19 `PNC_Trailer`	`RearCAN_PNC_Trailer_Tx`	**TX**	`TrailerCmd_Pdu`, `TailLampCmd_Pdu`

Save with `Ctrl+S` and click **Close**.

The Finished Result

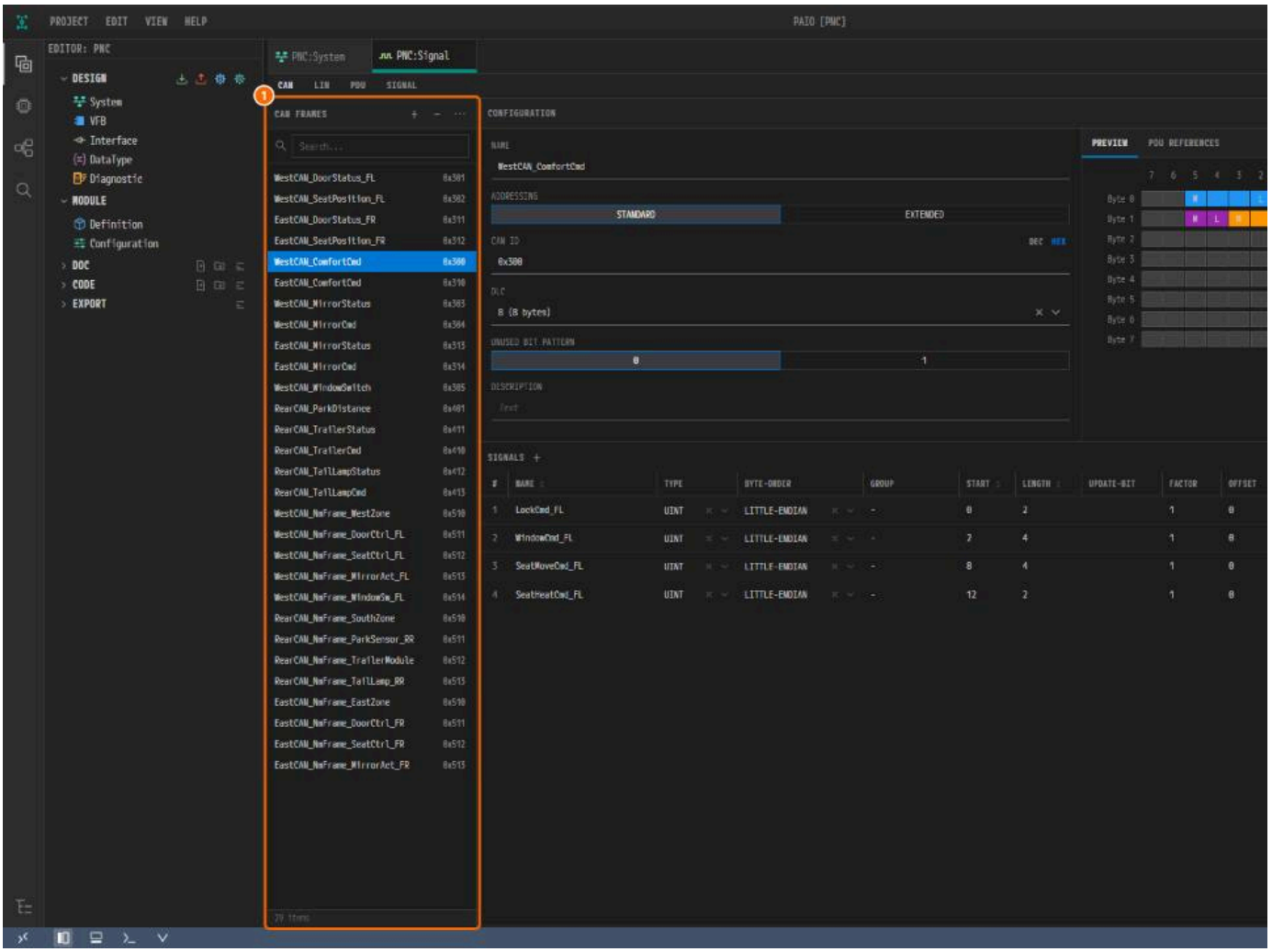


*Figure 4-31. ① **CAN FRAMES** (29 items) with `WestCAN_ComfortCmd` selected.*

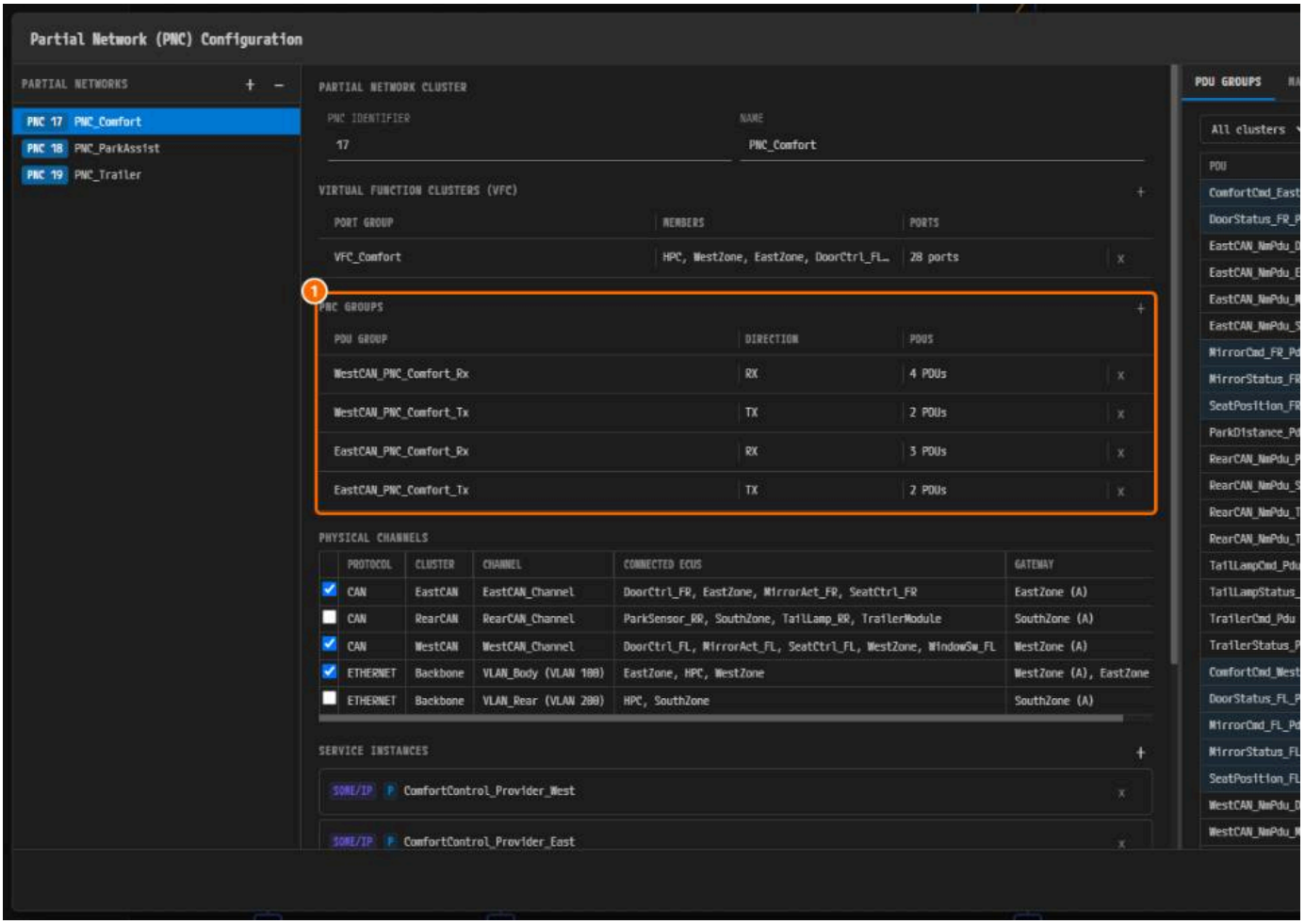


*Figure 4-32. ① The four groups of **PNC 17 PNC_Comfort**. The sections below them are filled in PNC Dialog — Per-PNC Bindings.*

Click a group row to see its **CONTAINED PDUS**.

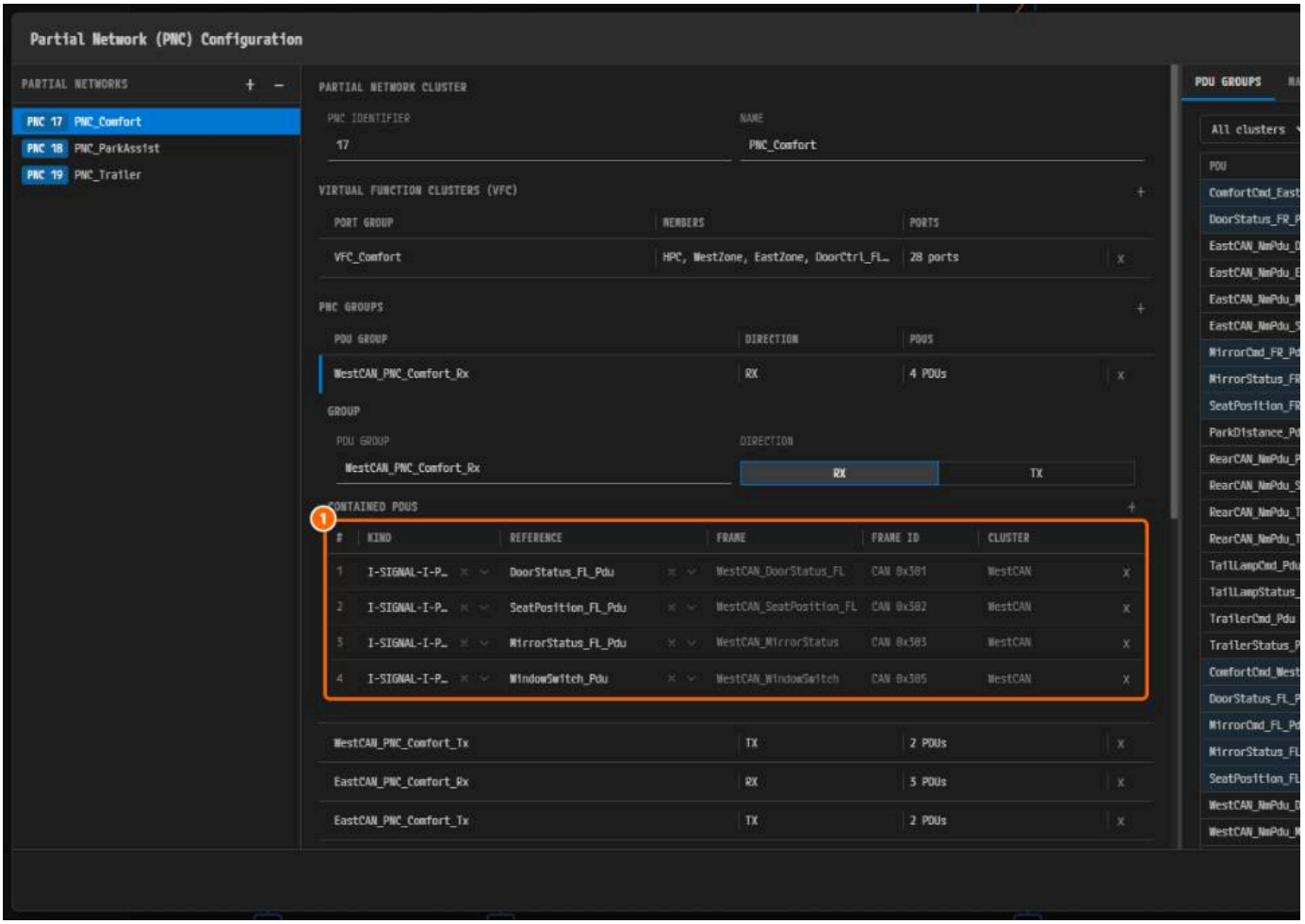


Figure 4-33. ① The four PDUs of `WestCAN_PNC_Comfort_Rx`.

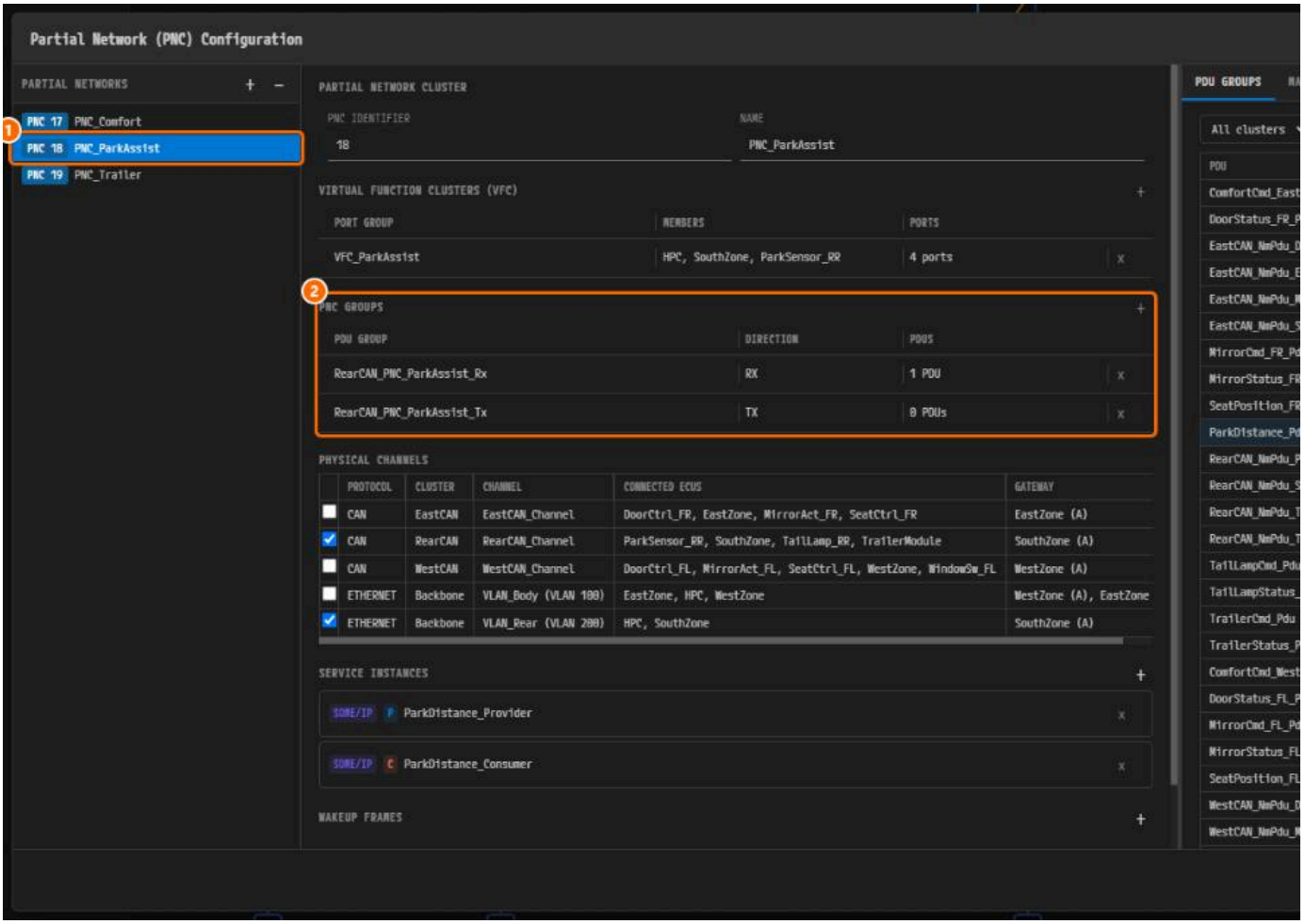


*Figure 4-34. ① **PNC 18 PNC_ParkAssist**, ② its two groups.*

Naming Convention

PAIO names I-PDU groups by this convention, and ARXML import and export use it too.

Form	Name	Example	Contents in this example
Container	`{cluster}_Rx` / `{cluster}_Tx`	`RearCAN_Rx`	The leaf groups of that bus
PNC leaf	`{cluster}_{pnc}_Rx` / `{cluster}_{pnc}_Tx`	`WestCAN_PNC_Comfort_Rx`	The PDUs of that PNC on that bus
Non-participating leaf	`{cluster}_NoPnc_Rx` / `{cluster}_NoPnc_Tx`	`WestCAN_NoPnc_Tx`	PDUs of no PNC. Empty in this example.
NM-PDU	`{cluster}_NmPdu_{node}`	`WestCAN_NmPdu_WestZone`	One per node, 8 bytes
NM frame	`{cluster}_NmFrame_{node}`	`WestCAN_NmFrame_WestZone`	From `0x510` in node order

- `_Rx` / `_Tx` is seen from the Zone (gateway). A status PDU from an end ECU is `_Rx`; a command PDU to an end ECU is `_Tx`.
- Only the PNC leaf groups are bound to a PNC.

Container and NoPnc groups come from ARXML import and do not appear in **PNC GROUPS**. Do not create them with the **+** of **PNC GROUPS**. A project you bu
Appendix). In the finished example you see them as candidates on a **KIND** = **GROUP** row.

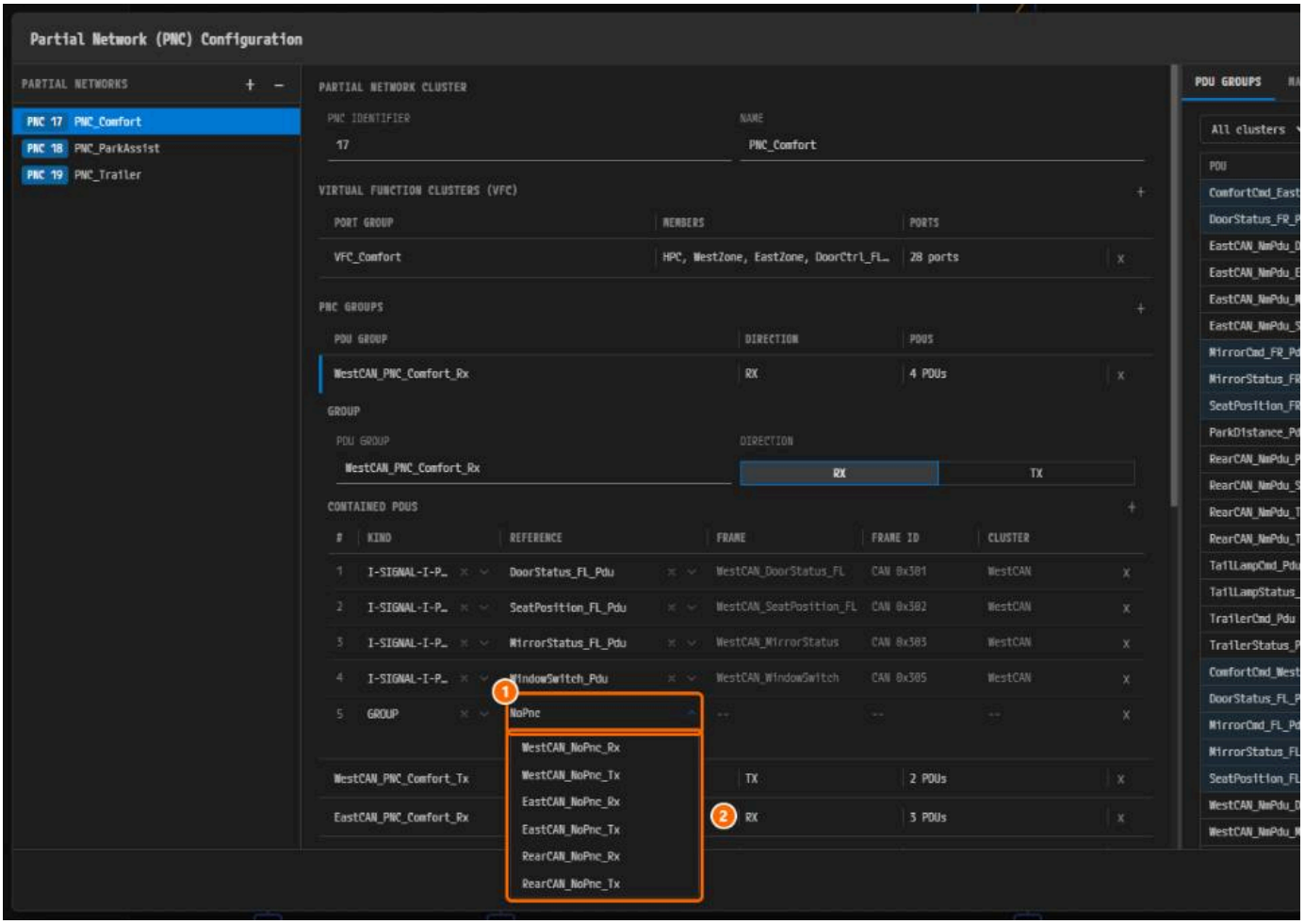


*Figure 4-35. ① **REFERENCE**, ② the `NoPnc` candidates. For inspection only: press `Esc` and delete the row with ×.*

Why Is It Designed This Way?

- **Groups per PNC.** A PNC starts and stops its I-PDU groups, so each bus has an Rx and a Tx group per PNC.
- **No NM-PDUs in PNC groups.** NM must run whether a PNC is on or off.
- **Separate `_FL` and `_FR` signals.** Mapping one I-Signal to two PDUs violates `constr_3514`.

Checklist

- [ ] The Signal editor holds 34 PDUs (**IS** 16, **NM** 18), 42 signals and 16 application frames (29 frames in the finished example).
- [ ] `DoorStatus_FL_Pdu` has **CYCLE TIME (MS)** `50` and **TRANSMISSION MODE CYCLIC**.
- [ ] The practice signal `ChildLock_FL` is gone.
- [ ] **PNC GROUPS** holds 4 groups for PNC 17, 2 for PNC 18 and 2 for PNC 19, as in the table, with no NM-PDU.
- [ ] Neither the **PNC:Signal** tab nor the PNC dialog title carries a modified mark.

Next Step

Go back to NM Settings to fill the NM-PDU references, then continue with Interfaces and SOME/IP Services.

Interfaces and SOME/IP Services

What You Will Do

In this chapter you create the Sender-Receiver interfaces for the CP ports and the service interfaces for SOME/IP between the HPC and the zones. You ac to each service, give the Ethernet connectors IP addresses and UDP ports, and bind the instances to the connectors.

Before You Begin

- Finish Signals, PDUs, Frames, and I-PDU Groups.
- The Ethernet connectors `HPC_Eth_Body_Conn`, `HPC_Eth_Rear_Conn`, `WestZone_Eth_Conn`, `EastZone_Eth_Conn`, and `SouthZone_Eth_Conn` exist (fr
- You reuse the service names, Service IDs, and instance names below verbatim in The PNC Dialog — Binding Per PNC.

Service interface	Service ID	Provider	
`SVC_ParkDistance` (events DistanceUpdate, SensorFault / method SetSensitivity)	`0x1001`	SouthZone (`ParkDistance_Provider`)	HPC Park (`ParkDis`
`SVC_VehicleState` (events GearPosition, TrailerAttached)	`0x1002`	HPC VehicleStateApp (`VehicleState_Provider`)	SouthZor
`SVC_ComfortControl` (event DoorStatus / methods LockAll, FoldMirrors)	`0x1003`	WestZone (`ComfortControl_Provider_West`, instance 1), EastZone (`ComfortControl_Provider_East`, instance 2)	HPC Com (`Comfort`

Creating the Sender-Receiver Interfaces

1. In the **EDITOR** view, click **Interface** under **DESIGN**. The `PNC:Interface` tab opens.
2. Click the **SENDER-RECEIVER** tab.

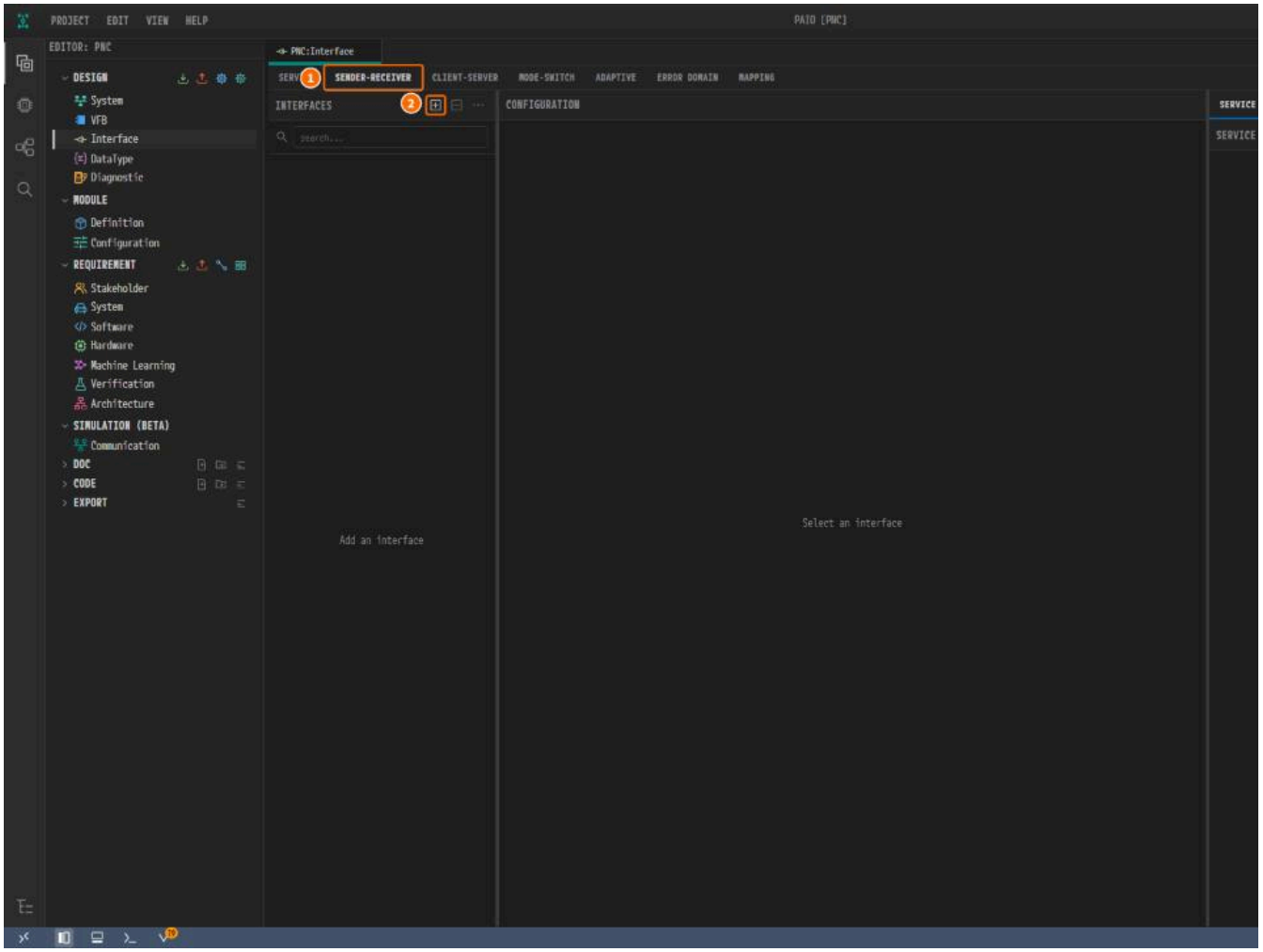


*Figure 5-1. ① **SENDER-RECEIVER**, ② **+** of **INTERFACES**.*

3. Click **+** to the right of **INTERFACES**. `SenderReceiverInterface_0` is added.

4. Enter `SRI_ParkDistance` in **NAME** and press `Enter`.

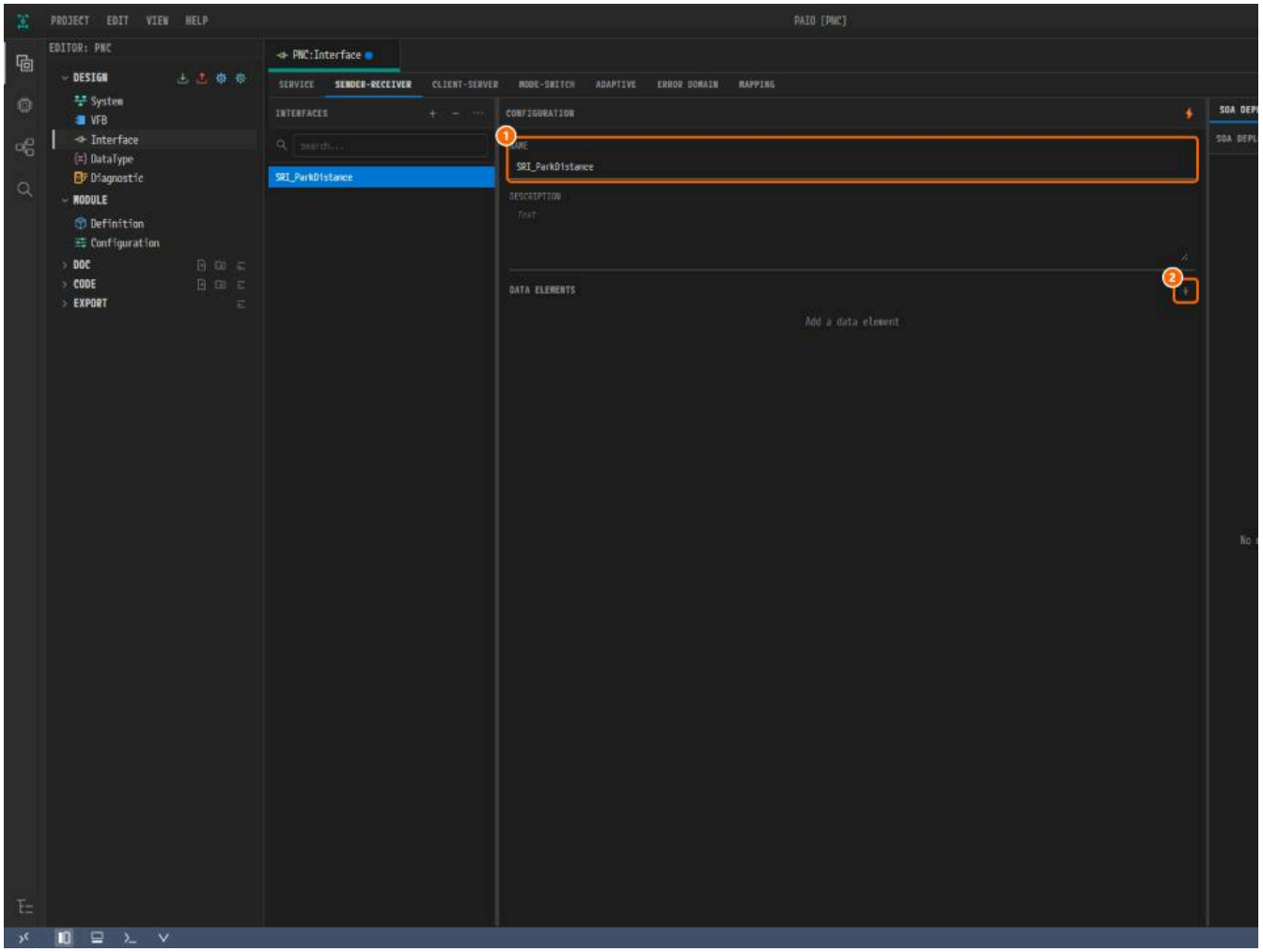


*Figure 5-2. ① **NAME**, ② **+** of **DATA ELEMENTS**.*

5. Click **+** to the right of **DATA ELEMENTS**. A `DataElement_0` row (`uint8`) is added.

6. Enter `Distance1` in **NAME**.

7. Click the **TYPE** field to open the type list.

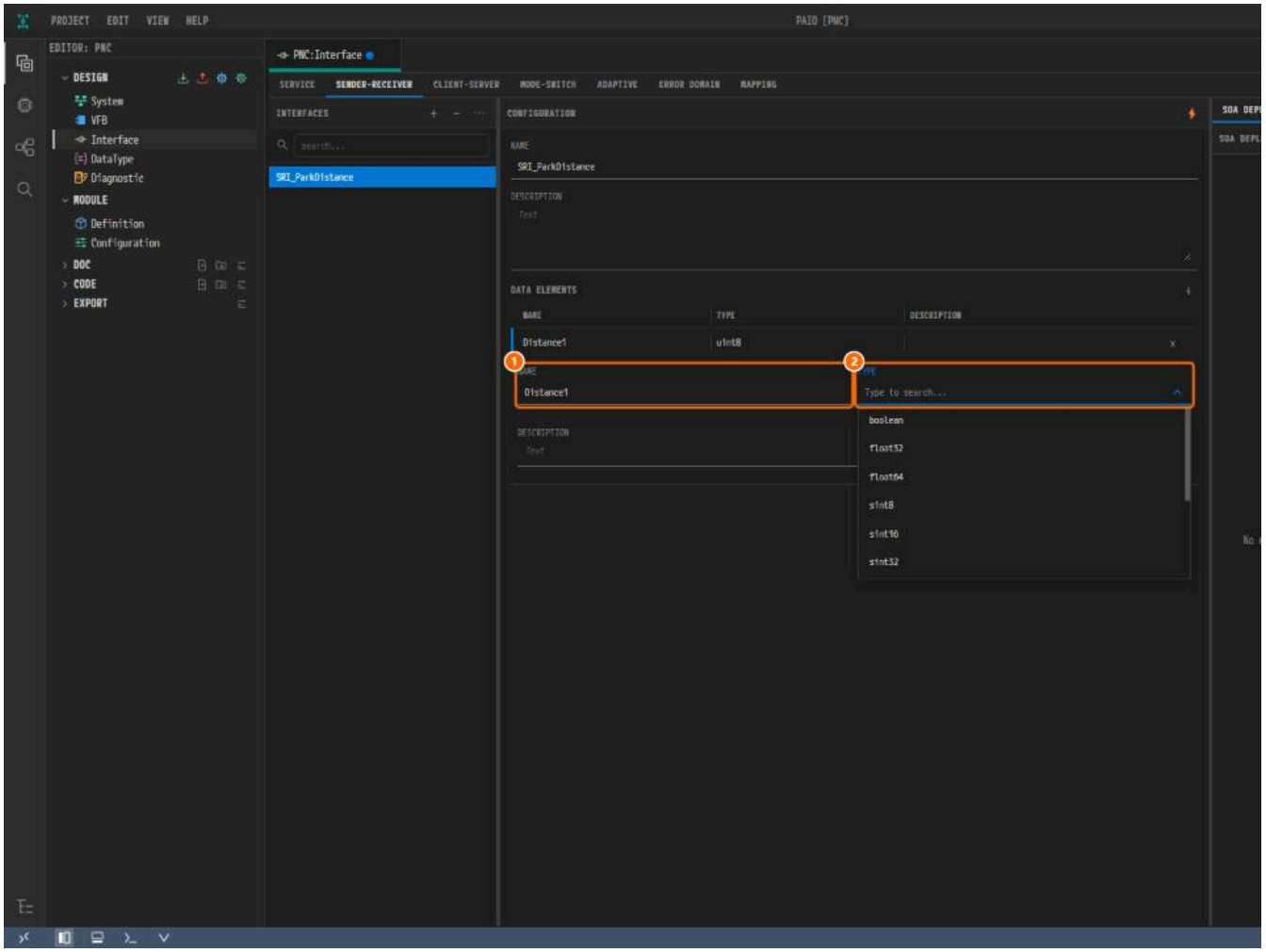


*Figure 5-3. ① **NAME**, ② the **TYPE** list. Scroll or type in the search box to find a type.*

8. Select `uint8` (the default, so you may also just close the list).

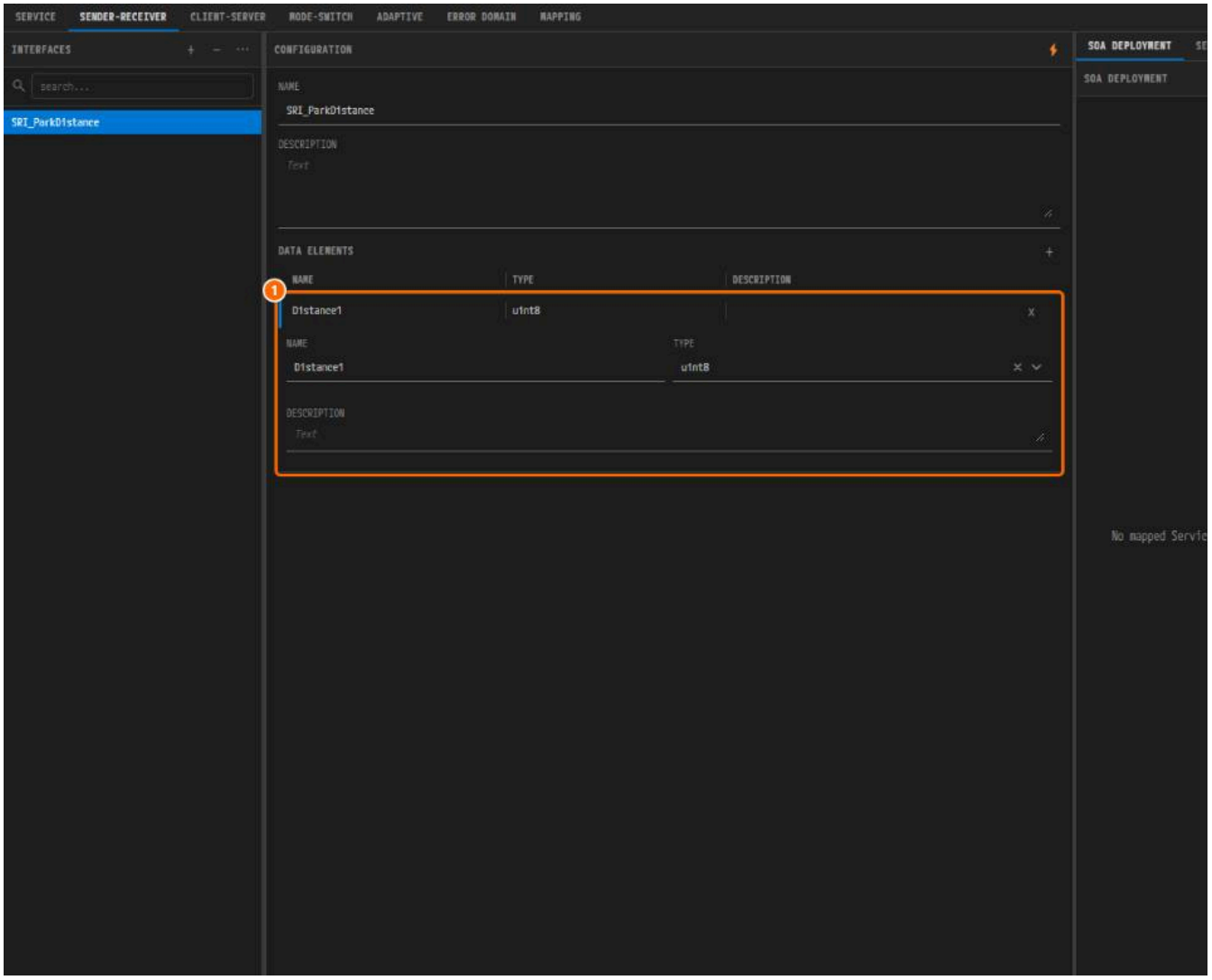


Figure 5-4. ① `Distance1` (`uint8`).

9. Add `Distance2`, `Distance3`, `Distance4` (`uint8`) and `SensorFault` (`boolean`) the same way.

The Remaining Sender-Receiver Interfaces

Create the others the same way from the table. Every type is a built-in primitive.

Interface	Data elements (type)
`SRI_DoorStatus`	`DoorOpen` (boolean), `LockState` (uint8), `WindowPos` (uint8)
`SRI_SeatStatus`	`SeatPos` (uint8), `SeatHeat` (uint8)
`SRI_ComfortCmd`	`LockCmd`, `WindowCmd`, `SeatMoveCmd`, `SeatHeatCmd` (all uint8)
`SRI_MirrorStatus`	`FoldState`, `TiltX`, `TiltY` (all uint8)
`SRI_MirrorCmd`	`FoldCmd`, `MoveCmd` (all uint8)
`SRI_ParkDistance`	`Distance1`, `Distance2`, `Distance3`, `Distance4` (uint8), `SensorFault` (boolean)
`SRI_TrailerStatus`	`Attached` (boolean), `LampFault` (uint8), `BrakeLamp` (boolean)
`SRI_TrailerCmd`	`LampCmd` (uint8)
`SRI_WindowSwitch`	`Up`, `Down`, `Auto` (all boolean)
`SRI_TailLamp`	`LampState` (uint8)
`SRI_GearPosition`	`Gear` (uint8)

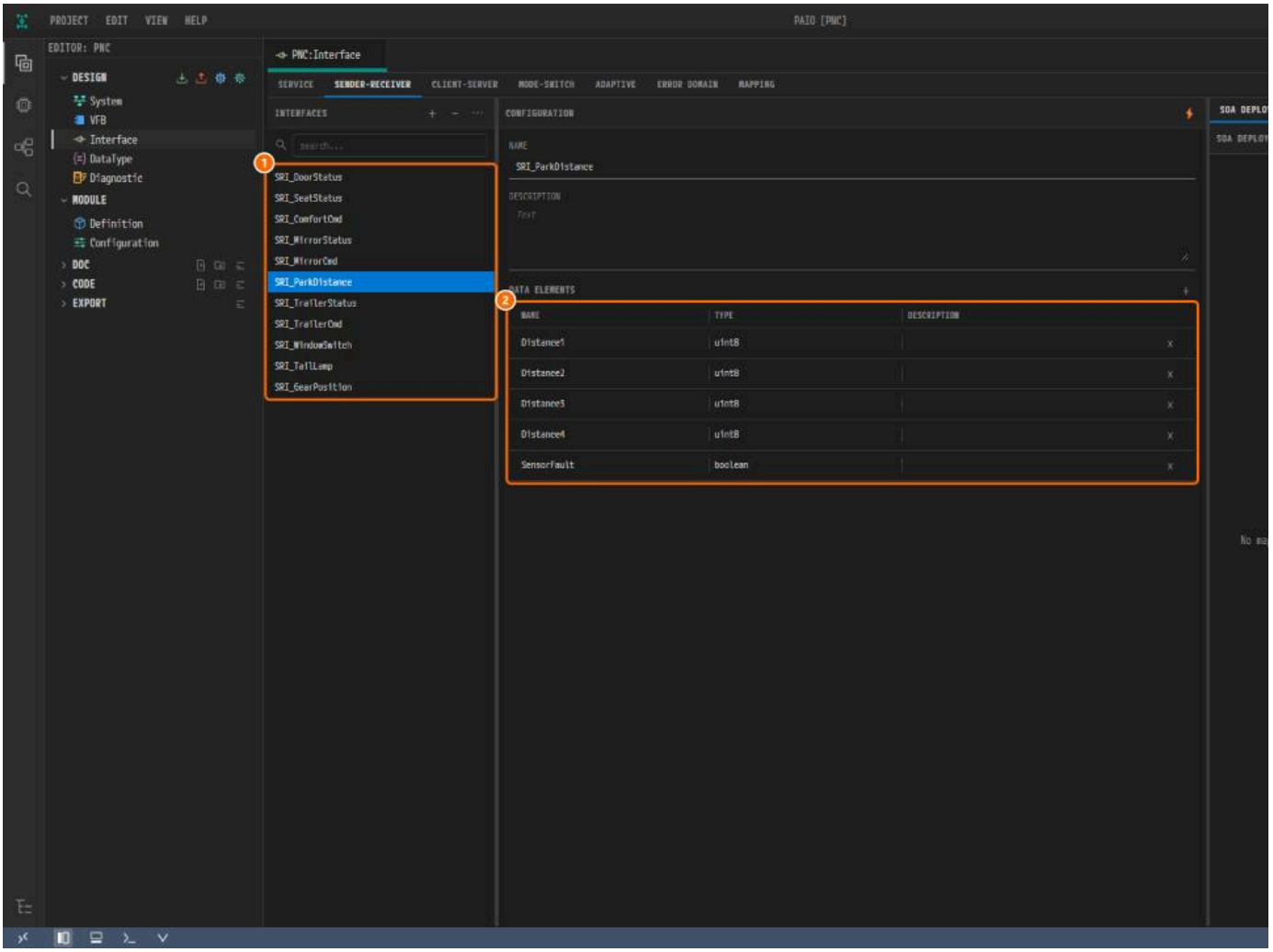


Figure 5-5. ① The 11 `SRI_*` interfaces, ② the data elements of `SRI_ParkDistance`.

Creating the Service Interface

1. Click the **SERVICE** tab.

2. Click **+** to the right of **INTERFACES**. `ServiceInterface_0` is added.

3. Enter `SVC_ParkDistance` in **NAME**.

4. Enter `pnc::park` in **C++ NAMESPACE** and press `Tab`. Leave **MAJOR VERSION** `1` and **MINOR VERSION** `0`.

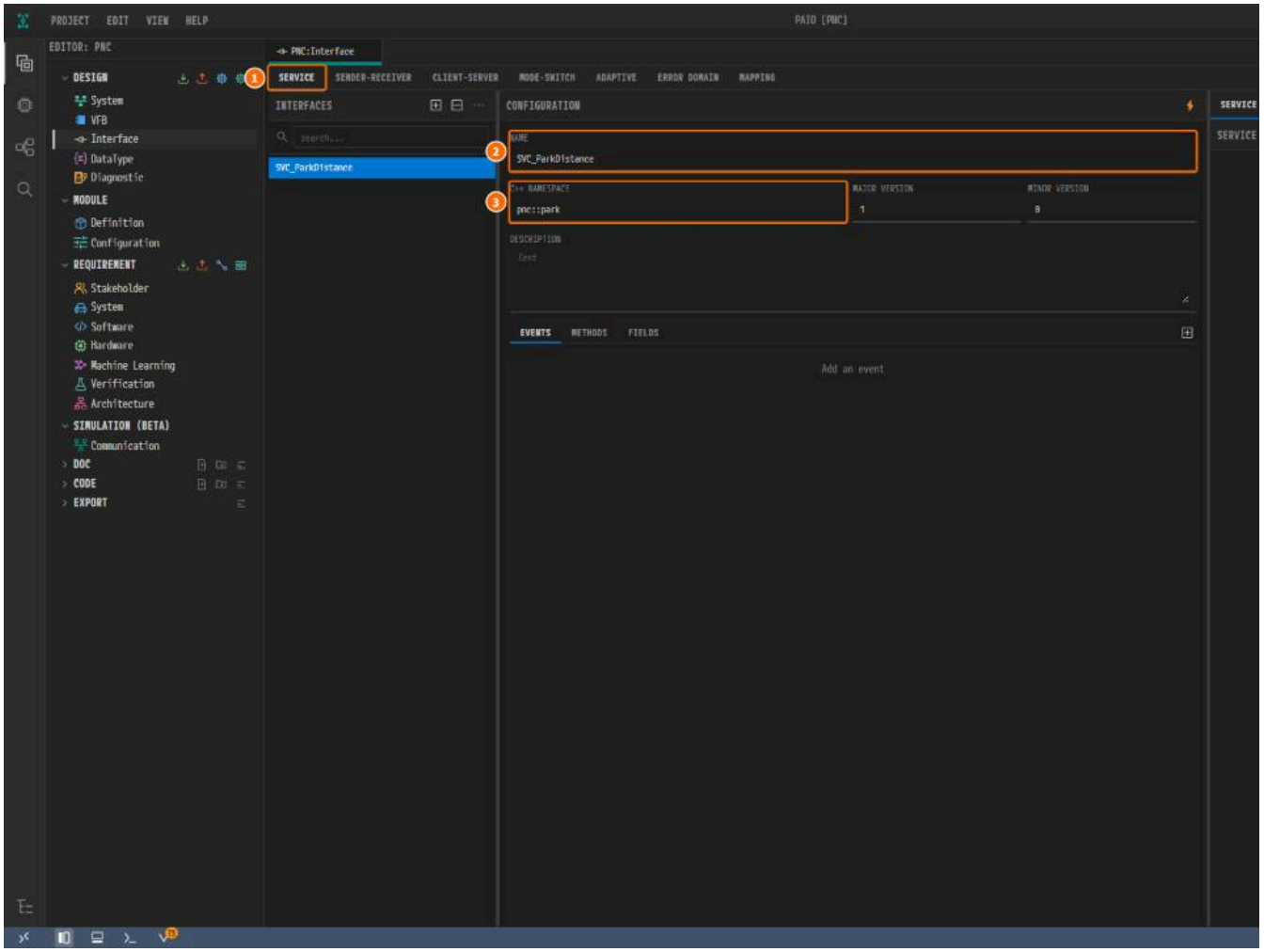


*Figure 5-6. ① **SERVICE**, ② **NAME**, ③ **C++ NAMESPACE**.*

Adding Events

1. Click the **EVENTS** subtab.

2. Click **+** at the right end of the subtab row. An `Event_0` row (`uint8`) is added.

3. Enter `DistanceUpdate` in **NAME**. Keep `uint8`.

4. Click **+** again, enter `SensorFault`, and select `boolean` in **TYPE**.

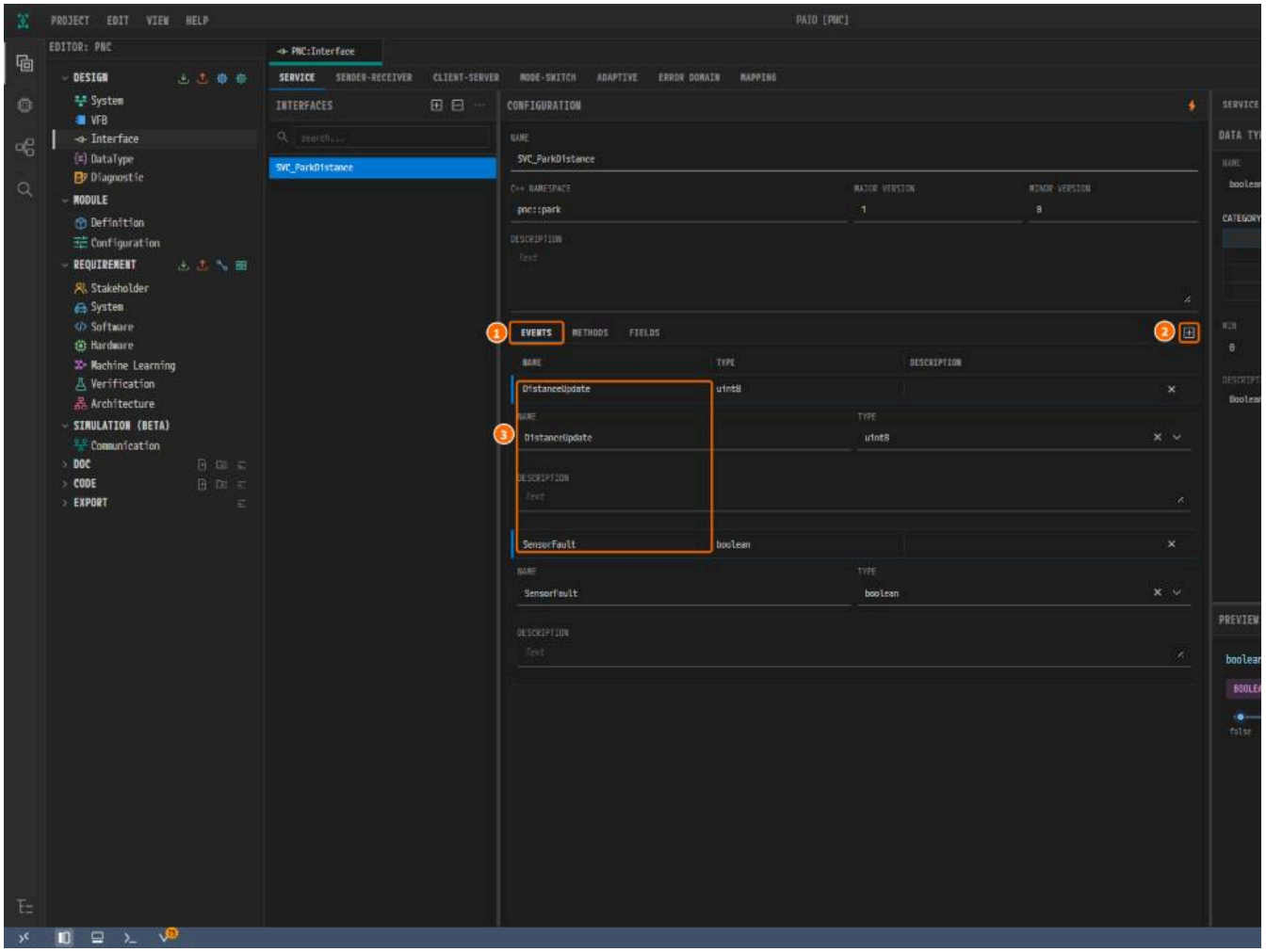


*Figure 5-7. ① **EVENTS**, ② **+**, ③ `DistanceUpdate` and `SensorFault`.*

> **Note.** Picking a **TYPE** switches the right-hand panel to **DATA TYPE**. Click **SERVICE DEPLOYMENT** to return.

Adding a Method

1. Click the **METHODS** subtab.
2. Click **+** at the right end of the subtab row. A `Method_0` row is added.
3. Enter `SetSensitivity` in **NAME**.
4. Leave **METHOD TYPE** at **REQUEST AND RESPONSE**.

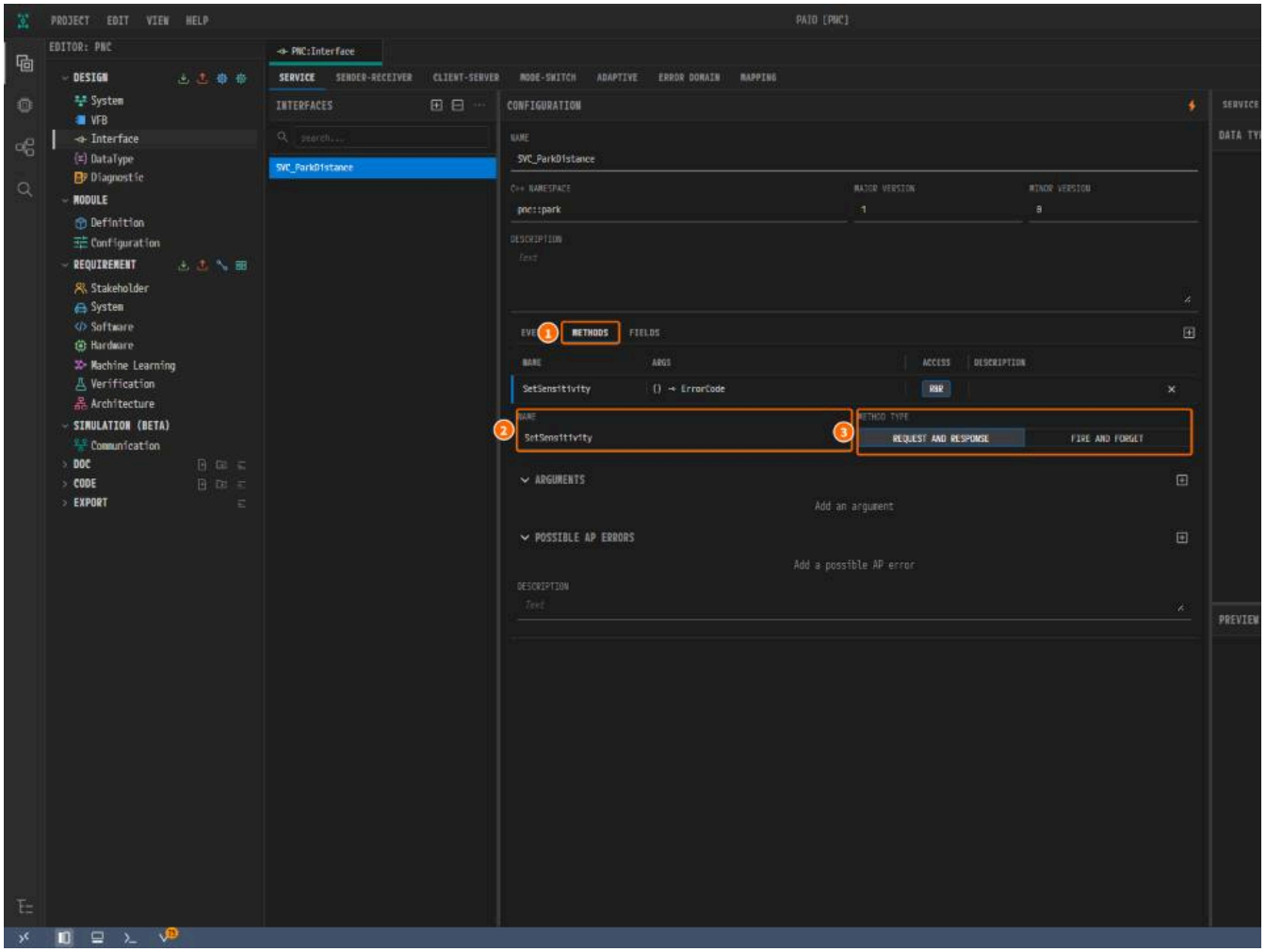


*Figure 5-8. ① **METHODS**, ② **NAME**, ③ **METHOD TYPE**. A method with no response uses **FIRE AND FORGET**.*

Adding the SOME/IP Deployment

A deployment takes the events and methods that exist when you add it, so add it last.

1. In the right-hand panel, click the **SERVICE DEPLOYMENT** subtab.
2. Click **+** to the right of **SERVICE DEPLOYMENT**.

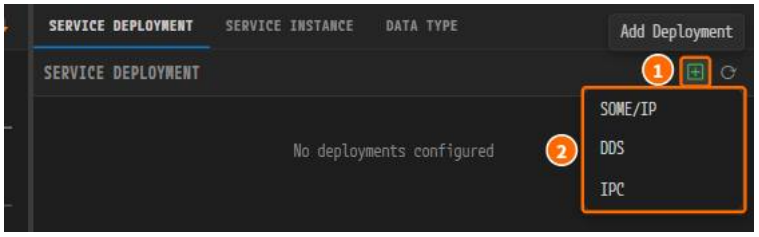


*Figure 5-9. ① **+**, ② the protocol menu.*

3. Select **SOME/IP**. A collapsed card **SOME/IP** `SVC_ParkDistance` appears.
4. Click the card header to expand it.
5. Click **HEX** next to **SERVICE ID**.
6. Enter `1001` in **SERVICE ID** and press `Tab`. It reads `0x1001`.
7. Check **MAJOR VERSION** `1` and **MINOR VERSION** `0`.

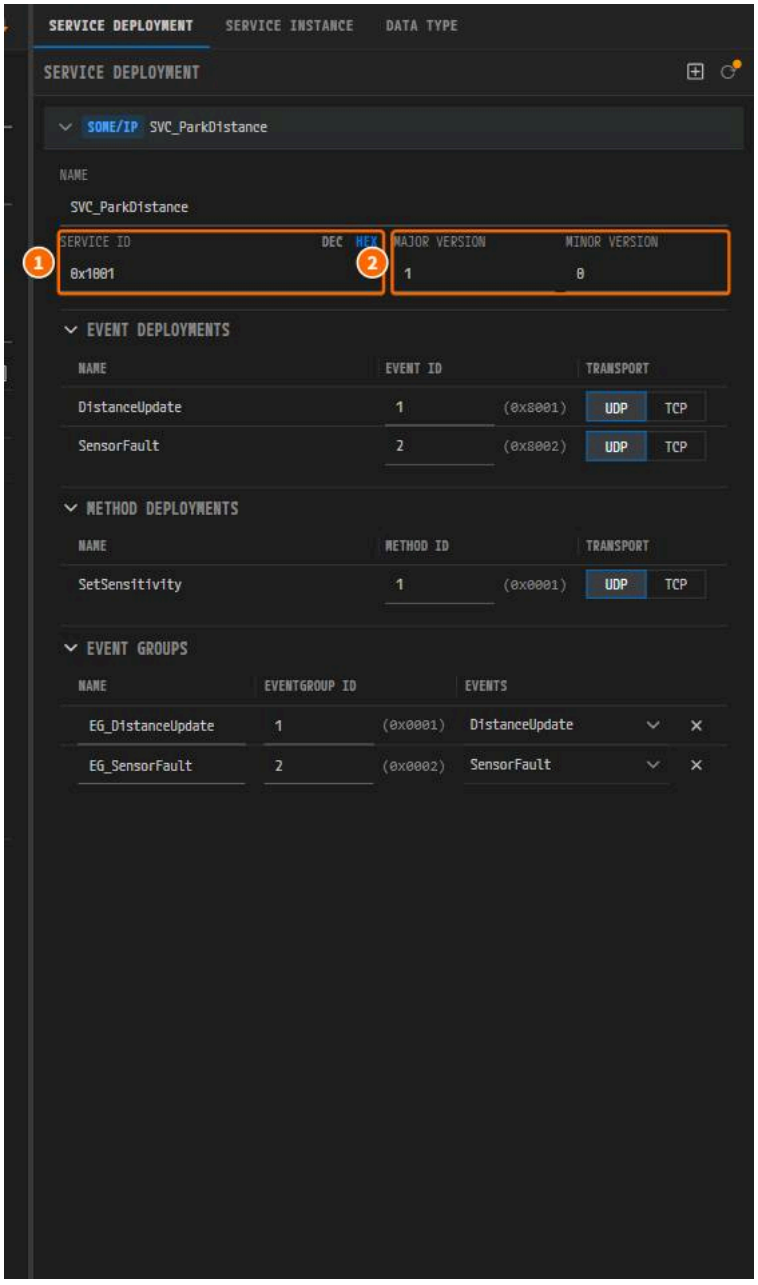


*Figure 5-10. ① **SERVICE ID**, ② the versions. Event and method deployments are filled in automatically, with one event group per event.*

8. Merge the two event groups into `EG_SVC_ParkDistance` (**EVENTGROUP ID** `1`):

   i. In **EVENT GROUPS**, click **x** on the `EG_SensorFault` row.
   ii. Rename `EG_DistanceUpdate` to `EG_SVC_ParkDistance`. Keep **EVENTGROUP ID** `1`.
   iii. Click its **EVENTS** field and also check `SensorFault`. It reads `DistanceUpdate, SensorFault`.

Adding Provider and Consumer Instances

1. Hover over the deployment card header. A **+** appears at its right.

2. Click **+** and select **Provider**.



*Figure 5-11. ① **+**, ② **Provider / Consumer**.*

3. Click **+** again and select **Consumer**.
4. Click the card header to collapse it.

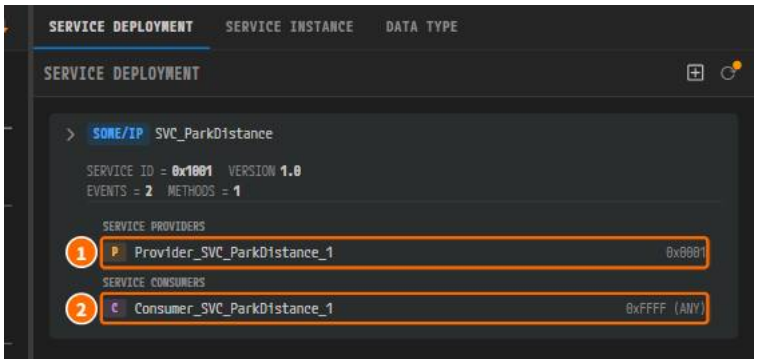


Figure 5-12. ① The default provider, ② the default consumer.

5. Click the provider row under **SERVICE PROVIDERS**. The **SERVICE INSTANCE** subtab opens.
6. Change **NAME** to `ParkDistance_Provider`.
7. Enter `1` in **INSTANCE ID** and press `Tab`.

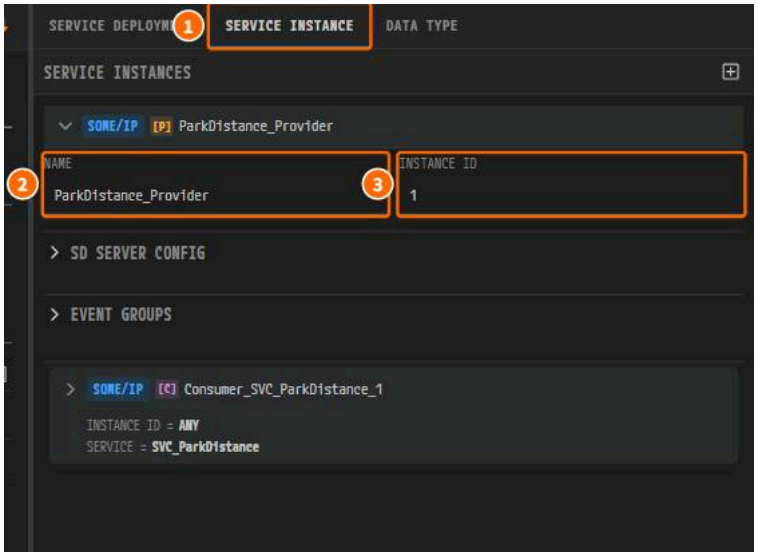


*Figure 5-13. ① **SERVICE INSTANCE**, ② **NAME**, ③ **INSTANCE ID**.*

8. Collapse the provider card and expand the consumer card.
9. Change **NAME** to `ParkDistance_Consumer`.
10. Enter `1` in **REQUIRED INSTANCE ID**.

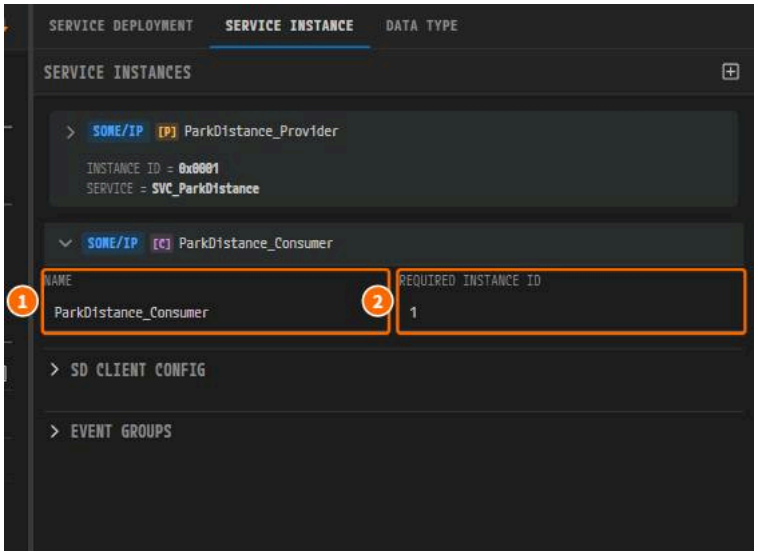


*Figure 5-14. ① **NAME**, ② **REQUIRED INSTANCE ID**.*

11. Save with `Ctrl+S`.

> **Note.** PAIO also generates `SVC_ParkDistance_SR` and four `SVC_ParkDistance_*_PDU` PDUs. The finished example does not contain them, and they rai

The Remaining Service Interfaces and Instances

Create `SVC_VehicleState` and `SVC_ComfortControl` the same way from the tables. Every version is `1.0` and every transport is UDP. Reduce the event g `SVC_ComfortControl` just rename its one group.

Service	C++ NAMESPACE	Events (type, EVENT ID)	Methods (METHOD ID, METHOD TYPE)	
`SVC_ParkDistance`	`pnc::park`	`DistanceUpdate` (uint8, 1), `SensorFault` (boolean, 2)	`SetSensitivity` (1, REQUEST AND RESPONSE)	`0` (4
`SVC_VehicleState`	`pnc::vehicle`	`GearPosition` (uint8, 1), `TrailerAttached` (boolean, 2)	none	`0` (4
`SVC_ComfortControl`	`pnc::comfort`	`DoorStatus` (uint8, 1)	`LockAll` (1, REQUEST AND RESPONSE), `FoldMirrors` (2, FIRE AND FORGET)	`0` (4

Instance	Role	Deployment	INSTANCE ID / REQUIRED INSTANCE ID
`ParkDistance_Provider`	Provider	`SVC_ParkDistance`	`1` (0x0001)
`ParkDistance_Consumer`	Consumer	`SVC_ParkDistance`	`1`
`VehicleState_Provider`	Provider	`SVC_VehicleState`	`1` (0x0001)
`VehicleState_Consumer`	Consumer	`SVC_VehicleState`	`1`
`ComfortControl_Provider_West`	Provider	`SVC_ComfortControl`	`1` (0x0001)
`ComfortControl_Provider_East`	Provider	`SVC_ComfortControl`	`2` (0x0002)
`ComfortControl_Consumer`	Consumer	`SVC_ComfortControl`	default `0xFFFF (ANY)`

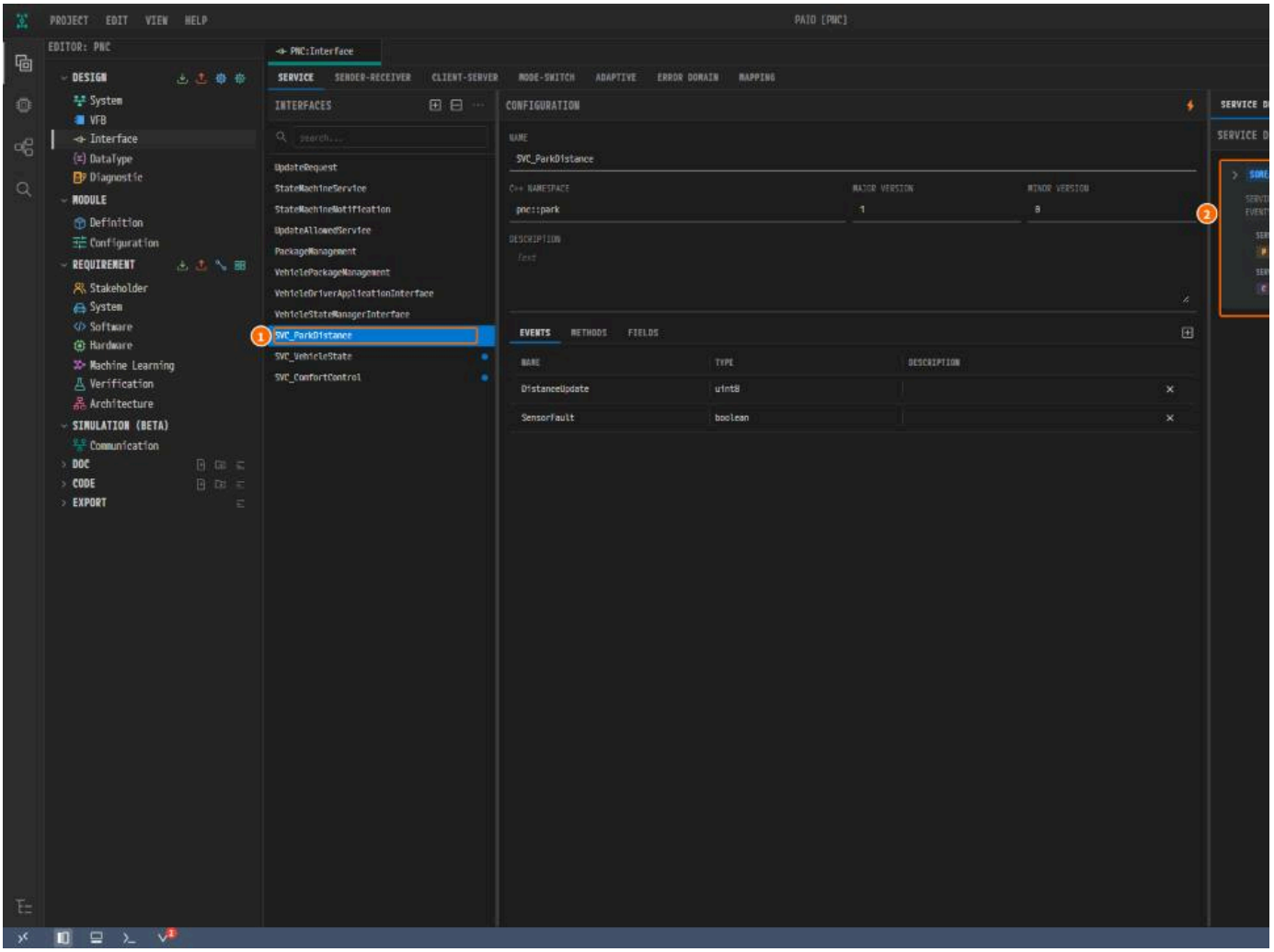


Figure 5-15. ① `SVC_ParkDistance`, ② its deployment summary. Services such as `UpdateRequest` come later, from Importing the PARA Platform and the F

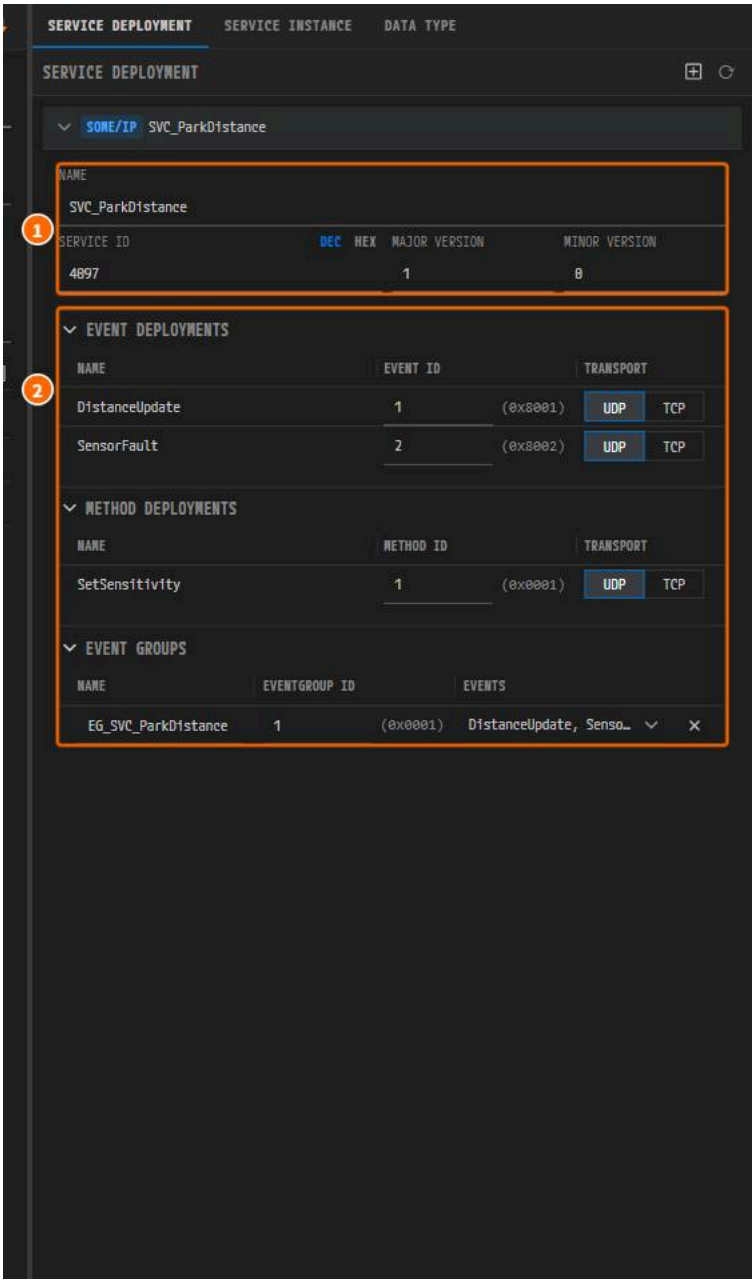


*Figure 5-16. ① **NAME**, **SERVICE ID** (in **DEC** mode, `4097`), version, ② the deployments and the event group `EG_SVC_ParkDistance`.*

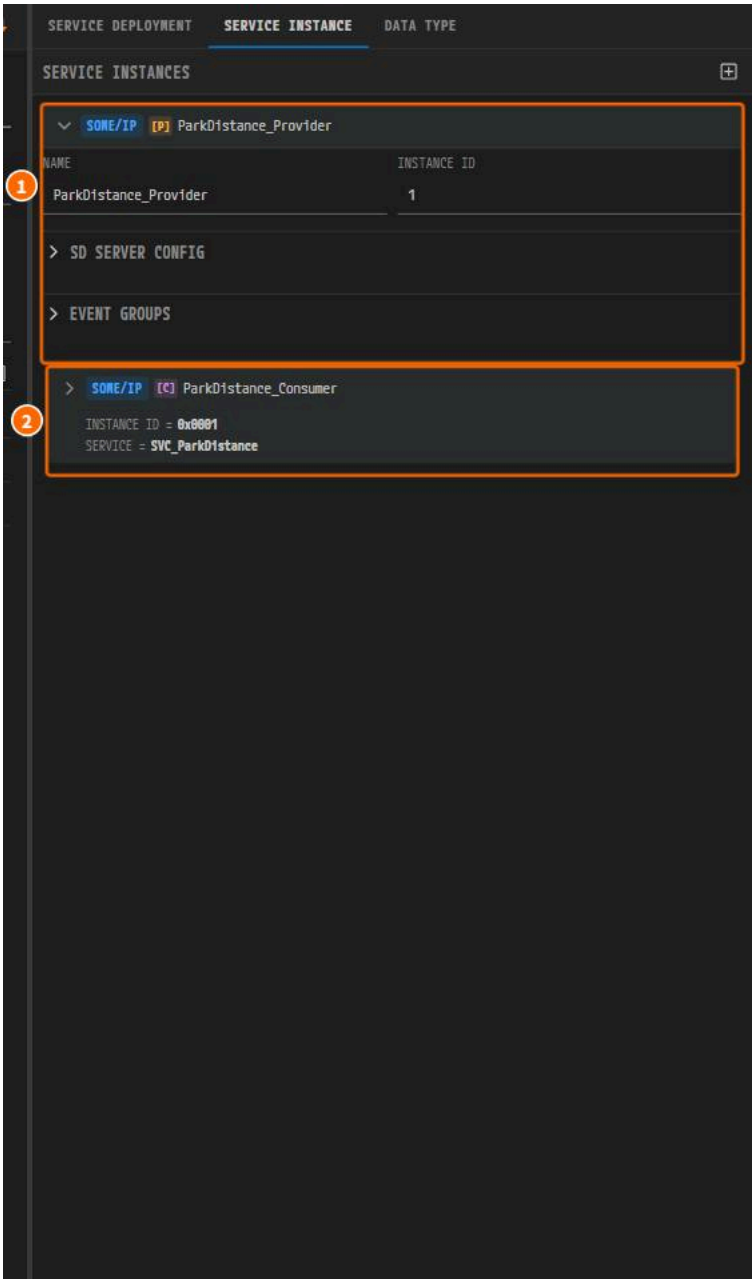


Figure 5-17. ① `ParkDistance_Provider`, ② `ParkDistance_Consumer`.

> **Note.** The finished example uses these SD settings (**SD SERVER CONFIG** on a provider, **SD CLIENT CONFIG** on a consumer), which differ from the defaults.

SD setting	Provider example value	Consumer example value
initial-delay min / max	0 / 0.01	0 / 0.01
repetition-base-delay	0.15	0.02
repetition-max	2	5
offer-cyclic-delay	3	—
ttl	10	10

Giving the Connectors IP Addresses and Application Endpoints

Follow `HPC_Eth_Rear_Conn` first.

1. In the **EDITOR** view, click **System** under **DESIGN**. The `PNC:System` tab opens.

2. Click the connector `HPC_Eth_Rear_Conn` on the lower edge of `HPC` (① in Figure 5-22).

3. Expand **NETWORK ENDPOINT** and leave **IPv4** selected.

4. Enter `192.168.200.1` in **UNICAST IPV4 ADDRESS** and `255.255.255.0` in **NETWORK MASK**. Select **FIXED** in **ADDRESS SOURCE**.



*Figure 5-18. ① **IPv4**, ② **UNICAST IPV4 ADDRESS**, ③ **ADDRESS SOURCE**, ④ **NETWORK MASK**.*

5. Click **+** in the **APPLICATION ENDPOINTS** header. A row (TCP) is added.

6. Expand the row, set **TYPE** to **UDP**, and enter `30501` in **PORT NUMBER**.

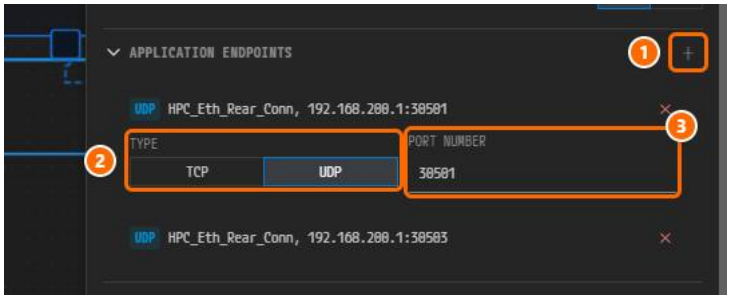


*Figure 5-19. ① **+**, ② **TYPE**, ③ **PORT NUMBER**.*

7. Add a second UDP endpoint, `30503`, the same way.

Fill in the other connectors the same way from the table. **NETWORK MASK** is always `255.255.255.0` and **ADDRESS SOURCE** is always **FIXED**.

Node / connector	UNICAST IPV4 ADDRESS	UDP application endpoint (PORT NUMBER)
`HPC` / `HPC_Eth_Body_Conn`	`192.168.100.1`	`30501`
`HPC` / `HPC_Eth_Rear_Conn`	`192.168.200.1`	`30501`, `30503`
`WestZone` / `WestZone_Eth_Conn`	`192.168.100.11`	`30502`
`EastZone` / `EastZone_Eth_Conn`	`192.168.100.13`	`30502`
`SouthZone` / `SouthZone_Eth_Conn`	`192.168.200.12`	`30502`, `30503`

> **Note.** PAIO names the endpoints itself, so the names may differ from the finished example.

Binding Service Instances to the Connectors

1. With `HPC_Eth_Rear_Conn` selected, click **+** in the **SERVICE INSTANCES** header. The **Select Service Instance** dialog opens.
2. Check `ParkDistance_Consumer` and click **Bind**.

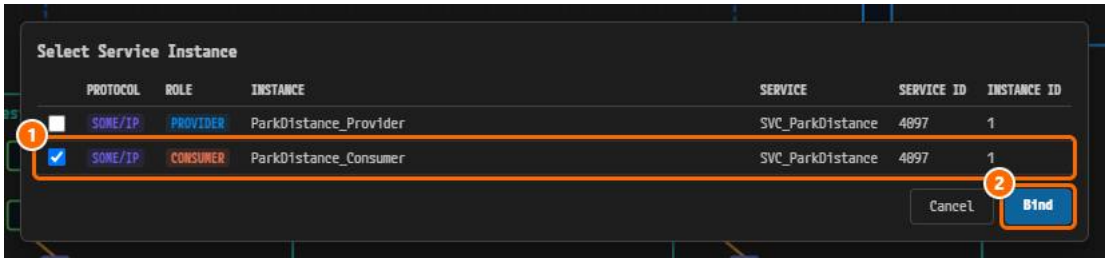


*Figure 5-20. ① `ParkDistance_Consumer`, ② **Bind**.*

3. Click the new card **SOME/IP C** `ParkDistance_Consumer` to expand it.
4. Click ▾ next to **UDP PORT**.
5. Select `30503`.



*Figure 5-21. ① **UDP PORT**, ② `30503`. `VehicleState_Provider` uses `30501` on this connector.*

6. Click elsewhere on the canvas, then click `HPC_Eth_Rear_Conn` again. The summary reads `UDP:30503`.

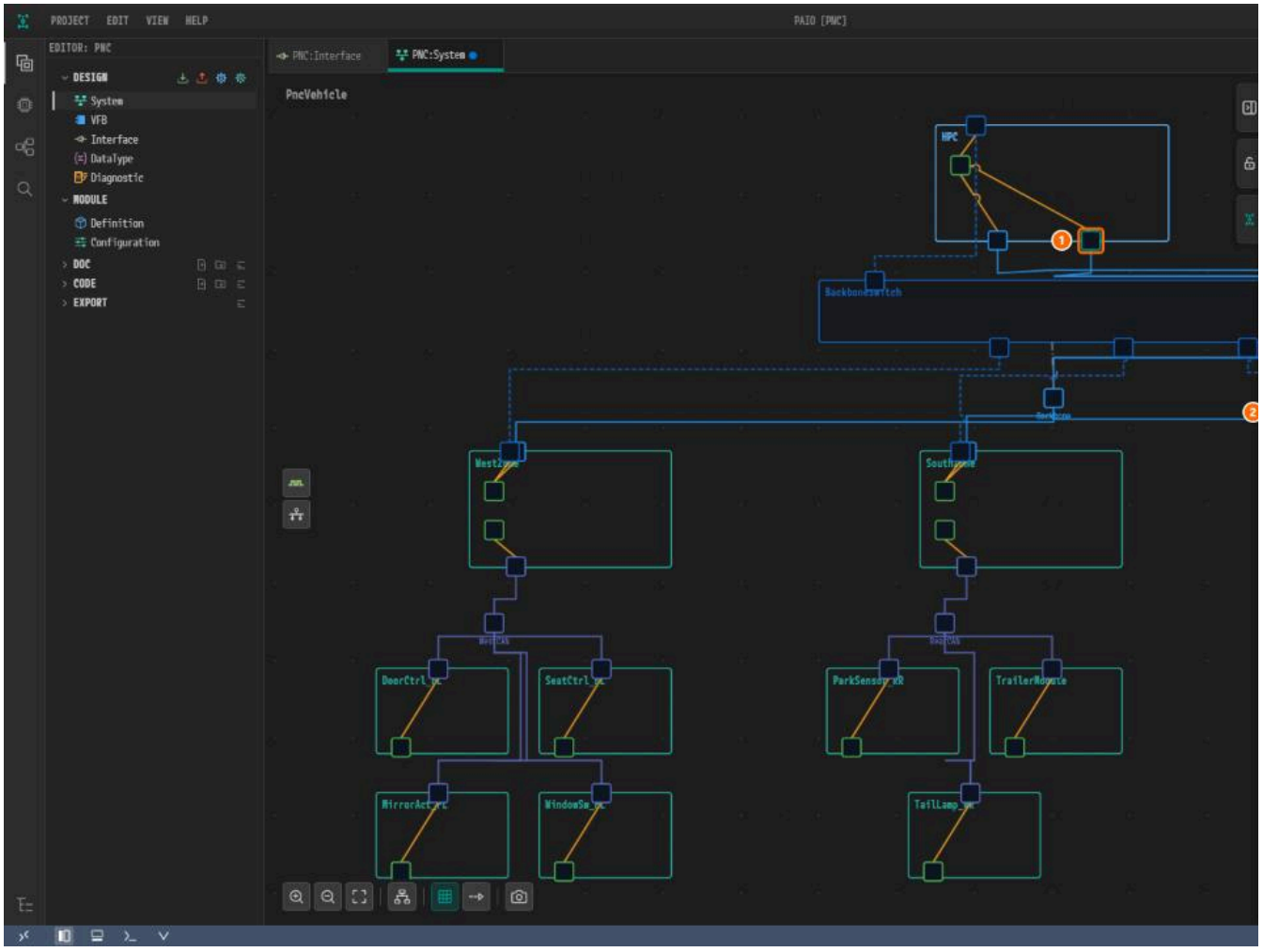


Figure 5-22. ① `HPC_Eth_Rear_Conn`, ② `ParkDistance_Consumer` `UDP:30503`.

7. Save with `Ctrl+S`.

> **Note.** Until the UDP port is set, validation reports one error for the instance. It clears in step 5.

The Remaining Bindings

Bind all seven instances the same way from the table.

Node / connector	Instances to bind (role, UDP PORT)
`HPC` / `HPC_Eth_Body_Conn`	`ComfortControl_Consumer` (C, 30501)
`HPC` / `HPC_Eth_Rear_Conn`	`VehicleState_Provider` (P, 30501), `ParkDistance_Consumer` (C, **30503**)
`WestZone` / `WestZone_Eth_Conn`	`ComfortControl_Provider_West` (P, 30502)
`EastZone` / `EastZone_Eth_Conn`	`ComfortControl_Provider_East` (P, 30502)
`SouthZone` / `SouthZone_Eth_Conn`	`ParkDistance_Provider` (P, 30502), `VehicleState_Consumer` (C, **30503**)

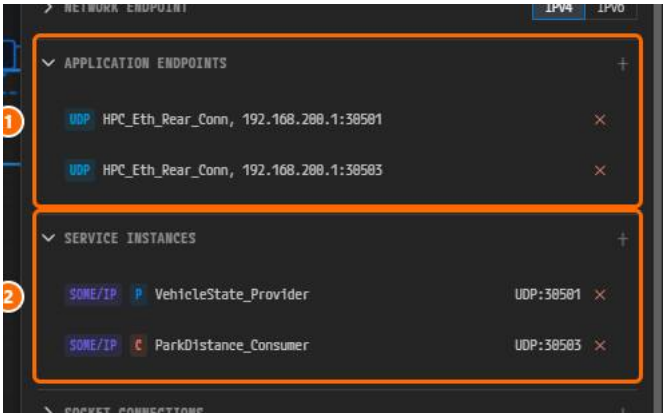


Figure 5-23. ① The two UDP endpoints, ② the two bound instances.

> **Caution.** Two instances on the same IP address and UDP port raise the validation error `paio_someip_address_port_unique`.

Why Is It Designed This Way?

- **PNC and services.** A PNC references service instances to record which services need that PNC awake; when `PNC_ParkAssist` sleeps, `SVC_ParkDista`
- **Binding.** Only a bound instance exports the `SomeipServiceInstanceToMachineMapping` that ties it to a machine.
- **Separate ports.** Instances that share a connector need different UDP ports.

Checklist

- [ ] The **SENDER-RECEIVER** tab holds 11 `SRI_*` interfaces.
- [ ] The **SERVICE** tab holds `SVC_ParkDistance`, `SVC_VehicleState`, and `SVC_ComfortControl` with Service IDs `0x1001`, `0x1002`, `0x1003` and version `1.0`
- [ ] There are four providers and three consumers named as in the table.
- [ ] `HPC_Eth_Rear_Conn` shows `VehicleState_Provider` `UDP:30501` and `ParkDistance_Consumer` `UDP:30503`.
- [ ] Neither editor tab shows the ● marker.

Next Step

In VFB and VFC (PortGroup) you group the SWC ports of each function into PortGroups to form VFCs.

VFB and Virtual Function Clusters

What You Will Do

In this chapter you build one composition per node in the VFB editor, with `ParkSensor_RR` and its atomic SWC `ParkDistanceSensing` as the worked exa Clusters), which AUTOSAR expresses as PortGroups, and check them with the PNC Overlay.

Before You Begin

- Finish Interfaces and SOME/IP Services. The ports use `SRI_ParkDistance` from Chapter 5.
- The VFB root `PncVehicle` is empty. You create **14** compositions: `ParkSensor_RR` step by step, the other 13 from the table in The Remaining 13 Comp
- `ParkSensor_RR` connects to `SouthZone`, so **create the 13 compositions in the table first**. The figures were captured that way.
- The node `ParkSensor_RR [ECU]` exists in the System editor (from Topology Design).

Item	Value
Composition (**NAME** and **AS PROTOTYPE**)	`ParkSensor_RR`
Atomic SWC (**NAME** and **AS PROTOTYPE**)	`ParkDistanceSensing`
SWC port and boundary port	`ParkDistance`, **PPort**, `SenderReceiver: SRI_ParkDistance`
Assembly target	`ParkDistance` (R) of `SouthZone`
ECU mapping	`ParkSensor_RR [ECU]`
PortGroup on `ParkDistanceSensing`	`VFC_ParkAssist`, **PARTIAL_NETWORKING**, OUTER PORTS `ParkDistance`

Creating the SWC and the Composition

Creating the Composition

1. In the **EDITOR** view, click **VFB** under **DESIGN**. The **PNC:VFB** tab opens.
2. Click **Auto Layout** (the fourth button on the toolbar under the canvas). Click **Fit to View** if needed.
3. Right-click the empty canvas near `SouthZone` and click **Add Composition**.

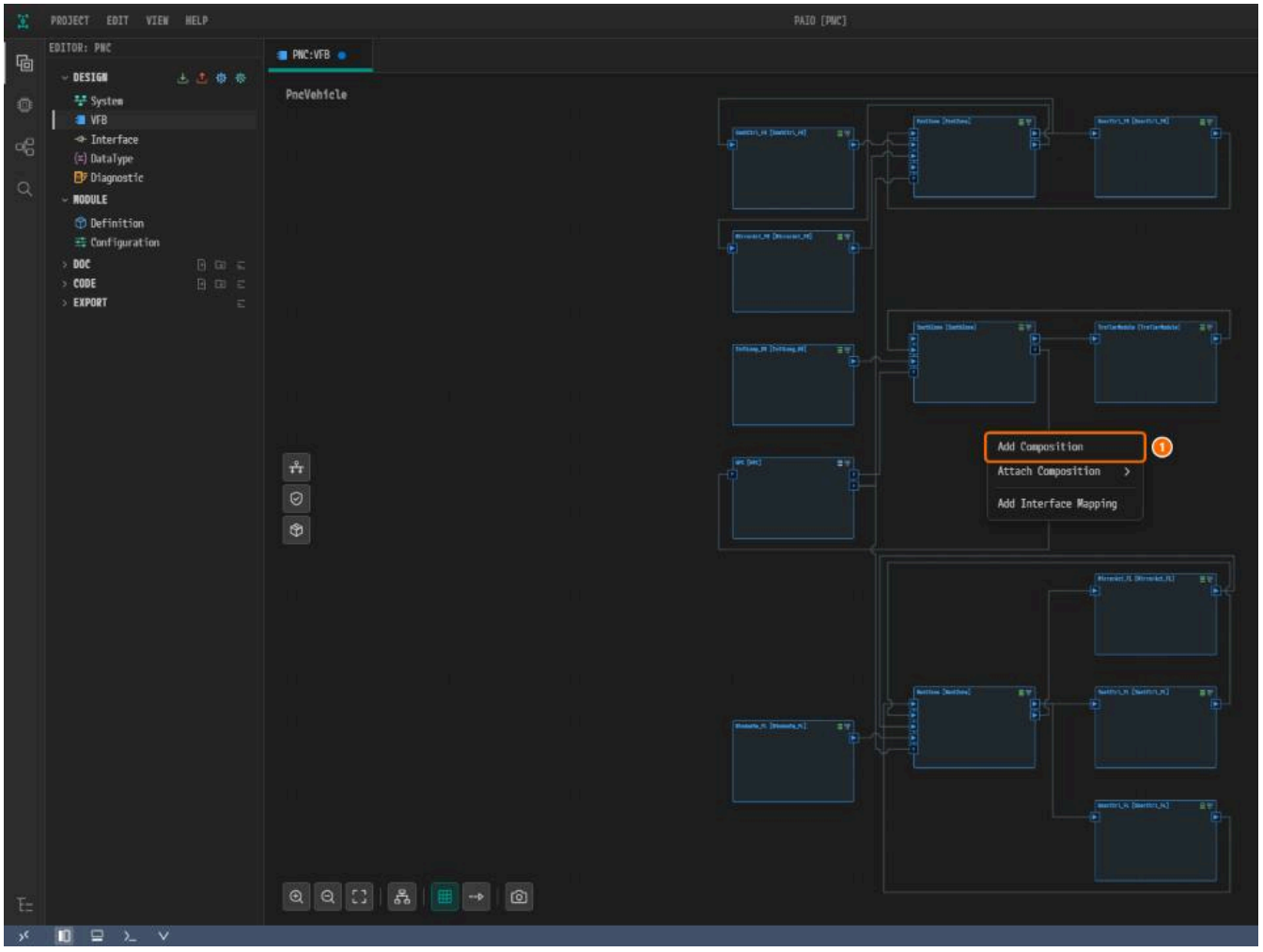


*Figure 6-1. ① **Add Composition.***

4. Click the new node `Composition_0`.

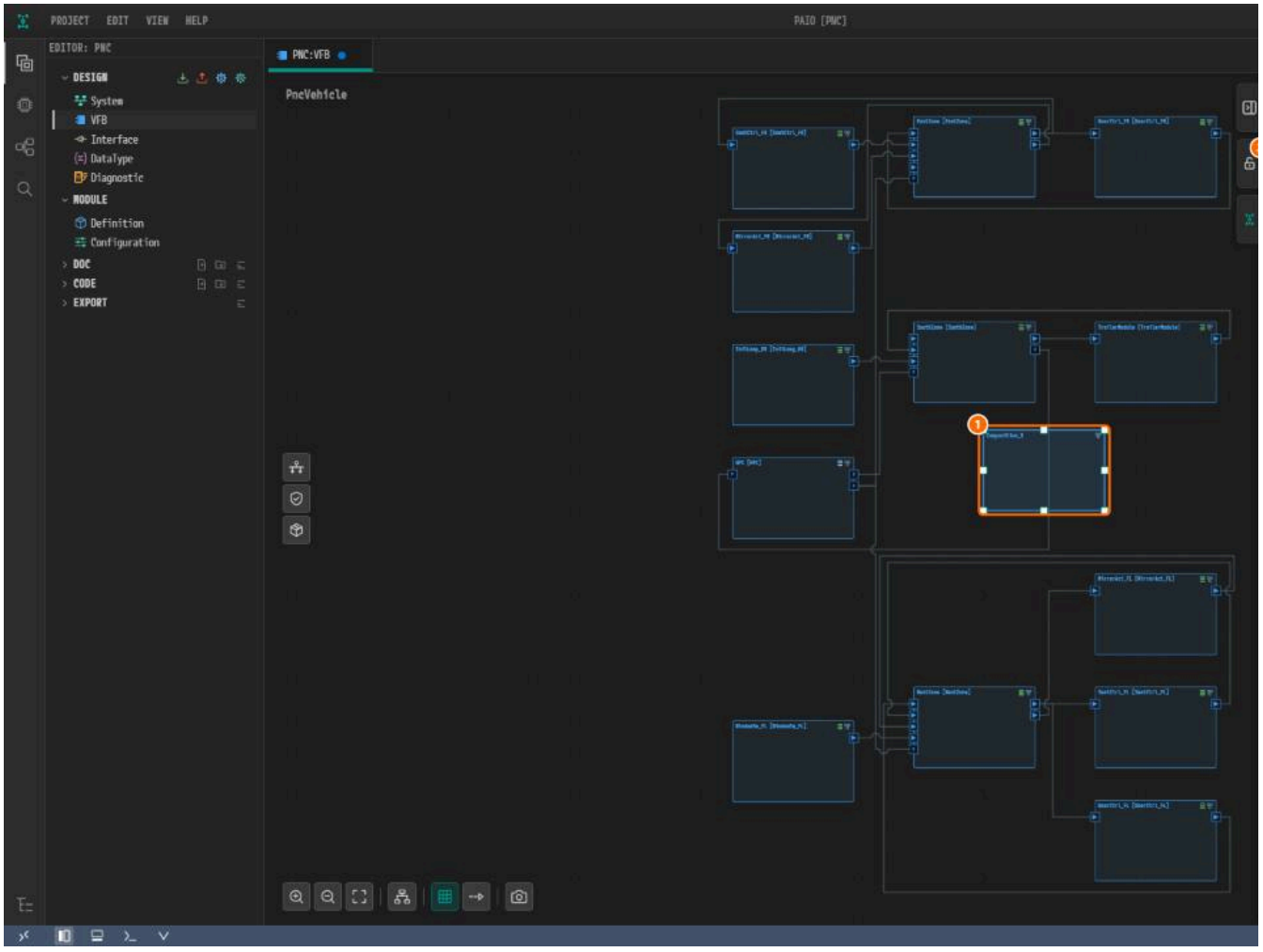


*Figure 6-2. ① The node, ② **NAME**, ③ **AS PROTOTYPE**.*

5. Enter `ParkSensor_RR` in **NAME** and in **AS PROTOTYPE**, pressing **Tab** after each.

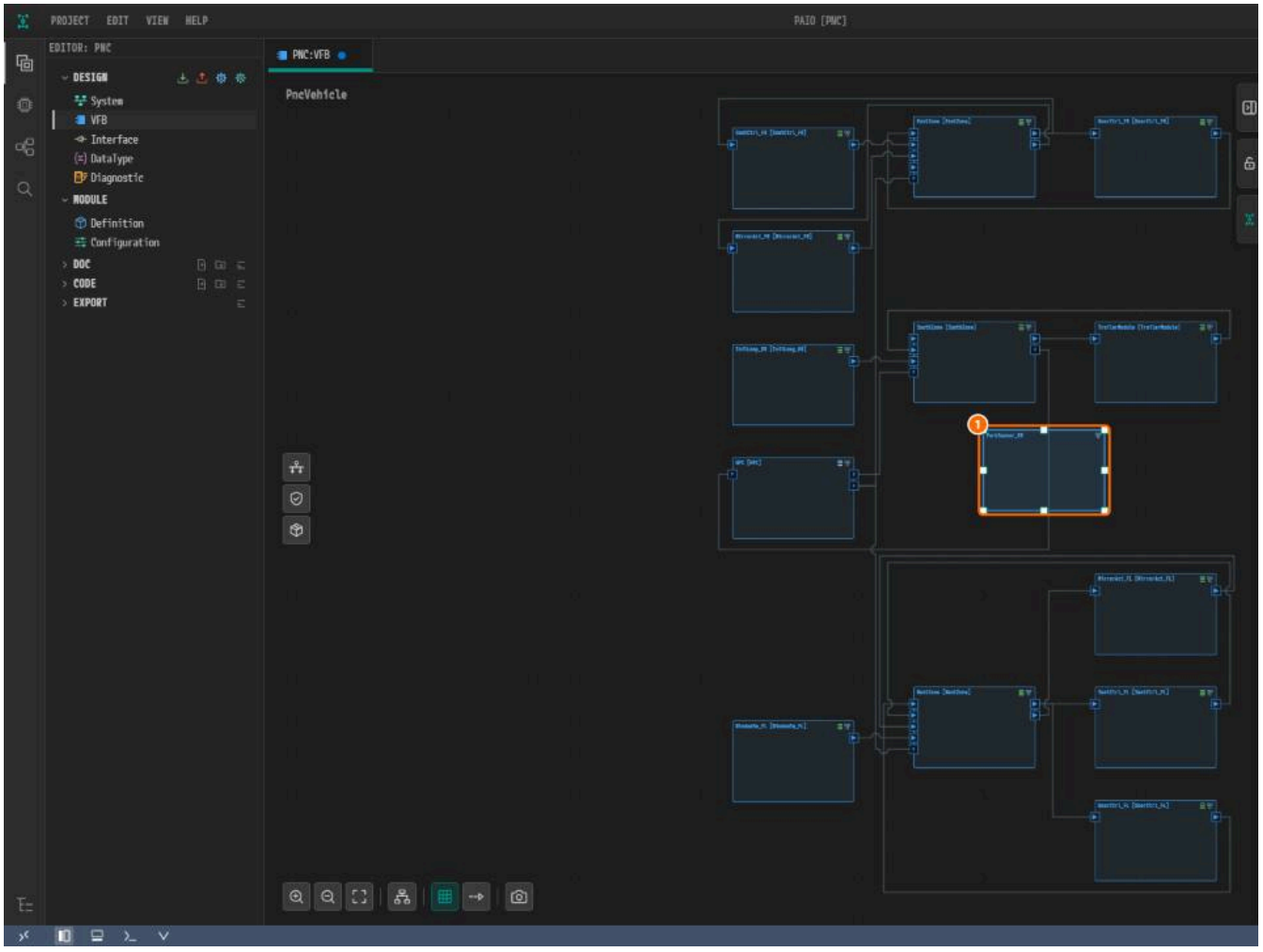


Figure 6-3. ① The renamed node.

> **Caution. AS PROTOTYPE** does not follow **NAME**. Edit both: the SWC-ECU mapping uses **AS PROTOTYPE** as its key.

Creating the Atomic SWC

1. Double-click `ParkSensor_RR`. The path **System > ParkSensor_RR** and an empty frame appear. Click **System** in the path to return to the root later.
2. Right-click inside the frame. Hover over **Add SWC** and click **Application**.

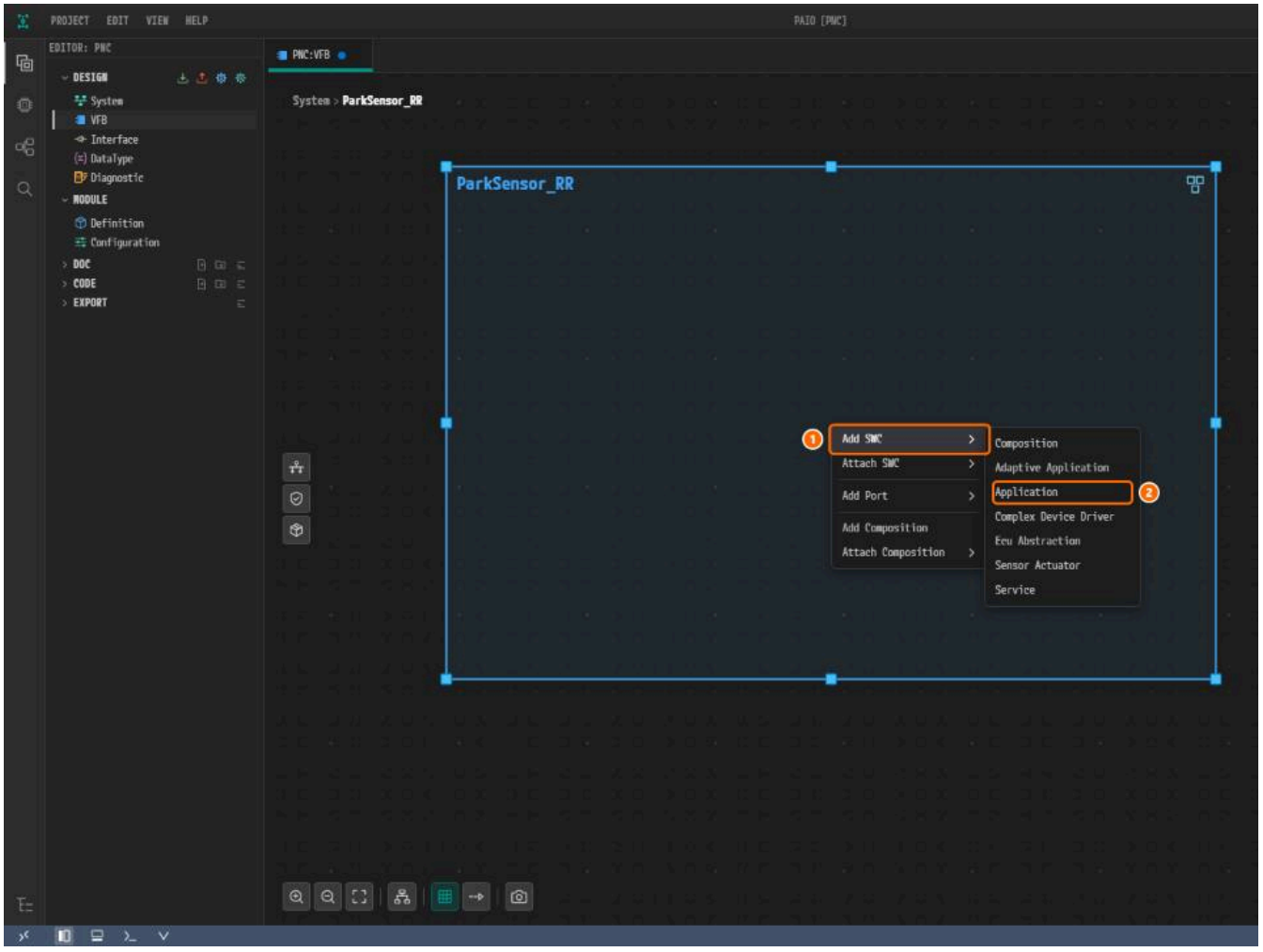


*Figure 6-4. ① **Add SWC**, ② **Application**.*

3. Click the new node `Application_0`.

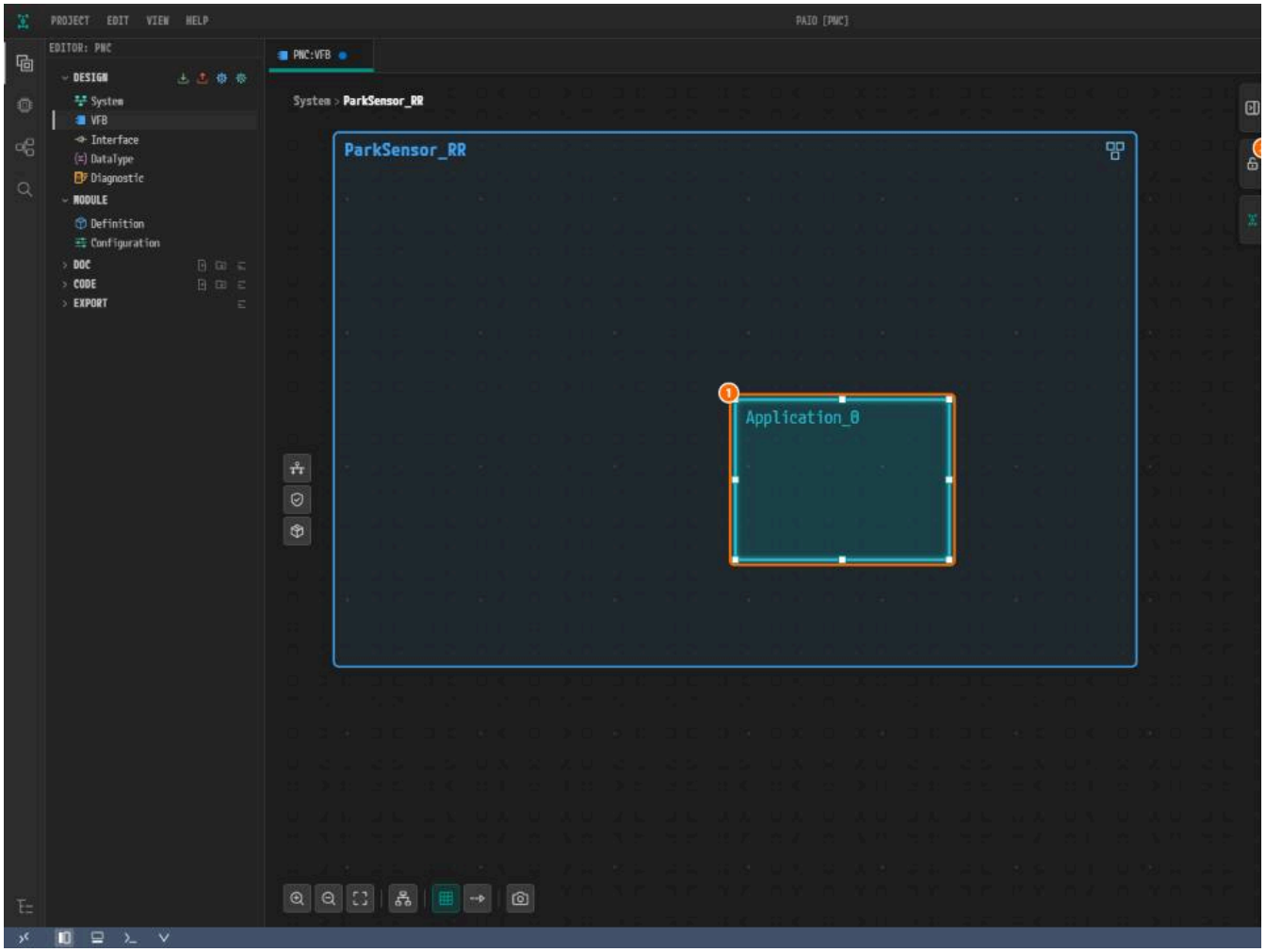


*Figure 6-5. ① The node, ② **NAME**, ③ **AS PROTOTYPE** (a different default).*

4. Enter `ParkDistanceSensing` in **NAME** and in **AS PROTOTYPE**, pressing **Tab** after each.

Creating the Ports

1. Right-click near the right edge of `ParkDistanceSensing`. Hover over **Add Port** and click **PPort**.

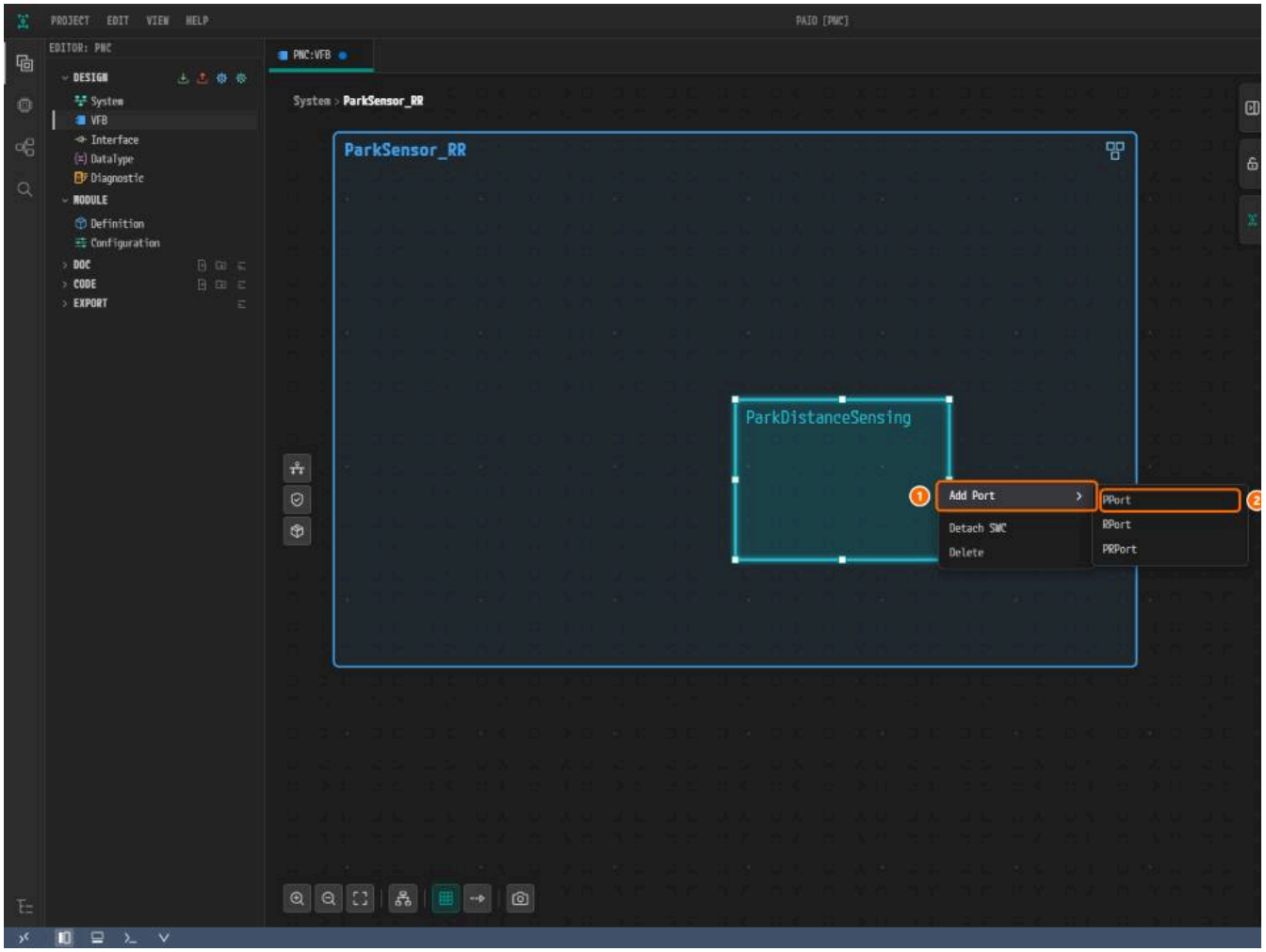


*Figure 6-6. ① **Add Port**, ② **PPort**. The port lands on the edge nearest the click.*

2. Click the new port. Enter `ParkDistance` in **NAME** and select `SenderReceiver: SRI_ParkDistance` in **PORT INTERFACE**.

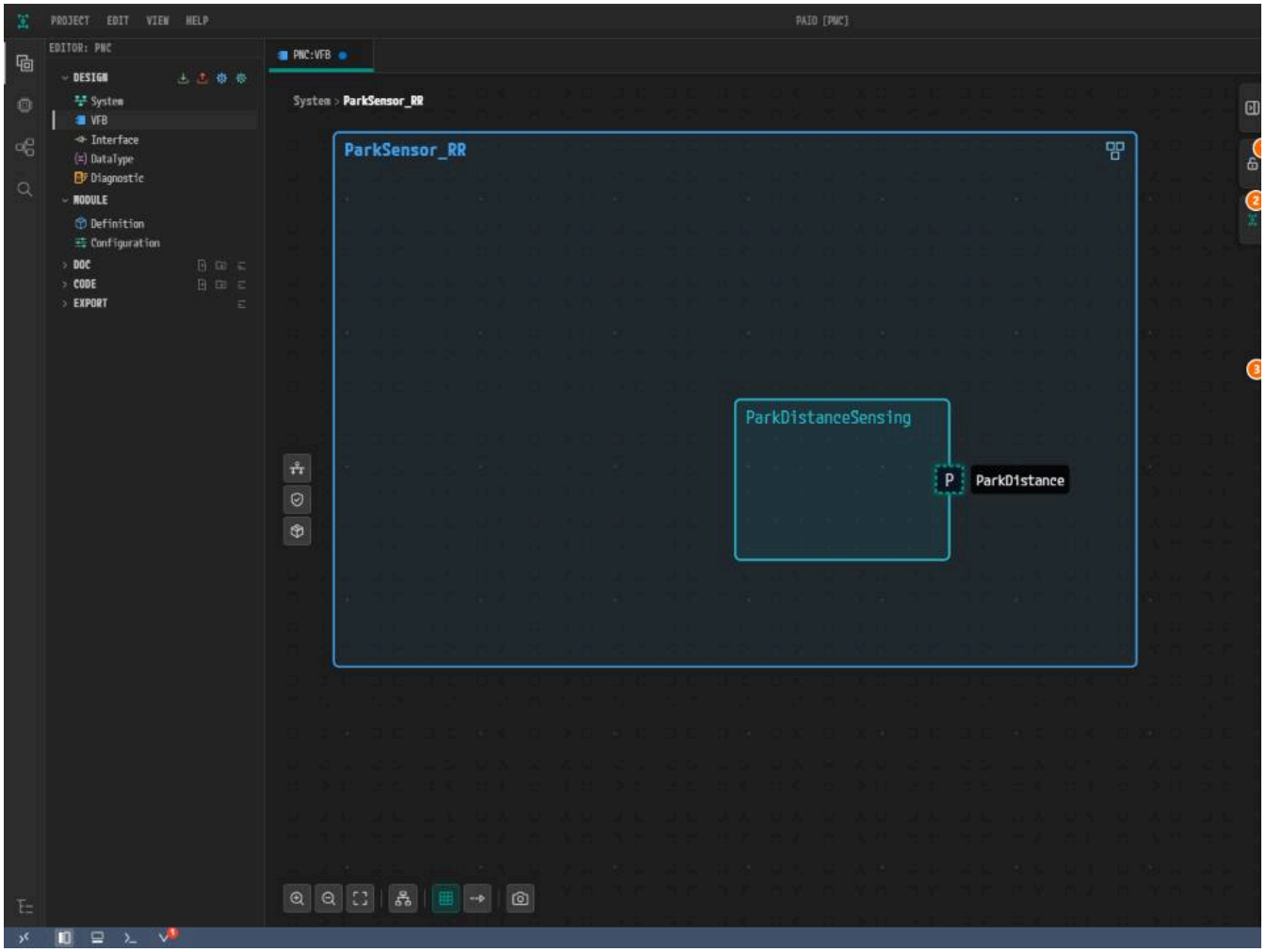


*Figure 6-7. ① **NAME**, ② **PORT INTERFACE**, ③ `SenderReceiver: SRI_ParkDistance`.*

> **Note.** If clicking the port selects the node, click the empty canvas first, then the port. Zoom with **Ctrl**+wheel if needed.

3. Right-click the right edge of the frame and add a **PPort** with the same **NAME** and **PORT INTERFACE**.

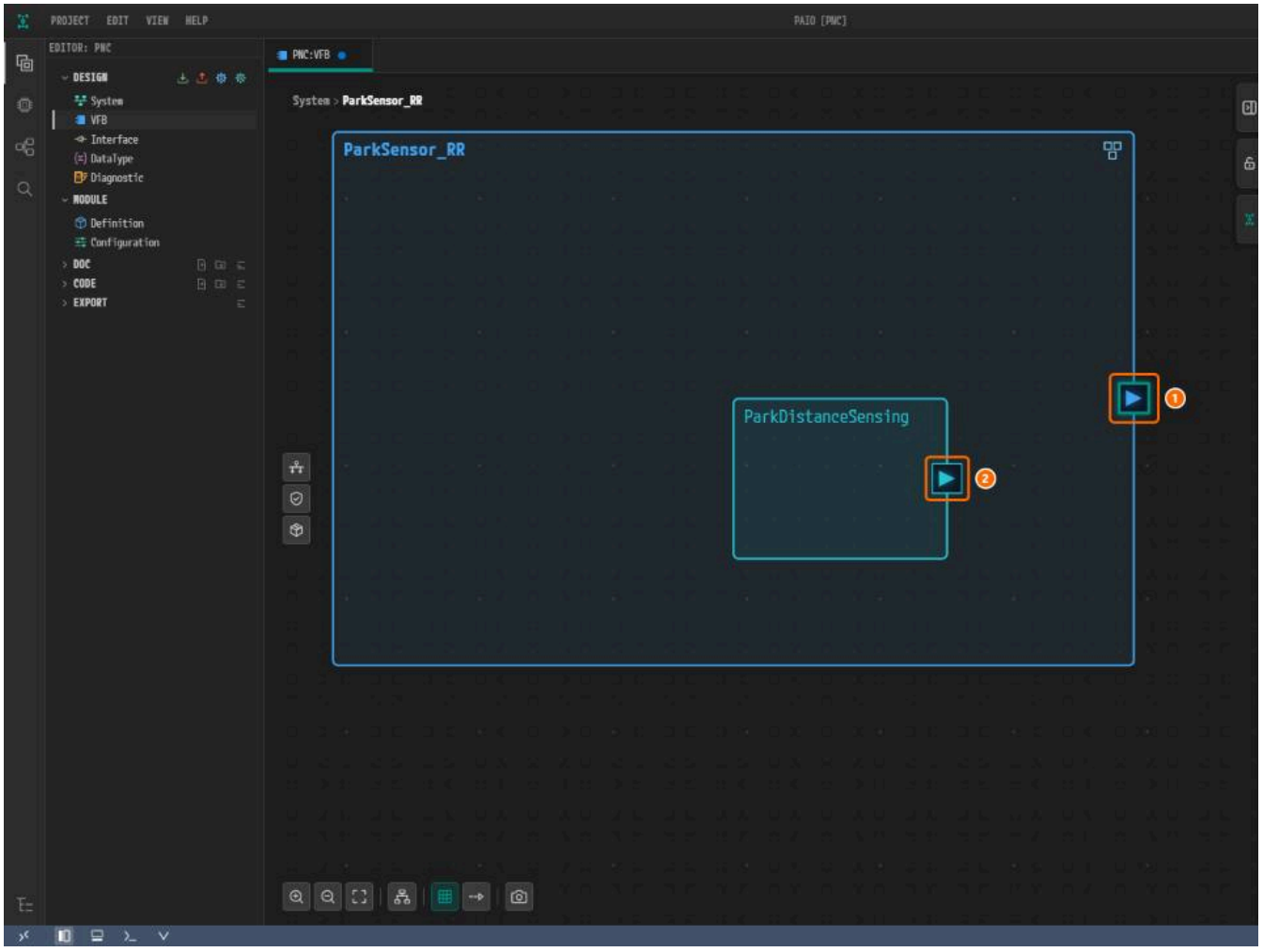


Figure 6-8. ① The boundary port, ② the SWC port.

Connecting with a Delegation Connector

1. Right-click a port: **Connect to** offers no candidates here.

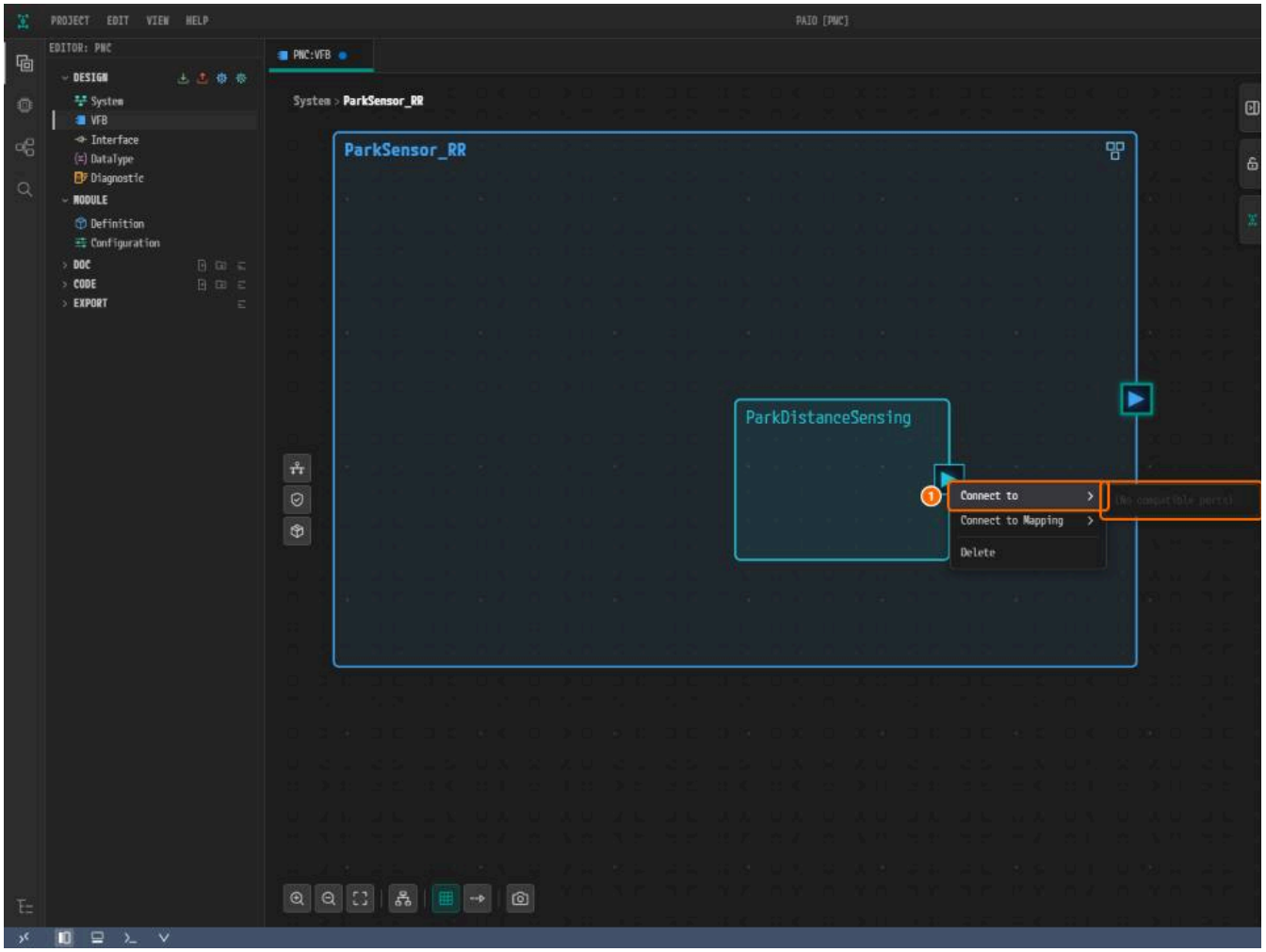


*Figure 6-9. ① **Connect to**, ② `(No compatible ports)`.*

2. Hold **Shift** and drag the SWC port onto the boundary port. A line appears.

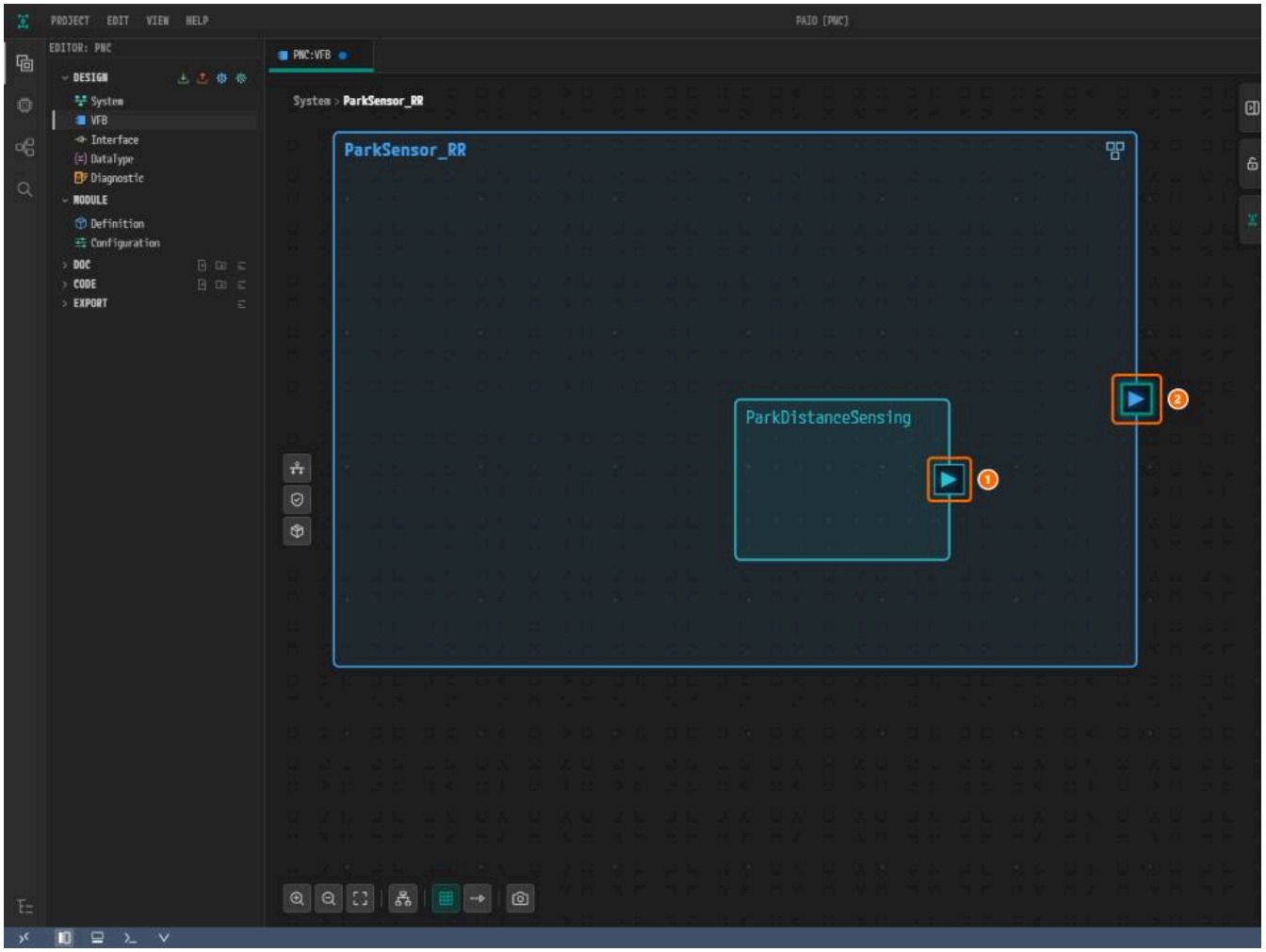


Figure 6-10. ① Drag from the SWC port, ② drop on the boundary port.

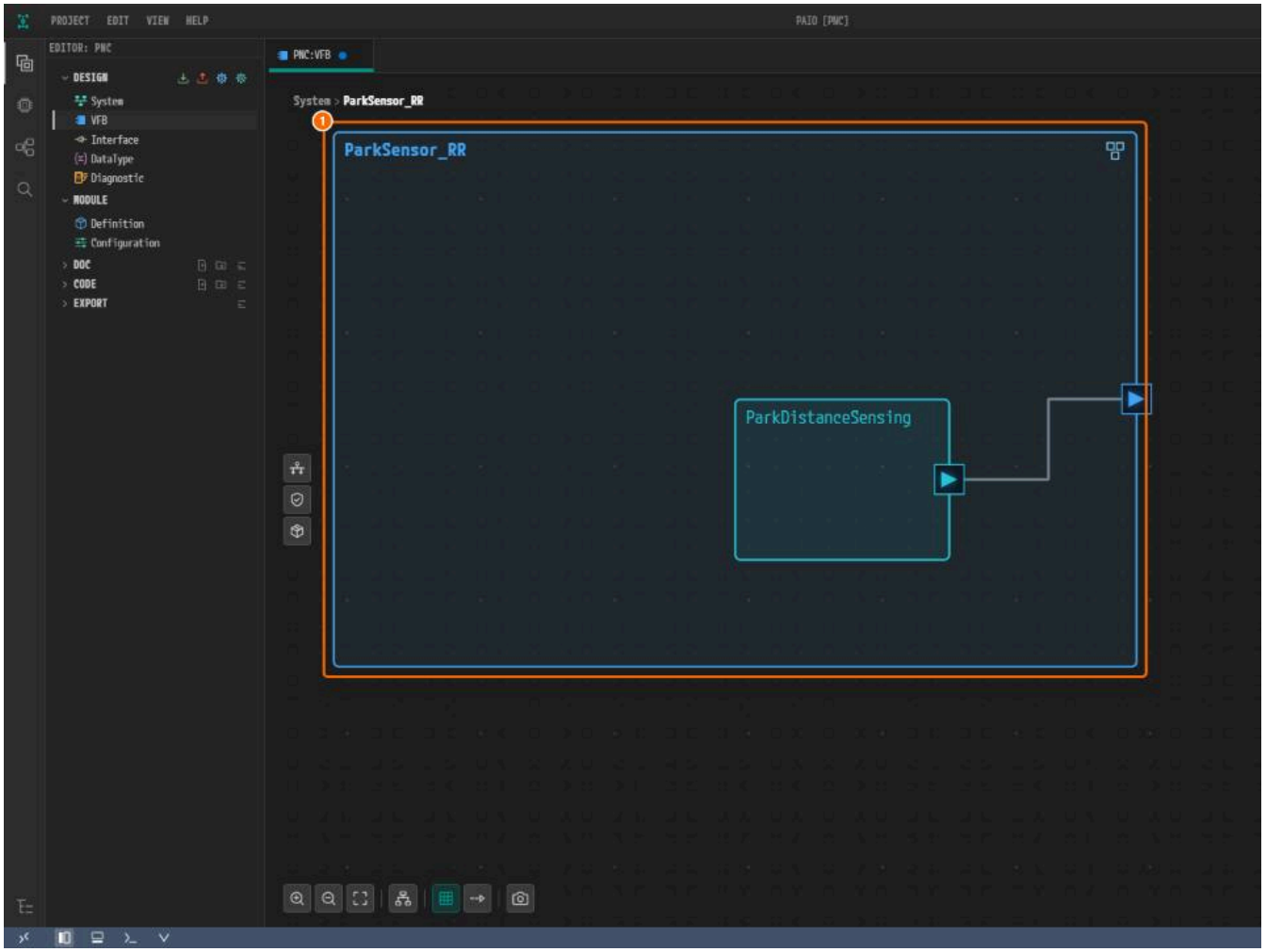


Figure 6-11. ① `ParkSensor_RR` with the delegation connector.

Connecting with an Assembly Connector

1. Click **System** in the path to return to the root.

2. **Connect to** is empty here as well.

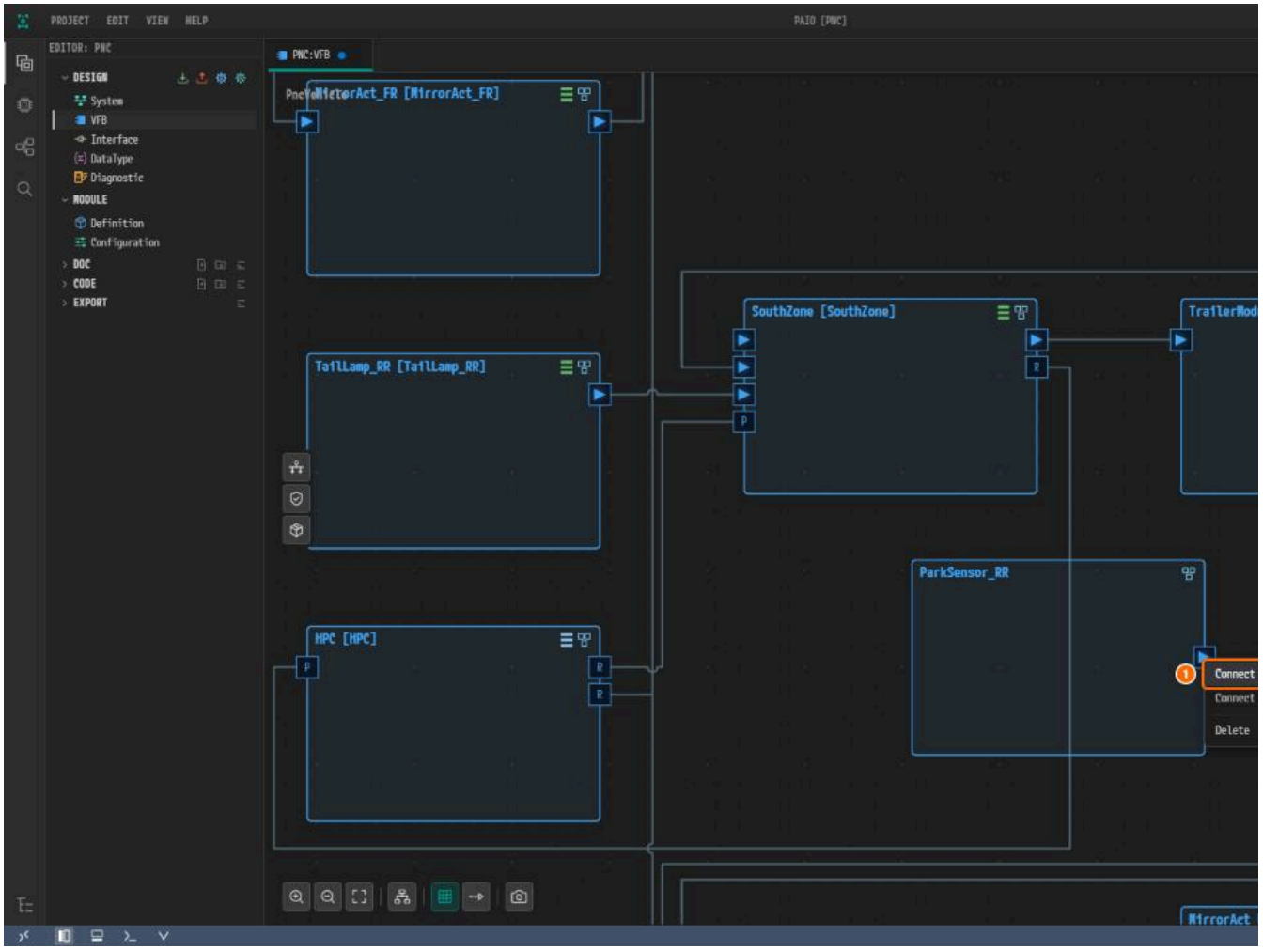


*Figure 6-12. ① **Connect to**, ② `(No compatible ports)`.*

3. Hold **Shift** and drag the port of `ParkSensor_RR` onto the `ParkDistance` (R) port of `SouthZone`.

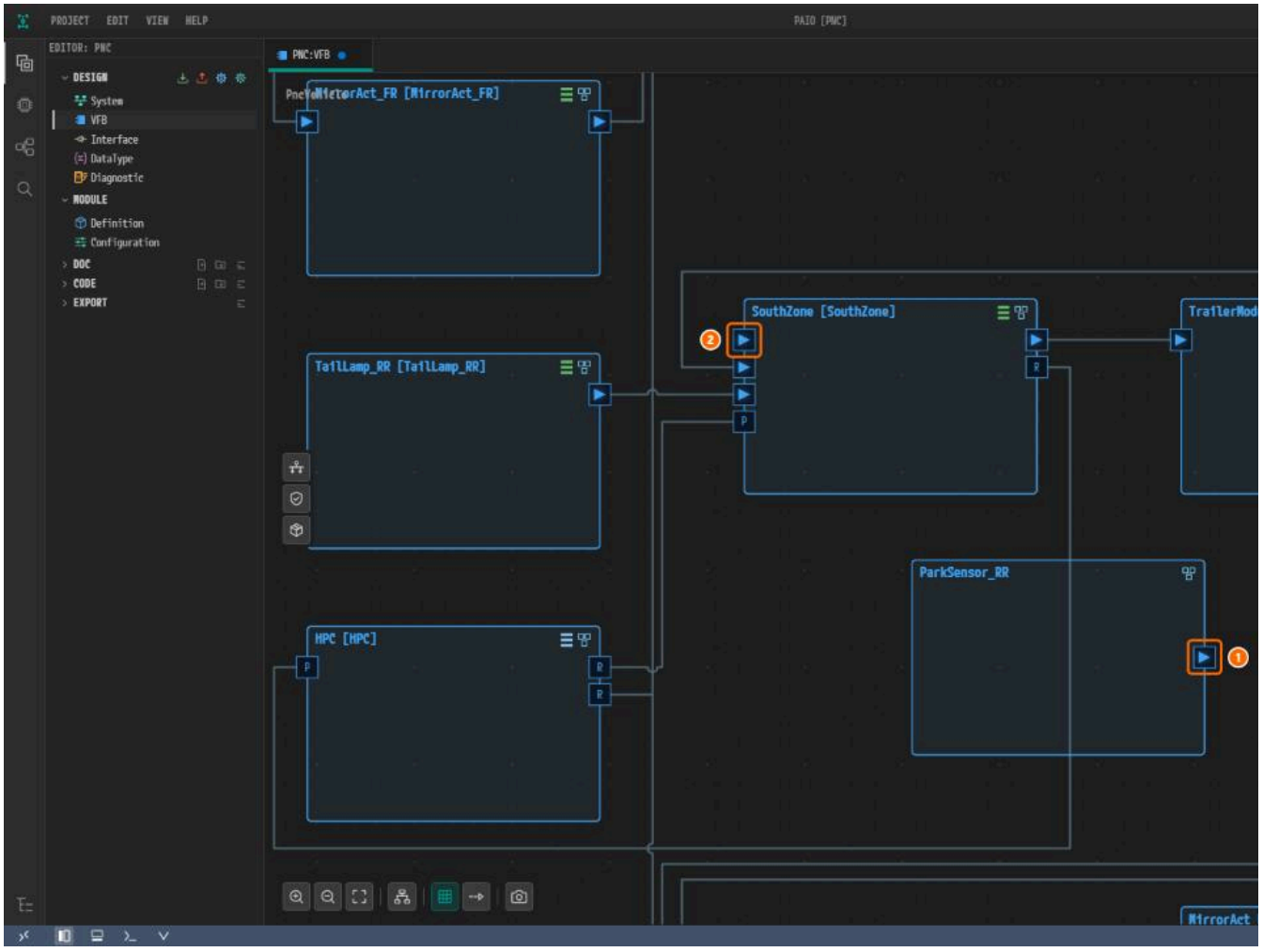


*Figure 6-13. ① `ParkDistance` of `ParkSensor_RR`, ② `ParkDistance` of `SouthZone`. Click a port to check its **NAME** if unsure.*

Mapping to an ECU and Saving

1. Right-click `ParkSensor_RR`. Hover over **ECU/HPC Mapping** and click `ParkSensor_RR [ECU]`.

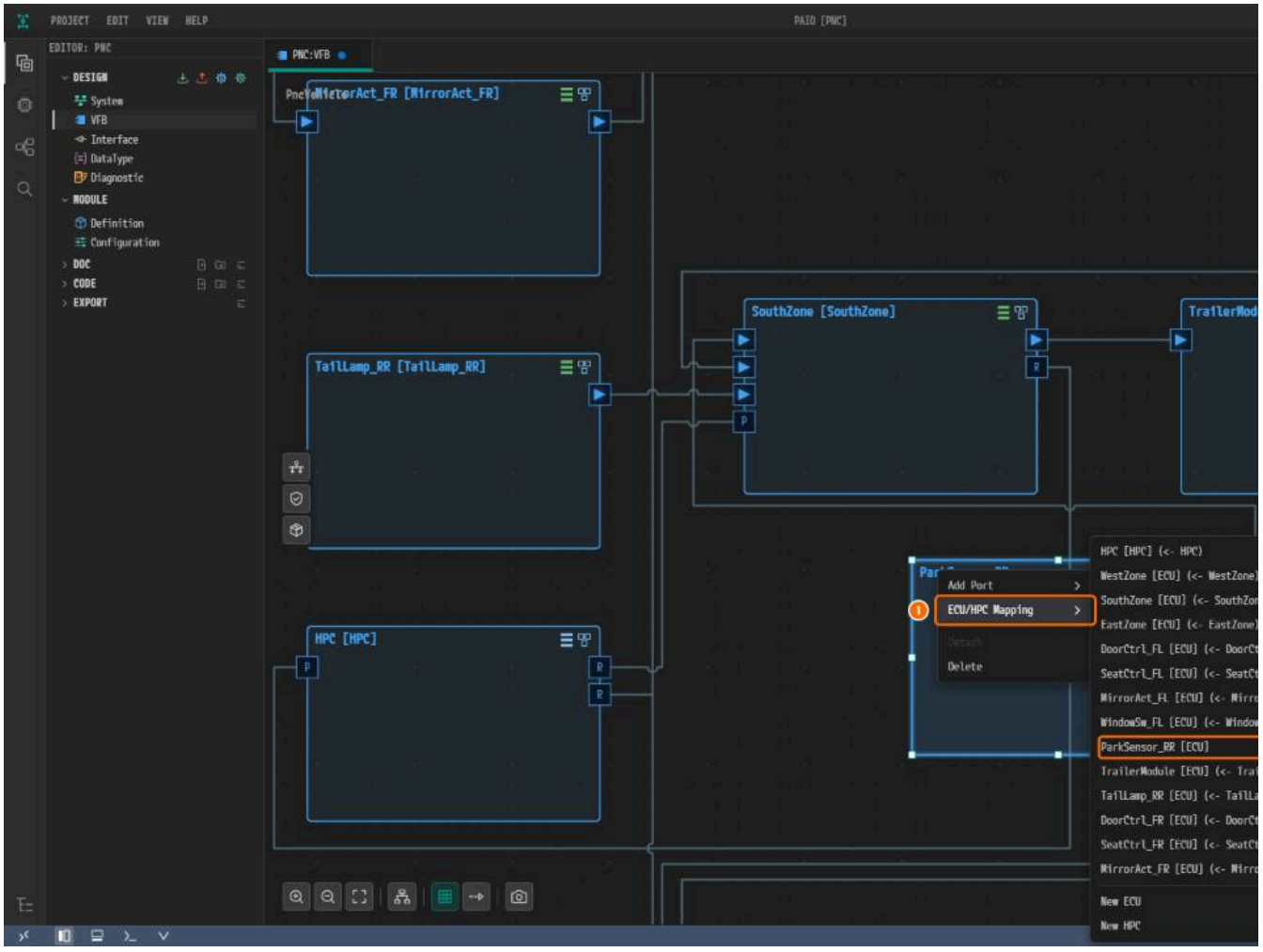


*Figure 6-14. ① **ECU/HPC Mapping**, ② `ParkSensor_RR [ECU]`.*

> **Note.** The node header shows the mapping (`ParkSensor_RR [ParkSensor_RR]` with an ECU icon) only after you reopen the editor.

2. Save with **Ctrl+S**.

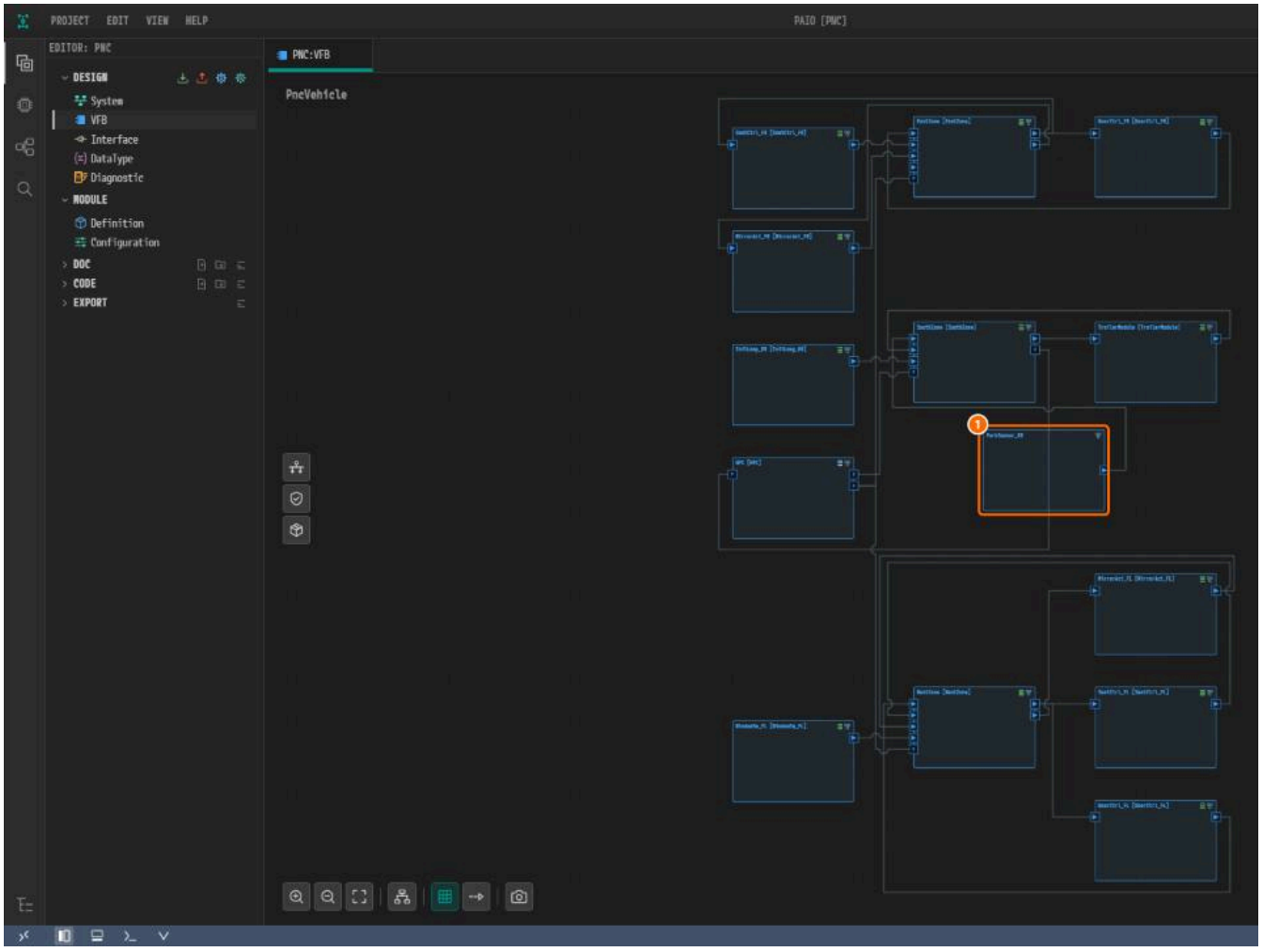


Figure 6-15. ① `ParkSensor_RR`, connected to `SouthZone`. The root holds 14 compositions.

The Remaining 13 Compositions

Create the others the same way from the table. Name each composition after its node, delegate each boundary port 1:1 to the internal SWC port of the s
is `prototype = type`; `(P)`/`(R)` is the port direction, followed by the PortInterface.

Composition (= mapped node)	Node kind	Internal SWC	Composition boundary p
`HPC`	HPC	`ParkAssist` = `ParkAssistApp`, `VehicleState` = `VehicleStateApp`, `Comfort` = `ComfortApp`	`ParkDistanceSvc` (R) `SVC_ParkDistance`, `VehicleStateSvc` (P) `SVC_Vehicl`
`WestZone`	ECU	`Gateway` = `ZoneGateway`	`ComfortCmd` (P) `SRI_ComfortCmd`, `MirrorCmd` (P) `SRI_MirrorCmd`, `DoorStatu` `SRI_SeatStatus`, `MirrorStatus` (R) `SRI_MirrorStatus`, `WindowSwitch` (R) `SVC_ComfortControl`
`EastZone`	ECU	`Gateway` = `ZoneGateway`	Same as `WestZone`
`SouthZone`	ECU	`Gateway` = `RearGateway`	`ParkDistance` (R) `SRI_ParkDistance`, `TrailerStatus` (R) `SRI_TrailerStat` `SRI_TailLamp`, `ParkDistanceSvc` (P) `SVC_ParkDistance`, `VehicleStateSvc`
`DoorCtrl_FL`, `DoorCtrl_FR`	ECU	`DoorControl` = `DoorControl`	`DoorStatus` (P) `SRI_DoorStatus`, `ComfortCmd` (R) `SRI_ComfortCmd`
`SeatCtrl_FL`, `SeatCtrl_FR`	ECU	`SeatControl` = `SeatControl`	`SeatStatus` (P) `SRI_SeatStatus`, `ComfortCmd` (R) `SRI_ComfortCmd`
`MirrorAct_FL`, `MirrorAct_FR`	ECU	`MirrorControl` = `MirrorControl`	`MirrorStatus` (P) `SRI_MirrorStatus`, `MirrorCmd` (R) `SRI_MirrorCmd`
`WindowSw_FL`	ECU	`WindowSwitch` = `WindowSwitch`	`WindowSwitch` (P) `SRI_WindowSwitch`
`ParkSensor_RR`	ECU	`ParkDistanceSensing` = `ParkDistanceSensing`	`ParkDistance` (P) `SRI_ParkDistance` (created in this chapter)
`TrailerModule`	ECU	`TrailerControl` = `TrailerControl`	`TrailerStatus` (P) `SRI_TrailerStatus`, `TrailerCmd` (R) `SRI_TrailerCmd`
`TailLamp_RR`	ECU	`TailLampControl` = `TailLampControl`	`TailLamp` (P) `SRI_TailLamp`

- The three SWCs of `HPC` are created with **Add SWC > Adaptive Application**. The `PARA` prototype in `HPC` is added later by PARA Platform Import and PN
- The root has 21 assembly connectors: end-ECU status ports (P) to a Zone (R), Zone command ports (P) to the end ECUs (R), and Zone service ports t `VehicleStateSvc` of `VehicleState` and `ParkAssist`.
- Internal behavior (runnables, RTE events) is not covered; see VFB Editor.

Reviewing the VFB Root Structure

1. With all 14 compositions in place, click **Auto Layout**, then **Fit to View**.

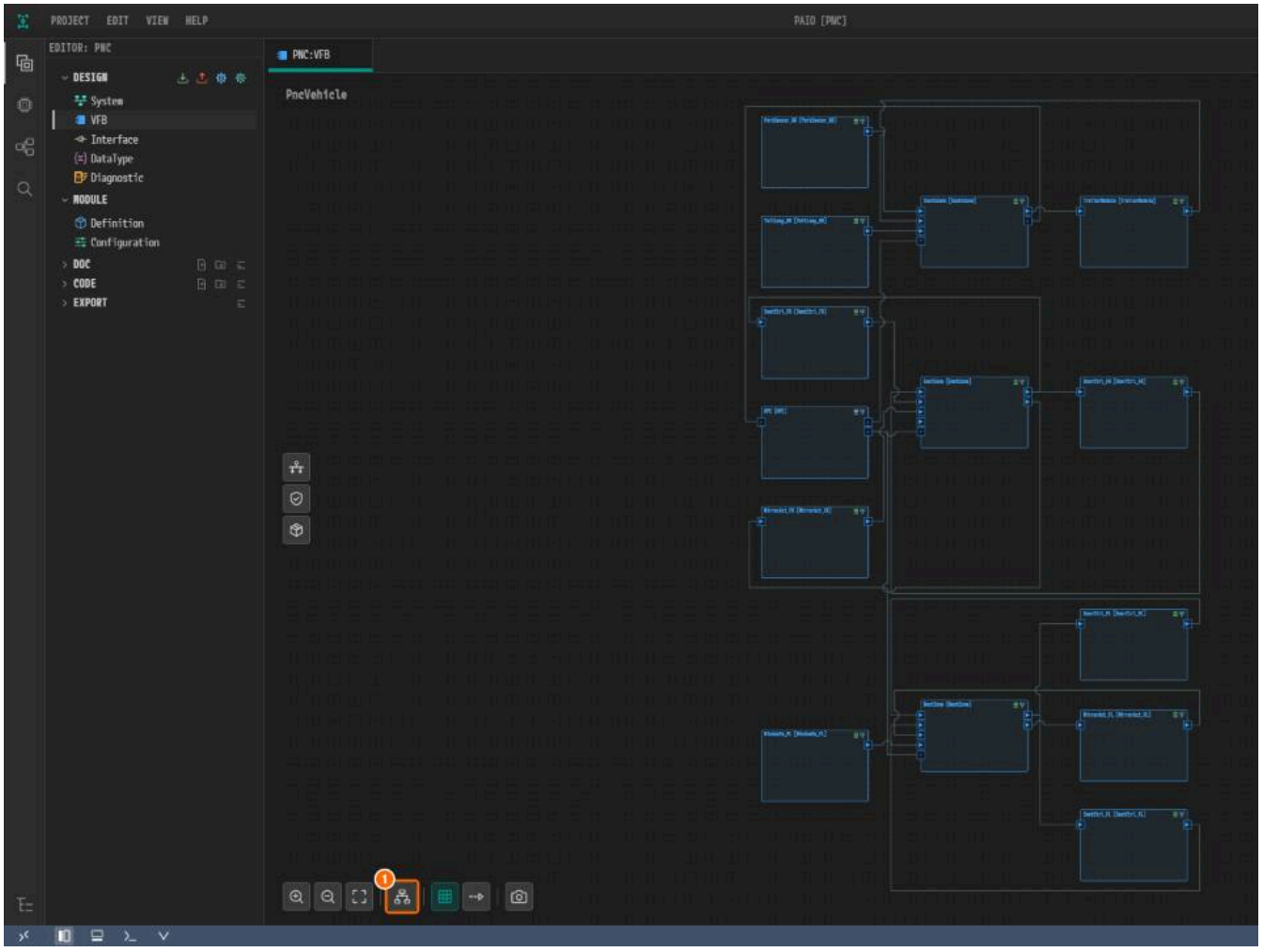


*Figure 6-16. ① **Auto Layout**. `ParkSensor_RR` moves to the top left.*

Composition (= mapped node)	Role
`HPC`	HPC (AP). Hosts the application AAs
`WestZone`, `EastZone`, `SouthZone`	Zone ECUs. Host the gateway SWCs
`DoorCtrl_FL`, `SeatCtrl_FL`, `MirrorAct_FL`, `WindowSw_FL`, `DoorCtrl_FR`, `SeatCtrl_FR`, `MirrorAct_FR`	Comfort end ECUs
`ParkSensor_RR`	Park distance sensor end ECU
`TrailerModule`, `TailLamp_RR`	Trailer end ECUs

VFC and PortGroup

- Collect the ports of one function into a single PortGroup named `VFC_<function>`: `VFC_Comfort`, `VFC_ParkAssist`, `VFC_Trailer`.
- Set the CATEGORY of every PortGroup to `PARTIAL_NETWORKING`.

The VFC Family Rule

A VFC uses **the same PortGroup name** at every level. For `VFC_ParkAssist`:

Level	Example	PortGroup name	Contents
Atomic SWC	`ParkDistanceSensing`	`VFC_ParkAssist`	OUTER PORTS = `ParkDistance`
ECU composition	`ParkSensor_RR`	`VFC_ParkAssist`	OUTER PORTS = the delegated port `ParkDistance`, INNER GROUPS = `ParkDistanceSensing` / `VFC_ParkA`
Root composition	`PncVehicle`	`VFC_ParkAssist`	INNER GROUPS = `VFC_ParkAssist` of the participating direct prototypes `HPC`, `SouthZone`, and `ParkSen`

- **OUTER PORTS**: the ports of that SWC or composition in this VFC.
- **INNER GROUPS**: the same-name PortGroups of prototypes inside the composition.

> **Caution.** A differently named PortGroup, such as `PG_ParkAssist`, is not a VFC member. The PNC Dialog counts only root prototypes with a PortGroup n

Creating a PortGroup

1. At the root, double-click `ParkSensor_RR`.

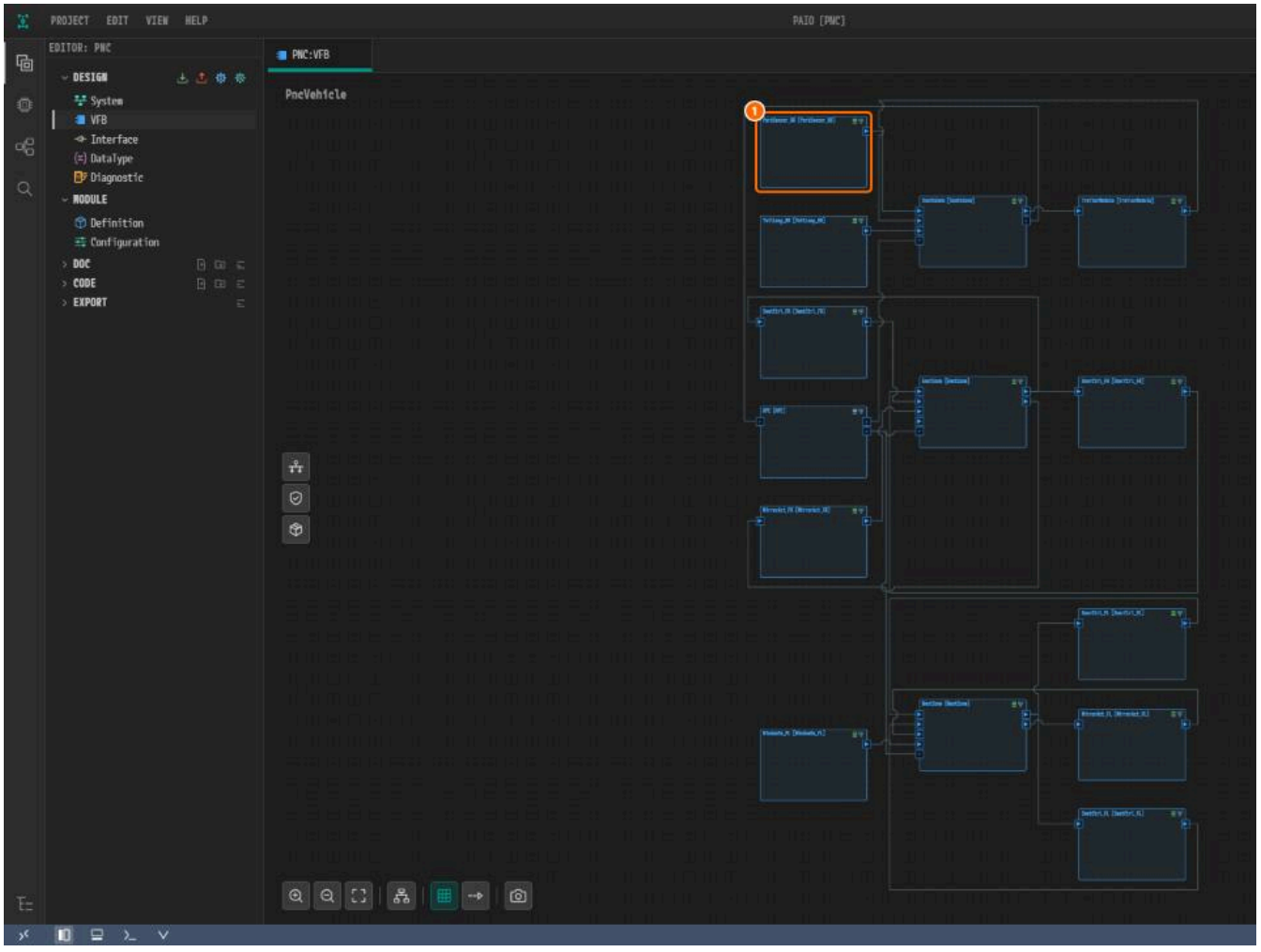


Figure 6-17. ① `ParkSensor_RR`.

2. Click **Fit to View**.

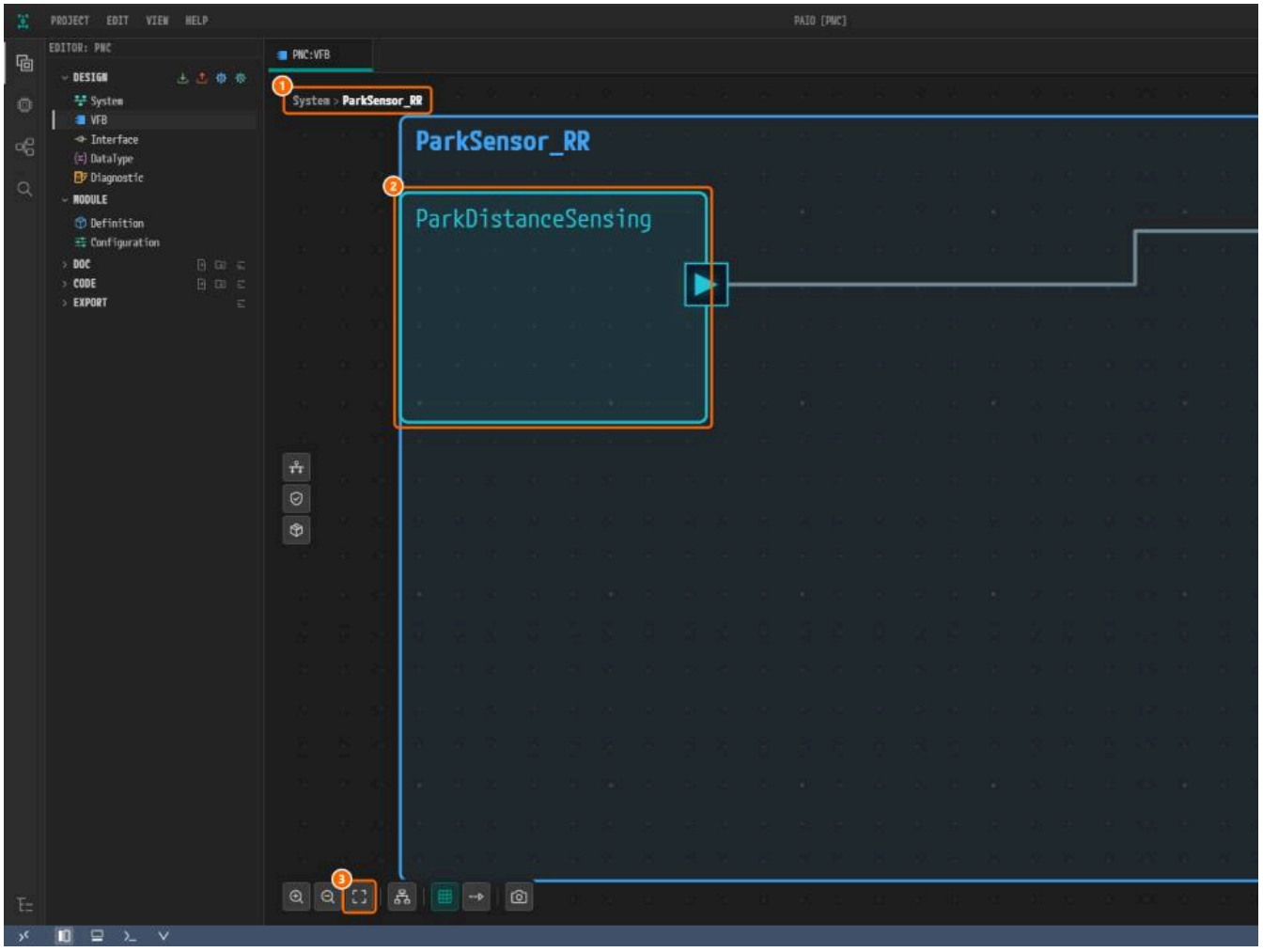


*Figure 6-18. ① The path, ② `ParkDistanceSensing`, ③ **Fit to View**.*

3. Double-click `ParkDistanceSensing`. The **SWC Internal Behavior [ParkDistanceSensing]** dialog opens.

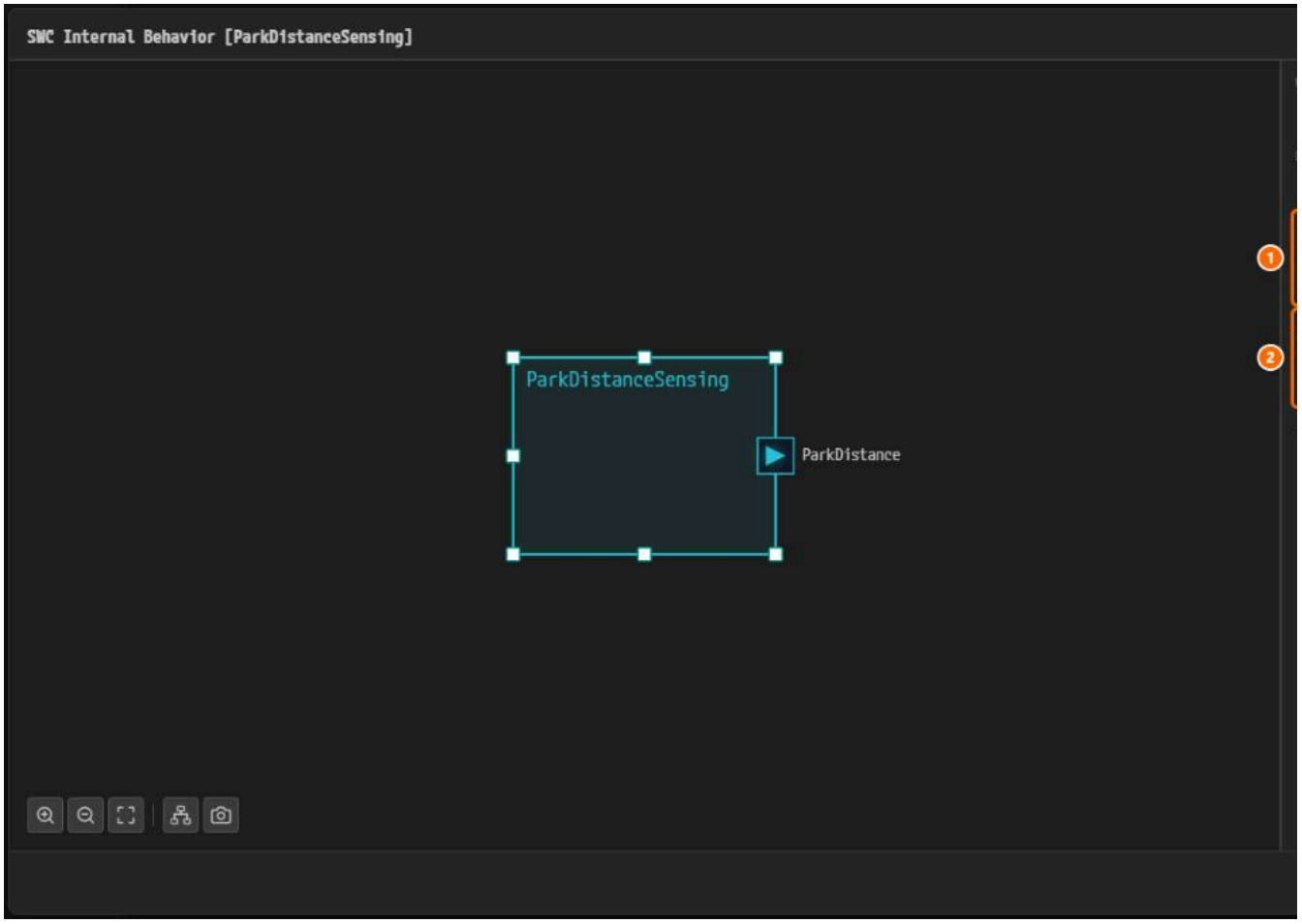


*Figure 6-19. ① **PORTS** with `ParkDistance`, ② **PORT GROUPS** (empty).*

4. Click **+** to the right of **PORT GROUPS**. A block `PortGroup_0` appears.

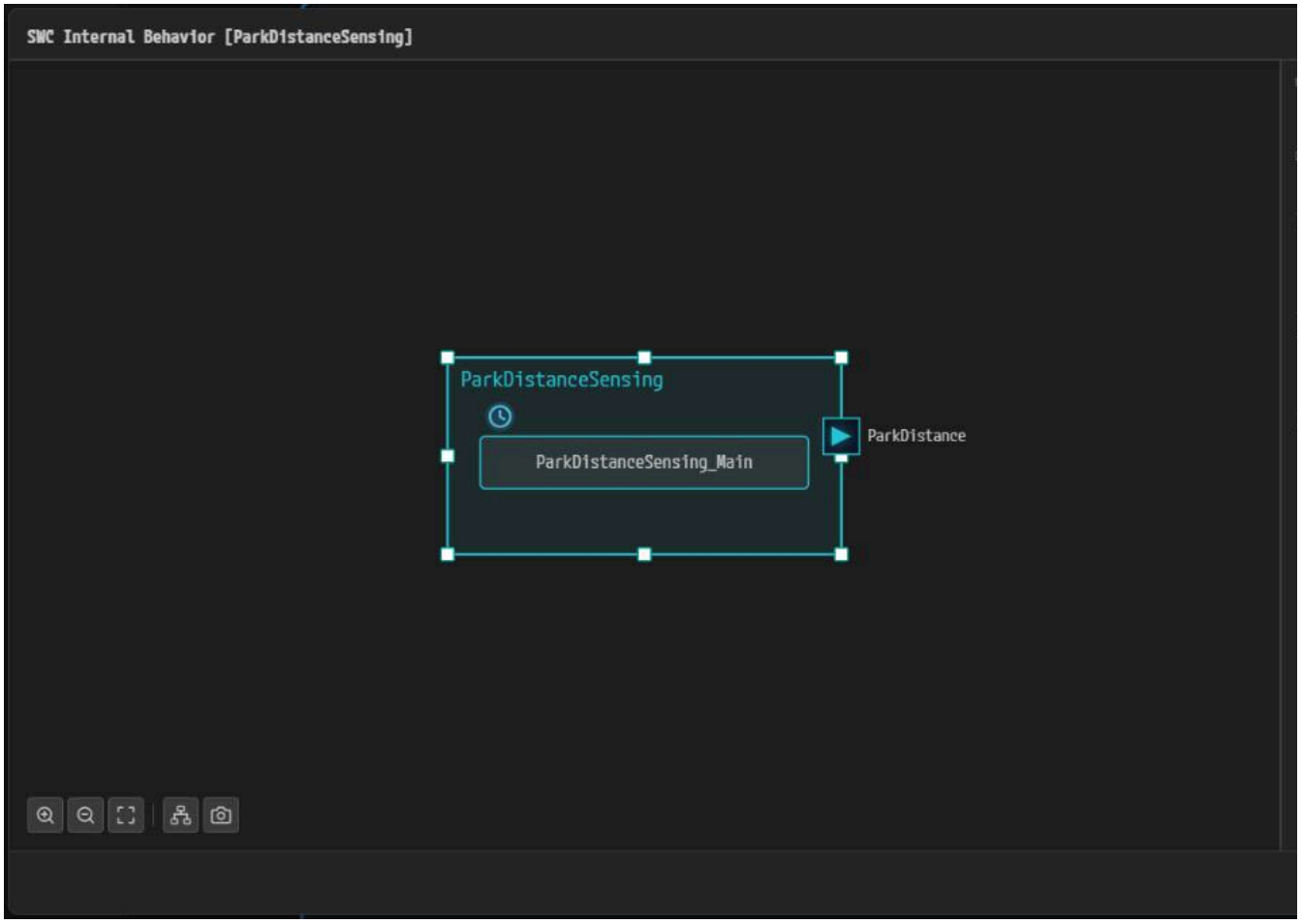


*Figure 6-20. ① **+** of **PORT GROUPS**. Figures 6-20 to 6-22 come from the finished example, so its canvas also shows a runnable.*

5. Replace **NAME** with `VFC_ParkAssist` and press **Tab**.
6. In **CATEGORY**, click **PARTIAL_NETWORKING**.
7. Click **+** to the right of **OUTER PORTS**. A row `1` appears.
8. Click ▾ in the row and select `ParkDistance`.

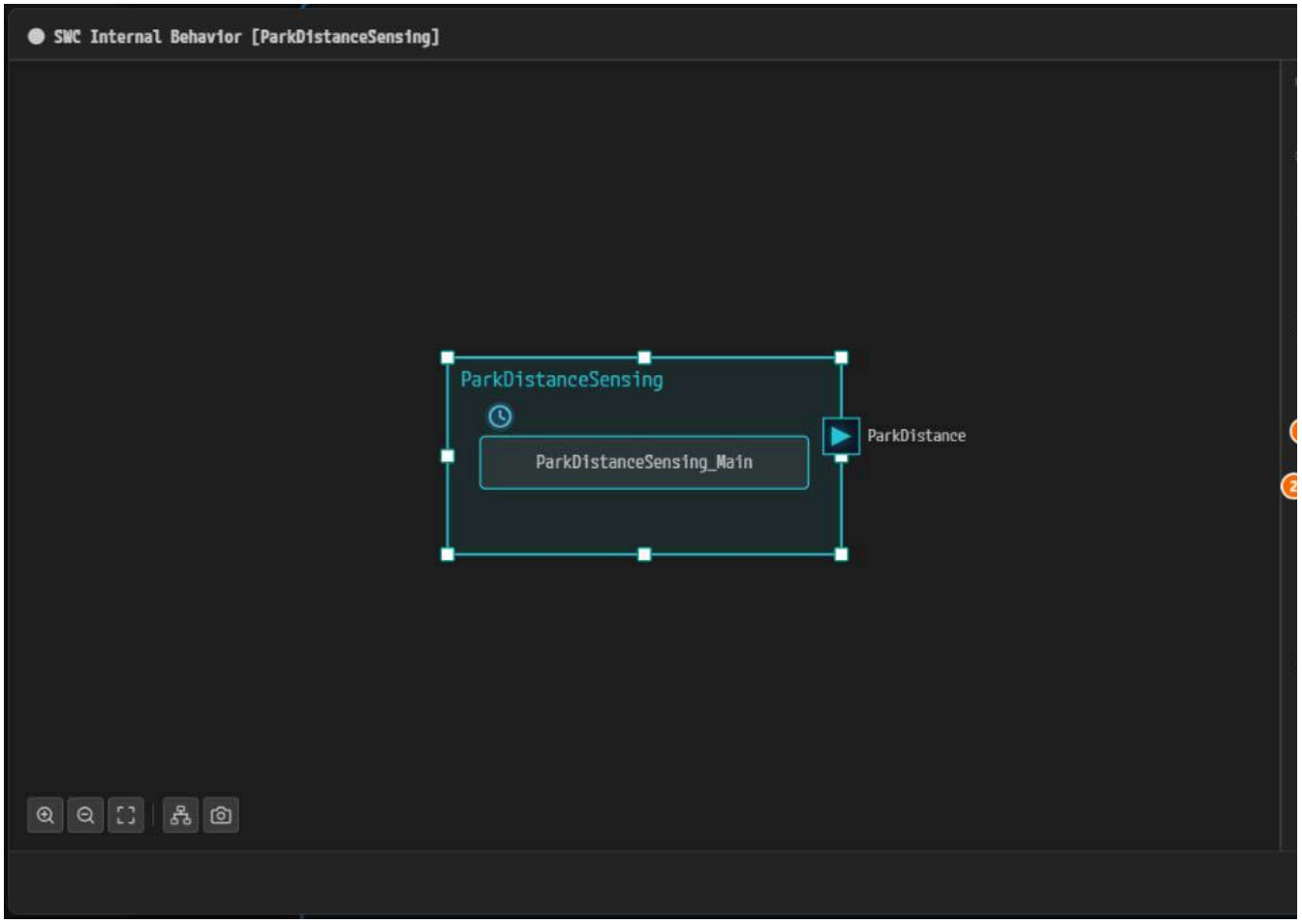


*Figure 6-21. ① **NAME**, ② **PARTIAL_NETWORKING**, ③ **+** of **OUTER PORTS**, ④ `ParkDistance`.*

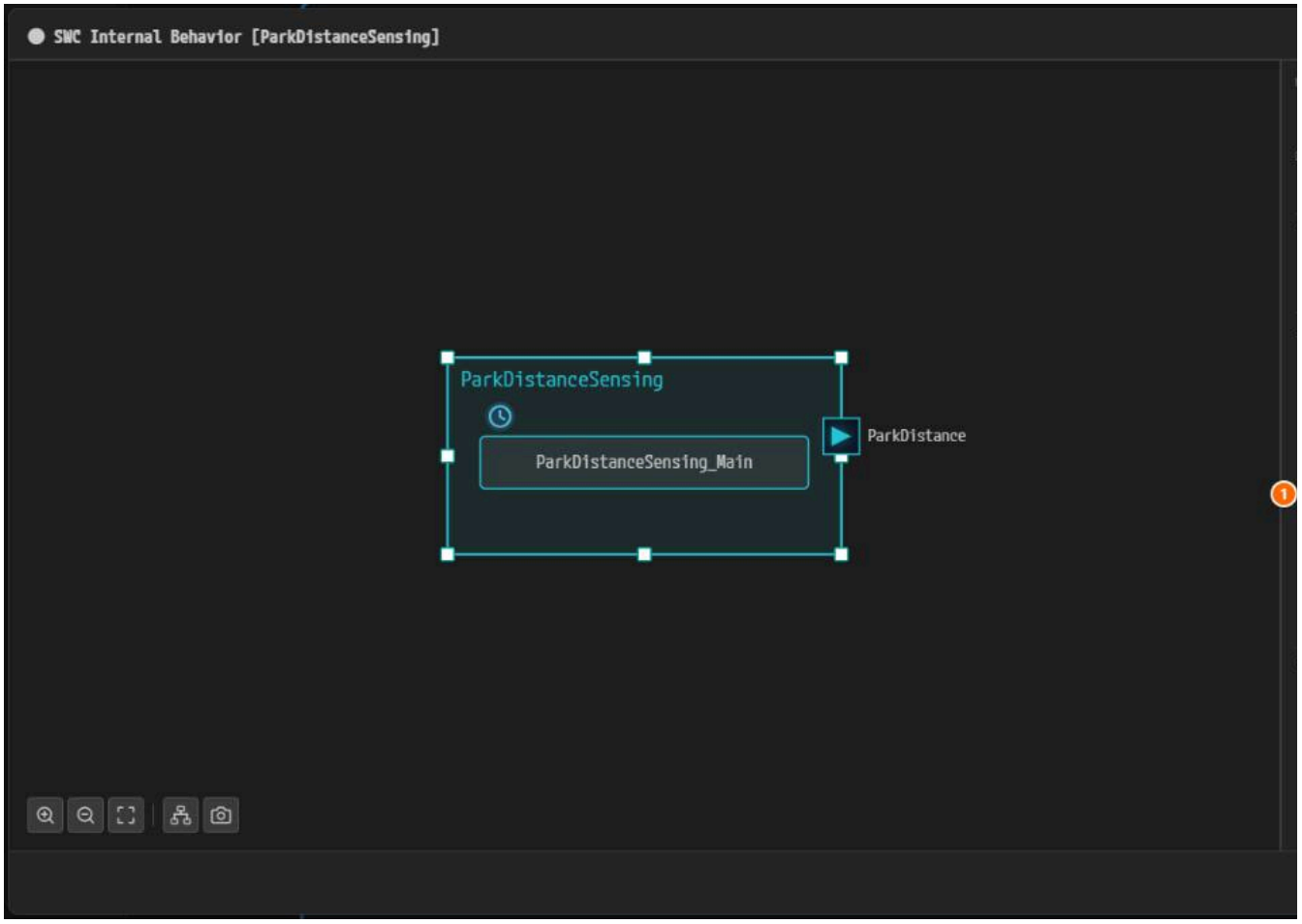


Figure 6-22. ① The finished PortGroup.

9. Save the dialog with **Ctrl+S**.
10. Click **Close**.
11. Save the VFB editor with **Ctrl+S**.

On save, PAIO creates a PortGroup of the same name on the parents.

Composition	PortGroup created automatically
`ParkSensor_RR`	`VFC_ParkAssist`, INNER GROUPS = `ParkDistanceSensing` / `VFC_ParkAssist`
`PncVehicle` (root)	`VFC_ParkAssist`, INNER GROUPS = `ParkSensor_RR` / `VFC_ParkAssist`

Checking the Automatically Created PortGroup

1. Click the title row of the `ParkSensor_RR` frame.
2. Check **PORT GROUPS** in the **COMPOSITION** panel.

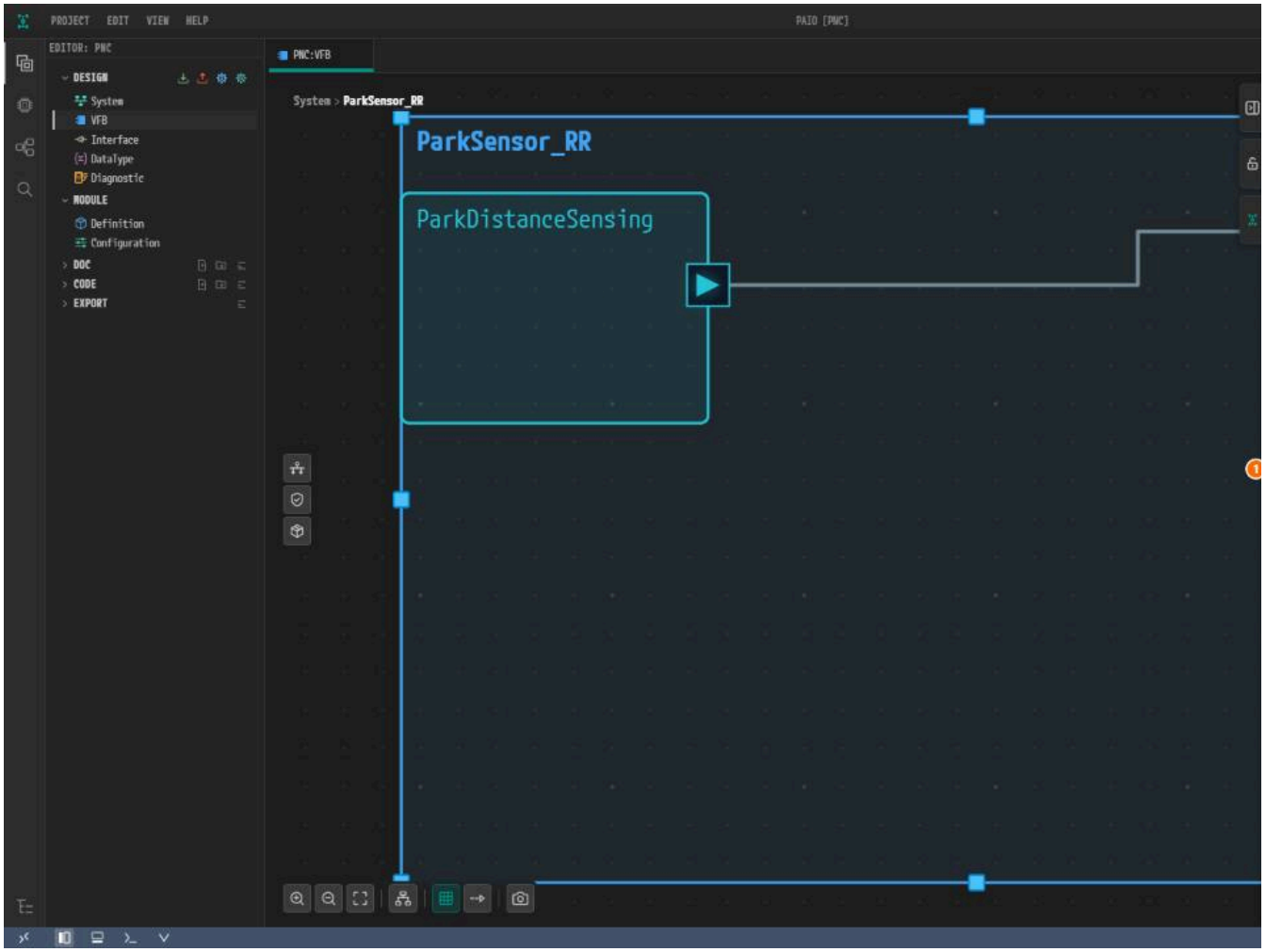


Figure 6-23. ① `VFC_ParkAssist` with INNER GROUPS `ParkDistanceSensing` / `VFC_ParkAssist`.

> **Note.** An automatically created composition PortGroup has no OUTER PORTS. You fill them in PNC Dialog — Per-PNC Binding.

The Remaining PortGroups

There are 33 PortGroups, all `PARTIAL_NETWORKING`.

- **Atomic SWCs:** create them as above.
- **ECU compositions and the root:** created automatically on save, with **INNER GROUPS**.
- **OUTER PORTS of ECU compositions, and the PortGroups of the application AAs** (`ComfortApp`, `ParkAssistApp`, `VehicleStateApp`): cannot be edited **HIERARCHY** checkboxes in PNC Dialog — Per-PNC Binding; the **OUTER PORTS** columns below are the ports to check there.

Atomic SWCs and application AAs (OUTER PORTS)

SWC	PortGroup	OUTER PORTS
`DoorControl`	`VFC_Comfort`	`DoorStatus`, `ComfortCmd`
`SeatControl`	`VFC_Comfort`	`SeatStatus`, `ComfortCmd`
`MirrorControl`	`VFC_Comfort`	`MirrorStatus`, `MirrorCmd`
`WindowSwitch`	`VFC_Comfort`	`WindowSwitch`
`ZoneGateway`	`VFC_Comfort`	`ComfortCmd`, `MirrorCmd`, `DoorStatus`, `SeatStatus`, `MirrorStatus`, `WindowSwitch`, `ComfortControlSvc`
`ParkDistanceSensing`	`VFC_ParkAssist`	`ParkDistance` (created in this chapter)
`RearGateway`	`VFC_ParkAssist`	`ParkDistance`, `ParkDistanceSvc`
`RearGateway`	`VFC_Trailer`	`TrailerStatus`, `TrailerCmd`, `TailLamp`, `VehicleStateSvc`
`TrailerControl`	`VFC_Trailer`	`TrailerStatus`, `TrailerCmd`
`TailLampControl`	`VFC_Trailer`	`TailLamp`
`ComfortApp` (AA)	`VFC_Comfort`	`ComfortControlSvc`
`ParkAssistApp` (AA)	`VFC_ParkAssist`	`ParkDistanceSvc`, `VehicleStateSvc`
`VehicleStateApp` (AA)	`VFC_Trailer`	`VehicleStateSvc`

ECU compositions (OUTER PORTS + INNER GROUPS)

Composition	PortGroup	OUTER PORTS	INNER G
`DoorCtrl_FL`, `DoorCtrl_FR`	`VFC_Comfort`	`DoorStatus`, `ComfortCmd`	`DoorCo`
`SeatCtrl_FL`, `SeatCtrl_FR`	`VFC_Comfort`	`SeatStatus`, `ComfortCmd`	`SeatCo`
`MirrorAct_FL`, `MirrorAct_FR`	`VFC_Comfort`	`MirrorStatus`, `MirrorCmd`	`Mirror`
`WindowSw_FL`	`VFC_Comfort`	`WindowSwitch`	`Window`
`WestZone`, `EastZone`	`VFC_Comfort`	`ComfortCmd`, `MirrorCmd`, `DoorStatus`, `SeatStatus`, `MirrorStatus`, `WindowSwitch`, `ComfortControlSvc`	`Gatewa`
`ParkSensor_RR`	`VFC_ParkAssist`	`ParkDistance`	`ParkDi`
`SouthZone`	`VFC_ParkAssist`	`ParkDistance`, `ParkDistanceSvc`	`Gatewa`
`SouthZone`	`VFC_Trailer`	`TrailerStatus`, `TrailerCmd`, `TailLamp`, `VehicleStateSvc`	`Gatewa`
`TrailerModule`	`VFC_Trailer`	`TrailerStatus`, `TrailerCmd`	`Traile`
`TailLamp_RR`	`VFC_Trailer`	`TailLamp`	`TailLa`
`HPC`	`VFC_ParkAssist`	`ParkDistanceSvc`	`ParkAs`
`HPC`	`VFC_Comfort`	`ComfortControlSvc`	`Comfor`
`HPC`	`VFC_Trailer`	`VehicleStateSvc`	`Vehicl`

The root `PncVehicle` (INNER GROUPS only, created automatically when the VFB is saved)

PortGroup	Participating direct prototypes
`VFC_Comfort`	`HPC`, `WestZone`, `EastZone`, `DoorCtrl_FL`, `SeatCtrl_FL`, `MirrorAct_FL`, `WindowSw_FL`, `DoorCtrl_FR`, `SeatCtrl_FR`, `MirrorAct_FR` (10)
`VFC_ParkAssist`	`HPC`, `SouthZone`, `ParkSensor_RR` (3)
`VFC_Trailer`	`HPC`, `SouthZone`, `TrailerModule`, `TailLamp_RR` (4)

Checking VFCs with the PNC Overlay

PortGroups are not drawn on the canvas. The **VFC (PNC Overlay)** icon shows which compositions take part in a VFC.

1. Return to the root and click **Fit to View**.
2. Click the first icon in the strip at the left edge of the canvas, **VFC (PNC Overlay)**. The **PNC OVERLAY** panel opens, and compositions outside the select

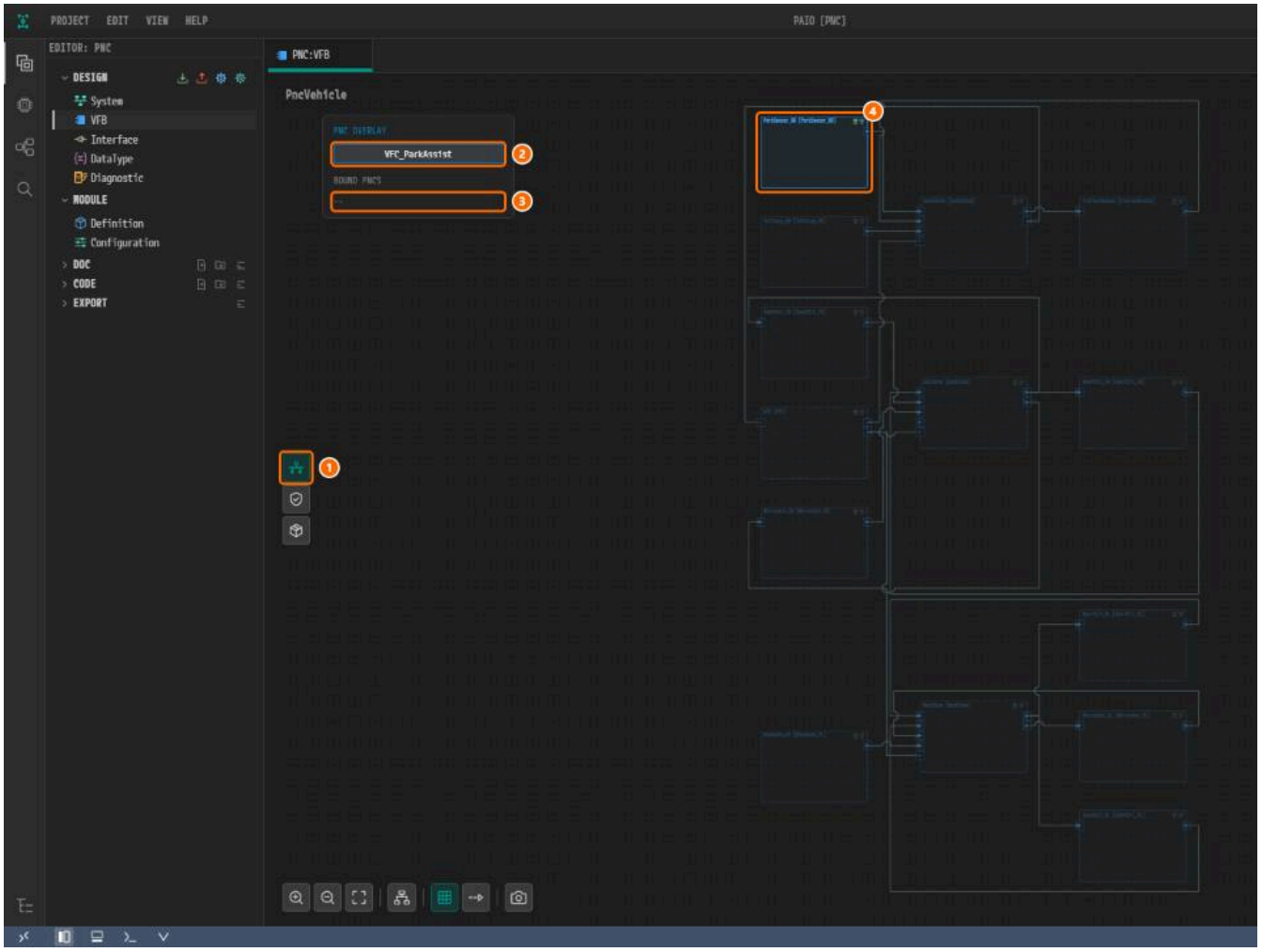


*Figure 6-24. With only `VFC_ParkAssist` on `ParkDistanceSensing` created. ① **VFC (PNC Overlay)**, ② `VFC_ParkAssist`, ③ **BOUND PNCS** = `--`, ④ `Park`.*

3. Zoom with **Ctrl**+wheel to read the names.

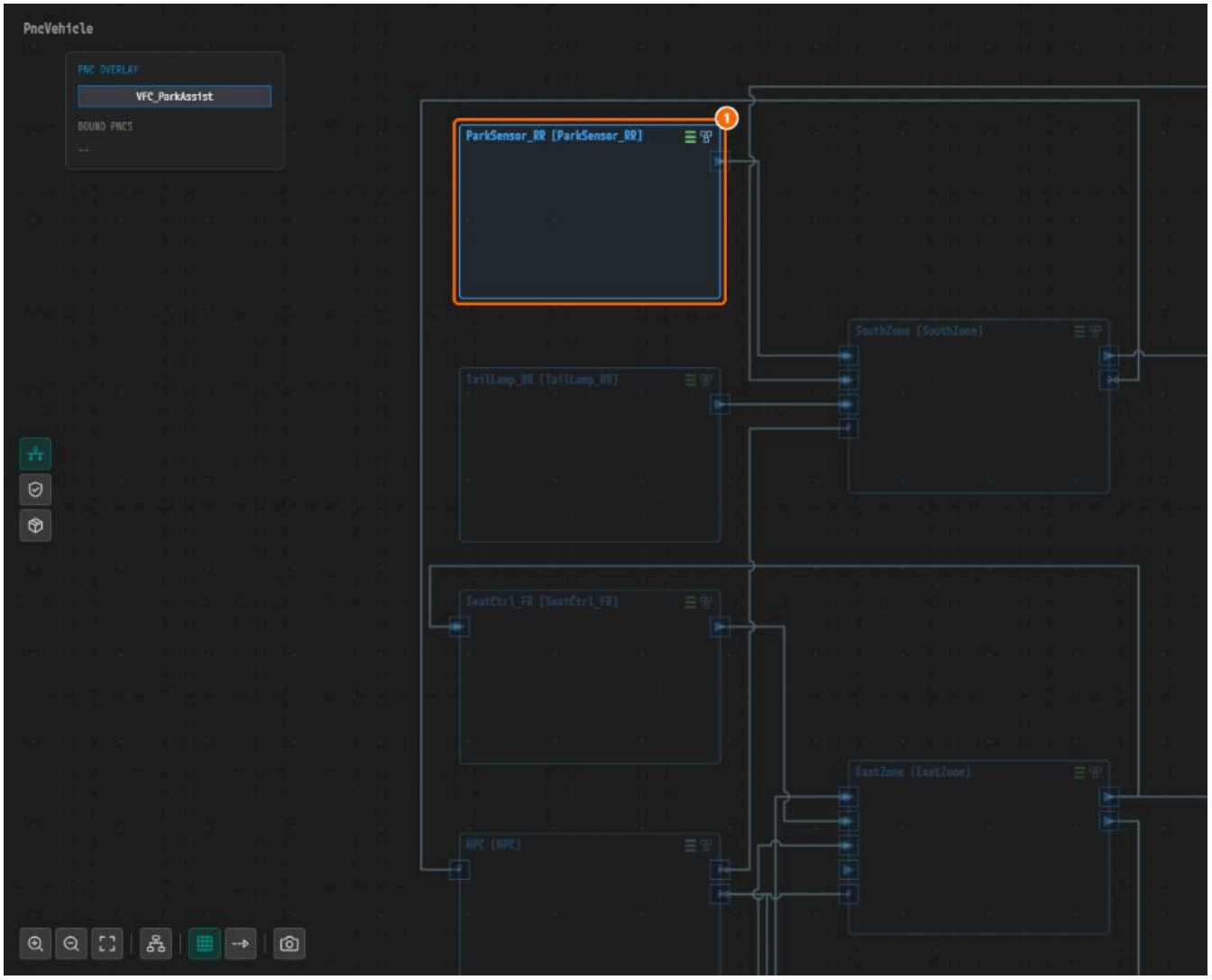


Figure 6-25. ① `ParkSensor_RR`, not dimmed.

4. Click the icon again to turn the overlay off.

The Overlay in the Finished Example

With all 33 PortGroups and the PNC bindings in place, the list holds `VFC_Comfort`, `VFC_ParkAssist`, and `VFC_Trailer`.

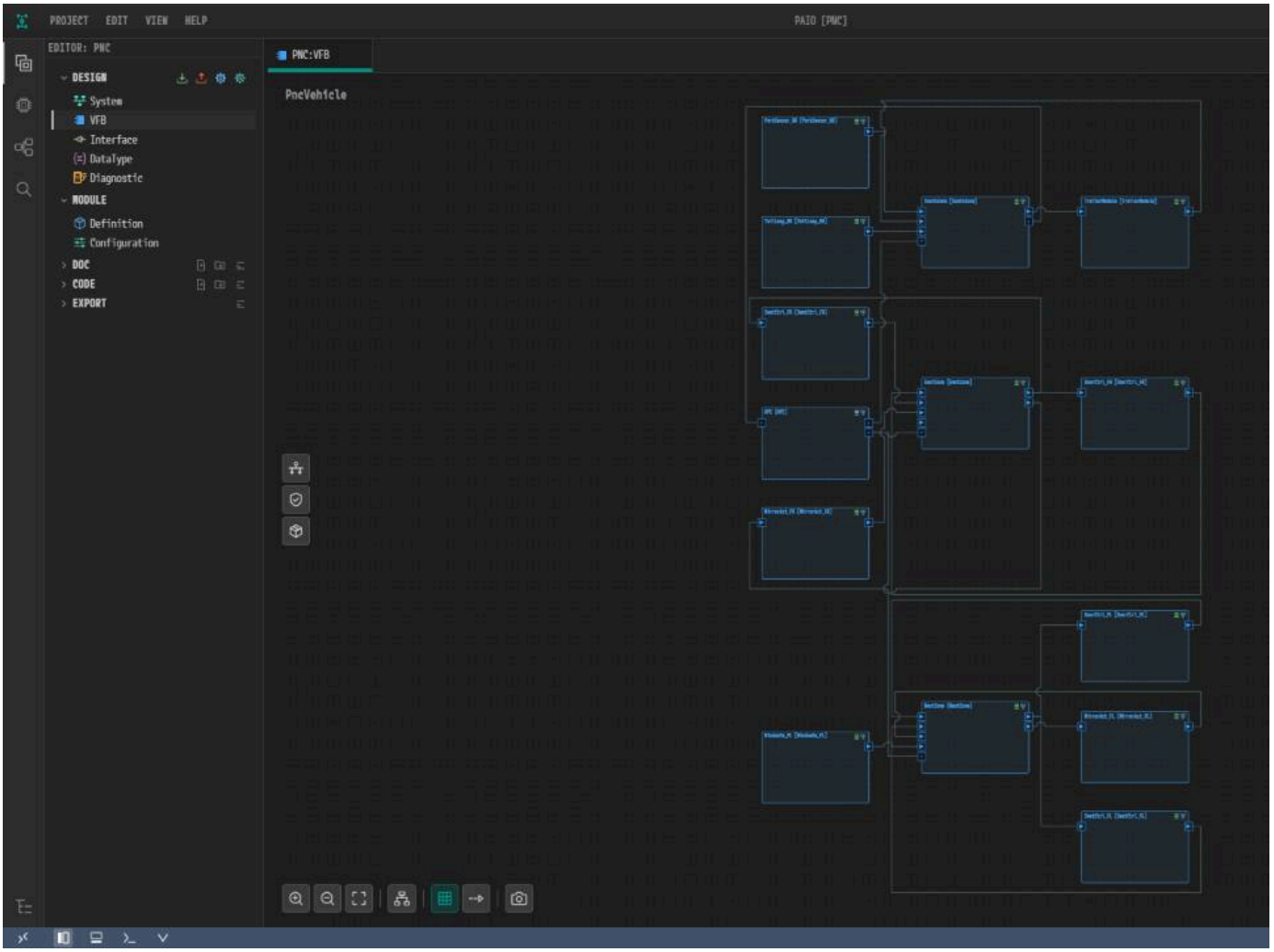


Figure 6-26. The finished root. It looks the same as Figure 6-16.

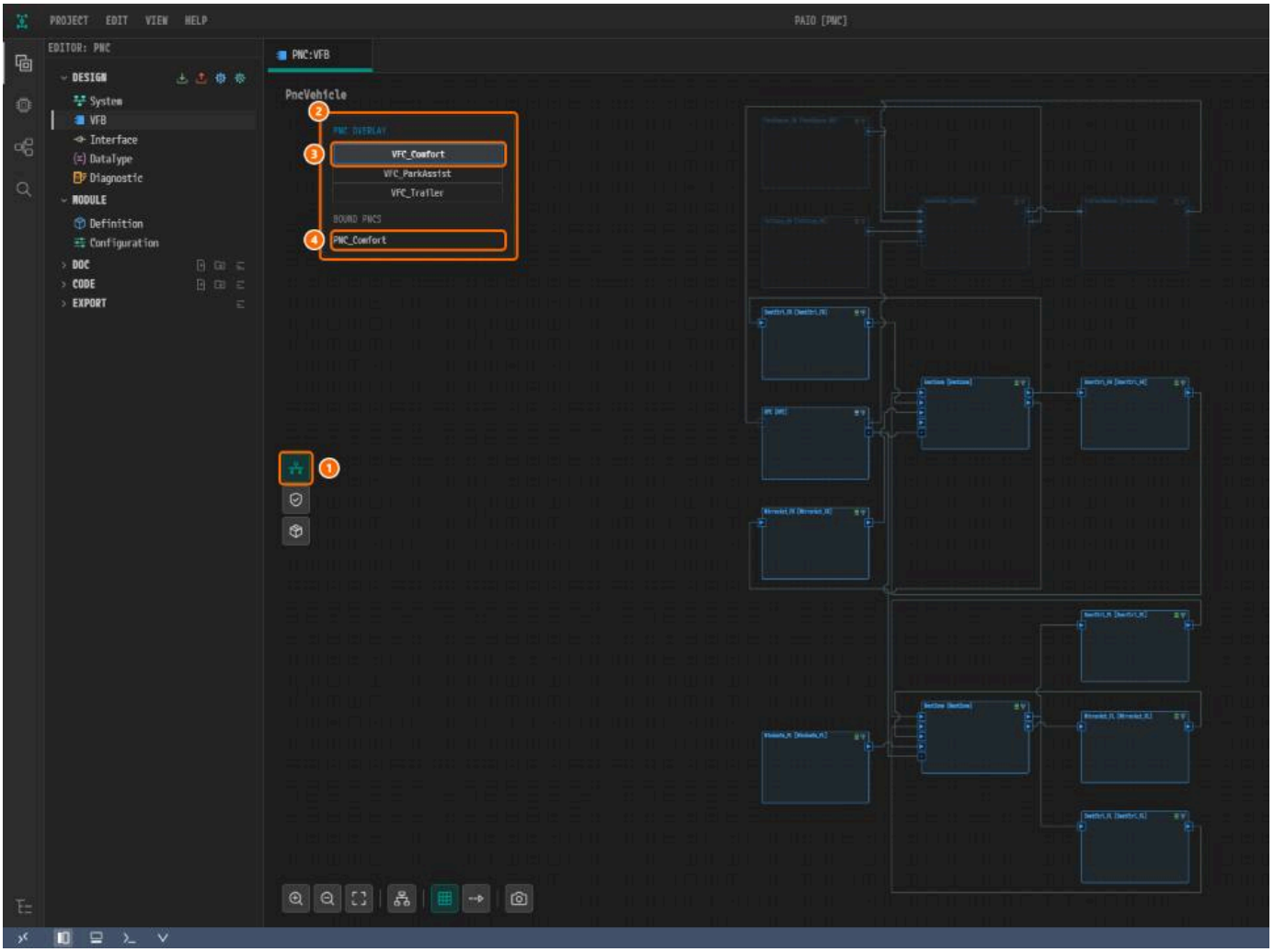


*Figure 6-27. ① **VFC (PNC Overlay)**, ② **PNC OVERLAY**, ③ `VFC_Comfort`, ④ **BOUND PNCS** = `PNC_Comfort`.*



*Figure 6-28. The **PNC OVERLAY** panel. **BOUND PNCS** lists the PNCs that reference the VFC (read-only).*

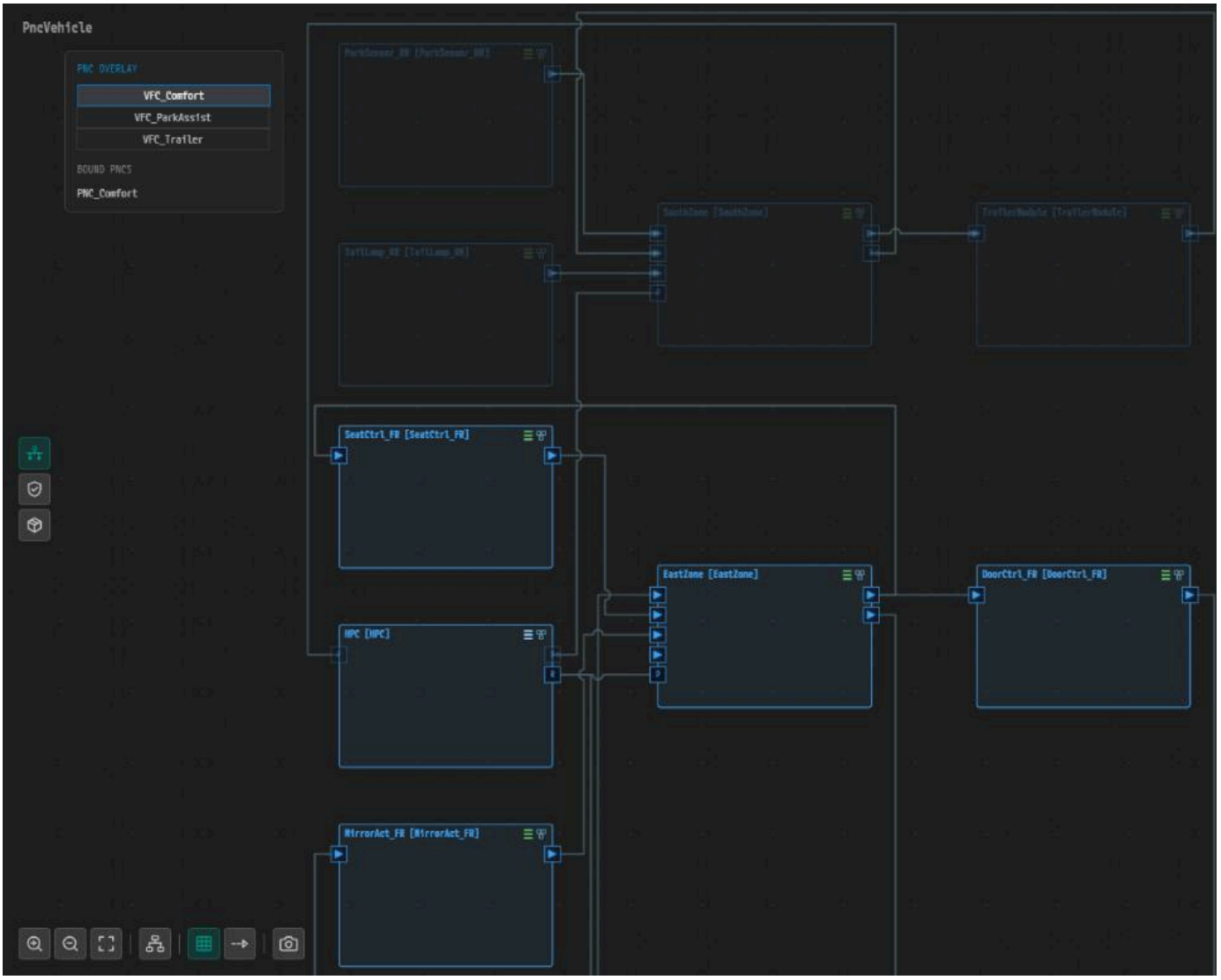


Figure 6-29. `ParkSensor_RR`, `TailLamp_RR`, `SouthZone`, and `TrailerModule` are dimmed.

Why Is It Designed This Way?

- **PNC and VFC.** A PNC references a VFC to record which functions need it awake.
- **Same name at every level.** A VFC spans several ECUs and the HPC; same-name PortGroups let you follow every port from one VFC at the root.
- **Composition name = node name.** The composition leads straight to its node, so you know which ECUs and channels a VFC spans.

Checklist

- ☐ The root `PncVehicle` holds 14 compositions named after the System editor nodes.
- ☐ `ParkSensor_RR` contains `ParkDistanceSensing`, delegated to the boundary port `ParkDistance`, which connects to `ParkDistance` (R) of `SouthZon`
- ☐ After reopening the editor, the header reads `ParkSensor_RR [ParkSensor_RR]` with an ECU icon.
- ☐ `ParkDistanceSensing` has `VFC_ParkAssist` (**PARTIAL_NETWORKING**, OUTER PORTS `ParkDistance`), and `ParkSensor_RR` has `VFC_ParkAssist` with IN `VFC_ParkAssist`.
- ☐ You created every atomic SWC PortGroup in The Remaining PortGroups.
- ☐ The overlay lists `VFC_Comfort`, `VFC_ParkAssist`, `VFC_Trailer`; for `VFC_ParkAssist` only `HPC`, `SouthZone`, and `ParkSensor_RR` are not dimmed. **BOU**
- ☐ The **PNC:VFB** tab has no unsaved-change marker.

Next Step

In PNC Dialog — Per-PNC Binding you bind a VFC, an I-PDU Group, physical channels, service instances, and wakeup frames to each PNC.

PNC Dialog: Per-PNC Bindings

What You Will Do

In this chapter you bind the VFC, I-PDU groups, physical channels, service instances and wakeup frames of each PNC in the PNC dialog. You follow `PNC_Pa` `PNC_Trailer`, and enter the rest from a table.

Before You Begin

- Start after finishing VFB and VFC (PortGroup).
- The PNCs `PNC_Comfort` 17, `PNC_ParkAssist` 18 and `PNC_Trailer` 19 exist, and the I-PDU groups were bound to them in Chapter 4. The PortGroups `V` exist but are not bound to a PNC yet.
- Figures 7-2 to 7-9, 7-14 and 7-15 were captured with nothing bound. On your screen **PNC GROUPS** already shows two rows.

Section	Value for `PNC_ParkAssist`
VIRTUAL FUNCTION CLUSTERS (VFC)	`VFC_ParkAssist`
PNC GROUPS	`RearCAN_PNC_ParkAssist_Rx` (`ParkDistance_Pdu`), `RearCAN_PNC_ParkAssist_Tx` (no PDUs)
PHYSICAL CHANNELS	CAN `RearCAN` / `RearCAN_Channel`, ETHERNET `Backbone` / `VLAN_Rear (VLAN 200)`
SERVICE INSTANCES	`ParkDistance_Provider`, `ParkDistance_Consumer`
WAKEUP FRAMES	None

Opening the PNC Dialog

1. In the **EDITOR** view, click **System** under **DESIGN** to open the tab `PNC:System`.

2. Click the second (lower) icon on the left edge of the canvas (tooltip **Partial Network (PNC)**). The **Partial Network (PNC) Configuration** dialog opens.

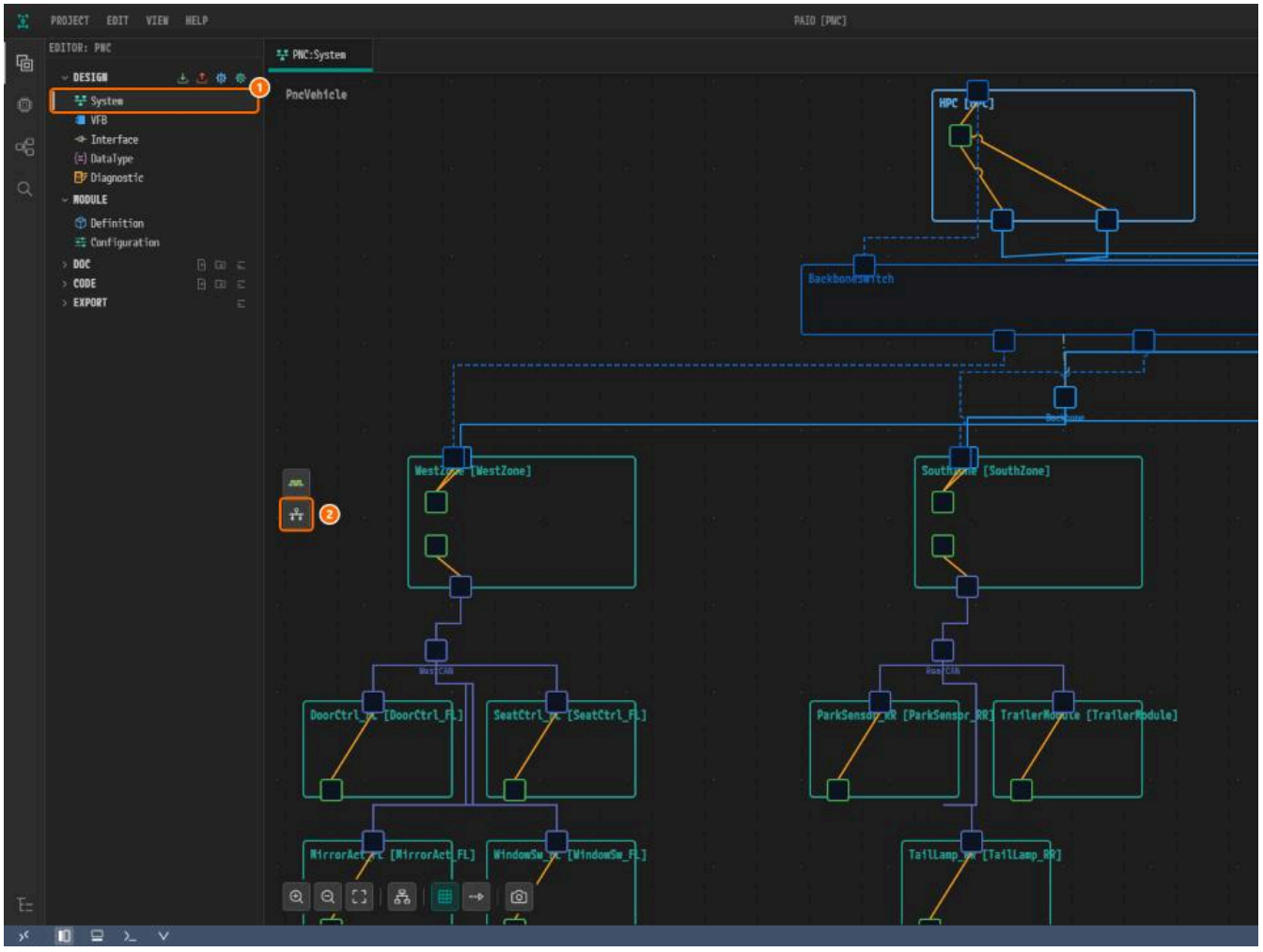


*Figure 7-1. ① **System** under **DESIGN**, ② the PNC button.*

3. In the **PARTIAL NETWORKS** list, click **PNC 18 PNC_ParkAssist**.

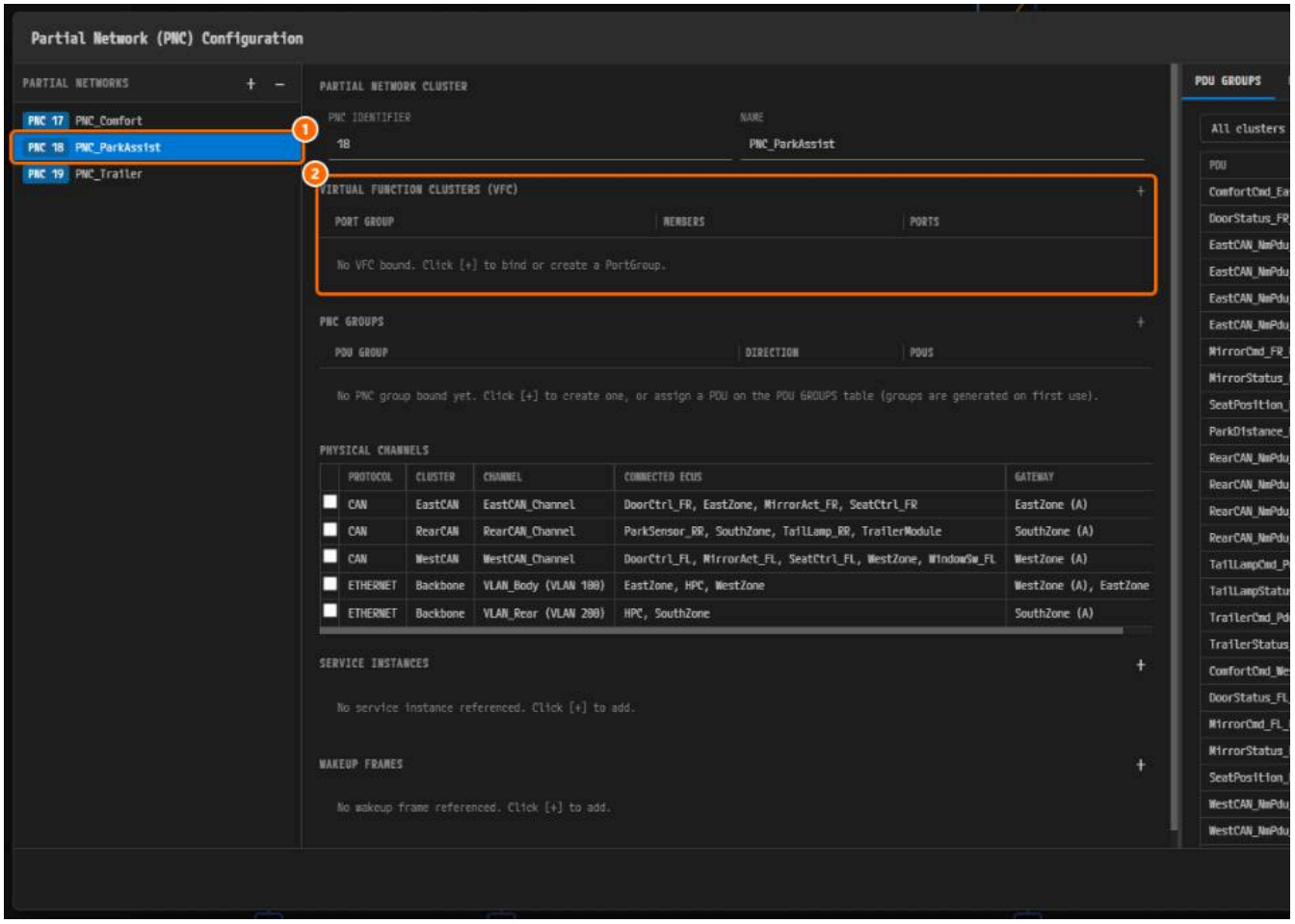


*Figure 7-2. ① **PNC 18 PNC_ParkAssist**, ② **VIRTUAL FUNCTION CLUSTERS (VFC)**, still empty.*

The middle shows the sections of the selected PNC, and the right side holds the viewer tabs **PDU GROUPS**, **MATRIX** and **NM VECTOR**. A ● before the dialog

Binding a VFC

1. Click the **+** right of the **VIRTUAL FUNCTION CLUSTERS (VFC)** header. A menu opens.
2. Click **VFC_ParkAssist**.

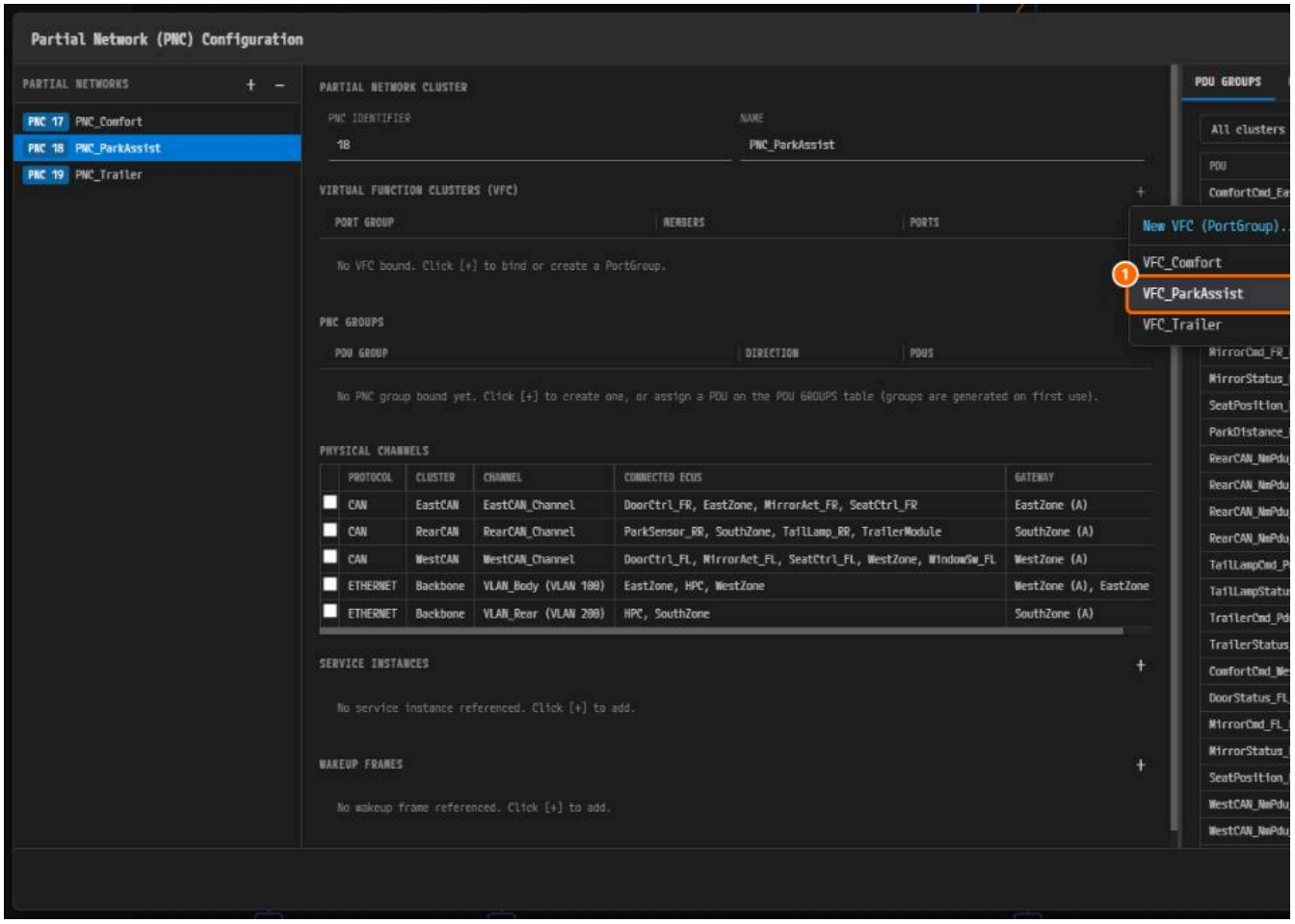


Figure 7-3. ① `VFC_ParkAssist`.

A row appears in the VFC table.

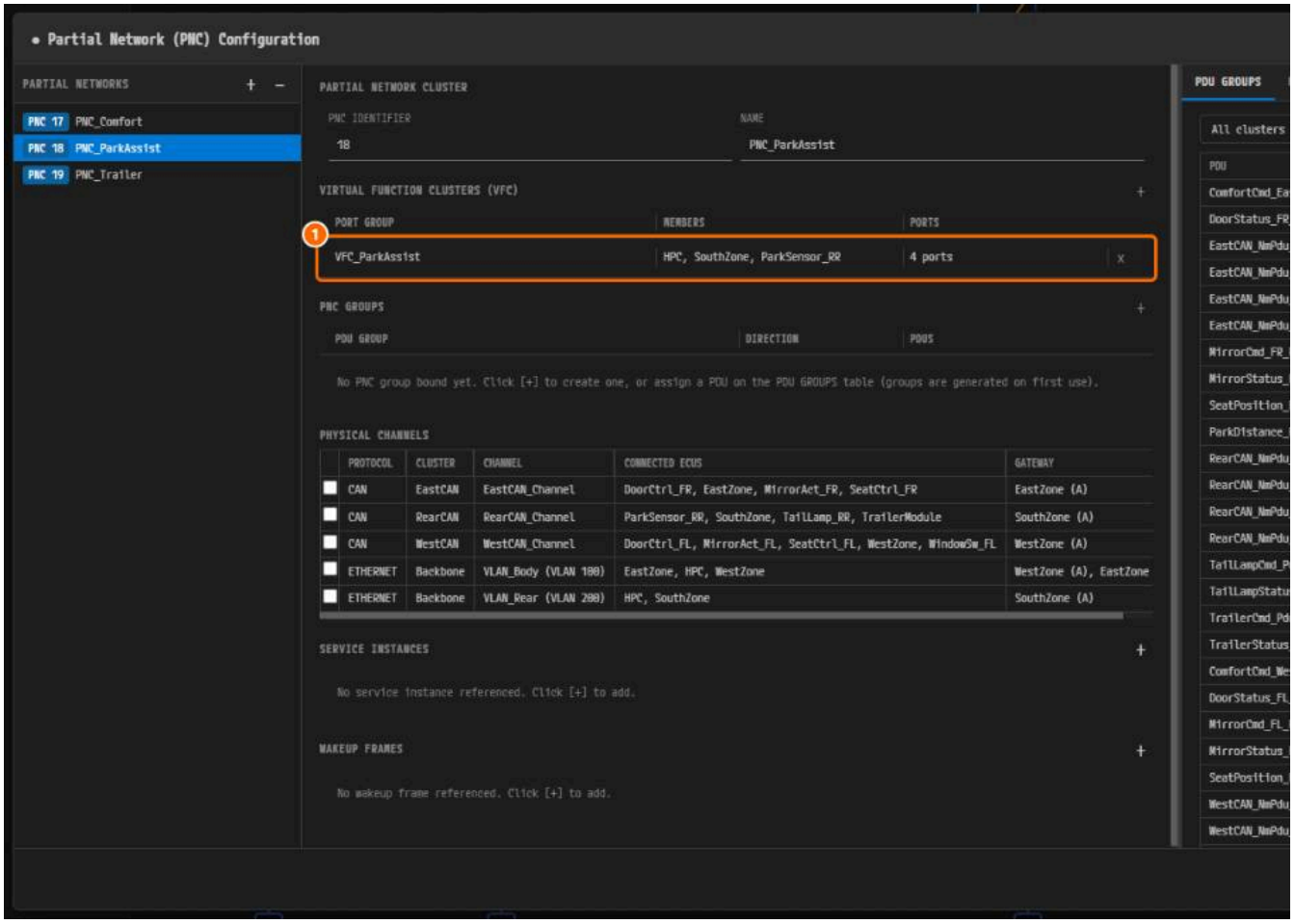


*Figure 7-4. ① **MEMBERS** `HPC, SouthZone, ParkSensor_RR`, **PORTS** `4 ports`.*

3. Click the `VFC_ParkAssist` row to expand it.

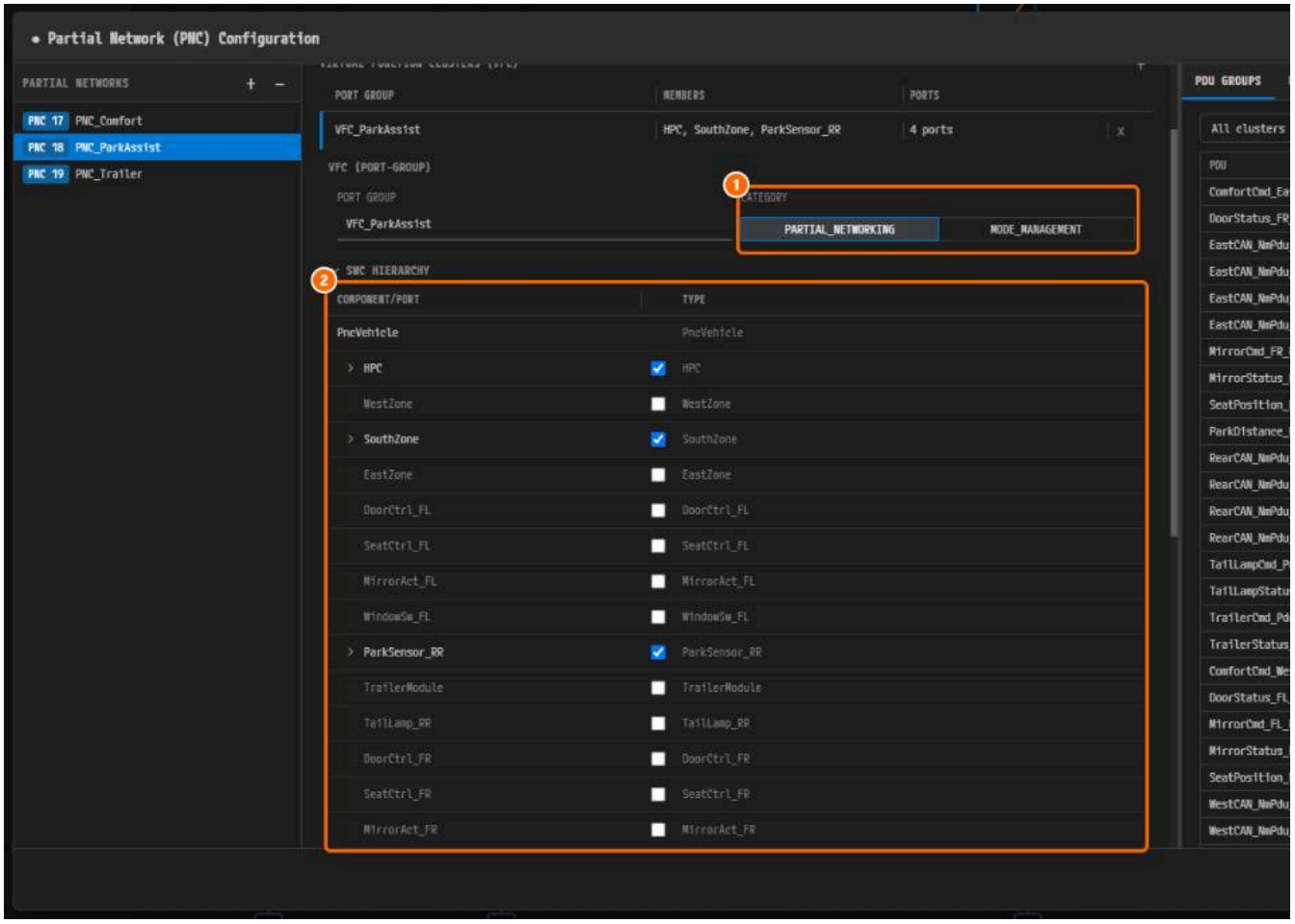


*Figure 7-5. ① **CATEGORY PARTIAL_NETWORKING**, ② **SWC HIERARCHY**.*

4. In **SWC HIERARCHY**, check the compositions `HPC`, `SouthZone` and `ParkSensor_RR`. Expand each one and check its ports.

Composition	Ports to check
`HPC`	`ParkDistanceSvc`
`SouthZone`	`ParkDistance`, `ParkDistanceSvc`
`ParkSensor_RR`	`ParkDistance`

   Also check the ports `ParkDistanceSvc` and `VehicleStateSvc` of the AA `ParkAssistApp` inside `HPC`.

5. Click the row again to collapse it.

> **Caution.** A check in **SWC HIERARCHY** is written to the model at once, without **Ctrl+S**. Clearing a composition check deletes its PortGroup, so touch only

Checking PNC GROUPS

PNC GROUPS already holds the groups you created in Chapter 4. Confirm that it shows `RearCAN_PNC_ParkAssist_Rx` (**RX**, `1 PDU`) and `RearCAN_PNC_Park`

The steps below show the other way to bind a group: assigning a PNC to a PDU in the **PDU GROUPS** table. Use it only for a PDU that has no PNC yet. In this

1. Type `ParkDistance` in the search field of **PDU GROUPS**. Drag the divider left of the viewer to the left until **PNC MEMBERSHIP** is visible.

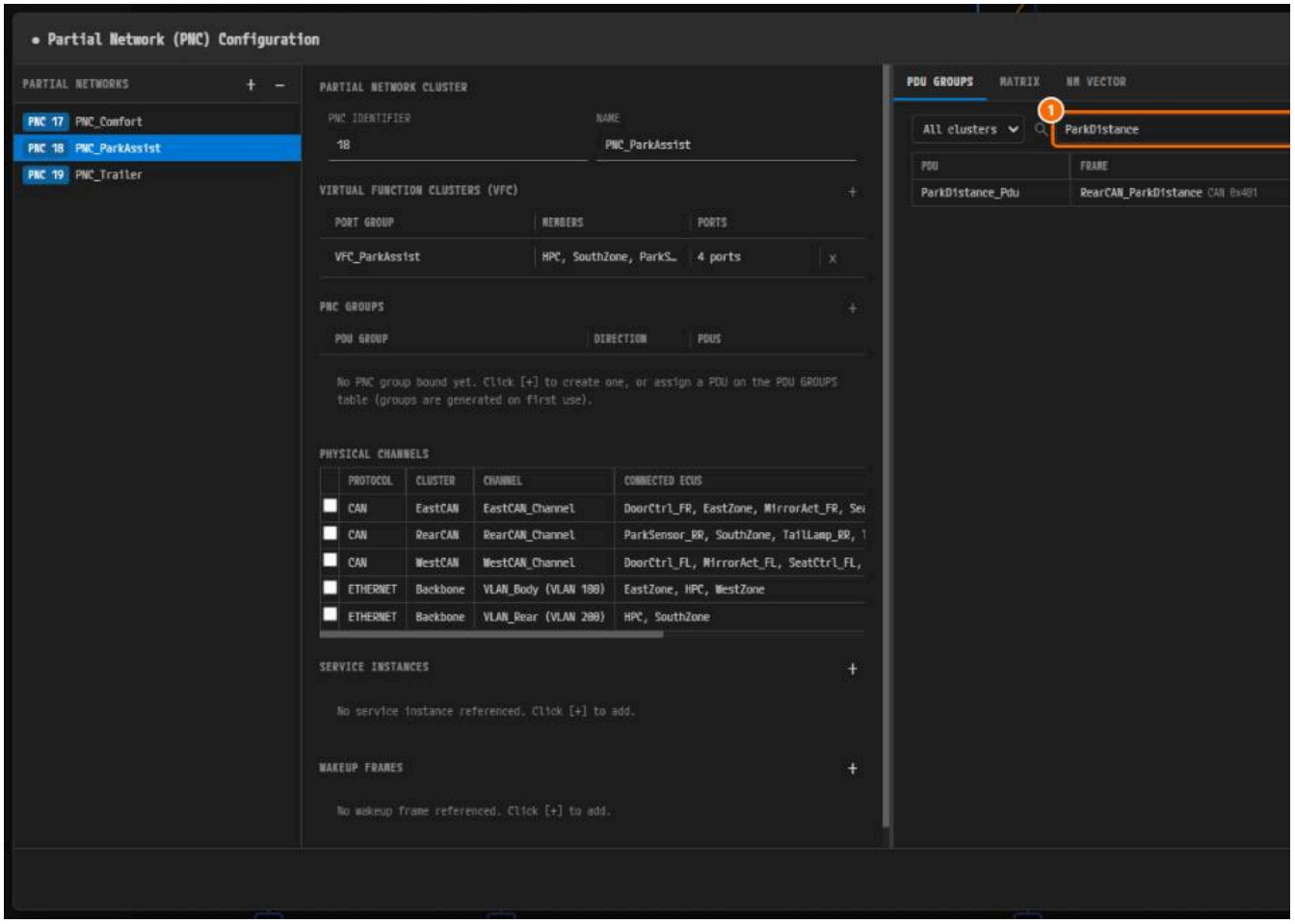


*Figure 7-6. ① The search field, ② **PNC MEMBERSHIP**.*

2. Click the **PNC MEMBERSHIP** cell and check **PNC_ParkAssist**.

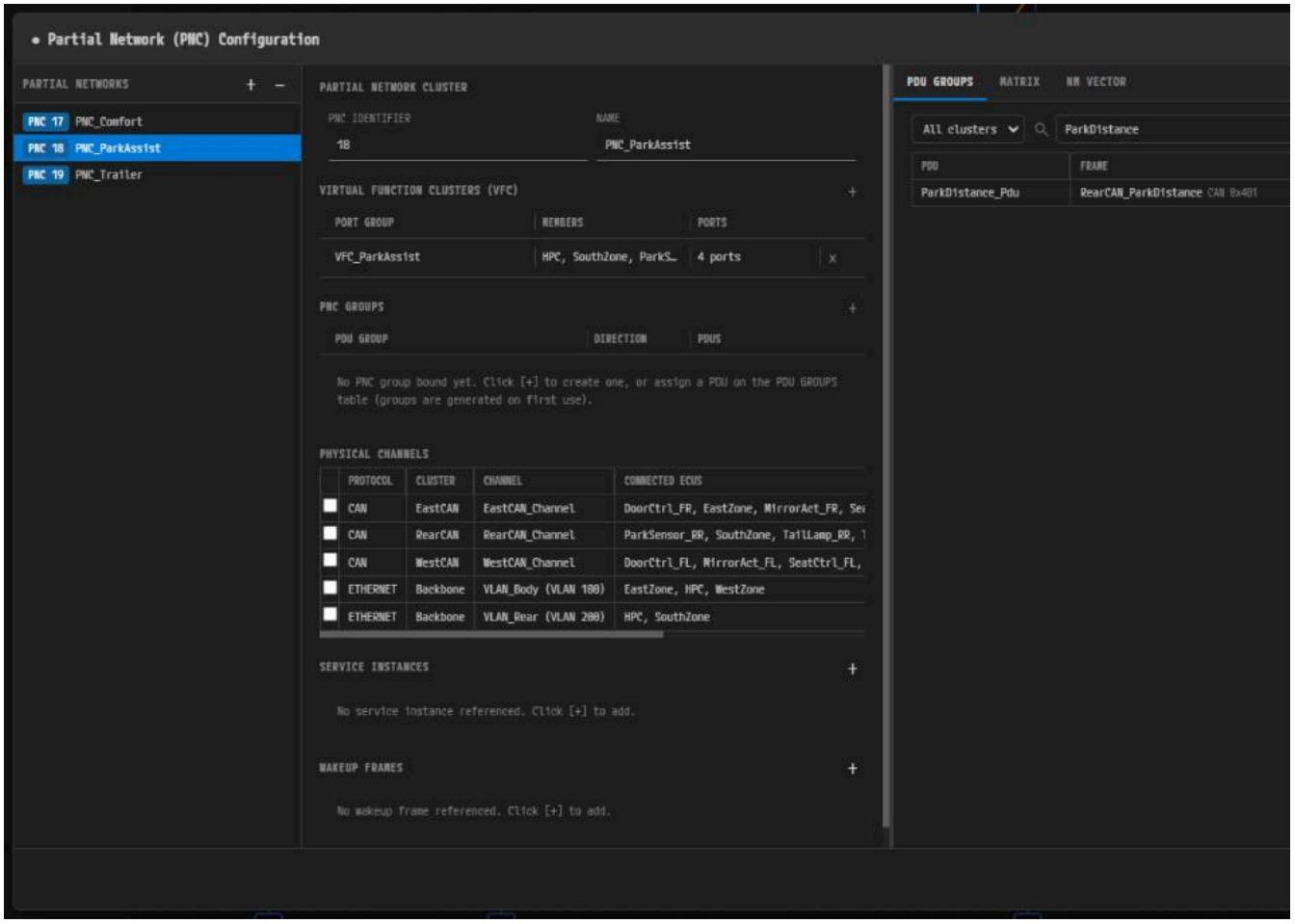


Figure 7-7. ① `PNC_ParkAssist`.

3. Press **Esc.** The groups `RearCAN_PNC_ParkAssist_Rx` and `RearCAN_PNC_ParkAssist_Tx` are bound.

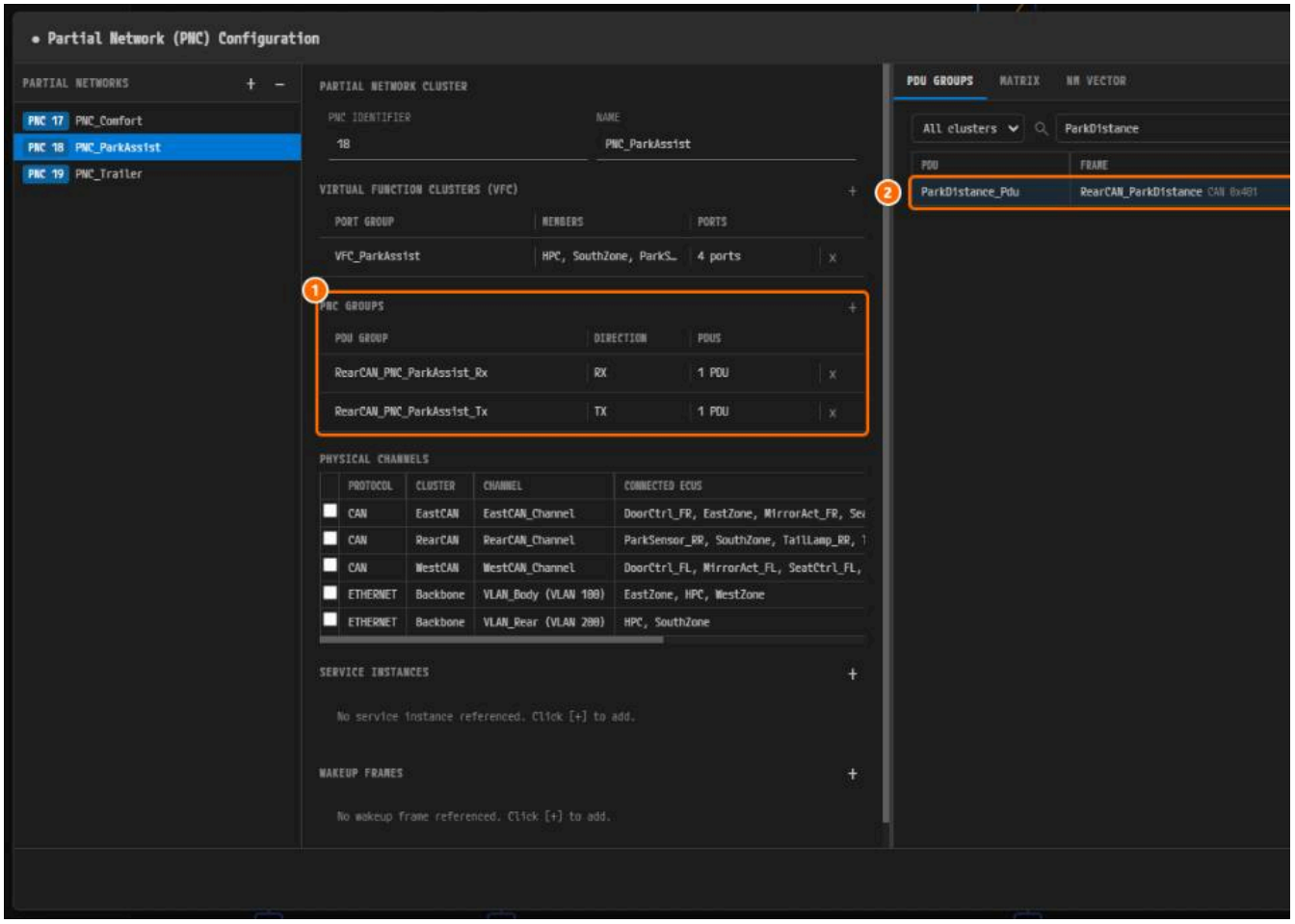


Figure 7-8. ① The bound group pair, ② `ParkDistance_Pdu` with `PNC_ParkAssist`.

The PDU enters both groups. `SouthZone` receives it, so it belongs in the Rx group only.

4. Drag the divider back. Expand `RearCAN_PNC_ParkAssist_Tx` and click the **×** on the `ParkDistance_Pdu` row of **CONTAINED PDUS**.

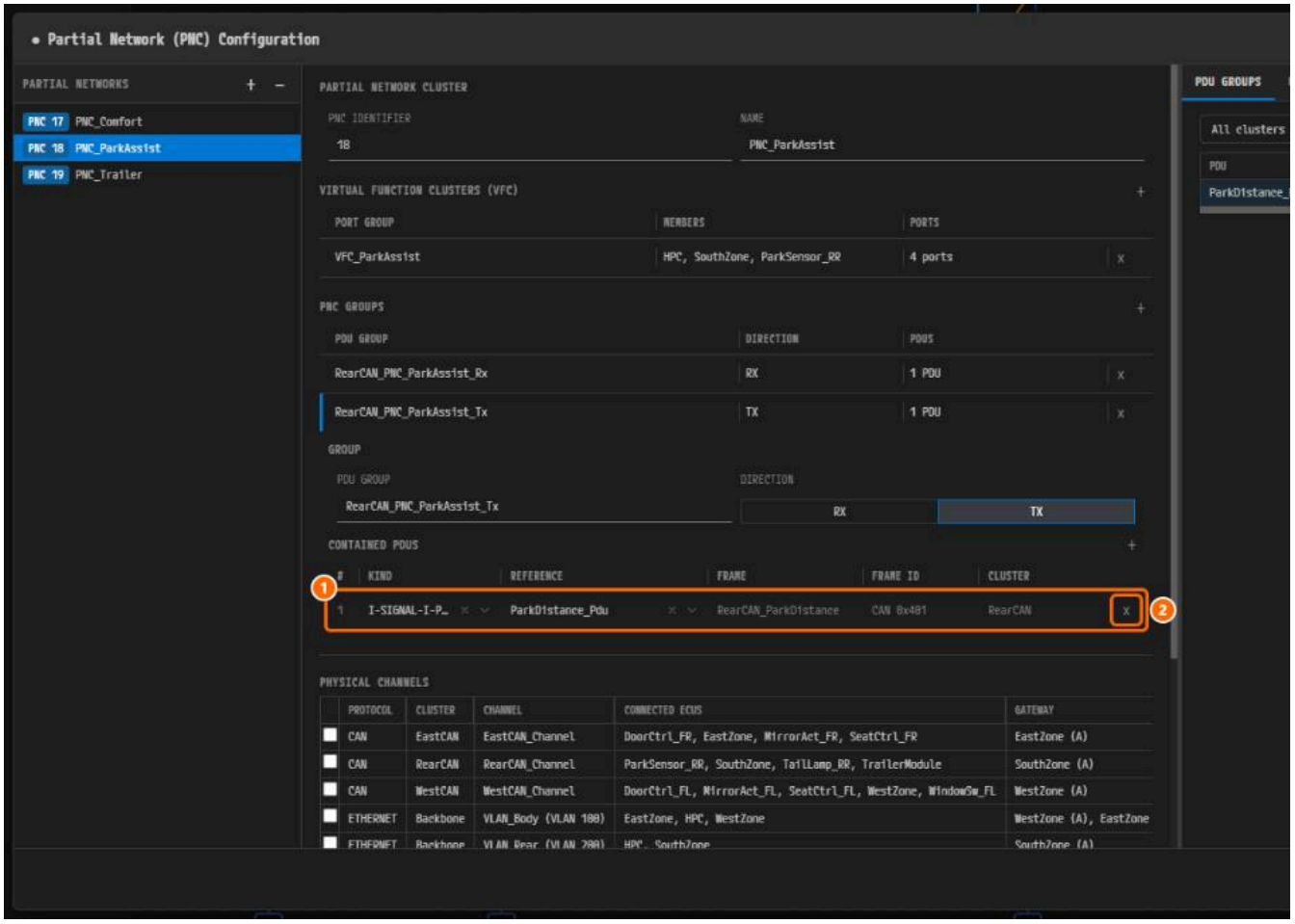


Figure 7-9. ① `ParkDistance_Pdu`, ② **×**.

The Tx group shows `0 PDUs`.

5. Collapse the row and clear the search field.

> **Caution.** The × at the right end of a group row deletes the group itself.

Selecting Physical Channels

1. In **PHYSICAL CHANNELS**, check **CAN** / `RearCAN_Channel` and **ETHERNET** / `VLAN_Rear (VLAN 200)`.

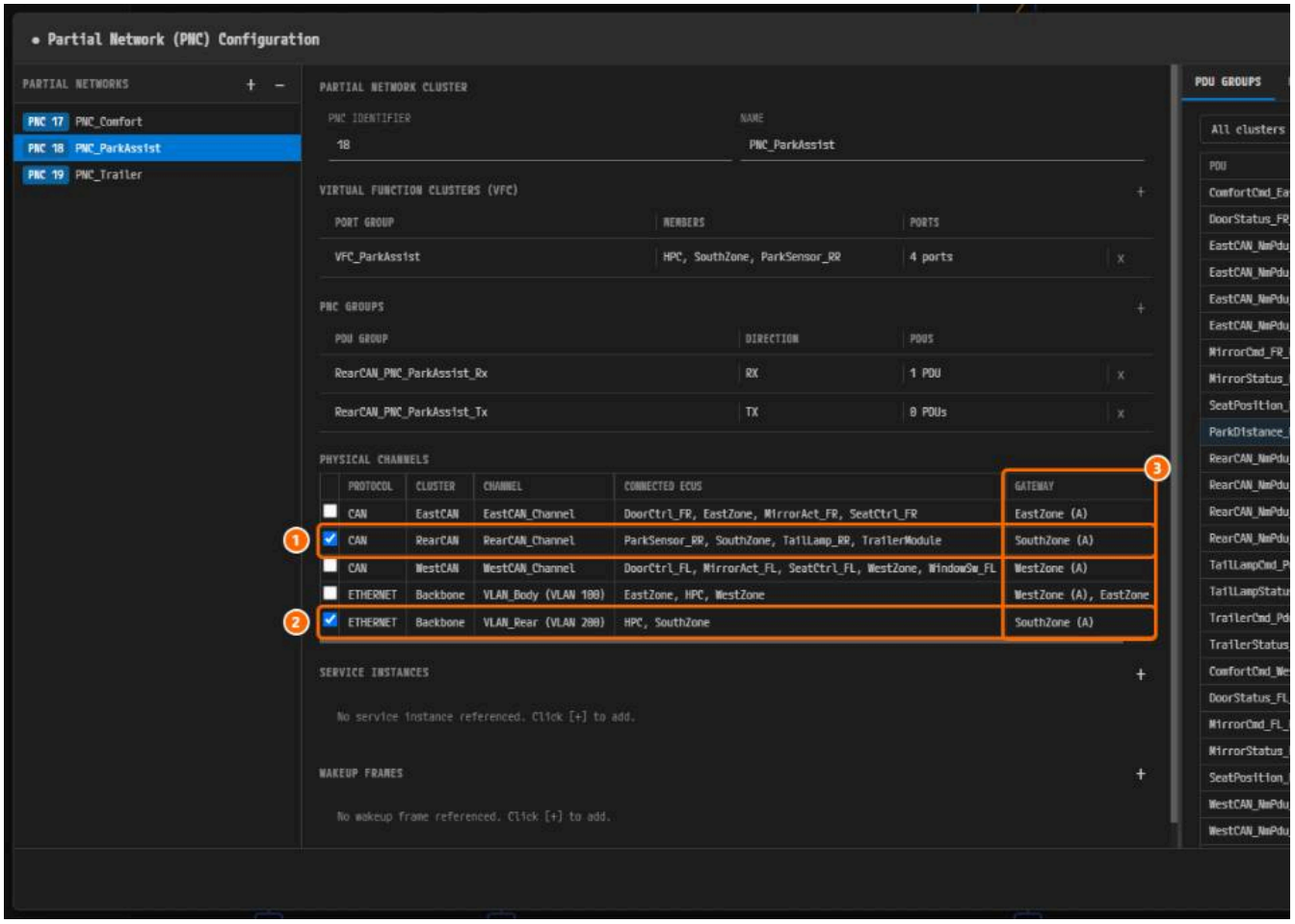


*Figure 7-10. ① `RearCAN_Channel`, ② `VLAN_Rear (VLAN 200)`, ③ **GATEWAY**, display only.*

Adding Service Instances

1. Click the **+** right of **SERVICE INSTANCES**, type `ParkDistance` and click **ParkDistance_Provider**.

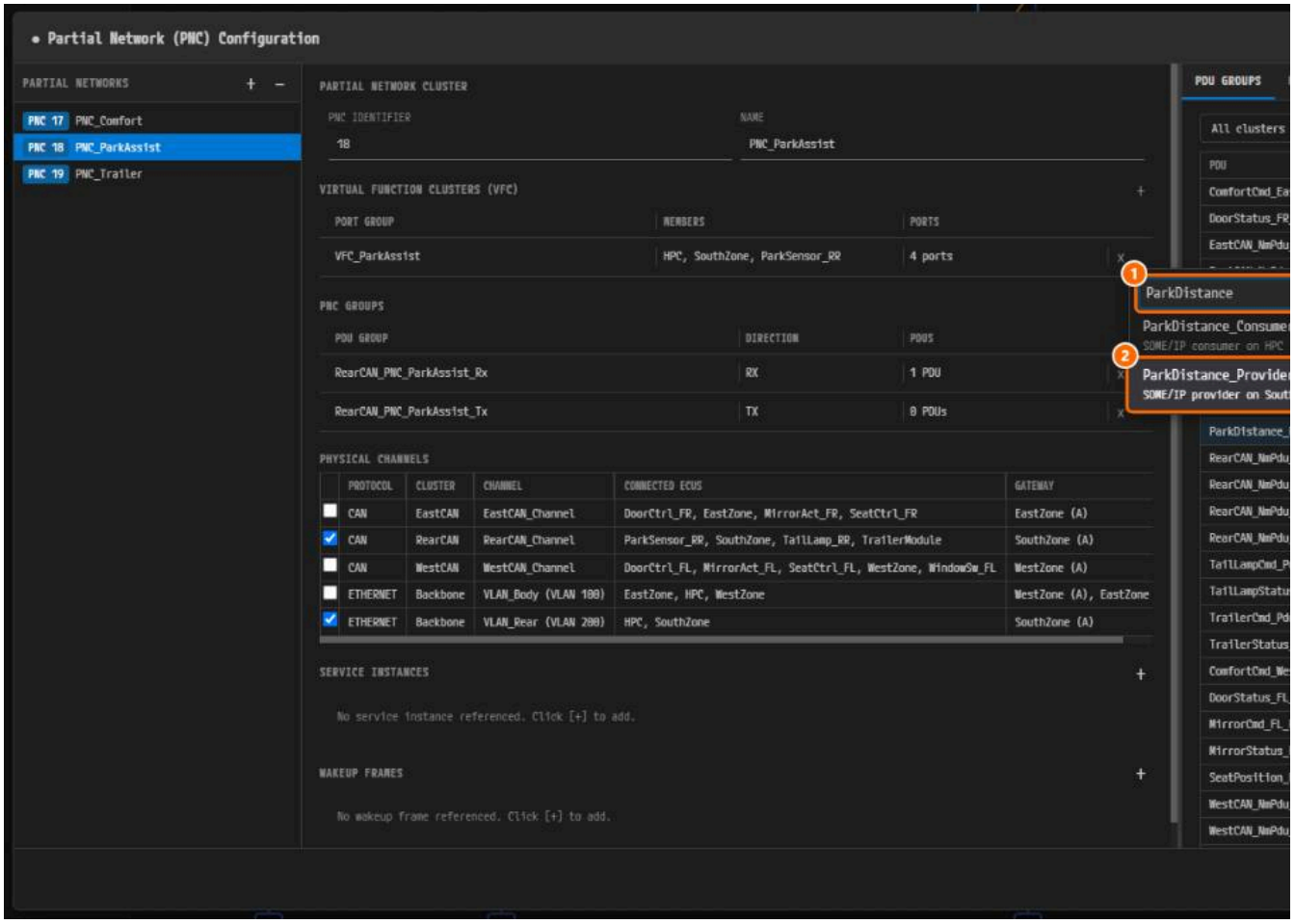


Figure 7-11. ① The search field, ② `ParkDistance_Provider`.

2. Add **ParkDistance_Consumer** the same way. Two cards appear. Click a card to see its details.

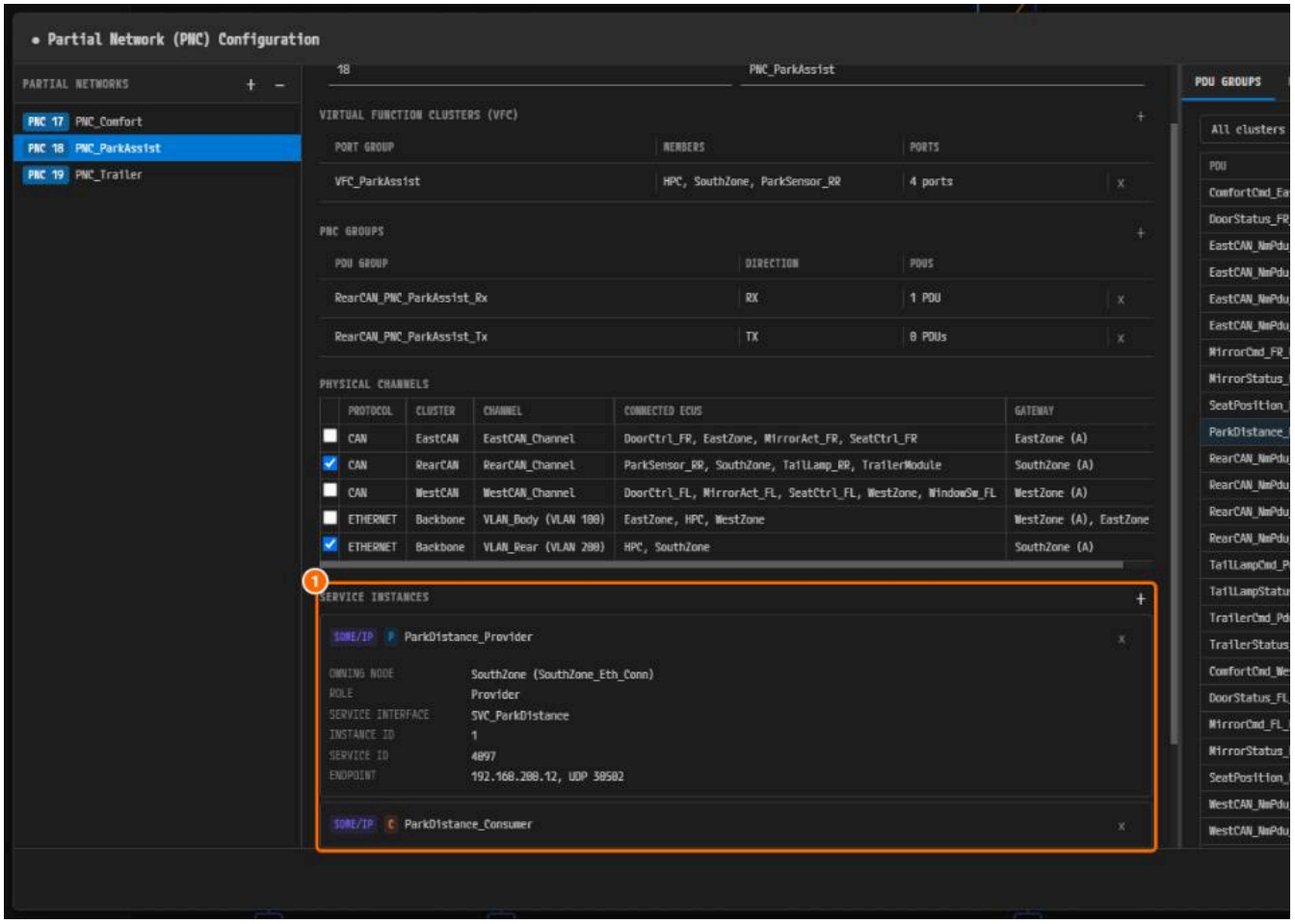


*Figure 7-12. ① **SERVICE INSTANCES**, with `ParkDistance_Provider` expanded.*

Saving

1. Press **Ctrl+S**. The ● before the title disappears.

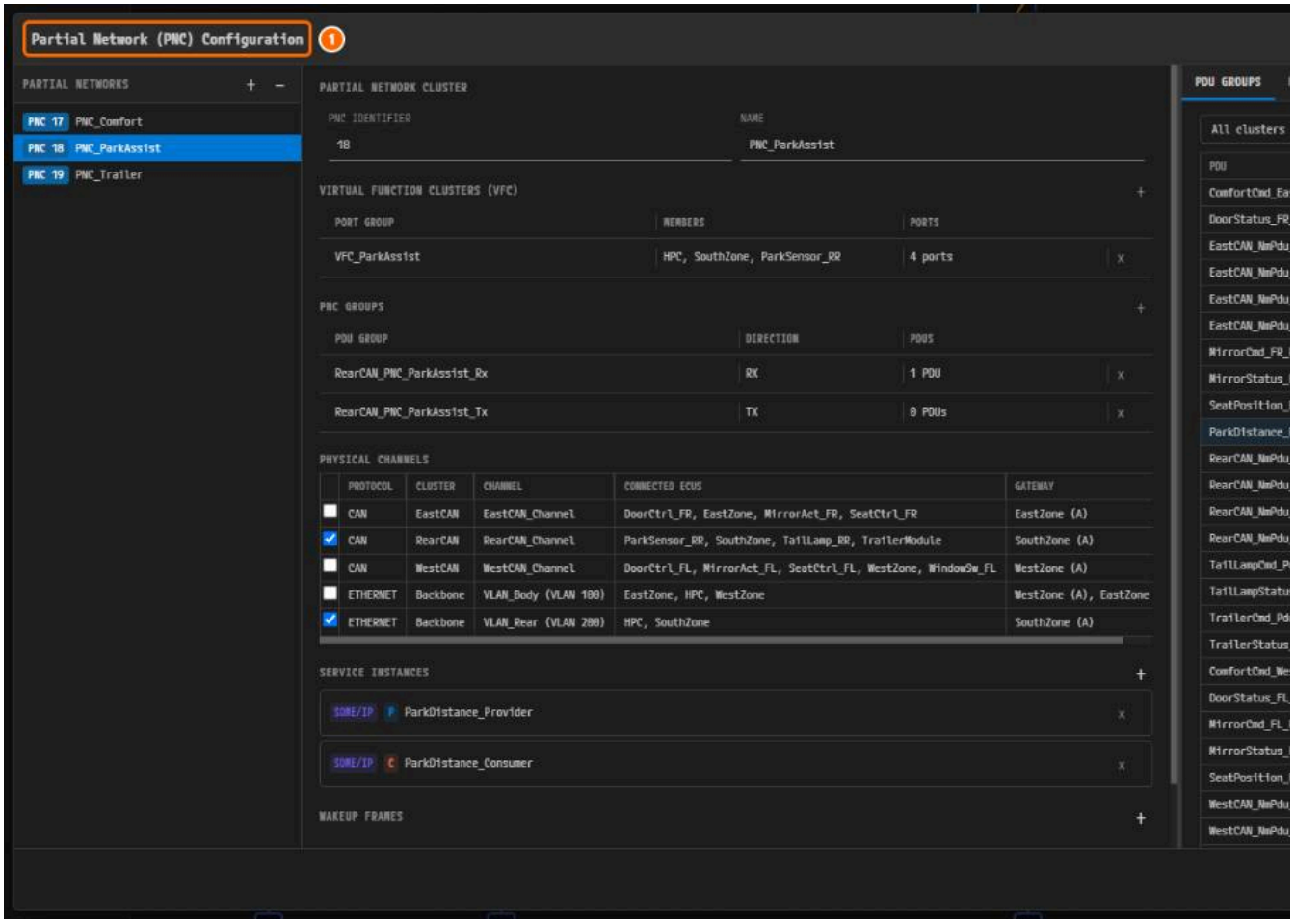


Figure 7-13. ① The title without ●.

Adding Wakeup Frames

`PNC_ParkAssist` has no wakeup frame. Enter one on `PNC_Trailer`.

1. In the list, click **PNC 19 PNC_Trailer**.
2. Click the **+** right of **WAKEUP FRAMES**, type `TrailerModule` and click **RearCAN_NmFrame_TrailerModule**.

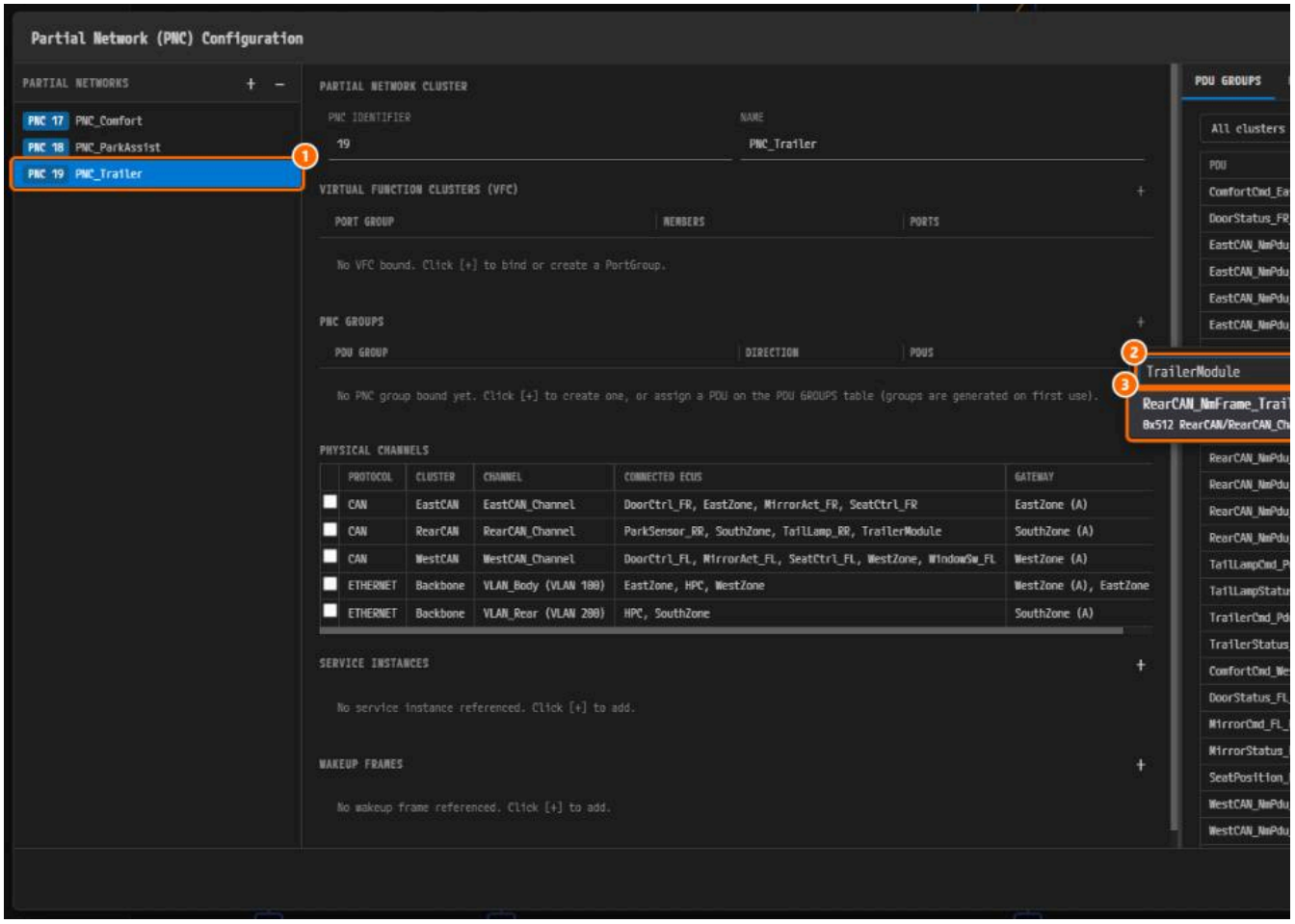


*Figure 7-14. ① **PNC 19 PNC_Trailer**, ② the search field, ③ `RearCAN_NmFrame_TrailerModule` (`0x512`). On your screen the list starts with a **PNC RX FR***

3. A **CAN** card appears. Click it to see its details.

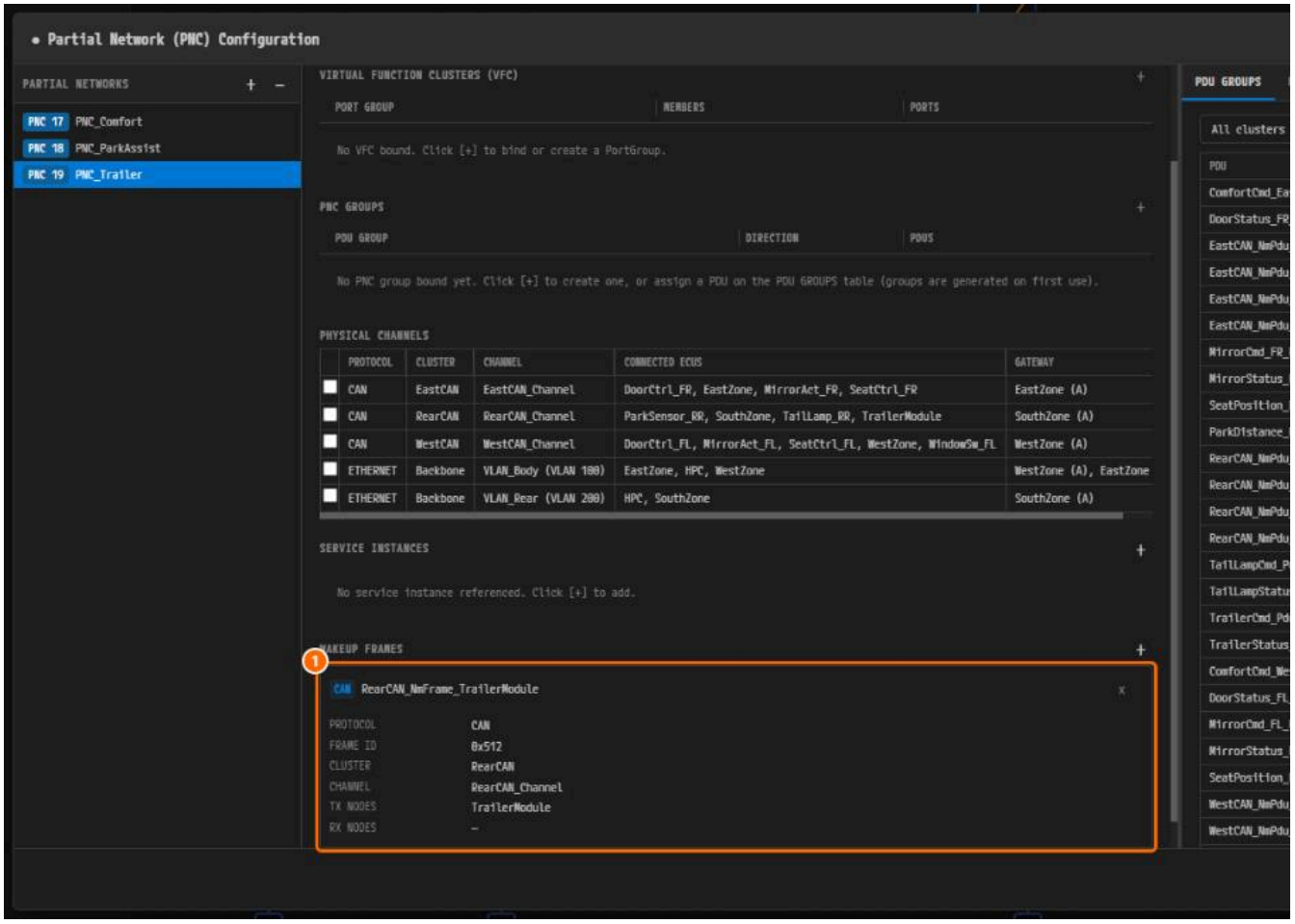


*Figure 7-15. ① **FRAME ID** `0x512`, **TX NODES** `TrailerModule`.*

4. Press **Ctrl+S**, then click **Close**.

Values for the Remaining PNCs

Bind `PNC_Comfort` and `PNC_Trailer` the same way from the table. **PNC GROUPS** is already filled, so only confirm it.

PNC	VFC	PNC GROUPS	PHYSICAL CHANNELS	SERVICE INSTANCES
PNC 17 `PNC_Comfort`	`VFC_Comfort`	`WestCAN_PNC_Comfort_Rx`, `WestCAN_PNC_Comfort_Tx`, `EastCAN_PNC_Comfort_Rx`, `EastCAN_PNC_Comfort_Tx`	`EastCAN_Channel`, `WestCAN_Channel`, `VLAN_Body (VLAN 100)`	`ComfortControl_Provider_West` (P), `ComfortControl_Provider_East` (P), `ComfortControl_Consumer` (C)
PNC 18 `PNC_ParkAssist`	`VFC_ParkAssist`	`RearCAN_PNC_ParkAssist_Rx`, `RearCAN_PNC_ParkAssist_Tx`	`RearCAN_Channel`, `VLAN_Rear (VLAN 200)`	`ParkDistance_Provider` (P), `ParkDistance_Consumer` (C)
PNC 19 `PNC_Trailer`	`VFC_Trailer`	`RearCAN_PNC_Trailer_Rx`, `RearCAN_PNC_Trailer_Tx`	`RearCAN_Channel`, `VLAN_Rear (VLAN 200)`	`VehicleState_Provider` (P), `VehicleState_Consumer` (C)

After binding a VFC, check its members in **SWC HIERARCHY**, and their ports from the **OUTER PORTS** in VFB and VFC (PortGroup).

VFC	MEMBERS	PORTS
`VFC_Comfort`	`HPC`, `WestZone`, `EastZone`, `DoorCtrl_FL`, `SeatCtrl_FL`, `MirrorAct_FL`, `WindowSw_FL`, `DoorCtrl_FR`, `SeatCtrl_FR`, `MirrorAct_FR` (10)	`28 ports`
`VFC_ParkAssist`	`HPC`, `SouthZone`, `ParkSensor_RR` (3)	`4 ports`
`VFC_Trailer`	`HPC`, `SouthZone`, `TrailerModule`, `TailLamp_RR` (4)	`8 ports`

> **Note.** If you skipped Chapter 4, assign **PNC_Trailer** in **PDU GROUPS** to `TrailerStatus_Pdu`, `TailLampStatus_Pdu`, `TrailerCmd_Pdu` and `TailLampCmd_`
> `TailLampStatus_Pdu` from the Tx group and `TrailerCmd_Pdu` and `TailLampCmd_Pdu` from the Rx group. Do not use this method on `PNC_Comfort`, wh

The Finished Result

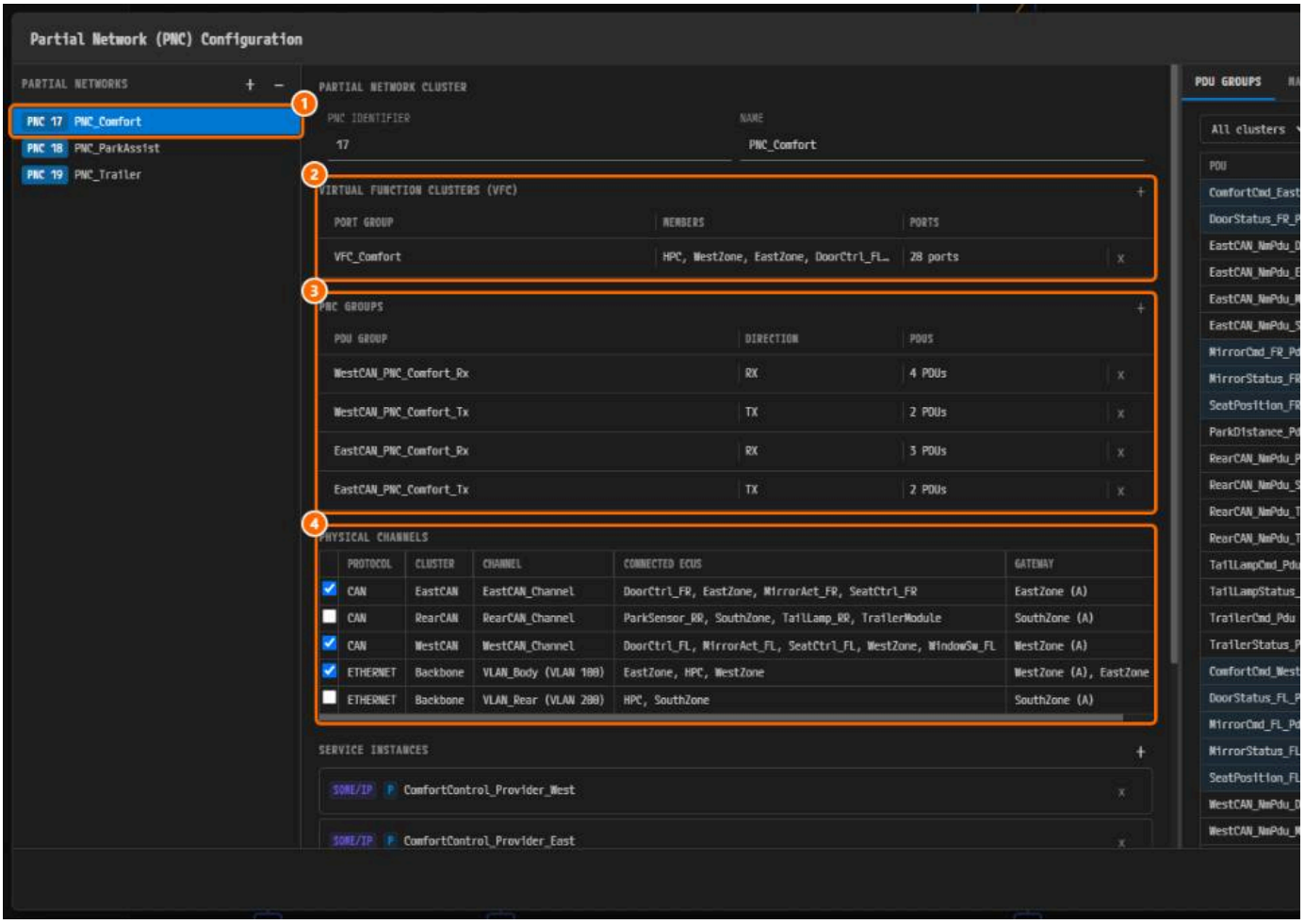


*Figure 7-16. `PNC_Comfort`, upper part. ① **PNC 17 PNC_Comfort**, ② `VFC_Comfort`, ③ four **PNC GROUPS**, ④ three **PHYSICAL CHANNELS**.*

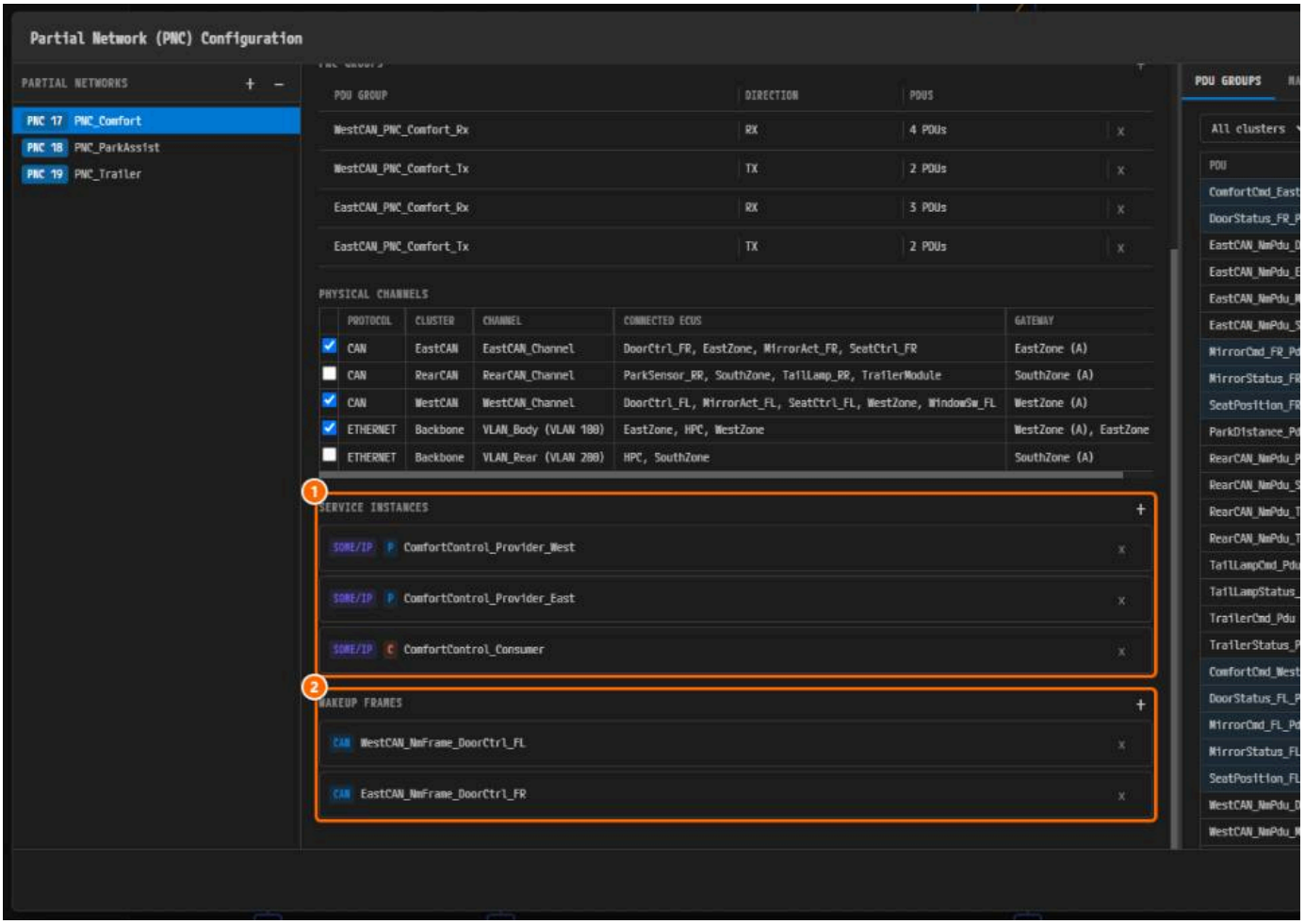


*Figure 7-17. `PNC_Comfort`, lower part. ① **SERVICE INSTANCES**, ② **WAKEUP FRAMES**.*

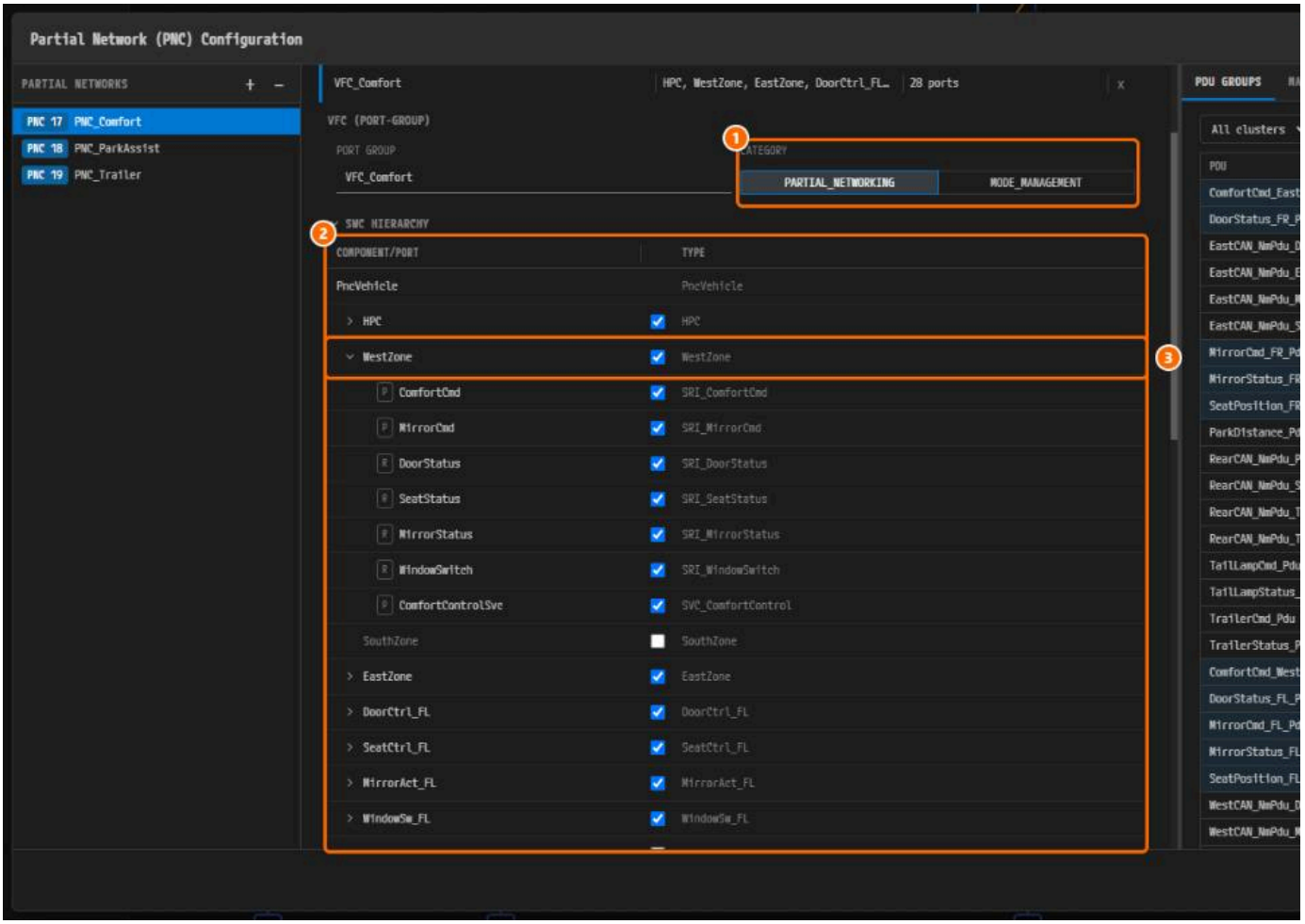


*Figure 7-18. `VFC_Comfort` expanded. ① **CATEGORY**, ② **SWC HIERARCHY**, ③ `WestZone` with all seven ports checked.*

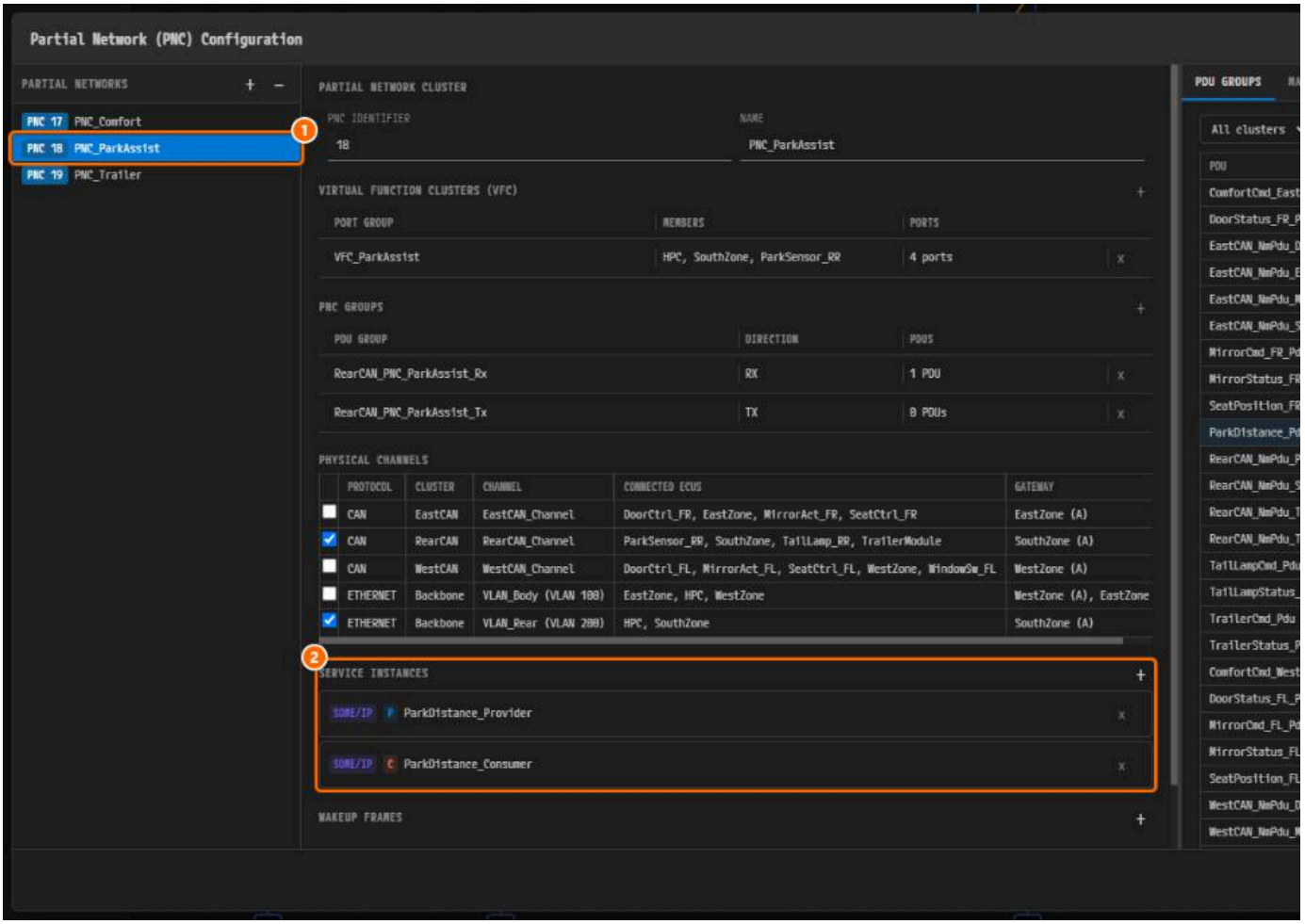


*Figure 7-19. ① **PNC 18 PNC_ParkAssist**, ② **SERVICE INSTANCES**.*

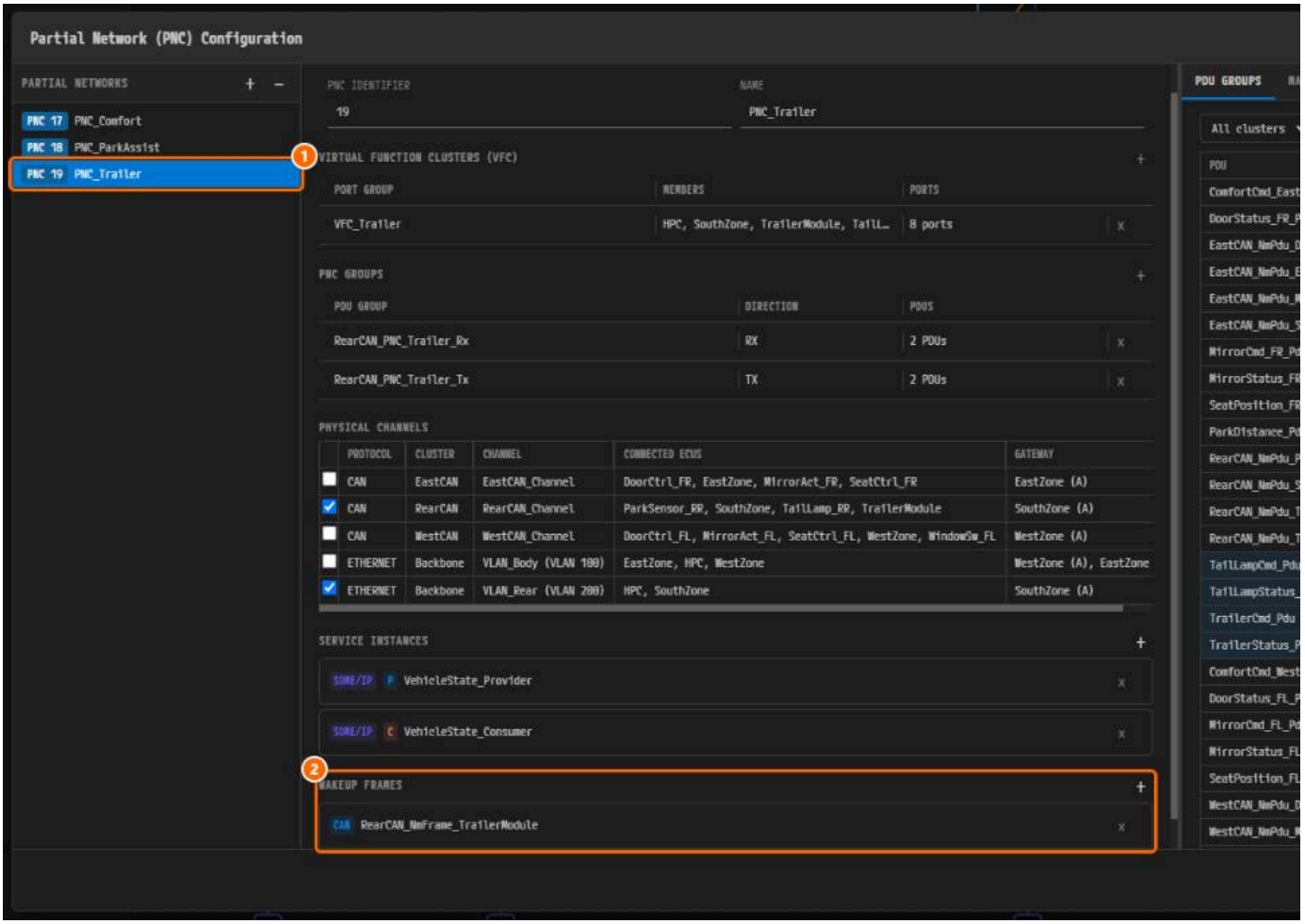


Figure 7-20. `PNC_Trailer`*, scrolled down.* ① ***PNC 19 PNC_Trailer***, ② ***WAKEUP FRAMES***.

Why Is It Designed This Way?

- **One screen per PNC.** All references of a PNC are in one place, so a missing one shows at once.
- **You check the physical channels.** They are not derived from the groups or PDUs.
- `PNC_ParkAssist` **has no wakeup frame.** A FunctionGroup state on the HPC requests it (see AP Machine — NM Handles and FunctionGroup).
- `PNC_Trailer` **wakes up on** `RearCAN_NmFrame_TrailerModule`. `0x512` lies inside the wakeup filter block `0x500`..`0x5FF`.

Checklist

- [ ] `PNC_ParkAssist` has `VFC_ParkAssist` (`4 ports`), the two groups (**RX** `1 PDU`, **TX** `0 PDUs`) and the two service instances.
- [ ] Only `RearCAN_Channel` and `VLAN_Rear (VLAN 200)` are checked in its **PHYSICAL CHANNELS**.
- [ ] **WAKEUP FRAMES** of `PNC_Trailer` contains `RearCAN_NmFrame_TrailerModule`.
- [ ] `PNC_Comfort` and `PNC_Trailer` match the table in Values for the Remaining PNCs.
- [ ] There is no ● before the dialog title.

Next Step

In Reviewing PDU GROUPS, MATRIX and NM VECTOR you cross-check the design with the three viewer tabs.

Reviewing PDU Groups, Matrix and NM Vector

What You Will Do

In this chapter you open the three viewer tabs of the PNC dialog, **PDU GROUPS**, **MATRIX** and **NM VECTOR**, and cross-check the PNC design. You only read; y

Before You Begin

- Start after finishing The PNC Dialog — Binding Each PNC, with every PNC bound as in Values for the Remaining PNCs.
- The NM values come from NM Settings. If yours differ, compare them with All NM Values.

Item	Value
PNC	`PNC_Comfort` 17, `PNC_ParkAssist` 18, `PNC_Trailer` 19
PNC VECTOR OFFSET / PNC VECTOR LENGTH	`2` / `6` (Byte 2 to 7 of the NM PDU, bits 16 to 63)
NM clusters	`NmCluster_EastCAN`, `NmCluster_RearCAN`, `NmCluster_WestCAN` (CanNm), `NmCluster_VLAN_Body`, `NmCluster_VLAN_Rear` (UdpNr

Opening the Viewer Tabs

1. Open the PNC dialog from the System editor (tab `PNC:System`) as in the previous chapter.
2. Click **PNC 19 PNC_Trailer**.

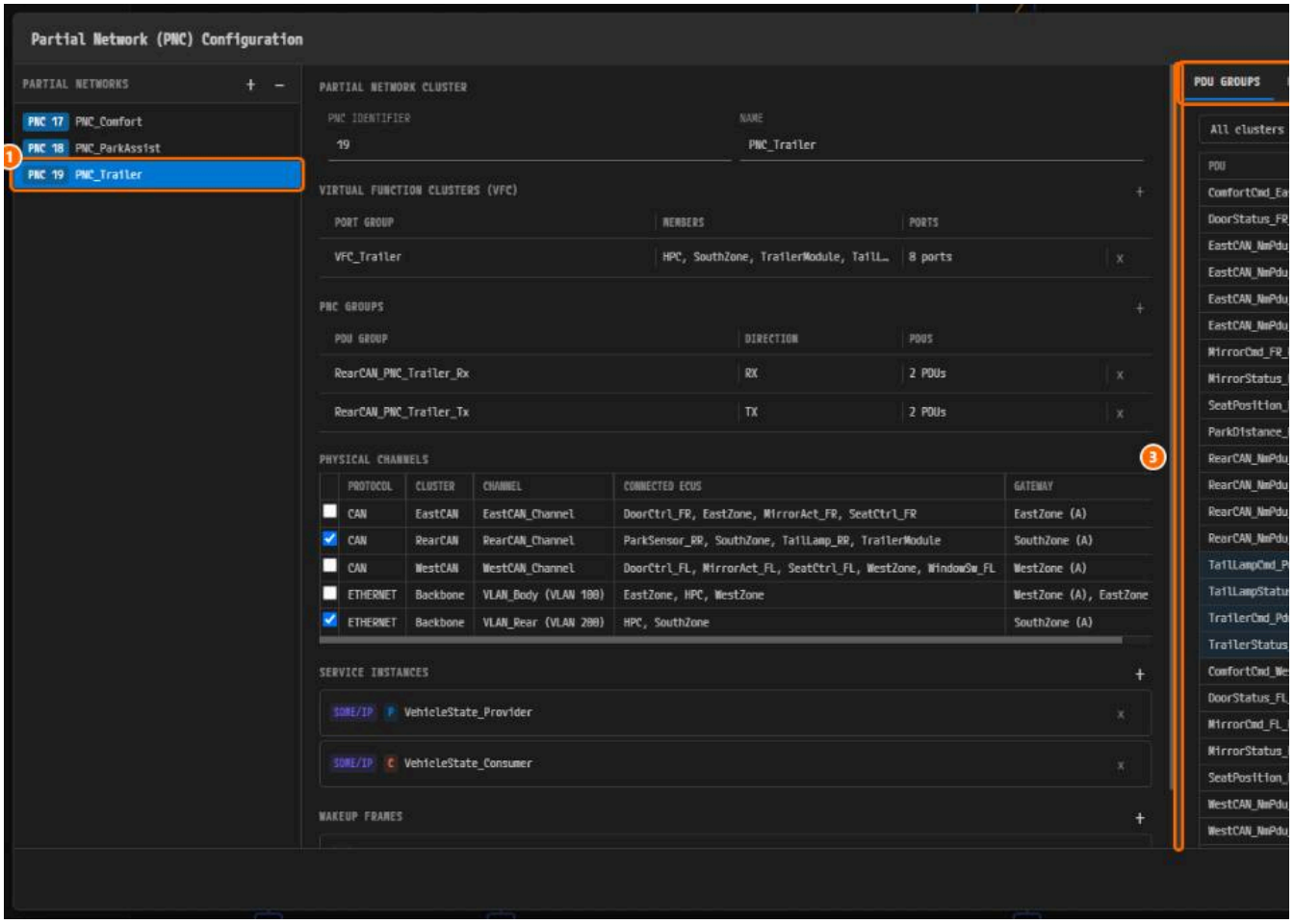


*Figure 8-1. ① **PNC 19 PNC_Trailer**, ② the viewer tabs, ③ the splitter.*

3. Drag the splitter about 340 px to the left, until **PDU**, **FRAME**, **CLUSTER** and **PNC MEMBERSHIP** are all visible.

Reading PDU GROUPS

Each row is one PDU. **PNC MEMBERSHIP** shows the PNC whose bound I-PDU group holds the PDU.

1. Check that the four rows of `PNC_Trailer` are highlighted and read `PNC_Trailer` in **PNC MEMBERSHIP**.

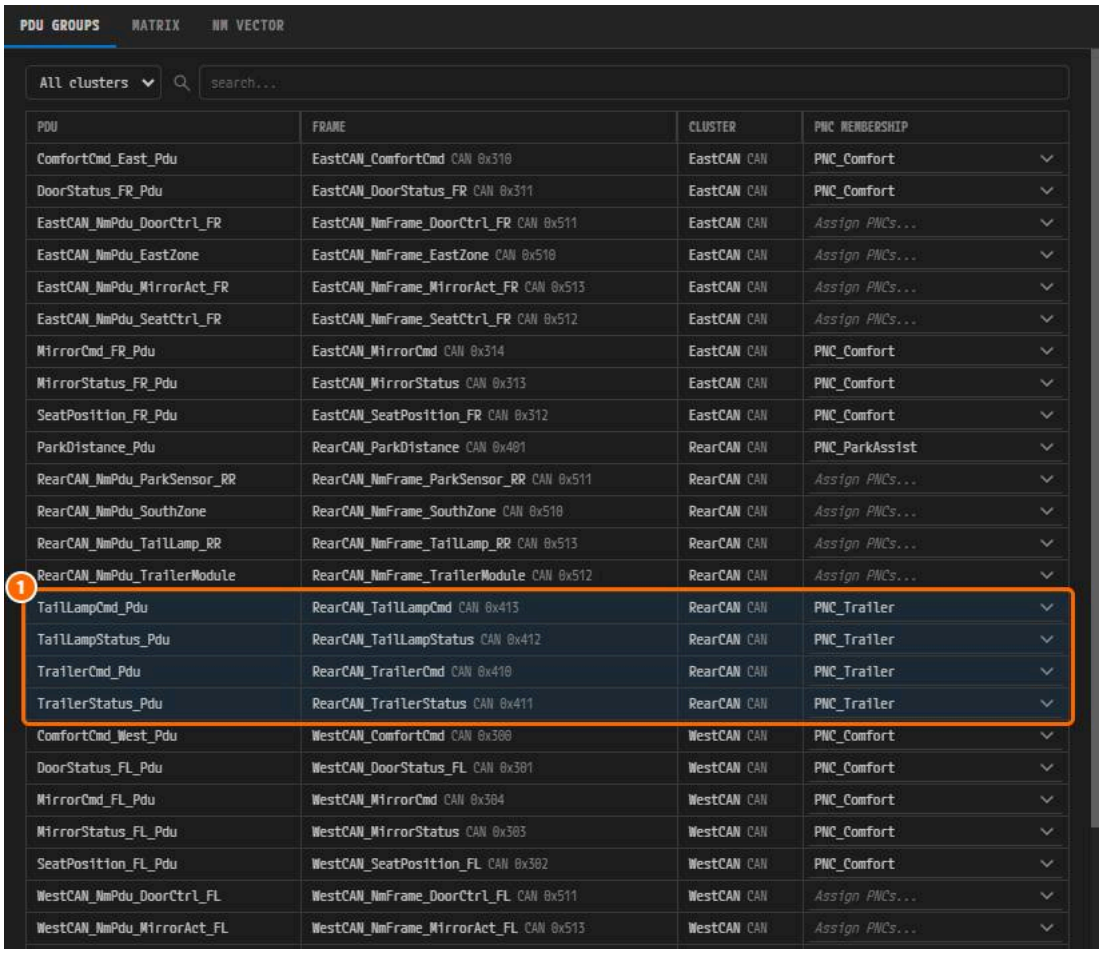


Figure 8-2. ① The four rows of `PNC_Trailer`.

The NM PDUs (`<cluster>_NmPdu_<ECU>`) show "Assign PNCs...". This is correct: NM PDUs are in no PNC's group.

> **Note.** Your own project may show fewer rows. See Appendix — Differences Between Your Project and the Finished Example.

Narrowing by Cluster

1. Set the cluster filter (**All clusters**) to **RearCAN**. Nine rows remain.

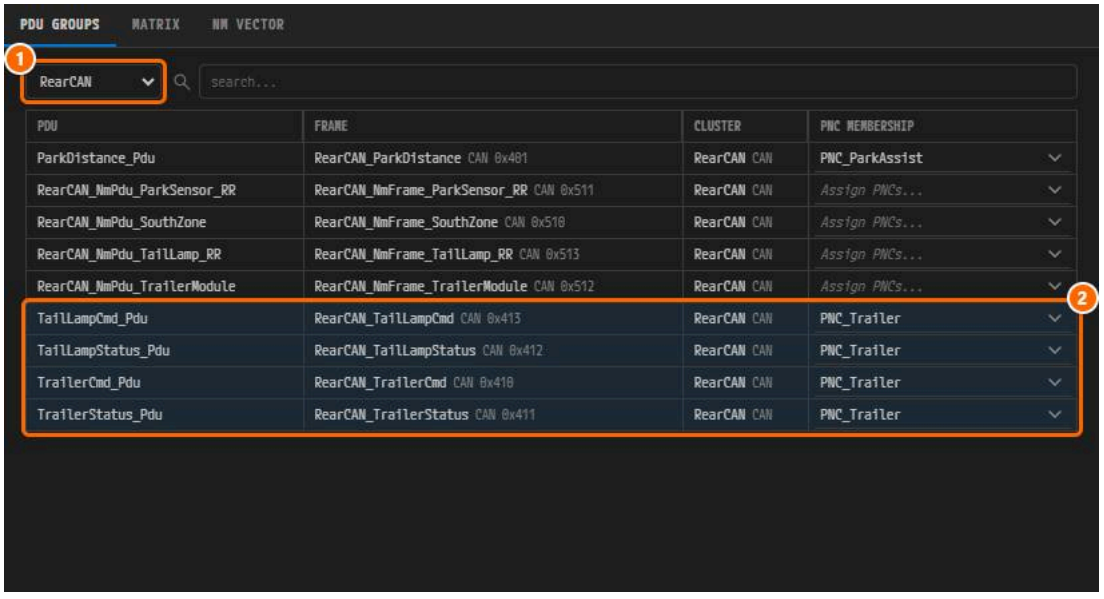


*Figure 8-3. ① **RearCAN**, ② the four rows of `PNC_Trailer`.*

2. Set the filter back to **All clusters**.

Searching by Name

1. Type `Trailer` in the search field. Three rows remain.

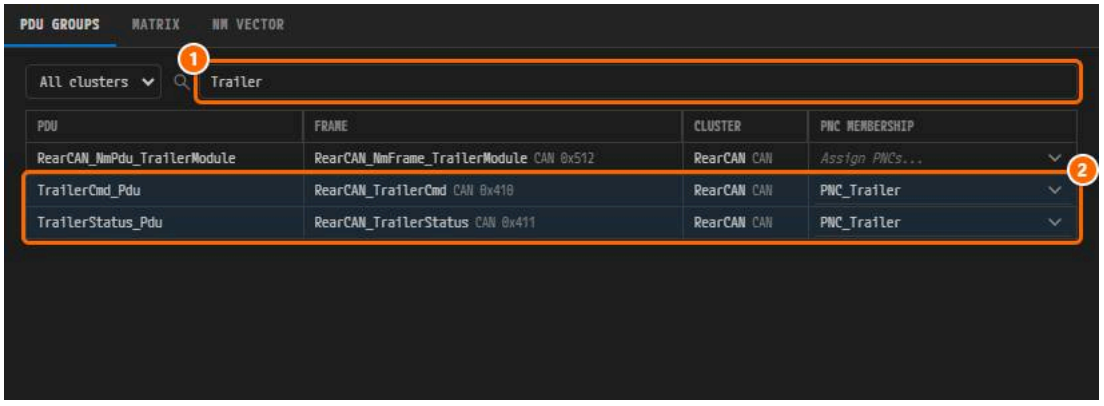


Figure 8-4. ① `Trailer`, ② `TrailerCmd_Pdu` and `TrailerStatus_Pdu`.

The `TailLamp*` PDUs of `PNC_Trailer` do not appear, because their names do not contain `Trailer`.

2. Clear the search field.

Reading MATRIX

The rows are PNCs and the columns are physical channels.

1. Drag the splitter back to the right and click the **MATRIX** tab.

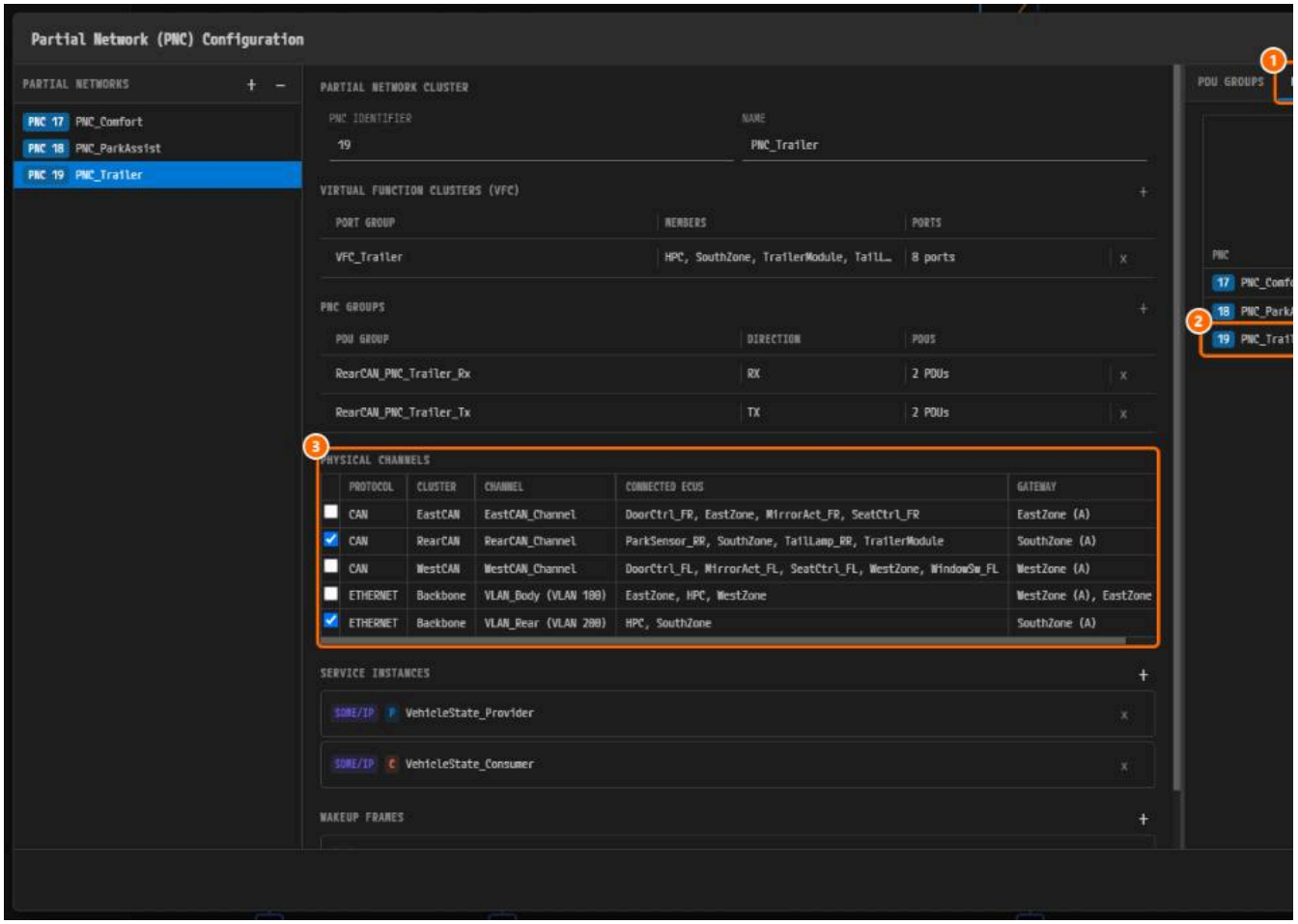


*Figure 8-5. ① **MATRIX**, ② the `PNC_Trailer` row, ③ **PHYSICAL CHANNELS**, with the same values as ②.*

2. Compare the matrix with the table.



Figure 8-6. ① `PNC_Comfort`, ② `PNC_ParkAssist` and `PNC_Trailer`.

PNC	EastCAN / EastCAN_Channel	RearCAN / RearCAN_Channel	WestCAN / WestCAN_Channel	Backbone / VLAN_Body	Backbone / VLAN_R
17 `PNC_Comfort`	✓		✓	✓	
18 `PNC_ParkAssist`		✓			✓
19 `PNC_Trailer`		✓			✓

> **Caution.** Clicking a matrix cell changes the physical channel reference at once. Do not click. If you do, click the same cell again or press **Ctrl+Z**.

Reading NM VECTOR

PNC identifier N sits in the NM PDU at Byte `N ÷ 8 quotient`, bit `N ÷ 8 remainder`.

1. Click **PNC 17 PNC_Comfort**, widen the viewer again and click the **NM VECTOR** tab.

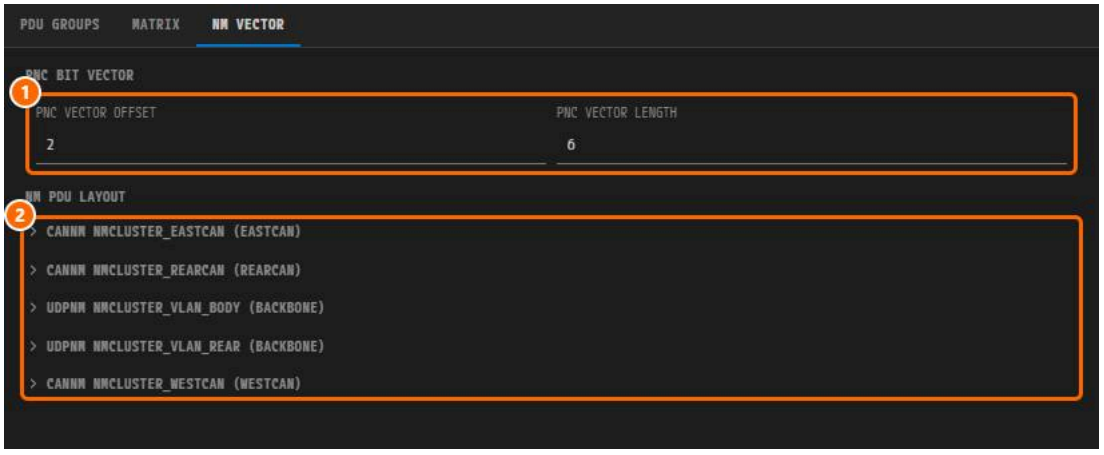


*Figure 8-7. ① **PNC VECTOR OFFSET** `2` and **PNC VECTOR LENGTH** `6`, ② the five NM cluster headers.*

The two values apply to every NM cluster. You entered them under Setting the PNC Bit Vector.

2. Click the last header, **CANNM NMCLUSTER_WESTCAN (WESTCAN)**, to expand it.

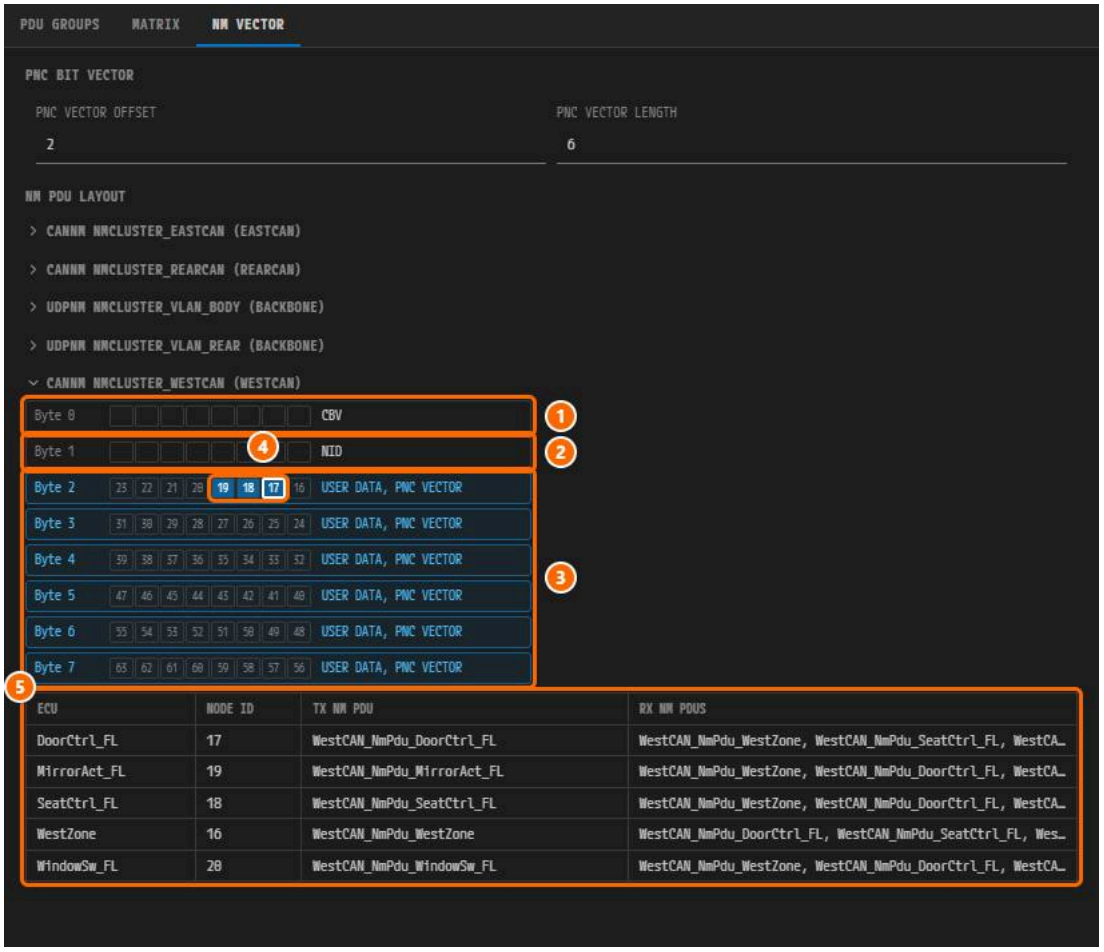


*Figure 8-8. ① Byte 0 **CBV**, ② Byte 1 **NID**, ③ Byte 2 to 7 **USER DATA, PNC VECTOR**, ④ the filled cells `19`, `18`, `17`; the selected PNC 17 has a white border,*

3. Check the node table of `NmCluster_WestCAN`.

ECU	NODE ID	TX NM PDU
`DoorCtrl_FL`	17	`WestCAN_NmPdu_DoorCtrl_FL`
`MirrorAct_FL`	19	`WestCAN_NmPdu_MirrorAct_FL`
`SeatCtrl_FL`	18	`WestCAN_NmPdu_SeatCtrl_FL`
`WestZone`	16	`WestCAN_NmPdu_WestZone`
`WindowSw_FL`	20	`WestCAN_NmPdu_WindowSw_FL`

4. Expand the other NM clusters the same way. All have the same byte layout. The table lists the ECUs in NODE ID order.

NM cluster	ECU (NODE ID)
`NmCluster_EastCAN` (EastCAN)	`EastZone` (16), `DoorCtrl_FR` (17), `SeatCtrl_FR` (18), `MirrorAct_FR` (19)
`NmCluster_RearCAN` (RearCAN)	`SouthZone` (16), `ParkSensor_RR` (17), `TrailerModule` (18), `TailLamp_RR` (19)
`NmCluster_VLAN_Body` (Backbone)	`HPC` (0), `WestZone` (1), `EastZone` (2), `SouthZone` (3), `HPC` (4)
`NmCluster_VLAN_Rear` (Backbone)	`HPC` (0), `SouthZone` (1), `WestZone` (2), `EastZone` (3), `HPC` (4)

> **Note.** Both UdpNm clusters list a node for every connector on `Backbone`. Nodes on the other VLAN have empty NM PDU cells, and `HPC` appears tw

5. Click **Close**. The dialog closes without asking to save.

How Filter Masks Relate to NM VECTOR

The **PNC FILTER ARRAY MASKS** you entered under Calculating the Filter Mask hold only Byte 2 to 7 of the NM PDU. Byte 0 of the mask is Byte 2 of the NM P

```
byteIndex = quotient of pncId ÷ 8 - pncVectorOffset    # the byte in the mask array
bit       = remainder of pncId ÷ 8                     # the bit within that byte
value     = 2 ^ bit
```

PNC	Position in the NM PDU (tooltip)	Position in the mask array	Value
17 `PNC_Comfort`	byte 2, bit 1	Byte 0, bit 1 (17 ÷ 8 = 2, 2 − 2 = 0)	`0x02`
18 `PNC_ParkAssist`	byte 2, bit 2	Byte 0, bit 2	`0x04`
19 `PNC_Trailer`	byte 2, bit 3	Byte 0, bit 3	`0x08`

- A connector that passes only PNC 17 (`WestZone_CanFd_Conn`) has the mask `02 00 00 00 00 00`.
- A connector that passes PNC 18 and 19 (`SouthZone_CanFd_Conn`) ORs them: `0x04` | `0x08` = `0x0C`, that is `0C 00 00 00 00 00`.
- The mask length must equal **PNC VECTOR LENGTH**, 6 (`constr_3686`).

Why Is It Designed This Way?

- **Three views of one design.** PDU to PNC, PNC to channel, PNC to NM PDU bit: a gap in one shows in another.
- **The PNC identifier is a bit number.** It fixes the position in the NM PDU and the filter mask value.

Checklist

- [ ] In **PDU GROUPS**, `PNC_Trailer` highlights `TailLampCmd_Pdu`, `TailLampStatus_Pdu`, `TrailerCmd_Pdu` and `TrailerStatus_Pdu`.
- [ ] **MATRIX** matches the table in Reading MATRIX.
- [ ] In **NM VECTOR**, the offset is `2`, the length is `6`, and cells 17, 18 and 19 of Byte 2 are filled.
- [ ] **Close** does not ask whether to save.

Next Step

In AP Machines — NM Handles and FunctionGroup you configure NM handles and FunctionGroups on the HPC machine.

AP Machine: NM Network Handles and Function Groups

What You Will Do

The HPC is an AP machine, not a PNC gateway. It requests and releases PNCs through NM network handles. In this chapter you create one FunctionGroup, then create the NM network handles and map them to the FunctionGroup states.

Before You Begin

- Start after finishing Reviewing PDU GROUPS, MATRIX and NM VECTOR.
- You build `ParkAssistFG` and `NH_ParkAssist` step by step and create the rest from the tables.
- PAIO adds `MachineFG` itself. Do not create or delete it.

Item	Value
FunctionGroup **NAME**	`ParkAssistFG`
FunctionGroup **DESCRIPTION**	`Park-assist function group driven by PNC_ParkAssist`
FunctionGroup **GROUP NAME / INITIAL MODE**	`ParkAssistState` / `Off`
FunctionGroup **MODE DECLARATION**	`Off`, `Active`
MAX ACTION LIST DURATION	`1` s
Handle **NAME**	`NH_ParkAssist`
Handle **VLANS (ETHERNET CONNECTORS)**	`HPC_Eth_Rear_Conn`
Handle **PARTIAL NETWORKS**	`PNC_ParkAssist (PNC 18)`
Mapping 1	`ParkAssistFG:Active` → `FUNCTION-GROUP-STATE-TO-NM-HANDLE`
Mapping 2	`ParkAssistFG:Off` → `NM-HANDLE-INACTIVE-TO-FUNCTION-GROUP-STATE`

Opening the Machine Configuration Dialog

1. Under **DESIGN** in the **EDITOR** view, click **VFB**. The tab **PNC:VFB** shows the root composition `PncVehicle`.
2. Click the light blue machine icon to the right of the **HPC [HPC]** header. Do not click the square icon at the far right.

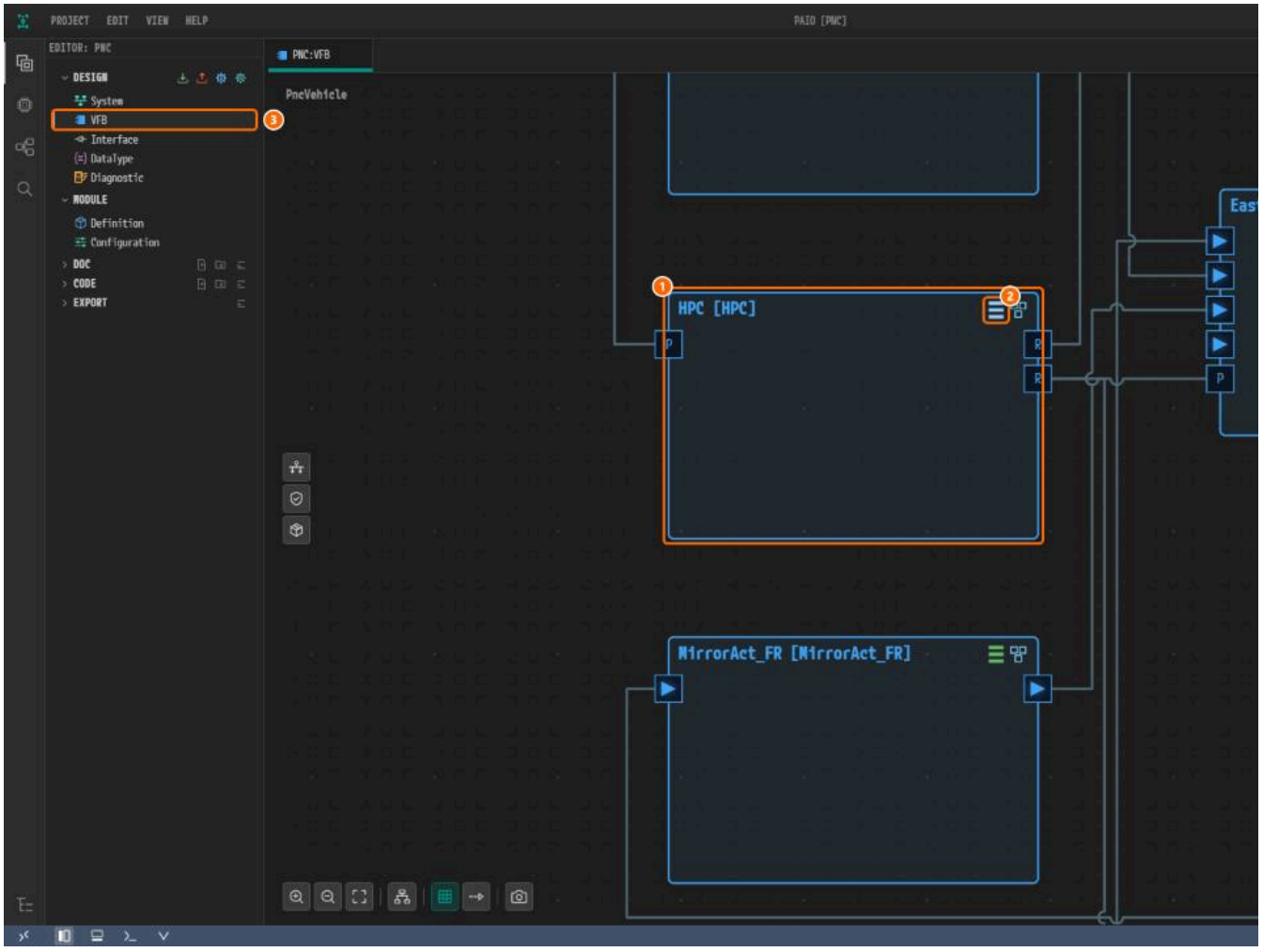


*Figure 9-1. ① **HPC [HPC]**, ② the machine icon, ③ **VFB**.*



Figure 9-2. ① The machine icon.

3. The **Machine Configuration [HPC]** dialog opens on the **MACHINE** tab.

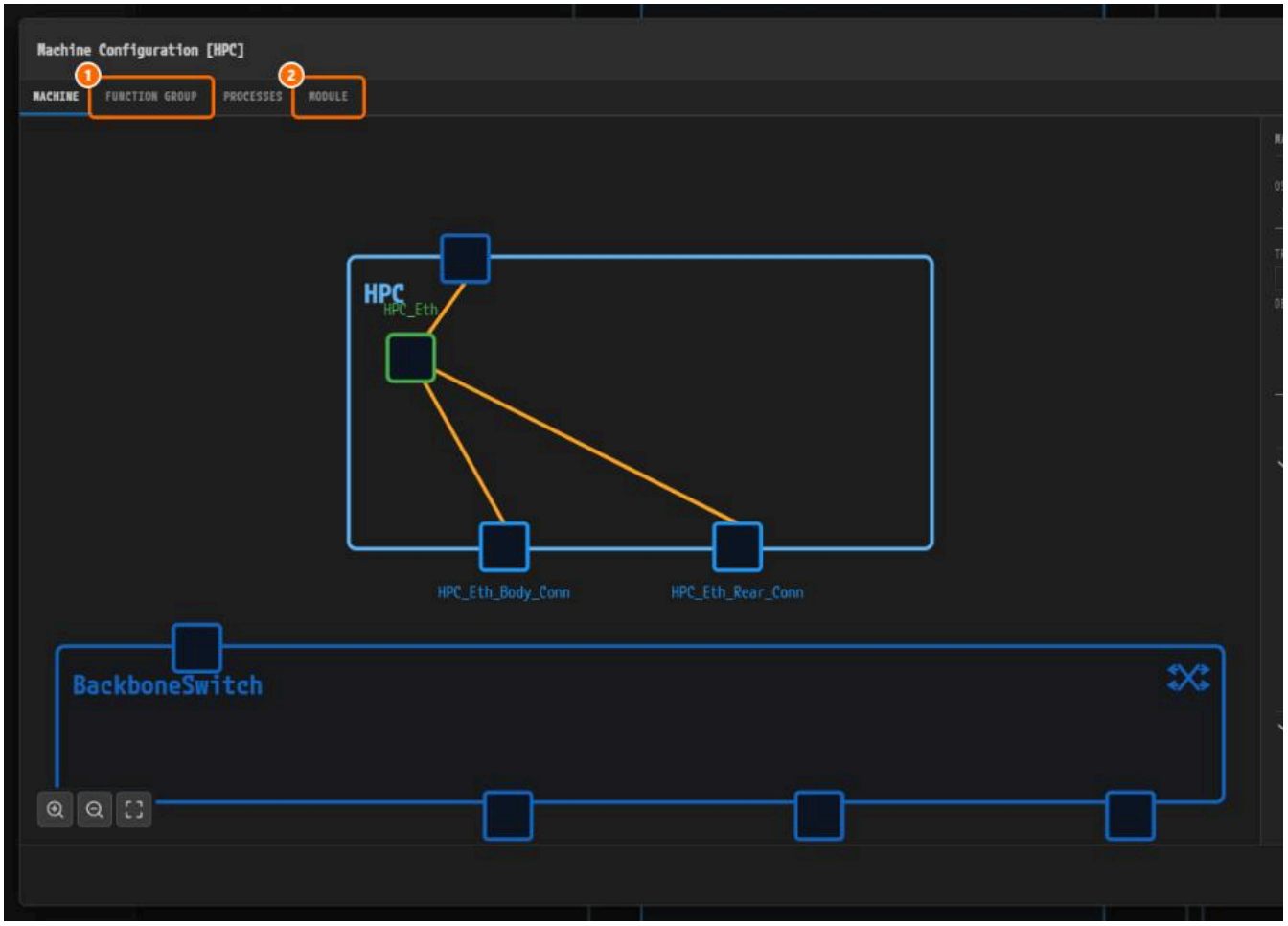


*Figure 9-3. ① **FUNCTION GROUP**, ② **MODULE**. Do not click **PARA Platform Import** in this chapter.*

Reviewing the FunctionGroup Screen

An NM handle references FunctionGroup states, so you create the FunctionGroups first.

1. Click the **FUNCTION GROUP** tab, then click `MachineFG`. Leave its values as they are.

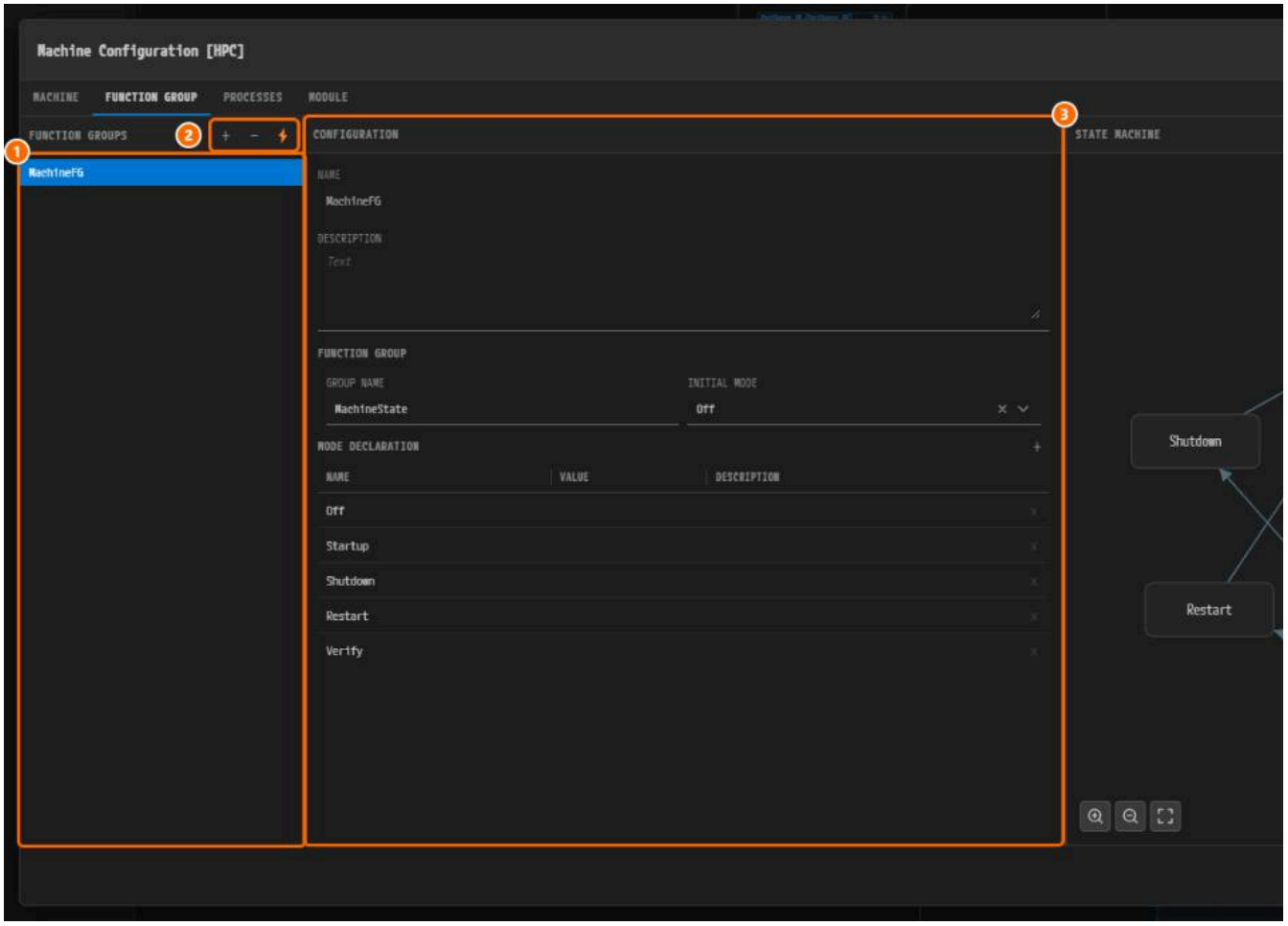


*Figure 9-4. ① **FUNCTION GROUPS**, ② **+**, **−** and ⚡, ③ **CONFIGURATION** of `MachineFG`. Do not click ⚡ in this chapter.*

Creating a FunctionGroup

1. Click **+** (**Add Function Group**). `FunctionGroup_2` is added and selected.

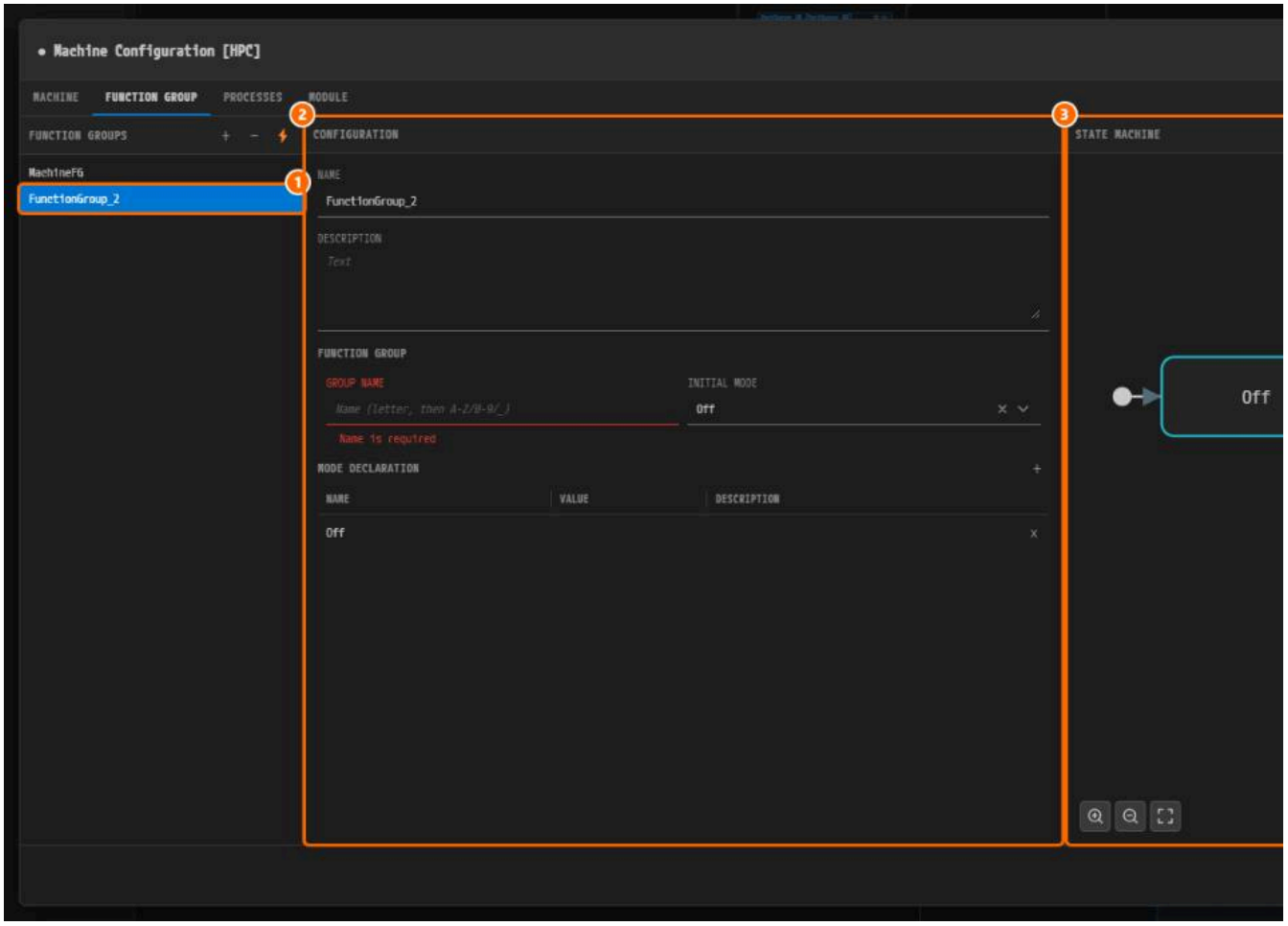


*Figure 9-5. ① `FunctionGroup_2`, ② **CONFIGURATION**, ③ **STATE MACHINE**.*

2. Enter `ParkAssistFG` in **NAME** and press `Tab`. The list label updates later.
3. Enter `Park-assist function group driven by PNC_ParkAssist` in **DESCRIPTION**.
4. Enter `ParkAssistState` in **GROUP NAME** and press `Tab`.
5. **MODE DECLARATION** already holds `Off`. Click **+** (**Add Mode Declaration**), change the new row's **NAME** to `Active` and click outside the table.
6. Check that **INITIAL MODE** is `Off`.

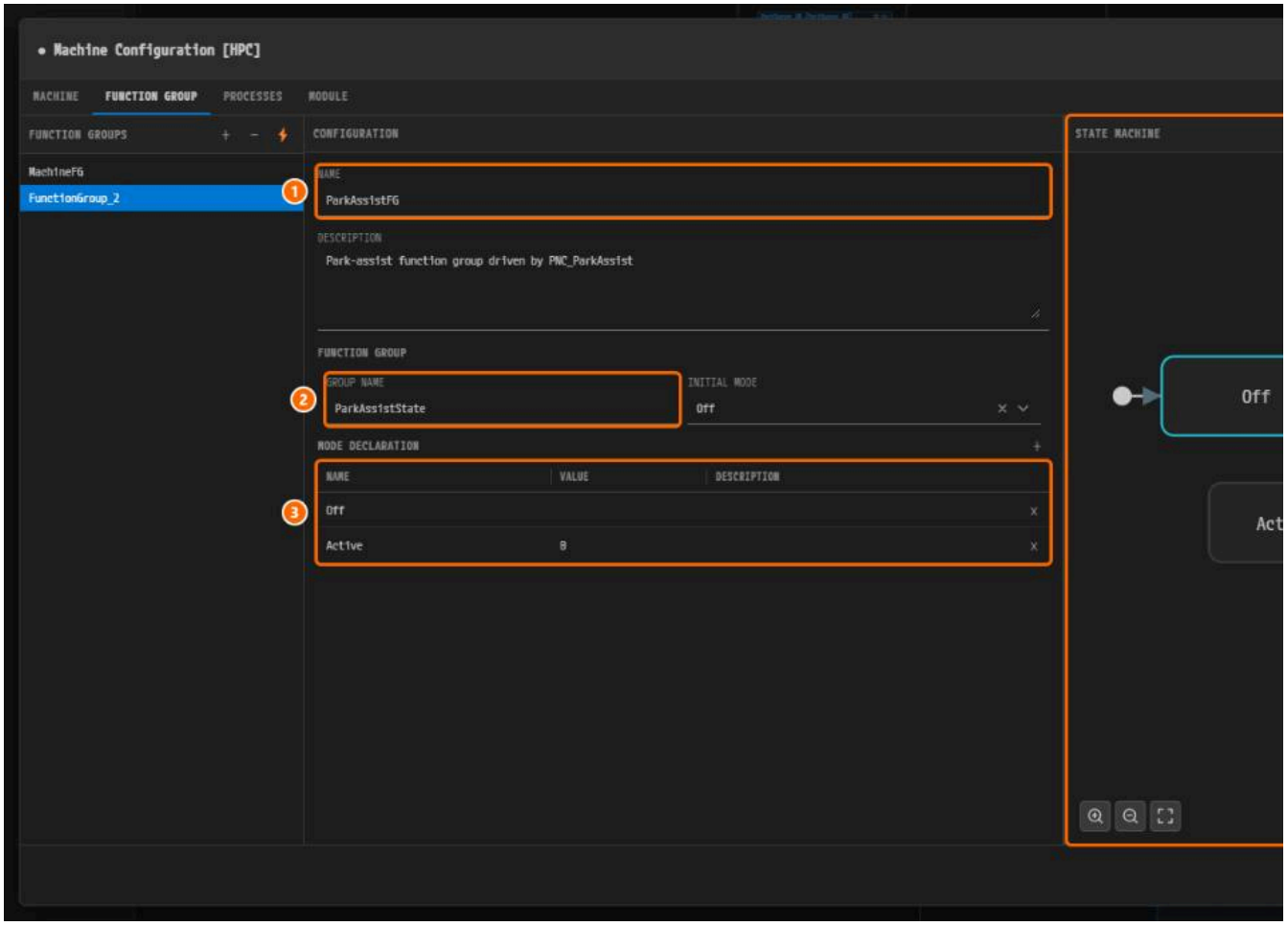


*Figure 9-6. ① **NAME**, ② **GROUP NAME**, ③ `Off` and `Active`, ④ **STATE MACHINE** without transitions.*

7. In **STATE MACHINE**, right-click the `Off` box and click **Transition to** > **Active**. On the `Active` box, click **Transition to** > **Off**.



*Figure 9-7. ① **Transition to**, ② **Active**.*

8. Click `FunctionGroup_2` in the list. Its label changes to `ParkAssistFG`.

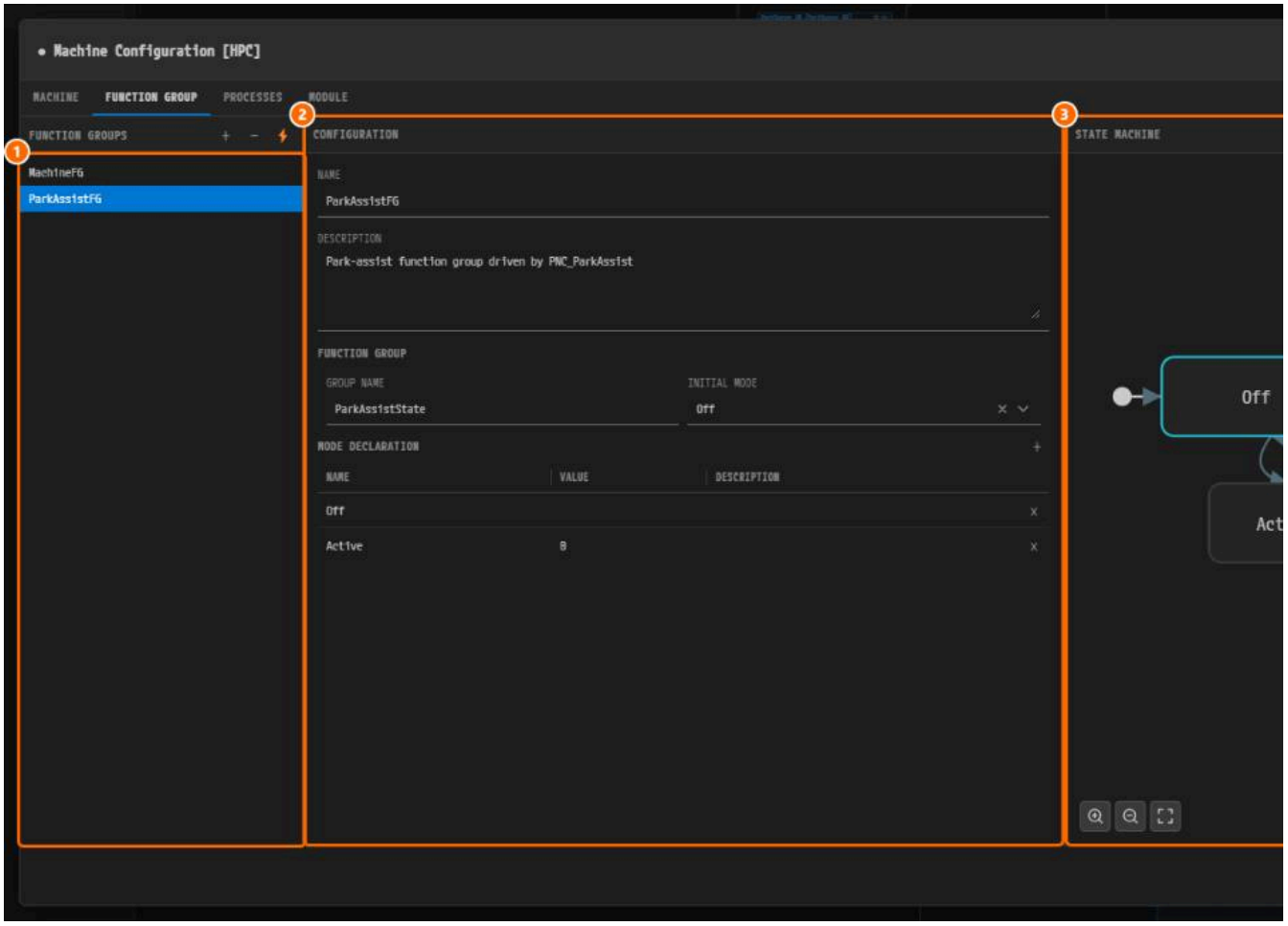


*Figure 9-8. ① **FUNCTION GROUPS**, ② **CONFIGURATION**, ③ the two transitions.*

9. Save with `Ctrl+S`.

> **Note.** A new mode row gets **VALUE** `0`, shown as `Active (0)`. The example model has no value there. Leave it; it does not change the result.

Creating the Remaining FunctionGroups

Create `TrailerFG` and `ComfortFG` the same way, with the transitions `Off` → `Active` and `Active` → `Off`.

NAME	DESCRIPTION	GROUP NAME	INITIAL MODE	MODE DECLARATION	
`MachineFG`	(empty)	`MachineState`	`Off`	`Off`, `Startup`, `Shutdown`, `Restart`, `Verify`	Pr
`ParkAssistFG`	`Park-assist function group driven by PNC_ParkAssist`	`ParkAssistState`	`Off`	`Off`, `Active`	0
`TrailerFG`	`Trailer function group driven by PNC_Trailer`	`TrailerState`	`Off`	`Off`, `Active`	0
`ComfortFG`	`Comfort function group driven by PNC_Comfort`	`ComfortState`	`Off`	`Off`, `Active`	0

1. Create `TrailerFG` with steps 1 to 9 of Creating a FunctionGroup.
2. Create `ComfortFG` the same way.
3. Click `ParkAssistFG` and compare with Figure 9-9.

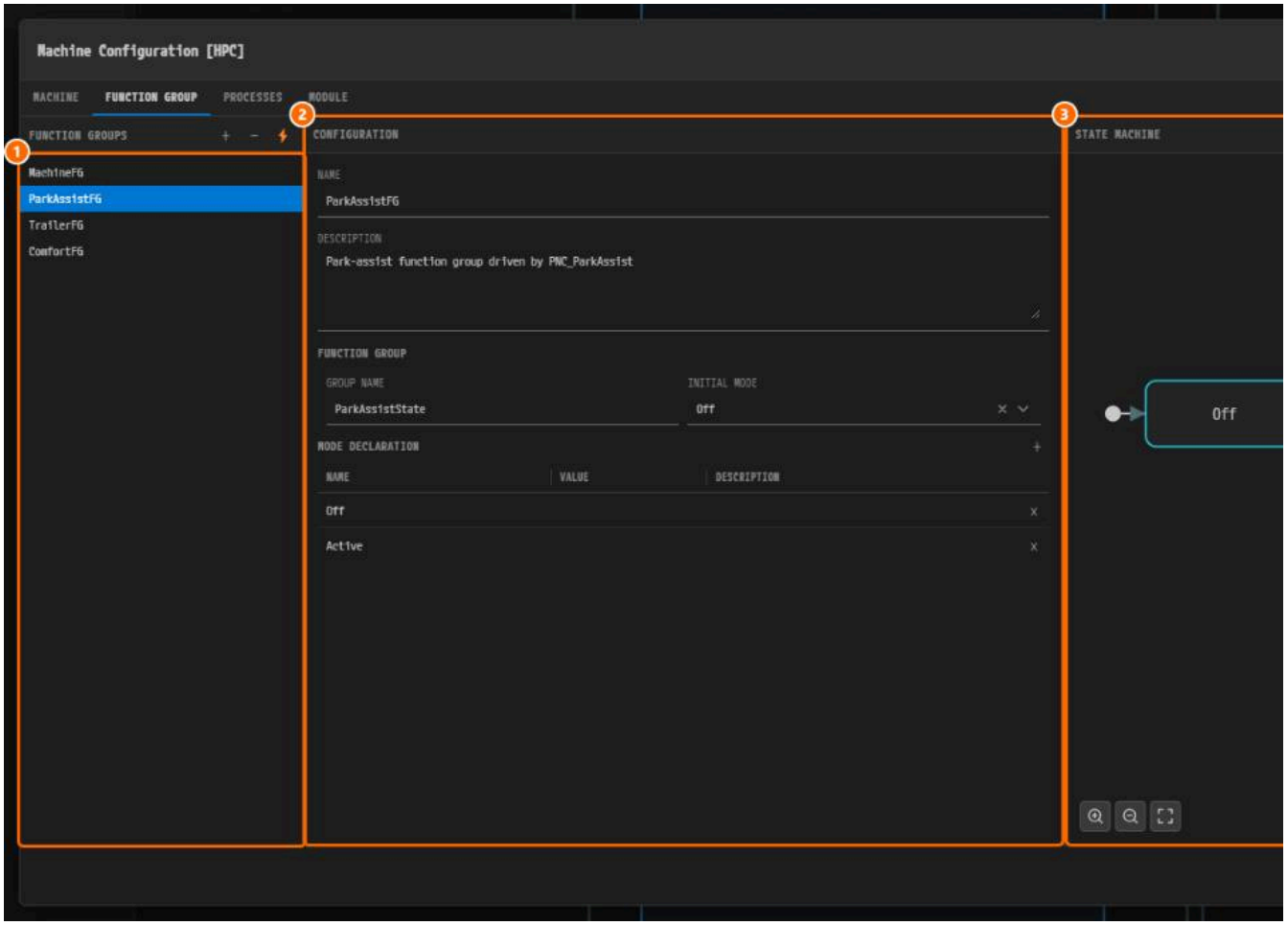


*Figure 9-9. ① The four FunctionGroups, ② **CONFIGURATION** of `ParkAssistFG`, ③ **STATE MACHINE**.*

These FunctionGroups later appear as the 11 `<FunctionGroup>:<state>` candidates in Adding FunctionGroup State Mappings. A name that differs from t

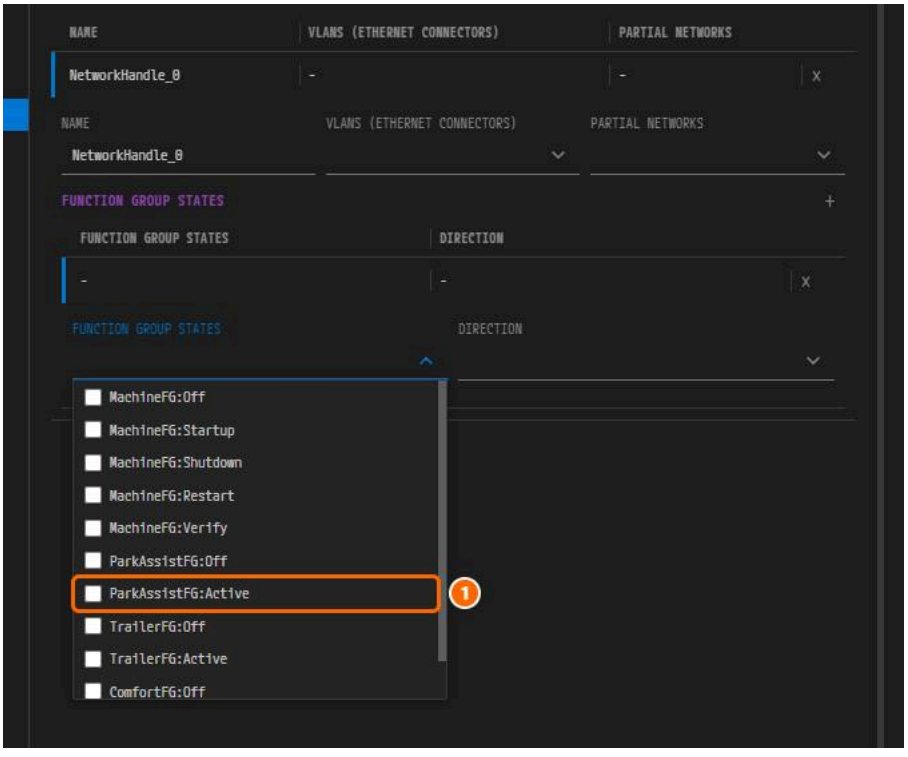


Figure 9-10. The list from a later section. ① `ParkAssistFG:Active`.

Entering MAX ACTION LIST DURATION in State Management

1. Click the **MODULE** tab, then **State Management** in the **MODULES** list. Leave every module enabled.
2. Click the **ACTION ITEMS** sub-tab.
3. Enter `1` in **MAX ACTION LIST DURATION** and click elsewhere. Leave the unit `s`.

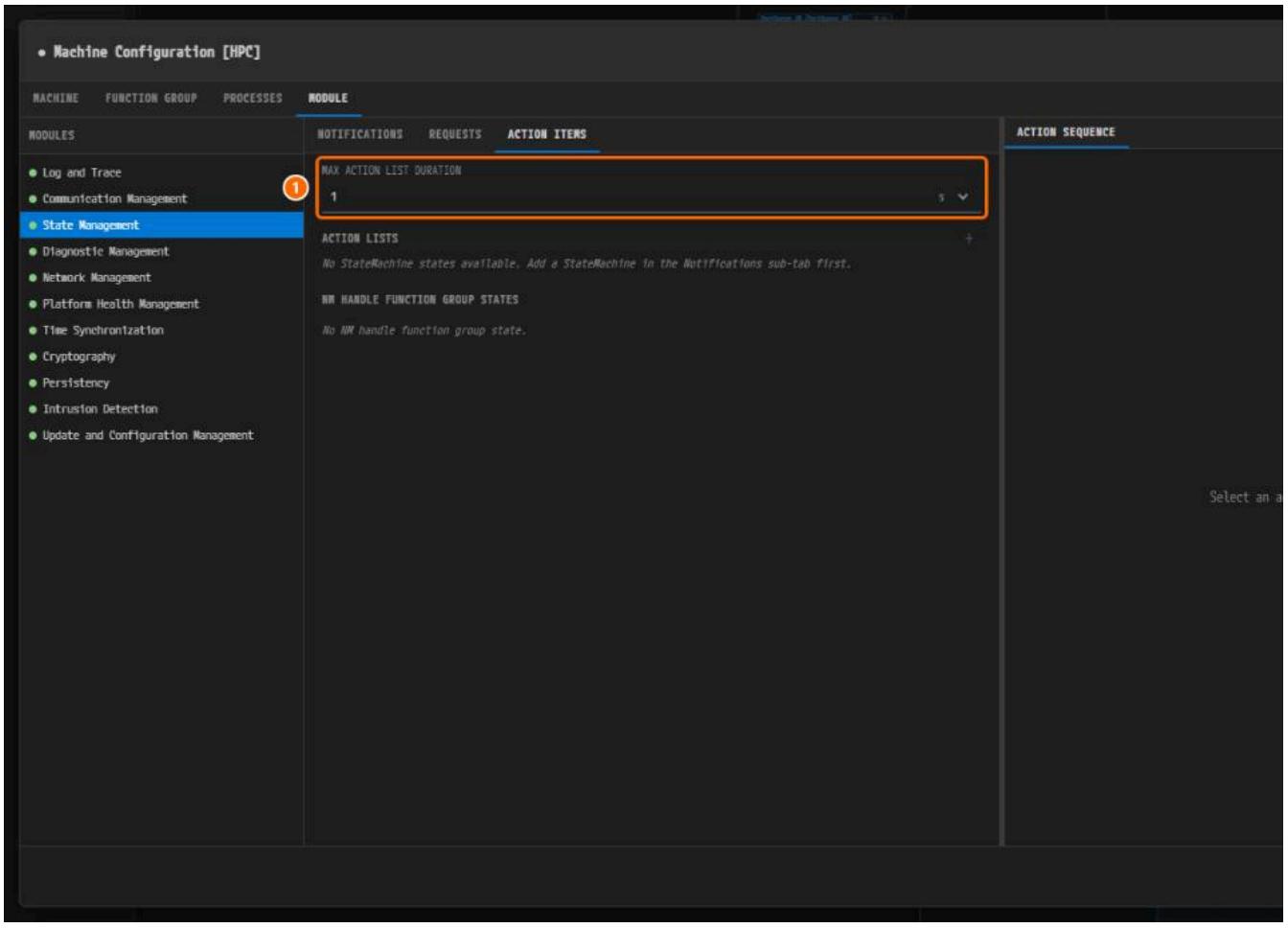


*Figure 9-11. ① **MAX ACTION LIST DURATION** `1` `s`.*

4. Save with `Ctrl+S`.

> **Caution.** Without this value, validation reports `constr_10641`.

Adding an NM Network Handle

1. Click **Network Management** in the **MODULES** list.

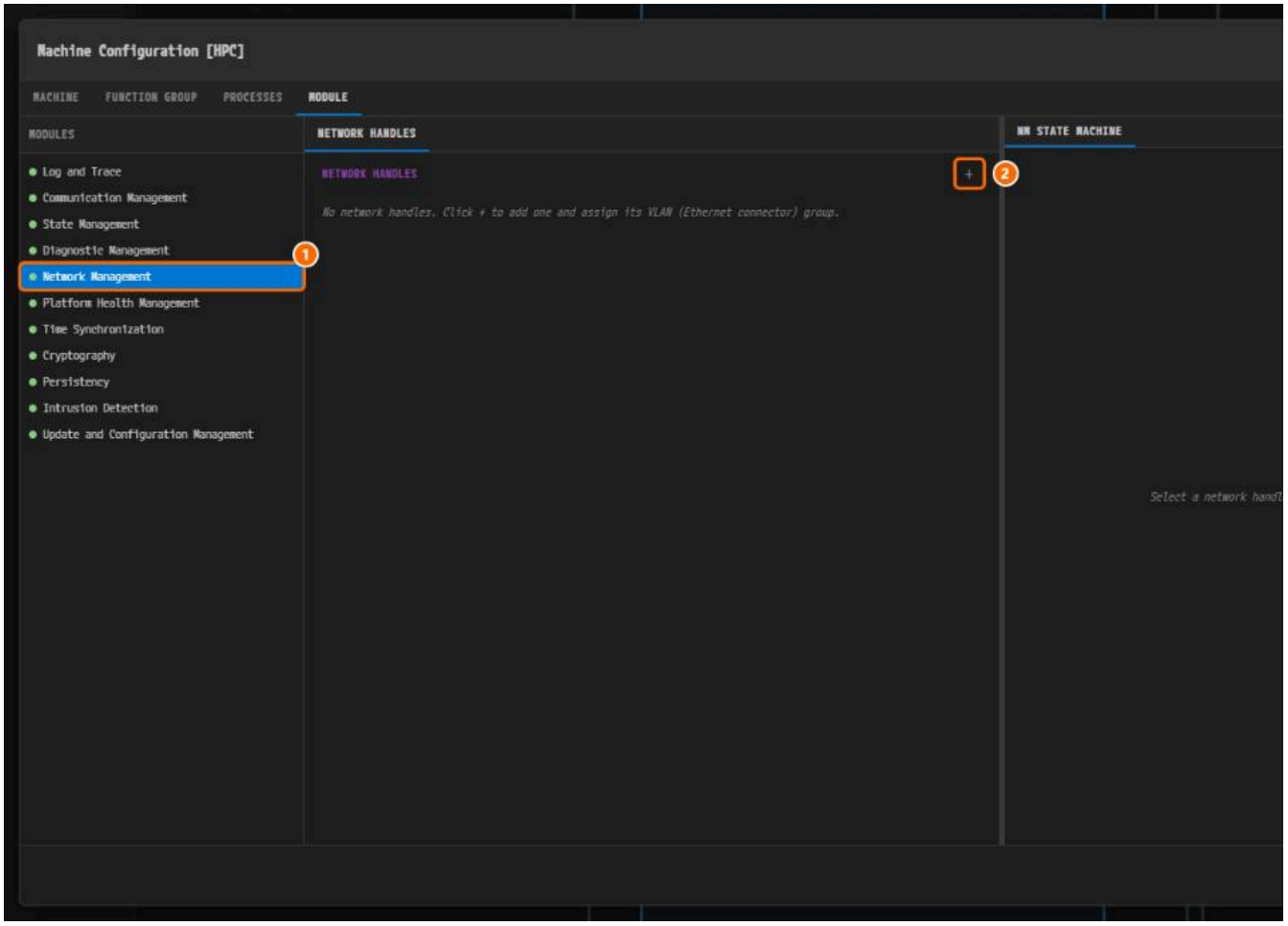


*Figure 9-12. ① **Network Management**, ② **+ (Add Network Handle)**.*

2. Click **+** next to **NETWORK HANDLES**. `NetworkHandle_0` is added and expanded.

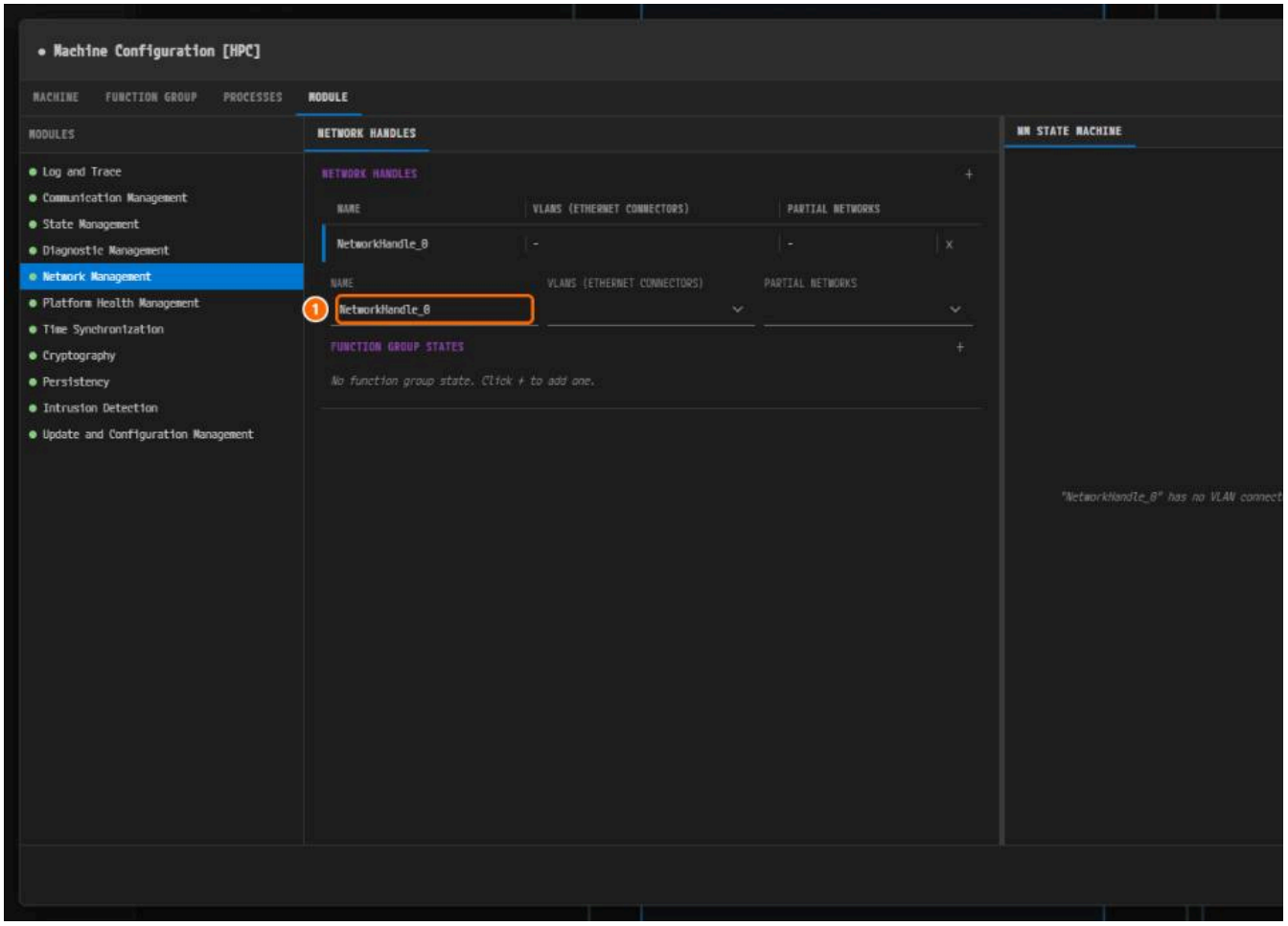


*Figure 9-13. ① **NAME.***

3. Enter `NH_ParkAssist` in **NAME** and press `Tab`.

4. The summary row shows `NH_ParkAssist`.

5. Click **VLANS (ETHERNET CONNECTORS)**, check **HPC_Eth_Rear_Conn** and press `Esc`.

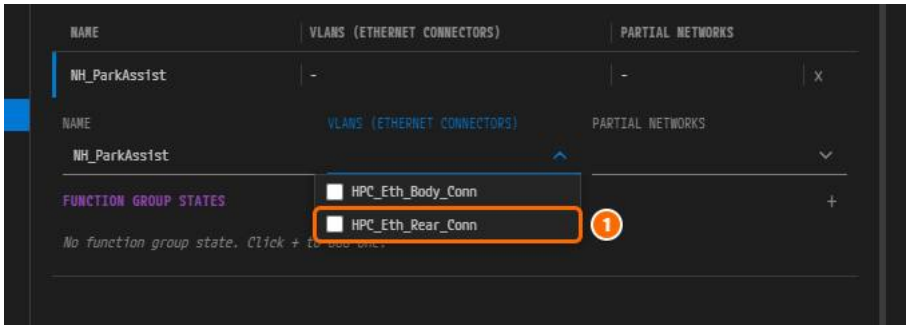


*Figure 9-14. ① **HPC_Eth_Rear_Conn.***

6. Click **PARTIAL NETWORKS**, check **PNC_ParkAssist (PNC 18)** and press `Esc`.

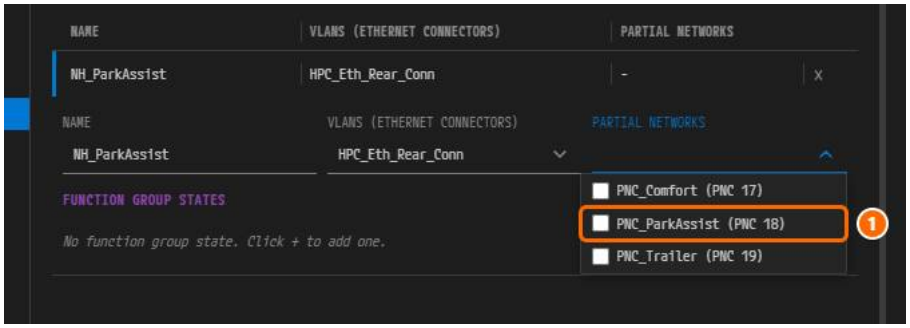


*Figure 9-15. ① **PNC_ParkAssist (PNC 18)**.*

Adding FunctionGroup State Mappings

The first mapping requests `PNC_ParkAssist` when `ParkAssistFG` becomes `Active`. The second sets `ParkAssistFG` to `Off` when the PNC becomes ina

1. Inside the handle, click **+** (**Add Function Group State**) next to **FUNCTION GROUP STATES**. A mapping row opens.
2. Click **FUNCTION GROUP STATES**, check **ParkAssistFG:Active** and press `Esc`.

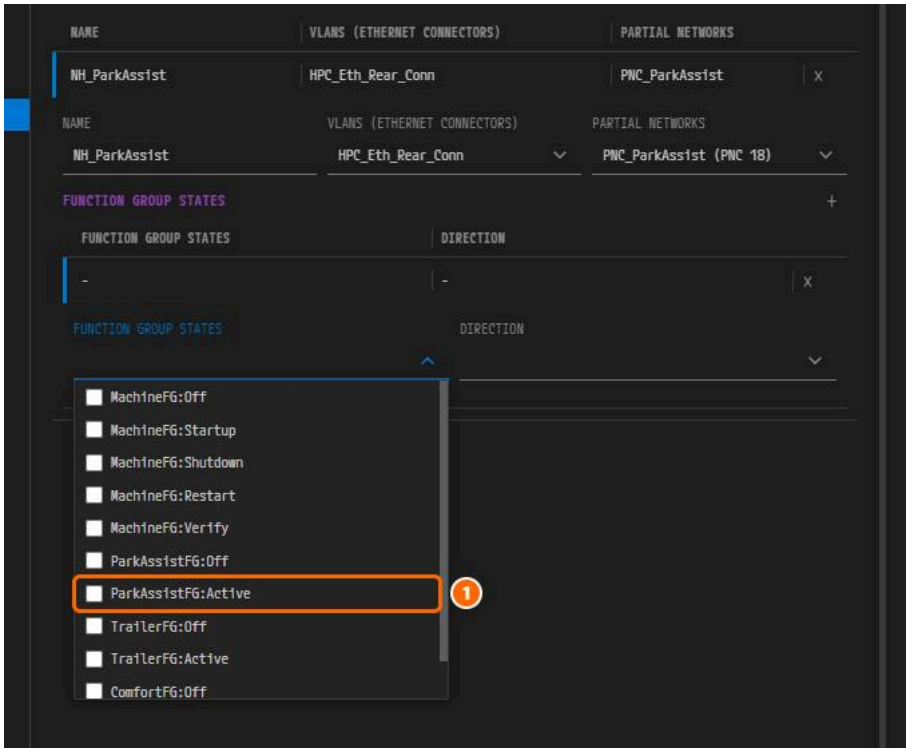


*Figure 9-16. ① **ParkAssistFG:Active**.*

3. In **DIRECTION**, select **FUNCTION-GROUP-STATE-TO-NM-HANDLE**.

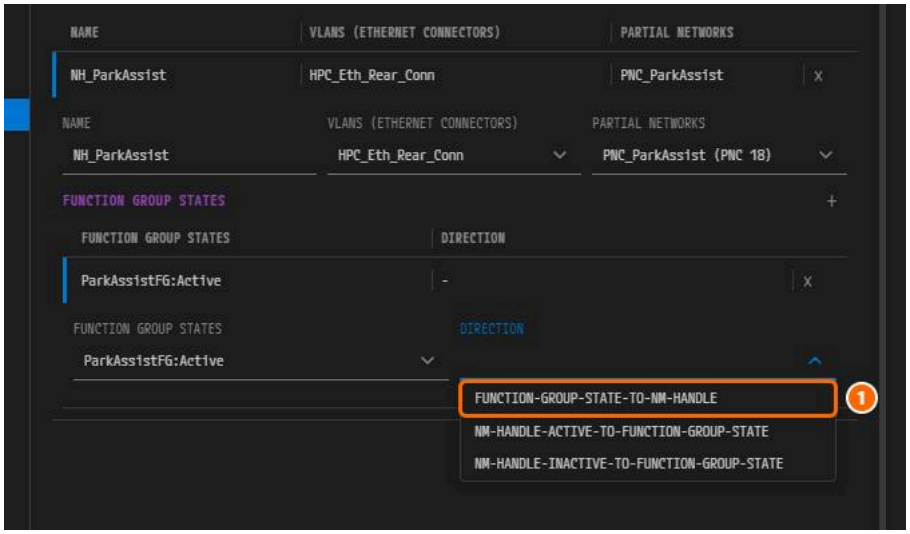


*Figure 9-17. ① **FUNCTION-GROUP-STATE-TO-NM-HANDLE.***

4. Click **+** next to **FUNCTION GROUP STATES** again.

5. In the new row, select **ParkAssistFG:Off** and **NM-HANDLE-INACTIVE-TO-FUNCTION-GROUP-STATE**.

6. Save with `Ctrl+S`.

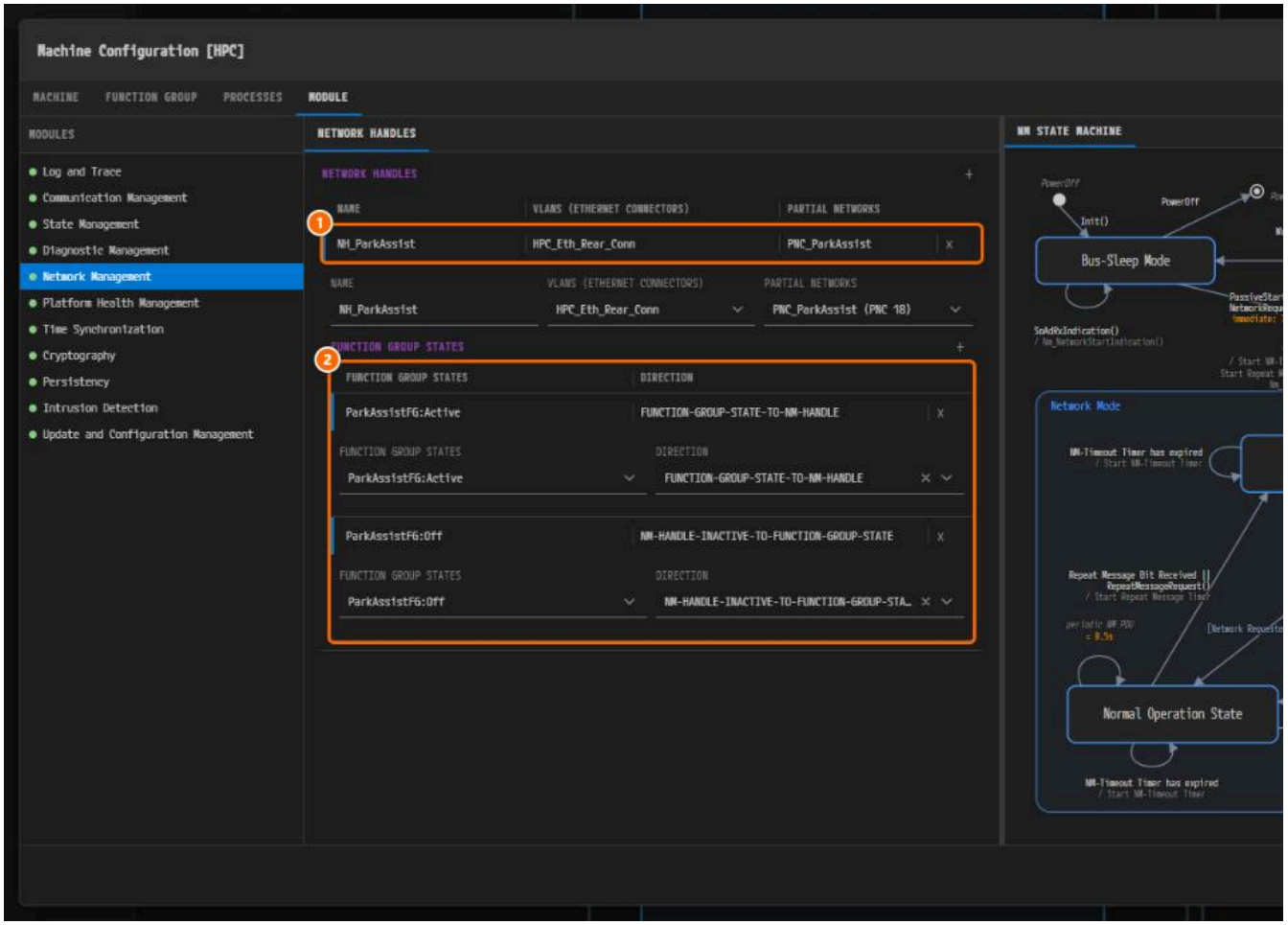


Figure 9-18. ① The handle row, ② the two mappings.

7. Click **State Management** > **ACTION ITEMS**. **NM HANDLE FUNCTION GROUP STATES** now lists the two mappings.

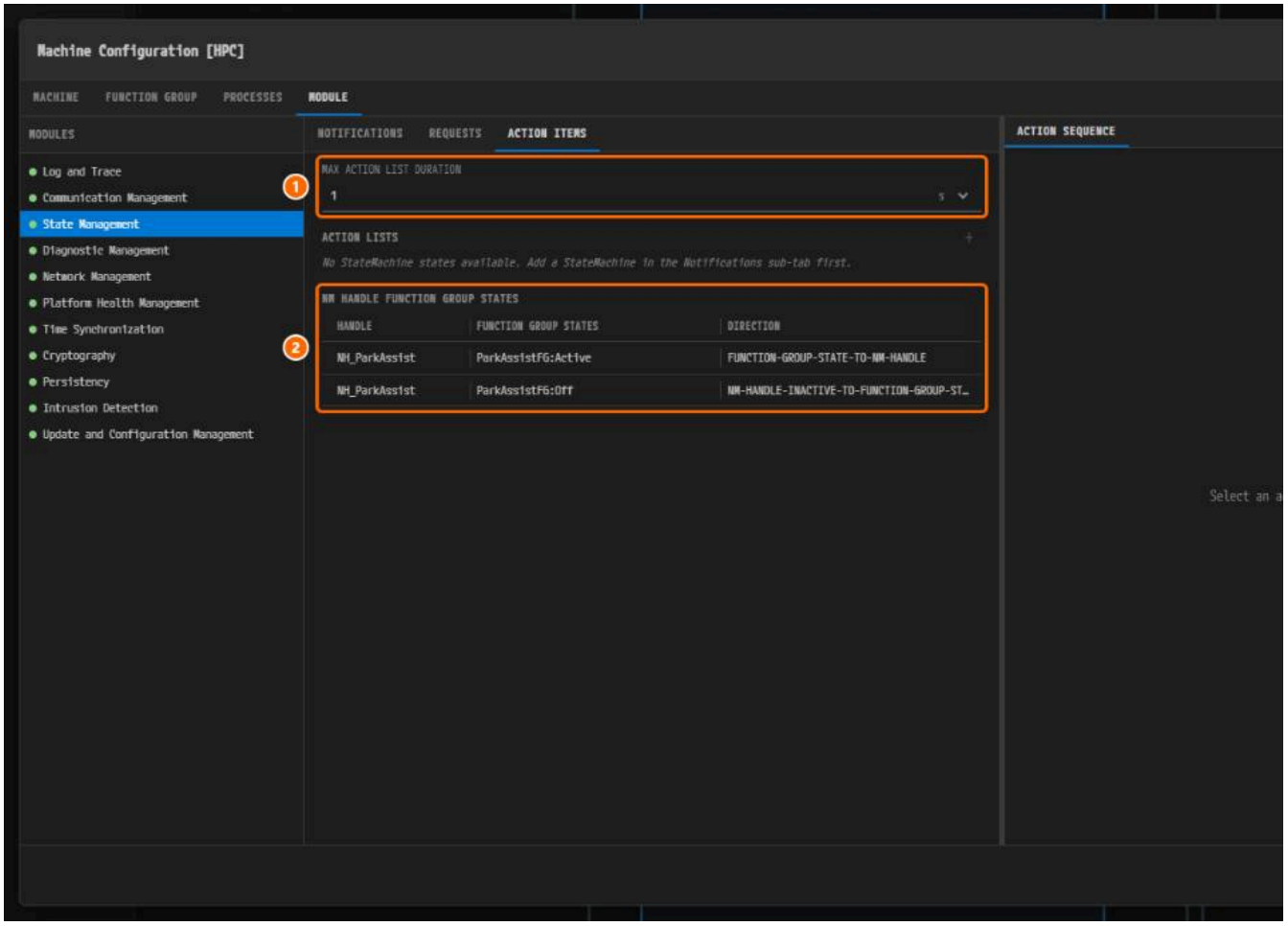


*Figure 9-19. ① **MAX ACTION LIST DURATION**, ② **NM HANDLE FUNCTION GROUP STATES** (read-only).*

Adding the Remaining Handles

Create `NH_Comfort` and `NH_Trailer` the same way.

NAME	VLANS (ETHERNET CONNECTORS)	PARTIAL NETWORKS	FUNCTION GROUP STATES	DIRECTION
`NH_Comfort`	`HPC_Eth_Body_Conn`	`PNC_Comfort (PNC 17)`	`ComfortFG:Active`	`NM-HANDLE-ACTIVE-TO-FUNCTION-GROUP-STATE`
			`ComfortFG:Off`	`NM-HANDLE-INACTIVE-TO-FUNCTION-GROUP-STATE`
`NH_ParkAssist`	`HPC_Eth_Rear_Conn`	`PNC_ParkAssist (PNC 18)`	`ParkAssistFG:Active`	`FUNCTION-GROUP-STATE-TO-NM-HANDLE`
			`ParkAssistFG:Off`	`NM-HANDLE-INACTIVE-TO-FUNCTION-GROUP-STATE`
`NH_Trailer`	`HPC_Eth_Rear_Conn`	`PNC_Trailer (PNC 19)`	`TrailerFG:Active`	`NM-HANDLE-ACTIVE-TO-FUNCTION-GROUP-STATE`
			`TrailerFG:Off`	`NM-HANDLE-INACTIVE-TO-FUNCTION-GROUP-STATE`

DIRECTION	Meaning
`FUNCTION-GROUP-STATE-TO-NM-HANDLE`	The FunctionGroup enters this state → the handle requests its PNC.
`NM-HANDLE-ACTIVE-TO-FUNCTION-GROUP-STATE`	The PNC becomes active → the FunctionGroup moves to this state.
`NM-HANDLE-INACTIVE-TO-FUNCTION-GROUP-STATE`	The PNC becomes inactive → the FunctionGroup moves to this state.

1. Click **Network Management** again.
2. Create `NH_Comfort` with steps 2 to 6 of Adding an NM Network Handle and steps 1 to 5 of Adding FunctionGroup State Mappings.

3. Create `NH_Trailer` the same way.

4. Save with `Ctrl+S`.

5. Expand `NH_Comfort` and compare with Figure 9-20. The row order and expanded rows may differ.

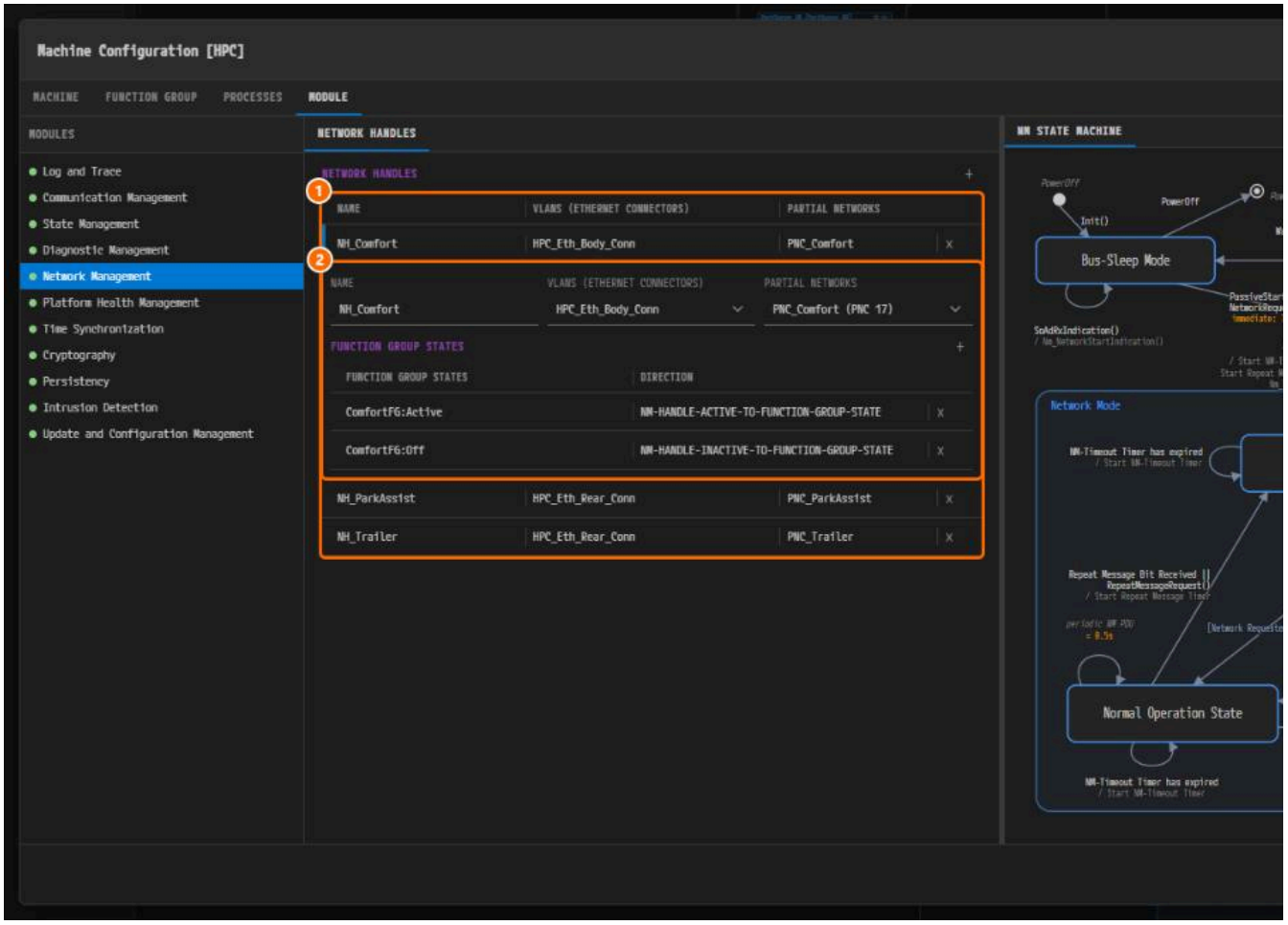


Figure 9-20. ① The three handles, ② the detail of `NH_Comfort`.

6. **State Management** > **ACTION ITEMS** now lists six mappings.

7. Click **Close**.

> **Caution.** A handle with a PNC but no `NM-HANDLE-INACTIVE-TO-FUNCTION-GROUP-STATE` mapping fails validation with "PNC release stops nothing".

Why Is It Designed This Way?

- **No gateway on the HPC. PNC GATEWAY TYPE** exists only on CP ECU connectors, so the AP machine takes part in PNCs through ara::nm network handle
- **One FunctionGroup per PNC.** The applications that run while a PNC is alive share one state.
- **Mappings in both directions.** Reverse gear sets `ParkAssistFG` `Active` and requests `PNC_ParkAssist`; an active `PNC_Comfort` sets `ComfortFG` `Act`
- **No wakeup frame for `PNC_ParkAssist`.** A FunctionGroup state on the HPC requests it, not a bus frame.

Checklist

- [ ] **FUNCTION GROUPS** holds `MachineFG`, `ParkAssistFG`, `TrailerFG` and `ComfortFG`, with the values and transitions of the table.
- [ ] **MAX ACTION LIST DURATION** is `1` s.
- [ ] **NETWORK HANDLES** holds `NH_Comfort`, `NH_ParkAssist` and `NH_Trailer` with the values of Adding the Remaining Handles.
- [ ] **NM HANDLE FUNCTION GROUP STATES** holds six rows.
- [ ] The dialog closed without a save prompt.

Next Step

In PARA Platform Import and PNC Request Ports you import the PARA platform and connect the AAs that request PNCs through these handles.

Importing the PARA Platform and PNC Request Ports

What You Will Do

In this chapter you import the PARA base platform into the HPC machine. You then give each application AA an ara::nm P-Port and map it to an NM networ PNC. `ParkAssistApp` is shown step by step; the other two AAs follow the table.

Before You Begin

- Start after finishing AP Machine — NM Handles and FunctionGroups. The HPC machine has the handles `NH_Comfort`, `NH_ParkAssist` and `NH_Traile`
- Reference: sections 8.9 to 8.11 of the VFB editor and section 9.3.5 of the Interface editor.

Item	Value
NM port interface **NAME** / **C++ NAMESPACE**	`NMI_PncRequest` / `pnc::nm`
P-Port **NAME** on `ParkAssistApp`	`PncParkAssist`
PORT INTERFACE of the P-Port	`NM.NetworkHandle: NMI_PncRequest`
NetworkHandle mapping **PORT** / **NETWORK HANDLE**	`PncParkAssist` / `NH_ParkAssist`

What the PARA Platform Is

PARA is the base configuration set of the PopcornSAR AUTOSAR Adaptive Platform. One import adds what an AP machine needs to run.

Item	Contents
11 platform AAs	`EM`, `CM`, `PHM`, `SM`, `TBM`, `DM`, `IDSM`, `NM`, `UCM`, `VUCM`, `CSM`, in the composition `PARA_HPC` (prototype `PARA` in the HPC)
Interfaces	8 service interfaces and the state management interface `StateClient_HPC_MachineFG`
Machine	The FunctionGroup `MachineFG` and 7 modules (`LOG`, `CM`, `SM`, `NM`, `PHM`, `TIME_SYNC`, `CRYPTO`)
Others	4 error domains, 62 data types, 5 connectors inside `PARA_HPC`

PARA has no NM port interface for the application AAs. You create `NMI_PncRequest` in Creating the NM Port Interface.

Importing the PARA Platform

1. Click **VFB** under **DESIGN**. In the root view, click the machine icon at the right of the **HPC [HPC]** header, as in Chapter 9.

2. In **Machine Configuration [HPC]**, click **PARA Platform Import** at the top right.

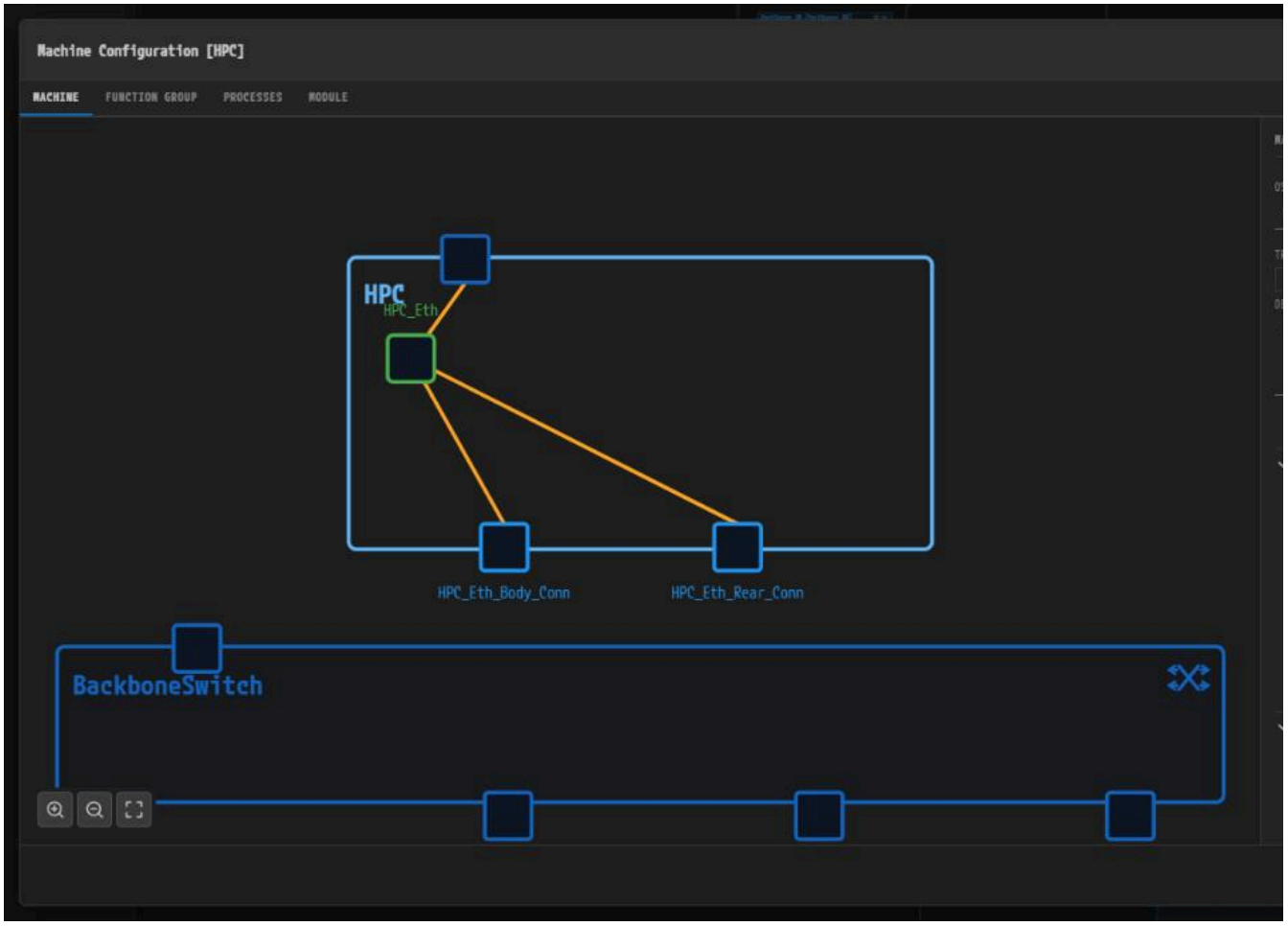


*Figure 10-1. ① **PARA Platform Import.***

3. Click **Import.**

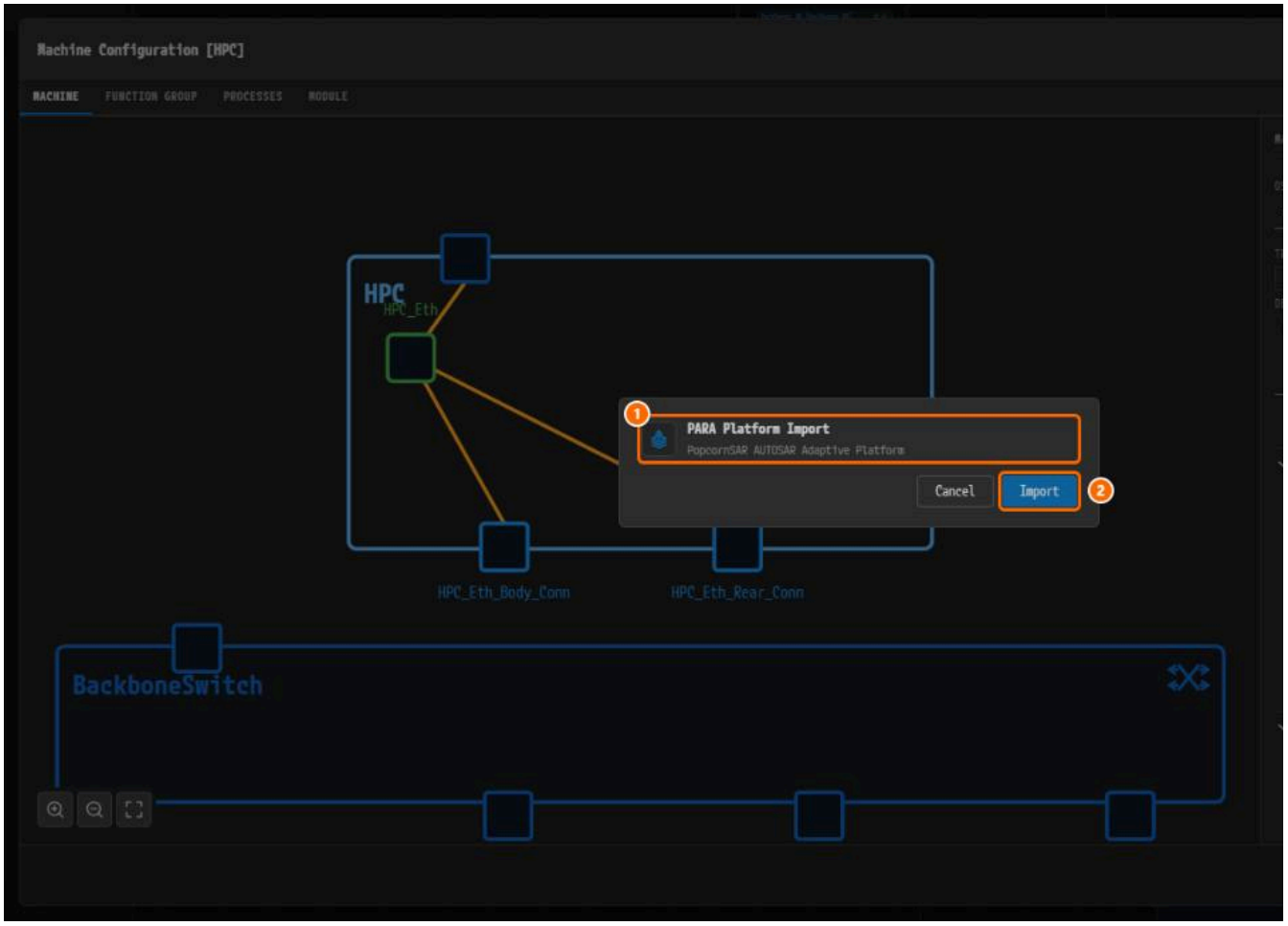


*Figure 10-2. ① The **PARA Platform Import** window, ② **Import**.*

4. The **Conflict Resolution** dialog opens with one conflict, **Machine (1)** > **HPC**: PARA sets **Max Action List Duration** to `5`, and you entered `1` in Chapter 9. then click **Resolve 1/1**.

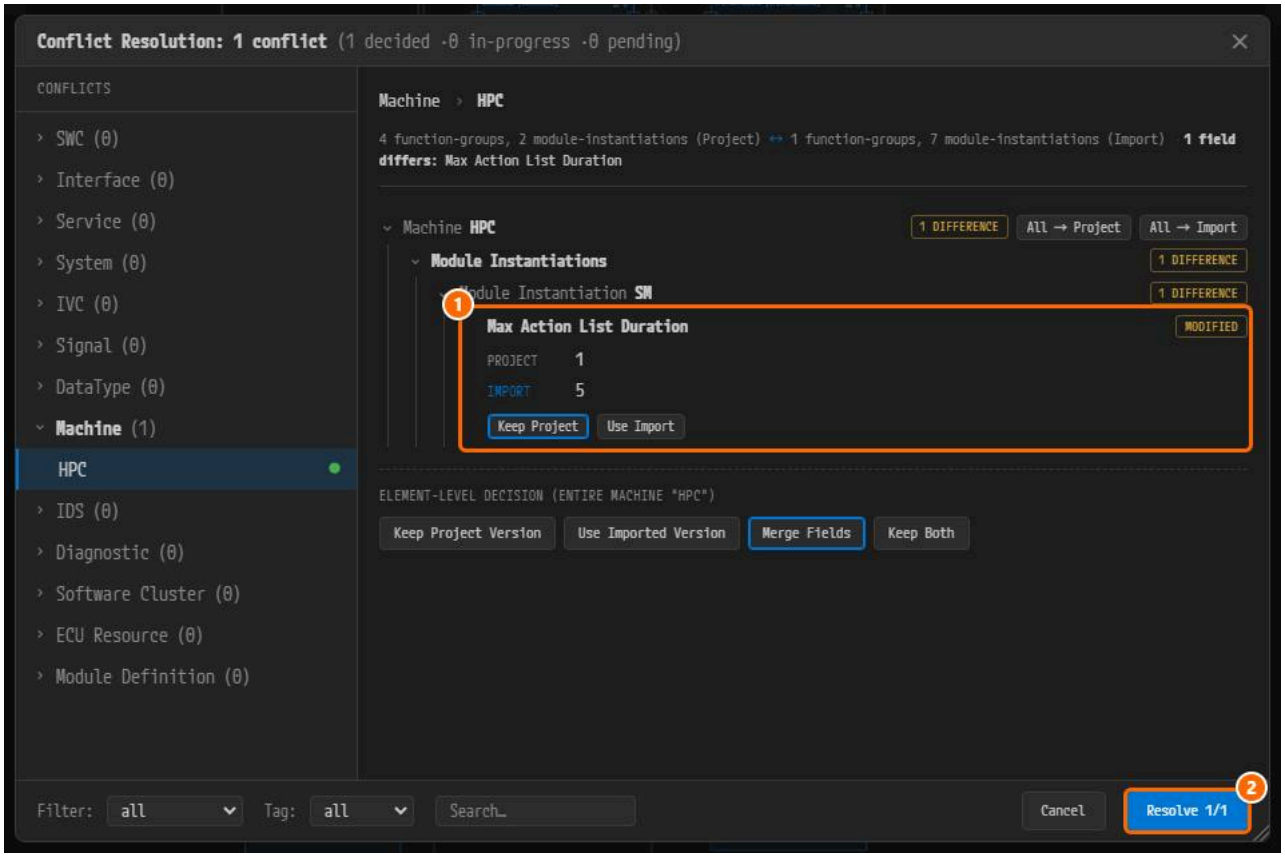


*Figure 10-19. ① **Keep Project** under **Max Action List Duration**, ② **Resolve 1/1**.*

> **Caution.** Decide before you click **Resolve**: an undecided conflict is overwritten with the imported value. Do not use the machine-level buttons. **Use** handles, and **Keep Project Version** merges no PARA module.

5. When the import completes, the **Machine Configuration [HPC]** dialog reopens.

Checking the Import Result

1. Click the **MODULE** tab, then **Network Management**. The **NETWORK HANDLES** table still holds your three handles.

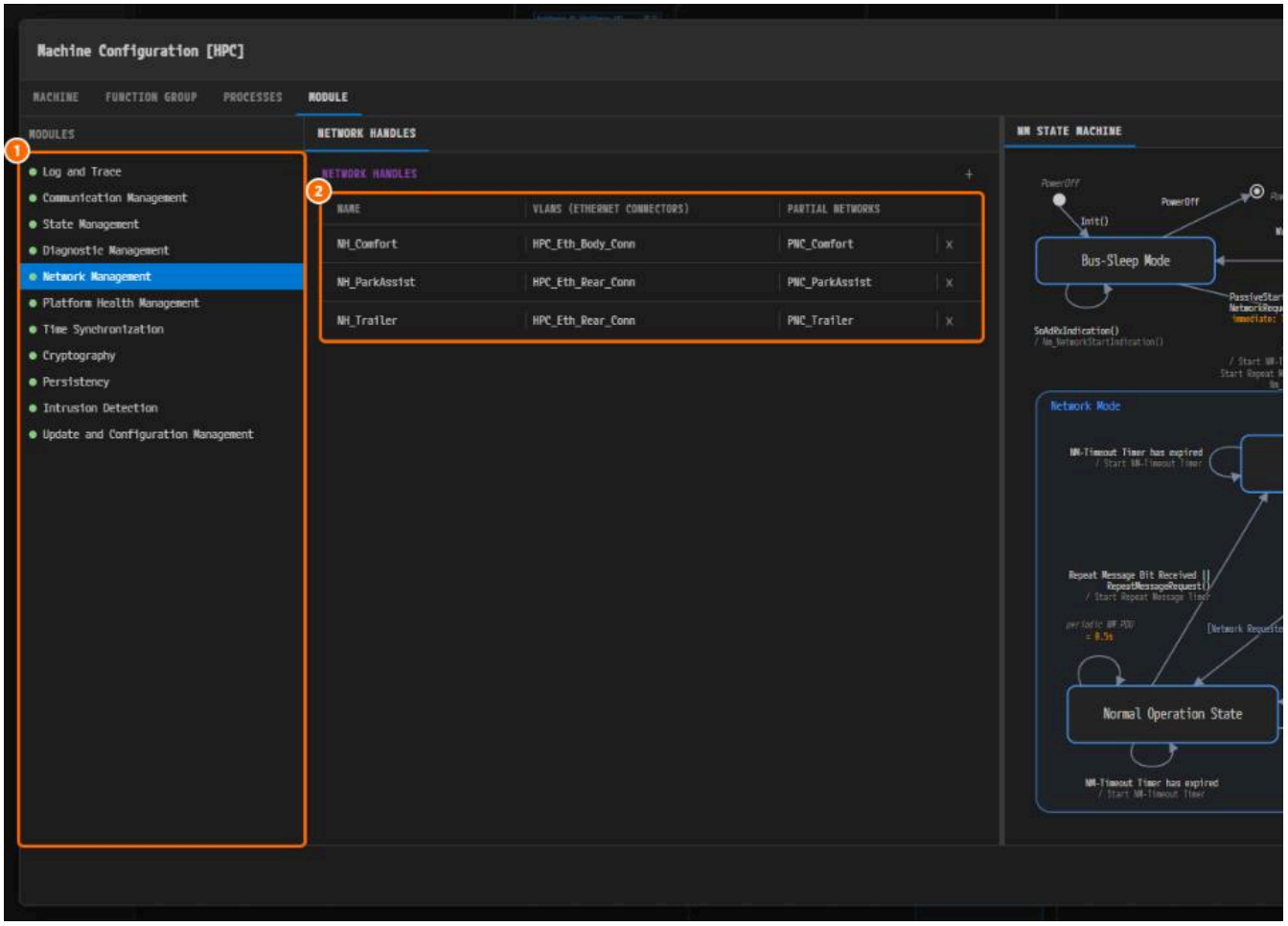


*Figure 10-3. ① **MODULES** (all 11 module kinds are always listed), ② **NETWORK HANDLES**.*

2. Click the **PROCESSES** tab. The list shows the three application processes and the 11 PARA processes.

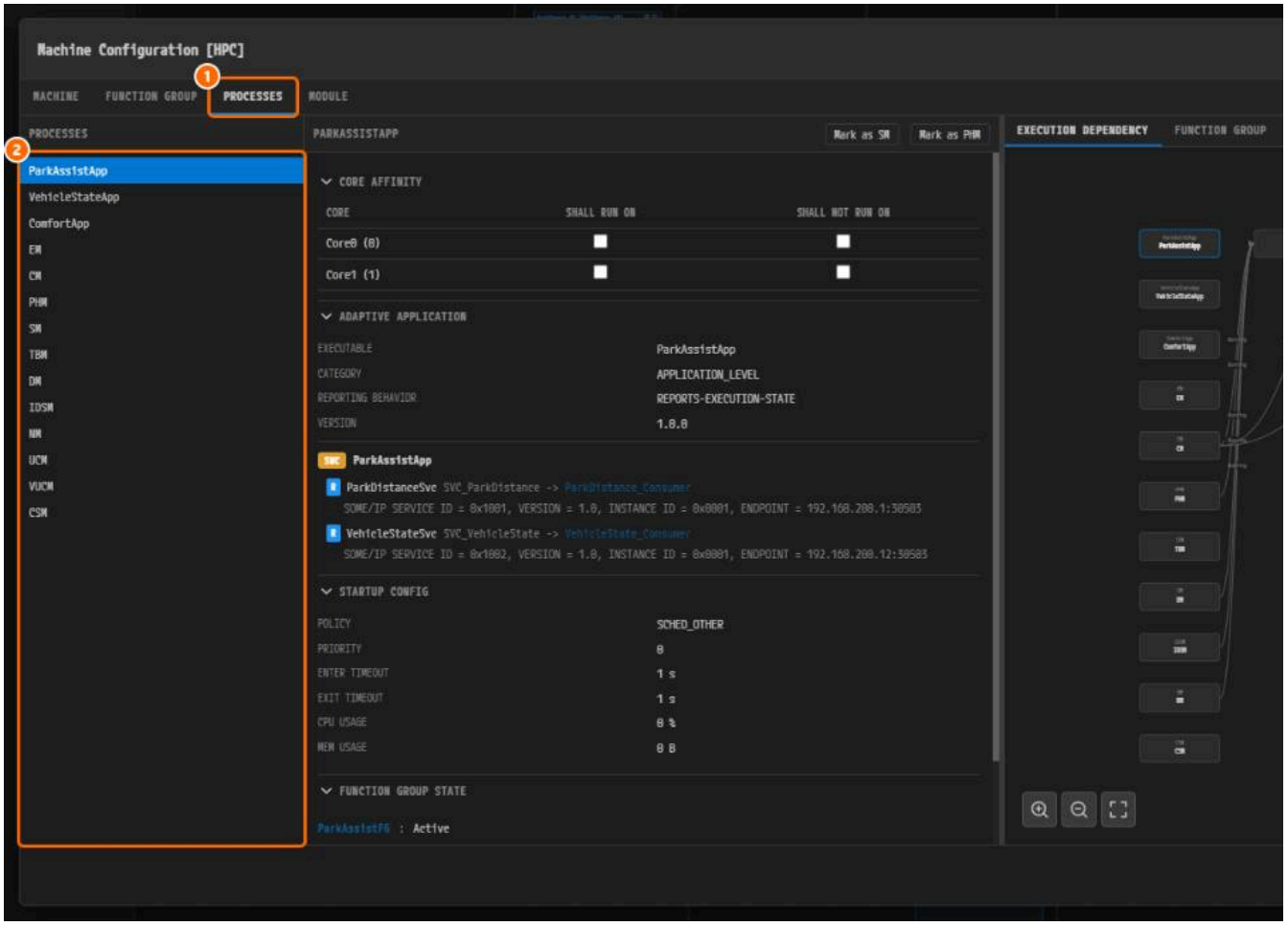


*Figure 10-4. ① **PROCESSES**, ② the 14 processes.*

3. Click **Close**.

Tidying Up the HPC Composition

1. On the VFB canvas, double-click **HPC [HPC]**. If the new node **PARA_HPC** overlaps **ComfortApp**, click **Auto Layout**.

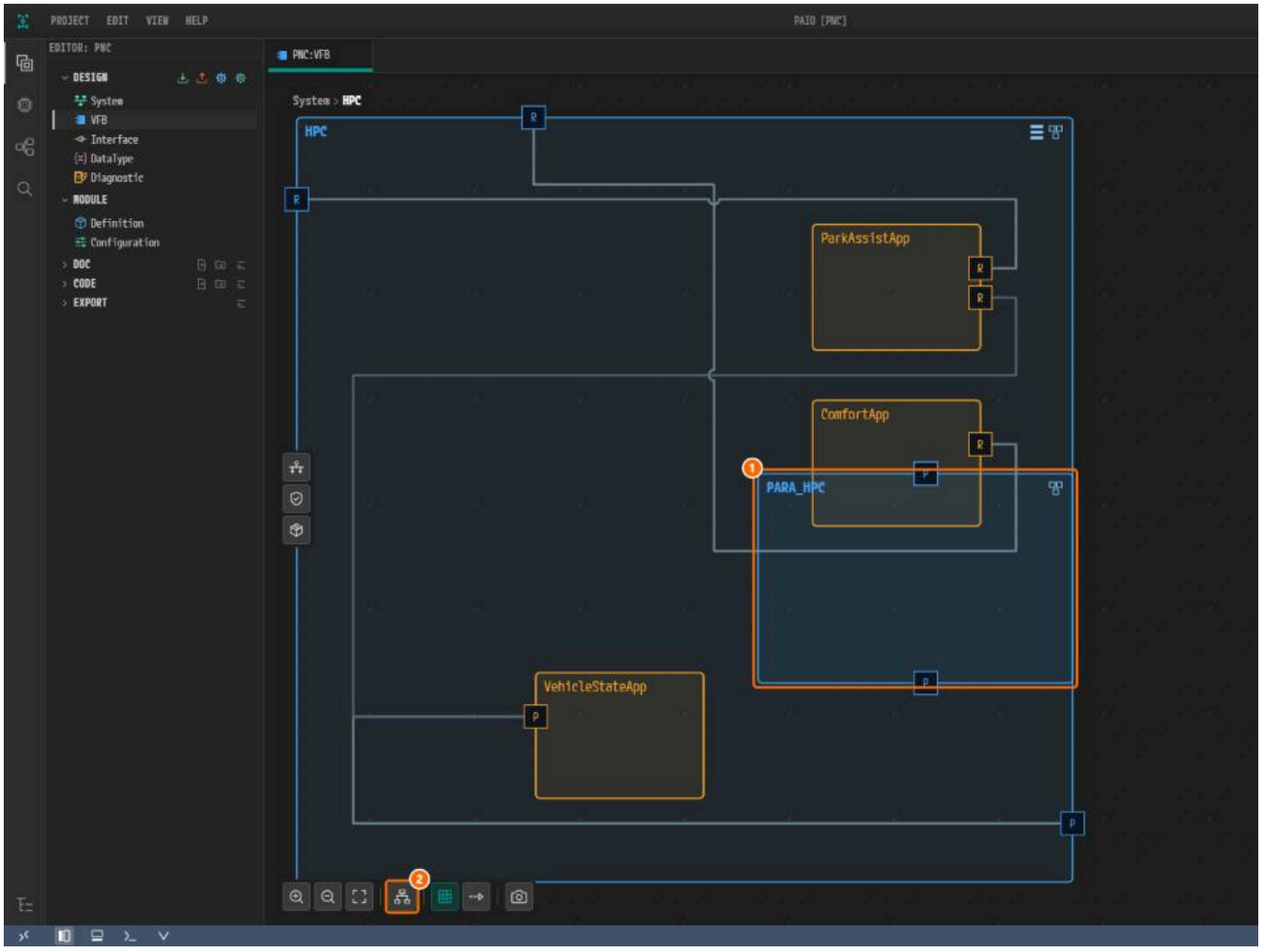


*Figure 10-5. ① **PARA_HPC**, ② **Auto Layout**.*

2. Save with `Ctrl+S`.

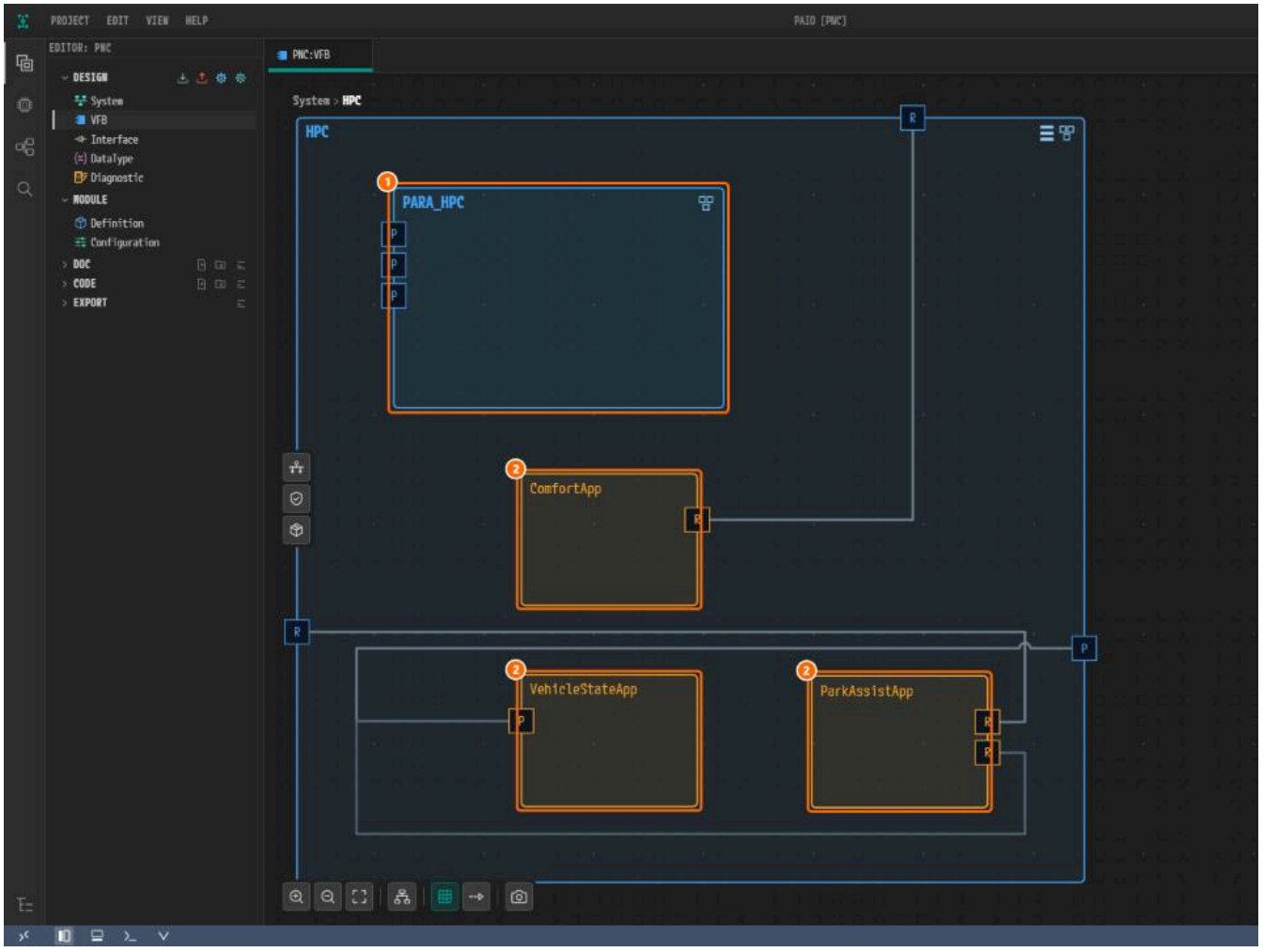


*Figure 10-6. ① The prototype `PARA`, shown as **PARA_HPC**, ② the three application AAs.*

Creating the NM Port Interface

1. Click **Interface** under **DESIGN** and open the **ADAPTIVE** tab.
2. Click **+** next to **INTERFACES** and choose **ara::nm** > **NetworkManagement**.

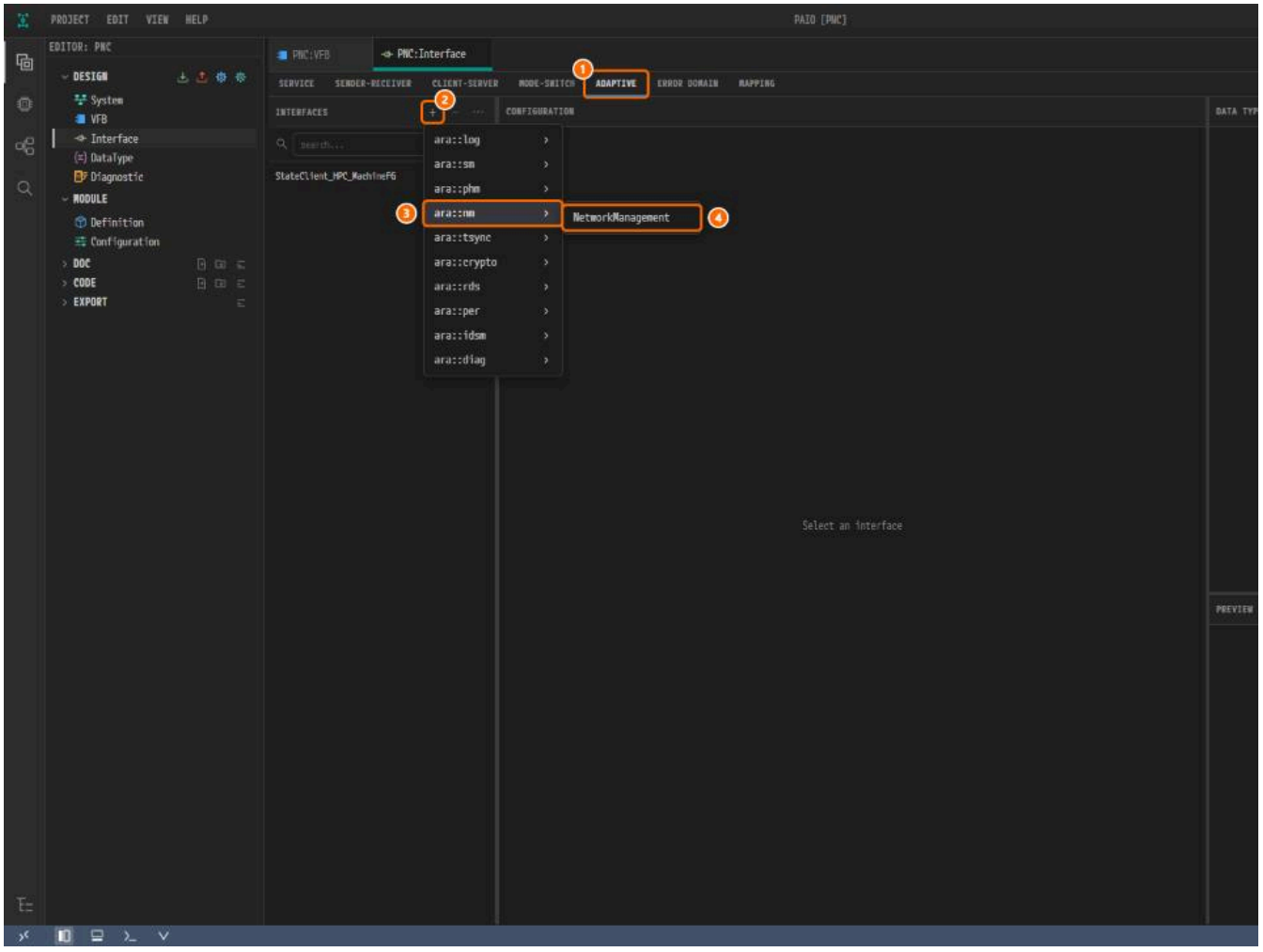


*Figure 10-7. ① **ADAPTIVE**, ② **+**, ③ **ara::nm**, ④ **NetworkManagement**.*

3. Under **CONFIGURATION**, enter `NMI_PncRequest` in **NAME** and `pnc::nm` in **C++ NAMESPACE**. Save with `Ctrl+S`.

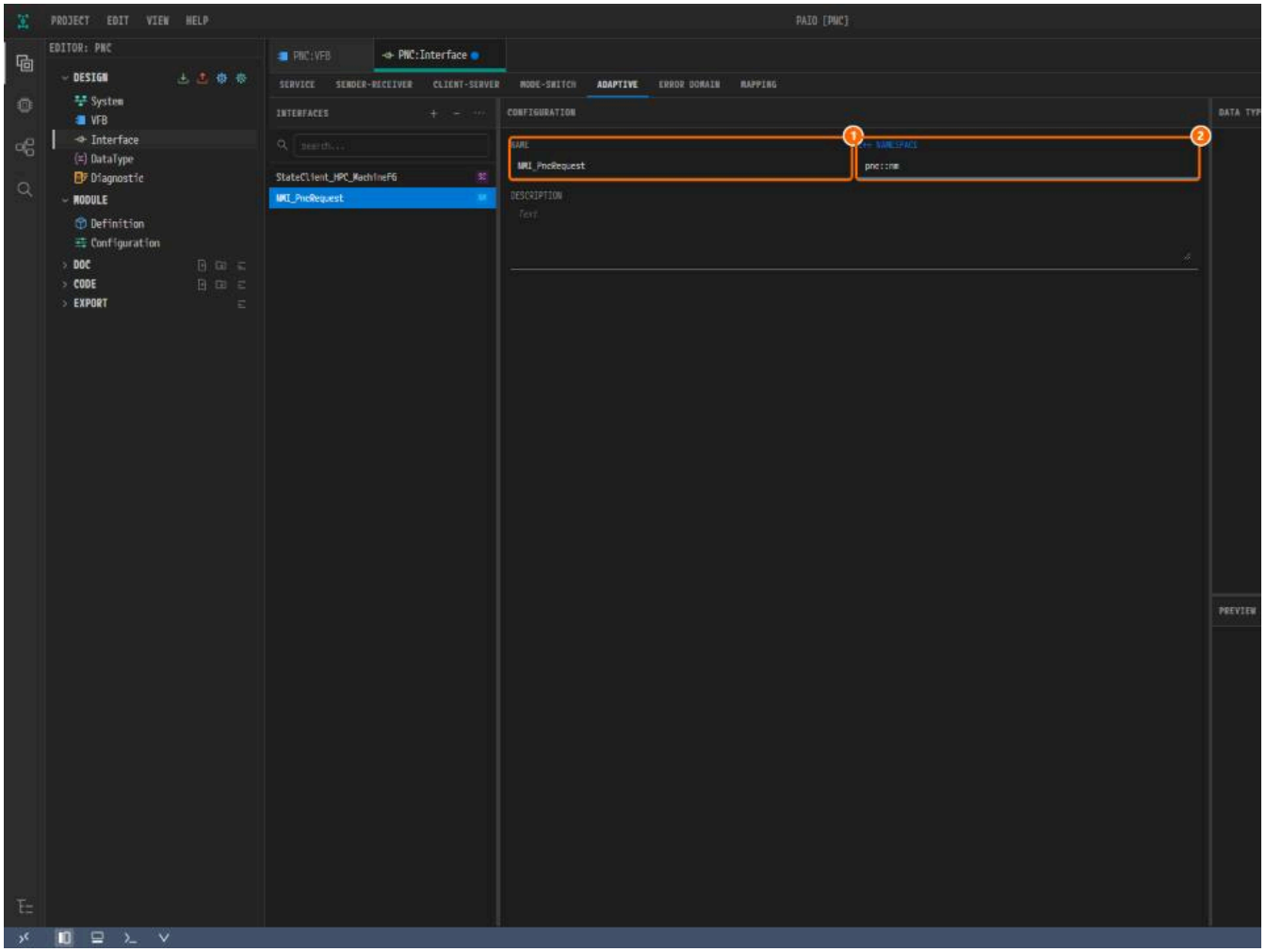


*Figure 10-8. ① **NAME**, ② **C++ NAMESPACE**.*

Adding the PNC Request Port to ParkAssistApp

1. Return to the tab **PNC:VFB**, still inside the HPC. Right-click **ParkAssistApp** and choose **Add Port** > **PPort**.

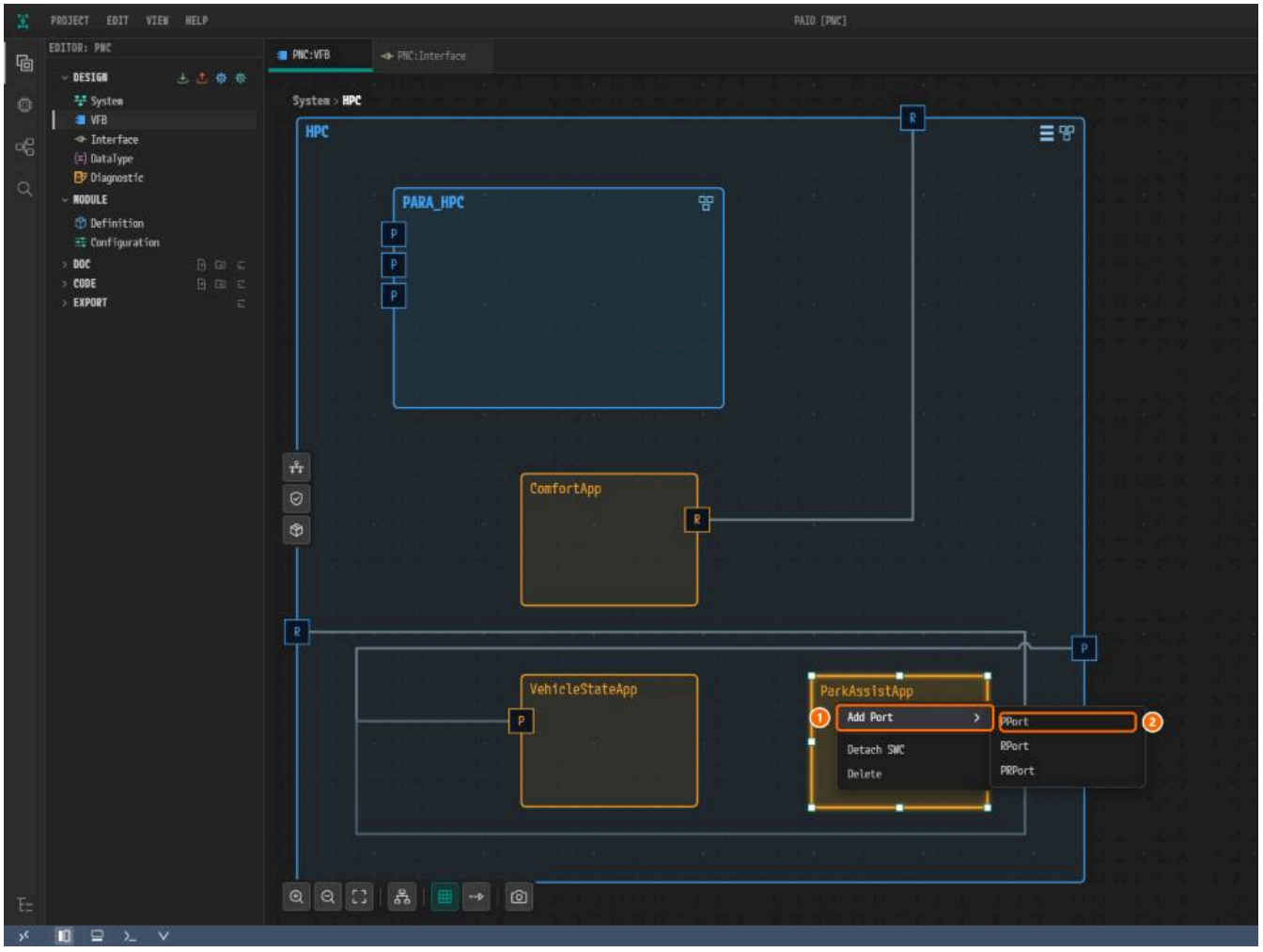


*Figure 10-9. ① **Add Port**, ② **PPort**.*

2. Click the new port. Change **NAME** to `PncParkAssist` and press `Tab`.

3. In **PORT INTERFACE**, type `NMI` and choose **NM.NetworkHandle: NMI_PncRequest**.



*Figure 10-10. ① **NAME**, ② **PORT INTERFACE**, ③ **NM.NetworkHandle: NMI_PncRequest**.*

4. Save with `Ctrl+S`. The port becomes a circle, the mark of an NM port.

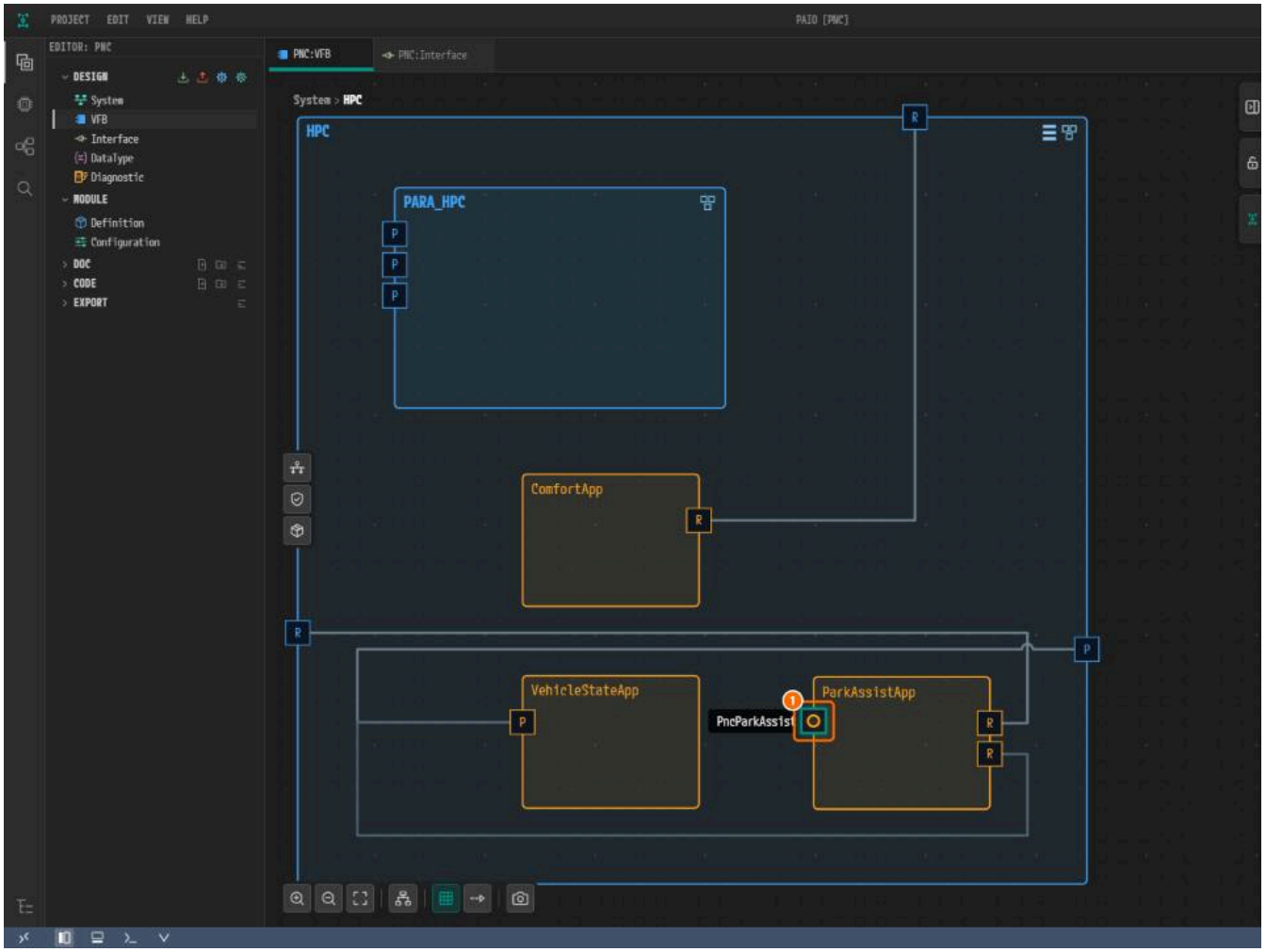


*Figure 10-11. ① `PncParkAssist`, ② its **PORT INTERFACE**.*

Mapping the NetworkHandle

1. Double-click **ParkAssistApp**. The **AA Internal Behavior [ParkAssistApp]** dialog opens.

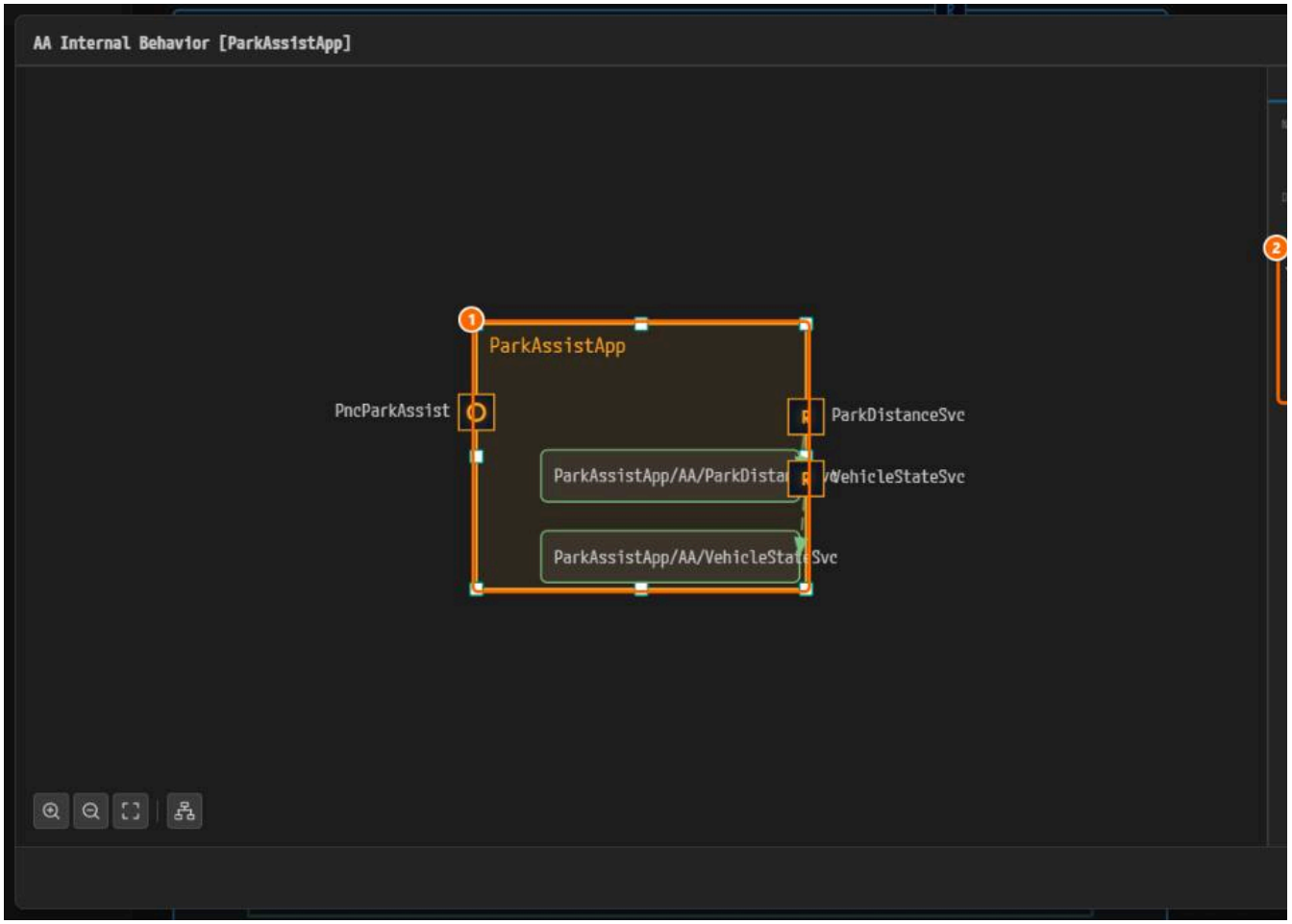


*Figure 10-12. ① The AA box with the port `PncParkAssist`, ② **PORTS**.*

2. Right-click an empty spot inside the AA box and choose **Add Instance** > **ara::nm** > **Network Handle**.

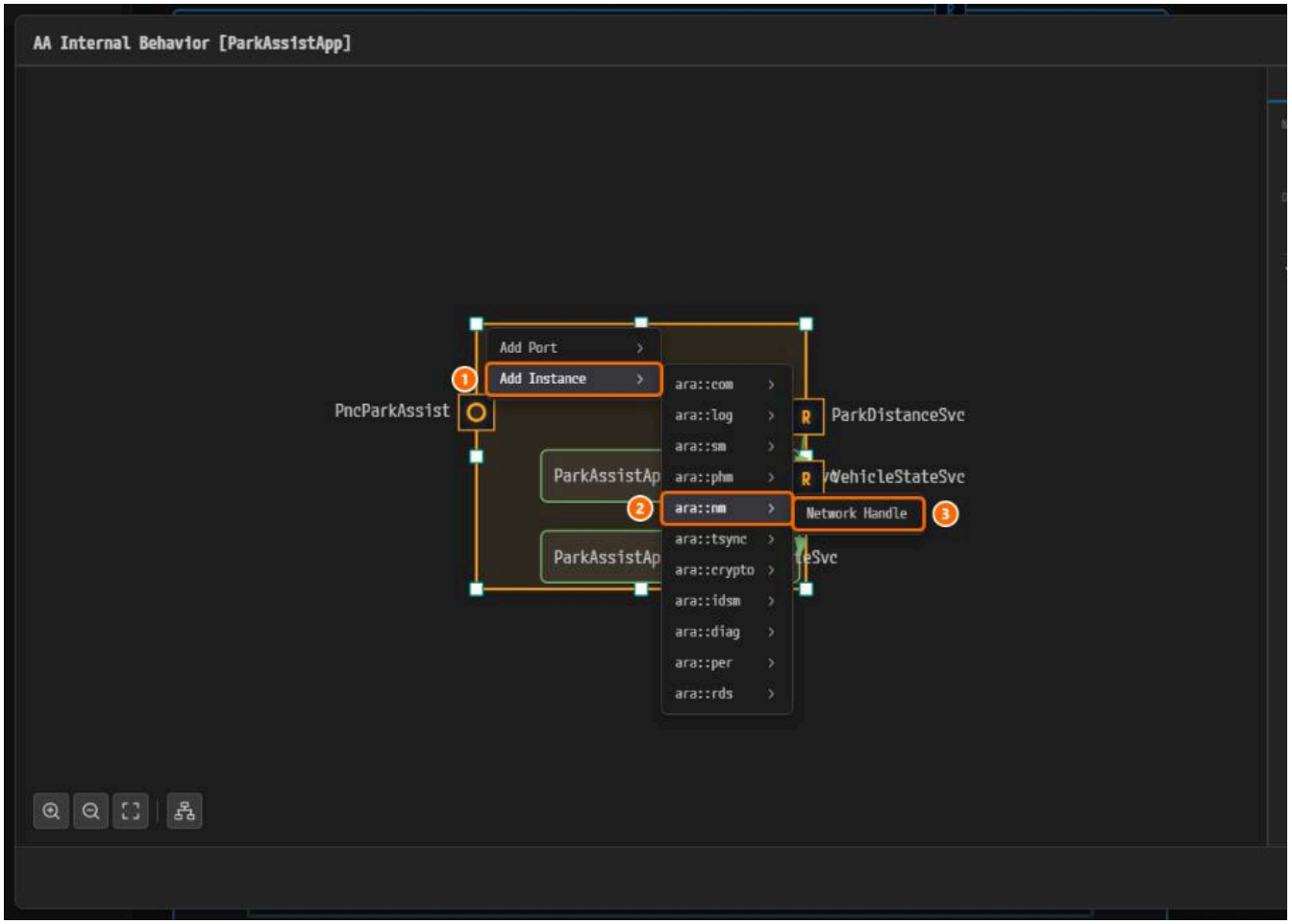


*Figure 10-13. ① **Add Instance**, ② **ara::nm**, ③ **Network Handle**.*

3. In the **NETWORK HANDLE MAPPING** panel, pick `PncParkAssist` in **PORT**.

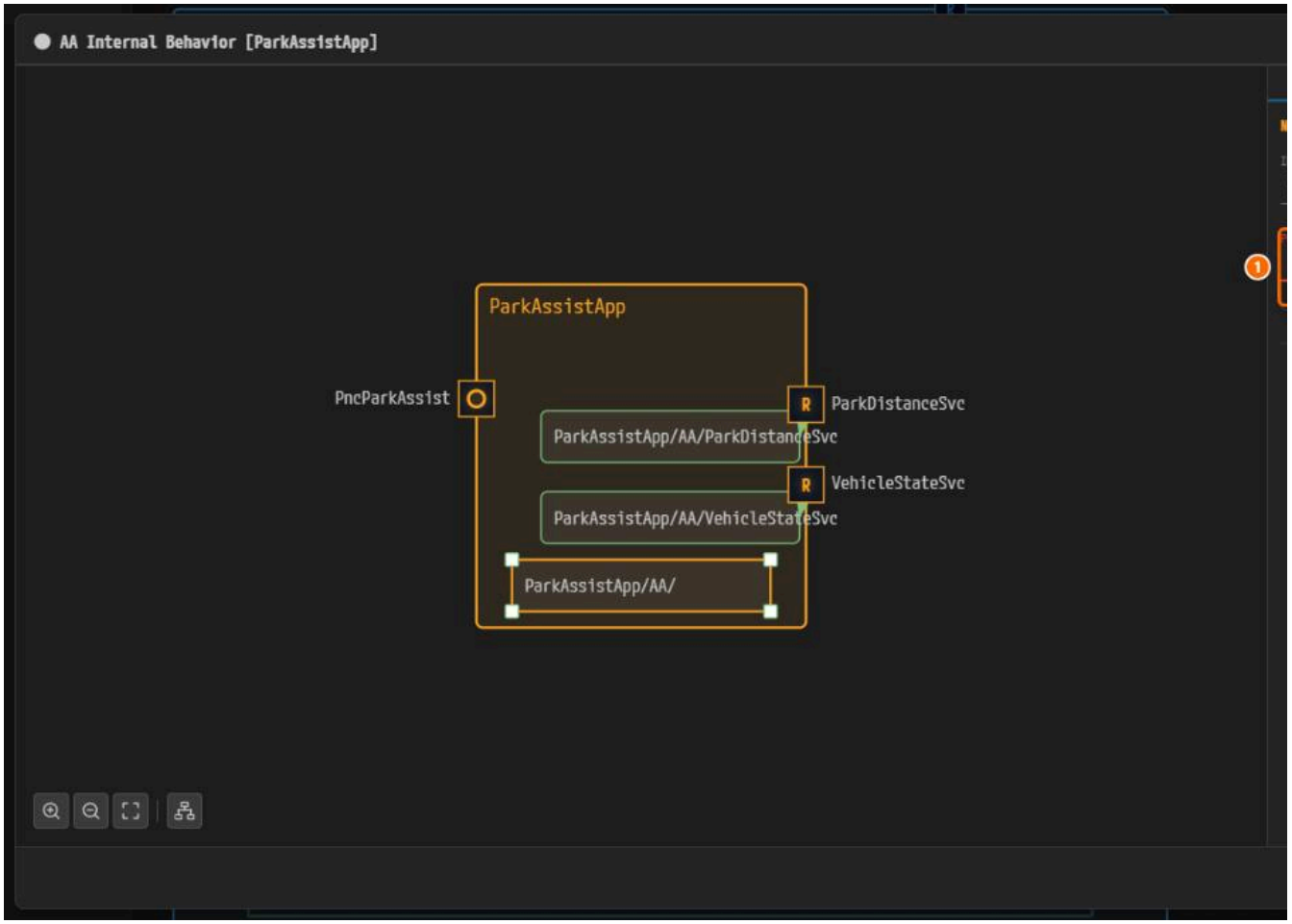


*Figure 10-14. ① **PORT**, ② `PncParkAssist`.*

4. Pick `NH_ParkAssist` in **NETWORK HANDLE**.

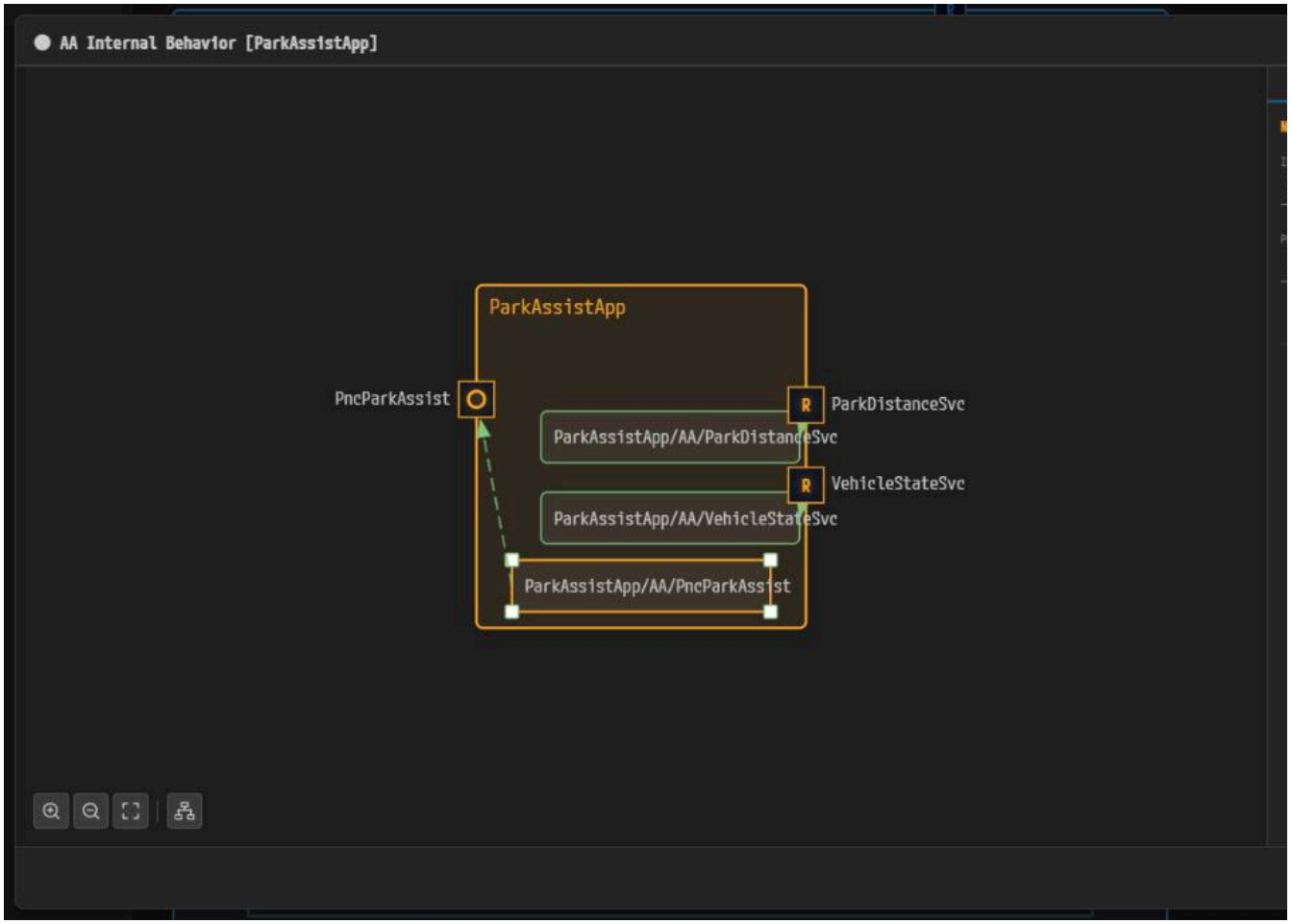


*Figure 10-15. ① **NETWORK HANDLE**, ② `NH_ParkAssist`.*

5. Save with `Ctrl+S`.

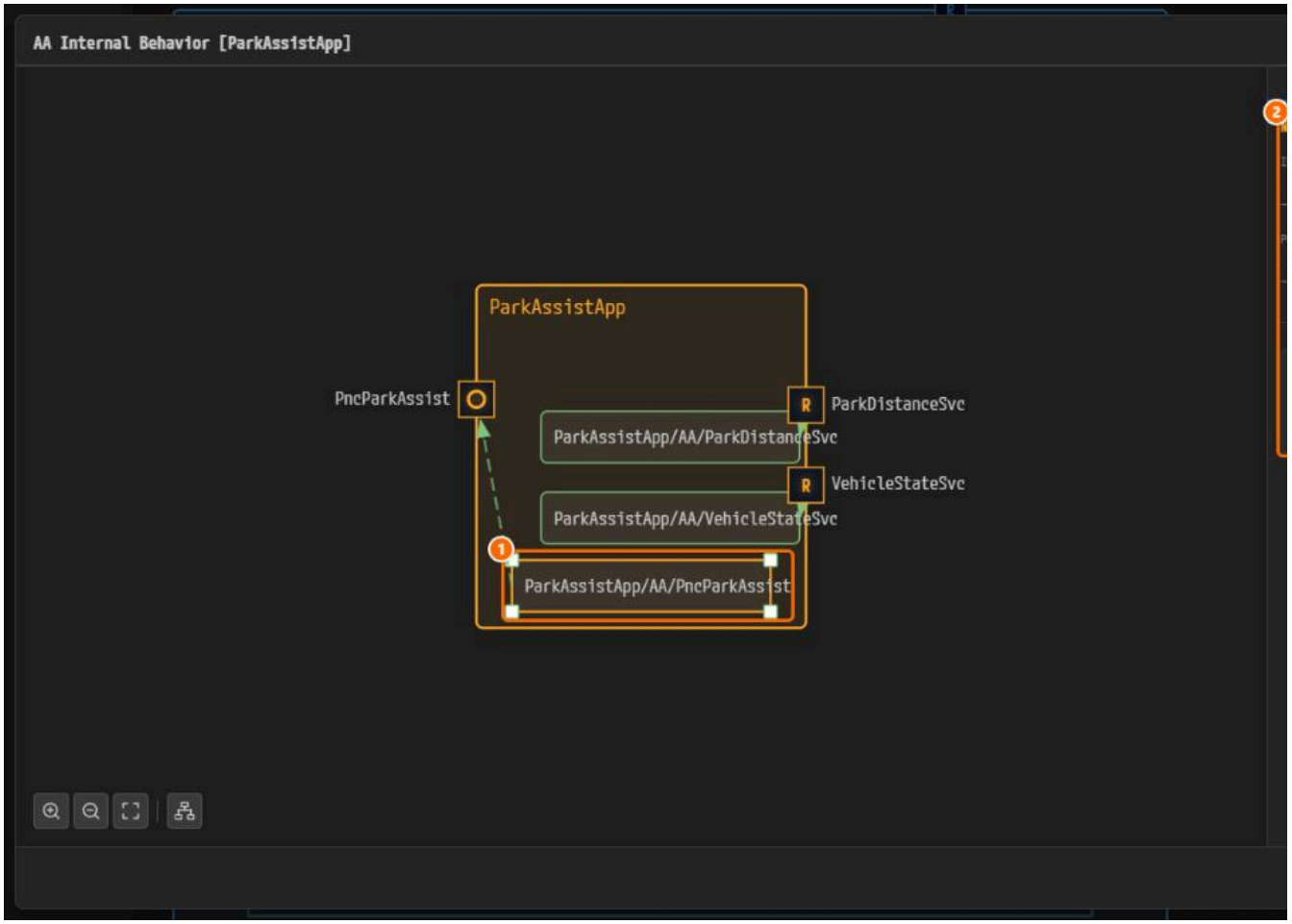


Figure 10-16. ① The mapping `ParkAssistApp/AA/PncParkAssist`, ② its values and the handle summary.

> **Caution.** The **NM CLUSTER** line of the summary shows `NmCluster_VLAN_Body (Backbone)`, though `NH_ParkAssist` belongs to `NmCluster_VLAN_F` **PARTIAL NETWORKS** instead.

6. Click **Close**.

Connecting the Remaining AAs

Do the same for `VehicleStateApp` and `ComfortApp`: add the P-Port, then the NetworkHandle mapping. Every port uses **PORT INTERFACE** `NM.NetworkHand`

AA (node label / prototype)	P-Port NAME	NETWORK HANDLE	Saved mapping name
ParkAssistApp / `ParkAssist`	`PncParkAssist`	`NH_ParkAssist`	`ParkAssistApp_AA_PncParkAssist`
VehicleStateApp / `VehicleState`	`PncTrailer`	`NH_Trailer`	`VehicleStateApp_AA_PncTrailer`
ComfortApp / `Comfort`	`PncComfort`	`NH_Comfort`	`ComfortApp_AA_PncComfort`

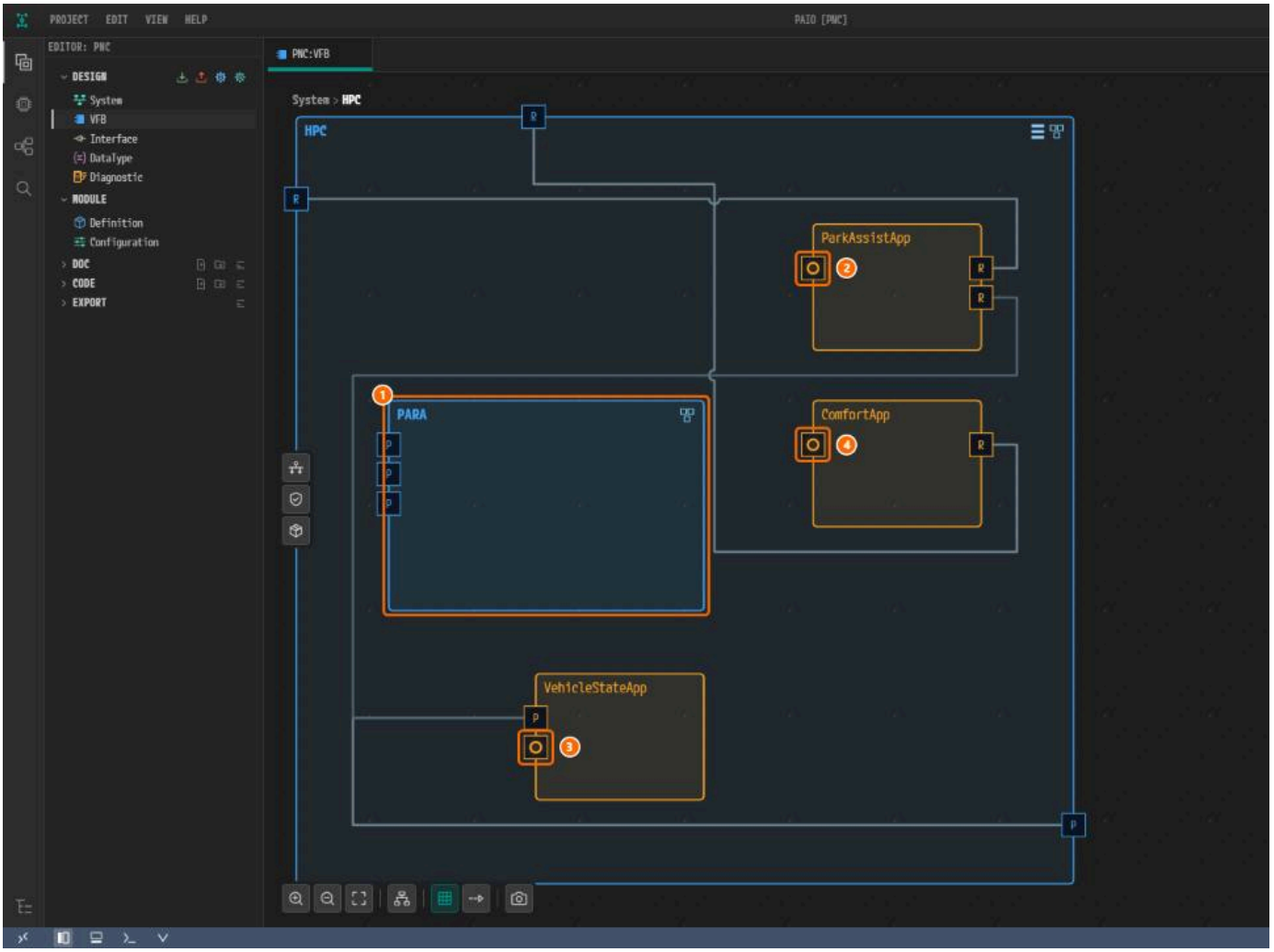


*Figure 10-17. The finished example. ① `PARA`, ② `PncParkAssist`, ③ `PncTrailer`, ④ `PncComfort`. The example uses an older PARA, so its node reads **PARA** .*

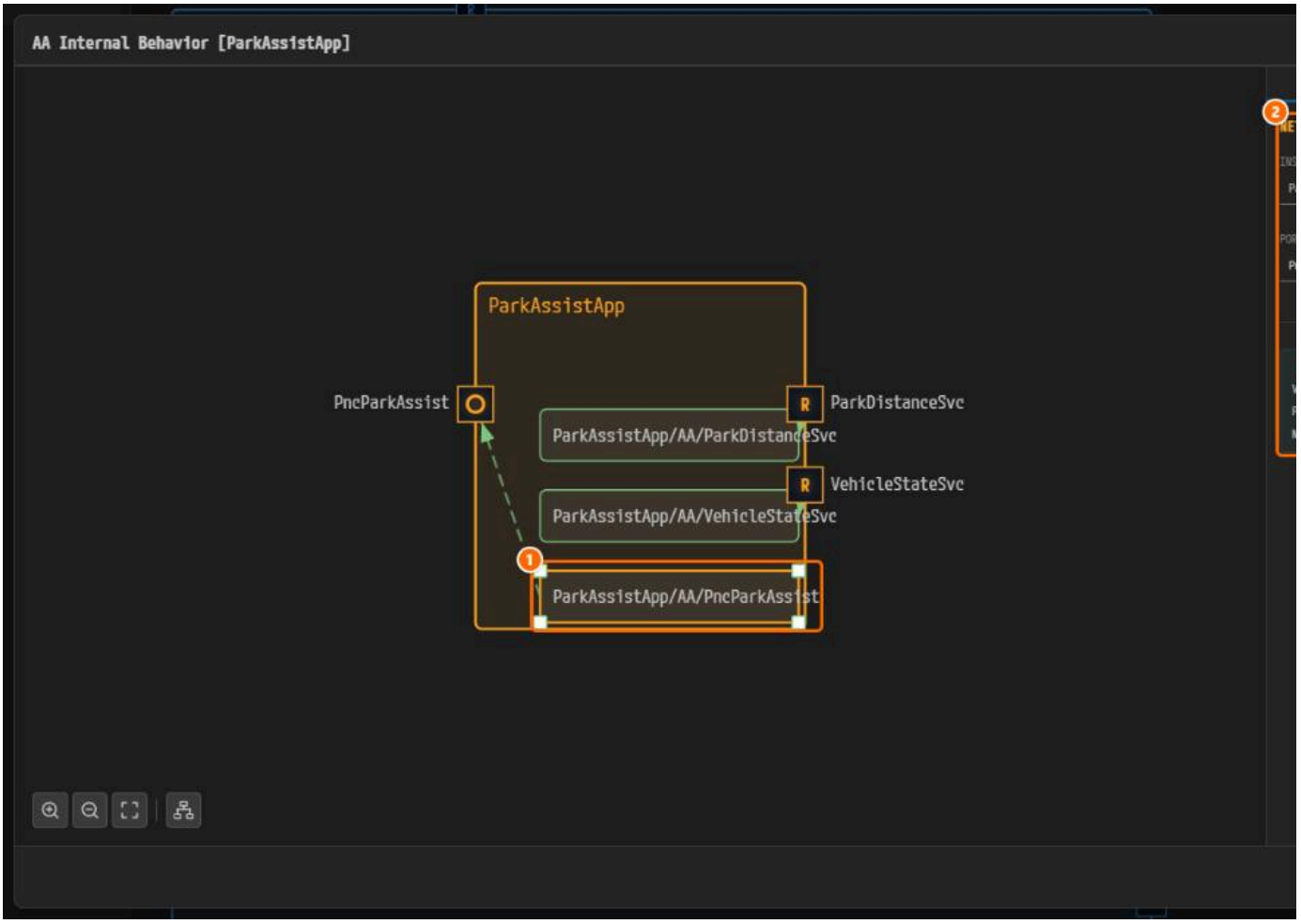


*Figure 10-18. **AA Internal Behavior [ParkAssistApp]** in the finished example. ① The NetworkHandle mapping, ② its values. It must match Figure 10-16.*

What Gets Exported

What you created in this chapter	Exported element
`NMI_PncRequest`	`NETWORK-MANAGEMENT-PORT-INTERFACE` in the interface document
The three NetworkHandle mappings of the application AAs	`NETWORK-HANDLE-PORT-MAPPING` in the AA Executable Manifest

Why Is It Designed This Way?

- **Two decisions.** The NM network handles of Chapter 9 decide which PNC on which VLAN; the mappings of this chapter decide which application reque
- **The application sees only the handle.** It requests and releases it through its ara::nm P-Port, never a VLAN or a PNC number.
- **The import is a merge.** It connects the machine modules to the platform AAs (`SM`, `NM`) and keeps the machine configuration of the previous chapter

Checklist

- [ ] **Network Management** of the HPC machine still has `NH_Comfort`, `NH_ParkAssist` and `NH_Trailer`.
- [ ] The HPC contains **PARA_HPC**, not overlapping another node.
- [ ] The **ADAPTIVE** tab has `NMI_PncRequest` with **C++ NAMESPACE** `pnc::nm`.
- [ ] Each application AA has its circular P-Port and a NetworkHandle mapping as in Connecting the Remaining AAs.
- [ ] No tab or dialog title shows a modified mark.

Next Step

In The System Editor and Validation you validate the design and fix the reported findings.

System Editor and Validation

What You Will Do

In this chapter you check the design on the SYSTEM panels of the System and VFB editors. Then you open the **VALIDATION** panel, fix the errors it shows, a

Before You Begin

- Start after finishing PARA Platform Import and PNC Request Ports.
- Figures 11-4 to 11-12 show a draft in which the SOME/IP consumer ports collided with the providers. If you followed the earlier chapters, you have no procedure.
- For differences in your own project, see Appendix — Differences Between Your Project and the Finished Example.
- The full reference is Validation and System Editor in the PAIO manual.

Viewing the SYSTEM Panels

1. In the **EDITOR** view, click **System** under **DESIGN** (tab `PNC:System`). Click the **System** button (the PAIO logo), the third button on the right edge of the c

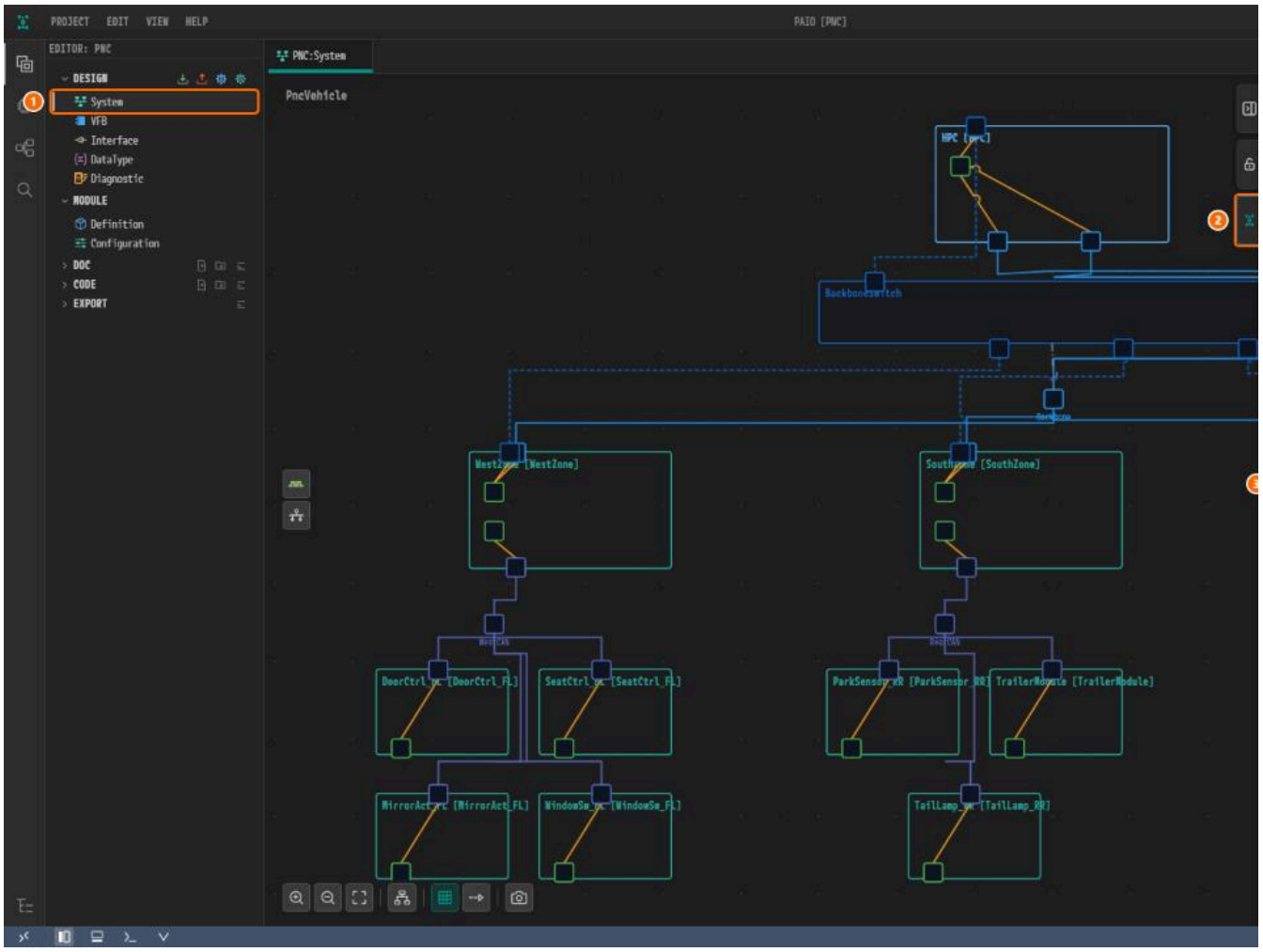


*Figure 11-1. ① **System** under **DESIGN**, ② the **System** button, ③ **ECU MAPPING**.*

2. Check that **ECU MAPPING** has 14 rows, each node mapped to the composition of the same name.

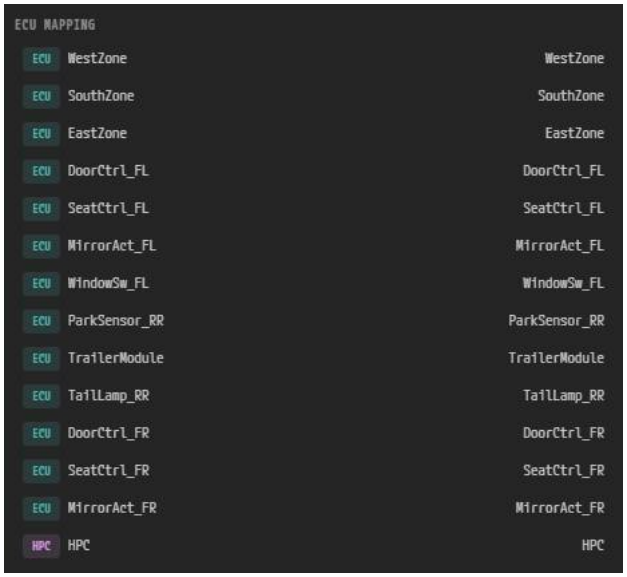


*Figure 11-3. **ECU MAPPING**: 13 **ECU** rows and the **HPC** row.*

3. Click **VFB** under **DESIGN** (tab `PNC:VFB`) and click the **System** button of this canvas.

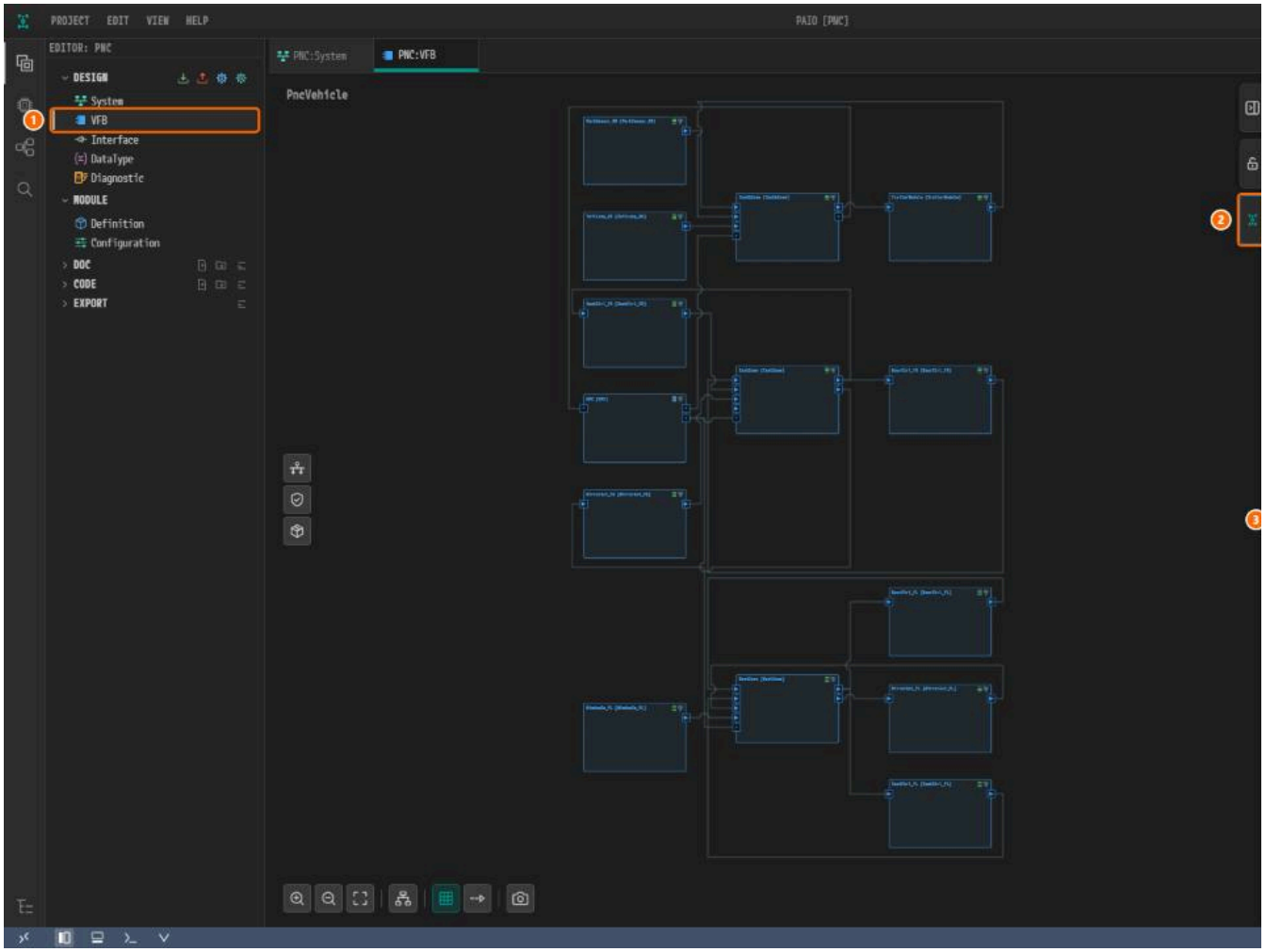


*Figure 11-17. ① **VFB**, ② the **System** button, ③ **COMPOSITIONS**, shown after step 4.*

4. In **COMPOSITIONS**, collapse the 14 compositions under the root `PncVehicle`.

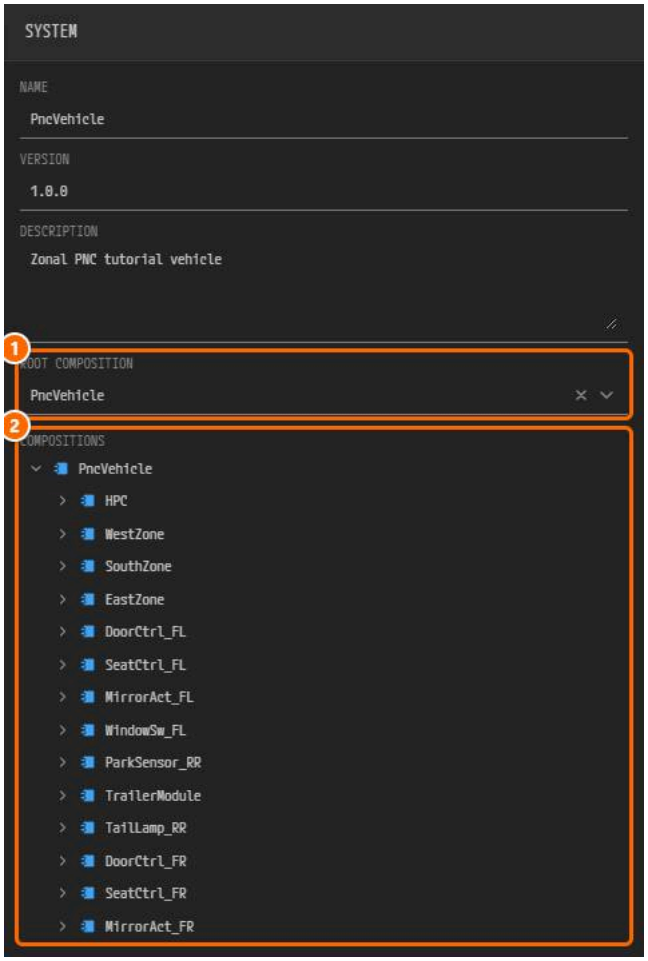


*Figure 11-2. ① **ROOT COMPOSITION** `PncVehicle`, ② **COMPOSITIONS**.*

> **Note.** `PARA` lies inside `HPC`, so it has no row of its own in **ECU MAPPING**.

Opening the Validation Panel

Validation has no run button. PAIO checks the project when it opens and again after every edit.

1. Find the validation icon (`V`, tooltip **Toggle Validation**) at the left end of the status bar. A red badge shows the error count.

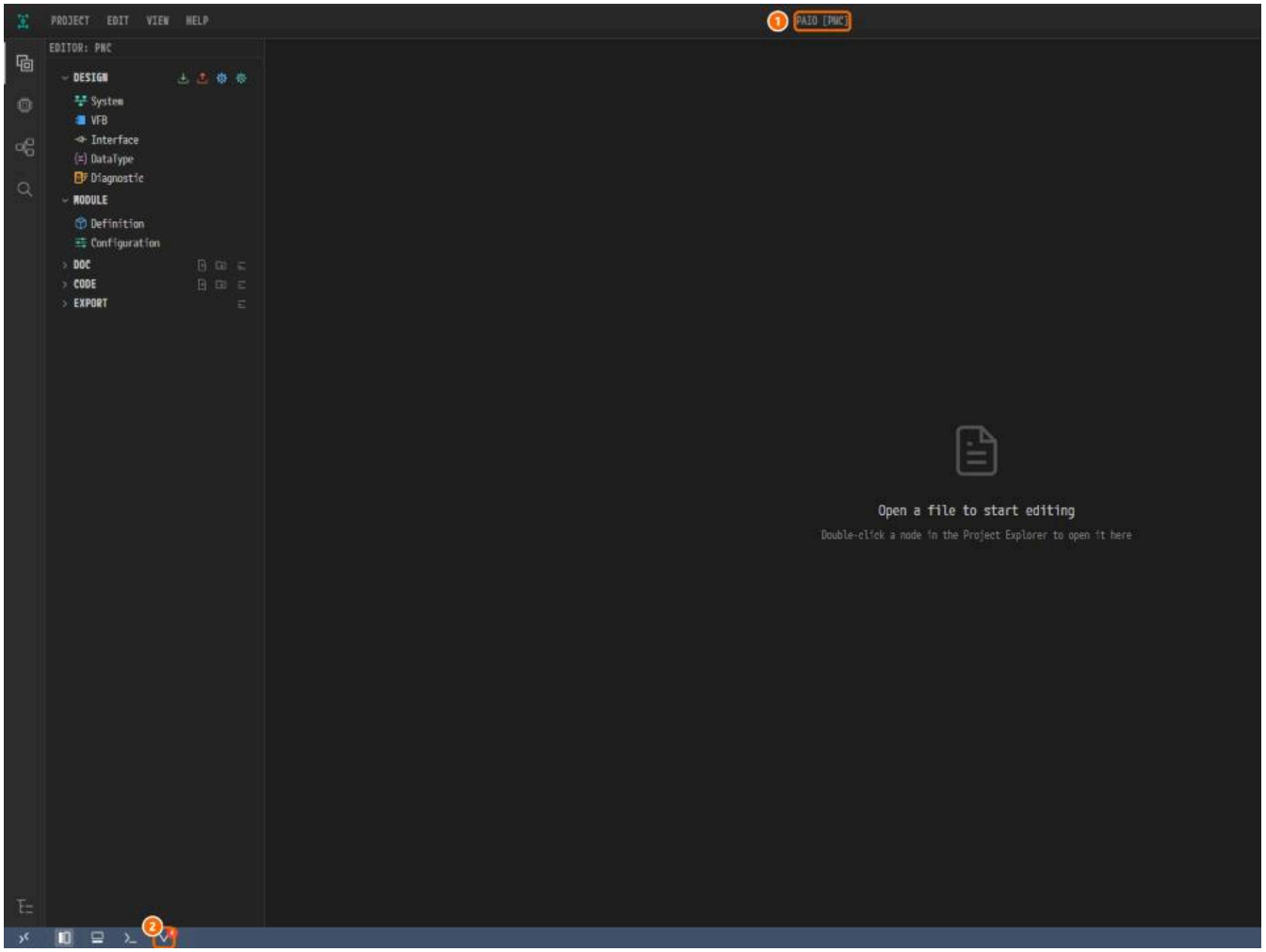


Figure 11-4. The draft state. ① The title bar `PAIO [PNC]`, ② the validation icon with the badge `4`.

2. Click the icon. The **VALIDATION** panel opens below the editor.

Reading the Validation Panel

Each row is one violation. Click a row to jump to the element.

Column	Contents
ELEMENT TYPE	The kind of element, such as `SomeipServiceInstanceToMachineMapping`, `PhysicalChannel`, `PncMapping`
NAME	The element name
DESCRIPTION	What is wrong
CONSTR	The AUTOSAR constraint, such as `constr_5093`. Empty for a PAIO rule



*Figure 11-5. Four errors of a PAIO rule (empty **CONSTR**). ① The `VehicleState_Consumer` row.*

Finding and Fixing an Error: Duplicate SOME/IP Ports

In the draft, two consumers used the same UDP port as the provider on their connector. The fix moves them to `30503`.

Connector (IP address)	Instance (role)	Draft UDP port	After the fix
`HPC_Eth_Rear_Conn` (`192.168.200.1`)	`VehicleState_Provider` (P)	`30501`	`30501` (unchanged)
`HPC_Eth_Rear_Conn` (`192.168.200.1`)	`ParkDistance_Consumer` (C)	`30501`	**`30503`**
`SouthZone_Eth_Conn` (`192.168.200.12`)	`ParkDistance_Provider` (P)	`30502`	`30502` (unchanged)
`SouthZone_Eth_Conn` (`192.168.200.12`)	`VehicleState_Consumer` (C)	`30502`	**`30503`**

Finding the Element from an Error Row

1. Click the `VehicleState_Consumer` row. The `PNC:Interface` tab opens with the card marked in red.

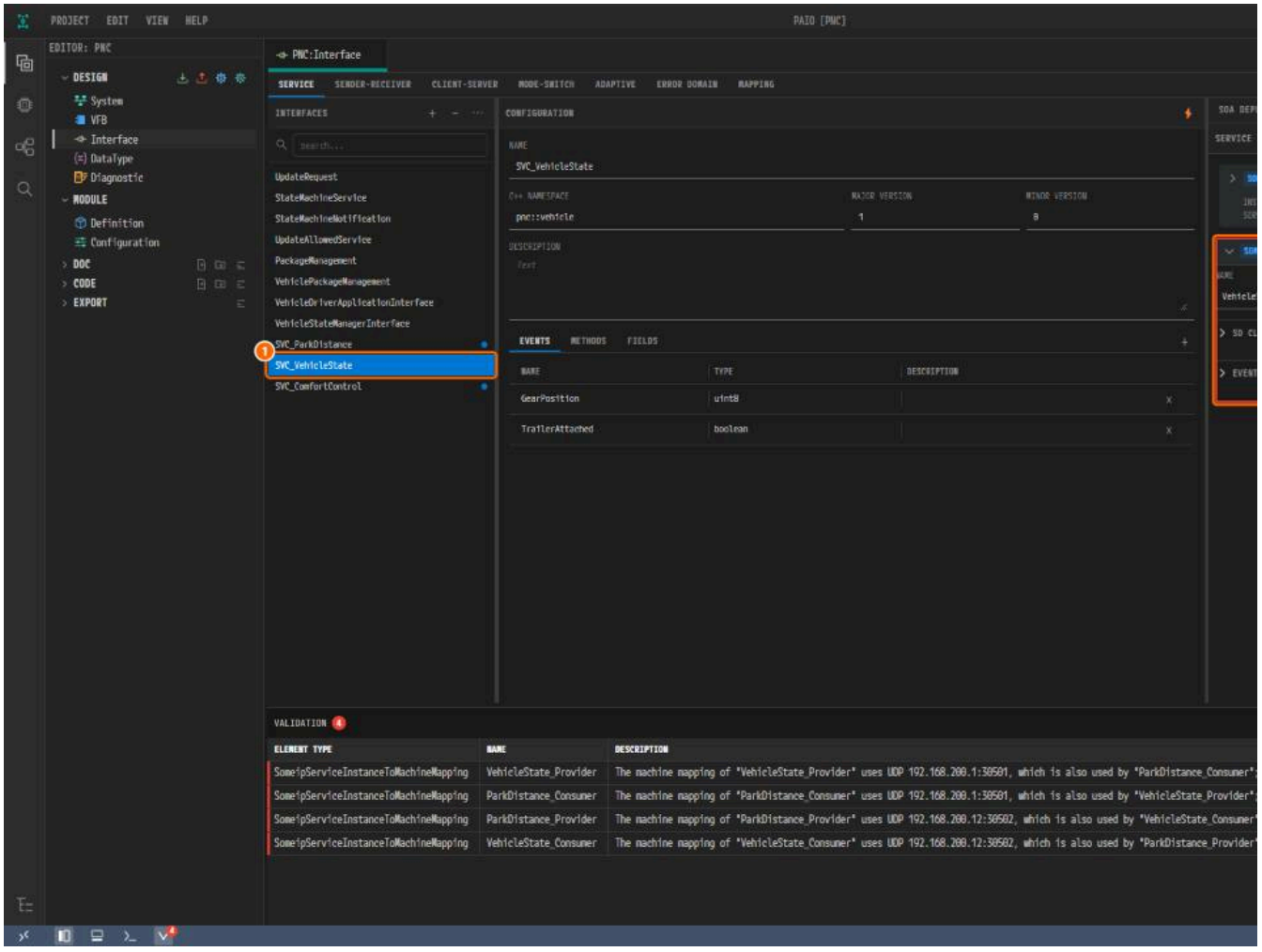


Figure 11-6. ① `SVC_VehicleState`, ② the `VehicleState_Consumer` card.

The card has no port field. The port is set on the connector in the System editor.

Fixing the SouthZone_Eth_Conn Port

1. Open the `PNC:System` tab and click the Ethernet connector above `SouthZone` (**NAME** `SouthZone_Eth_Conn`).
2. Under **SERVICE INSTANCES**, click `VehicleState_Consumer` to expand it.

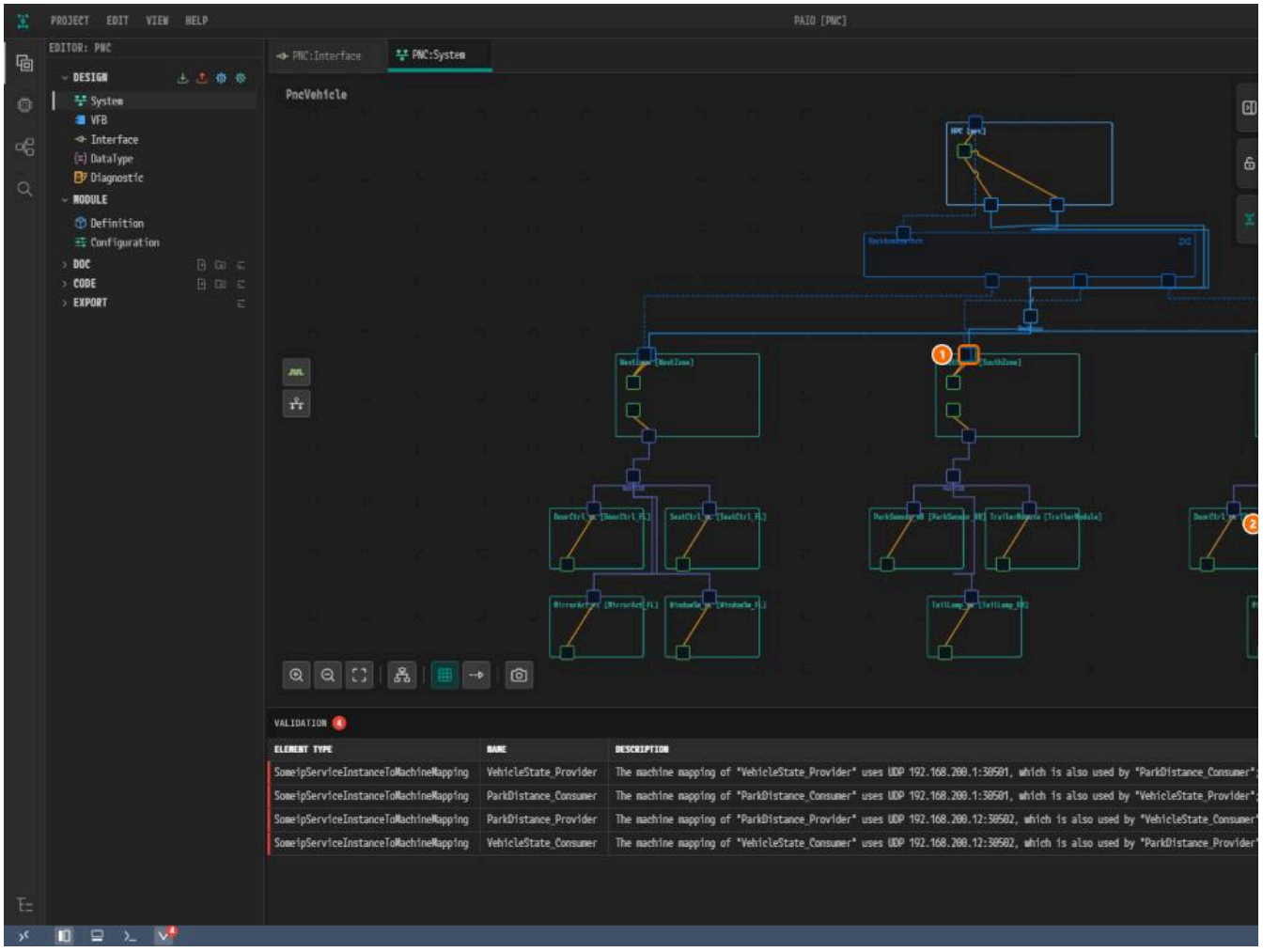


*Figure 11-7. ① `SouthZone_Eth_Conn`, ② **SERVICE INSTANCES**, both on `UDP:30502`, ③ **UDP PORT** `30502`.*

3. Type `30503` in **UDP PORT** and press **Tab**. The badge drops to `2`.

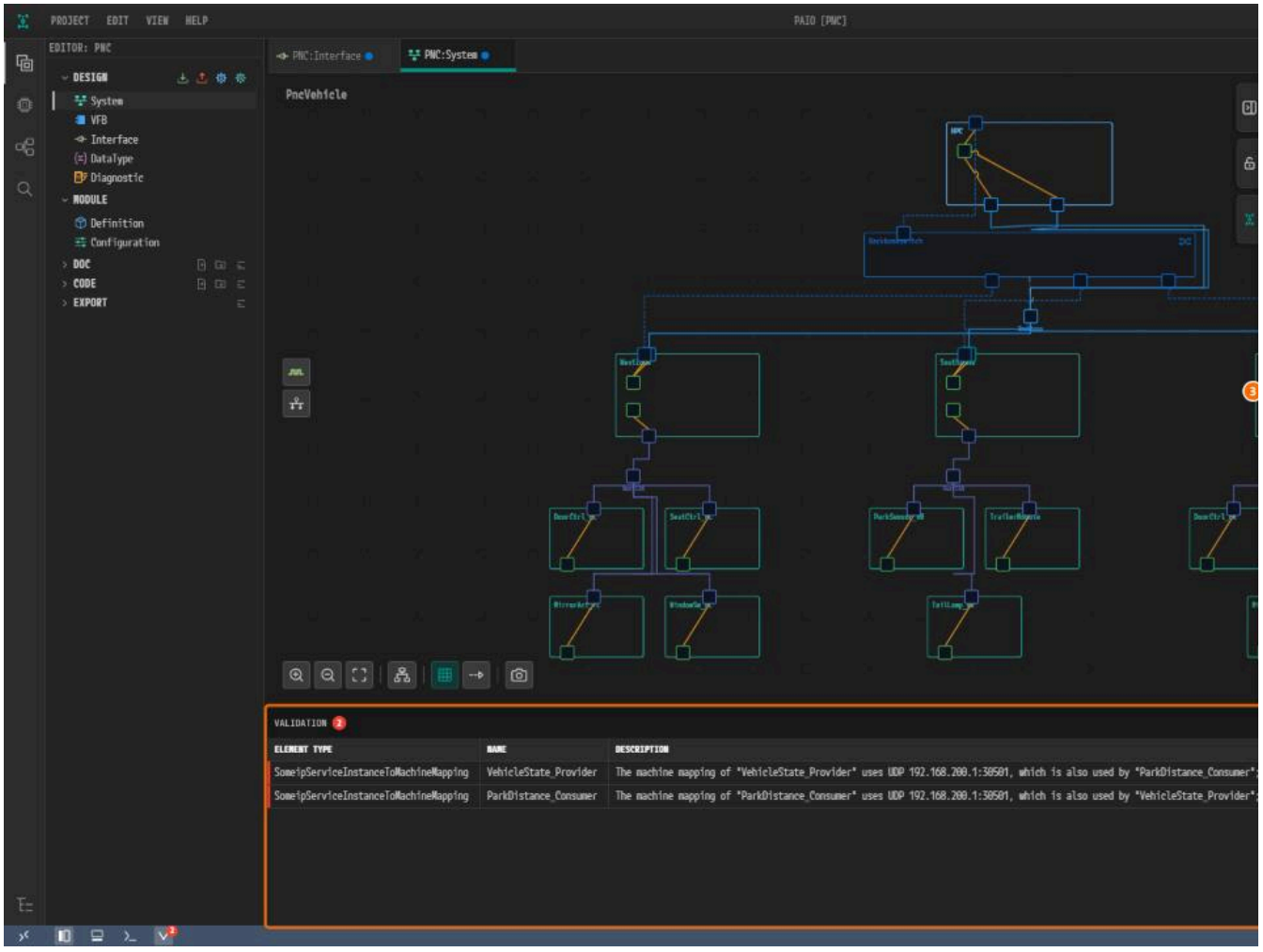


*Figure 11-10. ① **UDP PORT** `30503`, ② the **VALIDATION** panel with `2`, ③ the new endpoint `...:30503`.*

The port change also adds derived interfaces to `PNC:Interface`, so that tab gets ● too.

4. Press `Ctrl+S`, then choose **PROJECT** > **Save All** to save `PNC:Interface` as well.

Fixing the HPC_Eth_Rear_Conn Port

1. Click the `ParkDistance_Consumer` row.



Figure 11-11. ① `ParkDistance_Consumer`.

2. In `PNC:System`, click the right-hand connector on the lower edge of `HPC` (**NAME** `HPC_Eth_Rear_Conn`, **CONNECTED CHANNEL** `Backbone / VLAN_Rear`

3. Expand `ParkDistance_Consumer`, type `30503` in **UDP PORT** and press **Tab**. The panel shows "No validation issues.".

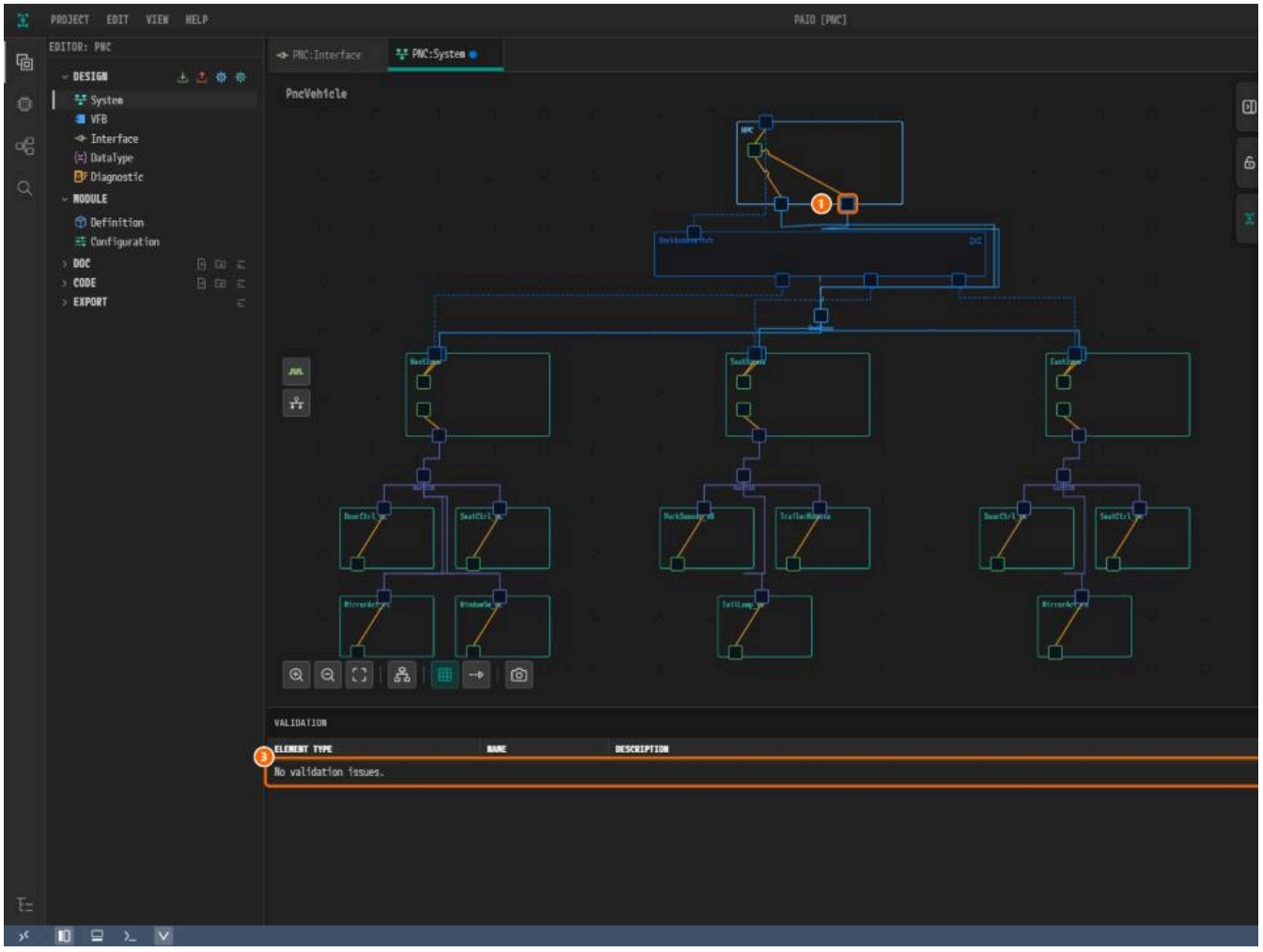


*Figure 11-12. ① `HPC_Eth_Rear_Conn`, ② **UDP PORT** `30503`, ③ "No validation issues.". The summary line reads `UDP:30501` until you select the connecte*

4. Press `Ctrl+S`.

Re-validating the Saved Result

1. Saving re-checks the project. Confirm "No validation issues." and no badge on the status bar icon.

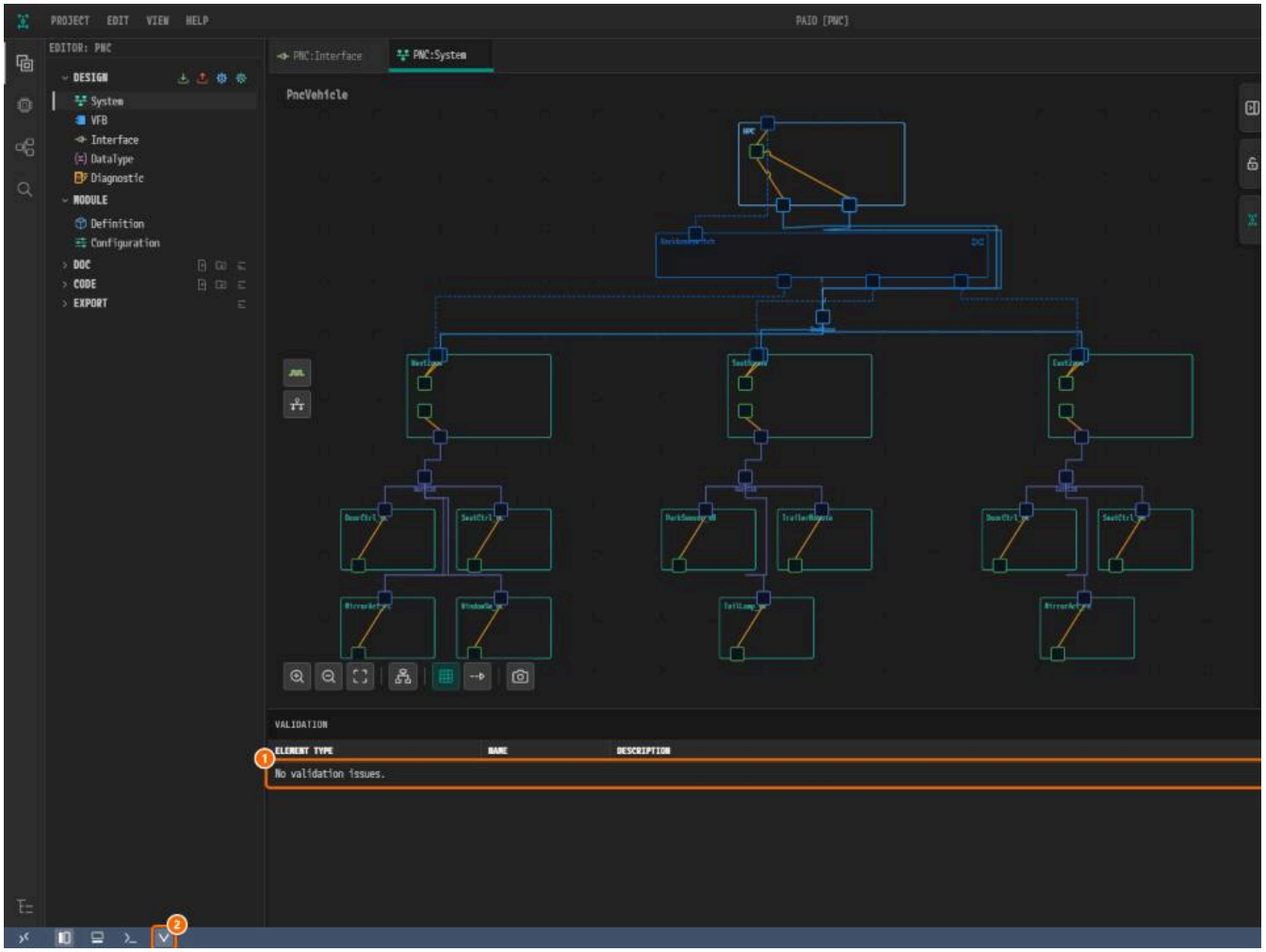


Figure 11-13. ① "No validation issues.", ② the icon without a badge.

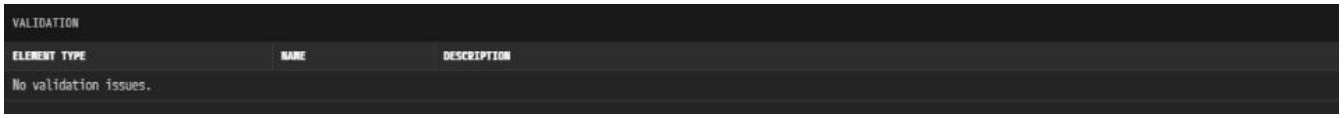


*Figure 11-14. The **VALIDATION** panel with zero errors.*

Checking Live Validation: PNC GATEWAY TYPE

Validation reacts to unsaved edits as well. Change one setting, watch the errors, and revert it. Do not save.

1. In `PNC:System`, click the connector above `EastZone` (**NAME** `EastZone_Eth_Conn`). Under **PARTIAL NETWORK**, **PNC GATEWAY TYPE** is **PASSIVE**.
2. Click **ACTIVE**. Two errors appear.

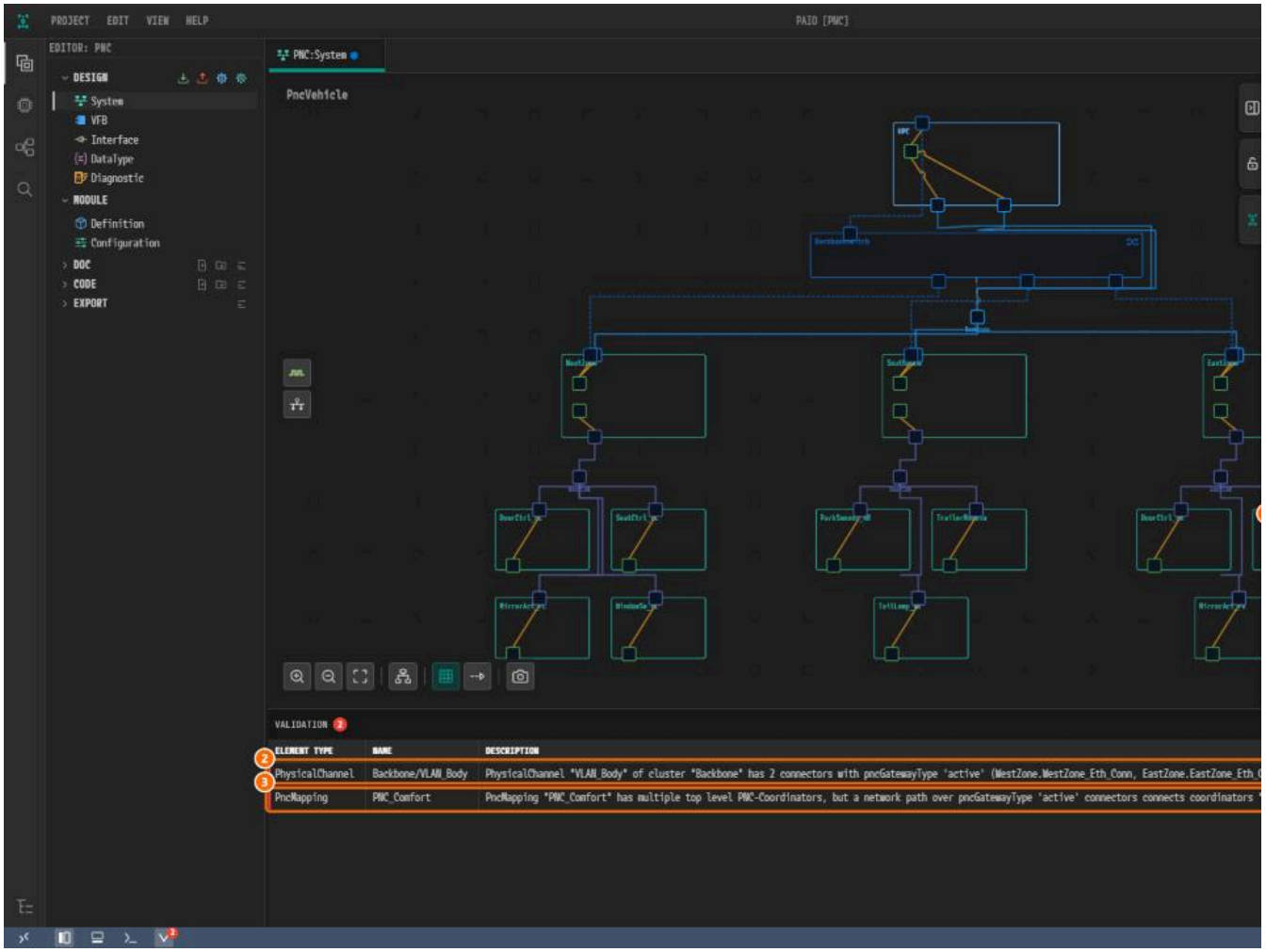


*Figure 11-15. ① **ACTIVE**, ② `constr_5093`: two active connectors on `VLAN_Body`, ③ `constr_3714`: the coordinators `WestZone` and `EastZone` of `PNC_(*

3. Click **PASSIVE**. Both errors disappear, but the tab keeps ●.

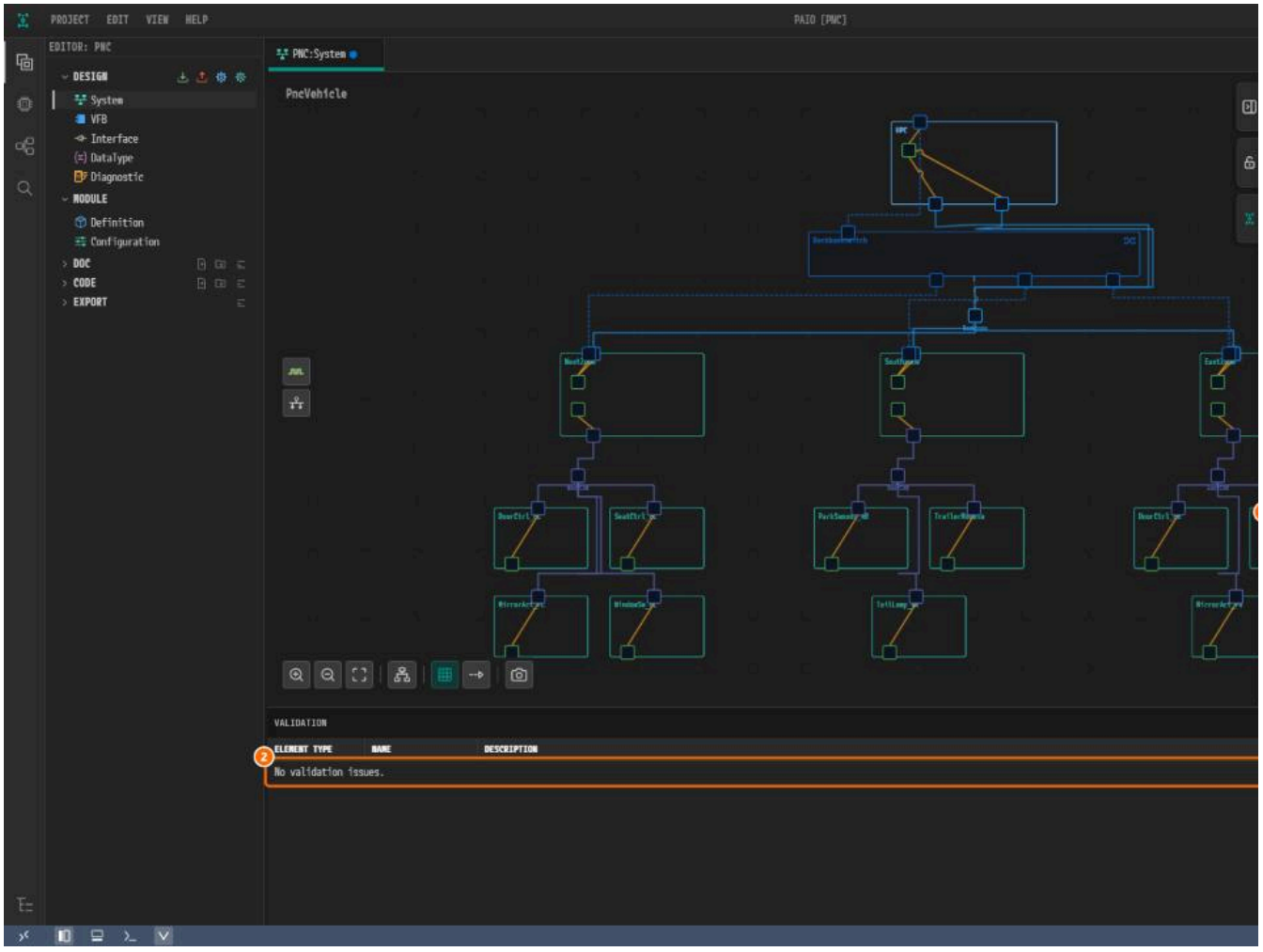


*Figure 11-16. ① **PASSIVE**, ② "No validation issues.".*

4. Close the `PNC:System` tab with its **×** and click **Don't Save**.

Key Constraints Related to PNC Design

The finished example satisfies every constraint below. These are the findings reported while the example was first built. If one appears, review the relate

Constraint	What the validation panel reported	Correction
`constr_3040`	PNC identifiers 8/9/10 lie outside the PNC vector bit range (16..63)	Changed the identifiers to 17/18/19
`constr_3686`	The filter mask is 3 bytes, which differs from PNC VECTOR LENGTH 6	Recalculated it to 6 bytes
`constr_5093`	The backbone channel has three active gateway connectors	Split the backbone into `VLAN_Body` and `VLAN_Rear`, with one active connector pe
`constr_3714` / `constr_3781`	Top-level coordinators are connected over an active path / no coordinator exists	`WestZone` and `SouthZone` are active on both sides (top-level), and `EastZone` is p backbone side
`constr_3514`	One I-signal (such as `LockCmd`) is mapped to two PDUs	Split the left and right signals into `_FL` and `_FR`
`constr_9157` – `9159`, `constr_9162`	Mandatory properties of the CanNm cluster are missing	Filled in `nmBusloadReductionActive`, `nmImmediateNmTransmissions`, `nmMessage` `nmRemoteSleepIndicationTime` on the **CAN NM CLUSTER** tab
`constr_5222`	Mandatory properties of the UdpNm cluster are missing	Filled in `nmMessageTimeoutTime` and `nmRemoteSleepIndicationTime` on the **UDP** `NmCluster_VLAN_Rear` cannot be opened from the screen
`constr_9165` / `9166`	Mandatory flags of the NM cluster coupling are missing	Couplings cannot be edited from the screen; their values come with an ARXML imp
(PAIO NM rule)	The NetworkHandle has no INACTIVE-to-FunctionGroup state mapping ("PNC release stops nothing")	Added the Off mappings for `ParkAssistFG` and `TrailerFG`
`constr_10641`	The max-action-list-duration of the SM module is missing	Set it to 1.0 s

Why Is It Designed This Way?

- **Validation does not block.** Errors stop neither saving nor export.
- **Each finding is still a real fault.** It turns into wrong behavior once another tool, such as a ComM/Nm generator, reads the exported ARXML, so aim for
- **The schema check is separate.** The ARXML schema (XSD) is checked at export.

Checklist

- [ ] **ECU MAPPING** lists 14 nodes, and **COMPOSITIONS** shows 14 compositions under `PncVehicle`.
- [ ] The **VALIDATION** panel shows "No validation issues." and the status bar icon has no badge.
- [ ] `ParkDistance_Consumer` on `HPC_Eth_Rear_Conn` and `VehicleState_Consumer` on `SouthZone_Eth_Conn` are on `UDP:30503`.
- [ ] **PNC GATEWAY TYPE** of `EastZone_Eth_Conn` is **PASSIVE**, and no tab carries ●.

Next Step

In ARXML Export you export the validated design as ARXML and inspect the files.

ARXML Export

What You Will Do

You export the validated design as AUTOSAR ARXML, find where the PNC design is recorded, and check the schema check results. Four of the five files rep
that come from the export itself (12 in the finished example).

Before You Begin

- Finish The System Editor and Validation so that the **VALIDATION** panel reads "No validation issues."
- Save every tab with `Ctrl+S` so that no tab shows the ● marker.
- The export needs an active license and a server connection. For details, see Import and Export.

File	PNC-related content
`PNC_System.arxml`	Three `<PNC-MAPPING>` elements, `<PNC-VECTOR-LENGTH>` `6` and `<PNC-VECTOR-OFFSET>` `2` (once on the SYSTEM), the SWC-to-ECU mapping
`PNC_COM.arxml`	ECU instances and the machine design: connector PNC properties, clusters and VLANs, 20 `<I-SIGNAL-I-PDU-GROUP>` elements, NM settings, SOME/IP
`PNC_SWC.arxml`	SWC types and the `<PORT-GROUP>` elements of the VFCs, the three `<NM-NETWORK-HANDLE>` elements, the three `<NETWORK-HANDLE-PORT-MAPPING>` e
`PNC_Interface.arxml`	Port and service interfaces, including one `<NETWORK-MANAGEMENT-PORT-INTERFACE>`
`PNC_DataType.arxml`	Data types. No PNC-specific element

The counts are those of the finished example. A hand-built project gives some different counts, such as `I-SIGNAL-I-PDU-GROUP`; see Appendix — Differe
Example.

Exporting the Project to ARXML

1. In the **EDITOR** view, point at the second icon on the right of the **DESIGN** header. Its tooltip reads **Export**.

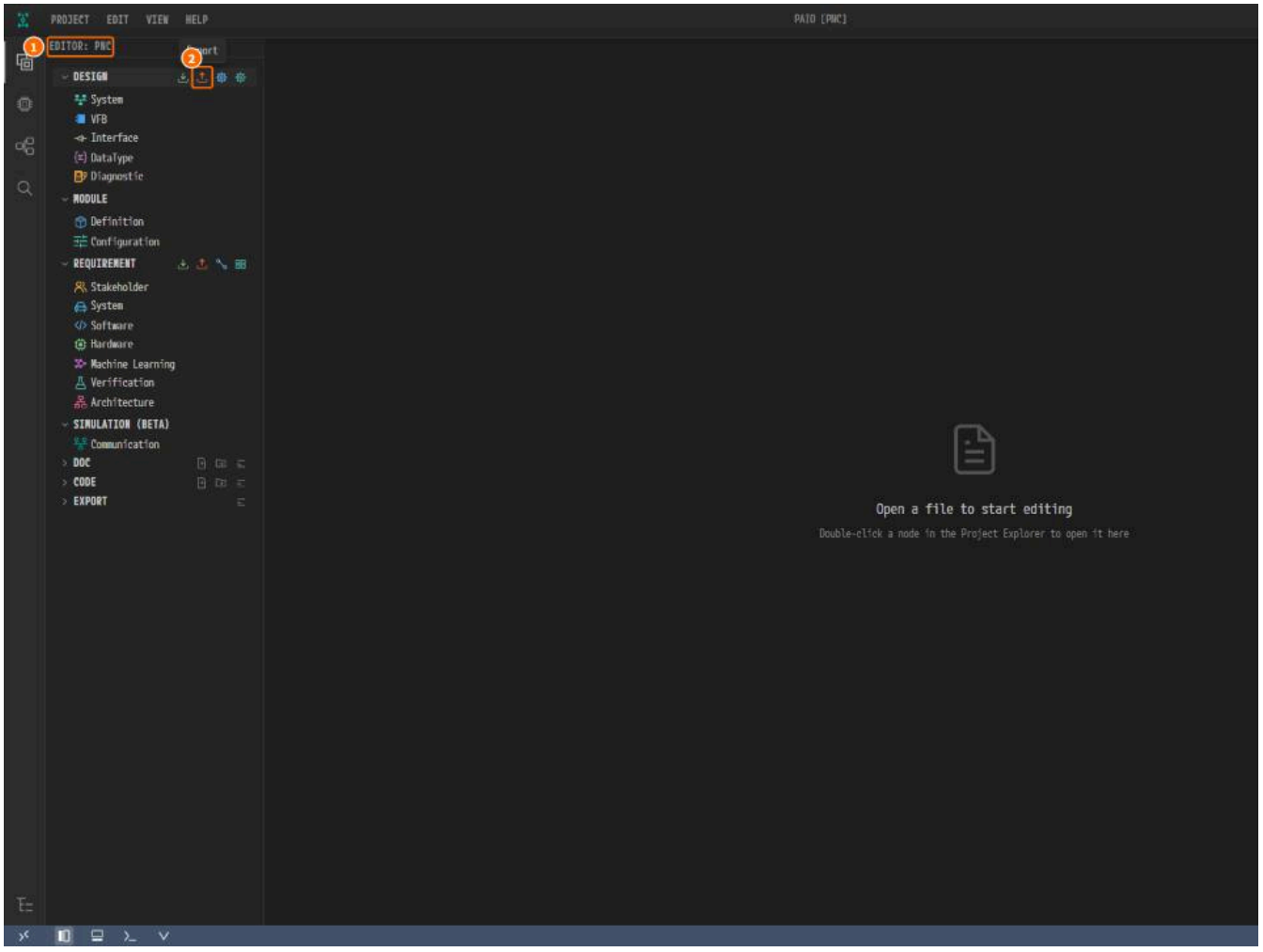


*Figure 12-1. ① `EDITOR: PNC`, ② **Export**.*

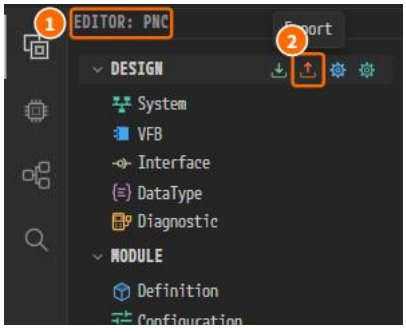


*Figure 12-2. ① `EDITOR: PNC`, ② **Export**. The other icons are **Import**, **PRTE Code Generation** and **PARA Code Generation**.*

2. Click **Export**. The **Export** dialog opens. Leave **FOLDER** at `export` and **FORMAT** at **ARXML (R25-11)**, and click **Export**.



*Figure 12-3. ① **FOLDER**, ② **FORMAT**, ③ **Export**.*

3. The dialog shows `✓ Complete!` and the line **ARXML** `R25-11, 298 elements`. The element count depends on your model. Click **Close**.

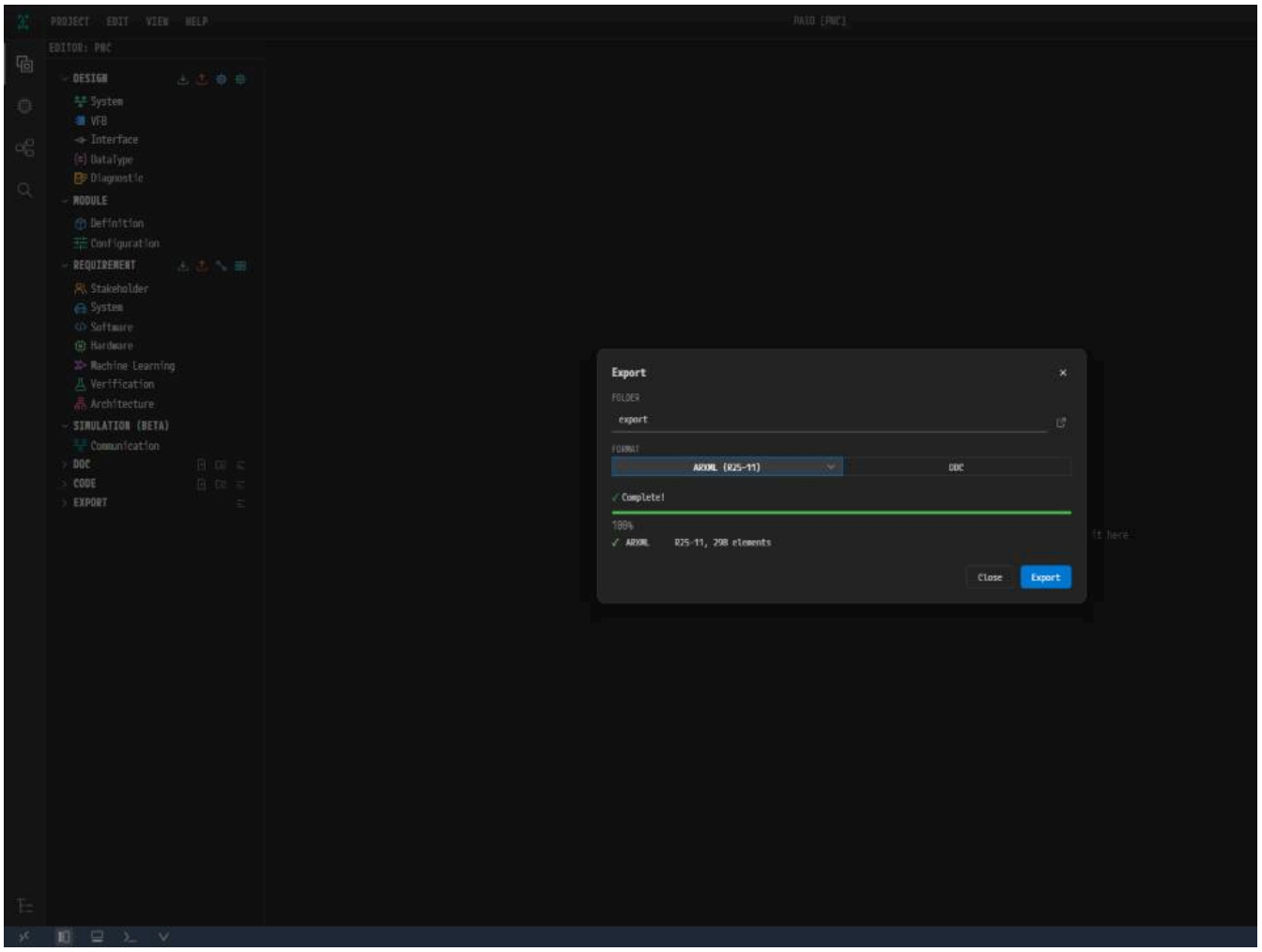


Figure 12-4. The whole window when the export finishes.

> **Caution.** If the export fails, `X Finished with errors` appears and the **ARXML** line shows the reason. `OFFLINE_EXPORT_DENIED` means a server conne problem.

Checking the Generated Files

1. Click the **EXPORT** header in the **EDITOR** view. It lists the five files `PNC_COM.arxml`, `PNC_DataType.arxml`, `PNC_Interface.arxml`, `PNC_SWC.arxml` anc

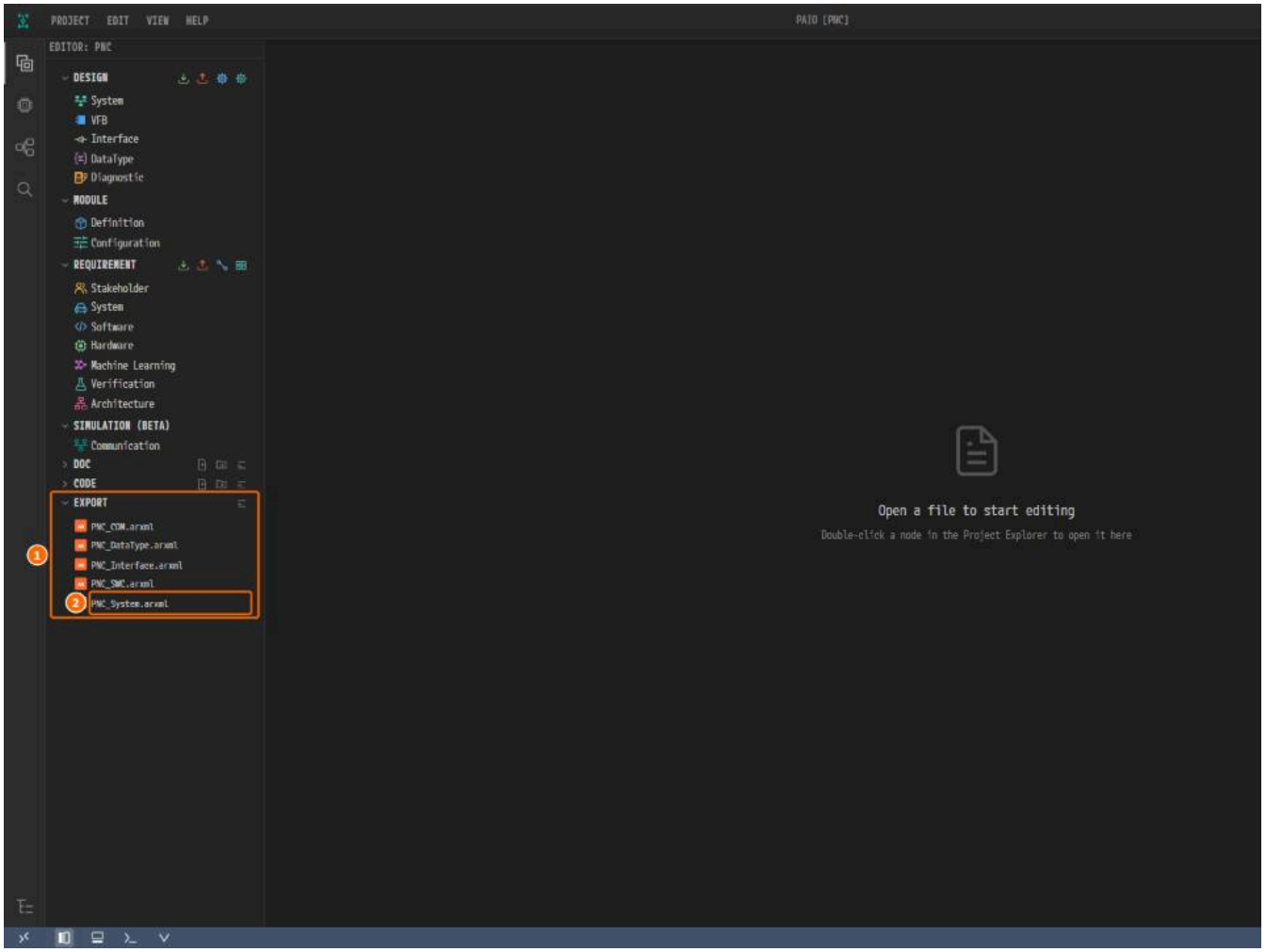


*Figure 12-5. ① **EXPORT** and the five files, ② `PNC_System.arxml`.*

Reading PNC-MAPPING in PNC_System.arxml

1. Double-click **PNC_System.arxml** under **EXPORT**. The `PNC:PNC_System.arxml` tab opens.
2. Press `Ctrl+F` and type `<PNC-MAPPING>`. The find box shows `1 of 3` and jumps to the mapping `PNC_Comfort`.

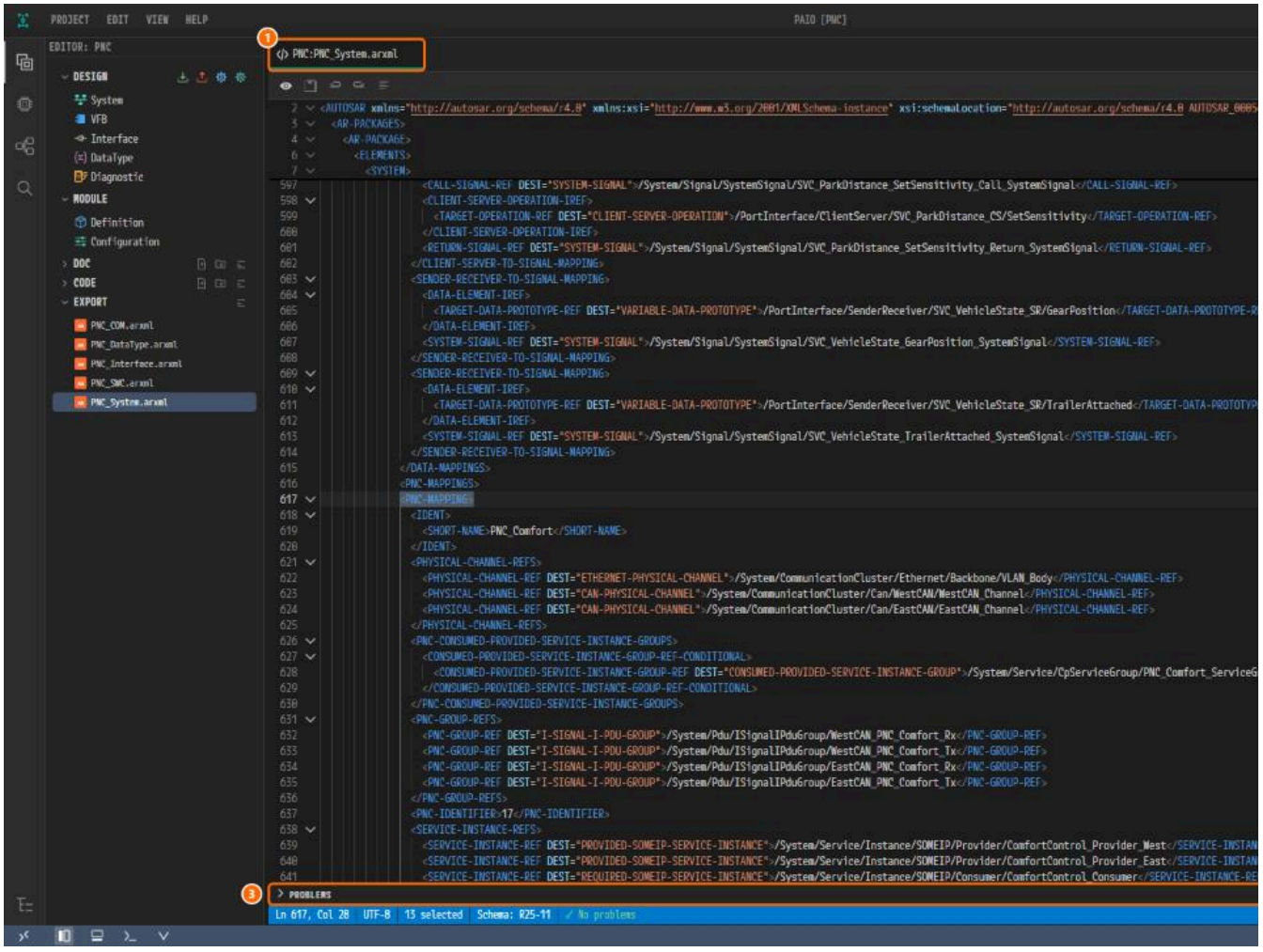


*Figure 12-6. ① `PNC:PNC_System.arxml`, ② the find box with `1 of 3`, ③ the **PROBLEMS** panel. The status bar shows `Schema: R25-11` and `✓ No pro*

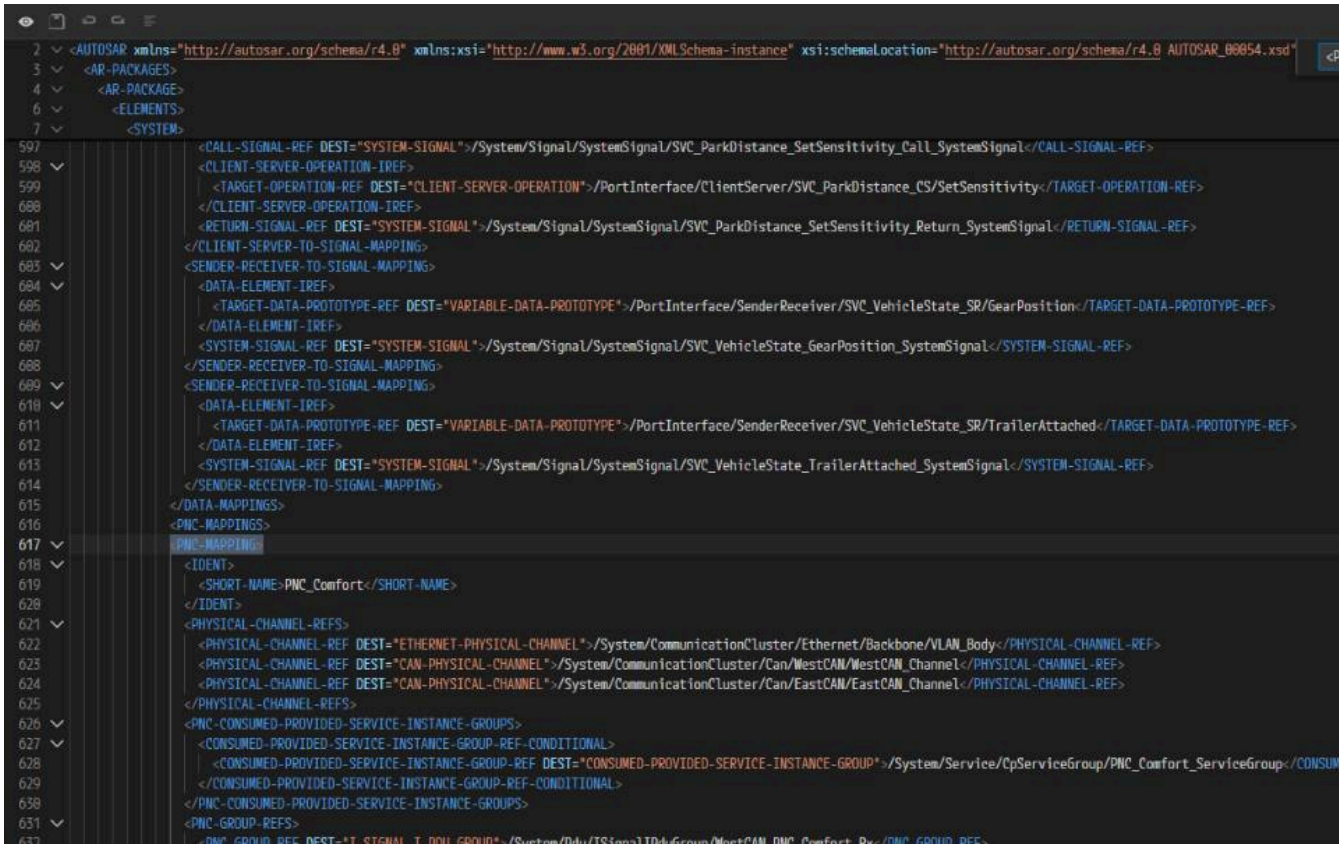


Figure 12-7. The first `<PNC-MAPPING>` block, `PNC_Comfort`.

3. Match the block against the chapters that built it.

Element	Value	Chapter
`PHYSICAL-CHANNEL-REFS`	`VLAN_Body`, `WestCAN_Channel`, `EastCAN_Channel`	Topology Design
`PNC-IDENTIFIER`	`17`	PNC Definition and Connector PNC Properties
`PNC-GROUP-REFS`	Four I-PDU Groups	Signals, PDUs, Frames, and I-PDU Groups
`SERVICE-INSTANCE-REFS`	Three SOME/IP instances	Interfaces and SOME/IP Services
`VFC-IREFS`	`VFC_Comfort` (`PORT-GROUP`)	VFB and VFC
`WAKEUP-FRAME-REFS`	`FT_WestCAN_NmFrame_DoorCtrl_FL`, `FT_EastCAN_NmFrame_DoorCtrl_FR`	Signals, PDUs, Frames, and I-PDU Groups

4. Use ↓ in the find box to see `PNC_ParkAssist` and `PNC_Trailer`.

Checking the Schema Check Results

The XML editor checks a file against the AUTOSAR XSD when it opens it.

1. Click the **PROBLEMS** header under the editor. For `PNC_System.arxml` the table shows only `No problems found`.

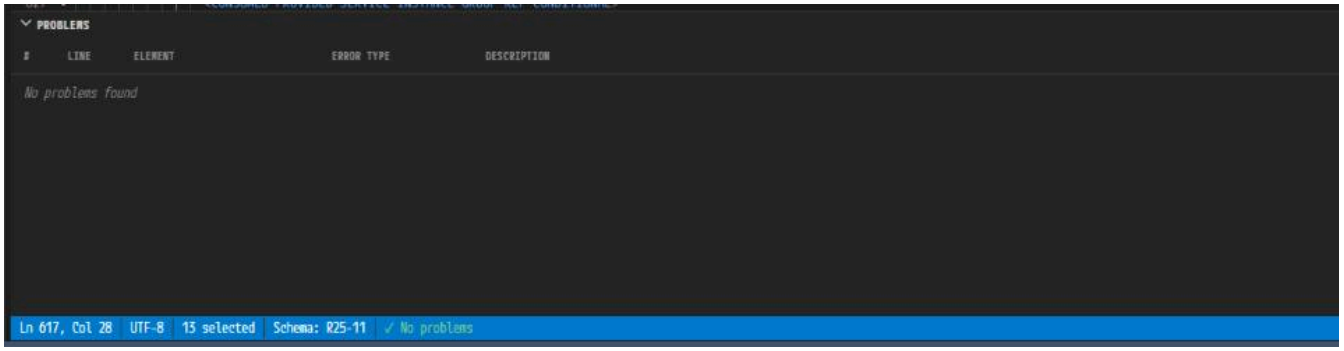


*Figure 12-8. **PROBLEMS** with `No problems found`, and `Schema: R25-11` · `✓ No problems` in the status bar.*

2. Open the other four files the same way. `PNC_DataType.arxml`, `PNC_Interface.arxml` and `PNC_SWC.arxml` show zero errors. `PNC_COM.arxml` opens `errors` in the status bar.

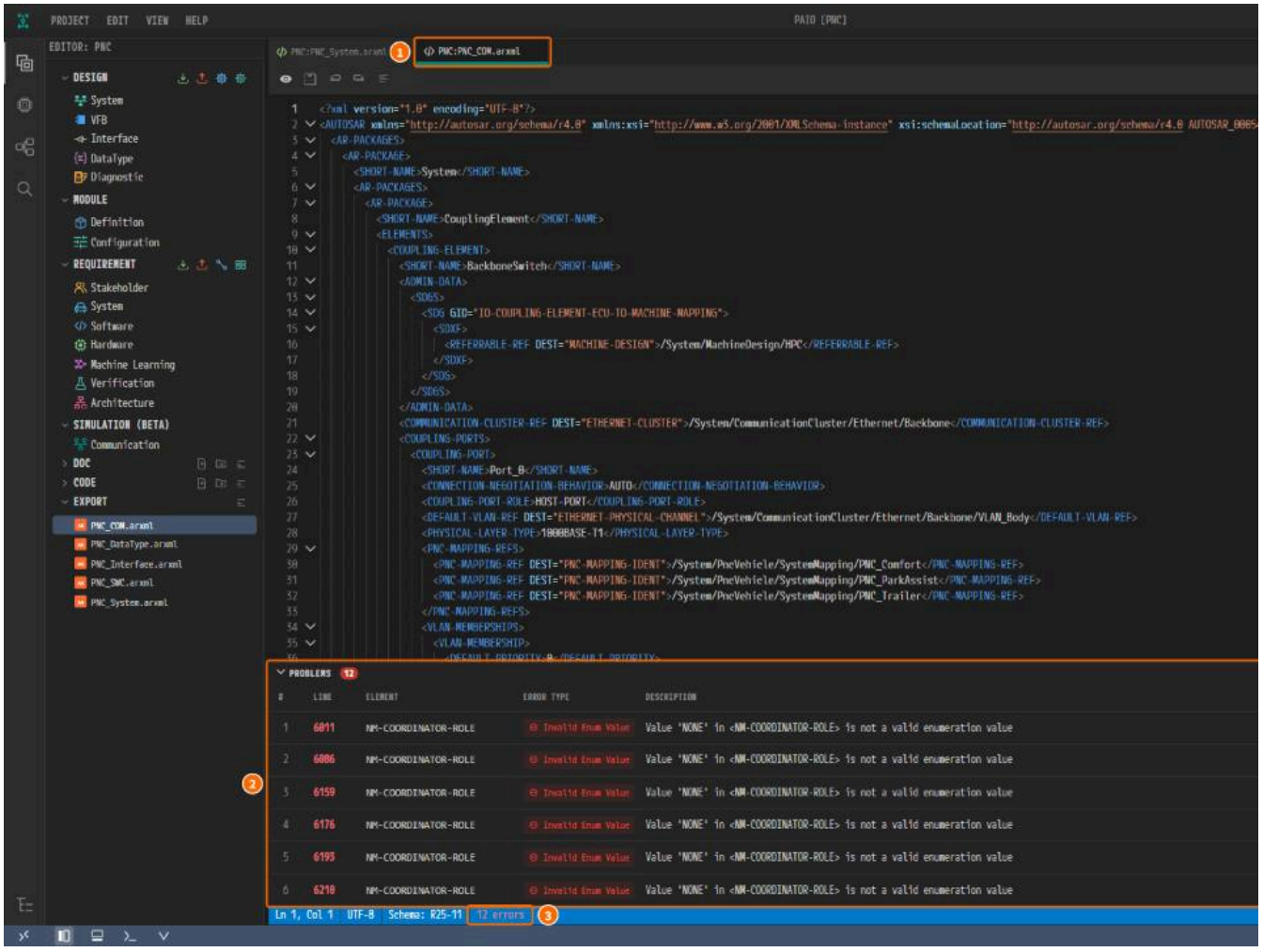


*Figure 12-9. ① `PNC:PNC_COM.arxml`, ② **PROBLEMS** with the badge `12`, ③ `12 errors`.*

> **Note.** The 12 errors come from the export, not from your input. They are `NM-COORDINATOR-ROLE` written as `NONE` for the NM nodes whose **NM COORDIN** schema allows only `ACTIVE` and `PASSIVE`.

Verifying by Importing Again

This round-trip check is optional.

1. Create an empty project with **PROJECT** > **New Project...**. It opens in a new window.
2. Click **Import**, the first icon on the right of the **DESIGN** header.
3. Select all five ARXML files in `PNC/export`.
4. When the import finishes, open **System** under **DESIGN** and check that the three PNCs, the identifiers `17` / `18` / `19` and the connectors' **PNC GATEWAY**

> **Caution.** Importing into the same project merges instead of overwriting, and different values open the **Conflict Resolution** dialog.

Why Is It Designed This Way?

- **Five files by role.** Data types and interfaces are shared definitions, SWC is the software structure, COM is the ECU and network deployment, and Syst
- **The PNC design is split.** Which channels, PDU groups, service instances and VFCs a PNC binds is a mapping, so it goes into System; gateway type and COM.
- **Schema check and validation differ.** The schema check tests the XSD grammar; the **VALIDATION** panel tests the AUTOSAR constraints (`constr_*`).

Checklist

- [ ] **EXPORT** lists the five `PNC_*.arxml` files.
- [ ] `<PNC-MAPPING>` in `PNC_System.arxml` shows `1 of 3`, and the first is `PNC_Comfort` with identifier `17`.
- [ ] `PNC_System.arxml`, `PNC_DataType.arxml`, `PNC_Interface.arxml` and `PNC_SWC.arxml` show zero errors.
- [ ] `PNC_COM.arxml` shows only the export errors (12 in the finished example).

Next Step

The Appendix holds the design checklist, the project file layout, and the differences from the finished example.

Appendix

What You Will Do

This appendix is reference material, not a chapter you follow along with. Use it to review your PNC design, to see where a value is stored, or when the scre

Design Checklist

Each item links to the chapter that covers it.

- [ ] Is every PNC identifier at least `pncVectorOffset × 8` and at most `(pncVectorOffset + pncVectorLength) × 8 - 1` (`constr_3040`)? Chapter 1, Cha
- [ ] Does every connector's PNC filter array mask have the same length as `pncVectorLength`? Chapter 3
- [ ] Does each physical channel have at most one connector whose `pncGatewayType` is active (`constr_5093`)? Chapter 3, Chapter 11
- [ ] Does every PNC have a top-level PNC coordinator, an ECU connected to two or more of its channels with all those connections active (`constr_3781`)?
- [ ] Does each PNC have an Rx/Tx pair of I-PDU Groups under **PNC GROUPS**? Chapter 4, Chapter 7
- [ ] On every bus the PNC crosses, is **NM PNC PARTICIPATION YES** and are the mandatory NM timing properties filled in (`constr_9157`, `constr_9158`, `const` UdpNm)? Chapter 3
- [ ] Is each **NM NODE ID** unique in its cluster, does **TX-NM-PDU-REFS** hold exactly one NM-PDU of its own (`constr_9388`), and does **RX-NM-PDU-REFS** hold t
- [ ] Is the VFC name identical on the atomic SWC, the ECU composition and the root composition? Chapter 6
- [ ] Does each NM network handle reference its PNC and carry FunctionGroup state mappings for both ACTIVE and INACTIVE? Chapter 9
- [ ] Does the requesting application AA have a `NetworkManagementPortInterface` P-Port and a NetworkHandle mapping, and is the PARA platform import
- [ ] Does each wakeup frame point to a frame that exists on the bus? Chapter 7
- [ ] Is **VALIDATION** at zero findings, and does the export pass the schema check apart from the export errors in `PNC_COM.arxml`? Chapter 11, Chapter 12

Project File Layout

Every design file sits under `design/`.

```
PNC/
├ PNC.io.prj
└ design/
   ├ IVC.io.model            Topology, connector PNC properties, switch port PNC references, NM configuration
   ├ SIGNAL.io.model         Signals, PDUs, frames, NM-PDUs, I-PDU Groups
   ├ SYSTEM.io.model         PNC mappings, PNC vector offset and length, SWC-to-ECU mappings
   ├ INTERFACE.io.model      SR interfaces, service interfaces, NM interfaces
   ├ SERVICE.io.model        SOME/IP deployment, provider and consumer instances
   ├ VFB.io.model            Composition list
   ├ DATATYPE.io.model       Data types
   ├ SWC/
   │  ├ Application/                The nine atomic SWCs. PortGroups (the VFCs) live here.
   │  ├ AdaptiveApplication/        The three AP applications. The PARA platform AAs are in the subfolder HPC/.
   │  ├ Composition/                The 14 node compositions, the root PncVehicle, and PARA_HPC
   │  └ Layout/                     VFB canvas placement (view state)
   └ Module/
      └ Machine/
         └ HPC.machine.io.model     FunctionGroups, NM network handles, AP module configuration
```

The project folder also holds `code`, `doc`, `export`, `import` and `workflow`. The ARXML files go under `export`.

Model Excerpts

Values of the finished example that the screen does not show directly. You never edit these files by hand.

File	Field	Value	
`SYSTEM.io.model`	`pnc-vector-offset`, `pnc-vector-length`	`2`, `6`	The identifier range of The Example and the De
`IVC.io.model` (`SouthZone_CanFd_Conn`)	`pnc-filter-array-masks`	`[12, 0, 0, 0, 0, 0]`	Length 6 = `pnc-vector-length`. `12` is `000011` and `19`
`IVC.io.model` (`SouthZone_CanFd_Conn`)	`pnc-wakeup-can-id`, `pnc-wakeup-can-id-mask`	`1280`, `1792`	`0x500` and `0x700`: frames `0x500` to `0x5FF` ar
`IVC.io.model` (`SouthZone_CanFd_Conn`)	`pnc-mappings`	PNC 18, 19	The PNCs this connector participates in. No fie
`HPC.machine.io.model` (`NH_ParkAssist`)	`function-group-state-mappings`	`ParkAssistFG` `Active`, `Off`	`Active` requests the PNC; when the PNC goes

How a Hand-Built Project Differs from the Finished Example

The current version has no screen for a few values, so a project you build from Designing the Topology on differs from the finished example. None of the

Item	Finished example	Hand-built project	
CAN NM frames	13 NM frames, 29 CAN frames	No NM frame, 16 application frames. A CAN frame cannot carry an NM-PDU	NM co wake impor
NM cluster couplings	One UdpNm and one CanNm coupling	None: no screen	`PNC_` `<CAN-`
Container and NoPnc I-PDU Groups	20 groups: 8 PNC leaves, 6 containers, 6 NoPnc	8 PNC leaves only: no screen	`PNC_` instea
`tp-config` and VLAN configuration	No `tp-config`; `Backbone` has `vlan-configs`	An empty `tp-config` (added by `Ctrl+S` in the **Communication Cluster Configuration** dialog); no `vlan-configs`	No eff
`NmCluster_VLAN_Rear` values	NM timing, **USER DATA**, IPv4 `239.200.0.1`	Defaults: double-clicking `Backbone` always opens `NmCluster_VLAN_Body`	Its ro canno impor
Connector `pnc-mappings`	On 13 CAN connectors	None: no field, and ARXML has no such element	Expor same
Root composition and system information	`PncVehicle`, **VERSION** `1.0.0`, **DESCRIPTION** `Zonal PNC tutorial vehicle`	`System` and the new-project defaults	Name unaff
AP machine modules	Seven: `LOG`, `CM`, `SM`, `NM`, `PHM`, `TIME_SYNC`, `CRYPTO`	Eleven: also `PERSISTENCY`, `IDSM`, `UCM`, `DM`	Four e unaff
PARA platform	Composition `PARA`, one interface `StateClient_MachineFG`	Composition `PARA_HPC`, AAs under `HPC/`, `StateClient_HPC_MachineFG` and one `StateClient_` per FunctionGroup	The n
SOME/IP-derived signals and PDUs	None	`SVC_ParkDistance_SR` and `SVC_…_PDU`	No va
SOME/IP detail values	Saved SD values, endpoints like `HPC_Rear_UDP_30501`	SD defaults, endpoints `<node>_<connector>_UDP_<port>`	Unaff Servic
CAN FD detail values	2 Mbps data phase, 16 CAN FD frames	Only the **CAN FD** toggle on the controller	Unaff **KBPS**
Placement and defaults	Example placement, no FunctionGroup state **VALUE**, `pnc-synchronous-wakeup: false` on the zone ECUs	**Auto Layout** placement, **VALUE** `0`	File va

What to Do About It

If you need a model identical to the example, open the example project or import the example ARXML. Otherwise keep the differences and aim for "No va
Validation.

Problems You Will Likely Hit

Most of these come from how the current version displays things, not from wrong input.

Canvas and Selection

Symptom	What to do
A cluster's kind and baud rate are not shown	Hover over the cluster rectangle
Right-clicking a node opens another menu	Right-click an empty spot inside the node, not a connector or controller
A connector or controller does not get selected	Click its small icon exactly; for a cluster, click the rectangle
A node behind the properties panel cannot be clicked	Drag the canvas aside or collapse the panel
Auto Layout does not order nodes top to bottom	Ignore it; placement does not affect the model
VFB node names are too small	Zoom with `Ctrl` and the wheel
The VFB drill-down path starts with `System`	Display only; the root is `PncVehicle`

When a Value Is Not Shown or Has No Field

Symptom	What to do
`pnc-mappings` of a connector has no field	It is kept in the file; see Model Excerpts
An HPC connector has no **PNC GATEWAY TYPE**	Expected: only Classic Platform ECU connectors have it
No field for VLAN names	Enter only the VLAN ID on the channel
Frame-level CAN FD is not shown	It cannot be checked on screen
TRANSPORT PROTOCOL of a CAN cluster is off	Expected: this tutorial leaves it off
`Backbone` always opens `NmCluster_VLAN_Body`	No screen opens `NmCluster_VLAN_Rear`; see the table above
An NM node name appears twice	Rename **NAME**, for example `WestZone_CanNm` and `WestZone_UdpNm`
TX-NM-PDU-REFS has no candidate	Finish Creating the NM-PDUs in Signals, PDUs, Frames, and I-PDU Groups, then return to NM Configuration
No screen for container and NoPnc groups	Do not create them with + under **PNC GROUPS**: that binds them to the selected PNC

Lists and Tables

Symptom	What to do
NM-PDUs appear among the I-PDU Group candidates	Pick only application PDUs ending in `_Pdu`
× at the end of a group row	It deletes the group, not just unlinks it
PNC MEMBERSHIP puts a PDU in both Rx and Tx	Remove it from the other group with × if needed
Viewer columns are cut off	Widen the viewer, or hover over a cell
NM cluster headers are in capitals	Header style only; the name is `NmCluster_WestCAN`
The search box cannot find a cluster	Use the filter next to it

Behavior While Editing

Symptom	What to do
NM network handle rows stay expanded	Click a row summary to collapse it
A two-state button does nothing	It has no value yet: click the other side, then click back
MAX does not follow a smaller **LENGTH**	Edit **MAX** yourself
A new signal goes after the last signal	Enter **START** yourself
`PNC:Interface` stays unsaved after a port change	Choose **PROJECT** > **Save All** (The System Editor and Validation)
A SOME/IP port error opens the Interface editor	Fix the port on the connector in the System editor
Machine Configuration always shows eleven modules	Saving writes all eleven
The NM CLUSTER line of a handle summary looks wrong	Check **VLANS** and **PARTIAL NETWORKS** instead
One fix removes two errors	One cause can break several constraints, such as `constr_5093` and `constr_3714`

Closing

You now have one model in which three PNCs are recorded from the topology to the ARXML. For more, see Editors and Workflow.